C++ Proqramlaşdırma Dili

Dördüncü Buraxılış

08.10.2018

Əhməd Sadıxov

Mündəricat

§1 Təməl Biliklər.....1

§2 İstifadəçi ilə əlaqə.....11

§3 Dəyişənlər.....36

§4 Şərt operatoru.....80

§5 Dövr Operatorları.....101

§6 Switch operatoru.....129

§7 Cərgələr.....136

§8 Göstəricilər.....162

§9 Funksiyalar.....195

§10 Sətirlər.....242

§11 Struct tiplər.....264

§12 Siniflər.....280

§13 Direktivlər.....306

§14 Fayllar.....313

§15 Siyahılar.....321

Əlavələr.....356

§1 Təməl Biliklər 1

1.1 Proqram anlayışı.....1

1.1.1 Proqram nədir ?.....1

1.2 Kompilyatorların inkişaf tarixi.....3

1.3 Proqram tərtib eləmək üçün bizə nə lazımdır?.....4

1.4 Visual Studio –nun quraşdırılması.....5

1.5 DevC++ -un quraşdırılması.....6

1.6 C++ dilində ilk proqram.....6

§2 İstifadəçi ilə əlaqə 11

2.1 cout operatoru - Məlumatın çap olunması.....11

2.1.1 Məlumatın kansol pəncərəsinə göndərilməsi.....11

2.1.2 cout operatoru.....12

2.2 Sətirlər.....13

2.3 Simvolların çapı.....18

2.4 Tam ədədlərin çapı.....18

2.5 Onluq kəsrlər.....19

2.6 Riyazi ifadələrin çapı.....20

2.7 Xüsusi simvollar.....23

2.7.1 Cütdırnaq simvolu - \".....24

2.7.2 Yeni sətir simvolu.....27

2.7.3 Tabulyasiya simvolu.....29

2.8 Axına əlavə et.....30

2.9 Proqramda Şərhlər.....31

2.9.1 Təksətirli şərhlər.....31

2.9.2 Çoxsətirli şərhlər.....32

§3 Dəyişənlər 36

3.1 Dəyişən nədir?.....36

3.2 Dəyişənlərin adlandırılması.....37

3.3 Dəyişənin tipi.....38

3.4 Dəyişənin elan olunması.....38

3.5 Dəyişənə qiymət mənimsədilməsi.....40

3.6 Dəyişənin qiymətinin ekranda çap edilməsi.....41

3.7 İki dəyişənin qiymətlərinin bir-biri ilə dəyişdirilməsi.....45

3.8 İstifadəçi ilə əlaqə - cin operatoru.....48

3.9 Hesablama operatorları.....54

3.9.1 Bölmə operatoru.....55

3.9.2 Qalıq operatoru.....56

3.10 Kəsr tipi.....58

3.10.1 Kəsr ədədlərlə bölmə.....58

3.10.2 Kəsr hissədən tam hissənin alınması.....60
3.11 Mərtəbə vahidləri.....62
3.12 Simvol tipi.....64
3.12.1 Simvol və Tam tip.....66
3.12.2 Simvolu ədəd qarşılığı.....66
3.12.3 Simvollar üzərində hesab əməlləri.....68
3.13 Sadə cəbri ifadələr.....69
3.14 Sağ tərəfdə dəyişənlərdən istifadə.....70
3.15 Dəyişənin öz qiymətindən istifadə etmə.....72
3.16 Inkrement və Dekrement.....74
3.16.1 İknrement operatoru.....74
3.16.2 Dekrement operatoru.....76
3.17 Digər mənimsətmə operatorları.....76
3.17.1 Cəm mənimsətməsi.....77

§4 Şərt operatoru 80

4. 1 Şərt operatoru – if.....80
4.2 Şərtlərin qurulması – Müqaisə operatorları.....86
4.3 Şərtlərin qiymətləri – true/false (doğru/yalan).....87
4.4 Iç-içə if operatoru.....88
4.5 Mürəkkəb şərtlər – və/və ya operatorları.....88

4.5.1 və operatoru.....89

4.5.2 və ya operatoru.....90

4.6 Inkar operatoru.....91

4.7 Alqoritmik biliklər.....93

§5 Dövr Operatorları 101

5.1 while dövr operatoru.....101

5.2 do while dövr operatoru.....105

5.3 for dövr operatoru.....105

5.4 Cəmin hesablanması.....114

5.5 Hasilin hesablanması.....116

5.6 İç - içə dövr.....117

5.7 Sadə ədədlərin tapılması.....119

5.8 break və continue.....122

5.8.1 break operatoru.....123

5.8.2 continue operatoru.....124

§6 Switch operatoru 129

6.1 Switch operatoru.....129

§7 Cərgələr 136

7.1 Birölçülü Cərgələr.....136

7.1.2 Cərgənin Elementlərinin indeksi.....137
7.1.3 Cərgələrin Elementlərinə Müraciət.....138
7.1.4 Verilmiş indeksli elementin çap olunması.....142
7.1.5 Verilmiş elementin indeksinin tapılması.....142
7.1.6 Elementlərin yerlərinin dəyişdirilməsi.....144
7.1.7 Ən böyük elementin tapılması.....147
7.1.8 Ən böyük elementinin indeksinin tapılması.....149
7.1.9 Cərgənin ən böyük elementinin sona sürüşdürülməsi.....150
7.1.10 Elementlərin artan sırada düzülməsi.....151
7.1.11 Qabarcıq düzüm alqoritmi.....152
7.2 İkiölçülü Cərgələr.....155

§8 Göstəricilər 162

8.1 Kompüterin Yaddaşı.....162
8.1.1 Dəyişənlərin yaddaşda yerləşməsi.....163
8.1.2 Dəyişənin ünvanı.....164
8.1.3 Ünvan operatoru.....164
8.2 Göstərici.....165
8.2.1 Göstəricilərin elanı.....166
8.2.2 Göstəriciyə ünvan mənimsədilməsi.....166
8.2.3 Məlumatın əldə olunması – İstinad operatoru.....170
8.3 Göstəricilər və Cərgələr.....174

8.3.1 Cərgənin adı.....179
8.4 Göstəricilər üzərində hesab.....180
8.5 İkiqat Göstəricilər.....187
8.6 Dinamik yaradılma.....189
8.6.1 new əmri.....190
8.6.2 Dinamik cərgə yaratmaq.....191
8.7 İkiqat Göstəricilər və İkiölçülü Cərgələr.....193

§9 Funksiyalar 195

9.1 Funksiyaların tərtib olunması.....195
9.1.1 Funksiyaya aid numunə.....195
9.1.2 Funksiyanın Tipi.....196
9.1.3 Funksiyanın Adı.....196
9.1.4 Funksiya parametrləri.....197
9.1.5 C++ dilində funksiyaların elan olunmasına aid çalışmalar.....198
9.1.6 Funksiyanın bədəni.....200
9.1.7 Funksiyanın elanı ilə funksiya bədəninin birləşdirilməsi.....201
9.2 Funksiyanın çağrılması.....201
9.3 Funksiyanın nəticə qaytarması.....204
9.4 Funksiyaya parametr ötürülməsi.....207
9.5 Funksiya parametrindən istifadə.....209
9.6 Dəyişənlərin funksiyaya məlumat kimi ötürülməsi.....211

9.7 Funksiyalara aid çalışmalar.....212

9.8 Lokal və qlobal dəyişənlər.....218

9.9 Funksiyanın qayıtması barədə ətraflı.....222

9.10 Məlumatın qiymətə və ünvana görə ötürülməsi.....224

9.10.1 Göstəricinin ötürülməsi.....226

9.10.2 Cərgələrin funksiyaya parametr kimi ötürülməsi.....230

9.10.3 Funksiyalara ikiölçülü cərgələrin ötürülməsi.....237

9.11 Rekursiv funksiyalar.....238

9.11.1 n faktorial.....239

§10 Sətirlər 242

10.1 Sətirlər.....242

10.1.1 Sətrin sonu simvolu.....244

10.1.2 Sətir funksiyaları.....246

10.1.2.1 Strlen funksiyası.....246

10.1.2.2 strcpy funksiyası.....247

10.1.2.3 strncpy funksiyası.....248

10.1.2.4 strcat funksiyası.....249

10.1.2.5 strcmp funksiyası.....250

10.1.2.6 strtok funksiyası.....250

10.1.2.7 getline funksiyası.....252

10.2 string tipi.....257

10.2.1 append funksiyası.....260

10.2.2 insert funksiyası.....260

10.2.3 replace funksiyası.....261

10.2.4 c_str funksiyası.....262

§11 Struct tiplər 264

11.1 Struct tiplər.....264

11.1.1 Yeni tipdən dəyişən elan etmək.....265

11.1.2 Obyektin həddlərinə müraciət.....266

11.1.3 Obyektlərin funksiyalara ötürülməsi.....270

11.2 İç-içə strukt tiplər.....273

11.3 Strukt tipindən olan göstəricilər.....275

11.4 Dinamik yaradılma.....277

§12 Siniflər 280

12.1 Siniflər.....281

12.2 İlk sinif.....282

12.3 Obyektlərin Yaradılması.....283

12.3.1 Obyektin həddlərinə müraciət.....284

12.4 Sinfin Funksiya Həddləri.....284

12.4.1 Sinfin funksiya həddlərinə müraciət.....286

12.5 Private – Gizli həddlər.....289

12.6 Kanstruktor.....291

12.7 Destruktor.....295

12.8 Varislik.....297

§13 Direktivlər 306

13.1 Əmr direktivi.....306

13.2 Təyin direktivləri.....309

13.2.1 define direktivi.....310

13.2.2 undef direktivi.....311

13.3 Şərt direktivləri.....311

§14 Fayllar 313

14.1 Fayllar.....313

14.2 Stream – Axınlar.....314

14.3 Fayldan məlumatın oxunması.....316

§15 Siyahılar 321

15.1 Siyahılar.....321

15.2 Siyahı yaratmaq.....322

15.2.1 Siyahının ilk və son elementləri.....325

15.2.2 Siyahının təyin olunması.....326

15.2.3 Siyahıya elementlərin əlavə olunması.....327

15.2.4 Siyahıya elementin əlavə olunması.....331

15.3 Siyahının elementlərinə müraciət.....333

15.4 Elementlərin dinamik əlavə olunması.....337

15.5 Siyahıdan elementlərin silinməsi.....344

15.6 Iki istiqamətli siyahılar.....350

15.6.1 İkiistiqamətli siyahının yaradılması.....350

Əlavələr 356

Əlavə A.....356

ASCİİ cədvəli.....356

Önsöz

Dördüncü buraxılış

C++ proqramlaşdırma dili kitabının növbəti dördüncü buraxılışında hamınızı xoş gördük. Bu buraxılışda əvvəlki buraxılışda olmayan bir çox yeni başlıqlar əlavə olundu, mövcud olanların isə mətn və kod hissələrində bəzi dəyişikliklər edildi. O cümlədən fayllarla iş, siyahılar bölmələrini əlavə etdik. Makroslar, OYP –nin mövcud bölmələri təkmilləşdirilib.

Proqramçı olmaq istəyirəm

21-ci əsrin ən çox tələb olunan peşələrindən biri yəqin ki, proqramçı peşəsidir. Səbəbi isə çox sadədir. Kompüterlər həyatın hər sahəsinə daha da dərin nüfuz etməkdədir. Bunun yaxşı və ya pis olmasını gələcək göstərər. Amma bizim üçün əsas odur ki, kompüterlər nə qədər çox istifadə olunsa proqramçılara da ehtiyac o qədər çox artar. Çünki kompüterlər proqramsız heç bir iş görə bilməz.

Gələcəkdə özünü proqramçı kimi görmək istəyənlərin qarşısında müxtəlif seçimlər dayanır. Xüsusən də ilk başlanğıcda təcrübənin az olması dəqiq hədəfi müəyyənləşdirməyə çətinlik yaradır. Proqramçı olmaq necə mümkündür? Buna nə qədər vaxt tələb olunur? Yalnız proqramlaşdırma təcrübəsi öz karyeranızı qurmağa, həyatda istədiyiniz mövqeyə sahib olmağa yetərlimidir? Əlbəttə hər bir fərd bu suallara ayrılıqda cavab tapır. Ən başlanğıcda isə orta səviyyəli proqramlaşdırma biliklərinə yiyələnmək vacibdir. İnformasiya Texnologiyalarının hansı sahəsində özünüzü sınamaq istəsəniz orta səviyyəli proqarmlaşdırma biliyi sizə öz mövqeyinizi düzgün qiymətləndirmək və düzgün istiqamət seçmək üçün ehtiyacınız olan texniki, praktiki dəstəyi artıqlaması ilə verəcək.

Məhs proqramlaşdırma təməli olan fərdlər daha sonradan İnformasiya Texnologiyalarının birbaşa proqramlaşdırma ilə bağlı olmayan digər sahələrinə müraciət etdikləri zaman proqramlaşdırma bilikləri olmayanlara nisbətən qat-qat artıq və asan istədiklərini əldə edirlər.

Bu kitab sizə tam başlanğıcdan orta səviyyəli proqramçı olmağa kömək edəcək. Əlbəttə kitabın son mövzuları çətinlik dərəcəsinə görə orta səviyyəni çox-çox geridə qoysa da, ümumilikdə biz təcrübəli bir proqramçının yetişməsi üçün bir kitabın, bir neçə ilin kifayət etməməsi qənaətindəyik.

Niyə məhs C++ dili?

C++ dili ötən əsrin 70-ci illərində Byarn Straustrup tərəfindən yaradılıb. Təxminən 40 ilə yaxındır ki C++ dili istifadə olunur və bu müddət ərzində dilin standartları xeyli təkmilləşdirilib, müxtəlif platforlamalarda işləyən kompilyatorları, kitabxanaları inkişaf etdirilib.

90-cı illərdə obyekt yönümlü proqramlaşdırma sahəsində C++ öz məşhurluğunun pik həddini yaşayırdı. Lakin 2000-ci illərdən etibarən Java-nın məşhurlaşması C++ əhəmiyyətini bir qədər zəiflətdi. Düzdür C++ öz aktuallığını heç vaxt itirmədi, buna səbəb onun kampilyatorunun irihəcmli, sürətli, resurslardan qənaətlə istifadə edə bilən layihələrin yaradılması üçün unikal alqoritm əsasında hazırlanmasıdır. Bütün bu müddət ərzində istər kompilyator dizaynı, istər oyun proqramlarının hazırlanması, istər əməliyyatlar sistemlərinin hazırlanması və bir çox başqa sahələrdə C++ dili əvəzedilməzliyini qorudu. Lakin digər dillərin başqa sahələrdə hədsiz məşhurluğu, kompüterlərin hesablama imkanlarının xeyli dərəcədə inkişaf etdirilməsi bir çox sadə və orta səviyyəli layihələrdə C++ -u sıxışdırdı. Lakin bununla belə C++ heç də köhnəlmədi əksinə dilin yeni standartları və kitabxanalarının təkmilləşdirilməsi üzərində müəlliflər öz çalışmalarını əzmlə davam etdirdilər. Artıq C++ dilinin son 11 və 14 standartları tamamilə müasir standart hesab olunur və bir çox o cümlədən mobil həllərin hazırlanmasında uğurla tətbiq olunur. Məhs bu yeniləmələr sayəsində C++ dili tədricən öz əvvəlki şöhrətini özünə qaytarmaqdadır və getdikcə ondan istifadənin artan istiqamət üzrə inkişaf edəcəyi gümən edilir.

Müəllifin qənaətinə gəlincə şəxsən mən belə hesab edirəm ki C++ dili hər-bir proqramçının bilməsi gərəkən bir dildir. Buna görə biz bütün bu illər ərzində seçimimizi məhs C++ dili üzərində etmişik və çox güman ki gələcək illərdə də çalışmalarımızı məhs bu dilin tədrisi, dərsliklərin hazırlanması istiqamətində davam etdirəcəyik.

Başlamaq üçün nələri bilməliyəm?

Tək C++ dili deyil, istənilən digər proqramlaşdırma dilini örgənməyə başlamaq üçün ilk öncə şübhəsiz ki, orta səviyyədə riyazi biliklər tələb olunur. Orta səviyyədə riyaziyyat dedikdə söhbət inteqral və ya triqonometrik tənlik həll etməkdən getmir. Amma ən azından tam ədəd, kəsr ədəd, nisbət, qalıq, faiz, sadə ədəd, mürəkkəb ədəd, kvadrat tənliklərin həlli v.s. bu kimi orta məktəb səviyyəsindəki riyazi savadınız olmalıdır.

Kitab kimlər üçün nəzərdə tutulub.

C++ dili sənaye tətbiqli dildir. Bu dil real vaxt rejimli, yüksək performans tələb edən, məhdud resurslardan qənaətlə istifadə edilməsi gərəkən irihəcmli proqram layihələrinin tərtibi üçün nəzərdə tutulub. Aydın məsələdir ki, bütün bu tələblərin qarşılanması öz yükünün bir hissəsini dilin sintaksisi üzərində əks etdirir. C++ dilini örgənmək və onda proqram yazmaq əlbəttə ki digər dillərlə müqaisədə bir qədər çətindir. Ümumiyyətlə C++ dilini örgənmək yenibaşlayanlara məsləhət görülmür. Amma bu qətiyyən o demək deyil ki, yenibaşlayanlar C++ dilindən başlaya bilməzlər. Cəhd eləməklə heç kim heç nə itirmir. Hər halda biz bu kitabı tam yeni başlayanlar üçün anlaşılacaq şəkildə tərtib etməyi qərara aldıq.

Kitabdan istifadə qaydası.

Bu kitab C++ dilini tam başlanğıc səviyyədən peşəkar səviyyəyə kimi tədris edir. Kitab proqramlaşdırmanı özü örgənənlər və ya müəllim ilə örgənənlər üçün nəzərdə tutulub. Bu və ya digər mövzuların ətraflı izahından sonra örgənilən biliyin nə dərəcədə mənimsənildiyini yoxlamaq məqsədilə mövzuya dair suallar təqdim olunur. Daha sonra isə praktiki biliyi möhkəmləndimək üçün proqram çalışmaları daxil edilib. Məhs bu çalışmalar keçilən mövzunun mənimsənilməsini əhəmiyyətli dərəcədə artırır və növbəti mövzuların başa düşülməsini asanlaşdırır. Qayda belədir: Əgər hər paraqrafın sonunda verilmiş çalışmaları eləməsəniz növbəti mövzunu başa düşməyi yaddan çıxarın. Müəllif hər paraqrafın sonunda mövzuya aid çalışmaları gözəllik xatirinə kitaba daxil etməyib. Onların hər biri mütləq yerinə yetirilməlidir. Bu vacib şərtdir, əks halda siz irəlidəki mövzularda çox ciddi çətinliklərlə üzləşəcəsiniz. Amma

hər-bir paraqrafın sonundakı çalışlamara ciddi əhəmiyyət vermək və onlara xeyli vaxt sərf etmək sizə kitabdakı mövzularla irəliləməyə böyük köməklik göstərəcək.

Müəllif hüquqları:

Kitabla bağlı iradlar, təkliflər və səhvlərin bildirilməsi müəllif tərəfindən çox müsbət qarşılanır və aşağıdakı ünvandan müəlliflə əlaqə saxlamağınız xahiş olunur.

ahmed.sadikhov@gmail.com

§1 Təməl Biliklər.

1.1 Proqram anlayışı

Məqsədimiz C++ dilində proqram tərtib etməyi örgənməkdir. Ona görə əvvəl yaxşı olar ki **proqram** anlayışına bir qədər aydınlıq gətirək.

1.1.1 Proqram nədir ?

Kompüterlə bağlı məsələlərdə çox sıx qarşılaşdığımız anlayışlardan biri **proqram** anlayışıdır. Hər-hansı proqramı quraşdırmaq, ofis proqramları, sənin kompüterində filan proqram varmı? v.s. kimi ifadələrdən tez-tez istifadə edirik. Bəs konkret olaraq proqram nədir? Onun rəngi, qoxusu, forması, ölçüləri, mahiyyəti nədən ibarətdir?

Proqramı sadə şəkildə "**əmrlər ardıcıllığı**" kimi təsəvvür edə bilərik. Misal üçün aşağıdakı əmrlər ardıcıllığına nəzər yetirək:

```
Qalx!
Sağa dön!
10 addım irəli get!
Pərdələri çək!
Pəncərəni aç!
5 dəqiqə gözlə!
Pəncərəni bağla!
```

Baxdığımız bu **əmrlər ardıcıllığı** hansı ki biz onu proqram adlandırırıq kompüter dilində otağın havasını dəyişmək üçün hər-hansı şəxs tərəfindən yerinə yetirilməli olan əmrlər ardıcıllığına bir nümunədir. Bu əmrlər ardıcıllığı **insan** tərəfindən icra olunur. Kompüter proqramları isə kompüter tərəfindən icra olunur. Bizim gündəlik həyatımızda gördüyümüz işlər, başqa sözlə desək insanın yerinə yetirə biləcəyi işlər barəsində məlumatımız var. Bəs kompüter hansı tipdə əmrləri icra edə bilir? Yəni kompüter üçün hansısa bir əmrlər ardıcıllığı siyahısı tərtib etsək ora nə kimi əmrlər daxil edə bilərik?

Biz kompüterin nələrə qadir olduğuna gündəlik həyatımızdan kifayət qədər bələdik. İnternetdə müxtəlif məlumatların axtaılmasından tutmuş qrafik dizayna kimi v.s. , v.s. saymaqla bitməyən müxtəlif işlər. Amma bunlar hamısı əlbəttə ki visual olaraq belədir. Yəni biz kompüterin gördüyü işin yekun nəticəsini hiss edirik. Ən dərində isə kompüter ancaq bir iş görə bilir. Ədədlər üzərində hesab əməlləri aparmaq. Bəli bu fikir bir qədər qəribə səslənə bilər, necə ki bu qədər rəngarəng işlər görə bilən kompüterin reallıqda yalnız və yalnız ədədlərlə işləyə bilər. Lakin bu bir həqiqətdir. Əgər siz kompüter işləyə-işləyə, misal üçün deyək ki internetdə youtube-dan hansısa bir videoya baxarkən, xəyali olaraq kompüteri yarıb onun nə işlə məşğul olduğuna nəzər salsanız görərsiniz ki, kompüter qarşısına böyük sayda müxtəlif ədədlər yığıb və onlar üzərində müxtəlif hesab əmməlləri: toplama, çıxma, vurma, bölmə, qalıq hesablanması v.s. əməlləri aparmaqla məşğuldur.

Lakin əlbəttə ki bunlar hamısı ən dərində baş verənlərdi. Bu qədər dərinə biz enməyəcəyik. Biz – yəni siz proqramçılar istifadəçilərlə kompüterin dərinlikləri arasında bir layda mövqe tutacağıq.

Yaxşı biz izah elədik ki, kompüter necə işlər görə bilir, bəs onda kompüter proqramı necə olar? Proqram qeyd elədiyimiz kimi əmrlər ardıcıllığı, siyahısı olduğuna görə komüterin gördüyü işlərin hər-hansı bir ardıcıllığını tərtib etsək proqram alarıq. Dedik ki kompüter yalnız ədədlər üzərində hesab əmməlləri apara bilir. Onda nümunə bir kompüter proqramı aşağıdakı kimi olar:

```
5+3*67-21
456+2
12/34*2-1
82+345-71
```

Sizin nə hiss elədiyinizi anlayıram: bu nə darıxdırıcı və mənasız bir şeydir. Hələ bu son deyil. Əslində bizim adət etdiyimiz ədədlər və hesab əməliyyatları kompüterə ikili dildə təqdim olunur, aşağıdakı kimi:

```
01110100 11001110 10011110 10110110
00101110 10000110 01000110 00000000
01110100 11001110 00101110 01001110
00101110 10000110 01000110 00000000
01110100 11001110 00010110 11001110
00101110 01001110 00101110 10000110
01000110 00000000 01110100 00101110
10100110 00011110 00101110 00000000
01110100 00100110 10000110 00101110
```

Bu ədədlər yığınından sizə nəsə bir şey aydın olur? Şəxsən mənə heçnə aydın deyil, amma mən bu ədədlər ardıcıllığını bir proqramın içindən əldə etmişəm. Yəni bu

gördüyümüz dəqiq icra olunan bir proqram parçasıdır və sizi inandırmaq istəyirəm ki kompüter bu yuxarıda yazdığımız ədələr yığınının nə demək istədiyini çox gözəl başa düşür.

Bu yerdə mən bu ədədlərlə bağlı narahatçılığa birdəfəlik son qoymaq istəyirəm. Bizim bu ədədlərlə heç vaxt işimiz olmayacaq. Çünki biz kompüter proqramı yaratmaq üçün xüsusi vastələrdən – **kampilyatorlardan** istifadə edəcəyik.

1.2 Kompilyatorların inkişaf tarixi

Proqramlaşdırmanı örgənməyə başlamadan öncə biz proqramçıların işini bu qədər asanlaşdıran kompilyatorlar barəsində bir az söz açmaq məncə vacibdir. Gəlin görək bizi bu ədədlər cəncəlindən xilas edən kompilyatorlar necə yaranıb. Əlbəttə kompüterlərin yarandığı ilk günlərdə, hardasa keçən əsrin 20-ci illərində , bütün proqramlar məhs bu ikili ədədlər ardıcıllığı şəklində tərtib olunurdu. Bunun nə qədər cansıxıcı və çətin bir iş olduğunu yəqin ki təsəvvür edirsiniz.

Təxminəm 50-ci illərdə formal dillər nəzəriyyəsi yaranmağa başladı. Grammatikaları sadədən mürəkkəbə doğru qruplara ayıran və xassələrini tədqiq edən bir sahə. Qramatikaları bir-birinə yaxın olan dillərdəki mətnlərin bir-dildən digərinə tərcümə olunması əsas tədqiqat istiqamətlərindən idi. Burada qeyd ediyimiz kimi söhbət təbii yəni bizim danışdığımız dillərdən yox, məhs formal yəni qramatik qaydaları sadə olan dillərdən gedir. Təbii dillərin qrammatik qaydaları hədsiz dərəcədə mürəkkəb olduğuna görə onları kompüter tərəfindən tərcümə etmənin elmi əsasları hələ ki mükün olmayıb.

Formal dillər sahəsinin tədqiqi nəticəsində proqramçılar qrammatikası ikili dilə yaxın olan assembler dillərini yaratdılar və assembler dillərində tərtib olunmuş proqram mətnini ikili dilə tərcümə edə bilən kompilyator proqramlarını qurdular. Əlbəttə assembler dillərinin yaranması proqramlaşdırmanı çox irəli apardı, lakin irəlidə hələ həll olunması gərəkən çox məsələ qalırdı. Belə ki assembler dilləri ikili dil qədər çətin olmasa da, iri həcmli kodların tərtibi üçün elə də əlverişi deyildi. Assembler dilinə nümunə kod aşağıdakı kimi olar:

```
pushl   %ebp
movl    %esp, %ebp
subl    $8, %esp
andl    $-16, %esp
movl    $0, %eax
movl    %eax, -4(%ebp)
```

```
movl    -4(%ebp), %eax
call    __alloca
call    ___main
movl    $LC0, (%esp)
call    _printf
movl    $0, %eax
```

Növbəti onilliklər ərzində proqramlaşdırma dilləri, kompilyatorların dizaynı nəzəriyyələri akademik sferada özünə xeyli diqqət cəlb elədi. Artıq 70-ci illərdə kifayət qədər mürəkkəb qrammatikaya malik dilləri təhlil edə bilən alqoritmlər mövcud idi. Bu isə insan tərəfindən daha rahat başa düşülən proqramlaşdırma dillərinin yaradılmasına və bu dillərdə tərtib edilmiş proqram mətnlərini ikili dilə tərcümə edən kompilyatorların yaradılmasına imkan verdi.

Assembler dillərindən sonra yaradılan yeni nəsil dillər 3-cü səviyyəli dillər adlandırılır. Onların günümüzdə məşhur olan nümunələrinə misal olaraq C, C++, C#, Java, PHP, Vbasic, Paskal v.s. kimi proqramlaşdırma dillərini göstərə bilərik. Misal üçün C++ dilində nümunə proqram kodu aşağıdakı kimi olar:

```
#include <iostream>

int main (){

 int x,y,z;

 x = 10;
 y = 5;
 z = x + y;

 std::cout<<x<<" + "<<y<<" = "<<z<<"\n";

}
```

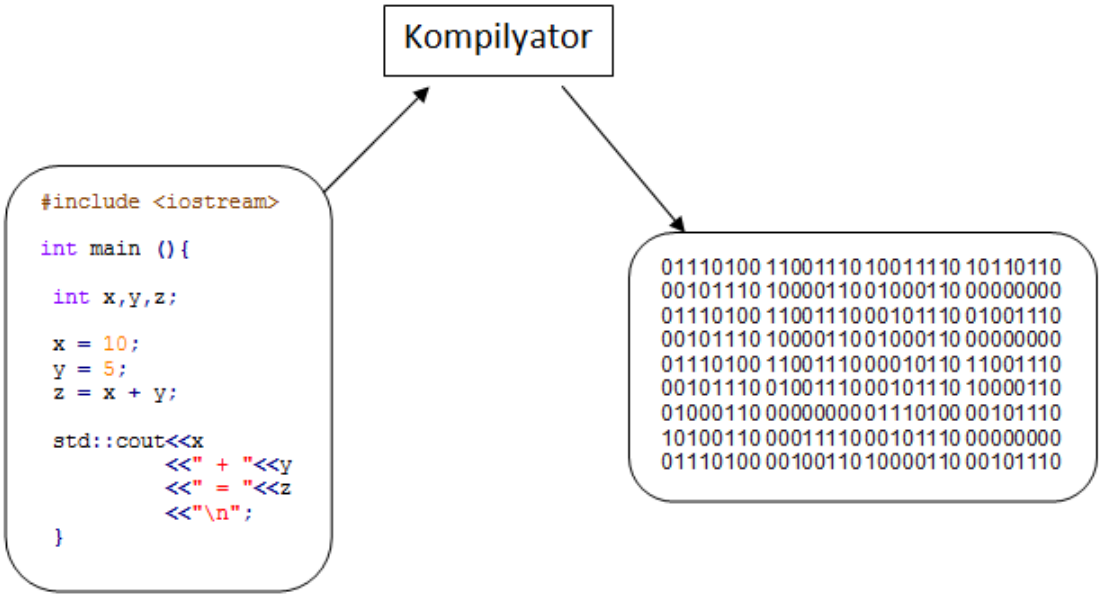
Proqramçılar bu cür dillərdə proqram tərtib edirlər, daha sonra kompilyatorlar vastəsilə həmin proqram mətnini ikili dildə olan koda çevirirlər (bax şəkil 1.1).

Adətən proqramın ikili dildə olan formasına **ikili kod** və ya **icraolunabilən kod**, 3-cü səviyyəli dillərdə olan mətninə isə **mənbə kodu** deyirlər.

1.3 Proqram tərtib eləmək üçün bizə nə lazımdır?

Yuxarıda kompilyatorlar barəsində etdiyimiz izahdan aydın olur ki, bizə proqram tərtib etmək üçün məhs kompilyator proqramı tələb olunur. Kompilyator proqramını nə cür əldə edə bilərik?

Kompilyator proqramı dedikdə söhbət kankret bir proqramdan getmir. Hər-bir dilin öz kompilyator proqramı olur. Bu kompilyator proqramları həmin dillərin yaradıcıları tərəfindən tərtib olunur. Bir çox sənaye tətbiqli dillər mövcuddur ki, artıq onların beynəlxalq standartları yaradılıb və bu standart əsasında müxtəlif şirkətlər həmin dilin kompilyatorlarını tərtib edirlər.



Şəkil 1.1

C++ dilinin müxtəlif şirkətlər tərəfindən çoxlu kompilyatorları mövcuddur və onların çoxunu qeyri kommersiya məqsədi ilə istifadə etmək üçün pulsuz əldə etmək mümkündür. Bunlara ən məşhur nümunə olaraq Microsoft şirkətinin istehsal etdiyi Visual C++ və FSF (Free Software Foundation) tərəfindən istehsal olunan GCC kompilyatorlarını göstərmək olar. Hər iki kompilyatoru pulsuz əldə edib kompüterinizə yükləyə bilərsiniz. Əlbəttə kompilyatorla yanaşı sizə proqram mətnini tərtib etmək üçün inteqrəolunmuş sintaksisrəngləmə dəstəkli mətn redaktoru də təqdim olunacaq. Bütün bunlar biryerdə İDE adlandırılır – İntegrated Development Environment.

Visual C++ kompilyatoru Visual Studio İDE –si, GCC kompilyatoru isə DEVC++ İDE –si ilə ən çox istifadə olunur. Hər iki İDE-nin quraşdırılması və istfadə qaydası ilə bağlı bəzi qeydlərlə tanış olaq.

1.4 Visual Studio –nun quraşdırılması

Visual Studio peşəkar proqramlaşdırma mühiti nisbətə mürəkkəb quraşdırılma və istifadə qaydalarına sahibdir. Visual Studionun quraşdırılması, tələblə v.s. barədə ətraflı məluamtı Microsoft şirkətinin rəsmi veb səhifəsindən əldə etmək mümkündür:

https://docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation

Təəssüf ki həmin səhifə ingilis dilindədir.

1.5 DevC++ -un quraşdırılması

Nisbətən yenibaşlayanlar üçün daha münasib olan C++ dili proqramlaşdırma mühiti DevC++ İDE –dur. DevC++ İDE – nin quraşdırılması və istifadəsi barədə ətraflı məlumatı aşağıdakı veb səhifədən əldə edə bilərsiniz:

https://en.wikiversity.org/wiki/Installing_and_using_Dev-C%2B%2B

Təəssüf ki bu səhifə də ingilis dilindədir.

Ümumiyyətlə isə quraşdırma və ingilis dili ilə problemi olanlar heç bir proqram qurmadan da belə veb brovzerin köməyi ilə onlayn proqramlaşdırma mühitlərindən istifadə etməklə C++ dilində proqramları kompilyasiya və icra edəbilərlər. Əlbəttə onlayn İDE –lər desktop İDE –lər kimi geniş imkanlara sahib olmasa da, hər halda məhdud imkanlarla belə C++ proqramlarını test etməyə imkan verir və ilk başlanğıcda onlayn İDE-lərin təqdim etdiyi imkanlar desktop İDE-lərinkindən elə də çox geri qalmır. Aşağıda C++ proqramlarını veb səhifə üzərindən kompilyasiya və icra eləmək üçün bir neçə məşhur onlayn İDE –nin ünvanı göstərilir.

http://cpp.sh/

https://www.onlinegdb.com/online_c++_compiler

https://ideone.com/

https://code.hackerearth.com/3ae1bar

1.6 C++ dilində ilk proqram

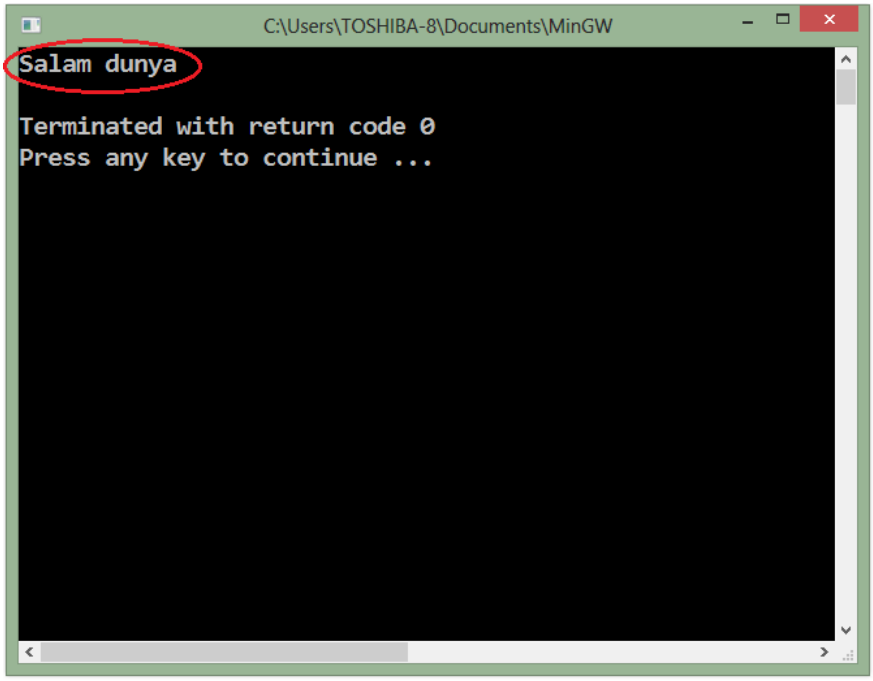
Biz indi sizə C++ dilində ilk proqramımızı təqdim edəcəyik. Adət-ənənəyə uyğun olaraq hər bir proqramlaşdırma dilini örgənməyə başlayanda ilk olaraq məşhur Hello World - Salam dünya proqramı ilə başlayırlar. Biz də bu adət-ənənəyə sadiq qalırıq.

C++ dilində “Salam dünya” proqramı aşağıdakı kimi olar:

```
#include <iostream>
```

```
using namespace std;

int main ()

{

    cout<<"Salam dunya";

}
```

Əgər bu proqramı kompilyasiya və icra eləsək ekranda Salam dunya sətri çap olunar, aşağıdakı kimi:



Gəlin bu proqram mətninin hər bir sətrini ayrı-ayrılıqda nəzərdən keçirək.

İlk sətir:

```
#include <iostream>
```

Burada **include** direktivi vastəsilə **iostream** başlıq faylı(header file) proqrama əlavə olunur. Bu bizə ekranda məlumat çap etmək, istifadəçidən məlumat qəbul etmək v.s. kimi giriş-çıxış əməliyyatlarını yerinə yetirən standart funksiyalardan istifadə etməyə imkan verir. Başlıq faylları ilə daha ətraflı 10-cu paraqrafda tanış olacayıq.

Növbəti sətir:

```
using namespace std;
```

Bu sətirdə biz std ad sahəsindən istifadə etdiyimizi bildiririk. Hələlik bunun nə demək olduğunu başa düşməyə çox da çalışmayaq. Sadəcə hər bir proqramımızın əvvəlinə bu sətri əlavə etməklə kifayətlənək. Çünki elə məqamlar var ki yeni başlayanda onların üzərində çox dayanmaq sadəcə vaxt itkisinə səbəb olur.

Növbəti sətir:

```
int main ()
```

Bu sətirdə **main** adlı funksiya elan edirik. C++ proqramları müxtəlif funksiyalardan ibarət olur. **main** funksiyası proqramın əsas funksiyasıdır və proqram icra olunmağa main funksiyasından başlayır. Hər bir C++ proqramında main funksiyası mütləq olmalıdır. Funksiyalar barəsində daha ətraflı 6-cı paraqrafda məşğul olacayıq.

Növbəti sətir:

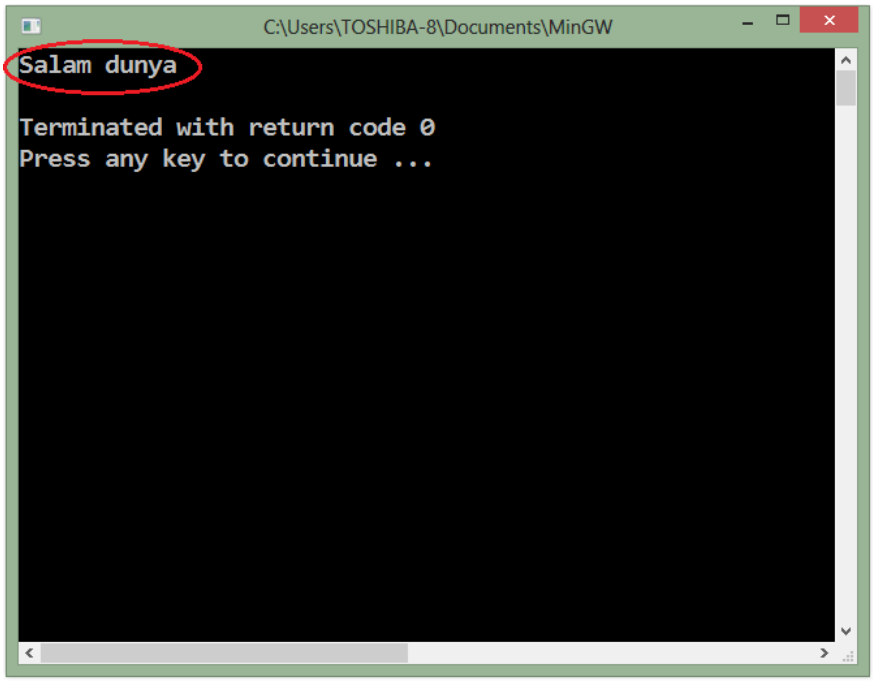
```
{
```

Bu açan fiqurlu mötərizə main funksiyasının kod hissəsinin başladığını bildirir.

Növbəti sətir:

```
cout<<"Salam dunya";
```

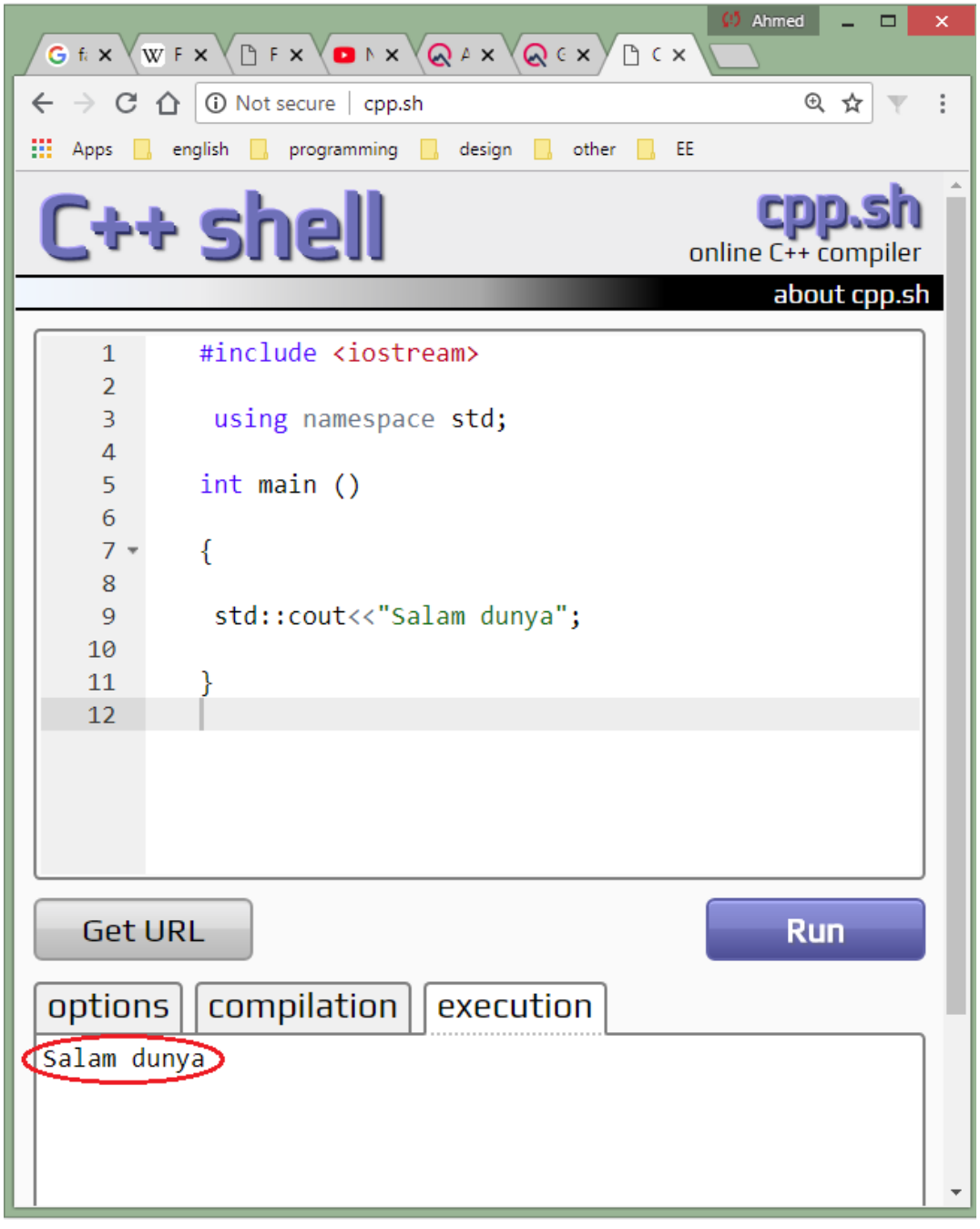
Tərtib etdiyimiz proqramda nəsə bir iş görən yeganə yer məhs bu hissədir. Bu kod icra olunduqda kansol pəncərəsində **Salam dunya** çap olunur. Yuxarıdakı şəklə bir daha nəzər salaq.



Son sətir:

```
}
```

Bağlayan fiqurlu mötərizə ilə main funksiyasının kod hissəsinin bitdiyini bildiririk. Proqramı kompüterinizə quraşdırdığınız və ya hər-hasnsı münasib bildiyiniz onlayn İDE vastəsilə kompilyasiya və icra edin. Misal üçün http://cpp.sh/ onlayn İDE –sində nəticə aşağıdakı kimi olar:



İlk proqramın mətninə bir daha nəzər salaq:

```
#include <iostream>

using namespace std;

int main ()

{

    cout<<"Salam dunya";

}
```

Gələn dərsimizdə biz main funksiyasının açan və bağlayan fiqurlu mötərizələri arasındakı kodda yəni

```
    cout<<"Salam dunya";
```

kodundakı istifadə etdiyimiz **`cout`** operatorunu örgənəcəyik. İndi isə növbəti bölməyə keçmədən öncə aşağıdakı tapşırıqları yerinə yetirməyiniz tələb olunur.

Tapşırıq 1.

Salam dunya proqramının hər bir sətrini ətraflı nəzərdən keçirmək və tam mətni əzbərləmək. Tapşırığı nə cür yerinə yetirməyinizi yoxlamaq üçün ağ bir vərəq götürüb kitaba baxmadan öz beyninizdən Salam dunya proqramını yazmağa çalışın və kitabdakı nümunə ilə yoxlayın. Harada işarə səhvi etdiyinizi müəyyənləşdirməyə çalışın.

Tapşırıq 2.

Salam dunya proqramını kompüterinizə quraşdırdığınız və ya hər-hasnsı münasib bildiyiniz onlayn İDE vastəsilə kompilyasiya və icra edin.

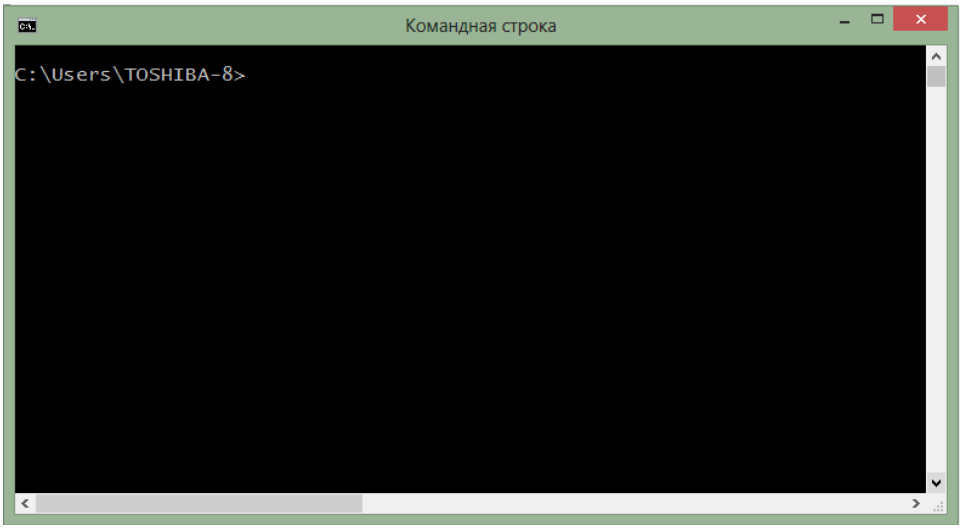
§2 İstifadəçi ilə əlaqə.

2.1 cout operatoru - Məlumatın çap olunması

Proqramların gördüyü işləri 3 qrupa ayırmaq olar. Məlumatın qəbul olunması, məlumatın emal olunması və məlumatın xaric olunması. Hər 3 növ iş olduqca müxtəlif üsullarla yerinə yetirilə bilər. Misal üçün müasir zamanda kompüterin məlumatları hansı mənbələrdən əldə etməsinə aid misallar çəkməklə bitməz. Kompüter məlumatı müşahidə kamerasından tutmuş, süni peykə qədər müxtəlif qurğulardan əldə edə bilər. Lakin proqramlaşdırmaya yeni başlayanda bizim hələlik istifadə edəcəyimiz qurğu klaviatura olacaq və giriş qurğusu dedikdə biz klaviaturanı, giriş məluamtı dedikdə istifadəçinin klaviaturadan daxil etdiyi hər-hası ifadəni nəzərdə tutacayıq.
Eyni qayda ilə kompter məlumatı müxtəlif mənbələrə ötürə bilər. Sadəlik üçün yeni başlayanda biz çıxış qurğusu olaraq kansol pəncərəsindən istifadə edəcəyik.

2.1.1 Məlumatın kansol pəncərəsinə göndərilməsi

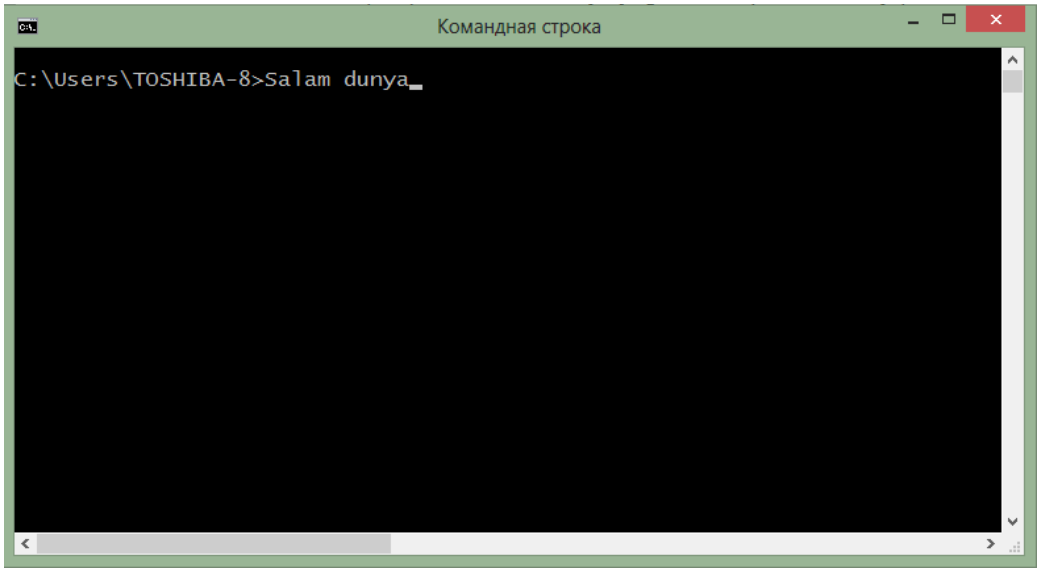
Sizlərdən kansol pəncərəsinin nə olduğunu bilməyənlər qoy elə də təəssüflənməsinlər. Əslində kansol pəncərəsi dedikdə ümumi anlayış nəzərdə tutulur. Əgər İDE –dən istifadə edirsinizsə onda kansol pəncərəsi proqramı icra edərkən(bunun üçün İDE-nin müvafiq menyularından istifadə olunur) İDE tərəfindən çağrılacaq. Onlayn İDE –də də eyni qayda ilə.



Hər hansı məlumatı proqramdan standart çıxışa göndərməyə məlumatı çap etmək deyirlər. Məlumatın çap olunması dedikdə printerdə çap nəzərdə tutulmur. Standart çıxış istənilən mənbəyə pərçimlənə bilər, lakin susmaya görə konsol pəncərəsi standart çıxış qurğusu hesab olunur.

2.1.2 cout operatoru

Məlumatı çap etmək üçün **cout** operatorundan istifadə olunur. Bu zaman çap elədiyiniz bütün məlumatlar həmin kansol pəncərəsində öz əksini tapır. Buna deyillər məlumatı ekranda çap eləmək.



Gördüyümüz kimi yuxarıdakı kansol pəncərəsində Salam dunya sətri çap olunmuşdur.

Tipindən asılı olmayaraq **cout** ilə istənilən məlumatı çap etmək üçün aşağıdakı sintaksis qaydadan istifadə edirik:

```
cout << Melumat ;
```

Əvvəlcə operatorun adını yazırıq

cout

Daha sonra isə birləşdrimə operatoru adlandırdığımız ardıcıl olaraq iki dənə kiçikdir – **<** işarəsi kimi təsvir olunan operatoru yazmalıyıq.

<<

Burada diqqət eləmək lazəmdır ki bu iki kişikdir işarəsi bir-birindən aralı olması, yəni

`< <` və ya `<  <` kimi yazılışlar birləşdirmə operatoru deyil, sintaksis səhv hesab olunur.

Daha sona çap etmək istədiyimiz məlumatı yazırıq

```
Melumat
```

Sonda isə nöqtəvürgül işarəsi qoyuruq.

```
;
```

Bunları bir-birinə bitişik və ya məsafə ilə aralı yaza bilərik. Yəni

```
cout<<Melumat;
```

ilə

```
cout     <<       Melumat      ;
```

eyni kod hesab olunur.

Bəs cout operatoru ilə hansı tip məlumatları çap etmək olar. cout bizə eyni qayda ilə müxtəlif tip məlumatları çap etməyə imkan verir, o cümlədən aşağıdakı:

- sətirlər
- simvollar
- tam ədədlər
- onluq kəsrlər
- riyazi ifadələr
- xüsusi simvollar
- dəyişənlərin qiyməti və ünvanı
- kompüterin hər-hansı ünvanında yerləşən məlumat
- müxtəlif funksiyaların qiymətləri v.s.

Bunların bəziləri ilə irəlidəki mövzularımızda tanış olacayıq. Bəzilərini isə isə bu paraqrafda örgənəcəyik.

2.2 Sətirlər

C++ dilində **cüt dırnaq** işarələri arasında verilən istənilən yazı sətir hesab olunur. Burada C++ proqramlarında tez-tez istifadə edəcəyimiz cütdırnaq və təkdırnaq işarələrini qeyd etmək istəyirəm.

Qeyd:

Cütdırnaq işarəsi aşağıdakı kimi:

```
"
```

təkdırnaq isə aşağıdakı kimi işarə olunur:

```
'
```

Misal üçün böyük A simvolunu təkdırnaq arasında yazaq:

```
'A'
```

və ya Ahmed sözün cütdırnaq arasında yazaq:

```
"Ahmed"
```

Burda nəyə diqqət yetirməliyik? Ola bilər ki təkdırnaq arasında heçnə yazmayaq, aşağıdakık kimi:

```
''
```

Bu zaman onu bir ədəd cütdırnaq işarəsindən fərqləndirmək lazımdır. İrəlidə xüsusi simvolları keçərkən bu barədə daha ətraflı danışacağıq. İndi qayıdaq sətirlərə.

C++ dilində cütdırnaq işarələri arasında göstərilmiş istənilən simvollar ardıcıllığı **sətir** adlanır.

Sətirlərə nümunə:

```
"C++"
"Men derse gedirem. Hami telesir."
"Proqramlashdirma"
"5 + 2 goresen nece eliyir"
"5 + 2"
"riyazi hesab emelleri: + - / % * "
```

Göstərilən nümunələrdən o aydın olmalıdır ki C++ kompilyatoru üçün sətrin içərisində yazılanların elə də əhəmiyyəti yoxdur. C++ sətrin içində yazılanları belə desək şəkil kimi görür. Yəni əgər sətrin içində hansısa riyazi ifadə yazmışıqsa, misal üçün 5 + 2 C++ kompilyatoru onun riyazi ifadə olduğunu bilmir və onu hesablayıb nəticə verə bilməz.

Əlbəttə sətirlərin emalı proqramlaşdırmanın ən mühüm hissələrindən birini təşkil edir və sətirlərlərin emalı ilə bağlı ətraflı paraqraf 9-da tanış olacayıq. Hələik isə sətirləri cout vastəsilə yalnız ekranda çap edəcəyik və bunun üçün bilməli olduğumuz bir vacib məqam var: Sətirləri çap edərkən onun mətni olduğu kimi ekranda çap olunur. Yəni siz istənilən bir mətni sətir kimi elan edib, yəni cütdırnaq işarələri arasına alıb cout ilə ekranda çap edə bilərsiniz. Nümunəyə baxaq.

Nümunə 1-1.

Ekranda **Men derse gedirem** sətrini çap edən kod tərtib edin.

Həlli:

Tələb olunan sətri ekranda çap eləmək üçün əvvəlcə onu C++ dilində sətir kimi elan etməliyik. Bunun üçün onu cütdırnaq işarəsi arasında yazmalıyıq:

```
"Men derse gedirem"
```

cout operatorunun sintaksisinə bir daha nəzər salaq.

```
cout << Melumat ;
```

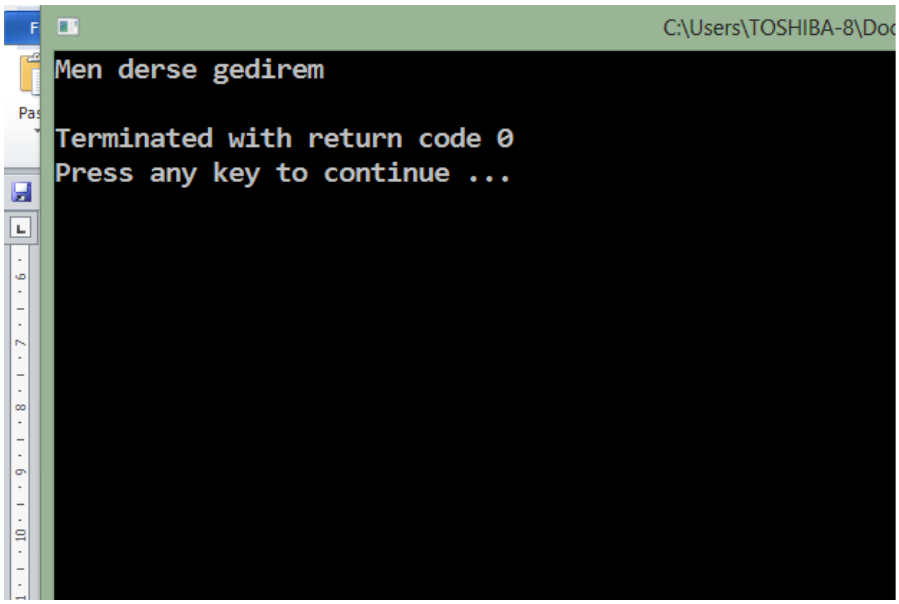
Baxdığımız halda məlumat olaraq `"Men derse gedirem"` sətrini çap etmək tələb olunur. Buna görə kod aşağıdakı kimi olar:

```
cout << "Men derse gedirem" ;
```

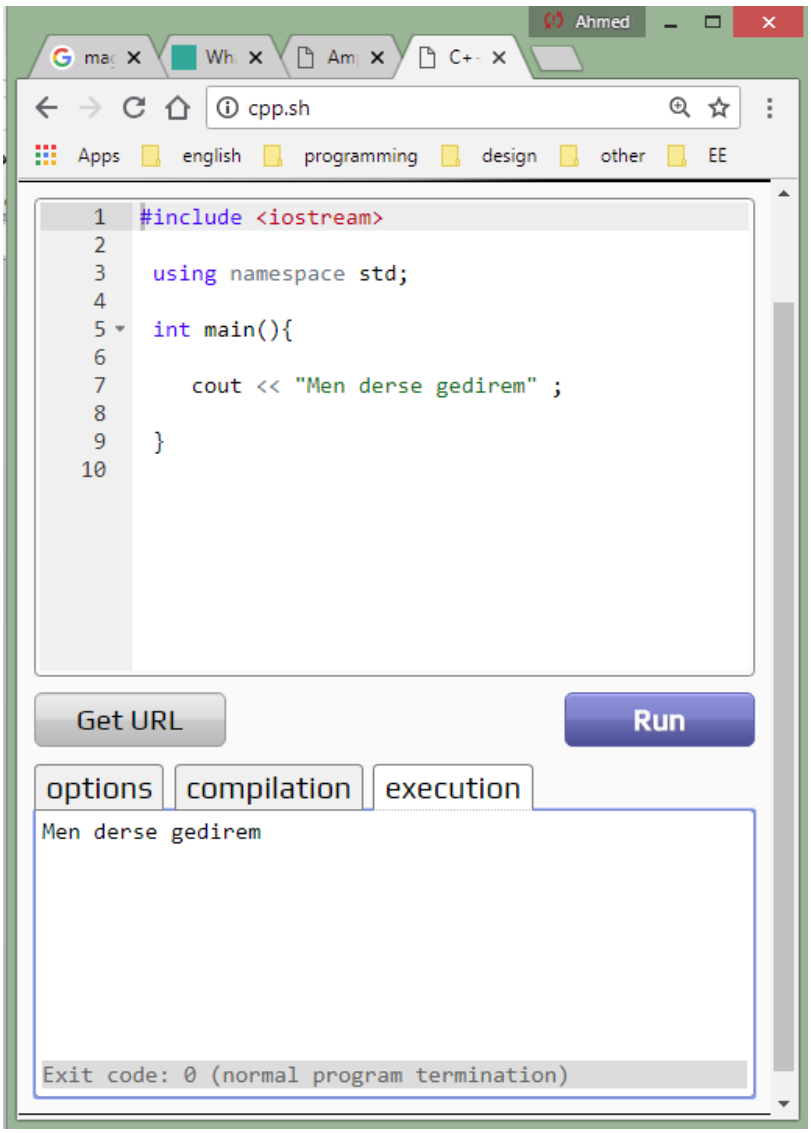
Əvvəlki paraqdafda ilk proqram nümunəsini daxil edərkən hələlik tərtib edəcəyimiz kodları main funksiyasının daxilində yerləşdirməli olduğumuzu demişdik. Bunları nəzərə alsaq son olaraq C++ dilində ekranda **Men derse gedirem** sətrini çap edən yekun proqram kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main(){

   cout << "Men derse gedirem" ;

}
```

Əgər bu kodu kompilyasiya və icra eləsək aşağıdakı nəticəni verər:



Yuxarıdakı nəticə MinGW İDE –sinin kansol pəncərəsində çap olunub. cpp.sh –da nəticə aşağıdakı kimi olar:



Gördüyümz kimi cütdırnaq işarələri çap olunmur, çünki kompilyator cütdırnaq işarələrinin arasında olan məlumatı sətir kimi qəbul edir və çap edir. Bu zaman maraqlı sual yaranır bəs əgər bizə ekranda cütdırnaq işarəsi çap eləmək lazım olsa onda necə yazmalıyıq? Əgər biz cütdırnağın özünü digər cütdırnaq işarələri arsına alsaq, aşağıdakı kimi:

```
cout << " " " " ;
```

Bu zaman komilyator ilk iki cütdırnaq simvolunu sətir kimi qəbul edər, növbəti cütdırnaq simvolunu isə tamamlanmamış sətir kimi qəbul edər və bizə sintaksis səhimizin olduğunu bildirər.

```
cout << "Ahmed" ;
```

Bəs onda bizə cütdırnaq işarəsinin çap etmək lazım olsa onda nə etməliyik? Bu barədə biraz düşünün, irəlidə xüsusi simvollarla tanış olarkən onu izah edəcəyik, hələlik isə çap etmək ilə bağlı nümunələrimizi davam edək.

Digər bir proqram koduna baxaq:

```
#include <iostream>

using namespace std;

int main (){

   cout << "Menim adim      Emildir." ;

}
```

Artıq bildiyimiz kimi bu proqramı icra etsək cütdırnaq arasında yazdığımız sətir ekranda çap olunar, aşağıdakı kimi olar:

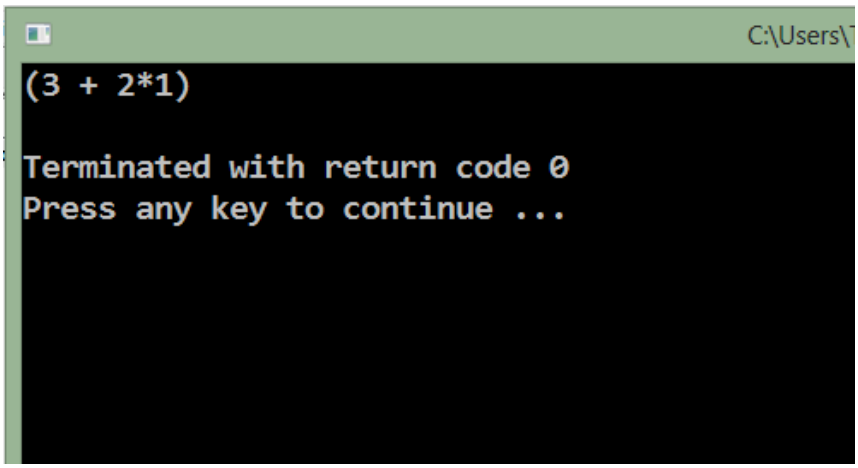
```
Menim adim      Emildir.
```

Gördüyümüz kimi cout operatoru cütdırnaq arasında sözlər arasındakı məsafə simvollarını da nəzərə alır, yəni olduğu kimi çap edir(as – is ingilis dilində).

Çalışma 1 -1. Aşağıdakı kod icra olunsa ekranda nə çap olunar?

```
cout << "(3 + 2*1)";
```

Göstərilən proqram kodunda məlumat olaraq `"(3 + 2*1)"` sətri çap olunur. Gördüyümüz kimi bu adi sətirdir və bu sətirdə cütdırnaq simvolları arasında `(3 + 2*1)` ifadəsi yazılıb. Bu bir riyazi ifadə olsa da cütdırnaq simvolları arasında yazıldığına görə, yəni cout –a sətir kimi təqdim olunduğuna görə cout onun riyaziyyat tərəfi ilə qəti maraqlanmır və olduğu kimi, yəni cütdırnaq arasında nə varsa hamısını eyni cür ekrana göndərir. Beləliklə yuxarıdakı kodu icra eləsək ekranda `(3 + 2*1)` sətri çap olunar.



2.3 Simvolların çapı

C++ dilində simvollar bir cüt təkdırnaq arasında göstərilir, aşağıdakı kimi:

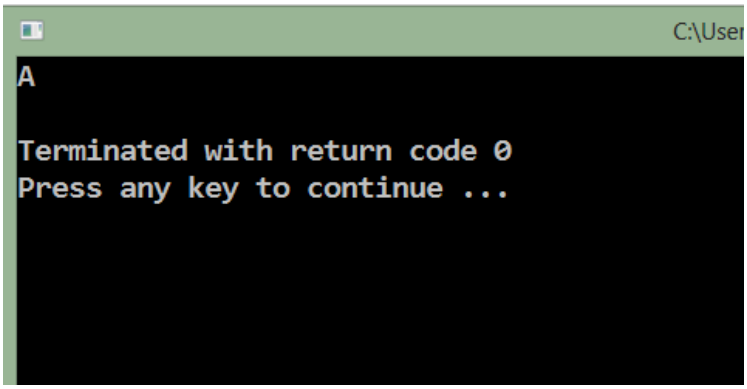
`'A'` – Böyük A hərfi simvolu
`'>'` – Böyükdür işarəsi simvolu
`'%'` – faiz işarəsi simvolu
`'('` – Açan mötərizə işarəsi simvolu
`'5'` – 5 rəqəmi simvolu

Simvolları bildirən təkdırnaq işarələri arasında adətən bir dənə simvol göstərilir. Əgər təkdırnaq işarələri arasına birdən çox sayda simvol yazsaq bu zaman kompilyator bizə xəbərdarlıq edər və ilk simvoldan başqa yerdə qalanlarını inkar edər.

Simvolları da eynilə sətirlər kimi cout vastəsilə çap edə bilərik. Misal üçün böyük A simvolunu aşağıdakı kod ilə çap etmək olar:

```
cout << 'A';
```

Bu zaman ekranda böyük A hərfi çap olunar, aşağıdakı kimi:



Əlbəttə böyük A hərfini biz cütdırnaq arasında da yazmış olsaydıq da yenə də eyni nəticəni alardıq,

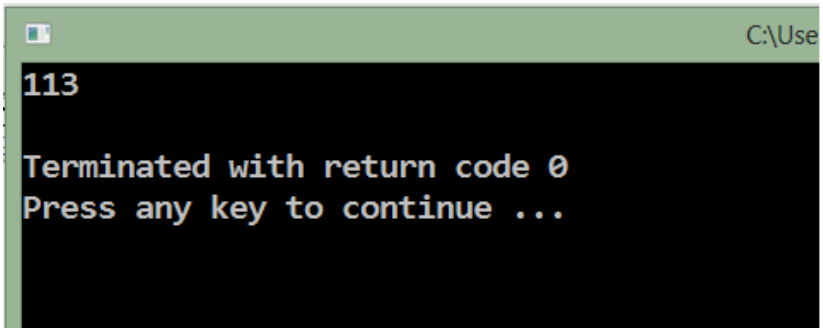
```
cout << "A";
```

Bu zaman məntiqli bir sual yaranır ki, əgər çap üçün sətirlər kifayət edirsə, onda somvollara nə gərək var. Açıq etiraf edim ki, çap üçün sətirlər tam kifayət edir. Lakin əlbəttə ki digər məsələlərdə sətirlərlə simvollar fərqlənir və bu barədə daha ətraflı irəliləyən paraqraflarda tanış olacağıq.

2.4 Tam ədədlərin çapı

cout ilə çap edə biləcəyimiz növbəti məlumat tipi tam ədədlərdir. Tam ədədlər dedikdə onluq hissəsi olmayan ədədlər nəzərdə tutulur, misal üçün 3, 5, 17, v.s. Bu zaman ədədləri cütdırnaq və ya təkdırnaq arasına yazmağa ehtiyac yoxdur. Misal üçün ekranda 113 ədədini aşağıdakı kimi çap edə bilərik:

```
cout << 113 ;
```



Təbii ki bu nəticəni biz 113- ü sətir kimi çap etməklə də ala bilərdik:

```
cout << "113";
```

və ya 1 və 3 simvollarının köməyi ilə aşağıdakı kimi:

```
cout << '1' ;
cout << '1' ;
cout << '3' ;
```

Əgər çap edəcəyimiz sadəcə bir ədəddirsə bu zaman onun sətir kimi çap olunmaqdan özünüz də gördüyünüz kimi fərqi yoxdu. Amma riyazi ifadələri tərkibində sətir ilə ədəd kimi çap olunmanın fərqi var və biz bunu tezliklə yoxlayacayıq.

2.5 Onluq kəsrlər

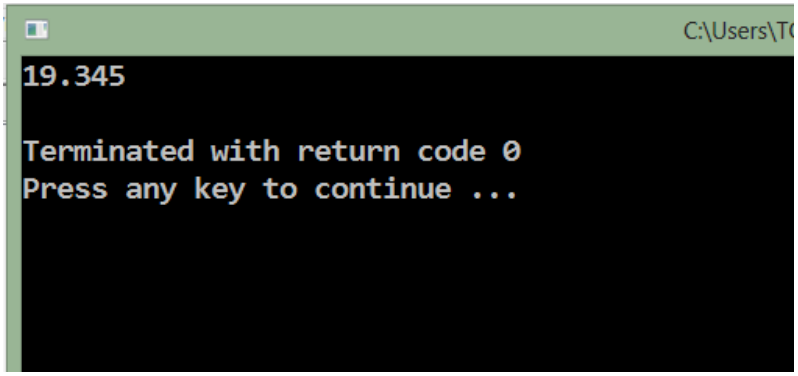
C++ dilində onluq kəsrlər tam hissə və kəsr hissə nöqtə ilə ayrılmaqla yazılır, məsələn:

```
12.45, 24.567, 83.021
```

Onluq kəsrləri də eynilə tam ədədlər kimi çap edirik

```
cout << 19.345 ;
```

Nəticə aşağıdakı kimi olar:



Tam ədədlərdə olduğu kimi onluq kəsrləri də sətir və ya simvolların köməyi ilə çap edə bilərik.

```
cout << "19.345";
```

və ya

```
cout << '1' ;
cout << '9' ;
cout << '.' ;
cout << '3' ;
cout << '4' ;
cout << '5' ;
```

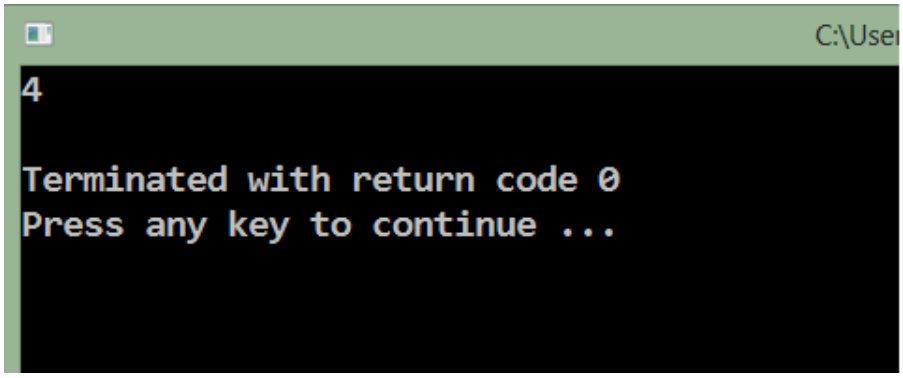
2.6 Riyazi ifadələrin çapı

İndiyə kimi çap elədiyimiz məlumatlar sətir, simvol və ya ədəd olduğu kimi çap olunurdu. Yəni sanki cout onların şəklini çəkib ekrana göndərirdi. İndi cout –un çap ilə yanaşı yeni bir funksiyası - riyazi ifadələri hesablamaq funksiyası ilə tanış olacayıq.

cout operatoru riyazi ifadələri hesablayıb nəticəni çap edə bilir. Bunun üçün sadəcə nəticəsini hesablamaq istədiyimiz riyazi ifadəni cout ilə çap etməliyik, misal üçün aşağıda kimi:

```
cout << 2*2 ;
```

Bu zaman C++ kompilyatoru çap olunmalı məlumatın riyazi ifadə olduğunu anlayır və əvvəlcə onun qiymətini hesablayır, daha sonra isə **yalnız** nəticəni çap edir. Yəni yuxarıdakı kodu icra etsək ekranda ancaq `2*2` riyazi ifadənin qiyməti 4 çap olunar.



Gəlin bir çox yeni başlayanların xüsusilə daha çox səhv elədiyi məqama bir daha nəzər salaq:

```
cout << "2*2";
```

`2*2` cütdırnaq arasına alındığı üçün C++ onları sətir kimi qəbul edir və olduğu kimi ekranda çap edir. Nəticədə çap olunur 2*2.

```
cout << 2*2;
```

Cütdırnaq olmadığına görə C++ 2*2 –ni riyazi ifadə olaraq tanıyır və əvvəlcə onun nəticəsini hesablayır və çap edir. Nəticədə 4 çap olunar.

cout ilə istənilən qədər mürəkkəb riyazi ifadələrin qiymətini hesablamaq olar.

Riyazi ifadələri tərtib edərkən mötərizələrdən () və aşağıdakı riyazi operatorlardan istifadə edə bilərik:

```
*   Vurma
/   Bölmə
-   Çıxma
+   Toplama
%   Qalıq hesablama
```

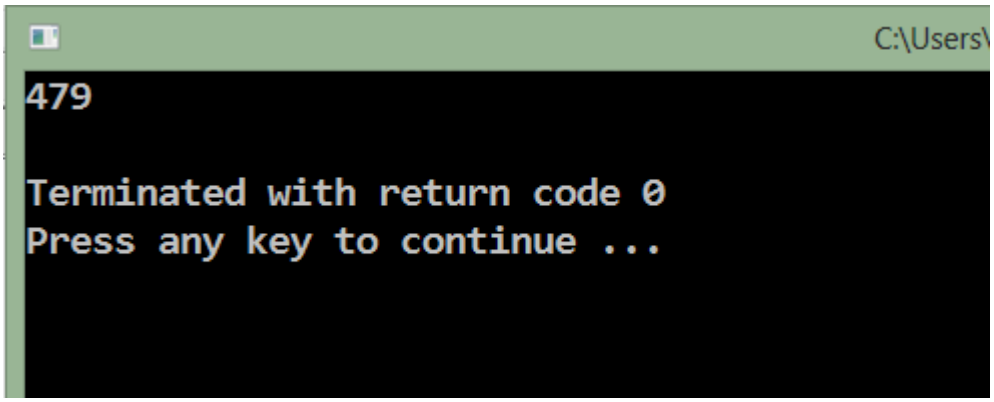
Çalışma 1-2. Aşağıdakı riyazi ifadənin qiymətini hesablayan proqram tərtib edin:

```
34 + 89 * (22 - 17)
```

Koda aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

   cout << 34 + 89 * (22 - 17) ;

}
```

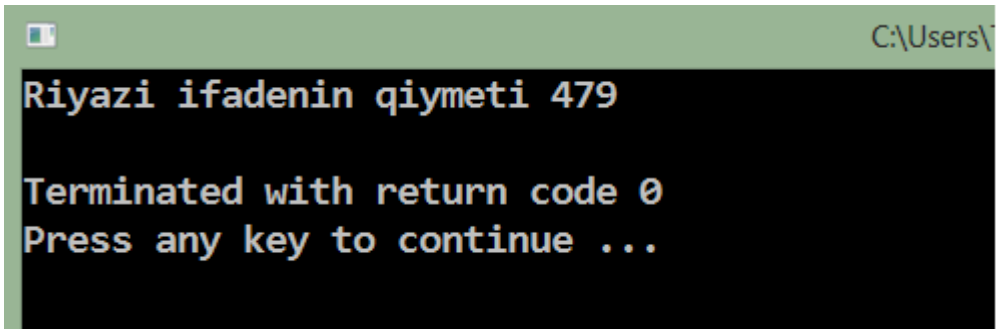
Bu kodu icra eləsək aşağıdakı nəticəni alarıq:



Bu kodda diqqətimizi çəkən məqamlardan biri də odur ki, proqram sadəcə verilmiş riyazi ifadənin nəticəsini çap edir. Yəni çap olunan nəticənin nəyi bildirdiyini anlamaq olmur. Adətən proqramlar çap elədiyi məlumata əlavə izahaedici şərhlər də qatsa bu zaman ondan istifadə daha da rahat olar. Misal üçün yuxarıda baxdığımız nümunə kodda istifadəçi üçün nəticə ilə yanaşı hansı əlavə izahedici məlumat da çap edə bilərik?

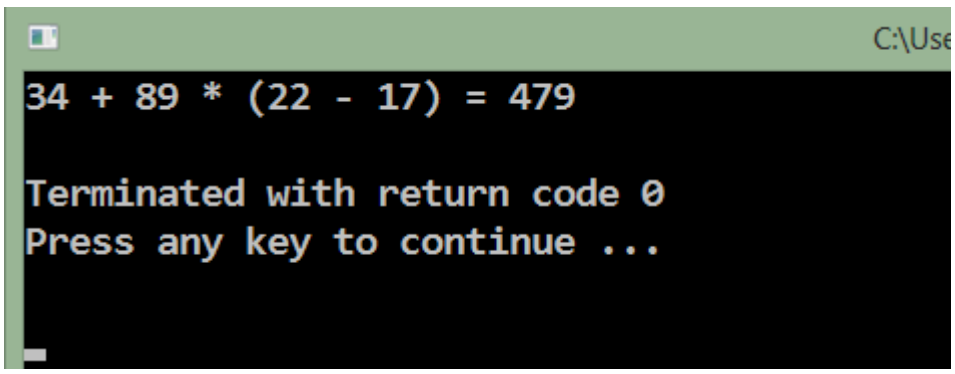
Misal üçün 429 –un bir riyazi ifadənin nəticəsi olduğunu istifadəçiyə bildirək. Aşağıdakı koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

    cout << "Riyazi ifadenin qiymeti ";
    cout << 34 + 89 * (22 - 17) ;

}
```



Təbii ki bu zaman birinci variantdan fərqli olaraq 479 –un hansısa bir riyazi ifadənin qiyməti olduğu istifadəçiyə bildirilir. Lakin biraz da irəli gedərək və cout –un imkanlarından istifadə edərək biz nəinki riyazi ifadənin nəticəsini, həm də çox asanlıqla bu nəticənin konkret hansı riyazi ifadəyə aid olmasını da istifadəçiyə bildirə bilərik. Kod belə olar:

```
#include <iostream>

using namespace std;

int main (){


    cout << "34 + 89 * (22 - 17) = ";
    cout << 34 + 89 * (22 - 17) ;

}
```



Artıq aydın olmalıdır ki, birinci cout çütdırnaq işarəsi arasında yazılanı, ikincisi isə verilmiş iafədənin nəticəsini çap edir. Bu imkanlardan yararlanaraq, yəni hər-hansı ifadəni cütdırnaq

arasında sətir kimi yerləşdirərək onu olduğu kimi çap etməklə və cütdırnaqdan kənar verib nəticəsini çap etmək kimi manevrlərdən istifadə etməklə biz istifadəçilər üçün çox rahat, oxunaqlı proqram interfeysləri tərtib edirik.

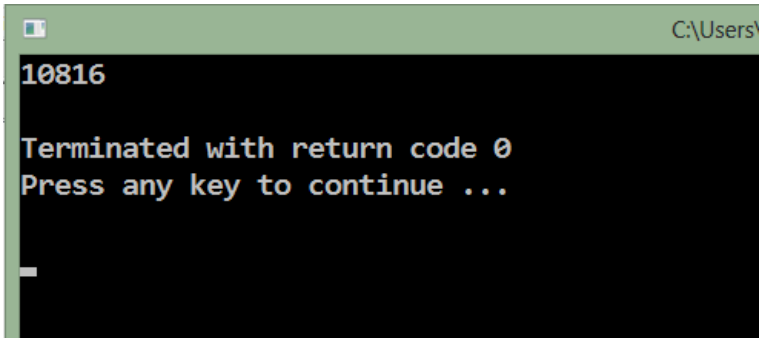
Çalışma 1-2. Aşağıdakı riyazi ifadənin qiymətini hesablayan proqram tərtib edin:

```
1024 + (8 + 4*(23*76 - 56 + (4+9)*(81 - 23)))
```

Tələb olunan riyazi ifadənin qiymətini hesablamaq üçün onu cout ilə çap etməyimiz kifayətdir. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    cout << 1024 + (8 + 4*(23*76 - 56 + (4+9)*(81 - 23))) ;

}
```

Nəticə belə olar:



2.7 Xüsusi simvollar

C++ dilində müxtəlif məqsədlər üçün xüsusi simvol adlandırılan simvollardan istifadə olunur. Bu xüsusi simvolların hər-biri birincisi arxaya bölmə olmaqla müxtəlif digər simvolların birləşməsindən əmələ gəlir. C++ dilində aşağıdakı xüsusi simvollar təyin olunub:

```
\"
\'
\\
\n
\t
\0
```

Gördüyümüz kimi bu simvolların hər-biri birincisi arxaya bölmə yəni \ simvolu olmaqla cütdırnaq - ", təkdırnaq - ' , təkrar arxaya bölmə - \ və digər simvollardan əmələ gəlir. Xüsusi simvolları tərtib edərkən birinci arxaya bölmə simvolu ilə ikinci simvol arasında məsafənin olmamasına diqqət yetiməliyik. Yəni misal üçün yeni sətir simvolunda \n arxaya bölmə \ ilə n arasına məsafə yerləşdirsək \ n artıq yeni sətir simvolunu itirmiş olarıq.

Nədir bu simvolları xüsusi edən?

Adi halda biz bu simvolların hər birinin nə məqsəd daşıdığını bilirik:

" – Sətri açır və ya bağlayır
' – Simvolu açır və ya bağlayır
\ – Xüsusi simvolun başladığını bildirir
n – n simvolunu bildirir
t – t simvolunu bildirir
0 – Sıfır simvolunu bildirir

Elə ki bu simvolların hansısa birinin əvvəlinə (arada məsafə olmadan) arxaya bölmə simvolu yerləşdirildi, onda vəziyyət təmamilə dəyişir və bu simvollar tam fərqli məna daşımağa başlayır.

```
\"
\'
\\
\n
\t
\0
```

Gəlin proqramlaşdırmada çox tex-tez rastlaşacağımız bu xüsusi simvolların hər-biri ilə ayrı-ayrılıqda tanış olaq.

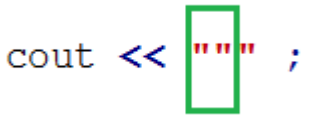
2.7.1 Cütdırnaq simvolu - \"

Bildiyimiz kimi C++ dilində cütdırnaq simvolundan sətir elan etmək üçün istifadə edirik. Belə ki birinci cütdırnaq simvolu sətri açır, növbəti isə bağlayır və sətir tamamlanmış olur. Lakin bu simvolun əvvəlinə arxaya bölmə simvolu yerləşdirdikdə artıq bu simvol sətir açıb-bağlama funksiyalarını itirir və özünü sətir daxilində çap olunan digər simvollar kimi aparır. Adətən cütdırnaq xüsusi simvolundan ekranda cütdırnaq simvolunun özünü çap etmək tələb olunduqda istifadə edirik.

Adi halda bunu edə bilmərik, yəni əgər aşağıdakı kimi ekranda cütdırnaq simvolunun özünü çap eləmək üçün onu digər iki cütdırnaq simvolları arasına yerləşdirsək

```
cout << """ ;
```

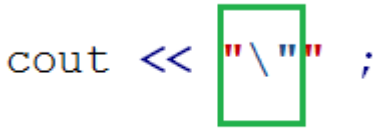
onda ilk iki cütdırnaq simvolu sətri qapayacaq və C++ kompilyatoru bizə 3-cü cütdırnaq simvolunun natamam sətri açdığını bildirəcək:



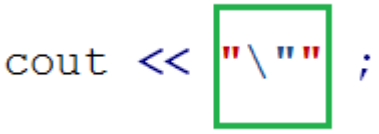
Beləliklə bu şəkildə ekranda cütdırnaq simvolu çap eləmək cəhdimiz alınmaz. Çıxış yolu çap etmək istədiyimiz cütdırnaq simvolunu qeyd etdiyimiz kimi xüsusi simvol kimi sətir daxilində yazmaqdır, aşağıdakı kimi:

```
cout << "\"" ;
```

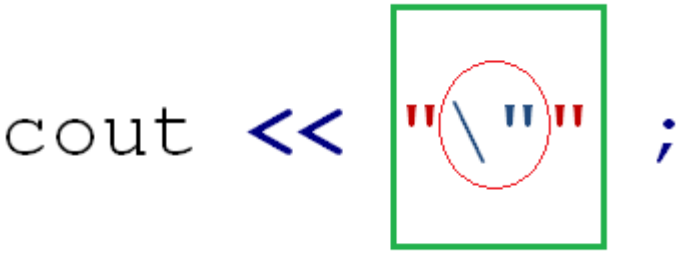
Bu halda ikinci cütdırnaq simvolundan əvvəl arxaya bölmə simvolu gəldiyinə görə o sətir bağlamaq funksiyasını itirmiş olur, yəni aşağıdakı kimi bir sətir əmələ gəlməsinin qarşısı alınır



Əksinə aşağıdakı kimi sətir alınır:



Hansı ki özündə məlumat olaraq yalnız bir xüsusi simvolu saxlayır:



Əgər bu kodu icra eləsək onda ekranda nə çap olunar

```
\"
```

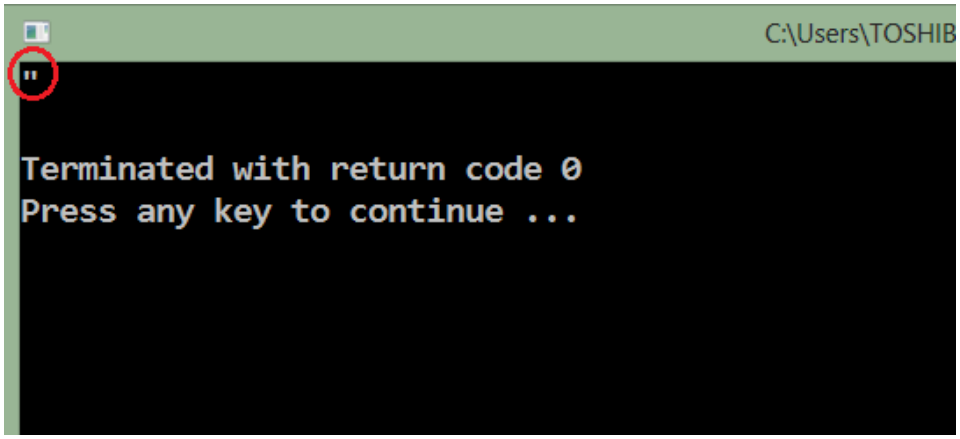
yoxsa

```
"
```

?

Əlbəttə ki yalnız cütdırnaq işarəsi çap olunar, arxaya bölmə simvolunun funksiyası həmin cütdırnağın sətri bağlamasının qarşısını almaq və onun adi simvol kimi çap olunmasını təmin etmək idi. Proqramın tam mətnin və nəticəsini daxil edək:

```
#include <iostream>
using namespace std;
int main (){

    cout << "\"";
}
```



Artıq cütdırnaq simvolunun özünün ekranda nə cür çap edilməsini örgəndikdən sonra bir qədər maraqlı ifadələr çap edə bilərik.

Çalışma ekranda Ahmed sözünü cütdırnaq arasında çap edən kod tərtib edin.

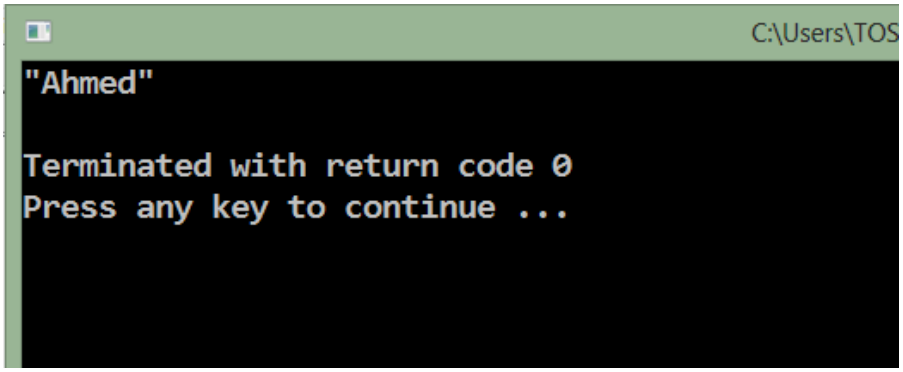
Həlli. Bu nəticəni almaq üçün cout ilə aşağıdakı sətir çap edə bilərik

```
cout << "\"Ahmed\"";
```

Tam kod və nəticə aşağıdakı kimi olar:

```
#include <iostream>
using namespace std;
int main (){
```

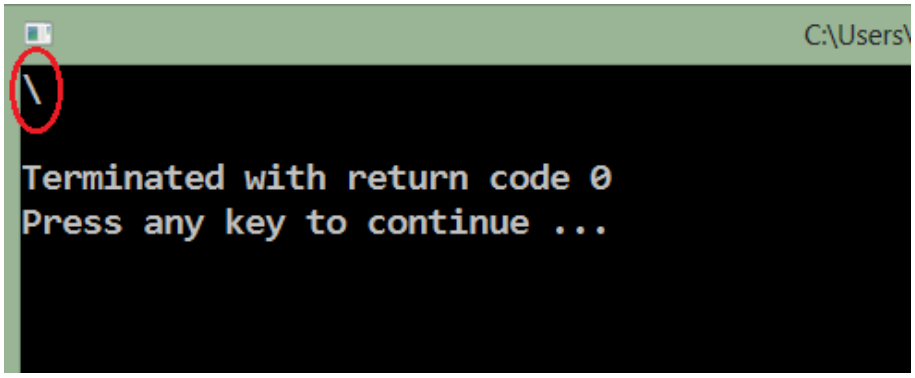
```
    cout << "\"Ahmed\"";
}
```



Eyni qayda ilə ekranda arxaya bölmə simvolunun özünü çap etmək istəsək aşağıdakı kimi kod tərtib etməliyik

```
cout << "\\";
```

Nəticədə ekranda arxaya bölmə simvolu çap olunar



2.7.2 Yeni sətir simvolu

Xüsusi simvollardan bəlkə də ən çox istifadə olunanı yeni sətir simvoludur. Yeni sətir simvolu arxaya bölmə simvolu \ ilə n simvolunun birləşməsindən ibarətdir:

```
\n
```

Bu simvolu çap etdikdə nə baş verir?

cout ilə iki müxtəlif sətri ardıcıl çap etsək, misal üçün aşağıdakı kimi, ekranda hansı nəticə alınar?

```
cout << "Bir bayt sekkiz bitdir" ;
cout << "Bir kilobayt 1024 baytdir" ;
```

Əlbəttə biz gözləyirik ki, bu iki sətir alt-alta çap olunacaq, amma bu belə deyil. Səbəb isə odur ki, **cout** operatoru məlumatları ardıcıl çap edir. Yəni bu kodu icra eləsək aşağıdakı nəticəni alarıq:

```
Bir bayt sekkiz bitdirBir kilobayt 1024 baytdir
```

Gördüyümüz kimi bu iki sətir alt-alta deyil ardıcıl, bir-birinə bitişik çap olundu. Bəs iki sətri **alt-alta** necə çap edə bilərik?.

Bu üçün **yeni sətir simvolundan** istifadə olunur. Yeni sətir simvolu **cout** operatoruna cari sətirdə çap etməni dayandırmağı, növbəti sətrə keçməyi və çapetməni növbəti sətirdən davam etdirməyi bildirir. Aşağıdakı kimi:

```
#include <iostream>

using namespace std;

int main (){

    cout<<" Bir bayt sekkiz bitdir ";
    cout<<"\n" ;
    cout<<"Bir kilobayt 1024 baytdir";

}
```

Bu zaman nəticə aşağıdakı kimi olar:

```
Bir bayt sekkiz bitdir
Bir kilobayt 1024 baytdir
```

Əgər yeni sətir simvolun ardıcıl olaraq bir neçə dəfə çap eləsək onda həmin say qədər yeni sətrə keçid baş verər, yəni arada boş sətirlər buraxarıq, aşağıdakı kimi:

```
#include <iostream>

using namespace std;

int main (){

    cout<<" Bir bayt sekkiz bitdir ";
    cout<<"\n" ;
```

```
    cout<<"\n" ;
    cout<<"\n" ;
    cout<<"\n" ;
    cout<<"Bir kilobayt 1024 baytdir";

}
```

Bu zaman nəticə aşağıdakı kimi olar:

```
Bir bayt sekkiz bitdir



Bir kilobayt 1024 baytdir
```

Yeni sətir simvolunu cütdırnaq arasında da vermək olar:

```
#include <iostream>

using namespace std;

int main (){

    cout<<"Bir bayt \n sekkiz bitdir ";

}
```

Bu zaman nəticə aşağıdakı kimi olar:

```
Bir bayt
 sekkiz bitdir
```

Sual: Əgər diqqət edirsizsə altdakı sətir bir qədər sağa sürüşüb, bunun səbəbi nədir?

2.7.3 Tabulyasiya simvolu

Tabulyasiya simvolu `'\t'` kimi işarə olunur. Bu simvoldan məlumatları çap edərkən sütunlar üzrə nizamlamaq üçün istifadə olunur. Nümunə koda baxaq:

```
#include <iostream>
```

```
using namespace std;

int main (){

    cout << "123.3 / 5.89 \t = " <<  123.3 / 5.89  << "\n";
    cout << "21 / 3.9 \t = " <<  21 / 3.9  << "\n";
    cout << "4.456 / 1.965 \t = " <<  4.456 / 1.965  << "\n";

 }
```

```
123.3 / 5.89      = 20.9338
21 / 3.9          = 5.38462
4.456 / 1.965     = 2.26768
```

Bu nəticəni oxumaq daha rahatdı.

2.8 Axına əlavə et

cout operatoru ilə biz məlumatı proqramdan xaric edirik. İndiyə kimi çap elədiyimiz hər yeni məlumat üçün ayrı bir **cout** operatorundan istifadə edirdik. Lakin axına əlavə et operatoru ilə cəmi bir **cout** -dan istifadə etməklə istədyimiz qədər məluamtı çap edə bilərik. Əlbəttdə heç kim bizə hər məlumat üçün ayrıca **cout** operatorundan istifadə etməyimizi qadağan etmir, burada seçim zövq məsələsinə bağlıdır.

Axına əlavə et operatoru aşağıdakı kimidir:

```
    <<  melumat
```

Bir neçə məluamtı əlavə etmək istəsək

```
     <<  melumat1 <<  melumat2 <<  melumat3 ... v.s.
```

Misal üçün yalnız bir std::cout operatorundan istifadə etməklə axına əlavə et operatorunun köməyi ilə ekranda alt-alta `"Adil"` və `"Emin"` sətirlərini ardıcıl çap edən proqram aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

    cout << "Adil" << "\n" <<"Emin";

}
```

Axına əlavə et operatorlarını ardıcıl və ya alt-alta yazmağımız da zövq məsələsidir:

```
#include <iostream>

using namespace std;

int main (){

    cout << "Adil"
         << "\n"
         <<"Emin";

}
```

Nöqtəvergülü ən axırda qoyuruq. Hər-iki halda nəticə belə olar:

```
Adil
Emin
```

2.9 Proqramda Şərhlər

İlk mövzumuzu tamamlamadan öncə tanış olmalı olduğumuz daha bir məsələ də **proqram şərhləri** anlayışıdır. Çox sadə bir məsələ olduğuna görə onu da ilk mövzuya daxil etdik. Lakin sadə olmaqlarına baxmayaraq şərhlər proqramlaşdırmada çox əhəmiyyətli rol oynayır. Düzdür real icra olunan kod hissəsi ilə şərhlərin heç bir bağlılığı yoxdur, amma proqramçıların özləri, ya da onların tərtib etdikləri kodu sonradan digər proqramçıların asanlıqla anlaya bilməsi üçün şərhlərdın istifadə etmək çox mühümdür.

Şərhləri proqram koduna onun bu vəya digər hissəinin gördüyü işi bildirmək üçün qoyurlar. Sual verərsiniz ki, məgər proqramçı özü tərtib elədiyi kodun nə iş gördüyünü bilmir ki, əlavə bir də şərh yazsın həmin hissəyə. Əlbəttə bilir. Lakin real tətbiqlər bəzən bir neçə yüz min, hətta bir neçə milyon sətir koddan ibarət ola bilər və yalnız bir proqraçının deyil, onlarla proqramçıların birgə əməyi ilə yaradıla bilər. Bu zaman proqramçı özünün tərtib etdiyi yüz min sətir kod içində, onun hansısa hissəsinin nə iş gördüyünü bir aydan sonra asanlıqla yadından çıxara bilər. O ki, qaldı digər proqramçılara hansı ki siz tərtib etdyiniz kodu oxuyub başa düşmək və inkişaf etdirmək istəyirlər. Şərhlərsiz sizin kodu sonralar başqaları üçün anlamaq çox çətin olar. Ona görə həm özünüz üçün, həm də sonradan siniz kodu oxuyacaq digərləri üçün şərhlərdən istifadə etməyi unutmayın. Şərhlərdən istifadə etməyi özünüzə bir növü adət etdirin.

C++ dilində 2 cür şərhlərdən istifadə olunur: **çoxsətirli** və **təksətirli**.

2.9.1 Təksətirli şərhlər

Proqrama təksətirli şərh əlavə etmək üçün iki ardıcıl bölmə işarəsi qoyub şərhi yazmaq lazımdır, aşağıdakı kimi:

```
// bu təksətirli şərhdir
```

Proqramın istənilən yerinə təksətirli şərh yaza bilərik, nümunəyə baxaq:

```
#include <iostream>

using namespace std;


 int main (){

    // aşağıdakı kod 3 ilə 4 -ün cəmini hesablayır
    cout << "3 + 4 = " <<  3 + 4  << "\n";

 }
```

Təksətirli şərh iki ardıcıl bölmə ilə başlayır və yalnız cari sətrin sonuna qədər olan hissəyə aid olur. Əgər şərhimizin mətni çoxdusa onda həm bir neçə təksətirli şərh, həm də çoxsətirli şərhdən istifadə edə bilərik.

2.9.2 Çoxsətirli şərhlər

Çoxsətirli şərh ardıcıl bölmə və ulduz simvolları ilə başlayır və növbəti ardıcıl ulduz və bölmə simvoluna qədər olan hissəni əhatə edir. Nümunəyə baxaq:

```
#include <iostream>

using namespace std;

  int main (){

    /* aşağıdakı kod 3 -ü 4 -ə bölməndə alınan nəticəni
      hesablayır. Əgər 3 -ü qalıq ilə göstərməsək
      onda std::cout hər iki ədədi tam ədəd kimi qəbul
      edər və bölmə nəticəsində alınan yalnız tam hissəni
      çap edər.
    */
    cout << "3.0 / 4 = " <<  3.0 / 4  << "\n";

  }
```

Qeyd elədik ki, şərhlərin icra olunan kod ilə heç bir əlaqəsi yoxdur, kompilyator, şərhləri nəzərə almır. Bəzən isə proqramçılar bundan (kompilyatorun şərhləri nəzərə almamasından) proqram kodunda səhvləri tapmaq və ya digər məqsədlər üçün istifadə edirlər. Belə ki, proqramçılar iri proqram mətnində səhvin olduğunu ehtimal etdikləri hissəni şərh kimi elan edib kompilyasiya edirlər. Bəzən isə test məqsədilə proqramın bir hissəsini kompilyasiyadan kənarlaşdırmaq tələb olunur. Həmin hissəni şərh kimi göstərməklə bu məsələ həll olunur.

Tapşırıq 1. Ekranda adınızı çap edən proqram tərtib edin.

Tapşırıq 2. Ekranda adınızı və soyadınızı eyni sətirdə çap edən proqram tərtib edin.

Tapşırıq 3. Ekranda sizin barədə məlumatları aşağıdakı formatda çap edən proqram qurun.

```
Ad         : Ahmed
Soyad      : Sadikhov
Tevvelldu  : 2001.01.01
Mobil      : +994 00 000 00 00
```

Tapşırıq 1. Ekranda əvvəlcə bir sətirdə adınızı və soyadınızı, daha sonra isə altdakı sətirdə yaşadığınız ünvanı çap edin proqram tərtib edin.

Tapşırıq 2. Ekranda cütdırnaq - " işarəsi çap edən proqram tərtib edin

```
"
```

Tapşırıq 3. Ekranda öz adınızı cütdırnaq işarələri arasında çap edən proqram tərtib edin.

```
"Ahmed"
```

Tapşırıq 4. Ekranda xüsusi simvolların çapı zamanı istifadə olunan tərs bölmə - backslash simvolunu çap edən proqram tərtib edin.

```
\
```

Tapşırıq 5. Ekranda backslash simvolunu cütdırnaq arasında çap edən proqram tərtib edin.

```
"\"
```

Tapşırıq 6. Ekranda cütdırnaq simvolunu backslash simvolları arasında çap edən proqram tərtib edin.

```
\"\
```

Tapşırıq 7. Ekranda aşağıdakı riyazi ifadələrin qiymətlərini çap edən proqram tərtib edin.

```
1+2
1+2*(4 - 9)
22-37*5+78(2 + 41*(82 - 52))
```

Tapşırıq 8. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
* *
* *
```

Tapşırıq 9. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
* * *
```

```
* * *
* * *
```

Tapşırıq 10. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Tapşırıq 11. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
* * *
*   *
* * *
```

Tapşırıq 12. Ekranda bir-birindən 3 məsafə simvolu aralı iki ulduz simvolu çap edən proqram tərtib edin.

```
* * * * *
*       *
*       *
*       *
* * * * *
```

Tapşırıq 13. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
*
* *
```

Tapşırıq 14. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
*
* *
* * *
```

Tapşırıq 15. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Tapşırıq 16. Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

```
* * * * * * *
* + + + + + *
* + + + + + *
* + + + + + *
* + + + + + *
* + + + + + *
* * * * * * *
```

Tapşırıq 17. Ekranda ulduz simvollarından ibarət adınızın ilk hərfini iri miqyasda çap edən proqram tərtib edin.

Tapşırıq 18**.** Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

Tapşırıq 19**.** Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

Tapşırıq 20**.** Ekranda aşağıdakı fiquru çap edən proqram tərtib edin.

§3 Dəyişənlər

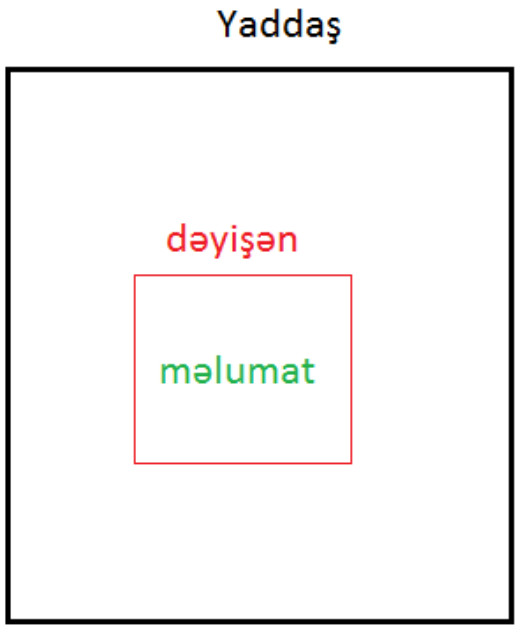
3.1 Dəyişən nədir?

Riyaziyyatdan az-çox təcrübəsi olanlar bu termin ilə tanışdırlar, funksiyanın dəyişəni v.s. Bu başdan deyim ki proqramlaşdırmada istifadə olunan dəyişən anlayışı riyaziyyatdan bildiyimiz dəyişənlə bir əlaqəsi yoxdur.

Proqramlaşdırmada dəyişəndən yaddaşda hər-hansı məlumat saxlamaq üçün istifadə edirlər. Proqram icra olunan zaman **məlumatlar** üzərində müxtəlif əməliyyatlar aparır. Məlumatlara misal olaraq aşağıdakıları göstərə bilərik:

İşçinin adı, aylıq maaşı, xərcləri, təvəllüdü, müəyyən bir ərazinin eni, uzunluğu, tələbənin topladığı ballar, şəklin hansısa nöqtəsinin rəngi, parlaqlığı, əlaqə qurulması gərəkən kompüterin şəbəkə ünvanı (ip, ay-pi), istifadəçinin adı, şifrəsi, hesab nömrəsi, bank hesabı v.s.

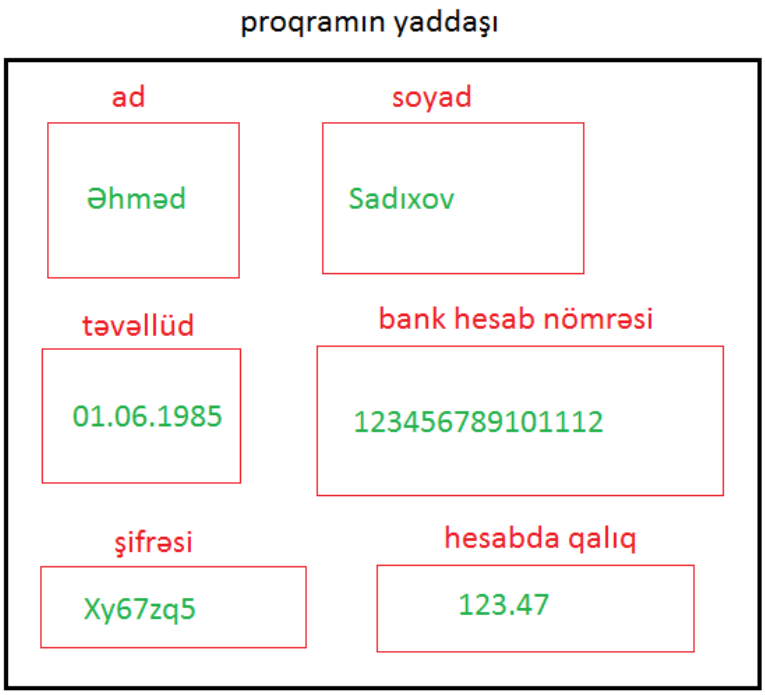
Bu məlumatları yaddaşda saxlamaq üçün və eləcə də lazım gəldikdə həmin bu məlumatları əldə etmək üçün dəyişənlərdən istifadə edirik.



Şəkil 1.

Mahiyyət etibarilə dəyişən yaddaşda məlumatın saxlanıldığı sahəni bildirir.

Yaddaşda istənilən **tipli** məlumat yadda saxlaya bilərik, bax şəkil 2.



Şəkil 2.

Yaddaşda saxlanan məlumatın həcmindən asılı olaraq onun üçün ayrılan yerin də, başqa sözlə həmin məlumatı özündə saxlayan dəyişənin də ölçüsü müxtəlif olur.

3.2 Dəyişənlərin adlandırılması

Dəyişənlərə müraciət etmək üçün onların adlarından istifadə edirik. Dəyişənlərə adları biz proqramçılar veririk və bu işdə tam sərbəstik, yəni istənilən dəyişənə istədiyimiz, ağlımıza gələn adı verə bilərik, misal üçün:

Eded, cem, S, deyishen, dey1, dey2, xerc, gelir, en, uzunluq, sahe v.s.

Gördüyünüz kimi yuxarıda sadaladığımız dəyişən adlarında həm böyük, həm kiçik hərflərdən və eləcə də rəqəmlərdən istifadə elədik. Lakin diqqət yetirirsinizsə yerli əlifbanın simvollarından yəni ə,ö,ğ v.s. kimi hərlərdən istifadə etmədik ki, bu da təsadüfi düyil. Dəyişənə ad verərkən aşağıdakı qaydaları nəzərə almalıyıq:

Dəyişənlərə ad verərkən aşağıdakı qaydalara əməl etməliyik:

1. Dəyişən adında yalnız ingilis əlifbasının simvolları, rəqəmlər və _, $ simvollarından istifadə edə bilərik.

2. Dəyişən adı mütləq ingilis əlifbasının simvolu ilə başlamalıdır.

3. Dəyişən adı təkrarlanmamalıdır.

4. Açar sözlərdən və əvvəlcədən elan olunmuş funksiya (bunlarla irəlidə tanış olacayıq) adlarından dəyişən adı kimi istifadə etmək olmaz.

Düzgün dəyişən adlarına misal: **`x, y, dey, dey1, dey2, cem, deyisen, S, s, vurma`** v.s.

Yanlış dəyişən adlarına misal: **`dəy`** (ə simvolundan istifadə olunur), **`5f`** (rəqəm ilə başlayır), **`vur%ma`** (% simvolundan istifadə olunur) v.s.

Bundan əlavə bilməliyik ki C++ dilində BÖYÜK və kiçik hərflər fəqrləndirilir. Yəni aşağıdakı adların hər biri ayrı bir dəyişən adını bildirir: **`SAHE, Sahe, SaHe, sahe`**

Qeyd: Dəyişənləri adlandırmada səhvə yol vermə yeni başlayanlar üçün tipik səhvlərdən sayılır. Nəticədə proqram uğurlu kompilyasiya olunmur.

Məsləhət: Dəyişənə ad verərkən mümkün qədər qısa və dəyişəndə yadda saxlanılan məlumata uyğun ad vermək lazımdır. Bu sizə həmin dəyişəndə hansı məlumatı yerləşdirdiyinizi yadda saxlamağa kömək edr.

3.3 Dəyişənin tipi

Dəyişənlərin adlandırılması qaydaları ilə tanış olduq. Dəyişənlərdən istifadəyə keçmədən öncə bilməli olduğumuz daha bir məhfum dəyişənin tipi anlayışıdır. Əgər dəyişənin adı onda olan məlumata müraciət etmək üçün istifadə olunurdusa, tip həmin məlumatın xarakterin, yaddaşda nə qədər yer tutmağın v.s. bildirir. Ümumiyyətlə proqramlaşdırmada kifayət qədər mürəkkəb tiplərdən istifadə olunur. Biz isə yeni başlayanda çox sadə 3 standart tipdən istifadə edəcəyik. Bunlar **`int`**, **`char`** və **`double`** tipləridir. Müvafiq olaraq bu tiplərdən olan dəyişənlərdə biz tam, simvol və kəsr tipli məlumatlar yerləşdirə bilərik.

3.4 Dəyişənin elan olunması

Yuxarıda tanış olduğumuz bilgilər bizə artıq dəyişənlərdən proqramlarımızda istifadə etməyə imkan verir. Proqramda istənilən bir dəyişəni hər-hansı hesablama, giriş/çıxış

v.s. digər əməliyyatlarda istifadə etmədə öncə **mütləq** həmin dəyişəni **elan** etməliyik. Dəyişəni elan edərkən biz onun yuxarıda qeyd etdiyimiz kimi adını və tipini göstəririk. C++ dilində dəyişənlərin elanı qaydası aşağıdakı kimidir:

```
int ad;
```

Əvvəlcə dəyişənin tipini, daha sonra isə adını yazıb nöqtəvergül qoyuruq. Misal üçün tam ədədlərlə işləmək üçün istifadə olunan **int** standart tipindən **x** adlı dəyişən elan etmək üçün aşağıdakı kimi yazmalıyıq:

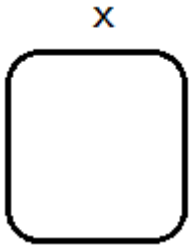
```
int   x ;
```

Bu zaman yaddaşda elan etdiyimiz **x** adlı dəyişənə yer ayrılır və biz orada müxtəlif tam tipli ədədlər yerləşdirə bilərik.

Proqramın tam kodu aşağıdakı kimi olar:

```
#include <iostream>

int main (){

  // deyishen elan edek
  int   x ;

}
```

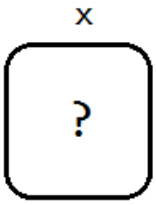
Yuxarıdakı kodda biz proqarmımızda yalnız bir dənə dəyişən elan etdik. Bu zaman x dəyişəni üçün yaddaşda yer ayrıldı:



lakin biz bu dəyişəni elan etdikdən sonra proqramımızda istifadə etmədik. Bəs dəyişəndən nə cür istifadə etmək olar? Ən sadə halda biz dəyişənin qiymətini çap edə bilərik. Misal üçün yuxarıda elan etdiyimiz x dəyişənin qiymətini cout operatorunun köməyi ilə aşağıdakı kimi çap edə bilərik:

```
cout << x ;
```

Bu zaman x dəyişəninin qiyməti(bundan sonra qısa olaraq x dəyişəni deyəcəyik) ekranda çap olunar. Bəs x dəyişəninin qiyməti neçədir?



Biz dəyişənini elan edərkən onun üçün yaddaşda yer ayrılır. Bu yerin nəcə ayrılmasına biz müdaxilə edə bilmərik, eləcə də bu yer ayrılarkən orda başlanğıc olaraq hansı məlumatın olmasını da bilə bilmərik. Bir çox proqramlaşdırma dilləri dəyişəni elan edərkən onun üçün ayırdıqları yerdə olan məlumatı silirlər, misal üçün Java. C++ belə dillərdən deyil və dəyişən elan edərkən ona başlanğıc qiyməti özümüz verməliyik. Əks halda dəyişəndə naməlum qiymət olacaq və bu qiymətdən istifadə sonradan çətin tapılan proqram səhvlərinə mənbə ola bilər. Başqa sözlə dəyişəni elan etdikdən sonra onu istifadə etmədən öncə mütləq ona hər-hansı qiymət verin. İndi biz bunu nə cür eləməyin qaydasını örgənəcəyik.

3.5 Dəyişənə qiymət mənimsədilməsi

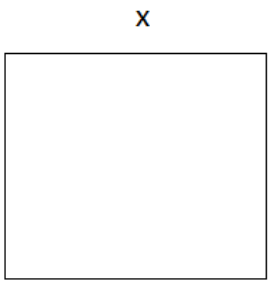
Dəyişənə qiymət mənimsətmək, başqa sözlə dəyişənə qiymət vermək o deməkdir ki, dəyişənin əvvəlki qiyməti silinir və o yenisi ilə əvəz olunur. Bunun üçün **mənimsətmə** operatorundan istifadə olunur. **Mənimsətmə** operatoru riyaziyyatdan çox yaxşı tanıdığımız bərabərlik = simvolu ilə işarə olunur (baxmayaraq ki proqramlaşdırmada bu simvol ayrı məna daşıyır) və sintaksisi aşağıdakı kimidir:

```
Dəyişən  =  Qiymət ;
```

Mənimsətmə operatorunun (bərabərlik simvolunun) sol tərəfində yeni qiymət mənimsətmək istədiyimiz dəyişən, sağ tərəfində isə həmin dəyişənə vermək istədiyimiz yeni qiymət yazılır. Yuxarıda elan etdiyimiz tam tipli x dəyişəninə gəlin hər-hansı qiymət mənimsədək. x dəyişənin elanını bir daha nəzərdən keçirək:

```
int x;
```

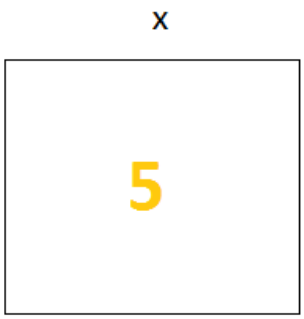
Dedik ki bu elan zamanı yaddaşda x dəyişəni üçün yer ayrılacaq:



x dəyişəninə 5 qiyməti mənimsətmək üçün yuxarıdakı sintaksisə əsasən yazmalıyıq:

```
x = 5;
```

Bu zaman **x** dəyişəninə 5 qiyməti mənimsədilir, yəni yaddaşda **x** dəyişəninə aid yerə 5 qiyməti yazılır, bax şəkil 6.



3.6 Dəyişənin qiymətinin ekranda çap edilməsi

Dəyişənin qiymətini istifadəçiyə bildirmək tez-tez tələb olunur. Bunun üçün onun qiymətini ekranda çap edə bilərik. Biz əvvəlki dərslərimizdə cout operatoru ilə ekranda sətir çap etməyi örgəndik. Dəyişənin qiymətinin çap olunması da cout ilə oxşar qaydada yerinə yetirilir.

```
cout << dəyişən ;
```

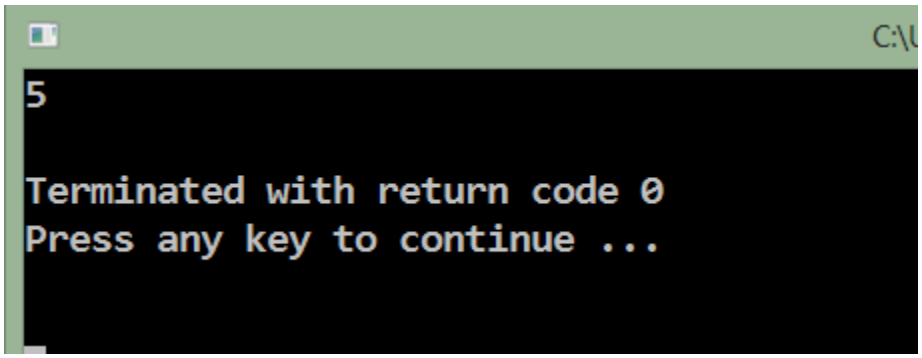
Bu zaman ekranda dəyişənin qiyməti çap olunacaq. Yuxarıdakı nümunədə x dəyişəninə 5 qiyməti mənimsətmişik. x dəyişənin qiymətini ekranda çap eləmək istəsək aşağıdakı kimi yaza bilərik:

```
cout << x ;
```

bu zaman x dəyişənin qiyməti – yəni 5 ədədi ekranda çap olunar. Proqramın tam kodu və icra nəticəsi aşağıda göstərilir.

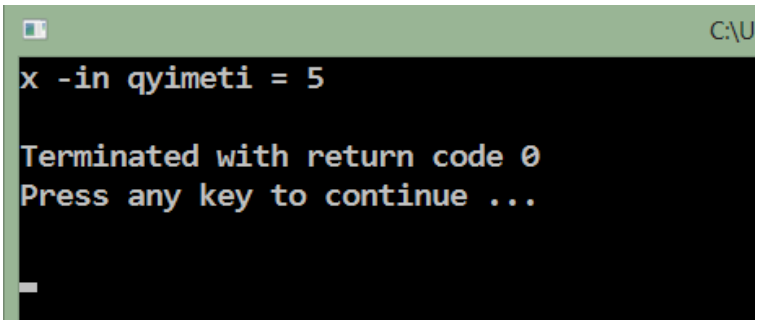
```
#include <iostream>
using namespace std;
int main (){
    int x;
    x = 5;
```

```
    cout << x;

}
```



Yuxarıdakı proqramın nəticəsi bizə aydın olsa da kənardan baxan istifadəçiyə aydın təsir bağışlamaz. Bunun üçün proqramımızı istifadəçilər üçün bir qədər izahlı etməliyik. x –in qiymətini çap eləmədən öncə həmin qiymətin x dəyişəninə məxsus olduğunu bildirən sətir də çap edək. Bu zaman proqramın nəticəsi kənardan baxan üçün daha aydın görünər.

```
#include <iostream>

using namespace std;

int main (){

    int x;

    x = 5;

    cout << "x -in qyimeti = ";
    cout << x;

}
```



Əvvəlki dərsimizdə birləşdirmə operatoru ilə tanış olduq. Buna görə eyni nəticəni aşağıdakı kod ilə də ala bilərik:

```
#include <iostream>

using namespace std;

int main (){

    int x;

    x = 5;

    cout << "x -in qyimeti = " << x;

}
```

Dəyişənin qiymətinin çap olunması

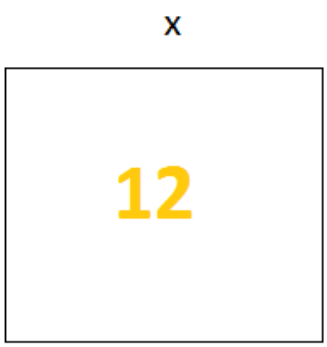
Sərbəst firikləşmək üçün sual:

Bəs x –i cütdırnaq arasında yazsaq nəticə necə olar?

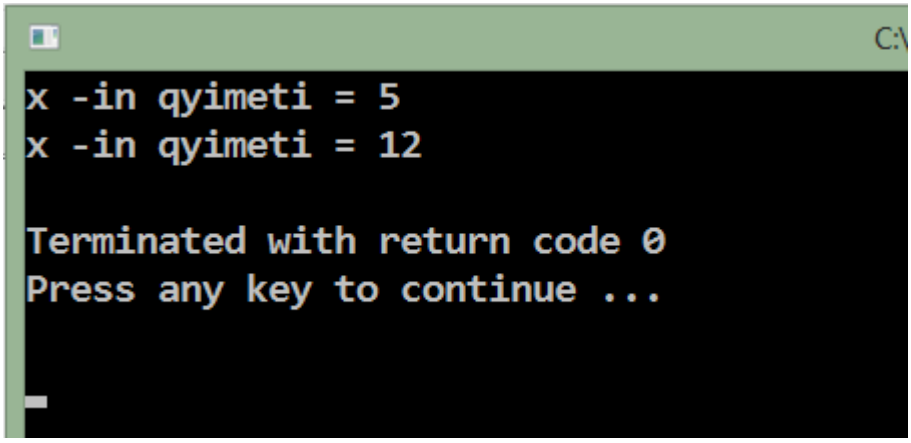
```
cout << "x -in qyimeti = " << "x";
```

Dəyişənə hər-hansı qiymət mənimsətdikdən sonra yenidən başqa bir qiymət mənimsətsək nə baş verər? Artıq yuxarıda qeyd etdiyimiz kimi dəyişənə yeni qiymət mənimsətdiyimiz zaman əvvəlki qiymət(hər nə olursa olsun fərqi yoxdur) silinər və yerinə yeni mənimsətdiyimiz qiymət yazılar. Misal üçün **x** dəyişəninə 5 qiyməti mənimsətdikdən sonra ayrı bir mənimsətmə operatoru ilə 12 qiyməti mənimsətsək, yaddaşdakı 5 qiyməti silinəcək və yerinə 12 yazılacaq. Artıq bundan sonra dəyişənin əvvəlki qiymətini bərpa etmək imkanımız olmur, yəni dəyişənə yeni qiymət mənimsətdiyimiz zaman köhnə qiyməti birdəfəlik itirmiş oluruq.

```
x = 12;
```



```cpp
#include <iostream>

using namespace std;

int main (){

    int x;

    x = 5;

    cout << "x -in qyimeti = " << x;

    x = 12;

    cout << "\n";

    cout << "x -in qyimeti = " << x;

}
```



Nəticələri alt-alta çap eləmək üçün arada yeni sətir simvolunu çap etməyimizə diqqət yetirin.

Yaxşı bəs necə etmək olar ki dəyişənə yeni qiymət mənimsədək, lain əvvəlki qiyməti itirməyək. Bilirəm yuxarıdakı izahdan sonra bu sual bir qədər ziddiyyətli səslənə bilər amma narahatçılığa heç bir əsas yoxdur. Bir çox hallarda bizə bu lazım olur. Dəyişənə yeni qiymət mənimsədən zaman onun əvvəlki qiymətini də yadda saxlamaq tələb olunur. Belə olan hallarda növbəti paraqrafda da görəcəyimiz kimi dəyişənə yeni

qiymət mənimsədərkən köhnə qiymətin itirməmək üçün onu başqa bir dəyişəndə yadda saxlayırlar. Baxaq görək bunu necə eliyirlər.

3.7 İki dəyişənin qiymətlərinin bir-biri ilə dəyişdirilməsi

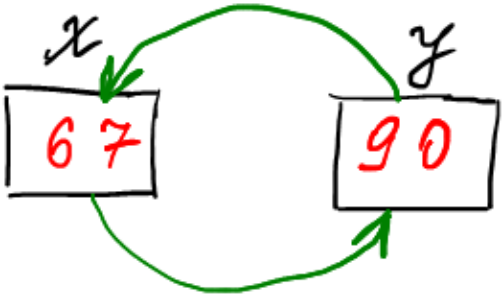
Dəyişənlərin qiymətlərinin dəyişdirilməsi gələcəkdə bizə tez-tez lazım olacaq, əsasən də 6-cı paraqrafda cərgələri keçəndə Tutaq ki, bizə `int` tipli x və y dəyişənləri verilmişdir.

```
int x, y;
```

Bu dəyişənlərə müxtəlif qiymətlər mənimsədək:

```
x = 67;
y = 90;
```

İndi isə bizdən tələb olunur ki, bu iki dəyişənin qiymətlərini bir-biri ilə əvəzləyək.



Yəni x-in qiyməti olsun 90, y-ki isə 67, aşağıdakı kimi:

Biz dəyişənlərin qiymətlərini əvəzləmək üçün aşağıdakı kimi mənimsətmə operatorları işə yaramaz.

```
x = y;
y = x;
```

Çünki mənimsətmə operatoru dəyişənin əvvəlki qiymətini **silir**, yerinə yenisini yazır. Misal üçün baxdığımız birinci mənimsətmə zamanı `x = y;` x-in qiyməti yəni 67 silinər və yerinə 90 yazılar. Nəticədə hər iki dəyişənin qiyməti 90-a bərabər olar. Bu halda biz 67 qiymətini itirmiş oluruq.

Daha sonra yazdığımız ikinci mənimsətmə isə `y = x;` heç nəyi dəyişməz.

Bu problemi həll etmək üçün əlavə 3-cü dəyişənə ehtiyacımız var. Əlavə dəyişəndən x-in başlanğıc qiymətini yadda saxlamaq üçün isitfadə edəcəyik. Ədəbiyyadta bu dəyişəni çox vaxt müvəqqəti dəyişən də adlandırırlar. Tutaq ki, x və y-dən əlavə int tipli hər-hansı z dəyişəni də elan etmişik:

```
int z;
```

Əvvəlcə x-in qiymətin z-tə yazaq:

```
z = x;
```

x –in qiyməti z-tə köçürülər. Təbii ki z-in öz əvvəlki qiyməti silinəcək, lakin bu bizi elə də narahat etmir.

Artıq x-in qiymətini z-də yadda saxladığımıza görə y-in qiymətini x-ə asanlıqla köçürə bilərik.

```
x = y;
```

x-in əvvəlki qiymətini z-də yadda saxladığımıza görə y-ə də onu mənimsədirik:

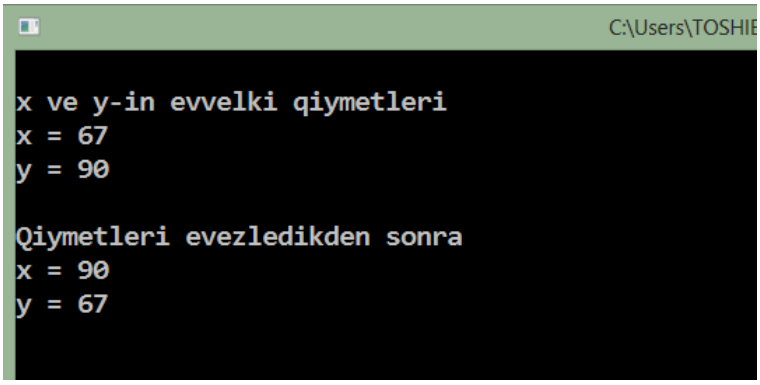
```
y = z;
```

Beləliklə x və y dəyişənlərinin qiymətlərini bir-biri ilə əvəzləmiş olduq. İki dəyişənin qiymətini əvəzləyən proqram kodunun tam mətni aşağıdakı kimi olar.

Nümunə

```cpp
#include <iostream>

using namespace std;

int main (){

    int x, y ,z;

    x = 67;
    y = 90;


    //x-ve y-in qiymetlerini cap edek
    cout << "\nx ve y-in evvelki qiymetleri\n";
    cout << "x = " << x << "\n";
    cout << "y = " << y << "\n";

    //x ve y-in qiymetlerini evezleyek
    z = x;
    x = y;
    y = z;

    //x -ve y-in yeni qiymetlerini cap edek
    cout << "\nQiymetleri evezledikden sonra\n";
    cout << "x = " << x << "\n";
    cout << "y = " << y << "\n";

}
```

Nəticə



3.8 İstifadəçi ilə əlaqə - **cin** operatoru.

Əvvəlki paraqraflarda biz öz proqramımızdan məlumatı ekranda çap eləməyi örgəndik . Eləcə də dəyişən elan etmək və ona mənimsətmə operatoru ilə qiymət yerləşdirməyi və daha sonra bu qiyməti ekranda çap etməyi örgəndik. Bütün bunlar çox yaxşıdır, lakin

əgər diqqət yetirirsinizsə indiyə kimi icra etdiyimiz bütün proqramlarda məlumat yalnız bir istiqamətdə - proqramdan istifadəçiyə doğru hərəkət edirdi. İndi isə biz əksinə məlumatın istifadəçidən nə cür alınaraq bizim proqrama daxil edilməsini örgənəcəyik.

Məlumatı istifadəçidən qəbul etmək üçün **cin** operatorundan istifadə olunur. (Oxunur - sin)

`cin` operatorunun istifadə qaydası aşağıdakı kimidir:

```
cin >> dəyişən ;
```

cin operatoru icra olunan zaman proqram istifadəçi tərəfindən məlumatın daxil edilməsini gözləyir. İstifadəçi məlumatı klaviaturadan daxil edib enter düyməsini basdıqdan sonra həmin məlumat **cin** operatoru ilə göstərilən dəyişənə yerləşdiriləcək. Misal üçün tam tipli x dəyişəninə istifadəçinin daxil etdiyi qyməti mənimsədən kod aşağıdakı kimi olar:

```
cin >> x ;
```

Bu zaman **`cin`** operatoru istifadəçinin klaviaturadan daxil etdiyi ədədi oxuyaraq göstərilən x **`dəyişənin`**də yadda saxlayır. Tam kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int   x ;

    cin >> x ;

}
```

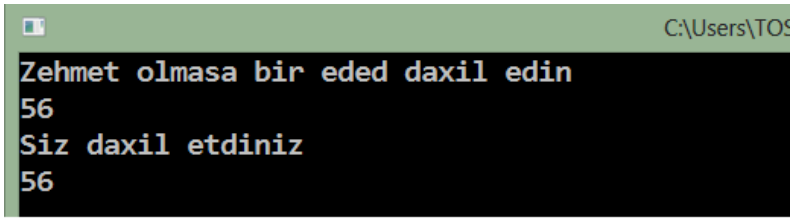
Əgər bu proqramı icra eləsək onda digər ekranda nəsə məlumat v.s. çap edən proqramlar kimi dərhal icra olub başa çatmayacaq. Bu proqram əvvəlkilərdən fərqli olaraq *gözləyəcək*. İstifadəçinin məlumatı daxil etməsini gözləyəcək.

Yalnız istifadəçi məlumatı yığıb enter düyməsi ilə təsdiqlədikdən sonra proqram icraolunmanı davam etdirəcək. İstifadəçi qiyməti daxil edib enter düyməsini basdıqdan sonra həmin qiymət x dəyişəninə yazılacaq.

Beləliklə biz məlumatı, baxdığımız halda tam ədədi istifadəçidən qəbul edib dəyişəndə yadda saxlamış olduq. Məlumatı istifadəçidən qəbul etdikdən sonra onun üzərində müxtəlif əməliyyatlar çap edə bilərik, misal üçün çap edə bilərik. Gəlin aşağıdakı kimi sadə bir proqram quraq. Əvvəlcə proqramımız istifadəçidən hər-hansı ədəd daxil etməsini istəsin. Daha sonra istifadəçinin daxil etdiyi ədədi proqramda elan etdiyimiz bir dəyişənə yerləşdirək. Daha sonra istifadəçinin daxil etdiyi və dəyişənə yerləşdirdiyimiz ədədi təkrar ekranda çap edək. Bu proqramı sadəcə istifadəçi ilə əlaqəni test etməyə aid bir çalışma kimi dəyərləndirək.

```
#include <iostream>

using namespace std;


int main (){

    int   x ;

    cout << "Zehmet olmasa bir eded daxil edin\n";

    cin >> x ;

    cout << "Siz daxil etdiniz\n";

    cout << x ;

}
```

Nəticə



Bu zaman proqram icra olanda istifadəçinin klaviaturadan hər-hansı ədəd daxil etməsini gözləyəcək, istifadəçi hər-hansı ədədi daxil edib(baxdığımız halda 56 ədədini) enter düyməsinə basdıqdan sonra proqram həmin ədədi qəbul edib **x** dəyişənində yadda saxlayacaq. Daha sonra biz istifadəçinin daxil elədiyi ədədi cout operatoru vastəsilə təkrar ekranda çap elədik.

Proqram kodunu sətir-sətir təhlil edək.

Əvvəlcə int tipli x dəyişəni elan edirik:

```
int   x ;
```

Növbəti sətir:

```
cout << "Zehmet olmasa bir eded daxil edin\n";
```

Bu sətirdə biz ekranda `"Zehmet olmasa bir eded daxil edin\n"` sətrini çap edirik. Bu sətri istifadəçiyə hər-hansı ədəd daxil etməli olduğunu bildirmək üçün çap edirik. Yəni belə deyək: qara ekran ilə istifadəçi bir-birinin üzünə baxmasınlar.

Növbəti sətir:

```
cin >> x ;
```

Bu zaman proqram istifadəçidən hər-hansı ədəd daxil etməsini gözləyəcək və istifadəçi hər-hansı ədəd daxil etdikdə həmin ədədi **x** dəyişəninə yazacaq. Yuxarıdakı nümunə icrada istifadəçi klaviaturadan **56** ədədini daxil edir.

Növbəti sətir:

```
cout << "Siz daxil etdiniz\n";
```

Bu sətir də istifadəçiyə məlumat vermək üçün çap edirik.

Növbəti sətir:

```
cout << x ;
```

Bu zaman **x** dəyişəninin qiyməti , yəni 56 ekranda çap olunur. Qeyd edim ki, sonuncu iki əməliyyatda `"Siz daxil etdiniz\n"` sətrini və **x** dəyişənini çap etmək üçün ayrıca **cout** operatorundan istifadə elədik. Lakin keçən bölmədə tanış olduğumuz **axına əlavə et - <<** operatorunun köməyi ilə bir **cout** operatoru ilə aşağıdakı kimi yaza bilərdik:

```
cout << "Siz daxil etdiniz\n" << x ;
```

və ya

```
    cout << "Siz daxil etdiniz\n"
         << x ;
```

kimi.

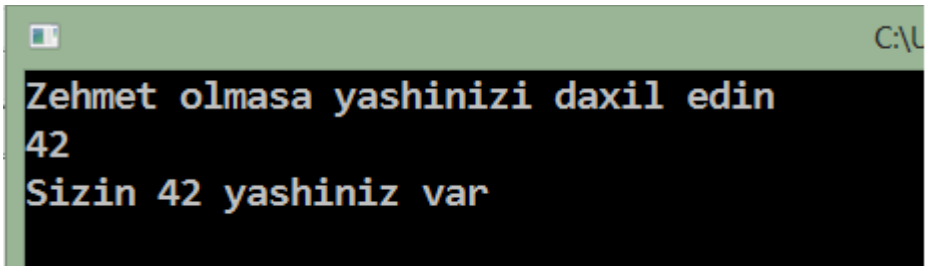
Artıq istifadəçi ilə əlaqəni təmin etdikdən sonra daha da maraqlı proqramlar qurmağa davam edək.

Çalışma. İstifadəçidən yaşını daxil etməsini istəyən proqram qurun. Proqram istifadəçinin yaşını təkrar ekranda çap etməlidir.

Həlli. Bu elə də çətin bir proqram deyil. Yuxarıda tərtib etdiyimiz proqramın biraz fərqli variantıdır. Həll aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;


int main (){

    int   x ;

    cout << "Zehmet olmasa yashinizi daxil edin\n";

    cin >> x ;

    cout << "Sizin " << x << " yashiniz var\n";

}
```

Nəticə:

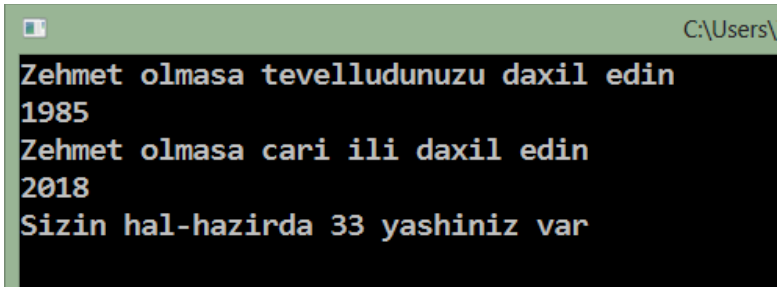


Çalışma. İstifadəçidən təvəllüdünü və hazırki ili daxil etməsini istəyən proqram tərtib edin. Daha sonra proqram istifadəçinin yaşını hesablayıb ekranda çap etməlidir.

Həlli. Bu çalışmanın həlli zamanı biz həm cin həm də mənimsətmə operatorundan istifadə etməliyik. Ümumiyyətlə bu bir qədər maraqlı çalışmadır. Əvvəlcə kodu daxil edək və nümunə nəticə ilə tanış olar, daha sonra izah ilə tanış olarıq.

```
#include <iostream>

using namespace std;


int main (){

    int   tev, il, y ;

    cout << "Zehmet olmasa tevelludunuzu daxil edin\n";

    cin >> tev ;

    cout << "Zehmet olmasa cari ili daxil edin\n";

    cin >> il ;

    y = il - tev;

    cout << "Sizin hal-hazirda " << y << " yashiniz var\n";

}
```

Nəticə



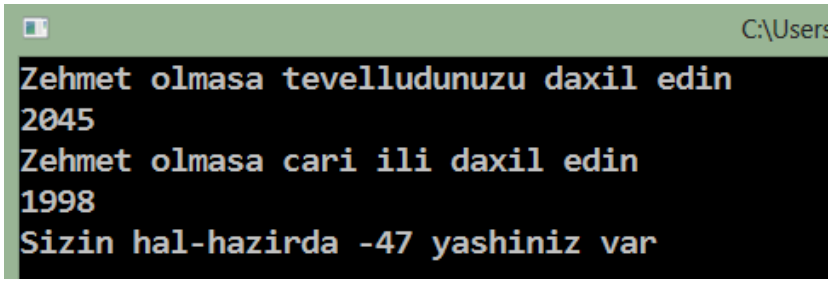
Deməli proqramda tevelludu, cari ili və yaşı yadda saxlamaq üçün int tipli tev, il və y adlı 3 dəyişən elan etdik. cin operatoru vastəsilə tev və il dəyişənlərinəd istifadəçinin daxil etdiyi müvafiq olaraq istifadəçinin təvəllüdünü və cari ili yazdıq. Daha sonra istifadəçinin yaşını hesablamaq üçün y dəyişəninə il ilə tev dəyişənlərinin fərqini mənimsətdik, aşağıdakı kimi:

```
y = il - tev;
```

Qeyd edək ki baxdığımız nümunə nəticədə istifadəçi təvüllüd olaraq – yəni doğulduğu il olaraq 1985, cari – yəni hal-hazırki il olaraq isə 2018 daxil etmişdir və bu ədədlər cin operatoru vastəsilə istifadəçidən oxunaraq müvafiq olaraq tev və il dəyişənlərinə yazılmışdır.

Bundan sonra yuxarıdakı mənimsətmə operatoru ilə bu iki dəyişənin fərqi y dəyişəninə mənimsədilən zaman əvvəlcə onların fərqi hesablanır (`il - tev`) və alınan nəticə (baxdığımız halda 33) y dəyişəninə yazılır.

Gördüyümüz kimi cout, cin və mənimsətmə operatorlarının köməyi ilə artıq nisbətən maraqlı kodlar tərtib edə bilirik. Növbəti çalışmaya keçməzdən öncə bu çalıma ilə bağlı başqa bir məqama toxunmaq istəyirəm. Proqramı bir daha icra edək və başqa qiymətlərlə nəticəni yoxlayaq



Gördüyümüz kimi burada mən bilərəkdən sınaq məqsədilə doğum tarixini hazırki ildən böyük daxil etdim və proqram hər şey qaydasındaymış kimi mənim mənfi 47 yaşım olduğunu çap etdi. Praktikada da istifadəçilər proqrama bu cür səhv məlumatlar daxil edə bilərlər. Lakin proqram istifadəçinin hər-nə daxil etdiyi məlumatı yoxlamamış nəticə hesablamağı yaxşı hal deyil. İrəlidəki paraqraflarda biz if operatorunun köməyi ilə proqramın nə cür yoxlama aparmasını və bu yoxlamanın nəticəsində müvafiq qərar qəbul etməsini örgənəcəyik.

3.9 Hesablama operatorları

C++ dilində müxtəlif hesablama operatorları təyin edilib. Onlardan bəziləri ilə tanış olaq. Qeyd edim ki bu operatorlar ilə ekranda çap bölməsində tanış olmuşduq. İndi isə dəyişənlərdən istifadə edərək onları biraz daha ətraflı örgənəcəyik.

+	Üstəgəl
-	Çıxma
*	Vurma
/	Bölmə
%	Qalıq

Bu operatorlar bizə dəyişənlər üzərində müvafiq əməliyyatları aparmağa imkan verir. Gəlin çalışmalar həlli ilə bu operatorların istifadəyə aid proqram nümunələri ilə tanış olaq.

Çalışma. 5 ilə 9 ədədinin cəmini hesablayan proqram tərtib edin.

Həlli. Nümunə kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  int   x;

  x = 5 + 9;

  cout << " 5 ile 9-un cemi =  " << x << "\n";

}
```

İzahı. Kod kifayət qədər sadədir. Bir məsələni qeyd eim ki, 5 ilə 9 –un cəmini hesablamaq üçün **cout** operatorundan da istifadə edə bilərdik,

```
cout << " 5 ile 9-un cemi =  " << 5 + 9 << "\n";
```

kimi. Lakin biz əlavə x dəyişəni elan etdik. Məqsədimiz hesab əməllərində dəyişənlərdən istifadəyə aid nümunə göstərməkdir. Çıxma və vurma operatorları da toplama operatoruna oxşardır. Lakin bölmə və qalıq operatorları bir qədər fərqlənir. Gəlin onlarla tanış olaq.

3.9.1 Bölmə operatoru

Bölmə operatoru - `'/'` kimi işarə olunur və iki ədədin birini digərinə bölür. Lakin dəyişənlərin tipindən asılı olaraq bölmə nəticəsində alınan qalıq nəzərə alınmaya bilər. Misal üçün əgər böldüyümüz ədədlər və nəticəni mənimsətdiyimz dəyişəninin tipi tamdırsa onda bölmə operatoru ancaq tam hissəsni hesablayacaq, qalığı inkar edəcək. Bu halda qalığı ayrıca **qalıq** - `'%'` operatoru ilə hesablamalıyıq. Yox əgər bölmədə iştirak edən ədədlər və nəticəni mənimsətdiyimiz dəyişən kəsr tiplidirsə onda bölmə operatoru nəticəni tam və onluq qalıq şəklində alacaq. Əvvəlcə tam hala aid nümunələrə baxaq.

Çalışma: 137 ədədinin 23 ədədinə bölünməsindən alınan tam hissəni hesablayın.

Həlli. Tam hissə tələb olunduğuna görə tam tipli dəyişən elan etməli və bölmə operatorundan istifadə etməklə 137 –nin 23-ə bölünməsindən alınan tam hissəni elan etdiyimiz dəyişənə mənimsətməliyik. Kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

  // tam tipli x deyisheni elan edek
  int   x;

  // 137 bolek 23 -un tam hissesini x-e minmsedek
  x = 137 / 23;

  // neticeni ekrana verek
  cout << " 137 bolek 23 -un tam hissesi =  "
       << x << "\n";

}
```

Nəticə belə olar:

```
137 bolek 23 -un tam hissesi =  5
```

İndi isə iki tam ədədin birinin digərinə bölünməsindən alınan qalığın nə cür hesablandığını örgənək.

3.9.2 Qalıq operatoru

Qalıq operatoru - `'%'` kimi işarə olunur və iki tam ədədin birinin digərinə böldükdə yerdə qalan qalığı müəyyən edir. Qalıq operatounun operandları və mənimsədildiyi nəticə tam tipli olmalıdı. Nümunə koda baxaq:

Çalışma: 137 ədədinin 23 ədədinə bölünməsindən alınan qalıq hissəni hesablayın.

Həlli. Qalığın hesablanması da tam hissənin hesablanmasına analojidir. Kod belə olar.

```
#include <iostream>

using namespace std;

int main (){
```

```
    // tam tipli x deyisheni elan edek
    int   x;

    // 137-ni 23 -e boldukde qalan qaliqi x-e minmsedek
    x = 137 % 23;

    // neticeni ekrana verek
    cout << " 137 bolek 23 -un qaliq hissesi =  "
         << x << "\n";

}
```

Nəticə belə olar:

```
 137 bolek 23 -un qaliq hissesi =  22
```

Çalışma. 593 ədədinin 38 ədədinə bölünməsindən alınan tam və qalıq hissəni hesablayan proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

  // tam tipli x ve y deyishenleri elan edek
  int   x,y;

  // 593-u 38 -e boldukde alinan tam hisseni x-e minmsedek
  x = 593 / 38;

  // 593-u 38 -e boldukde alinan qaliq hisseni y-e minmsedek
  y = 593 % 38;

  // neticeni ekrana verek
  cout << " 593 / 38  =  " << x << "\n"
       << " 593 % 38  =  " << y << "\n";

}
```

Kodu araşdırmaq oxuyucuya həvalə olunur. İcra nəticəsi belə olar.

```
593 / 38  =  15
593 % 38  =  23
```

Hesab edirəm ki, baxdığımız proqram nümunələri tam ədədlər üzərində aparılan bölmə və qalıq əməliyyatlarını və onların nəticələrini başa düşməyə kifayət edər. İndi isə kəsr ədədlər üzərində bölmə əməliyyatının aparılmasına aid proqramlara baxaq.

3.10 Kəsr tipi

C++ dilində kəsr tipi `double` açar sözü ilə işarə olunur. Bu tip kəsr ədədlərlə işləmək üçün istifadə olunur. Kəsr ədəd tam və qalıq hissədən ibarət olur. Tam və qalıq bir-birindən nöqtə ilə ayrılır, misal üçün `7.6`, `45.123` v.s.

Çalışma. Kəsr tipli **q** və **r** dəyişənləri elan edən və onlara müvafiq olaraq `21.9234` və `12.008` qiymətlərini mənimsədin.

Həlli. Kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

  double  q, r;

  q = 21.9234;
  r = 12.008;

  cout << "q = " << q << "\n"
       << "r = " << r << "\n";

}
```

Nəticə belə olar:

```
q = 21.9234
r = 12.008
```

3.10.1 Kəsr ədədlərlə bölmə

Biz dedik ki, bölmə operatoru qalıq hissəsində bölmədə iştirak edən və nəticənin mənimsədildiyi dəyişənin tipindən asılı olaraq tam və kəsr tiplər üçün fərqli nəticə verir. Tam tip ilə yuxarıda tanış olduq. İndi isə kəsr tipi ilə tanış olaq. Əgər bölmədə

iştirak edən operandlar kəsr tiplidirsə onda bölmə operatoru tam və qalıq hissəni kəsr şəklində qaytarır. Yəni tam hissə olduğu kimi, kəsr hissə isə onluq şəkildə tam hissədən nöqtə ilə ayrılaraq. Nümunə koda baxaq.

Çalışma. 137 ədədinin 23-ə bölünməsinin nəticəsini onluq kəsr şəklində hesablayan proqram tərtib edin.

Həlli. Nəticə kəsr şəklində tələb olunduğuna görə (tam və onluq qalıq nöqtə ilə ayrılmaqla) kəsr tipli dəyişən elan edib, bölmə nəticəsin bu dəyişənə mənimsətməliyik. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  // kesr tipli q deyisheni elan edek
  double  q;

  // 137 / 23 -u q-ye menimsedek
  q = 137.0 / 23;

  // neticeni ekrana gonderek
  cout << "137 / 23 = " << q << "\n";

}
```

Kodu icra etsək nəticə aşağıdakı kimi olar:

```
137 / 23 = 5.95652
```

Əgər koda diqqət yetirsək görərik ki, biz q-yə mənimsətmə zamanı 137 ədədini 137.0 kimi vermişik

```
q = 137.0 / 23;
```

Bunu ona görə etmişik ki, baxmayaraq ki, q dəyişəni **double** tiplidi əgər bölmədə iştirak edən operandların hər biri tam şəkildə olsa, yəni aşağıdakı kimi

```
q = 137 / 23;
```

onda bölmə operatoru qalığı inkar edəcək. Lakin operandlardan biri ya 137, ya da 23 kəsr şəklində olsa onda bölmə operatoru qalığı inkar eləməz. Ya da başqa bir kəsr tipli

dəyişən elan edib 137 –ni və ya 23 –ü ona mənimsətməliyik. Onda artıq bölmə operatoru bilər ki, biz nə istəyirik və bizə onu verər, yəni aşağıdakı kimi:

```
#include <iostream>

using namespace std;

int main (){

  // kesr tipli q ve r deyishenleri elan edek
  double  q, r;

  // kesr tipli r deyishenine 137 menimsedek
  r = 137;

  // q-ye r bolek 23 - u menimsedek
  q = r / 23;

  // neticeni ekrana gonderek
  cout << "137 / 23 = " << q << "\n";

}
```

Nəticə:

```
137 / 23 = 5.95652
```

Ümid eliyirəm ki, az da olsa bu qatmaqarışıq bölmə operatorun başa düşdük. Bunun üzərində baş sındırmağı məsləhət görmürəm. Yavaş-yavaş bölmə operatorunun işini mənimsəyəcəksiniz, biz isə nümunə proqramlarımızı davam edək.

3.10.2 Kəsr hissədən tam hissənin alınması

Əgər biz kəsr tipli dəyişəni tam tipli dəyişənə mənimsətsək onda qalıq inkar olunar və tam hissə tam tipli dəyişənə mənimsənər. Aşağıdakı çalışmada bu qaydadan istifadə olunur.

Çalışma. İstifadəçinin klaviaturadan daxil etdiyi kəsr ədədin tam hissəsini çap edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;
```

```
int main (){

  // kesr tipli q deyisheni elan edek
  double  q;

  // tam tipli k deyisheni elan edek
  int k;

  cout << "Her-hansi onluq kesri olan eded daxil edin \n";
  cin >> q;

  //q-nun tam hissesini k-ya menimsedek
  k = q;

  // neticeni cap edek
  cout << q  << " - nun tam hissesi = "
       << k  << "\n";

}
```

Nümunə nəticə:

```
Her-hansi onluq kesri olan eded daxil edin
34.8517
34.8517 - nun tam hissesi = 34
```

Çalışma. İstifadəçinin klaviaturadan daxil etdiyi kəsr ədədin tam hissəsini və onluq kəsr hissəsini ayrı-ayrılıqda çap edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  // kesr tipli q ve r deyishenleri elan edek
  double  q,r;

  // tam tipli k deyisheni elan edek
  int k;

  cout << "Her-hansi onluq kesri olan eded daxil edin \n";
  cin >> q;

  //q-nun tam hissesini k-ya menimsedek
  k = q;
```

```
    // onluq qaliq hisseni r-e menimsedek
    r = q - k;

    // neticeni cap edek
    cout << q  << " - nun tam hissesi = "
         << k  << "\n"
         << "qaliq hissesi = " << r << "\n";

}
```

Nümunə nəticə:

```
Her-hansi onluq kesri olan eded daxil edin
128.957
128.957 - nun tam hissesi = 128
qaliq hissesi = 0.957
```

İzahı. Yuxarıdakı kodda biz kesr tipli q və r dəyişənləri, tam tipli k dəyişəni elan etdik. əvvəlcə k-ya q-nün tam hissəsini mənimsətdik. Qalıq hissəsni almaq üçün isə q və k – nın fərqin r-ə mənimsətdik.

3.11 Mərtəbə vahidləri

Alqoritmik biliklərimizi, eyni zamanda proqramlaşdırma bacarıqlarımızı inkişaf etdirmək, örgəndiyimiz operatorlarının praktik məsələlərin həllində nə cür tətbiq olunması ilə tanış olmaq məqsədilə çalışmalarımızı davam etdiririk. İndi tanış olacağımız mövzu ədədin mərtəbə vahidləri ilə bağlıdır.

Əvvəlcə mərtəbə vahidləri barədə qısa izah verək. Bu barədə yetərli biliyi olanlar birbaşa çalışmalar hissəsinə keçə bilər.

Nədir ədədin mərtəbə vahidi?

İstənilən bir ədədi 1, 10, 100, 1000 v.s. ədədlərinin 0-dan 9-a kimi ədədlərlə hasillərinin cəmi şəklində yazmaq olar. Misal üçün 327 ədədini 3*100 + 2*10 + 7*1 kmi göstərə bilərik. Bu bütün ədədlərə aiddir. Burada verilmiş ədədi əmələ gətirmək üçün istifadə etdiyimiz 1, 10, 100 mərtəbələr adlanır və müfaviq olaraq təklik, onluq, yüzlük v.s. adlanır. Həmin bu mərtəbələrin vurulduğu ədədlər isə mərtəbə vahidləri adlanır, yəni bu ədəddə 7 dənə təklik, 2 dənə onluq və 3 dənə yüzlük mövcuddur.

Çalışma. Mərtəbə vahidləri 4 minlik, 9 yüzlük, 5 onluq və 3 təklikdən ibarət olan ədədi tapın.

Həlli: Ədədi tapmaq üçün mərtəbə vahidlərini göstərən rəqəmləri müvafiq mərtəbə vahidlərinə vurub üst-üstə gəlməliyik, aşağıdakı kimi:

4*1000 + 9*100 + 5*10 + 3*1 = 4953

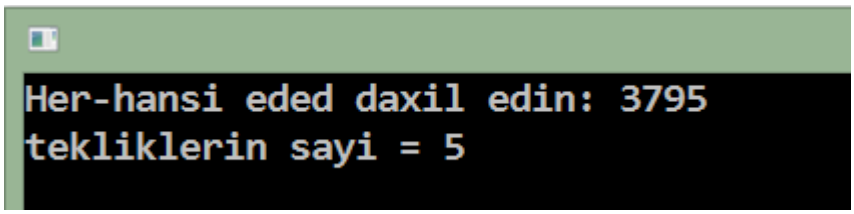
Cavab: 4953.

Hesab eliyirəm ki mərtəbə vahidləri barəsində tanışlığımız kifayət etdi, indi keçək çalışma həllinə.

Çalışma . Verilmiş ədədin təkliklərinin sayını müəyyən edən proqram tərtib edin.

Həlli. Bu çalışmada bizdən hər-hansı ədəd verildikdə onun təkliklərinin sayını tapmağımız tələb olunur. Misal üçün əgər 365 verilibsə, təkliklərin sayı 5-dir. Bunun üçün sadəcə ədədin 10-a bölünməsindən alınan qalığı hesablamağımız kifayətdir. Həqiqətən də 365 ədədini 10-a bölsək qismət 36 edər, 5 isə qalıq qalar. Bu məhs bizim axtardığımız təkliklərin sayını bərabərdir. Bu metodun ədədin təkliklərinin sayını hesabladığına əmin olmaq üçün müəyyən riyazi hesablamalar aparmaq oxucuya tövsiyyə olunur. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int i,x;

    cout << "Her-hansi eded daxil edin: ";
    cin >> x;

    cout << "tekliklerin sayi = "
         << x%10;

}
```



Çalışma . Verilmiş ədədin onluqlarının sayını müəyyən edən proqram tərtib edin.

Həlli. Onluqların sayını tapmaq təklikləri tapmaq qədər asan olmayacaq. Bunun üçün aşağıdakı kimi hərəkət etməliyik:

Əvvəlcə ədədi 10-a bölərək alınan qismәti hesablamalıyıq.

Daha sonra aldığımız qismətin 10-a bölünməsindən alınan qalığı tapmalıyıq.

Sonuncu nəticə bizə verilmiş ədəddə olan onluqlrın sayını gösttərəcək. Misal üçün tutaq ki bizə 259374 ədədi verilib. Əvvəlcə bu ədədin 10-a bölünməsindən alınan qismәti hesablayaq:

259374 / 10 = 25937

4 qalıq qalır hansı ki verilmiş ədədin təkliklərinin sayını göstərir, lakin bu çalışmada təkliklərin sayı tələb olunmadığına görə bu qalığa əhəmiyyət vermirik.

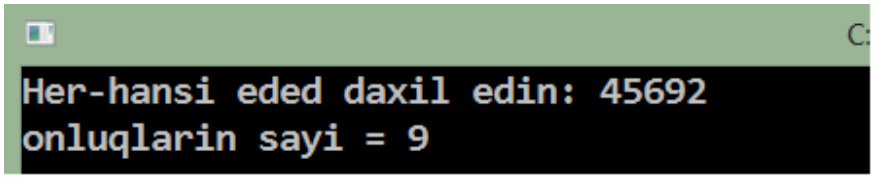
Daha sonra aldığımız qismәti 10-a bölürük lakin bu dəfə alınan qalığı götürürük:

25937 % 10 = 7

Deməli 259374 ədədinin onluq mərtəbə vahidlərinin sayı 7-dir. Bu metodun ədədin təkliklərinin sayını hesabladığına əmin olmaq üçün müəyyən riyazi hesablamalar aparmaq oxucuya tövsiyyə olunur.

Proqram kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int i, x, y;

    cout << "Her-hansi eded daxil edin: ";
    cin >> x;

    y = x / 10;

    cout << "onluqlarin sayi = "
         << y%10;

}
```



3.12 Simvol tipi

Simvol tipi **char** açar sözü ilə ifadə olunur və simvol tipli məlumatlarla işləmək üçündür. C++ dilində simvollar təkdırnaq arasında verilir, misal üçün

'A' – böyük a simvolu, 'b' - kiçik b simvolu, '+' – üstəgəl simvolu, '*' - ulduz simvolu, '1' - bir rəqəmi simvolu, '0' - sıfır rəqəmi simvolu , '\n' - yeni sətir simvolu v.s.

Əgər simvolları təkdırnaq işarəsi arasına almasaq onda həmin simvollar müvafiq mənalarına uyğun qəbul ediləcəklər, yəni simvol deyil, dəyişən, ədəd, operator v.s. kimi. Misal üçün:

'A' – böyük a simvolu, **A** dəyişəni

'+' – üstəgəl simvolu, **+** operatoru

'1'- bir rəqəmi simvolu, **1** ədədi v.s.

Əgər təkdırnaq arasında simvolların sayı birdən çox olarsa onda kompilyator ilk simvoldan sonra gələn simvolları inkar edəcək və xəbərdarlıq mesajı bildirəcək.

Simvol tipinə aid nümunə proqramlarla tanış olaq.

Çalışma. Simvol tipli bir neçə dəyişən elan edin və onları çap edin.

Həlli. Nümunə kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  char  s, t;

  s = 'A';
  t = 'z';

  cout << "s deyisheninin qiymeti = " << s << "\n"
       << "t deyisheninin qiymeti = " << t << "\n";

}
```

Nəticə:

```
s deyisheninin qiymeti = A
t deyisheninin qiymeti = z
```

3.12.1 Simvol və Tam tip

C++ dilində simvollar tam tipin bir forması hesab olunur. Yəni tam tipli dəyişənlər üzərində apardığımız bütün hesablama əməliyyatların simvol tipləri üzərində də apara bilərik. Ola bilər ki, proqramlaşdırma ilə tanış olmayan oxucuya `'A'` simvolu ilə `'e'` simvollarını cəmləmək kimi ifadələr qəribə gəlsin, amma proqramlaşdırmada və xüsusilə C++ dilində bu tam normaldır. Çünki əvvəldə də qeyd etdiyimiz kimi kompüterin yaddaşında hər şey ədədlərlə ifadə olunur. O cümlədən hər bir simvol da kompüterin yaddaşında bir ədəd kimi saxlanılır. Sadəcə çap edərkən və ya digər əməliyyatlar zamanı kompilyator simvol tipli dəyişənlərə bir qədər fərqli yanaşır.

3.12.2 Simvolu ədəd qarşılığı

Biz qeyd elədik ki, simvollar tam tipli ədədlərdirlər və onlardan ədədlər kimi istifadə edə bilərik. Elə isə əvvəlcə maraqlı olar ki, verilmiş simvolun, misal üçün a `'A'` simvolunun ədəd qarşılığı neçədir. Birazdan bunu müəyyən etmək üçün proqram nümunəsi ilə tanış olacayıq. Hələlik isə bu məsələyə biraz da aydınlıq gətirək. Belə ki, simvolların ədəd qarşılığı məsələsi əlbəttdə ki, şərtidir və razılaşma ilə müəyyənləşir. Hətta 7-ci əsrdə islam alimləri 10-luq say sistemini kəşf etməzdən əvvəl abcəd say sistemindən istifadə edirdilər ki, bu sistemdə hər bir ərəb simvoluna bir tam ədəd uyğun gəlirdi. Misal üçün Əlif - ا simvolunun ədəd qarşılığı – 1, nun - ن simvolunun ədəd qarşılığı isə 50 idi.

Müasir dövrümüzdə isə simvolların ədəd qarşılğını təyin edən müxtəlif standartlar, kodlaşdırmalar mövcuddur. Bunlardan ən geniş yayılan ikisi ASCİİ və Unicod simvollarıdır. ASCİİ simvolları ilə əlavə A –da tanış ola bilərsiniz. Misal üçün ASCİİ –də `'A'` simvolunun ədəd qarşılığı 65 – dir.

Gəlin bir neçə simvolun ədəd qarşılığını müəyyən etmənin qaydası ilə tanış olaq.

Çalışma. `'A'`, `'a'`, `'1'`, `'*'` simvollarının ədəd qarşılığını müəyyən edən proqram tərtib edin.

Həlli. Hər hansı simvolun ədəd qarşılığını müəyyən etmək üçün tam tipdən hər-hansı dəyişən elan edib verilmiş simvolu həmin dəyişənə məimsədib qiymətini çap etməliyik. Aşağıdakı kimi:

```
#include <iostream>

using namespace std;

int main (){

  //tam tipli x deyisheni elan edek
  int x;

  // x -e eded qarshilgini orgenmek istediyimiz
  // simvollari menimsedib qiymetlerini cap edek
  x = 'A';
  cout << "A simvolunun eded qarshiliqi = " << x << "\n";

  x = 'a';
  cout << "a simvolunun eded qarshiliqi = " << x << "\n";

  x = '1';
  cout << "1 simvolunun eded qarshiliqi = " << x << "\n";

  x = '*';
  cout << "* simvolunun eded qarshiliqi = " << x << "\n";

}
```

Nəticə:

```
A simvolunun eded qarshiliqi = 65
a simvolunun eded qarshiliqi = 97
1 simvolunun eded qarshiliqi = 49
* simvolunun eded qarshiliqi = 42
```

Bəs tərsi necə alınar, yəni hər-hansı ədədin simvol qarşılığını necə müəyyən edək. Növbəti çalışmada buna aid nümunə verilir.

Çalışma. 75, 80, 34 və 109 ədədlərinin simvol qarşılığını müəyyən edən proqram tərtib edin.

Həlli. Bu dəfə isə tərsinə edəcəyik. Yəni simvol tipindən hər-hansı dəyişən elan edib, onu simvol qarşılığını tapmaq istədiyimiz ədədlərə mənimsədib qiymətini çap edəcəyik. Kod aşağıdakı kimi olar.

```
#include <iostream>
```

```
using namespace std;

int main (){

  //simvol tipli x deyisheni elan edek
  char x;

  // x -e simvol qarshilgini orgenmek istediyimiz
  // ededleri menimsedib qiymetlerini cap edek
  x = 75;
  cout << "75 ededinin simvol qarshiliqi = " << x << "\n";

  x = 80;
  cout << "80 ededinin simvol qarshiliqi = " << x << "\n";

  x = 34;
  cout << "34 ededinin simvol qarshiliqi = " << x << "\n";

  x = 109;
  cout << "109 ededinin simvol qarshiliqi = " << x << "\n";

}
```

Nəticə:

```
75 ededinin simvol qarshiliqi = K
80 ededinin simvol qarshiliqi = P
34 ededinin simvol qarshiliqi = "
109 ededinin simvol qarshiliqi = m
```

3.12.3 Simvollar üzərində hesab əməlləri

Əgər hər bir simvolun ədəd qarşılığı varsa onda onlar üzərində ədədlər üzərində olduğu kimi hesablama apara bilərik. Aşağıdakı nümunəyə baxaq.

Çalışma. `'A'` simvolu ilə `'Q'` simvollarının cəmini və hasilini müəyyən edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

  //tam tipli x deyisheni elan edek
  int x;

  x = 'A' * 'Q';
```

```
    cout << "A * Q = " << x << "\n";

    x = 'A' + 'Q';
    cout << "A + Q = " << x << "\n";

}
```

Nəticə.

```
A * Q = 5265
A + Q = 146
```

Kəsr və simvol tipləri ilə də tanış olduq. İndi bir qədər daha mürəkkəb kod nümunələrini örgənək.

3.13 Sadə cəbri ifadələr.

Çalışma 1. Tam tipli x dəyişəni elan edin. x dəyişəninə 234 , 45 və 4973 ədədlərinin cəmini mənimsədin və onun qiymətini çap edin.

Həlli: Nümunə kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int x;

    x = 234 + 45 + 4973 ;
    cout << "x = " << x << "\n";

}
```

Nəticə:

```
x = 5252
```

Çalışma 2. (728 + 90) – (34*2 – 8*(54 + 65/3)) ifadəsinin qiymətini onluq kəsr şəklində hesablayan proqram tərtib edin.

Həlli: Nəticə onluq kəsr şəklində tələb olunduğundan kəsr tipli dəyişəndən istifadə edəcəyik. Nümunə kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  double x;

  x = (728 + 90)  - (34*2 - 8*(54 + 65/3));

  cout << "(728 + 90)  - (34*2 - 8*(54 + 65/3))  = "
       << x << "\n";

}
```

Nəticə:

```
(728 + 90)  - (34*2 - 8*(54 + 65/3))  = 1350
```

3.14 Sağ tərəfdə dəyişənlərdən istifadə

Mənimsətmə operatorunun sağ tərəfində təkcə ədələr deyil həmçinin dəyişənlərdən də istifadə edə bilərik. Aşağıdakı koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

  // tam tipli 3 deyishen elan edek
  int x,y,z;

  // y ve z deyishenlerine qiymetler
  // menimsedek
  y = 12;
  z = 90;

  // x-e y -ile z-in cemini menimsedek
  x = y + z;
```

```
    cout << y << " + " << z << " = " << x << "\n";

}
```

Nəticə:

```
12 + 90 = 102
```

Çalışma . İstifadəçinin daxil etdiyi iki ədədin hasilini hesablayan proqram tərtib edin.

Həlli: Nümunə kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

  // tam tipli 3 deyishen elan edek
  int x,y,z;

  //y -e qiymet daxil etmesini istifadeciye bildirek
  cout << "Zehmet olmasa birinci ededi daxil edin: \n";

  // y -e cin ile istifadecinin daxil etdiyi
  // birinci ededi menimsedek
  cin >> y;


  //z -te qiymet daxil etmesini istifadeciye bildirek
  cout << "Zehmet olmasa ikinci ededi daxil edin: \n";

  // z -te cin ile istifadecinin daxil etdiyi
  // ikinci ededi menimsedek
  cin >> z;


  //hal-hazirda istifadecinin daxil etdiyi ededler y ve z
  //deyishenlerinde saxlanilir, onlarin cemini x-e menimsedek
  x = y + z;

  //neticeni ekrana gonderek
  cout << "Sizin daxil etdiyiniz "
       << y << " ve " << z << " ededlerinin cemi =  "
       << x << "\n";

}
```

Nümunə nəticə.

```
Zehmet olmasa birinci ededi daxil edin:
45
Zehmet olmasa ikinci ededi daxil edin:
67
Sizin daxil etdiyiniz 45 ve 67 ededlerinin cemi =  112
```

3.15 Dəyişənin öz qiymətindən istifadə etmə

Bəs sağ tərəfdə sol tərəfdəki dəyişənin özündən istifadə etmək olar? Bu zaman konflikt yaranmazki?

Əlbəttdə olar. Misal üçün əgər biz x = x + 9; yazsaq bu zaman əvvəlcə sağ tərəf hesablanacaq və bu hesablama x-in *köhnə* qiyməti ilə aparılacaq, nəticə isə x-in yeni qiymətinə mənimsədiləcək. Bu üsuldan dəyişənin qiymətini müəyyən qədər(dəfə) artırıb(azaltmaq) üçün istifadə olunur. İrəlidəki proqram nümunələrində bu qaydadan istifadə edəcəyik. Aşağıdakı koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

  int x ;

  x = 10;
  cout << "x evvel  =  " << x << "\n";

  // x-in qiymetini 12 vahid artiraq
  x  = x + 12;
  cout << "x sonra  =  " << x << "\n";

}
```

Nəticə:

```
x evvel  =  10
x sonra  =  22
```

Çalışma. Ədədin faizini hesablayan proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  double x,y,z;

  cout<< "Zehmet olmasa ededi daxil edin\n";
  cin>>x;

  cout<< "Zehmet olmasa faizi daxil edin\n";
  cin>>y;

  //x -in y faizini z deyishenine menimsedek
  //riyaziyyatdan bilirik ki, faizi hesabalamaq ucun
  //ededi faiz gosteren reqeme vurub 100-e bolmeliyik
  z = (x*y)/100;

  cout<< x << " -in "<< y << " faizi = " << z <<"\n";

}
```

İcra 1.

```
Zehmet olmasa ededi daxil edin
234
Zehmet olmasa faizi daxil edin
17
234 -in 17 faizi = 39.78
```

Çalışma. Endirim proqramı. Müştərinin xərclərinə 20% endirim edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  double x,y;

  cout<< "Zehmet olmasa meblegi daxil edin\n";
  cin>>x;

  //endirimin meblegin hesablayaq, 20%
  y = (x*20)/100;

  //esab meblegden endirim olunani cixaq
  x -= y;
```

```
    cout<<" Size "<< y << " manat endirim olundu \n"
        <<" Zehmet olmasa " << x << " manat odenish edin \n"
        <<" Teshekkur edirik. \n";


}
```

Icra 1.

```
Zehmet olmasa meblegi daxil edin
97
 Size 19.4 manat endirim olundu
 Zehmet olmasa 77.6 manat odenish edin
 Teshekkur edirik.
```

3.16 Inkrement və Dekrement

C++ dilində dəyişənin qiymətdin 1 vahid artırmaq və ya azaltmaq üçün müvafiq olaraq inkrement və ya dekrement operatorlarından istifadə olunur.

3.16.1 İknrement operatoru

İnkrement operatoru **++** kimi işarə olunur və aid olduğu dəyişənin qiymətini 1 vahid artırır. Misal üçün

```
x++;
```

yazsaq x –in qiyməti 1 vahid artmış olar.

İnkrement işarəsini **++** həm dəyişəndən əvvəl, həm də sonra qoya bilərik.

```
++x;
```

və ya

```
x++;
```

Lakin bu iki formanın bir fərqi var, gəlin onu anlamağa çalışaq. Əgər məqsədimiz yalnız dəyişənin qiymətini bir vahid artırmaqdırsa onda **++** işarəsini əvvəl və ya sonra yazmağımızın heç bir fərqi yoxdu. Nümunə koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

  // tam tipli x deyisheni elan edek
```

```
    int x;

    // x-e 10 qiymeti menimsedek
    x = 10;

    // x-in qiymetin cap edek
    cout << "x =  " << x << "\n";

    // x-in qiymetin 1 vahid artiraq
    x++;

    // x-in qiymetin cap edek
    cout << "x =  " << x << "\n";

    // x-in qiymetini yene 1 vahid artiraq
    ++x;

    // x-in qiymetin cap edek
    cout << "x =  " << x << "\n";

}
```

Nəticə

```
x =  10
x =  11
x =  12
```

Amma əgər dəyişəni hər hansı ifadə daxilində işlədiriksə onda **++** işarəsini əvvəl və ya sonra yazmağımızın fərqi var. Belə ki, **++** işarəsini dəyişəndən əvvəl qoysaq onda əvvəlcə dəyişənin qiyməti artırılır və ifadə daxilində dəyişənin artmış qiyməti hesablanır. Yox əgər işarə dəyişəndən sonra qoyulubsa onda əvvəlcə ifadənin qiyməti dəyişənin ilkin qiyməti ilə hesablanır, sonra dəyişənin qiyməti artırılır. Nümunə koda baxaq.

```
#include <iostream>

using namespace std;

int main (){

    // tam tipli x ve  y deyishenleri elan edek
    int x,y;

    // x-e 10 qiymeti menimsedek
    x = 10;

    // x-in evvelki qiymetin cap edek
```

```
    cout << "x  evvel = " << x <<" ,  " ;

    // ifade daxilinde ++ ishersini deyishenden
    // evvel yazaq
    y = 5  +  ++x;

    // y ve x-in  yeni qiymetlerini cap edek
    cout << "5  +  ++x  =  " << y
              << " , x sonra =  " << x << "\n";

    // x-e yeniden 10 qiymeti menimsedek
    x = 10;

    // x-in qiymetin cap edek
    cout << "x  evvel = " << x <<" ,  ";

    // ifade daxilinde ++ ishersini deyishenden
    // sonra yazaq
    y = 5  +  x++;

    // y ve x-in yeni qiymetlerini cap edek
    cout << "5  +  x++  =  " << y
         << " , x sonra =  " << x << "\n";

}
```

Nəticə.

```
x  evvel = 10 ,  5  +  ++x  =  16 , x sonra =  11
x  evvel = 10 ,  5  +  x++  =  15 , x sonra =  11
```

Şərhlər kifayət qədər ətraflı olduğundan əlavə izaha məncə ehtiyac yoxdur.

3.16.2 Dekrement operatoru

Dekrement operatoru **--** kimi işarə olunur və dəyişənin qiymətin bir vahid azaltmaq üçün istifadə olunur. Misal üçün

```
x--;
```

yazsaq x –in qiyməti 1 vahid azalmış olar. Dekrement operatorunun istifadəsi qaydası inkrement operatoruna analojidir.

3.17 Digər mənimsətmə operatorları

Mənimsətmə operatorunun cəm, fərq, hasil, nisbət, qalıq mənimsətməsi v.s. kimi növləri mövcuddur, aşağıdakı kimi:

+=	cəm mənimsətməsi
-=	fərq mənimsətməsi
*=	hasil mənimsətməsi
/=	nisbət mənimsətməsi
%=	qalıq mənimsətməsi

Bu operatorların hamısının istifadəsi oxşar olduğuna görə yalnız cəm mənimsətməsi operatorunun izahını verəcəyik.

3.17.1 Cəm mənimsətməsi

Cəm mənimsətməsi operatoru **+=** kimi işarə olunur və mənimsətmənin sağında olan ifadənin qiymətini solda olan dəyişənin üzərinə əlavə eliyir. Nümunə koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

  int x;

  x = 10;
  cout << "x evvel " << x << "\n";

  x += 5;
  cout << "x sonra " << x << "\n";

}
```

Nəticə

```
x evvel 10
x sonra 15
```

Çalışmalar

Çalışmaları istədiyiniz ardıcıllıqla yerinə yetirə bilərsiniz. Əsas çalışın ki 14 və 16 –cı çalışmaları mütləq yerinə yetirəsiniz. Əgər bu iki çalışmanı yerinə yetirsəniz onda

mövzunu ortadan yuxarı səviyyədə mənimsəmisiniz. Bu mövzunun ən çətin çalışması olan 17-ci çalışmanı yerinə yetirmək çox uğurlu göstərici hesab olunur.

Çalışma 1: Proqramda tam tipli x dəyişəni elan edin. x dəyişəninə istifadəçinin daxil etdiyi qiyməti mənimsədin və bu qiyməti ekranda çap edin.

Çalışma 2: Çalışma 1 –i kesr və simvol tipli dəyişənlərlə tərtib edin.

Çalışma 3: İstifadəçinin daxil etdiyi iki ədədin hasilini ekranda çap edən proqram tərtib edin.

Çalışma 4: İstifadəçinin daxil etdiyi 5 tam ədədi ekranda əks sıra ilə (istifadəçinin daxil etdiyi sıranın əksinə) çap edən proqram tərtib edin.

Nümunə nəticə:

```
5 eded daxil edin:
12 3 54 67 890
Sizin daxil etdiyiniz ededler eks sira ile
890  54 3 12
```

Çalışma 7: Elə proqram tərtib edin ki, istifadəçidən 10 ədəd daxil etməsini istəsin və onlardan ilk beş ədədin cəmi ilə sonrakı beş ədədin cəmini ayrı-ayrılıqda çap etsin.

Nümunə nəticə:

```
10 eded daxil edin:
12 3 45 8 67 111 90 67 87 3
Ilk 5 ededin cemi
12  + 3 + 45 + 8 + 67 = 135
Son 5 ededin cemi
112 + 90 + 67 + 87 + 3 = 359
```

Çalışma 9: Aşağıdakı simvolların ədəd qarşılığını çap edən proqram tərtib edin.

```
'A', '(', ':', '7'
```

Çalışma 10: Yuxarıdakı çalışmaların həlli üçün tərtib etdiyiniz bütün proqram mətnlərinə öz adınızı şərh olaraq əlavə edin. Proqramın müxtəlif yerlərinə izahedici şərhlər əlavə edin.

Çalışma 12: Düzbucaqlının eni və uzunluğu veriləndə onun perimetrini hesablayan proqram qurun.

P = 2(en + uzunluq)

Çalışma 13: Elə proqram qurun ki, saatı və dəqiqəni daxil edəndə cəmi dəqiqələri hesablasın, misal üçün 1 saat 30 dəq = 90 dəq.

Çalışma 14: Elə proqram qurun ki, dəqiqələr veriləndə cəmi saatlar və qalan dəqiqələri çap etsin, misal üçün (90 dəq = 1 saat 30 dəq)

Çalışma 15: İstifadəçinin daxil etdiyi iki ədədin həndəsi(hasilin saya nisbəti) və ədədi(cəmin saya) ortalarını hesablayan proqram tərtib edin.

Çalışma 16. İstifadəçinin daxil etdiyi üçrəqəmli ədədin təklik, onluq və yüzlüklərinin sayını müəyyən edən proqram tərtib edin. Proqram təklik, onluq və yüzlüklərin sayını ayrı-ayrılıqda çap etməlidir. Misal üçün istifadəçi 492 ədədi daxil etsə proqram aşağıdakı nəticəni çap etməlidir:

4 yuzluk

9 onluq

2 teklik

Çalışma 17: Elə proqram qurun ki istifadəçidən onluq kəsr hissəsi olan ədəd daxil etməsini istəsin(misal üçün 78.435 ədədini). Daha sonra istifadəçi həmin ədədin kəsr hissəsinin ilk rəqəmini (baxdığımız nümunə üçün 4 rəqəmi) çap etsin.

Çalışma 18. İstifadəçidən tevelludun və cari ili daxil etməsini istəyən və 15 ildən sonra istifadəçinin neçə yaşı olacağını hesablayan proqram quun.

§4 Şərt operatoru

4. 1 Şərt operatoru - if.

Qərar qəbul etmək imkanı proqram üçün olduqca vacibdir. İndiyə kimi tərtib etdiyimiz proqramlarda əməliyyatlar ardıcıl icra olunub və bu zaman konkret addımda bizim icra etmək üçün bu və ya digər əməliyyatı seçmək kimi bir imkanımız olmayıb. Şübhəsiz ki belə bir imkanın olması çox vacibdir və bu cür imkan bizim proqramlaşdırma imkanlarımızı sadəcə hesablama/daxiletmə və xaricietmə kimi əməliyyatlarla məhdudlaşmasının qarşısını alaraq, onlara eyni zamanda məntiqi əməliyyatlar aparmağa imkan verər.

Bunun üçün C++ dilində şərt operatorundan istifadə edirlər. Şərt operatoru hər-hansı əməliyyatın mütləq deyil, verilmiş şərtdən asılı olaraq icra olunub-olunmamasına imkan verir.

Şərt operatorunun sitaksisi aşağıdakı kimidir:

```
if (şərt){

  // şərt ödənəndə icra olunan kod

}
else{

  // şərt ödənməyəndə icra olunan kod
}
```

Əvvəlcə if açar sözünü yazırıq, daha sonra mötərizə içərisində şərti ifadəni göstəririk. Bundan sonra fiqurlu mötərizələr arasında həmin şərtin ödənəcəyi halda icra olunmalı operatorları yerləşdiririk. Daha sonra else açar sözünü yazıb fiqurlu mötərizələr qoyuruq. else -ə aid olan fiqurlu mötərizələrə yerləşdirilmiş operatorlar isə şərt ödənmədiyi halda icra olunacaq.

Ümumi halı ifadə edən bu izah bizim üçün ilk başlanğıcda əlbəttə çətin görünə bilər. Ona görə onun incəliklərinə birdəfəyə yox tədriclə varmağa çalışacağıq. Qeyd eliyim ki if operatoru heç də həmişə yuxarıda yazdığımız formada olmağı tələb etmir. Xüsusən

də əgər şərtin ödənmədiyi halı nəzərə almaq tələb olunmazsa onda else hissəsini rahatlıqla ata bilərik. Nəticədə if operatoru aşağıdakı kimi daha sadə şəkil alar:

```
if (şərt){

  // icra olunmalı kod

}
```

Əgər icra olunmalı operatorların sayı 1-dən çox deyilsə onda hətta fiqurlu mötərizələri də yazmamaq olar, aşağıdakı kimi:

```
if (şərt)
  əməliyyat
```

İndi isə if operatorundan istifadəyə aid proqram nümunələri ilə məşğul olaq.

Çalışma: İstifadəçinin daxil etdiyi ədədin 20-dən böyük olduğunu müəyyən edən proqram tərtib edin.

Həlli: Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

  int main (){

    int x;

    cout<< "Zehmet olmasa her-hansi eded daxil edin\n";
    cin>>x;

    if ( x > 20)
        cout << "Siz 20-den boyuk eded daxil etdiniz \n";

}
```

Görürük ki biz bu proqramda `"Siz 20-den boyuk eded daxil etdiniz \n"` sətrini çap edən cout operatorundan

```
cout << "Siz 20-den boyuk eded daxil etdiniz \n";
```

əvvəl if operatorunu yazmışıq

```
if ( x > 20)
```

Bu o deməkdir ki həmin cout operatoru yalnız onun aid olduğu if operatorunun şərti ödəndiyi zaman icra olunacaq. Şərtimiz isə x dəyişəninin qiymətinin 20-dən böyük olmasıdır

```
x > 20
```

Beləliklə bu proqram icra olunan zaman `"Siz 20-den boyuk eded daxil etdiniz \n"` sətri yalnız istifadəçi 20-dən böyük qiymət daxil etdikdə çap olunacaq. Əks halda isə çap olunmayacaq. Bizdən də hələlik bu tələb olunur. Gəlin proqramı 20-dən kiçik və böyük müxtəlif ədədlər daxil etməklə icra edək və nəticələri təhlil edək. Əvvəlcə 20 –dən böyük ədəd daxil edək, misal üçün 34:

İcra 1:

```
Zehmet olmasa her-hansi eded daxil edin
34
Siz 20-den boyuk eded daxil etdiniz
```

Gördüyümüz kimi bu halda daxil etdiyimiz ədəd 20-dən böyük olduğuna görə if operatorunun şərti ödəndi, yəni `x > 20` şərti doğru qiymət aldı və buna görə if operatoru ilə verdiyimiz çap operatoru icra olundu.

İndi isə yenidən proqramı icra edək, lakin bu dəfə 20-dən kiçik qiymət daxil edək.

İcra 2:

```
Zehmet olmasa her-hansi eded daxil edin
5
```

Gördüyümüz kimi bu halda şərt ödənmədiyinə görə if operatoru ilə verilmiş cout icra olunmadı.

Çalışma: İstifadəçinin daxil etdiyi ədədin 3-ə bölündüyünü müəyyən edən proqram tərtib edin.

Həlli: Bizə bir ədədin digərinə bölünüb-bölünmədiyini tapmaq bir çox çalımalarda lazım olacaq. Bunu müəyyən üçün aşağıdakı qaydadan istifadə etmək lazımdır. Əgər bir ədəd digərinə bölünəndə qalıqda 0 (sıfır)qalırsa bu o deməkdir ki bu ədələr bölünür. If operatoru ilə qalığı hesablayıb 0-ra bərabərliyin yoxlasaq bölünmə şərtini müəyyən edə bilərik. Çalışmada bizdən istifadəçinin daxil etdiyi ədədin 3-ə bölünməsini müəyyən etmək tələb olunur. Bunun üçün biz həmin ədədin 3-ə bölünməsindən alınan qalığın 0-ra bərabər olub-olmamasını yoxlamalıyıq. Əgər qalıq 0-dırsa onda həmin ədəd 3-ə tam bölünür. Bərabərlik simvoluşərti C++ dilində == simvolu ilə işarə olunur. Riyaziyyatdan

bildiyimiz iki bərabərlik simvolunu bitişik yazırıq. Bir bərabərlik simvolu mənimsətmə operatorudur. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

  int main (){

    int x, qaliq;

    cout << "Her-hansi daxil edin. \n";

    cin >> x;

    qaliq = x % 3;

    if ( qaliq == 0 )
        cout << x << " 3-e bolunur \n";

}
```

İzahı: Əvvəlcə istifadəçinin daxil etdiyi ədədi ***x*** dəyişəninə mənimsədib, 3-ə bölünmədən alınan qalığı **qaliq** dəyişəninə yerləşdiririk. if operatorunun şərtini `qaliq == 0` kimi vermişik. Əgər qalıq 0 olarsa onda şərt ödənəcək və ekranda müvafiq sətir çap olunacaq. Şərt ödənmədiyi halda heçnə çap olunmayacaq.

Müxtəlif ədədlər daxil etməklə proqramın nəticələrini yoxlayaq:

İcra 1:

```
Her-hansi daxil edin.
126
126 3-e bolunur
```

İcra 2:

```
Her-hansi daxil edin.
44
```

Nəticələr bizə aydındı. Şərt ödənəndə if operatounun kodu icra olunur, ödənməyənədə yox.

Yuxarıdakı nümunələr ilə if operatorunun nə cür işlədiyini test elədik, lakin şərt ödənməyəndə proqramın reaksiya verməməsi biraz çatışmamazlıq hissi yaşadır. Yaxşı

olardı ki, şərt ödənmədiyi halda da proqram bizə hansısa məlumat versin. Bunun üçün if operatorunu tam versiyasından , yəni else –ni dən də istifadə edən versiyasından istifadə etmək lazımdır.

If operatorunun tam formasın bir daha yadımıza salaq

```
if (şərt){

  // şərt ödənəndə icra olunan kod

}
else{

  // şərt ödənməyəndə icra olunan kod
}
```

else hissəsində də əməliyyatların sayı 1-dən çox deyilsə fiqurlu mötərizələri yazmaya bilərik.

3-ə bölünən ədədin proqramını if else vastəsilə tərtib olunan daha düzgün nümunəsi aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

  int main (){

    int x, qaliq;

    cout << "Her-hansi daxil edin. \n";

    cin >> x;

    qaliq = x % 3;

    if ( qaliq == 0 )
        cout << x << " 3-e bolunur \n";
    else
        cout << x << " 3-e tam bolunmur \n";

}
```

Müxtəlif ədədlər daxil etməklə proqramın nəticələrini yoxlayaq:

İcra 1:

```
Her-hansi daxil edin.
```

```
561
561 3-e bolunur
```

İcra 2:

```
Her-hansi daxil edin.
458
458 3-e tam bolunmur
```

Gördüyümüz kimi bu dəfə ədəd 3-ə bölünmədiyi halda da proqram əməliyyat icra eliyir, istifadəçiyə məlumat çap edir. Biz istifadəçiyə ədədin 3-ə bölünmədiyini bildirməklə yanaşı digər bir cout operatoru ilə qalığı da çap edə bilərik. Bu zaman else – də hissəsində icra olunacaq əməliyyatların sayı 1-dən çox olduğuna görə onları fiqurlu { } mötərizələri arasına almalıyıq. Kod belə olar:

```
#include <iostream>

using namespace std;

  int main (){

    int x, qaliq;

    cout << "Her-hansi daxil edin. \n";

    cin >> x;

    qaliq = x % 3;

    if ( qaliq == 0 )
        cout << x << " 3-e bolunur \n";
    else
    {
       cout << x << " 3-e tam bolunmur \n";
       cout << "qaliq = " << qaliq << "\n";
    }

}
```

Müxtəlif ədədlər daxil etməklə proqramın nəticələrini yoxlayaq:

İcra 1:

```
Her-hansi daxil edin.
762
762 3-e bolunur
```

İcra 2:

```
Her-hansi daxil edin.
698
698 3-e tam bolunmur
qaliq = 2
```

İcra 3:

```
Her-hansi daxil edin.
340
340 3-e tam bolunmur
qaliq = 1
```

Şərt ödəndiyi halda isə bölmədən alınan tam hissəni çap edə bilərik. Bu halda da if –in əməliyyatlarının sayı 1-dən çox olduğuna görə onları fiqurlu mötəizə arasında verməliyik.

```
#include <iostream>

using namespace std;

  int main (){

    int x, qaliq, tam;

    cout << "Her-hansi daxil edin. \n";

    cin >> x;

    tam = x / 3;
    qaliq = x % 3;

    if ( qaliq == 0 )
    {
       cout << x << " 3-e bolunur \n";
       cout << "tam hisse = " << tam << "\n";
    }
    else
    {
       cout << x << " 3-e tam bolunmur \n";
       cout << "qaliq = " << qaliq << "\n";
    }

}
```

4.2 Şərtlərin qurulması – Müqaisə operatorları

C++ dilində şərtlər hansı dəyişənin və ya ifadənin qiymətinin digər bir kəmiyyətlə müqaisə edilməklə qurulur. Bu zaman aşağıdakı müqaisə operatorlarından istifadə olunur. Qeyd edək ki bu operatorlardan ikisi ilə (böyükdür və bərabərdir) biz artıq tanışıq.

```
Böyükdür:       >
Kiçikdir:       <
Bərabərdir:     ==
Fərqlidir:      !=
Boyuk beraberdir    >=
Kicik beraberdir    <=
```

Bu müqaisə operatorlarından istifadə etməklə tərtib edilən bir neçə şərti nəzərdən keçirək:

```
x > y
```

x dəyişəninin qiymətinin y-dən böyük olması şərti. Əgər x dəyişəninin qiyməti y-dən böyük olarsa onda bu şərt ödənər.

```
4 + x < 12
```

Bu şərt x –lə 4-ün cəmi 12 –dən kiçik olduğu alda ödənir. Digər müqaisə operatorları ilə çalışmalarda tanış olacayıq.

Çalışma. İstifadəçinin daxil etdiyi, göstərilən mərtəbə vahidinə qədər yuvarlaqlaşdıran proqram tərtib edin.

Həlli. Əvvəlcə ədədin yuvarlaqlaşdırılmasını bilməyənlər üçün qısa izah verək. Tutaq ki aşağıdakı kimi onluq kəsr hissəsi olan ədəd verilib, 45.86953. Bu ədədi yuvarlaqlaşdırmaq tələb olunur. Bu zaman ədədin yuvarlaqlaşdırılmalı olduğu mərtəbə vahidi də verilməlidir.

4.3 Şərtlərin qiymətləri – true/false (doğru/yalan)

C++ dilində şərtlər 2 qiymət alır ***doğru*** və ***yalan***. Əgər şərt ödənirsə o doğru qiymət alır, əks halda yalan. Bunların C++ dilində müvafiq ifadəsi true və false –dir. Biz şərtlərin yerinə birbaşa bu qimətlərdən də istifadə edə bilərik, misal üçün:

```
if (true)

    emeliyyat
```

Bu halda əməliyyat həmişə ödənəcək, çünki şərti ifadə yerinə birbaşa onun yekun qiymətini yazmışıq, doğru. Bu isə şərtin həmişə ödənməsi deməkdir.

4.4 Iç-içə if operatoru

Biz if operatorunun şərti ödənən zaman istənilən operator yerinə yetirə bilərik, o cümlədən digər bir if operatoru. Aşağıdakı kimi:

```
if (şərt)
    if (şərt)
        əməliyyat
else
    əməliyyat
```

Bu halda belə bir sual ortaya çıxır ki, **else** operatoru hansı **if** operatorunun şərtindən asılı olur, birinci **if** operatorunun, yoxsa ikinci. Bu həm sizdə, həm də sizin kodu sonralar oxuyan digər proqramçılarda çaşqınlıq yarada bilər. Bu zaman fiqurlu mötərizələrdən istfadə etməklə məsələni həll etmək olar, aşağıdakı kimi:

```
if (shert)
{
    if (sert)
        emeliyyat
}
else
    emeliyyat
```

Əgər **else** birinci **if** operatoruna aiddirsə, və ya

```
if (shert)
{
    if (sert)
        emeliyyat
    else
        emeliyyat
}
```

əgər **else** ikinci **if** operatoruna aiddir.

4.5 Mürəkkəb şərtlər – və/və ya operatorları

C++ dilində bir neçə şərt birləşdirilib bir şərt kimi verilə bilər. Bunun üçün **və**, **və ya** operatorlarından istifadə olunur.

4.5.1 və operatoru

və operatoru ilə birləşməsindən yaranan yekun mürəkkəb şərtin doğru qiymət alması üçün həmin mürəkkəb şərtə daxil olan bütün alt şərtlər ayrı-ayrılıqda doğru qiymət almalıdır. Əgər heç olmasa bir şərt yalan qiymət alarsa, onda digər şərtlərin qiymətlərindən asılı olmayaraq yekun şərt yalan qiyməti alar.

və operatorunun sintaksisi aşağıdakı kimidir:

```
şərt1  &&  şərt2
```

C++ dilində və operatoru && kimi işarə olunur. Biz istənilən sayda şərti **və** operatoru ilə birləşdirib mürəkkəb şərt yarada bilərik, aşağıdakı kimi

```
şərt1  &&  şərt2  &&  şərt3  &&  şərt4
```

Çalışma. Aşağıdakı şərtin qiymətini müəyyən edin:

```
5 > 2 && 67 < 123
```

Həlli: Verilən mürəkkəb şərt iki şərtin və operatoru ilə birləşməsindən yaranıb. Əvvəlcə hər iki şərti ayrı – ayrılıqda qiymətləndirək.

Birinci şərt: `5 > 2` – doğru.
İkinci şərt: `67 < 123` – doğru.

Hər iki şərt doğru olduğuna görə onların və operaotoru ilə birləşməsi də doğru qiymət alar.

Cavab: doğru.

Çalışma. Aşağıdakı şərtin qiymətini müəyyən edin:

```
15 == 3 && 6 < 8 && 21 != 22
```

Həlli: Verilən mürəkkəb şərt 3 şərtin və operatoru ilə birləşməsindən yaranıb. Əvvəlcə hər 3 şərti ayrı – ayrılıqda qiymətləndirək.

Birinci şərt: `15 == 3` – yalan.
İkinci şərt: `6 < 8` – doğru.
Üçüncü şərt: `21 != 22` – doğru.

Baxdığımız şərtlərdən biri yalan olduğuna görə hamısının birləşməsi də yalan olur.
Cavab: yalan.

Çalışma: İstifadəçidən 1 ilə 100 arasında ədəd daxil etdiyini müəyyən edən proqram qurun.

Həlli: Burada biz istifadəçinin daxil etdiyi ədədin 1 ilə 100 arasında oduğunu müəyyən etmək üçün mürəkkəb şərtdən istifadə edəcəyik. Beləki daxil olunan ədədin verilən aralıqda olması üçün ədəd eyni anda iki şərti ödəməlidir: həm 1-dən böyük olmalı, həm də 100-dən kiçik. Proqram oduna nəzər salaq:

```
#include <iostream>

using namespace std;

  int main (){

    int x;

    cout << "1 ile 100 arasinda olan eded daxil edin. \n";

    cin >> x;

    if ( x > 1 &&  x < 100 )
        cout << "Ela, siz ededleri yaxshi taniyirsiniz.\n";
    else
        cout << "Sizin riyazi bilikleriniz biraz zeifdir.\n";

}
```

Müxtəlif ədədlər daxil etməklə proqramın nəticələrini yoxlayaq:

İcra 1:

```
1 ile 100 arasinda olan eded daxil edin.
34
Ela, siz ededleri yaxshi taniyirsiniz.
```

İcra 2:

```
1 ile 100 arasinda olan eded daxil edin.
455
Sizin riyazi bilikleriniz biraz zeifdir.
```

4.5.2 və ya operatoru

və ya operatoru ilə birləşməsindən yaranan yekun mürəkkəb şərtin doğru qiymət alması üçün həmin mürəkkəb şərtə daxil olan alt şərtlərdən heç olmasa birinin doğru qiymət alması kifayətdir. Əgər heç bir şərt ödənmərsə onda yekun şərt yalan qiyməti alar.

və ya operatorunun sintaksisi aşağıdakı kimidir:

```
şərt1  ||  şərt2
```

C++ dilində və operatoru || kimi işarə olunur. Biz istənilən sayda şərti və ya operatoru ilə birləşdirib mürəkkəb şərt yarada bilərik, aşağıdakı kimi:

```
şərt1  ||  şərt2  ||  şərt3  ||  şərt4
```

Çalışma. Aşağıdakı şərtin qiymətini müəyyən edin:

```
54 > 2 || 67 > 123
```

Həlli: Verilən mürəkkəb şərt iki şərtin və ya operatoru ilə birləşməsindən yaranıb. Əvvəlcə hər iki şərti ayrı – ayrılıqda qiymətləndirək.

Birinci şərt: `54 > 2` – doğru.
İkinci şərt: `67 > 123` – yalan.

Şərtlərdən heç olmasa biri doğru olduğuna görə yekun şərt doğru qiymət alar.

Cavab: doğru.

4.6 Inkar operatoru

Bəzən verilmiş şərti ifadələrdə və ya sərbəst halda verilmiş şərtin əksinin doğruluğunu yoxlamaq tələb olunur. Bu zaman inkar operatorundan istifadə olunur. İnkar operatoru nida – ‘!’ işarəsi ilə işarə olunur və sintaksisi aşağıdakı kimidir:

```
!şərt
```

Nida işarəsini inkar etmək istədiyimiz şərtin əvvəlinə yazırıq və həmin şərtin qiyməti tərsinə dəyişir: doğrudursa yalan, yalandırsa doğru olur. Çalışmalara baxaq.

Çalışma. Aşağıdakı şərtin qiymətini müəyyən edin:

```
!(5 > 12)
```

Həlli. `5 > 12` şərti doğru deyil, yalandır və ona inkar operatorunu tətbiq etdikdə isə qiyməti əksinə dəyişir və doğru olur. Qeyd edək ki, burada sintaksis gərəyi olaraq şərti mötərizə daxilində yazmışıq.

Çalışma. Aşağıdakı kod icra olunduqda nə çap olunar?

```
#include <iostream>

using namespace std;

  int main (){

    int x;

    x = 5;

    if ( !(x > 6))
        cout << " Birinci setir cap olunacaq. \n";
    else
        cout << " Ikinci setir cap olunacaq. \n";

}
```

Həlli. x dəyişəninin qiyməti 5 olduğuna görə `x > 6` şərti yalan qiymət alır. Bu şərtə inkar operatorunu tətbiq etdikdə isə qiyməti əksinə dəyişərək doğru olur. Yekun şərt doğru olduğuna görə if operatoru daxilində olan əməliyyat yerinə yetirilir.

Cavab: `Birinci setir cap olunacaq.`

Çalışma. Aşağıdakı kod icra olunduqda nə çap olunar?

```
#include <iostream>

using namespace std;

  int main (){

    int x,y;

    x = 45;
    y = 12

    if ( (!(x > 6)) && (y == 30))
        cout << " Birinci setir cap olunacaq. \n";
    else
        cout << " Ikinci setir cap olunacaq. \n";

}
```

Həlli. Dəyişənlərə mənimsədilən qiymətləri nəzərə alaraq deyə bilərik ki, `x > 6` şərti doğru, `y == 30` şərti isə yalan qiyməti alır. Lakin birinci şərtə inkar operatorunu tətbiq etdiyimizdən onun qiyməti yalan olur - `!(x > 6).` Sonda hər iki şərtin qiymətini və -

`&&` operatoru ilə birləşdiririk. Yekun şərt yalan qiyməti alır və if operatorunun yox, else –nin əməliyyatı icra olunur.

Cavab: `Ikinci setir cap olunacaq.`

4.7 Alqoritmik biliklər.

Biz demək olar ki C++ dilində şərt operatoruna bir çox qaydalarla tanış olduq. İndi isə əldə etdiyimiz yeni biliklərdən istifadə edərək alqoritm qurma vərdişlərimizi daha da inkişaf etdirək, örəndiyimiz biliklərin konkret tətbiqlərin həllində nə cür istifadə edilməsi ilə tanış olaq.

Çalışma. İstifadəçinin daxil etdiyi iki ədəddən hansının böyük olduğunu müəyyən edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

  int main (){

    int x,y;

    cout<< "Zehmet olmasa iki eded daxil edin\n";
    cin>>x>>y;

    if ( x > y )
        cout << " En boyuk " << x <<"\n";
    else
        cout << " En boyuk " << y <<"\n";

}
```

İzahı. Məncə bu elə də çətin bir məsələ deyil. Maraqlı məsələ növbəti proqramdır.

Çalışma. İstifadəçinin daxil etdiyi üç ədəddən hansının böyük olduğunu müəyyən edən proqram tərtib edin.

Həlli. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

  int main (){
```

```
    int x,y;

    cout<< "Zehmet olmasa iki eded daxil edin\n";
    cin>>x>>y;

    if ( x > y  && x > z)
        cout << " En boyuk " << x <<"\n";

    if ( y > x  && y > z)
        cout << " En boyuk " << y <<"\n";

    if ( z > x  && z > y)
        cout << " En boyuk " << z <<"\n";
}
```

İzahı: Üç ədəd verilib: x, y, z. Bunlardan ən böyüyünü müəyyən etməliyik. Ən böyük digər yerdə qalan ədədlərin hamısından böyük olana deyilir. Misal üçün əgər x ən böyükdürsə onda o həm y-dən, həm də z-dən böyük olmalıdır. Birinci if operatorunda biz bu şərti yoxlayırıq.

```
if ( x > y  &&  x > z)
    cout << " En boyuk " << x <<"\n";
```

Burada x-in y-dən böyük olması və z –dən böyük olması şərtləri məntiqi və - **&&** operatoru ilə birləşdirilir. Nəticədə hər ikisinin eyni anda ödənməsi şərti təmin olunur. Başqa sözlə if operatoruna verdiyimi `( x > y  &&  x > z)` şərti o vaxt doğru olar ki, x həm y-dən, həm də z-dən böyük olsun, yəni ən böyük olsun. Əgər bu şərtlərdən heç olmasa biri doğru olmazsa, x məsələn z-dən böyük olmazsa onda `x > z` şərti yalan qiyməti alar. və ilə birləşdiyinə görə də yekun şərt ümumilikdə yalan qiyməti alar və if operatorunun əməliyyatı icra olunmaz.

Eyni qayda ilə digər dəyişənləri də yoxlayırıq. Əslində sonuncu dəyişəni yoxlamağa ehtiyac yox idi, çünki əgər ilk iki dəyişəndən heç biri ən böyük deyilsə, deməli avtomatik olaraq ən böyük üçüncü ədəd olur.

Çalışma. Əgər müştəri 100 manatdan çox məbləğdə alış-veriş edərsə onda onun 100 manatdan atrıq xərclədiyi məbləğin 20% həcmində endirim edilsin. Siz elə proqram tərtib etməlisiniz ki, istifadəçidən xərclədiyi məbləği daxil etməsini istəyir və onun ödəməli olduğu pulun məbləğini çap edir. Əgər endirim olunursa, endirim olunan vəsait də ayrıca bildirilir.

Çalışmanı elə dətiş ki, pulu xarici valyuta ilə daxil etdikdə də proqram işləsin.

Həlli.

```cpp
#include <iostream>

using namespace std;

  int main (){

    double x,y;

    cout<< "Zehmet olmasa meblegi daxil edin\n";
    cin>>x;

    //eger mebleg 100 manatdan coxdursa
    // onda 20% endirim edeceyik

   if (x > 100)
    {
        //endirimin meblegin hesablayaq, 20%
        y = (x*20)/100;

        //esab meblegden endirim olunani cixaq
        x -= y;

        cout<<" Size "<< y << " manat endirim olundu \n";
    }
   else
        cout<<" Size endirim olunmadi \n";

    cout<<" Zehmet olmasa " << x
        << " manat odenish edin \n"
        <<" Teshekkur edirik. \n";
}
```

Icra 1.

```
Zehmet olmasa meblegi daxil edin
112
Size 22.4 manat endirim olundu
Zehmet olmasa 89.6 manat odenish edin
Teshekkur edirik
```

Icra 2.

```
Zehmet olmasa meblegi daxil edin
96
Size endirim olumadi
Zehmet olmasa 96 manat odenish edin
Teshekkur edirik
```

Çalışma 1. İstifadəçinin daxil etdiyi iki ədədin birinin digərinə bölünməsindən alınan tam hissənin 0-dan böyük olub-olmadığını müəyyən edən proqram tərtib edin.

Nümunə nəticə 1

```
Iki eded daxil edin
34 6
34 -in 6 -e bolende tam hisse = 5
tam hisse 0 -dan boyukdur
```

Nümunə nəticə 2

```
Iki eded daxil edin
123
200
123 -in 200 -e bolende tam hisse = 0
Tam hisse 0 -dan boyuk deyil
```

Çalışma 2. İstifadəçinin daxil etdiyi üç ədədin birincisinin ikinciyə bölünməsindən alınan qalığın üçüncü ədədə bölünməsindən alınan tam hissənin 0-dan böyük olub-olmadığını müəyyən edən proqram tərtib edin.

Nümunə nəticə

```
Uc eded daxil edin
325 42 8
325 -in 42 -e bolende qaliq hisse = 31
31 -in 8 -e bolende tam hisse = 3
tam hisse 0 -dan boyukdur
```

Çalışma 3. 1 manatdan az qalığı qaytarmaq üçün neçə dənə 50,20, 10, 5,3 və 1 qəpik tələb olunduğunu hesablayan proqram tərtib edin.

Nümunə nəticə:

```
Mebleqi daxil edin ( < 1 man).
98
Qaliq qaytarilmalidir:
  1 elli qepik
  2 iyirmi qepik
  1 besh qepik
  1 uc qepik
```

Göstəriş: qaliq(1 manatdan kiçik qəpiklərin sayı) 100-dən kiçik tam ədəddir. Neçə dənə 50-lik yerləşdiyini bilmək üçün həmin ədədi 50-yə bölüb tam hissəni tapırıq.

Çalışma 4. İşçinin bir iş saatlıq pulu və bir həftə ərzində iş saatlarının miqdarı verildikdə, onun həftəlik məvacibini hesablayan proqram qurun. 40 saatdan artıq olan saatlar əlavə iş saataları kimi qiymətləndirilsin və bu saatların pulu ikiqat hesablansın.

Çalışma 5. İstifadəçinin daxil etdiyi 2 ədədi artan sırada çap edən proqram tərtib edin.

```
Iki eded daxil edin
23 4
Sizin daxil etdiyiniz ededler artan sirada
4 23
```

Çalışma 6. Ədəd tapmaq oyunu proqramını tərtib edin. Proqram yaddaşında bir ədəd saxlayır və istifadəçidən hər-hansı ədəd daxil etməsini istəyir. İstifadəçinin daxil etdiyi ədədi proqram yaddaşında tutduğu ədədlə müqaisə edərək “kiçikdir” və ya “böyükdür” çap edir və beləlilə istifadəçiyə ədədi tapmağa kömək edir. İstifadəçi ədədi düzgün daxil etdikdə isə “Siz axtarilan ededi tapdiniz” çap edir.

Calışma 7. Elə proqram tərtib edin ki, daxil olunan yaşa görə insanın uşaq(1-12), yeniyetmə(13-17), gənc(18-33), orta yaş(33-55), yaşlı(56-70) və ya qoca(> 70) olduğunu müəyyən etsin.

Calışma 8. Yuxarıdakı çalışmanı elə dəyişin ki, başlanğıc məlumat olaraq istifadəçidən hazırkı ili və təvəllüdün qəbul etsin.

Çalışma 9. 3 müxtəlif endirim faizi təklif edən alış-veriş proqramını tərtib edin. Belə ki, əgər xərclənən pulun miqdarı 100 manatdan azdırsa heç bir endirim edilmir. 100 manatdan 200 manata qədər olan alış-veriş zamanı 10%, 200 manatdan yuxarı alış-veriş zamanı isə 20 % endirim edilir.

Nümunə nəticələr:

Nəticə 1

```
Meblegi daxil edin
34
Odenmeli mebleg 34
```

Nəticə 2

```
Meblegi daxil edin
120
Endirim 12
Odenmeli mebleg 108
```

Nəticə 3

```
Meblegi daxil edin
345
Endirim 69
Odenmeli mebleg 276
```

Çalışma 10. İstifadəçinin daxil etdiyi ədədin ikiyə və ya üçə bölündüyünü müəyyən edən proqram tərtib edin.

Nümunə nəticələr:

Nəticə 1

```
Ededi daxil edin
34
34 2 ve ya 3-e bolunur
```

Nəticə 2

```
Ededi daxil edin
67
67 2 ve ya 3-e bolunmur
```

Çalışma 11. Elə proqram tərtib edin ki, istifadəçidən anadan olduğu ayı və günü və bügünkü tarixi (ayı və günü) soruşur və müəyyən edir ki, bu gün istifadəçinin doğum günüdür, yoxsa, bu il üçün artıq doğum günü keçib, yoxsa hələ qabaqdadır.

Nümunə nəticələr:

Nəticə 1

```
Anadan oldugunuz ay ve gunu daxil edin.
6 1
Bugun ucun ay ve gunu daxil edin
11 5
Sizin bu ilki ad gununuz kecib
```

Nəticə 2

```
Anadan oldugunuz ay ve gunu daxil edin.
8 23
Bugun ucun ay ve gunu daxil edin
8 22
Sizin bu ilki ad gununuz hele qabaqdadir
```

Düzgün proqramlaşdırma qaydaları: Proqram yalnız tələb olunan problemi həll etməklə kifayətlənməməlidir. O həm də başlanğıc məlumatların doğruluğunu yuxlamalıdır. Misal üçün yuxarıdakı çalışmaya nəzər salasaq, bilirik ki, ay 1-dən kiçik, 12 –dən böyük, ayın günləri isə 1-dən kiçik 31 –dən böyük ola bilməz. Amma əgəg siz bunu çalışma 11 –i həll edərkən nəzərə almamısınızsa istifadəçi bugünkü gün və dogulduğu gün üçün ay və günü hər ikisini 45 67 daxil etsə proqramınız çap edəcək ki,

bu gün sizin ad gününüzdür. Aşağıdakı çalışmada bunu nəzərə alaraq 11-ci çalışmanı tərtib etmək tələb olunur.

Çalışma 12. Çalışma 11-dəki proqramı elə dəyişin ki, istifadəçi ay və günü yanlış daxil etdikdə ona bu barədə məlumat versin.

§5 Dövr Operatorları

Dövr operatorları verilmiş kodu bir neçə dəfə ***təkrar*** icra etməyə imkan verir. Hər-hansı kod parçasının təkrar yerinə yetirilməsinə proqramlaşdırmada tez-tez ehtiyac yaranır. Bunun üçün C++ dilində **`while`**, **`do while`** və **`for`** dövr operatorlarından istifadə olunur. Bu operatorların hər biri təkraroluma sayını verilmiş şərt vastəsilə müəyyən edir və həmin şərt ödənənə qədər kodu təkrarlayırlar.

5.1 while dövr operatoru

while dövr operatorunda təkrarolunma sayı verilmiş şərt vastəsilə müəyyənləşir. Sintaksisi aşağıdakı kimidir:

```
while ( şərt )
   əməliyyat
```

Əgər əməliyyatların sayı birdən artıqdırsa onda onları fiqurlu mötərizə daxilinə almalıyıq, aşağıdakı kimi:

```
while ( şərt ){
   əməliyyat1
   əməliyyat2
 }
```

Əgər dövr daxilində yalnız bir əməliyyat iştirak edirsən onda onu fiqurlu mötərizə daxilinə yerləşdirmək məcburi deyil, lakin yerləşdirməyimizdə də bir yanlış olmaz. Bu artıq istəyimizə bağlı olan bir məsələdir.

```
while ( şərt ){
   əməliyyat
 }
```

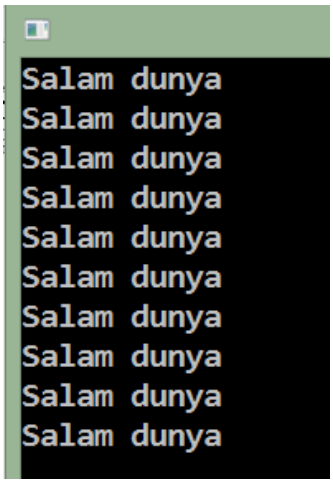
Bu zaman nə qədər ki, şərt ödənir, yəni doğru qiymət alır **while** operatoru daxilindəki əməliyyat təkrar yerinə yetiriləcək.

while dövr operatoruna aid proqram nümunələri ilə tanış olaq.

Nümunə 1:

```
#include <iostream>

using namespace std;

int main (){

    int x;

    x = 0;

    while ( x < 10 ){
        cout << "Salam dunya \n";
        x++;
    }

}
```

Əgər bu kodu icra etsək ekranda 10 dəfə Salam dunya sətri çap edər.



İzahı:

Gəlin proqramın izahı ilə tanış olaq. **while** operatorunun şərti olaraq `x < 10` yazmışıq. Bu o deməkdir ki, nəqədərki x dəyişəni 10-dan kiçik qiymət alır dövr daxilində göstərilən əməliyyatları təkrar et.

Dövr daxilində iki əməliyyat göstərmişik:

```
cout << "Salam dunya \n";
```

və

```
x++;
```

Bunlardan birincisi ekranda **"Salam dunya \n"** sətrini çap edir. İkincisi isə x dəyişəninin cari qiymətini 1 vahid artırır. Dövrün əvvəlində x -ə 0 qiyməti mənimsədirik:

```
x = 0;
```

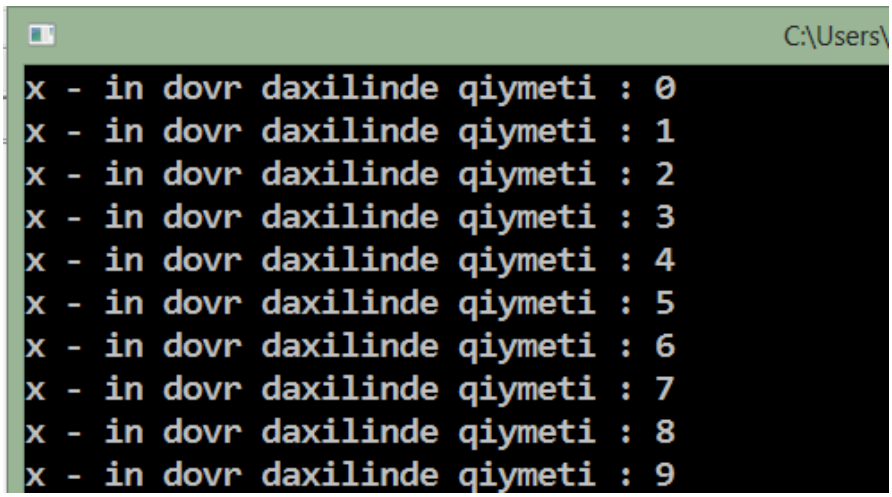
Dövrün daxilində hər dəfə x –in qiyməti 1 vahid artır və o növbə ilə 1,2,3, .. qiymətlərini alır. Hər dəfə dövr başa çatdıqdan sonra əvvələ qayıtdıqda **x < 10** şərti yenidən yoxlanılır. Nə qədər ki x dəyişəni 10-a çatmayıb **x < 10** şərti ödənir, buna görə də dövr təkrar olur. x dəyişəni 10 qiyməti aldıqdan sonra isə göstərilən şərt pozulur, **x < 10** şərti ödənmir və dövr operatoru bitir. İcraolunma **while** operatorundan sonra gələn operatorlara keçir(bağlayan fiqurlu mötərizədən sonra gələn operatorlar, əgər varsa).

Biz dövr daxilində şərtdə istifadə etdiyimiz x dəyişəninin qiymətini çap etməklə onun nə cür dəyişməsini izləyə bilərik. Aşağıdakı kodda bu göstərilir:

Nümunə 1:

```
#include <iostream>

using namespace std;


int main (){

    int x;

    x = 0;

    while ( x < 10 ){
        cout << "x - in dovr daxilinde qiymeti : "
        << x << "\n";

        x++;
    }

}
```

Bu proqramı icra eləsək aşağıdakı nəticəni verər:



İzahı:

Gördüyümüz kimi dövr daxilində x dəyişəni 0-dan 9-a kimi qiymətlər alır. Ən sonuncu dəfə dövr daxilində x-in qiyməti 9-dan 10-a dəyişdikdə təkrarolunma zamanı dövrün şərti ödənmir və dövr başa çatır. Əgər biz dövr başa çatdıqdan dərhal sonra x dəyişəninin qiymətini çap etsək onda onun 10 qiyməti aldığını görərik. Bunun yoxlanılması oxuculara bir çalışma kimi həvalə olunur.

while operatoruna aid növbəti nümunə ilə tanış olaq.

Nümunə 2:

```
#include <iostream>

using namespace std;


int main (){

    int x;

    x = 0;

    while(x != -1){
        cout <<"Her-hansi eded daxil edin  ";
        cin >>x;
        cout << "Siz " <<x <<" daxil etdiniz \n";
        cout << "Dovrden cixmaq ucun -1 daxil edin\n";
    }

}
```

Nəticə:

```
Her-hansi eded daxil edin  4
Siz 4 daxil etdiniz
Dovrden cixmaq ucun -1 daxil edin
Her-hansi eded daxil edin  6
Siz 6 daxil etdiniz
Dovrden cixmaq ucun -1 daxil edin
Her-hansi eded daxil edin  -1
Siz -1 daxil etdiniz
Dovrden cixmaq ucun -1 daxil edin
```

İzahı:

while operatoruna şərt olaraq `x != -1` vermişik. Yəni *nəqədərki* x –in qiyməti -1 –dən fərqlidir dövr daxilində verilmiş operatorları yerinə yetir. Dövr daxilində biz istifadəçidən x-in qiymətini daxil etməyini istəyirik və həmin qiyməti ekranda çap edirik. İstifadəçi -1 qiymətini daxil etdikdə dövr başa çatır.

5.2 do while dövr operatoru.

C++ dilində **while** operatoru ilə yanaşı **do while** dövr operatoru da təyin olunub. Sintaksisi aşağıdakı kimidir:

```
do{
    əməliyyat
} while  ( şərt );
```

Hər şey tamamilə while operatorunda olduğu kimidir, fərq yalnız ondadır ki, əməliyyatlar şərtin yoxlanışmasından əvvəl icra olunduğundan şərt ödənməsə belə ən azı bir dəfə icra olunurlar. while operatorunda isə belə deyildi, əgər şərt başlanğıcdan heç ödənmirdisə, onda ümumiyyətlə dövrə giriş olmurdu, yəni dövr başlamadan bitirdi.

5.3 for dövr operatoru

C++ dilinin nisbətən çətin dövr operatoru **for** dövr operatorudur.

for operatoru verilmiş əməliyyatı tələb olunan sayda, misal üçün 10, 50 , 100 dəfə təkrar yerinə yetirmək üçün istifadə olunur. Əgər **while** operatounda şərtin nə vaxt ödənməyəcəyini və deməli dövrlərin sayını əvvəlcədən bilmirdiksə, **for** operatorunda əksinə dövrlərin sayı əvvəlcədən bizə məlum olur. Daha düzgün desək, əgər dövrlərin sayı əvvəlcədən məlumdursa onda **for**, əks halda isə **while** operatorundan istifadə edirik. **for** operatorunda dövrlərin sayı sayğac vastəsilə tənzimlənir. Sayğac olaraq **tam** və ya **simvol** tipli dəyişənlərdən istifadə edə bilərik.

for operatorunun sintaksisi aşağıdakı kimidir:

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
   əməliyyat
```

Əgər əməliyyatların sayı birdən çoxdursa onları blok daxilində yerləşdirməliyik(fiqurlu mötərizələr), aşağıdakı kimi:

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
 {
    əməliyyat1
    əməliyyat2
 }
```

Əməliyyatın sayı bir dənə olanda da onu blok daxilinə yerləşdirə bilərik, bu artıq istəyimizdən asılıdır.

for operatoru aşağıdakı kimi işləyir: Əvvəlcə sayğac dəyişəninə ilkin qiymət mənimsədilir.

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
   əməliyyat
```

Bu iş for operatorunun bütün icrası zamanı yalnız bir dəfə, ən əvvəldə icra olunur. Buna görə o hissəni göy fonda vermişik. Dövrün sayğacına ilkin qiymət mənimsətdikdən sonra dövr operatorunun işi aşağıdakı addımların təkrar yerinə yetirilməsi şəklindən davam edir. Bu addımlar aşağıdakılardır:

Birinci addım: Şərt yoxlanılır

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
   əməliyyat
```

Buraya biz istənilən şərti ifadə yaza bilərik, C++ dili bizə qadağa qoymur, lakin əksər hallarda bu şərti ifadə sayğac dəyişəninin qiyməti ilə bağlı olur. Məntiq isə belədir: Əgər şərt ödənirsə onda göstərilən əməliyyatı icra et, əks halda dövrü başa çatdır.

Beləliklə şərt ödənsə onda dövrün ikinci addımı olan əməliyyatlar icra oluncaq.

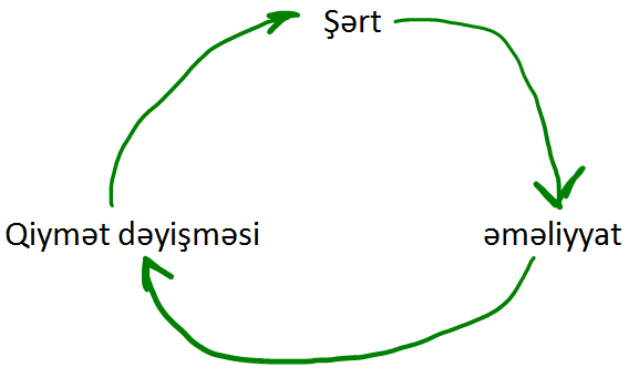
İkinci addım: Əməliyyatların icrası. Qeyd elədiyimiz kimi dövr daxilində bir və ya fiqurlu mötərizələr içində bir neçə əməliyyat yerləşdirə bilərik. İkinci adımda həmin əməliyyatlar yerinə yetiriləcək.

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
   əməliyyat
```

Əməliyyat icra olunduqdan sonra növbə nəhayət sonuncu addıma Sayğacın dəyişməsinə çatacaq:

```
for ( Sayğac = Başlanğıc_Qiymət; Şərt; Sayğacın Dəyişməsi)
   əməliyyat
```

Burada məqsəd sayğacın qiymətini dəyişməkdir. Sayğacın qiymətini istədiyimiz kimi dəyişə bilərik: artıra, azalda, vura, bölə v.s. Sayğacın qiyməti də dəyişdirildikdən sonra təkrar birini addıma – yəni şərtin yoxlanmasına keçilir. Bir daha qeyd edim ki, bu dövrü prosesdə sayğaca başlanğıc qiymətin mənimsədilməsi iştirak etmir.



Gördüyümüz kimi ilkin qiymət mənimsətməsi “oyunda” iştirak etmir, o bayaq da qeyd etdiyimiz kimi yalnız bir dəfə dövrə başlayanda icra olur.

Beləliklə sual: **for** dövr operatorunda əməliyyatlar neçə dəfə təkrar icra olunur?

Cavab: **for** operatorunda təkrarlanmaların sayı həm sayğaca mənimsətdiyimiz ilkin qiymətdən, həm verdiyimiz şərtdən, həm də sayğacın qiymətin necə dəyişməyimizdən asılı olaraq: heç icra olunmaya bilər, bir neçə dəfə və ya sonsuz sayda icra oluna bilər.

Sonuncu halda proram donacaq və heç nəyə cavab verməyəcək. Bu halda proqramı dayandırmaq üçün **CTRL + X** düymələr cütünü basmalıyıq.

Nümunə:

Aşağıdakı nümunə proqramda **for** dövr operatorundan istifadə etməklə ekranda 0-dan 9-a kimi ədədlər çap olunur.

```
#include <iostream>

using namespace std;

int main (){

    int i;

    for ( i=0;  i<10; i++)
        cout<<"i = " <<i << "\n";

}
```

Nəticə:

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

İzahı: Gəlin bu proqramın icra prosesini addım – addım izləyək. Proqramda əvvəlcə tam tipli i dəyişəni elan edirik.

```
int i;
```

Daha sonra for operatorunun başlıq hissəsi gəlir.

```
for ( i=0;  i<10; i++)
```

Burada sayğac olaraq i dəyişənindən istifadə edirik. Sayğaca başlanğıc qiymət olaraq 0 qiymətini mənimsədirik.

```
for ( i=0;  i<10; i++)
```

Ən birinci bu icra olunacaq:

Birinci addım: sayğaca ilkin qiymət mənimsət

```
for ( i=0;  i<10; i++)
```

Artıq bildiyimiz kimi bu yer yalnız bir dəfə dövr başlayanda icra olunur və daha heç vaxt ***təkrarlanmır***. Sayğaca ilkin qiymət mənimsədildikdən sonra başa çatma şərti yoxlanılır. Dövrün başa çatma şərti olaraq `i<10` göstərmişik. Bu o deməkdir ki, nə qədər ki, i dəyişəni 10-dan kiçikdir dövrü təkrar et.

İkinci addım: şərti yoxla

```
for ( i=0;  i<10; i++)
```

Şərt ödəndiyindən (0 < 10) əməliyyatlar icra olunacaq. Əməliyyat olaraq

```
cout<<"i = " <<i << "\n";
```

göstərmişik.

Üçüncü addım: Dövrün əməliyyatlarını icra et

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";
```

Əməliyyat olaraq biz sayğacın cari qiymətini ekranda çap edirik.

Sıra gəlib çatdı sayğacın qiymətinin dəyişdirilməsinə. Sayğacın dəyişməsi olaraq isə `i++` göstərmişik. Bu kod icra olunanada sayğacın qiymətin 1 vahid artır. `i++` –dan sonra nöqtəvergül işarəsinin qoyulmamasına diqqət yetirək.

Dördüncü addım: sayğacın qiymətin dəyiş

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";
```

Hal-hazırda i-nin qiyməti 1 olduğuna görə bu kod icra olunandan sonra i-nin qiyməti 1 vahid artaraq 2 olar. Bununla bizim dövrün ilkin tsikli tamamlanmış oldu. Növbəti dövrlərin addımları bu birincidən bir qədər fərqli olur. Beləki növbəti dövrlərdə birinci addım – yəni sayğaca ilkin qiymət mənimsədilməsi addımı icra olunmayacaq. İlkin qiymət mənimsədilməsi yalnız bir dəfə dövr başlayanda icra olunur. Növbəti dövrlərin addımları sayğacın qiymətinin dəyişməsi, şərtin yoxlanması, əgər şərt ödənirsə onda kodun icrasınnan ibarət olur. Proqramın icrasını addımlarla sıralamamızı davam edək. Axırıncı addımda sayğacın qiymətini dəyişməkdə qalmışdıq. Növbəti addım şərti yoxlamaqdır:

Beçinci addım: şərti yoxla

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";

  // i -in qiyməti 2 olduğundan şərt ödənir deməli kod icra olunacaq
```

Altıncı addım: Kodu icra elə

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";

  // Beşinci addımda şərt ödəndiyinə görə dövr daxilində verdiyimiz kod
  icra olunur və ekranda i = 2 çap olunur
```

Yeddinci addım: sayğacın qiymətin dəyiş

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";

  // i-in qiyməti 2 idi, olur 3
```

Səkkizinci addım: şərti yoxla.

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";

  // 3 < 10 şərt ödənir, deməli dövr davam eliyir
```

Doqquzuncu addım: kodu icra et

```
for ( i=0;  i<10; i++)
    cout<<"i = " <<i << "\n";


  // ekranda i = 3 sətri çap olunur
```

Bununla biz for dövr operatorumuzun başlıq hissəsini tamamlamış olduq. Əgər dövr daxilində sayğacın qiymətini dəyişməsək (bunu da eliyə bilərik, irəlidə baxacayıq) onda bu dövr 10 dəfə təkrar olunacaq.

Dövr daxilində isə cəmi bir əməliyyat yazmışıq:

```
cout<<"i = " <<i << "\n";
```

Bu əməliyyat icra olunduqda sayğacın hazırki qiyməti ekranda çap olunur.

Başqa nümunəyə baxaq.

Nümunə:

```
#include <iostream>

using namespace std;


int main (){

    int j;

    for ( j=5;  j<20; j = j + 3)
        cout<<"j = " <<j << "\n";

}
```

Nəticə:

```
j = 5
j = 8
j = 11
j = 14
j = 17
```

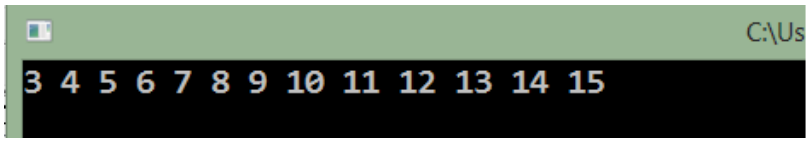
İzahı: Sayğac olaraq j dəyişənindən istifadə edirik, sayğaca başlanğıc qiymət olaraq 5 mənimsədilir, dövrün başa çatma şərti `j<20` vermişik. Dövr hər dəfə təkrarlandıqda sayğacın qiymətini 3 vahid artırırıq.

Gördüyümüz kimi dövrün daxilində sayğacın qiymətini çap etməklə müxtəlif maraqlı proqramlar qura bilərik. Misal üçün aşağıdakı nümunədə 3 –dən 12-yə kimi ədədlər çap olunur.

```
#include <iostream>

using namespace std;

int main (){

    int i;

    for ( i=3;  i<=15; ++i)
        cout << i << " ";
```

```
}
```

Nəticə:



Nümunə:

Aşağıdakı nümunə kod 1-dən 20-yə kimi ədədlər arasında 3-ə bölünənləri çap edir.

```
#include <iostream>

using namespace std;


int main (){

    int i;

    cout << "1 ile 20 arasinda 3-e bolunenler \n";

    for ( i=1;  i<20; i++)
        if ( i%3 == 0)
            cout << i << " ";

}
```

Nəticə:

```
1 ile 20 arasinda 3-e bolunenler
3 6 9 12 15 18
```

İzahı:

Dövrdə sayğac 1-dən 20-yə kimi dəyişir. Dövr daxilində if operatoru ilə sayğacın hazırki qiymətinin 3-ə bölündüyünü müəyyən edirik. Sayğacın 3-ə bölünən qiymətlərini ekranda çap edirik. Növbəti çalışmada isə 1 ilə 100 ədədləri arasında istifadəçinin daxil etdiyi ədədə bölünənləri tapacayıq.

Nümunə.

```
#include <iostream>
```

```
using namespace std;


int main (){

    int i, x;

    cout << "Her hansi eded daxil edin: ";
    cin >> x;

    cout << "1 ile 100 arasinda " << x <<" -e bolunenler \n";

    for ( i=1;  i<100; i++)
        if ( i%x == 0)
            cout << i << " ";

}
```

Nəticə.

```
Her hansi eded daxil edin: 13
1 ile 100 arasinda 13 -e bolunenler
13 26 39 52 65 78 91
```

İzahı: Proqram əvvəlki nümunəyə oxşardır. Sadəcə burada biz istifadəçinin daxil etdiyi ədədi yadda saxlamaq üçün əlavə olaraq x dəyişənindən istifadə edirik və sayğacın qiymətin 1-dən 100-ə qədər dəyişib, içərisindən x-ə bölünənləri çap edirik.

Görüdüyümüz kimi for dövr operatoru bizə sayğacın başlanğıc, son qiymətlərinin və dəyişməsinin təyini üçün tam sərbəstlik verir. Bu isə bizə həm dövrlərin sayını, həm də paralel sayğacın qiymətini məslənin tələbinə uyğun olaraq istədiyimiz kimi dəyişməyə imkan yaradır. Nümunə üçün növbəti baxacağımız kodda biz dövrün başlanğıc qiymətini son qiymətindən böyük götürəcəyik, sayğacın qiymətini isə hər dəfə bir vahid azaldacayıq. Koda baxaq:

Nümunə

```
#include <iostream>

int main (){

    int i;

    for ( i=10;  i > 0; i--)
        cout<<"i = " <<i << "\n";

}
```

Nəticə

```
i = 10
i = 9
i = 8
i = 7
i = 6
i = 5
i = 4
i = 3
i = 2
i = 1
```

for operatoruna aid bu cür çoxlu nümunələr gətirmək olar, lakin onların hamısının iş prinsipi eynidir.

Beləliklə biz C++ dilində **while** və **for** dövr operatorları ilə tanış olduq. Sadəcə bu operatorların nə cür işlədiyini yaxşı bilmək dövrlərə aid proqramları sərbəst qurmağa kifayət etmir. Növbəti hissədə biz dövrlərə aid olan əsas bir neçə məqama diqqət çəkəcəyik. Əsas başa düşməli olduğumuz məsələ dövrü olaraq cəmin və hasilin hesablanması və bu hesablanmada istifadə olunan proqramlaşdırma fəndlərini qavramaqdır.

5.4 Cəmin hesablanması

Hansısa kəmiyyətlərin cəmini hesablamaq proqramlaşdırmada tez-tez tələb olunur. Biz indi dövr vastəsilə cəmi hesablamanın qaydası ilə tanış olacayıq. İlk dəfə biraz adət etmədiyimiz qayda olsa da, qavradıqdan sonra istənilən cəmi hesablamaq bizə çətinlik yaratmayacaq.

Dövr vastəsilə cəmi hesablamağın qaydası aşağıdakı kimidir:

Əvvəlcə, dövrdən əvvəl cəmi yadda saxlamaq üçün hər hansı dəyişən elan edirik. Bu dəyişənə S və ya cem adı verə bilərik.

```
int cem;
```

Daha sonra cəmi yadda saxlamaq üçün elan etdiyimiz dəyişənə 0 qiyməti mənimsədirik.

```
cem = 0;
```

Yəni dövrün əvvəlində cəm boşdu, hələlik heçnə əlavə olunmayıb.

Daha sonra dövrün daxilində cəmin üzərinə nələrin cəmini hesablamaq istəyiriksə onu əlavə edirik.

```
cem = cem + ? ;
```

Nümunə proqrama baxaq.

Aşağıdakı proqramda 1-dən 10-a kimi ədədlərin cəmi hesablanır.

Nümunə:

```
#include <iostream>

using namespace std;

int main (){

    int i, cem;

    cem = 0;

    for ( i=1;  i<10; i++)
        cem = cem + i;

    cout << "1-den 10-a kimi ededlerin cemi = "
         << cem <<"\n";

}
```

Nəticə:

```
1-den 10-a kimi ededlerin cemi = 45
```

Başqa nümunəyə baxaq. Aşağıdakı nümunədə 1-dən 100-ə qədər ədədlər içərisində 3-ə bölünən ədədlərin cəmi hesablanır:

Nümunə:

```
#include <iostream>

int main (){

    int i, cem;

    cem = 0;

    for ( i=1;  i<100; i++)
        if ( i%3 == 0)
```

```
            cem = cem + i;

    cout << "1 ile 100 arasinda 3-e bolunenlerin cemi = "
            << cem <<"\n";

}
```

Nəticə:

```
1 ile 100 arasinda 3-e bolunenlerin cemi = 1683
```

1 ilə 1000 arasında 12 –yə bölünən ədədlərin sayını hesablayan proqram.

Nümunə:

```
#include <iostream>

using namespace std;


int main (){

    int i, say;

    say = 0;

    for ( i=1;  i<1000; i++)
        if ( i%12 == 0)
            say = say + 1;

    cout << "1 ile 1000 arasinda " << say
            << " dene eded 12-ye bolunur \n";

}
```

Nəticə:

```
1 ile 1000 arasinda 83 dene eded 12-ye bolunur
```

5.5 Hasilin hesablanması

Dövrü olaraq müxtəlif kəmiyyətlərin hasillərinin hesablanmasına da aid çox proqram tapşırıqları olur. Hasilin hesablanma qaydası da cəmin hesablanma qaydasına oxşardı. Fərq ondadır ki, hasili yadda saxlamaq üçün elan etdiyimiz dəyişəni, dövrdən əvvəl 1-ə mənimsətməliyik.

Nümunə. Aşağıdakı kod 1-dən 100-ə qədər ədəd arasında istifadəçinin daxil etdiyi ədədə bölünən ədədləri və onların sayını çap edir.

```
#include <iostream>

int main (){

    int k, x, say;

    cout << "Her-hansi eded daxil edin \n";
    cin >> x;

    cout << "1 ile 100 arasinda "<< x
<< " -e bolunen ededler:\n\n";

    say = 0;

    for ( k=1; k<100; k++ )
        if (k % x == 0) {
            cout << k << " ";
            say++;
        }

    cout << "\n Cemi " << say << " eded \n";

}
```

Nəticə:

```
Her-hansi eded daxil edin: 23
1 ile 100 arasinda 23 -e bolunen ededler:

23 46 69 92
 Cemi 4 eded
```

İzahı: Burada biz sayı yadda saxlamaq üçün əlavə say dəyişəni elan elədik və dövrdən əvvəl onu 0-ra mənimsətdik. Dövr daxilində hər dəfə tələb olunan ədəd tapılanda sayı bir vahid artırırıq.

5.6 İç - içə dövr

Dövr daxilində istənilən operatordan o cümlədən dövrdən istifadə edə bilərik. Bu adlanır dövr içərisində dövr və ya iç-içə dövr. Aşağıdakı nümunəyə baxaq:

Nümunə 4. Aşağıdakı kod ekranda alt-alta 10 sətir 1 –dən 10-a kimi ədələri çap edir.

```
#include <iostream>

int main (){

    int i,k;

    for (k=1; k<=10; k++){

        for (i=1; i<=10; i++)
            cout << i  << " ";

        cout << "\n";

    }

}
```

Nəticə:

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

İzahı: Burada biz k sayğacından istifadə edərək dovr qururuq və onun daxilində isə i sayğacından istifadə edərək başqa dövr qururuq. Birinci dovr hər dəfə təkrarlananda daxildəki dovr əvvəldən sona icra olunur və ekranda 1-dən 10-a kimi ədələri çap edir. Daha sonra əsas dövrün digər operatoru ekranda yeni sətir çap edir və əsas dövr təkrarlanır.

Nümunə 5. Aşağıdakı kod ekranda 1-dən 10-a kimi ədədləri alt-alta səliqəli üçbucaq şəklində çap edir.

```
#include <iostream>

int main (){

    int i,k;

    for (k=1; k<=10; k++){

        for (i=1; i <= k ; i++)
```

```
            cout << i  << " ";
        cout << "\n";
    }
}
```

Nəticə:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

İzahı: Daxili dövrün sayğacının son qiymətinə diqqət yetirin. Burada biz daxili dövrdə i sayğacını hər dəfə 1 –dən 10-a kimi yox, 1 –dən k-ya kimi artırırıq. k –nın da qiyməti hər dəfə dövrü olaraq 1-dən 10-a kimi dəyişir və k –nın hər yeni qiymətində i sayğacı 1 –dən k-ya qədər ədədləri ekranda çap edir.

5.7 Sadə ədədlərin tapılması

Bildiyimiz kimi sadə ədəd yalnız özünə və 1-ə bölünən ədədlərə deyilir. Yəni əgər ədəd özündən və 1-dən başqa eç bir ədəd bölünmürsə demək o sadə ədəddir, misal üçün 5, 13, 29 v.s.

Tutaq ki bizdən 1-dən 40-a qədər olan ədədlər içindən sadə ədədləri tapmaq tələb olunur. Bunun üçün 1-dən 40-a qədər dövr təşkil edib hər bir ədədin sadə olub-olmadığını yoxlamalıyıq. Verilmiş hər –hansı ədədin sadəliyini yoxlamaq üçün isə verilmiş ədədin 1-dən həmin ədədin yarısına kimi ədədlərə bölünüb – bölünmədiyini yoxlamalıyıq. Yəni tutaq ki, bizə 11 ədədinin sadəliyini yoxlamaq tələb olunur. Bu zaman onun yarısı 5.5 eliyir , biz yoxlamanı 6-ya kimi davam edə bilərk. 6-dan yuxarı (11 -ə kimi) ədədlərə bölünməyi yoxlamağa ehtiyac yoxdur. Çünki riyaziyyatdan bildiyimiz kimi heç bir ədəd öz yarısından yuxarı ədədlərə bölünmur. Beləliklə kod aşağıdakı kimi olar:

Nümunə

```
#include <iostream>

int main (){

    int i, j, sadedir;

    cout << "1-den 40-a qeder olan sade ededler\n";

    for ( i=1;  i < 40; i++){

        //sadedir -e 1 menimsedek
        sadedir = 1;

        for (j=2; j < i/2; ++j){
            if (i%j == 0){
                sadedir = 0;
                break;
            }
        }

        if (sadedir == 1)
            cout << i << " ";

    }

}
```

Nəticə

```
1-den 40-a qeder olan sade ededler
1 2 3 4 5 7 11 13 17 19 23 29 31 37
```

İzahı:

1-dən 40-a kimi dövr təşkil eliyiriy:

```
for ( i=1;  i < 40; i++){
```

Daha sonra sadedir dəyişəninə 1 qiyməti mənimsədirik.

```
sadedir = 1;
```

sadedir dəyişənini biz ədədin sadə olub-olmamasını yadda saxlamaq üçün istifadə edirik.

Daha sonra 2-dən başlayaraq ədədin yarısına kimi dövr təşkil edirik:

```
for (j=2; j < i/2; ++j){
```

Dövr daxilində ədədin 2-dən öz yarısına kimi olan ədədlərə bölünməsini yoxlayırıq:

```
if (i%j == 0){
```

Əgər ədədin bölündüyü heç olmasa bir dənə ədəd varsa onda deməli o artıq sadə ədəd sayılmır. Ona görə sadedir dəyişəninə 0 qiymətini mənimsədirik və digər ədədləri yoxlamağa ehtiyac qalmadığından dövrü bitiririk.

```
sadedir = 0;
break;
```

Dövrdən sonra isə yoxlayırıq:

```
if (sadedir == 1)
    cout << i << " ";
```

Əgər dövr daxilində ədədin hasısa böləni olubsa onda sadedir dəyişəninə 0 qiyməti mənimsədiləcək. Əks halda ona dövrün əvvəlində mənimətdiyimiz 1 qiyməti yerində qalacaq. Bunu yoxlamaqla müəyyən edirik ki, ədədin böləni olub-ya yox . Əgər olmayıbsa onda deməli ədəd sadədir və onu çap edirik.

Çalışma. İstifadəçinin daxil etdiyi ədədin ən böyük mərtəbə vahidini müəyyən edən proqram tərtib edin.

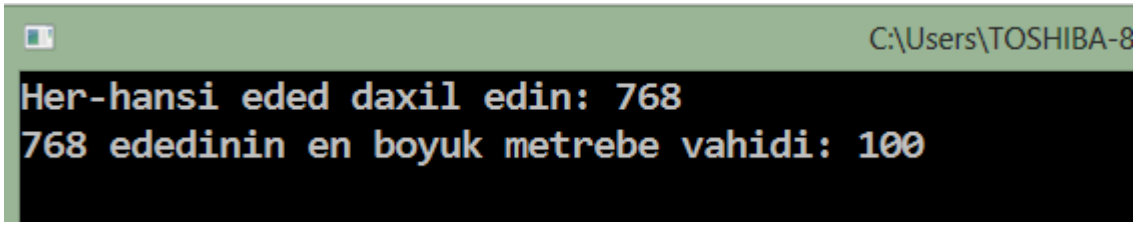
Həlli. Mərtəbə vahidləri barədə dəyişənlər mövzusunda tanış olmuşuq. O zaman dövr operatorları ilə tanış olmadığımıza görə hesabladığımız ədədlərin rəqəmlərinin sayı əvvəlcədən proqrama məlum olmalı idi. İndi isə istənilən sayda rəqəmdən ibarət olan ədədlərin mərtəbə vahidlərini tapa bilərik.

Sərbəst işləmək üçün təqdim olunan çalışmalarda ədədin bütün mərtəbə vahidləri və onların sayını tapmaq tələb olunur. Nümunə çalışmada isə sadəcə ən böyük mərtəbə vahidini tapmaq tələb olunur. Müfaviq proqram həlli isə aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int i,x, max_mert;

    cout << "Her-hansi eded daxil edin: ";
    cin >> x;

    max_mert = 1;
```

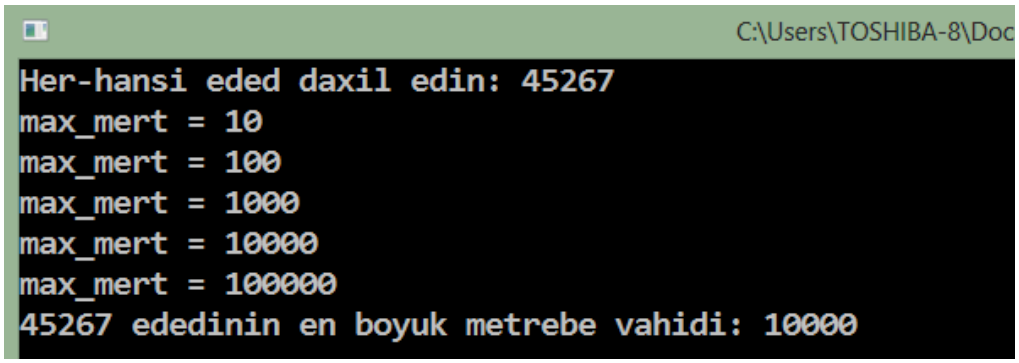
```
    while(max_mert <= x)
        max_mert *= 10;

    cout << x << " ededinin en boyuk metrebe vahidi: "
         << max_mert/10;

}
```

Nəticə



İzah: Mərtəbə havidlərinin sayını hesablamaq üçün max_mert adlı dəyişəndən istifadə edirik və bu dəyişənə başlanğıc olaraq 1 qiyməti mənimsədib while operatoru ilə dövrə başlayırıq. Şərt olaraq `max_mert <= x` vermişik. Bu o deməkdir ki nəqədər ki max_mert dəyişəni x-dən kiçik bərabərdir dövr davam edəcək. Dövr daxilində isə max_mert dəyişənin qiymətini hər dəfə 10-a vururuq.

Dövr bitdikdən sonra max_mert tələb olunandan bir vahid böyük mərtəbə vahidinə bərabər olur. Ona görə düzgün mərtəbə vahidini çap etmək üçün max_mert –i 10-a bölüb çap edirik. Bunun əyani şahidi olmaq üçün dövr daxilində max_mert dəyişəninin qiymətinin nə cür dəyişməsini nəzərdən keçirək.



Gördüyümüz kimi əgər dövr bitdikdən sonra max_mert –i 10-a bölmədən çap etsək 100000 nəticəsi verər.

5.8 break və continue

C++ dilində dövr operatorları, onların təkrarlanma sayının tənzimlənməsi məsələləri ilə tanış olduq. C++ dili dövrləri idarə etmək üçün **break** və **continue** adlı daha iki operator da təqdim edir. Gəlin bu operatorlarla ayrı-ayrı tanış olaq.

5.8.1 break operatoru

break operatoru dövrün dərhal başa çatdırılması və icranın dövr operatorundan sonra gələn operatora keçməsi üçün istifadə olunur. break operatorunun sintaksisi çox sadədir, break açar sözünü yazıb nöqtəvergül işarəsi qoyuruq, aşağıdakı kimi:

```
break ;
```

break operatoru dövrün içində olmalıdır və o yalnız yerləşdiyi dövrü başa çatdırır. Əgər break –in yerləşdiyi dövrün özü də başqa dövr içərisindədirsə break yuxarı qatdakı dövrə təsir etmir. Nümunələrlə fikrimiz daha yaxşı anlaşılar. Nümunələrə baxaq:

Nümunə:

```
#include <iostream>

using namespace std;


int main (){

    int i;

    for (i=0; i<10; i++){

      if (i == 6)
         break;

     cout << "i = " << i <<"\n";

    }

}
```

Nəticə:

```
i = 0
i = 1
i = 2
i = 3
```

```
i = 4
i = 5
```

İzahı:

Niyə bu cür nəticə aldıq? Axı sayğacın qiyməti 0-dan 10-a kimi dəyişməli idi, hər dəfə bir vahid artmaqla!? Səbəb dövr daxilində yazdığımız break operatorudur. Biz break operatorunun if şərt operatoru vastəsilə vermişik. Bu break –in yalnız tələb olunan şərt ödəndikdə icra olunması üçündür. Şərtimiz isə sayğacın 6 qiyməti almasıdır. Beləliklə sayğac 0-dan başlayaraq bir vahid olmaqla artır və 6 qiyməti alanda şərt ödəndiyinə görə break icra olunur. Nəticədə isə dövr sona çatdırılır. Əgər break –i if –siz versəydik, onda dövr ilə başlayan kimi bitərdi.

5.8.2 continue operatoru

continue operatoru dövrü break operatoru kimi başa çatdırmır, lakin özünündən sonra gələn operatorları icra etmədən dövrün növbəti addımına keçir. Sintaksisi break opratorunda olduğu kimi sadədir: continue açar sözünü yazıb nöqtəvergül işarəsi qoyuruq, aşağıdakı kimi:

```
continue ;
```

Nümunələrə baxaq:

Nümunə:

```
#include <iostream>

using namespace std;


int main (){

    int i;

    for (i=0; i<10; i++){

      if (i == 6)
         continue;

      cout << "i = " << i <<"\n";

    }

}
```

Nəticə:

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 7
i = 8
i = 9
```

İzahı: Sayğacın 6 qiyməti alanda continue icra olunur və özündən sonra gələn operatorlar buraxılaraq növbəti dövrə keçilir. Kodda continue-dən sonra ancaq bir operator var – cout ilə sayğacın qiyməti çap olunur. Əgər nəticəyə diqqətlə baxsaq orda sayğacın 6-ya bərabər olan qiyməti çap olunmayıb.

Çalışmalar.

Çalışma 1. 1-dən 100 –ə kimi ədədləri ekranda çap edən proqram tərtib edin.

Çalışma 2. 100 –dən 1-ə kimi ədədləri ekranda çap edən proqram tərtib edin.

Çalışma 3. 1-dən 100 –ə kimi olan cüt ədədləri ekranda çap edən proqram tərtib edin.

Çalışma 4. 1-dən 100 –ə kimi olan tək ədədlərin cəmini ekranda çap edən proqram tərtib edin.

Çalışma 5. İstifadəçinin daxil etdiyi ədədin rəqəmlərini axırdan əvvələ vergüllə çap edən proqram tərtib edin.

Çalışma 6. İstifadəçinin daxil etdiyi ədədi sözlərlə çap edən proqram tərtib edin.

Çalışma 7. İstifadəçinin daxil etdiyi ədədin bütün bölənlərini tapan proqram tərtib edin.

Çalışma 8. İstifadəçinin daxil etdiyi ədədin sadə ədəd olduğunu müəyyən edən proqram tərtib edin.

Çalışma 9. 1-dən 100 –ə kimi ədədlər arasında olan sadə ədələri çap edən proqram tərtib edin.

Çalışma10 dovr elə proqram qur ki istifadəçidən 3 ədəd daxil etməsini istəsin. Birinci ədədlə ikinci ədəd arsında qalan ədədlərin 3-cü ədəd hasilini ekranda çap etsin.

Çalışma 11. Sayğaclı dövr operatorundan istifadə etməklə 1-dən 100-ə kimi ədədlərin cəmini və ədədi ortasını hesablayan hesablayan proqram tərtib edin.

Çalışma 12. Şərtli dövr operatorundan istifadə etməklə 1-dən 100-ə kimi ədədlərin cəmini və ədədi ortasını hesablayan hesablayan proqram tərtib edin.

Çalışma 13. 1-dən 50-yə kimi ədədlərin harmonik cəmini hesablayan proqram tətrib edin.

Harmonik(n) = 1 + 1/2 + 1/3 + . . . + 1/n

Çalışma 14. 1-dən 20-yə kimi Fibonoççi ədədlərin çap edən proqram tətrib edin. Fibonoççi ədədləri çağıdakı kimi hesablanır.

```
F(n)=F(n−1)+F(n−2)
```

və F(1)=F(2)=1.

Çalışma 15. 1-dən 9-a kimi ədədlərin vurma cədvəlini ekranda çap edən proqram tərtib edin.

Çalışma 16. Elə proqram qur ki, istifadəçidən hər -hansı tam ədəd daxil etməsini istəsin və həmin ədəd üçün 1-dən -10-a kimi vurma cərvəlini çap eləsin. Yəni, tutaq ki, istifadəçi 4 ədədini daxil eləsə proqram aşağıdakı nümunı kimi nəticə çap eləməlidi:

Hər hansı eded daxil edin:
4
Sizin daxil etdiyiniz ədəd üçün 1-dən 10-a kimi vurma cedveli:

4*1 = 4
4*2 = 8
4*3 = 12
. . .

4*10 = 40

Çalışma 17. Çalışma 1-i elə dəyişin ki, istifadəçinin daxil elədiyi ədəd üçün 95-dən 20-yə kimi vurma cədvəlini çap eləsin.

Çalışma 18. Elə proqram qurun ki, istifadəçidən iki ədəd daxil etməsini istəsin. Birinci eded ikinciden kiçik olsun. Sonra proqram birinci əddəddən ikinci ədəd kimi olanları ekranda çap eləsin. Əgər istifadəçinin daxil elədiyi birinci ədəd ikincidən böyük olsa, və ya bərabər olsa, onda proqram istifadəçiyə bildirməlidir ki, siz səhv ədədlər daxil etdiniz. Aşağıdakı kimi:

[İcra1]

Zehmet olmasa iki eded daxil edin:
23
30
23-den 30-a kimi olan ededler:
23 24 25 26 27 28 29 30

[İcra2]

Zehmet olmasa iki eded daxil edin:
45
12
Sizin daxil etdiyiniz birinci eded ikinciden boyukdur.
Zehmet olmasa duzgun ededler daxil edin.

Çalışma 19. Elə proqram qurun ki, istifadəçidən iki ədəd daxil etməsini istəsin amma birinci ədəd ikincidən böyük olsun. Sonra birinci əddəddən ikinci ədəd kimi olanları ekranda çap eləsin. Əgər istifadəçinin daxil elədiyi birinci ədəd ikincidən kiçik olsa, və ya bərabər olsa, onda proqram istifadəçiyə bildirməlidir ki, siz səhv ədədlər daxil etdiniz. Aşağıdakı kimi:

[İcra1]

Zehmet olmasa iki eded daxil edin:
45
37
45-den 37-e kimi olan ededler:
45 44 43 42 41 40 39 38 37

[İcra2]

Zehmet olmasa iki eded daxil edin:
56
78
Sizin daxil etdiyiniz birinci eded ikinciden kicikdir.
Zehmet olmasa duzgun ededler daxil edin.

Çalışma 20. Elə proqram qurun ki, istifadəçidən iki ədəd daxil etməsini istəsin, fərqi yoxdu hansı böyük , hansı kiçik oldu. Sonra birinci əddəddən ikinci ədəd kimi olanları ekranda çap eləsin. Aşağıdakı kimi:

[İcra1]

Zehmet olmasa iki eded daxil edin:
34
41
34-den 41-e kimi olan ededler:
34 35 36 37 38 39 40 41

[İcra2]

Zehmet olmasa iki eded daxil edin:
23
16
23-den 16-a kimi olan ededler:
23 22 21 20 19 18 17 16

Qeyd: 5-ci çalışmada əvvəlcə ədədlərin hansının böyük kiçik olduğunu müəyyən etmək üçün if operatorundan istifadə etmək lazımdır. Daha sonra müvafiq (if -in daxilində) for-dan istifadə etmək lazımdır. Yəni yuxarıdakı iki çalışmanın birləşməsi, if ilə.

Çalışma . İstifadəçinin daxil etdiyi ədədin rəqəmlərinin sayını tapan proqram qurun.

Misal üçün istifadəçi 7345 ədədi daxil etsə proqram bu ədədi təşkil edən rəqəmlərin sayını yəni 4 çap etməlidir.

Calışma. İstifadəçinin daxil etdiyi ədədin rəqəməlrini söz ilə çap edən proqram qurun.

Yenə əgər istifadəçi 7345 ədədini daxil etsə proqram 7 min, 3 yüz, 4 10 və 5 kimi nəticə çap etməlidir.

§6 Switch operatoru

6.1 Switch operatoru

Əgər hər hansı parametri müxtəlif qiymətlərə görə müqaisə etmək tələb olunursa bu zaman çoxsaylı if/else operatorları yerinə daha münasib olan switch operatorundan istifadə olunur. switch operatorunun sintaksisi aşağıdakı kimidir:

```
switch(deyishen){

 case sabit1 :
      emeliyyat
      break;

 case sabit2 :
      emeliyyat
      break;

      ...

default:
       emeliyat
}
```

switch operatorunu tərtib etmək üçün aşağıdakı qaydalardan istifadə edirik:

1) Əvvəlcə switch açar sözünü yazıb mötərizə daxilində dəyişəni yazırıq, daha sonra fiqurlu { } mötərizələri içərisində switch operatorunun case –lərini, başqa sözlə hal-larını tərtib edirik.

```
switch(deyishen){

}
```

2) Hər bir case –dən sonra mütləq hər-hansı ədəd və ya simvol yazıb qoşanöqtə qoymalıyıq.

```
switch ( x ) {
 case 15:

}
```

3) switch operatoru verilmiş dəyişənin qiymətini case –lərdə göstərilənlərlə müqaisə edir, hansı düz gəlsə həmin case –in altında yazılan operatorları yerinə yetirir. Hər-bir case –in altında istənilən qədər istənilən operator yaza bilərik, hətta seçim operatoru da. Şərt və dövr operatorlarında olduğu kimi əməliyyatları bloka almamalıyıq.

```
switch ( x ) {

 case 15:
    cout<< "Salam dunya \n";
    y = 56 + z;

}
```

Baxdığımız nümunədə case -in qiyməti 3 verilib. Əgər x dəyişəninin qiyməti 3-ə bərabər olsa onda bu case -in altında yazdığımız bütün operatorlar icra olunacaq.

4) İstədiyimiz qədər case tərtib edə bilərik, amma hər birinin qiymətləri müxtəlif olmalıdır.

```
switch ( x ) {

  case 3:
    cout << "Salam dunya \n";
    y = 56 + z;

  case 4:
    cout << " Proqramlashdirma \n";
    y = 99 + z*z;

  case 5:
    cout << " Informatika \n";
    y = 100 + z*(q + 5);

}
```

5) Eyni əməliyyatların icra olunması üçün bir neçə case təyin edə bilərik, Bu zaman dəyişənin qiyməti həmin case-lərin istənilən birinin qiymətinə uyğun gəlsə müvafiq əməliyyat icra olunacaq.

```
switch ( x ) {

    case 1:
      cout << "x = 1 \n";

   case 2:
    case 3:
      cout << "x = 2 ve ya 3\n";

    case 4:
    case 5:
    case 6:
      cout << "x = 4 , 5 ve ya 6\n";

}
```

6) Hər hansı case ödənsə və onun əməliyyatları icra olunsa, yenə də seçim operatoru digər halları yoxlamaqda davam edəcək. Bunun qarşısını almaq üçün hər bir case -in əməliyyatlarının sonunda **break;** operatoru yazmalıyıq.

```
switch ( x ) {

  case 1:
     cout << "x = 1 \n";
    break;

  case 2:
```

```
        case 3:
            cout << "x = 2 ve ya 3\n";
             break;

        case 4:
        case 5:
        case 6:
          cout << "x = 4 , 5 ve ya 6\n";
          break;

    }
```

7) Heç bir hal ödənmədiyi zaman hansısa əməliyyat icra etmək istəyiriksə **default** halı təyin edirik. Bunun üçün **default** yazıb qoşanöqtə qoyuruq. **default** halından istifadə etmək zəruri deyil.

```
    switch ( x ) {

      case 1:
          cout << "x = 1 \n";
         break;

      case 2:
      case 3:
          cout << "x = 2 ve ya 3\n";
           break;

      case 4:
      case 5:
      case 6:
        cout << "x = 4 , 5 ve ya 6\n";
        break;

      default:
         cout << x ferqlidir 1,2,3,4,5,6 \n";

    }
```

Seçim operatoruna aid nümunə proqramlara baxaq:

Nümunə . Aşağıdakı kod istifadəçinin imtahandan aldığı qiymətə uyğun nəticəni çap edir.

```
#include <iostream>

using namespace std;

int main(){
```

```
    int x;

   cout << "Imtahandan aldiginiz qiymeti daxil edin\n";
   cin >> x;

    switch(x){

    case 5:
        cout << "Siz ela qiymet aldiniz, tebrikler \n";
        break;

    case 4:
        cout << "Siz yaxshi netice gosterdiniz \n";
        break;

    case 3:
        cout << "Siz kafi qiymet aldiniz \n";
        cout << "biraz da oz uzerinizde calishin \n";
        break;

    case 2:
        cout << "Siz imtahandan kesildiniz \n";
        cout << "oz uzerinizde ciddi calishmalisiniz \n";
        break;
    default:
      cout << "Duzgun qiymet daxil etmediniz \n";
      cout << "2,3,4 ve ya 5 qiymetlerinden birini secin \n";
    }

}
```

Nəticə:

```
Imtahandan aldiginiz qiymeti daxil edin
3
Siz kafi qiymet aldiniz
biraz da oz uzerinizde calishin
```

Nümunə . Aşağıdakı kod istifadəçinin daxil etdiyi simvola uyğun hesab əməliyyatı aparır. Bunu sadə kalkulyator proqramı da hesab etmək olar.

```
#include <iostream>

using namespace std;

int main (){

    int x, y;
    char emel;

    cout << "Her-hansi iki eded daxil edin: \n";
```

```
    cin >> x >> y;

     << "Zehmet olmasa emeli secin (+, - , *, /, %): \n";
    cin >> emel;

     switch(emel){
         case '+' :
            cout << x << " + " << y << " = " << x + y << "\n";
             break;
         case '-' :
             cout << x << " - " << y << " = " << x - y << "\n";
             break;
         case '*' :
            cout << x << " * " << y << " = " << x * y << "\n";
             break;
         case '/' :
             cout << x << " / " << y << " = " << x / y << "\n";
             break;
         case '%' :
            cout << x << " % " << y << " = " << x % y << "\n";
             break;
     default:
       << "Zehmet olmasa duzgun emeliyyat daxil edin.\n";
       << "Duzgun emeliyyatlar: +, -, *, /, %   \n";
     }
}
```

Nəticə:

```
Her-hansi iki eded daxil edin:
3 4
Zehmet olmasa emeli daxil edin (+, - , *, /, %):
+
3 + 4 = 7
```

Çalışma 1. Elə proqram tərtib edin ki, istifadəçidən ayı ədədlə daxil etməsini istəsin və ayın adını çap etsin. (1 – yanvar, ...) İstifadəçi səhv ay daxil etdikdə 13, 0 v.s. bu barədə məlumat versin.

Çalışma 2. Elə proqram tərtib edin ki, istifadəçidən həftənin gününü ədədlə daxil etməsini istəsin və günün adını çap etsin. (1 – bazar ertəsi, ...) İstifadəçi səhv gün daxil etdikdə 8, 9 v.s. bu barədə məlumat versin.

Çalışma 3. Ayın birinin həftənin 1-ci gününə düşdüyünü qəbul etməklə elə proqram tərtib edin ki, ayın gününü daxil etməklə həmin günün ayın neçənci həftəsi olduğunu və həftənin neçənci günü olduğunu müəyyən etsin. Həftənin gününü sözlə çap etsin.

Nümunə nəticə.

```
Ayin gununu daxil edin
27
Ayin 4 -cu heftesi, shenbe
```

Çalışma 4. Yuxarıdakı proqramı elə dəyişin ki, istifadəçinin təkrar –təkrar qiymətlər daxil etməsinə imkan versin. İstifadəçi 0 qiyməti daxil etdikdə proqram başa çatsın.

§7 Cərgələr

7.1 Birölçülü Cərgələr

İndiyə kimi biz proqramımızda hər hansı məlaumatı yerləşdirmək üçün ayrı dəyişəndən istifadə edirdik. Misal üçün 4 ədəd yadda saxlamaq üçün `int` tipindən aşağıdakı kimi 4 müxtəlif dəyişən elan etməliyik.

```
int a, b, c, d;
```

Bəs birdən 1000 dənə ədədi yadda saxlamaq tələb olunsa necə? Bu zaman hər bir ədədi yadda saxlamaq üçün əlavə dəyişən elan etmək çox zaman alar. Proqramlaşdırmada isə bəzən yüz min hətta milyonlarla məlumatlarla işləmək tələb olunur. Bu zaman ***cərgə*** adlandırılan dəyişənlər kolleksiyasından istifadə etmək məqsədəuyğun olar.

> **Qeyd:** Cərgələrə digər dərsliklərdə array və ya massiv deyirlər. Məncə cərgə sözü nisbətən daha başadüşüləndir, nəinki massiv.

Proqram tərəfinə keçməzdən qabaq gəlin əvvəlcə fikrimizdə cərgə barəsində daha aydın təsəvvür yaratmaq üçün bəzi praktik nümunələrə müraciət edək. Əgər dəyişənə ev kimi baxsaq, onda cərgəyə bütün evləri bir – birinə bitişik olan, bir xətt boyunca düzülmüz məhəllə kimi baxmaq olar. Əgər dəyişəni maşın kimi təsəvvür eləsək, onda cərgəni maşınlardan ibarət karvan kimi təsəvvür edə bilərik. Bu cür nümunələr çox gətirə bilərik, lakin gəlin keçək proqram tərəfinə. Proqramlaşdırmada cərgə elən etməyin bəzi tələbləri var. Bunlardan ən birincisi odur ki, cərgəni təşkil edən dəyişənlər mütləq ***eyni tipdən*** olmalıdır. Cərgələrə dəyişənlərə verdiyimiz kimi adlar verə bilərik.

Bütün bunları bildikdən sonra gəlin cərgələrin elan olunma sintaksisi ilə tanış olaq. C++ dilində hər-hansı tipdən cərgə elan etmək istəsək aşağıdakı kimi yazmalıyıq:

```
tip cərgənin_adı [elementlərin_sayı];
```

Əvvəlcə cərgənin tipini göstəririk, daha sonra cərgənin adını yazırıq. Daha sonra kvadrat mötərizələr içində cərgənin elementlərinin sayını göstəririk. Qeyd eliyim ki, bu cərgənin tipi və elementlərinin sayını elandan sonra dəyişmək olmaz, bunlar yalnız bir dəfə cərgə elan olunan zaman təyin olunur və proqramın sonuna kimi dəyişmir.

Yuxarıdakı nümunəyə qayıtsaq, yazdığımız sintaksisə görə 4 ədədi yadda saxlamaq üçün `int` tipli 4 elementdən ibarət x adlı cərgəni aşağıdakı kimi elan edə bilərik:

```
int x [4];
```

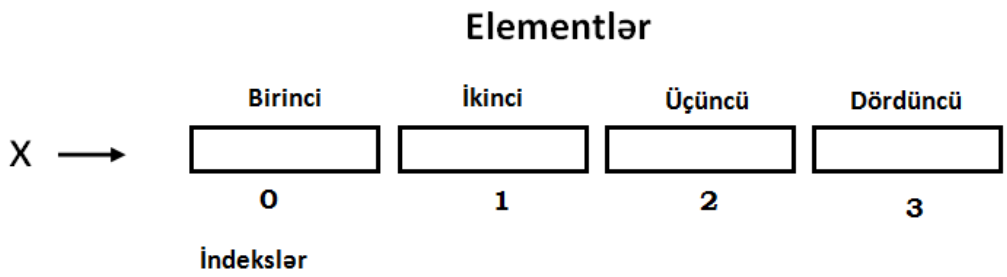
Bu zaman yaddaşda `int` tipli `4` dəyişən üçün yer ayrılacaq və bu dəyişənlər yaddaşda bir –birinin ardınca yanaşı düzüləcəklər, aşağıdakı kimi:

Cərgələrə dəyişənlərə verdiyimiz kimi istənilən ad verə bilərik, lakin çalışmalıyıq dəyişənlərdə olduğu kimi, cərgələrə də verdiyimiz ad saxlanılan məlumata uyğun olsun. Misal üçün **100** nəfər işçinin aylıq məvacibini yadda saxlamaq üçün elan etdiyimiz cərgəyə, **x**, **y**, **ss** v.s. kimi adlar verə bilərik, amma məncə **mevacib** daha münasib ad olar, aşağıdakı kimi:

```
double mevacib [100];
```

7.1.2 Cərgənin Elementlərinin indeksi

Cərgəni elan edən zaman elemenlərinin sayın kvadrat mötərizənin içərisində yazdığımız ədəd ilə göstəririk. Cərgənin elementləri yaddaşda bir düz xətt boyunca ardıcıl düzülür və nömrələnirlər. İlk elementin nömrəsi olur 0, növbəti elementin nömrəsi 1, 2, 3, v.s. Proqramlaşdırmada **nömrə** sözü əvəzinə **indeks** sözündən istifadəyə üstünlük verilir.



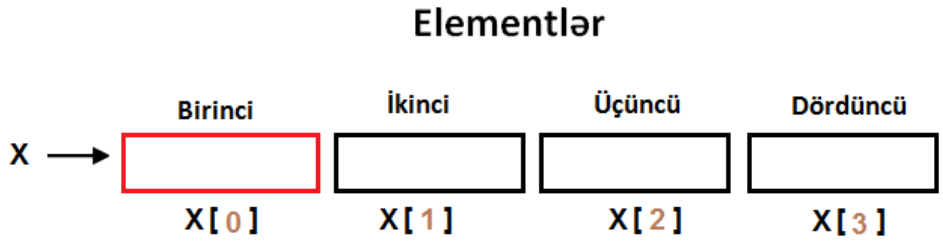
Şəkildən gördüyümüz kimi indekslər 0-dan başladığına görə cərgənin ilk elementinin indeksi 0 , ikinci elemeninin indeksi 1, sonuncu elemetinin indeksi isə 3 olur. Ümumiyyətlə bu qanunauyğunluğa əsasən cərgənin sonuncu elementinin indeksi cərgənin elementlərinin sayından bir vahid az olur.

7.1.3 Cərgələrin Elementlərinə Müraciət

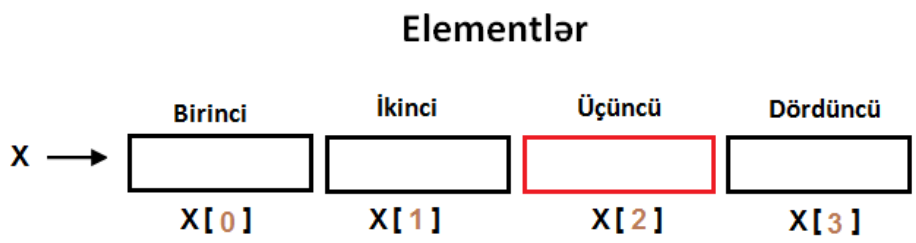
Cəranı elan etdikdən sonra onun elementlərinə müraciət edə bilərik. Bunun üçün cərgənin **adından** və müraciət etmək istədiyimiz elementin **indeksindən** istifadə olunur :

```
cərgənin_adı [elementin_indeksi] ;
```

Misal üçün yuxarıda elan elədiyimiz x cərgəsinin birinci elementinə müraciət etmək üçün `x[0]`,



üçüncü elementə müraciət etmək üçün `x[2]` yazmalıyıq:



Cərgənin elementlərindən müvafiq tipdən olan dəyişənlər kimi istifadə edə bilərik. Yəni onlara həm qiymətlər mənimsədə, həm cin operatoru ilə klaviaturadan daxil olunan qiyməti onlarda yadda saxlaya, həm müxtəlif hesablamalarda onlardan istifadə edə, cout ilə ekranda çap edə bilərik.

Nümunə

Misal üçün yuxarıda elan etdiyimiz x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədən kod aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

int main (){

    // 4 elementi olan x cergesi elan edek
    int x[4];



    // x cergesinin elementlerine qiymetler menimsedek
    x[0] = 12;
    x[1] = 3;
    x[2] = 45;
    x[3] = 648;

}
```

Nümunə

Proqram kodunda bir qədər dəyişilik edək, cərgənin elementləri cəmini hesablayıb çap edək. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  // 4 elementi olan x cergesi elan edek
  int x[4], cem;

  // x cergesinin elementlerine qiymetler menimsedek
  x[0] = 12;
  x[1] = 3;
  x[2] = 45;
  x[3] = 648;

  //elementlerin cemini hesablayaq
  cem = x[0] + x[1] + x[2] + x[3];

  cout << "elementlerin cemi = " << cem ;
```

```
}
```

Nəticə:

```
elementlerin cemi = 708
```

Nümunə 2: Indi isə elə proqram quraq ki, double tipdən 5 elementli y cərgəsi elan etsin. Onun ilk 4 elementinə istifadəçinin daxil etdiyi qiymətləri mənimsədək, sonuncu elementə isə digər elementlərin cəmini mənimnsədək. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  double y[5];

  cout << " y cegesinin birinci elementini daxil edin\n";
  cin >> y[0];

  cout << " y cegesinin ikinci elementini daxil edin\n";
  cin >> y[1];

  cout << " y cegesinin ucuncu elementini daxil edin\n";
  cin >> y[2];

  cout << " y cegesinin dorduncu elementini daxil edin\n";
  cin >> y[3];

  y[4] = y[0] + y[1] + y[2] + y[3];

  cout << "y cergesinin ilk 4 elementinin cemi = " << y[4];

}
```

Nəticə:

```
y cegesinin birinci elementini daxil edin
34
y cegesinin ikinci elementini daxil edin
12.6
y cegesinin ucuncu elementini daxil edin
56.9
y cegesinin dorduncu elementini daxil edin
76.89
y cergesinin ilk 4 elementinin cemi = 180.39
```

Baxdığımız nümunələrdə cərgə elan etmək, onun elementlərinə müraciət etməklə tanış olduq. Bütün bunlar əlbəttdə ki çox yaxşıdı, lakin diqqət yetirsəniz yuxarıdakı nümunələrdə hər bir elementə qiymət mənimdətmək üçün ayrı kod yazmalı olduq. Bəs əgər cərgənin elementlərinin sayı 1000 olsa necə?

Cərgələrlə işləyərkən dövrlərdən istifadə etmək çox əlverişlidir. Belə ki, dövrün sayğacın cərgənin indeksi kimi, cərgənin elementləri sayını isə sayğacın son qiyməti isə kimi verməklə dövrü olaraq cərgənin bütün elementlərinə müraciət etmək olar. Bütün bu dediklərimizi nümunə əsasında izah edək.

Nümunə: 5 elementli cərgənin elementlərinə müxtəlif qiymətlər mənimsədən proqram tərtib edin. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  double y[5];
  int i;

  cout << "y cegesinin elementlerini daxil edin\n";

  for (i=0; i<5; ++i)
      cin >>y[i];

  cout << "y cergesinin elementleri: ";

  for (i=0; i<5; ++i)
      cout << y[i] << " ";

}
```

Nəticə:

```
y cegesinin elementlerini daxil edin
1
23
4
5
56
y cergesinin elementleri: 1 23 4 5 56
```

İzahı: Kodda diqqət yetirsək indeks olaraq dövrün sayğacını vermişik və beləliklə sayğac 0-dan , cərgənin elementlərinin sayına kimi dəyişdikdə cərgənin bütün elementlərinə müraciət etmək mümükün olur. Növbəti nümunəyə baxaq.

7.1.4 Verilmiş indeksli elementin çap olunması

Çalışma : Elə proqram qurun ki, cərgənin verilmiş indeksli elementini çap etsin.

Həlli: Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  int x[7], i, k;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
  for (i=0; i<7; ++i)
      cin  >> x[i];

  //indeksi daxil edek
  cout << "indeksi daxil edin \n";
  cin  >> k;

  //verilmish indeksli elementi cap edek
  cout << "x[" << k << "] = " << x[k];

}
```

Nəticə:

```
x -in elementlerini daxil edin
2 34 5 21 56 8 90
indeksi daxil edin
4
x[4] = 56
```

7.1.5 Verilmiş elementin indeksinin tapılması

Çalışma: Elə proqram qurun ki, cərgənin verilmiş elementinin indeksini çap etsin.

```
#include <iostream>

using namespace std;

int main (){

  int x[7], y, i, k;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
```

```
    for (i=0; i<7; ++i)
        cin >> x[i];

    //elementi daxil edek
    cout << "elementi daxil edin \n";
    cin  >> y;

    //k-ya -1 menimsedek
    k = -1;

    for (i=0; i<7; ++i)
        if (x[i] == y){
            k = i;
            break;
        }

    //eger verilmish element cergede varsa onda
    // onun indeksi k-ya menimsedilib, dememli k -1 den ferqlidir
    if (k != -1)
        cout << y << " elementinin indeksi = " << k ;
    else
        cout <<"x cergesinde " << y << " elementi yoxdur \n";

}
```

İcra 1: `edin`

```
x -in elementlerini daxil edin
34 56 7 0 89 5 12
elementi daxil edin
7
7 elementinin indeksi = 2
```

İcra 2: `edin`

```
x -in elementlerini daxil edin
23 4 5 39 0 -23 7
elementi daxil edin
44
x cergesinde 44 elementi yoxdur
```

İzahı : Əvvəlcə cərgənin elementlərini və indeksi tələb olunan elementi daxil edirik.

Daha sonra indeksi yadda saxlamaq üçün istifadə etdiyimiz k dəyişəninə -1 qiyməti mənimsədirik.

```
k = -1;
```

Daha sonra dörv operatoru ilə cərgənin bütün elementlərini yoxlayırıq.

```
    for (i=0; i<7; ++i)
```

Əgər cərgənin hər-hansı elementi tələb olunan elementə bərabər olarsa onun indeksini k dəyişəninə yazırıq və dövrü bitiririk. Axtardığımız elementi tapdığımıza görə cərgənin digər elementlərini yoxlamağa ehtiyac yoxdur.

```
if (x[i] == y){
    k = i;
    break;
}
```

Sonda k-nın qiymətini yoxlayırıq. Əgər -1 deyilsə onda deməli axtarılan element cərgədə var və onun indeksini çap edirik. Əks halda istifadəçiyə elementin olmaması barədə məlumat veririk.

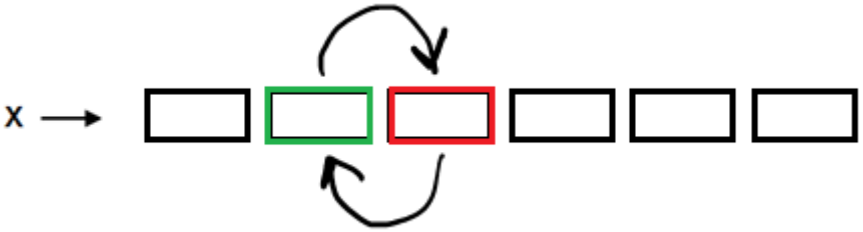
7.1.6 Elementlərin yerlərinin dəyişdirilməsi

Cərgənin elementlərinin yerlərini dəyişmək bizə tez-tez lazım ola bilər. Misal üçün irəlidə örgənəcəyimiz müxtəlif düzüm alqoritmlərində v.s. digər məqsədlərə görə. Gəlin bu işi örgənək. Əvvəlcə görməli olduğumuz işin qrafik təsviri ilə tanış olaq.

Tutaq ki, hər-hansı 6 elementli x cərgəsi verilib.

Bizdən onun 2-ci elementi ilə 3-cü elementlərinin yerlərini dəyişmək tələb olunur.

Yəni ikinci elementi keçirdib qoymalıyıq üçüncü elementin yerinə, üçüncünü isə ikincinin yerinə.



Nəticədə cərgə aşağıdakı şəklə düşməlidir

Qrafik izahdan sonra məncə məslənin proqram tərəfini başa düşmək daha asan olar. Əsas məsələ aydındır, qalır onu necə eləmək. Cərgənin hər iki elementinə müraciət edə bilirik müvafiq indeksləri vastəsilə, `x[1]` və `x[2]` . Məsələ iki dəyişənin qiymətlərinin dəyişdirilməsi qədər sadələşir. Əgər Dəyişənlər mövzusudan xatırlayırsınızsa bunun üçün biz hansısa başqa bir müvəqqəti dəyişəndən istifadə etmişdik. Burada da eyni qaydadan istifadə edəcəyik. Əgər müvəqqəti dəyişən olaraq hər-hansı y dəyişənindən istifadə etsək, cərgənin ikinci və üçüncü elementlərinin yerlərini dəyişmək üçün kod aşağıdakı kimi olar:

```
y = x[1];
x[1] = x[2];
x[2] = y;
```

Gəlin bu kodun hər-bir sətrini ayrılıqda nəzərdən keçirək və izahını verək. Əvvəlcə elementlərdən birini (fərqi yoxdu) müvəqqəti y dəyişənində yadda saxlayırıq.

```
y = x[1];
```

Bilirik ki, mənimsətmə operatoru dəyişənin əvvəlki qiymətini silir, yerinə yenisini yazır. Daha sonra birinci elementə ikinci elementin qiymətini mənimsədirik:

```
x[1] = x[2];
```

Bu zaman birinci elementin əvvəlki qiyməti silinir və yerinə ikinci elementin qiyməti yazılmış olur. Biz artıq birinci elementi ikinci ilə əvəz etdik. Qalır ikincinin yerinə birincinin qiymətini yazmaq. Birinci elementin qiymətini isə y –də yadda saxladığımıza görə yaza bilərik:

```
x[2] = y;
```

Beləliklə cərgənin iki elementinin qiymətlərini əvəzləmiş olduq. Növbəti örgənəcəyimiz alqoritmdə bu bizə lazım olacaq. Hələlik isə bu alqoritmə aid nümunə proqram ilə tanış olaq.

Nümunə. Cərgənin ilk elementi ilə son elementinin yelərini dəyişən proqram tərtib edin. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  int x[5], y, i;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
  for (i=0; i<5; ++i)
  cin  >> x[i];

  //x -in elementlerini cap edek
  cout << "\nx-in elementleri evvel\n";
  for (i=0; i<5; ++i)
      cout << x[i] << " ";

  //ilk elemenlt ile son elementin qiymetlerini
  //evezleyek
  y = x[0];
  x[0] = x[4];
  x[4] = y;

  //x -in elementlerini cap edek
  cout << "\n\nx-in elementleri sonra\n";
  for (i=0; i<5; ++i)
      cout << x[i] << " ";

}
```

Nəticə

```
x -in elementlerini daxil edin
12 3 45 6 98

x-in elementleri evvel
12 3 45 6 98

x-in elementleri sonra
98 3 45 6 12
```

Çalışma. Cərgənin elementlərini 1 vahid “sağa” sürüşdürən proqram tərtib edin. “Boş” qalan yerlərə 0 qiyməti yazın. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

  int x[10], i;
```

```
    cout<<"10 eded daxil edin\n";

    for (i=0; i<10; ++i)
        cin >> x[i];

    //cergenin elementlerini saga surushdurek
    for (i=9; i>0; --i)
        x[i] = x[i-1];

    x[0] = 0;

    //cereni cap edek
    for (i=0; i<10; ++i)
        cout << x[i] << " ";


}
```

Nəticə:

```
10 eded daxil edin
12 34 5 43 6 5 90 4 34 32
0 12 34 5 43 6 5 90 4 34
```

İzahı: Cərgənin qiymətlərini daxil etdikdən sonra dövrü olaraq cərgənin ən son elementindən başlayaraq ilk elementinə kimi hərəkət eliyirik (iterasiya).

```
for (i=9; i>0; --i)
```

Bu iterasiya zamanı cərgənin hər bir elementinə özündən əvvəlki elementi mənimsədirik.

```
x[i] = x[i-1];
```

Nəticədə cərgənin bütün elementləri cərgrə boyu bir indeks sağa sürüşmüş olur. Sonda isə cərgənin ilk elementinə 0 qiyməti mənimsədirik.

7.1.7 Ən böyük elementin tapılması

Cərgələrlə bağlı sadə alqoritmləri tədricən örgənirik. Növbəti baxacağımız məsələ cərgənin ən böyük elementinin tapılması ilə bağlıdır. Tutaq ki bizə aşağıdakı ədədlər ardıcıllığı verilib:

```
2, 34, 67, 892, 0, 31
```

Əgər bizdən tələb olunsa bu ədədlərin arasından ən böyüyünü tapaq, bunu asanlıqla edə bilərik: əlbəttə ki 892. Buna görə cərgələrdən istifadə eləyib, proqram qurmaq nəyə lazımdır, üstəlik hansısa alqorimtlərdən istifadə etmək?

İnsan beyni ədədlər arasından ən böyüyünü tapmaq üçün özünə məxsus olan alqorimtdən istifadə edir. Lakin biz bu alqoritmdən faydalana bilmərik. Üstəgəl ədədlərin sayını xeyli artırsaq insan beyninin alqoritmi ən böyük ədədi tapmaqda çox çətinlik çəkər və buna çox vaxt tələb olunar. Kompüter proqramı isə əksinə, 100 000 – lərlə ədəd arasından ən böyüyünü asanlıqla, çox kiçik zaman müddətinə tapar. Bunun üçün müxtəlif alqoritmlər icad olunub, onlardan sadə biri aşağıdakı kimidir:

Tutak ki, elementlərinin sayı **k** olan cərgə verilib və onun ən böyük elementini tapmaq tələb olunur. Bunun üçün aşağıdakı kimi hərəkət etməliyik:

1) Əvvəlcə ən böyük ədədi yadda saxlamaq üçün əlavə bir dəyişən elan etməliyik. Tutaq ki, max adlı dəyişən elan elədik. Adın heç bir önəmi yoxdur.

2) Ən böyük elementi yadda saxlamaq üçün dəyişən elan etdikdən sonra həmin dəyişəni cərgənin ilk elementinə mənimsətməliyik.

   ```
   max  = cerge [ 0 ]  ;
   ```

   Beləliklə biz ən böyük olub-olmamasından asılı olmayaraq cərgənin ilk elementini max dəyişəninə yerləşdirmiş olduq.

3) Dövr operatoru ilə cərgənin ikinci elementindən başlayaraq sonuncu elementinə qədər bir-bir bütün elementləri nəzərdən keçiririk. (Birinci elementi yoxlamağa ehtiyac yoxdur, çünki max artıq birinci elementlə mənimsədilib.). Əgər baxdığımız hansısa element max-dan böyük olarsa onda həmin elementin qiymətini max-da yadda saxlayırıq. Bu qayda ilə dövr operatoru vastəsilə cərgənini elementləri boyu irəlilədikcə qarşımıza çıxan ən böyük ədəd həmişə max dəyişənində yadda saxlanmış olacaq. Dövrün sonunda isə max özündə cərgənin bütün elementləri içərisində ən böyüyünü yadda saxlamış olacaq.

Gəlin təklif etdiyimiz bu alqoritmin proqram versiyası ilə tanış olaq:

```
#include <iostream>

using namespace std;

int main (){

  int x[10], i, max;

  cout << "x cegesinin elementlerini daxil edin\n";
```

```
    //x cergesinin elementlerini daxil ededk
    for (i=0; i<10; ++i)
        cin >>x[i];

    //ilk elementi max -a yazaq
    max = x[0];

    //bir-bir yerde qalan elementlere baxaq
    for (i=1; i<10; ++i){

        //eger baxdigimiz element max -dan boyukdurse
        //onu max-a yerleshdirek
        if ( x[i] > max )
            max = x[i];
    }

    //max -i cap edek
    cout << "x cergesinin en boyuk elementi: "
              << max ;

}
```

Nəticə:

```
x cegesinin elementlerini daxil edin
12 34 567 8 9 0 432 5 322 98
x cergesinin en boyuk elementi: 567
```

7.1.8 Ən böyük elementinin indeksinin tapılması

Cərgənin ən böyük elementini tapa bilirik. İndi isə onun indeksini tapmaq tələb olunur. Burada elə də çətin bir problem yoxdur, əgər ən böyük elementi tapmaq alqoritmini başa düşmüşüksə. Sadəcə ən böyük elementi yadda saxlamaq üçün istifadə etdiyimiz dəyişəndən əlavə digər bir dəyişən də götürəcəyik, hansında ki ənböyük elemetin indeksini yadda saxlayacayıq. Koda və nümunə nəticəyə nəzər salaq. Daha sonra izahla tanış olarıq.

```
#include <iostream>

using namespace std;

int main (){

  int x[5], y, i, k;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
  for (i=0; i<5; ++i)
  cin  >> x[i];
```

```
    y = x[0];
    k = 0;

    for (i=1; i<5; ++i){
        if (x[i] > y){
            y = x[i];
            k = i;
        }
    }

    //x -in en boyuk elementi
    cout << "x -in en boyuk elementi: "<< y  << "\n"
         << "en boyuk elementin indeksi: "   << k ;
}
```

Nəticə:

```
x -in elementlerini daxil edin
23 45 6 789 0 11
x -in en boyuk elementi: 789
en boyuk elementin indeksi: 3
```

İzahı: Məncə burada əlavə izaha elə də ehtiyac yoxdur. Ən böyük elementin indeksini yadda saxlamaq üçün k dəyişənindən istifadə edirik və başlanğıcda cərgənin ilk elementini ən böyük kimi yadda saxlayanda(y –də), onun indeksini də (0) k-da yadda saxlayırıq. Daha sonra alqoritmimizə uyğun olaraq ən böyük elementi yadda saxladığımız y dəyişənin qiyməti hər dəfə yeniləndikdə, k-ya da müvafiq yeni ən böyük elementin indeksini yazırıq. Sonda y cərgənin ən böyük elementini, k isə onun indeksini özündə saxlamış olur. Əgər bu alqoritmi başa düşmüşüksə, onda növbəti daha mürəkkəb alqoritmə keçə bilərik.

7.1.9 Cərgənin ən böyük elementinin sona sürüşdürülməsi

Cərgənin ən böyük ekemetini və onun indeksini tapa bilirik. İndi isə məqsədimiz cərgənin ən böyük elementini cərgənin sonuna sürüşdürməkdir. Bu alqoritmdən növbəti tapşırıqda – cərgənin elementlərini artan sıra ilə düzmək məsələsinin həllində istifadə edəcəyik. Həll yolumuz aşağıdakı kimidir: Əvvəlcə cərgənin ən böyük elementini və onun indeksini tapırıq. Əgər bu cərgənin sonuncu elementidirsə onda məsələ həll olunub, heçnə etmirik. Əks halda , yəni ən böyük element cərgənin aralıq elementlərindən biridirsə, onda onunla, cərgənin sonuncu elementinin yerlərini dəyişirik. Nəticədə cərgənin ən böyük elementini cərgənin sonuna köçürmüş olarıq. Cərgənin iki elementinin də yerini dəyişməyi bildiyimizə görə bu məsələni həll etmək üçün heç bir sual qalmır. Kod və nümunə nətiədən sonra izahla tanış olarıq:

```
#include <iostream>
```

```
using namespace std;

int main (){

  int x[5], y, z, i, k;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
  for (i=0; i<5; ++i)
      cin  >> x[i];

  //en boyuk elementi ve onun indeksini tapaq
  y = x[0];
  k = 0;

  for (i=1; i<5; ++i){
      if (x[i] > y){
          y = x[i];
          k = i;
      }
  }

  //eger en boyuk element sonuncu deyilse
  //onu sona kocurek
  if ( k != 4){
      z = x[k];
      x[k] = x[4];
      x[4] = z;
  }

  //cergeni cap edek
  cout << "x -in yeni duzumu\n";
  for (i=0; i<5; ++i)
      cout  << x[i] << " ";
}
```

Nəticə

```
x -in elementlerini daxil edin
2 345 67 0 98
x -in yeni duzumu
2 98 67 0 345
```

İzahı: Cərgənin ən böyük elementini və onun indeksini tapırıq. Ən böyük elementin indeksini sonuncu elementin indeksi ilə müqaisə edirik. Əgər fərqlidirsə, sonuncu elementlə ən böyük elementin yerini dəyişirik. Bu alqoritmdən növbəti daha çətin sayılan alqoritmdə istifadə edəcəyik.

7.1.10 Elementlərin artan sırada düzülməsi

Buna ingilis dilində **sort** –ing deyirlər. Düzmə proqramlaşdırmanın çox tez-tez tələb olunan əməliyyatlarındandır. Bunun üçün indiyə kimi müxtəlif optimal alqoritmlər tərtib olunub və hal-hazırda da bu işin vacibliyini nəzərə alaraq yeni optimal alqoritmlər üzərində tədqiqatlar aparılır. Bir neçə məşhur alqoritmlər isə aşağıdakılardır: Qabarcıq düzüm(buble sort), seçmə düzüm(selection sort) v.s.

Bunlardan ən sadəsi və primitivi qabarcıq düzüm alqoritmidir. Gəlin onunla tanış olaq.

7.1.11 Qabarcıq düzüm alqoritmi

Cərgənin elementlərini artan sırada düzmək, hətta qabarcıq alqoritmi kimi nisbətən sadə alqoritmlə də olsa belə, ilk dəfə örgənən üçün biraz çətin gəlir. Ona görə bu alqoritmin izahı zamanı qrafik təsvirlərin də köməyindən istifadə edəcəyik. Tutak ki, aşağıdakı kimi 5 elementdən ibarət x cərgəsi verilib.

Məqsədimiz bu cərgənin elementlərini aşağıdakı kimi artan sıra ilə düzməkdir:

Nizam ilə düzülmüş cərgələrdə axtarış aparmaq çox tez başa gəldiyindən cərgənin nizamlanması proqramlaşdırmanın əsas alqoritmlərindən biridir. İndi isə gəlin görək biz bu işin öhdəsindən nə cür gələcəyik. Əgər yuxarıdakı çalışmadakı cərgənin ən böyük elementinin sona sürüşdürülməsini başa düşmüşüksə onda bu alqoritmi də başa düşmək çətin olmamalıdır.

Baxdığımız qabarcıq alqortiminə görə düzümü aşağıdakı kimi aparacayıq. Əvvəlcə cərgənin ən böyük elementi müəyyən edilir və sona sürüşdürülür. Bu zaman cərgə aşağıdakı şəklə düşər:

Daha sonra isə cərgənin ***"yerdə qalan elementlər içərisində"*** ən böyüyü tapılaraq bundan əvvəl tapdığımız ən böyük elementin yanına sürüşdürülür.

Bu prosesi dövrü olaraq bütün digər yerdə qalan elementlər üçün də təkrar etməliyik. Növbəti element üçün yerdəyişməyə ehtiyac yoxdur. Çünki üçüncü element artıq öz yerindədir:

Qalır son iki elementə bu qaydanı tətbiq etmək:

Vəssalam! Cərrgənin elementləri artan sırada düzülüb. Kod və nümunə nəticələrlə tanış olaq, daha sonra izahı davam edərik.

```
#include <iostream>

using namespace std;

int main (){

  int x[10], y, z, i,j, k;

  //x -in elementlerini daxil edek
  cout << "x -in elementlerini daxil edin \n";
  for (i=0; i<10; ++i)
      cin  >> x[i];

  //ESAS DOVR
  for (j=9; j >= 0; --j){

      //en boyuk elementi ve onun indeksini tapaq
      y = x[0];
      k = 0;

      for (i=1; i<=j; ++i){
          if (x[i] > y){
              y = x[i];
              k = i;
          }
      }

      //eger en boyuk element sonuncu deyilse
      //onu sona kocurek
      if ( k != j){
```

```
            z = x[k];
            x[k] = x[j];
            x[j] = z;
        }

    }

    //cergeni cap edek
    cout << "x -in yeni duzumu\n";
    for (i=0; i<10; ++i)
        cout  << x[i] << " ";
}
```

Nəticə

```
x -in elementlerini daxil edin
2 34 5 76 0 89 2 33 76 8
x -in yeni duzumu
0 2 2 5 8 33 34 76 76 89
```

İzahı: Burada biz iç-içə iki for dövr operatorundan istifadə edirik. Biz cərgənin verilmiş elementləri içərisindən ən böyüyünü tapmaq üçün bir dəfə baxılan elementləri nəzərdən keçirməliyik dövr operatoru ilə. Lakin bu zaman biz yalnız baxılan elementlər içərisində ən böyüyünü tapmış oluruq və onu ən başa sürüşdürürük. Yerdə qalan elementlər içərisində də ən böyüyünü tapmaq üçün biz bir daha cərgənin yerdə qalan elementlərini nəzərdən keçirməliyik dövr operatoru ilə. İkinci böyük elementi tapıb birincinin yanına yerləşdirdikdən sonra üçüncü ən böyüyü tapmaq üçün bir daha yenidən yerdə qalan elementləri nəzərdən keçirməliyik. Hər dəfə də elementləri nəzərdən keçirmək üçün dövr operatorundan istifadə edirik. Bu "***hər dəfə yerdə qalan elementlərin içərisindən ən böyüyünü tapmaq***" prosesini isə cərgənin elementlərinin sayı dəfə ***təkrar*** etməliyik. İndi isə kod sətirlərini ayrı-ayrı izah edək.

Mənə elə gəlir ilk sətirlər aydındır. Bizə lazım olan dəyişənləri elan edirik və cərgənin elementlərinə istifadəçinin daxil etdiyi qiymətləri mənimsədirik. Növbəti sətir aşağıdakı kimidir:

```
for (j=9; j > 0; --j){
```

Burada biz əsas dövrümüzü elan edirik. Sayğac olaraq j dəyişənindən istifadə edirik. Başlanğıc qiymət `9`, son qiymət `0`, hər-dəfə sayğacın qiymətini bir vahid azaldırıq(`--j`).

Əsas dövrün sayğacının başlanğıc və son qiymətlərini niyə belə götürməyimiz və onun qiymətini hər dəfə niyə bir vahid azaltmağımız irəlidə aydın olacaq.

```
            y = x[0];
```

```
k = 0;
```

Biz artıq əsas dövrün içindəyik və y və k dəyişənlərinə müvafiq olaraq cərgənin ilk elementinin qiymətini və indeksini mənimsədirik. Hər dəfə ən böyük elementi tapmaq üçün gördüyümüz başlanğıc iş.

Birinci əsas dövr hər dəfə yerdə qalan elementlər içərisindən ən böyük elementin tapılması üçündür. Növbəti koda baxaq:

```
for (i=0; i<=j; ++i){
    if (x[i] > y){
        y = x[i];
        k = i;
    }
}
```

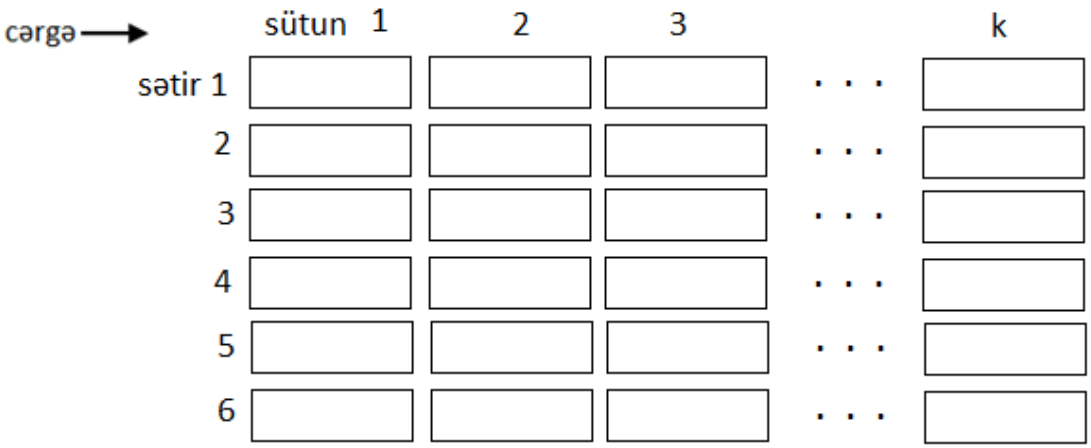
Burada biz birinci dövrün içərisində ikinci dövr yerləşdiririk. İkinci dövrün sayğacı 0-dan j-ə kimi davam edir. j yuxarıdakı dövrün sayğacıdır, hansı ki 9-dan 0-a kimi azalır. Yəni bir dəfə 9, ikinci dəfə 8, 7 v.s. 1 qiymətləri alır. Biz isə artıq bildiyimiz alqoritmə uyğun olaraq cərgənin ilk elementindən j-ci elementinə kimi olan elementləri arasında ən böyüyünü seçirik və sona sürüşdürürük:

```
if ( k != j){
    z = x[k];
    x[k] = x[j];
    x[j] = z;
}
```

Sonda isə cərəgnin yeni düzümünü çap edirik.

7.2 İkiölçülü Cərgələr.

Proqramlaşdırmada iki və daha çox ölçülü cərgələrdən istifadə olunur. İndiyə kimi baxdığımız cərgələrin hamısı birölçülü cərgələrə aiddir. İki ölçülü cərgələrdə isə cərgənin elementlərinin iki indeksi olur: sətir və sütun. Bir ölçülü cərgələrdə elementlər bir düz xətt boyunca ardıcıl düzülürdüsə, ikiölçülü cərgələrdə elementlrə sətir və sütunlarda cədvəl şəklində düzülürlər, aşağıda kimi:



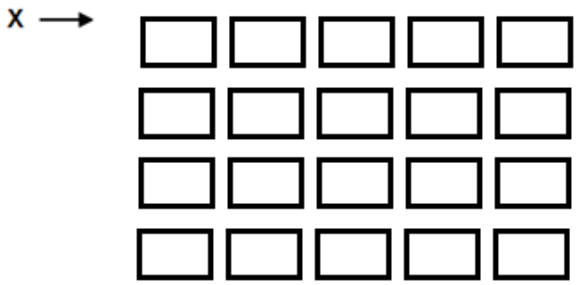
İkiölçülü cərgələri elan edərkən birölçülü cərgələrdə olduğu kimi adı yazırıq, lakin addan sonra bir yox iki kvadrat mötərizə açırıq, birinci kvadrat mötərizə içində **sətrilərin**, ikinci kvadrat mötərizə içində isə **sütunların** sayını yazırıq, aşağıdakı kimi:

```
tip ad [sətir_say] [sütun_say] ;
```

Misal üçün **4** **sətir** və **5** **sütundan** ibarət tam tipli x cərəsini aşağıdakı kimi elan edə bilərik:

```
int x [4][5] ;
```

Bu zaman aşağıdakı kimi 4 sətir və 5 sütundan ibarət tam tipli x cərgəsi yaranar:

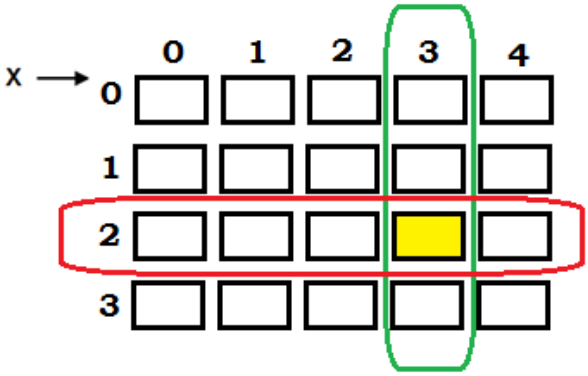


Cərgənin elementinə müraciət edərkən adı yazırıq və kvadrat mötərizə içərisində sətir və sütunun indeks nömrəsini göstəririk.

```
x [sətir_indeks] [sütun_index] ;
```

Burada diqqət yetirməli olduğumuz bir məqam odur ki, cərgədə sətir və sütunların indeksləri 0-dan nömrələnməyə başlayır. Belə ki, əgər sətirlərin sayı 4, sütunların sayı isə 5 –sə onda, sətir və sütunun ən böyük indeksləri müvafiq olaraq 3 və 4 olar.

Aşağıdakı təsvirdə bu daha aydın görünür:



Təsvirdə cərgənin `x[2][3]` elementi işarələnib, başqa sözlə cərgənin 2-ci sətrində və 3-cü sütununda yerləşən element.

İkiölçülü cərgələrlə bağlı nümunə çalışmalarla tanış olaq.

Nümunə. Aşağıdakı kod 10 sətir və 9 sütunu olan tam tipli x cərgəsi elan edir. x cərgəsinin bütün elementlərinə 1 qiyməti mənimsədir və x cərgəsinin elementlərini çap edir.

```
#include <iostream>

using namespace std;

int main (){

  int i, j, x[10][9];

  //cergenin butun elementlerine 1 qiymeti menimsedek
  for (i=0; i<10; ++i)
      for (j=0; j<9; ++j)
          x[i][j] = 1;

  //cergenin elementlerin cap edek
  cout << "Cergenin elementleri\n";
  for (i=0; i<10; ++i){

      for (j=0; j<9; ++j)
          cout << x[i][j] << " ";

      cout << "\n";
  }
}
```

Nəticə.

```
Cergenin elementleri
```

```
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
```

İzahı: Gördüyümüz kimi ikiölçülü cərgənin beten elementlərini nəzərdən keçirmək üçün iç-içə yerləşmiş iki for operatorundan istifadə elədik. Birinci for operatorunda i sayğacı cərgənin sətirləri boyu irəliləməyimizə, onun daxilində yerləşmiş for operatorunun j sayğacı isə cərgənin i-ci sətri boyu sütunlar üzrə hərəkət etməyimizə ikman verir. Bu şəkildə iç-içə təşkil olunmuş dövr operatoru vastəsilə istənilən an cərgənin konkret elementinə müraciət etdiyimiz üçün sətir nömrəsini bildirmək üçün i sayğacından, sütun nömrəsini bildirmək üçün isə j sayğacından istifadə edirik: `x[i][j]` şəklində. Başqa nümunəyə baxaq.

Nümunə. Aşağıdakı kod 10 sətir və 9 sütunu olan tam tipli x cərgəsi elan edir. x cərgəsinin elementlərinə 1-dən 90-a qədər qiymətlər mənimsədir və x cərgəsinin elementlərini çap edir.

```
#include <iostream>

using namespace std;

int main (){

  int i, j, k, x[10][9];

  k = 1;

  //cergenin butun elementlerine 1 qiymeti menimsedek
  for (i=0; i<10; ++i)
      for (j=0; j<9; ++j){
          x[i][j] = k;
          k++;
      }

  //cergenin elementlerin cap edek
  cout << "Cergenin elementleri\n";
  for (i=0; i<10; ++i){

      for (j=0; j<9; ++j)
          cout << x[i][j] << " ";

      cout << "\n";
  }
}
```

Nəticə.

```
Cergenin elementleri
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90
```

İzahı: Burada əlavə k dəyişənindən istifadə etdik və dövrün əvvəlində k-ya 1 qiyməti mənimsətdik. Dövr daxilində cərgənin elementlərini k-ya mənimsətdik, k-nın qiymətini isə hər dəfə bir vahid artırdıq.

Nümunə. Aşağıdakı kod ikiölçülü cərgənin ən böyük elementini tapır.

```
#include <iostream>

using namespace std;

int main (){

  int i, j, max, x[5][5];

  cout << "Cergenin elementlerini daxil edin \n\n";
  for (i=0; i<5; ++i)
      for (j=0; j<5; ++j)
          cin >> x[i][j];

  max = x[0][0];

  for (i=0; i<5; ++i)
      for (j=0; j<5; ++j)
          if (x[i][j] > max)
              max = x[i][j];

  cout << "\nCergenin en boyuk elementi: " << max;
}
```

Nəticə.

```
Cergenin elementlerini daxil edin

 1       2       43      5       67
 33      4       56      7       8
 445     6       76      890     0
 32      4       66      7754    8
 321     44      321     869     2

Cergenin en boyuk elementi: 7754
```

Kodun izahı birölçülü cərgələrdəki nümunə ilə eynidir.

Çalışmalar

Çalışma 1. Elə proqram tərtib edin ki, istifadəçidən 10 ədəd daxil etməsini istəsin və bu ədədlər içərisində 10-dan böyük olanlarının sayını tapsın.

Çalışma 2. Elə proqram tərtib edin ki, istifadəçidən 10 ədəd daxil etməsini istəsin və X ədədini daxil etməsini istəsin. Daha sonra proqram deməlidir ki, X ədədi 10 ədədin içərisində var, ya yoxdur.

Çalışma 3. Elə proqram tərtib edin ki, istifadəçidən 10 ədəd daxil etməsini istəsin və bu ədədlərdən ən böyüyünü və onun indeksini(sıradakı yerini) çap etsin.

Çalışma 4. Elə proqram tərtib edin ki, istifadəçidən 10 ədəd daxil etməsini istəsin və X ədədini daxil etməsini istəsin. Daha sonra proqram daxil olunan 10 ədəd arsında X-ə bərabər olanları tapmalı onları cərgədən silməli (yerdə qalanları onun yerinə sürüşdürməli), axıra isə 0 əlavə etməlidir. Yekun cərgəni çap edin.

Çalışma 5. 20 elementdən ibarət olan c cərgəsi və hər birində 10 element olan b ve d cərgələri elan edin. Proqram b və d cərgələrinin elementlərinə istifadəçinin daxil etdiyi qiymətləri mənimsədir. Daha sonra bu iki cərgənin elementlərini c cərgəsinə əlavə edir (əvvəlcə b, sonra d) və c cərgəsinin elemenlərini çap edir.

Çalışma 6. Elə proqram tərtib edin ki, istifadəçidən 5 sətir və 5 sütundan ibarət olan ikiölçülü cərgənin elementlərini daxil etməsini istəsin. Proqram ikiölçülü cərgənin bütün elementləri üzərinə 1 əlavə edib nəticəni çap etsin.

Çalışma 7. Elə proqram tərtib edin ki, istifadəçidən 7 sətir və 6 sütundan ibarət olan ikiölçülü cərgənin elementlərini daxil etməsini istəsin. Proqram bu cərgənin elemetlərindən14 sətir və 3 sütundan ibarət olan ikiölçülü cərgə yaratsın və çap etsin.

Çalışma 8. Elə proqram tərtib edin ki, istifadəçidən 7 sətir və 6 sütundan ibarət olan ikiölçülü cərgənin elementlərini daxil etməsini istəsin. Proqram bu cərgənin sətirləri ilə sütunlarının yerini dəyişib yeni 6 sətir və 7 sütundan ibarət cərgə yaratsın və onu çap etsin.

Çalışma 9. Cərgənin elementlərini tələb olunan sayda sola sürüşdürən proqram tərtib edin.

Çalışma 10. Elə proqram qurun ki, cərgənin verilmiş elementə bərabər olan bütün elementlərinin indekslərini çap etsin.

Çalışma 11. Elə proqram qurun ki, cərgənin verilmiş elementə bərabər olan bütün elementləri cərgənin sonuna sürüşdürsün.

Çalışma 12. 5 sətir cə 4 sütunu olan ikiölçülü cərgənin 2 və 4-cü sətirlərinin yerlərini dəyişən proqram tərtib edin.

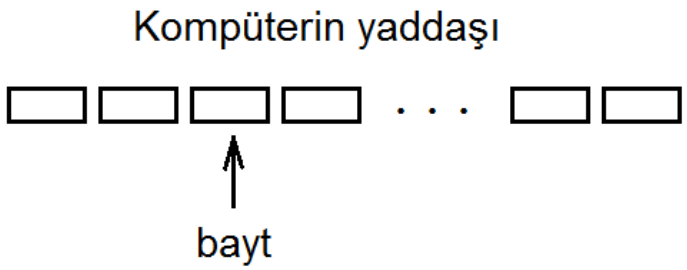
§8 Göstəricilər

Göstəricilər mövzusunda hamınızı xoş gördük. Olduqca maraqlı, maraqlı olduğu qədər də çətin bir mövzu üzərinə gəldik. Göstəriciləri başa düşmək proqramlaşdırmanın bütün qapılarının sizin üzünüzə açıq olması deməkdir.

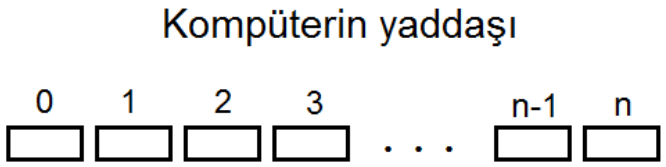
Göstəricilər iri həcmli məlumatları parametr kimi ötürmək, dinamik ölçülü verilənlər strukturları ilə işləmək, yaddaşı idarə etmək v.s. məqsədlər üçün istifadə olunur. Buna görə göstəriciləri başa düşməzdən qabaq kompüter yaddaşının nə cür təşkil olunması barədə təsəvvürümüz olmalıdır.

8.1 Kompüterin Yaddaşı

Kompüterin yaddaşı ***bayt*** adlandırılan və bir-birinin ardınca bir düz xətt boyunca düzülmüş ən kiçik yaddaş sahələrindən ibarət olur.



Bu sıra ilə düzülmüş baytlar 0-dan başlayaraq nömrələnir və hər baytın nömrəsi özündən əvvəlki baytın nömrəsindən bir vahid böyük olur.



Kompüterin yaddaşının sonuncu baytının nömrəsi kompüterin yaddaş qurğusunun həcmindən asılı olaraq dəyişir.

Baytların sıra nömrələri onların ***ünvanları*** adlanır. Kompüterin ilk baytının ünvanı 0, ikinci baytının ünvanı 1 v.s. olur.

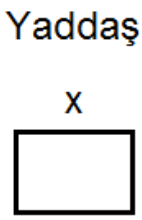
8.1.1 Dəyişənlərin yaddaşda yerləşməsi

Hər bir dəyişəni elan edərkən əvvəlcə onun tipini göstəririk. Dəyişənin tipinə görə kompilyator həmin dəyişənin yaddaşda nə qədər yer, başqa sözlə neçə bayt yer tutacağını müəyyənləşdirir. Misal üçün int tipli dəyişənlər yaddaşda 4 bayt, double tipli dəyişənlər 8, char tipli dəyişənlər isə 1 bayt yer tuturlar. Əlbəttə əməliyyatlar sistemindən və kompüterin arxitekturasından asılı olaraq tiplərin ölçüləri dəyişə bilər. Lakin sadəlik xətrinə tiplərin ölçülərini göstərilən şəkildə qəbul edək.

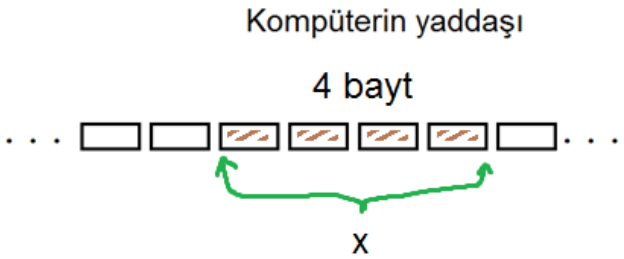
Tutaq ki int tipli hər hansı x dəyişəni elan etmişik:

```
int x;
```

Bu zaman deyirik ki, x dəyişəni üçün yaddaşda 4 baytlıq yer ayrılır və həmin yeri x adı ilə müraciət edirik. Əvvəlki dərslərimizdə bunun qrafik olaraq aşağıdakı kimi təsvir edirdik:



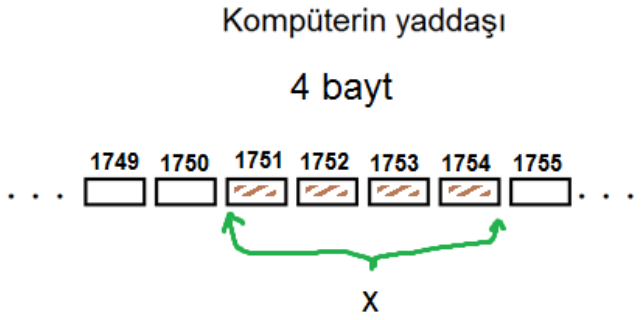
Lakin indi yaddaşın strukturunu daha dəqiq bildiyimizə görə x dəyişəninin yaddaşdakı yerini aşağıdakı kimi təsvir edə bilərik:



x dəyişəni int tipli olduğundan onun üçün yaddaşda 4 bayt yer tələb olunur.

8.1.2 Dəyişənin ünvanı

Biz yuxarıda qeyd etdik ki, yaddaşı təşkil edən baytlar sıra ilə düzülür və 0-dan başlayaraq nömrələnir. Gəlin yuxarıdakı x dəyişəninin yaddaşdakı vəziyyətini göstərən təsvirdəki baytarı şərti olaraq nömrələyək:



Baxdığımız şəkildə x dəyişəni yaddaşda 1751 ünvanlı baytdan başlayaraq növbəti, həmin bayt da daxil olmaqla 4 bayta yerləşdirilib. x dəyişəninin yerləşdiyi ilk baytın ünvanı, yəni 1751 x dəyişəninin yaddaşdakı ünvanı kimi hesablanır. Ünvanı göstərən ədədlər adətən 16-lıq (HEX) say sistemləri ilə ifadə olunur, misal üçün 0x00ce45d1 v.s. kimi.

8.1.3 Ünvan operatoru

Proqramın icrası zamanı dəyişənin yaddaşda hansı ünvanda yerləşməsini müəyyən etmək mümkündür. Bunun üçün ünvan operatorundan istifadə olunur. Ünvan operatoru & - kimi işarə olunur və sintaksisi aşağıdakı kimidir:

```
&dəyişən
```

Yəni hər-hansı dəyişənin ünvanın əldə etmək üçün onun adının əvvəlinə ünvan operatorunu yazmalıyıq. Gəlin ünvan operatorundan istifadəyə aid sadə bir proqram çalışması edək.

Çalışma. int tipli hər hansı x dəyişəni elan edin. Ünvan operatorundan istifadə etməklə bu dəyişənin ünvanın ekranda çap edin. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int main (){

    int x;

    cout << "x -in unvani = " << &x ;

}
```

Nəticə

```
x -in unvani = 0039F864
```

Gördüyümüz kimi mənim kompüterimdə x dəyişəninin yaddaşdakı ünvanı - `0039F864` oldu.

8.2 Göstərici

Vaxt gəlib çatdı konkret olaraq göstəricinin özü ilə taşın olmağa. Yuxarıda biz dəyişənləri ünvanları ilə tanış olduq. Yaddaş ünvanı proqramlaşdırmada, xüsusən də C/C++ dillərində çox mühüm məsələdir. Hər hansı dəyişənin ünvanın bilməklə onda olan məlumatı oxumaq, dəyişmək və başqa bir yerə ötürmək mümkündür. Adətən məlumatın ölçüsü böyük olduğu hallarda məlumatın özünü bir yerdən başqa yerə köçürməyə çox vaxt sərf olunur. Belə hallarda məlumatın yaddaşdakı ünvanın ötürmək çox əlverişlidir. C++ dilində ünvanla işləmək üçün göstəricilərdən istifadə olunur. Göstəricilər də biz bildiyimiz adi dəyişənlər kimidir. Fərq yalnız ondan ibarətdir ki, göstəricilər özlərində qiymət olaraq ***ünvan*** yadda saxlayırlar. Aşağıdakı təsvirdən də bunu anlamağa çalışaq.

8.2.1 Göstəricilərin elanı

Göstəricilər də adi dəyişənlərlə birlikdə eyni elan sətrində elan oluna bilər. Göstəricilərin adının əvvəlinə onların göstərici olduqlarını bildirmək üçün ulduz simvolu artırmaq tələb olunur, aşağıdakı kimi:

```
tip *gosterici;
```

Misal üçün `int` tipli x adlı göstərici elan etmək istəsək kod aşağıdakı kimi olar:

```
int *x;
```

Eyni elan sətrində həm adi dəyişən, həm də göstərici elan edə bilərik, aşağıdakı kimi:

```
int *x, y, *z, q;
```

Yuxarıdakı elan sətrində int tipindən olan x,z göstəriciləri və y,q dəyişənləri elan olunub. Digər tiplərdən də göstəricilər eyni qayda ilə elan olunur, nümunəyə baxaq:

```
char *s;
float *r;
```

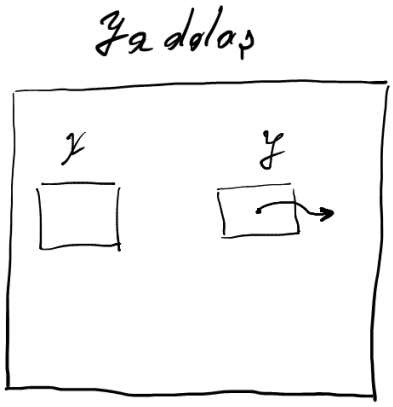
Yuxarıda char tipli s, float tipli r göstəriciləri elan olunub.

8.2.2 Göstəriciyə ünvan mənimsədilməsi

Göstərici elan etdikdən sonra ona ünvan mənimsədə bilərik. Lakin diqqətdə saxlamayıq ki, hər bir göstəriciyə yalnız onun öz tipindən olan dəyişənin ünvanın mənimsədə bilərik. Yəni int tipli göstəriciyə, char tipli dəyişənin ünvanın mənimsədə bilmərik, elə int tipli dəyişənin ünvanın mənimsətməliyik. Dəyişənin ünvanın nə cür əldə edəcəyimizi bilirik – ünvan operatoru vastəsilə - &.

Tutaq ki, int tipli hər hansı x dəyişəni və y göstəricisi elan etmişik.

```
int x, *y;
```

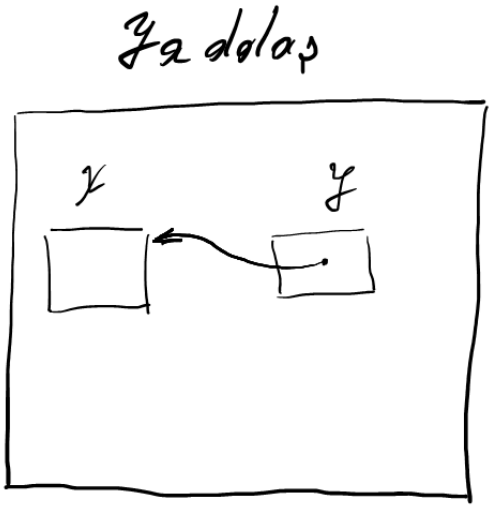


Yuxarıdakı təsvirdə y göstəricisinin istinad elədiyi ünvan ox ilə bildirilir. Elan olunan zaman göstəricinin hansı ünvana istinad elədiyi məlum olmadığına görə təsvirdə o göstərilməyib.

Ünvan operatorundan istifadə etməklə y göstəricisinə x dəyişəninin ünvanını aşağıdakı kimi mənimsədə bilərik:

```
y = &x;
```

Bu zaman y göstəricisi x dəyişəninin yaddaşdakı ünvanına istinad edir, aşağıdakı təsvirdəki kimi:



Daha sonra y –in ***qiymətini*** çap eləsək, x –in ***ünvanı*** çap olunar.

```
cout << y ;
```

Tam proqram kodu aşağıdakı kimi olar:

```
#include <iostream>
```

```
using namespace std;

int main (){

    int x, *y;

    y = &x;

    cout << "x -in unvani = " << y ;

}
```

Nəticə:

```
x -in unvani = 003AF8D4
```

Biz verilmiş ünvan dəyişəninə eyni tipdən olan müxtəlif dəyişənlərin ünvanlarını mənimsədə bilərik. Nümunəyə baxaq:

Nümunə: float tipindən göstərici və bir neçə dəyişən elan edin. Göstəricini həmin dəyişənlərin ünvanlarına bir –bir mənimsədib onları çap edin. Kod aşağıdakı kimi olar:

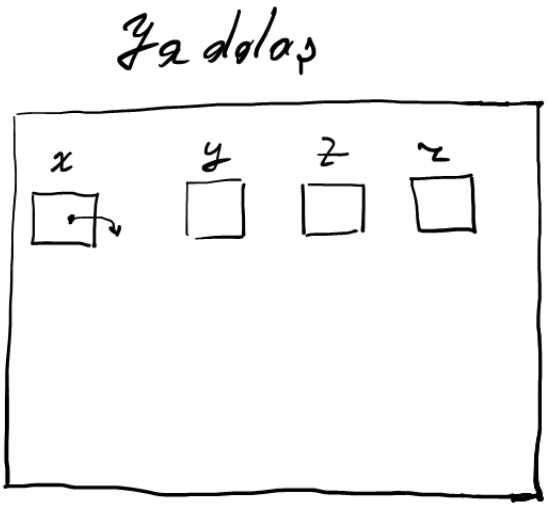
```
#include <iostream>

using namespace std;

int main (){

    float *x, y, z, r;

    //x gostericisine y -in unvanin meminsedek
    x = &y;
    cout << "y -in unvani = " << x << "\n";

    //x gostericisine z -in unvanin meminsedek
    x = &z;
    cout << "z -in unvani = " << x << "\n" ;


    //x gostericisine r -in unvanin meminsedek
    x = &r;
    cout << "r -in unvani = " << x << "\n" ;

}
```

Nəticə:

```
y -in unvani = 003CFB24
z -in unvani = 003CFB18
r -in unvani = 003CFB0C
```

İzahı: Proqramın ilk sətrində float tipindən olan x göstəricisi və y, z və r dəyişənləri elan olunur.

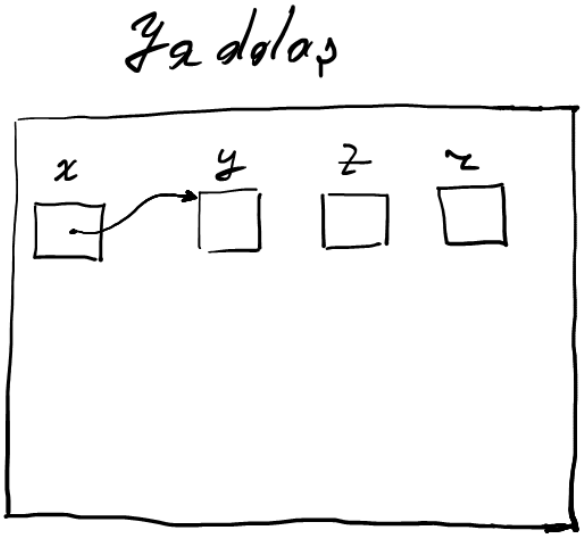
```
float *x, y, z, r;
```



Daha sonra ünvan operatoru ilə x göstəricisinə y dəyişəninin qiyməti mənimsədilir.

```
x = &y;
```

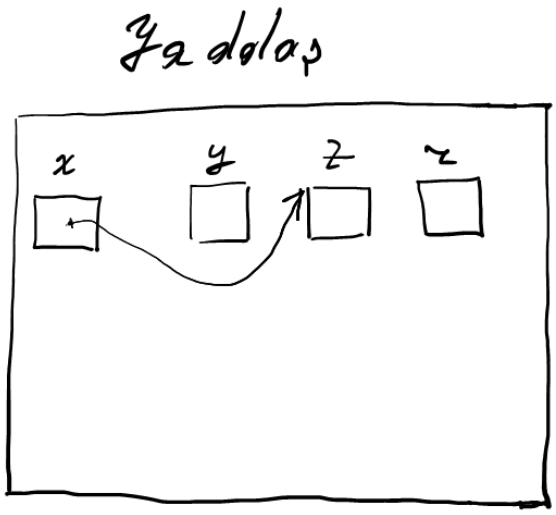
Aşağıdakı təsvirdəki kimi:



Təsvirdən də göründüyü kimi x göstəricisi hal-hazırda yaddaşda y dəyişəninə istinad edir. Proqramın növbəti sətrində isə x göstəricisinə z dəyişənin ünvanı mənimsədilir:

```
x = &z;
```

Nəticədə x göstəricisi z dəyişəninə istinad edər, təsvirdəki kimi:



x göstəricisinə r dəyişəninin ünvanının mənimsədilməsi də eyni qayda ilə yerinə yetirilir.

8.2.3 Məlumatın əldə olunması – İstinad operatoru

Göstəriciyə ünvan mənimsətdikdən sonra həmin ünvanda olan məlumatı əldə edə bilərik. Nəinki əldə edə, hətta o məlumatı dəyişədə bilərik. Bunun üçün istinad operatorundan istifadə etməliyik. İstinad operatoru ulduz - * simvolu ilə işarə olunur. İstinad operatorunun sintaksisi aşağıdakı kimidir:

```
*göstərici
```

Yəni ulduz operatorunu məlumatı əldə etmək istədiyimiz göstəricinin əvvəlinə yazırıq. Düzdü bu göstəricinin elanına oxşasa da, qəti şəkildə elandakı ulduz ilə, istinad operatoru olan ulduzu qarışdırmaq olmaz. Elanda sadəcə yazdığımız adın göstərici olduğunu bildirmək üçün əvvəlinə ulduz simvolu artırırdıq. İstinad operatorunda isə göstəricinin yadda saxladığı ünvandakı məlumata müraciət etmək üçün. Aşağıdakı nümunədə bu izah olunur:

Nümunə: int tipli x dəyişəni və y göstəricisi elan edin. x dəyişəninə hər hansı qiymət, y göstəricisinə isə x –in ünvanın mənimsədin. Istinad operatorundan istifadə etməklə y göstəricisinin istinad elədiyi ünvanda yerləşən məlumatı çap edin. Kod belə olar:

```
#include <iostream>
```

```
using namespace std;

int main (){

    int x, *y;

    //x -e her hansi qiymet verek
    x = 12;

    //y-e x-in unvanin menimsedek
    y = &x;

    //x-in unvanin cap edek
    cout << "x -in unvani = " << y << "\n";

    //x -de olan melumati cap edek
    cout << "x -in qiymeti = " << *y << "\n";

}
```

Nəticə:

```
x -in unvani = 0043FBEC
x -in qiymeti = 12
```

İzahı: Əvvəlcə int tipindən olan x dəyişəni və y göstəricisi elan edirik.

```
int x, *y;
```

Daha sonra x dəyişəninə hər hansı qiymət mənimsədirik:

```
x = 12;
```

Daha sonra ünvan operatorundan istifadə etməklə x dəyişəninin ünvanın y göstəricisinə mənimsədirik:

```
y = &x;
```

Bundan sonra x-in ünvanını çap etmək üçün sadəcə y, x-in qiymətini çap etmək üçün isə *y yazılışından istifadə edirik.

Göstəricinin adının əvvəlinə ulduz simvolu artıraraq onu adi dəyişənlər kimi istifadə edə bilərik. Bu zaman hesab etmək olar ki, göstərici təmsil etdiyi dəyişənə çevrilir. Başqa nümunə baxaq.

```
#include <iostream>
```

```
using namespace std;

int main (){

    int x, *y;

    cout << "Her-hansi eded daxil edin\n";
    cin >>x;

    //y-e x-in unvanin menimsedek
    y = &x;

    cout << *y << " + 5 = " << *y + 5 << "\n";

}
```

Nəticə:

```
Her-hansi eded daxil edin
23
23 + 5 = 28
```

Nümunədən də göründüyü kimi y göstəricisinə x dəyişəninin ünvanın mənimsətdikdən sonra y göstəricisinin adının əvvəlinə * simvolu – yəni istinad operatoru artırmaqla *y – dən elə x dəyişəni kimi istifadə edə biləik.

Başqa nümunəyə baxaq. Bu nümunədə isə göstərici vastəsilə istinad olunan dəyişənə qiymət mənimsədəcəyik.

```
#include <iostream>

using namespace std;

int main (){

    int x, *y;

    //x-e her-hansi qiymet verek
    x = 98;

    //x-in qiymetin cap edek
    cout << "x evvel = " << x << "\n";

    //y-e x-in unvanin menimsedek
    y = &x;

    //y gostericisi ile x-e yeni qiymet menimsedek
    *y = 95;

    //x-in yeni qiymetini cap edek
    cout << "x indi = " << x << "\n";
```

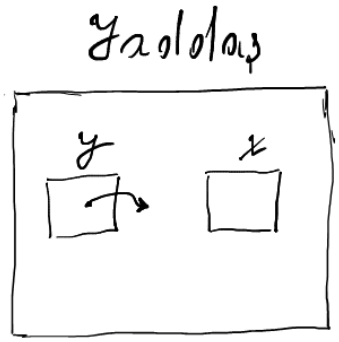
```
}
```

Nəticə:

```
x evvel = 98
x indi = 95
```

İzahı: Proqramda int tipli x dəyişəni və y göstəricisi elan edirik:

```
int x, *y;
```

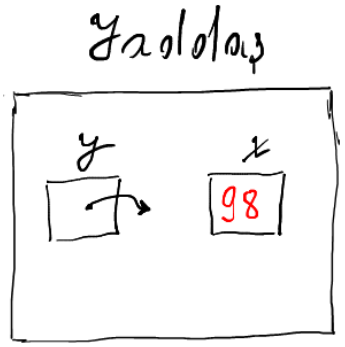


Əvvəlcə x-ə 98 qiymətini veririk:

```
x = 98;
```

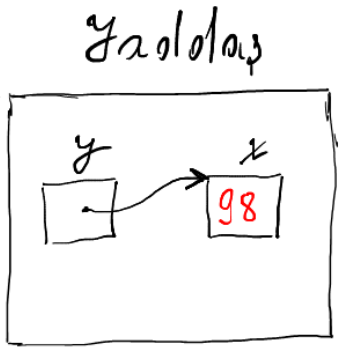
və x-in bu qiyməti ekranda çap edirik.

```
cout << "x evvel = " << x << "\n";
```



Daha sonra y göstəricisinə x-in ünvanın mənimsədirik:

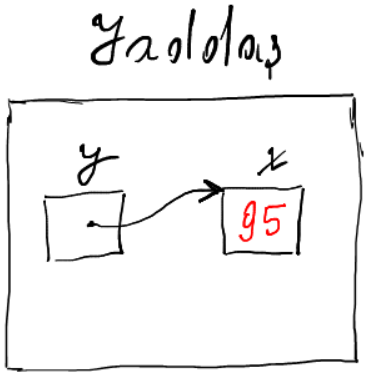
```
y = &x;
```



Bu zaman y göstəricisi x dəyişəninə istinad etmiş olur. Növbəti sətirdə isə istinad operatorundan istifadə etməklə y göstəricisi vastəsilə x-ə yeni qiymət veririk:

```
*y = 95;
```

Nəticədə x dəyişəninin əvvəlki qiyməti silinir və yerinə yeni qiymət yazılır.



Gördüyümüz kimi yuxarıdakı kodda x dəyişəni iştirak etmədi, lakin bu operatordan sonra x-in qiymətin çap etdikdə onun qiymətinin dəyişdiyini görürük:

```
cout << "x indi = " << x << "\n";
```

8.3 Göstəricilər və Cərgələr

C++ dilində göstəricilər cərgələrlə sıx bağlıdır. Cərgənin elementləri yaddaşda ardıcıl yerləşdiyindən göstəricini cərgə boyu "*yuxarı-aşağı*" sürüşdürməklə cərgənin elementlərinə müraciət etmək, onları dəyişmək, cərgələri funksiyalara parametr kimi ötürmək v.s. mümkündür. Funksiyalara ötürməni növbəti dərsimizdə keçəcəyik. Hələlik isə göstəricilər və cərgələrlərin əlaqəsini örgənək. İlk olaraq göstəricilərin cərgənin elementlərinə mənimsənilməsindən başlayaq. Adi dəyişənlərdə olduğu kimi, eyni

qayda ilə cərgə elementlərinin də ünvanlarını göstəricilərə mənimsədə, göstəricilər vastəsilə onların qiymətlərinə mürcaiət edə bilərik. Nümunələrə baxaq.

Nümunə: Aşağıdakı kod tam tipli 5 elementdən ibarət x cərgəsi və y göstəricisi elan edir. Daha sonra x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədilir.

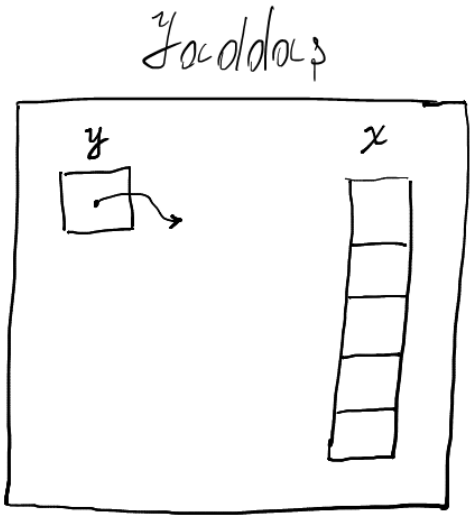
```
#include <iostream>

using namespace std;

int main (){

    int i, x[5], *y;

     x[0] = 12;
     x[1] = 4;
     x[2] = 32;
     x[3] = 26;
     x[4] = 17;

     cout << "x -in elementleri evvel:\n";

     for(i=0; i<5; ++i)
         cout << x[i] << "  ";

     y = &x[2];
     *y = 55;

     y = &x[4];
     *y = 88;

     cout << "\nx -in elementleri sonra:\n";

     for(i=0; i<5; ++i)
         cout << x[i] << "  ";

}
```

Nəticə:

```
x -in elementleri evvel:
12  4  32  26  17
x -in elementleri sonra:
12  4  55  26  88
```

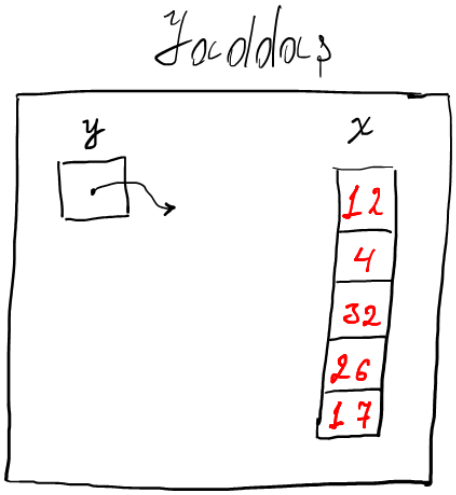
İzahı: Proqramda int tipli 5 elementdən ibarət x cərgəsi və y göstəricisi elan edirik(i-ni nəzərə almayaq sadəlik üçün).

```
int i, x[5], *y;
```



Sonra x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədirik:

```
x[0] = 12;
x[1] = 4;
x[2] = 32;
x[3] = 26;
x[4] = 17;
```



və bu qiymətləri çap edirik.

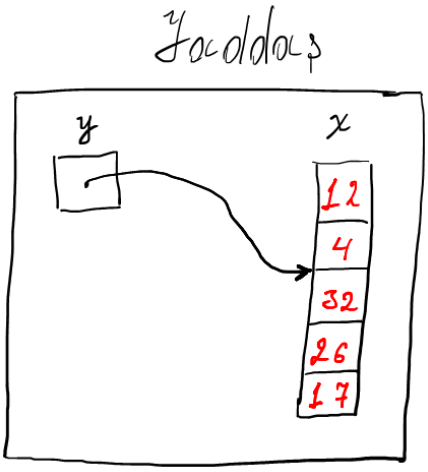
```
cout << "x -in elementleri evvel:\n";

for(i=0; i<5; ++i)
    cout << x[i] << "  ";
```

Aşağıdakı nəticə ekranda çap olunur:

```
x -in elementleri evvel:
12   4   32   26   17
```

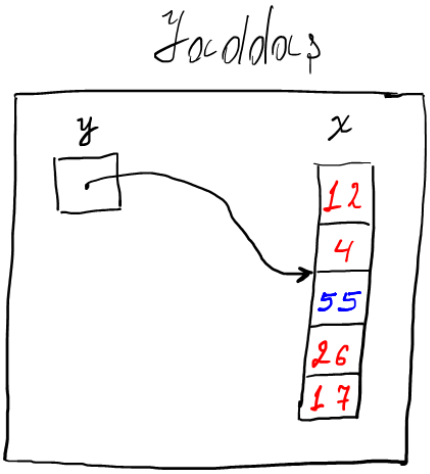
Daha sonra ünvan - **&** operatoru vastəsilə y göstəricisinə x cərgəsinin 3-cü elementinin ünvanını mənimsədirik:

```
y = &x[2];
```



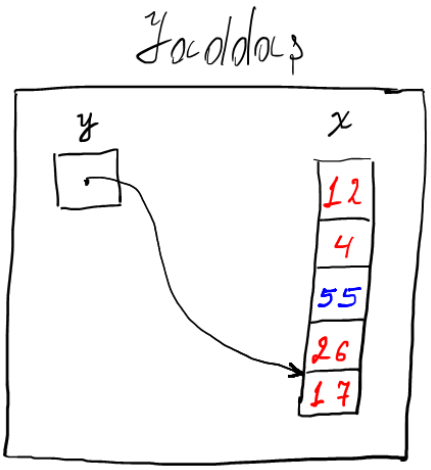
y göstəricisinə x-in 3-cü elementinin ünvanın mənimsətdikdən sonra istinad – ***** operatoru ilə həmin elementə 55 qiyməti veririk:

```
*y = 55;
```



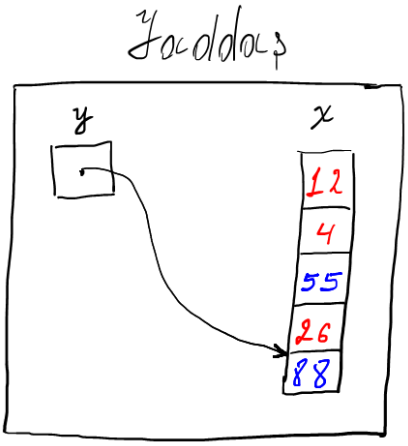
Daha sonra ünvan - **&** operatoru vastəsilə y göstəricisinə x cərgəsinin 5-ci elementinin ünvanını mənimsədirik:

```
y = &x[4];
```



y göstəricisinə x-in 5-ci elementinin ünvanın mənimsətdikdən sonra istinad – * operatoru ilə həmin elementə 88 qiyməti veririk:

```
*y = 88;
```



Sonda x cərgəsinin yeni qiymətlərini çap edirik:

```
cout << "\nx -in elementleri sonra:\n";

for(i=0; i<5; ++i)
    cout << x[i] << "  ";
```

Aşağıdakı nəticə ekranda çap olunur:

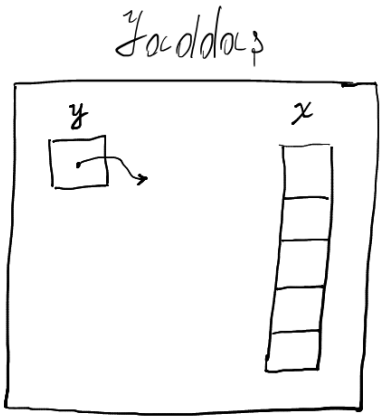
```
x -in elementleri sonra:
12  4  55  26  88
```

8.3.1 Cərgənin adı

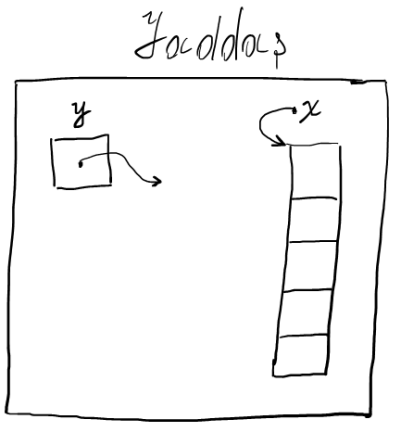
C++ dilində cərə adı cərgənin ilk elementinə istinad edən göstəricidir. Cərgə adının göstəricilərdən tək fərqi odur ki, göstəriciləri yaddaşın istənilən ünvanına mənimsətmək olar, cərdə adı isə proqramın bütün icrası boyu tək bir ünvana istinad edir – cərgənin ilk elementinə.

Tutaq ki, aşağıdakı kimi int tipli 5 elementdən ibarət x cərgəsi və y göstəricisi elan etmişik:

```
int   x[5], *y;
```



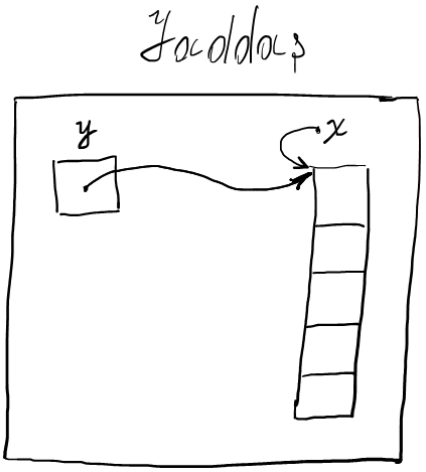
Bu şəkildə x-i cərgənin ilk elementinə istinad edən göstərici kimi təsvir etsək, aşağıdakı kimi olar:



Gördüyümüz kimi bu şəkildə cərgənin adı – x, cərgənin ilk elementinə - x[0] – ra istinad edən göstərici kimi təsvir olunub. Bir halda ki cərgə adı göstəricidir, onda eyni tipdən olan göstəricini cərgə adına mənimsətməkdə heç bir qayda pozuntusu olmaz. Yəni yuxarıda elan etdiyimiz y göstəricisini x -ə aşağıdakı kimi mənimsədə bilərik:

```
y = x;
```

Nəticədə y göstəricisi x –in ***göstərdiyi*** yerə ***göstərər***, aşağıdakı kimi:



Gördüyümüz kimi y göstəricisi x cərgəsinin ilk elementinə istinad edir. Bundan əlavə cərgə adından a da göstəricilərdə olduğu kimi istinad - * operatoru ilə göstərdiyi ünvandakı məlumatı – cərgənin ilk elementini əldə etmək olar, aşağıdakı kimi:

```
*x
```

Misal üçün aşağıdakı kod cərgənin ilk elementini, yəni `x[0]`- rı ekranda ekranda çap edər:

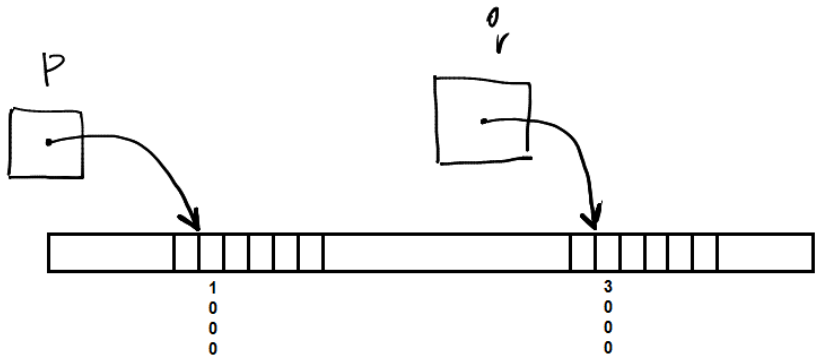
```
cout << *x;
```

8.4 Göstəricilər üzərində hesab

C++ dilində göstəricilər üzərində hesab əməlləri aparmağa icazə verilir. Lakin ancaq üstəgəl və çıxma əməllərinə. Göstəricilər üzərində vurma, bölmə, qalıq kimi hesab əməllərinin aparılmasının heç bir mənası yoxdur. Üstəgəl və çıxma əməlləri isə göstəriciləri yaddaşda müxtəlif ünvanlara istinad etdirmək üçün aparılır. Daha çox yaddaşla işləmək üçün nəzərdə tutulduğundan bu əməllər bizim riyaziyyatdan bildiyimiz məntiqdən biraz fərqlənir. Daha dəqiq desək müxtəlif tiplərdən olan göstəriciləri eyni qiymət qədər artırıb-azaldanda fərqli nəticələr alırıq. Burada tiplərin

ölçüləri əsas rol oynayır. Yuxarıda tiplərin ölçülərinin müxtəlif sistemlərdə müxtəlif ola biləcəyi haqda danışmışdıq. İndi şərti olaraq hesab edək ki, char tipinin ölçüsü 1 bayt, int tipininki isə 4 baytdır. Tutaq ki, char və int tipindən uyğun olaraq p və q göstəriciləri elan etmişik və onlar yaddaşda müvafiq olaraq 1000-ci və 3000 –ci baytlara istinad edirlər:

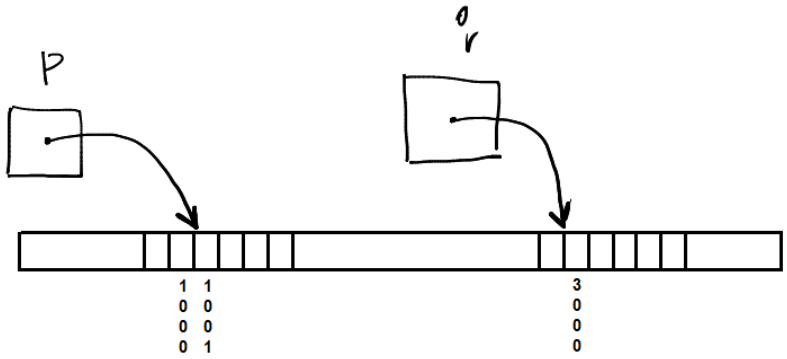
```
char *p;
int *q;
```



Əgər p göstəricisinin qiymətini 1 vahid artırsaq:

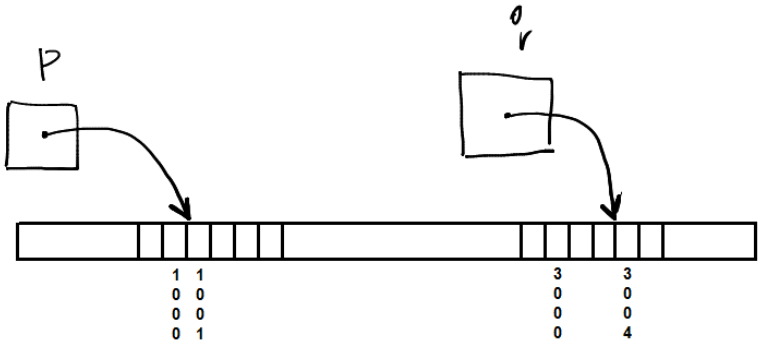
```
p = p + 1;
```

Bu zaman p göstəricisi char tipindən olduğuna görə onun qiyməti 1 ədəd artaraq 1001 olar, başqa sözlə p göstəricisi yaddaşda 1001 –ci baytın üzərinə sürüşər.



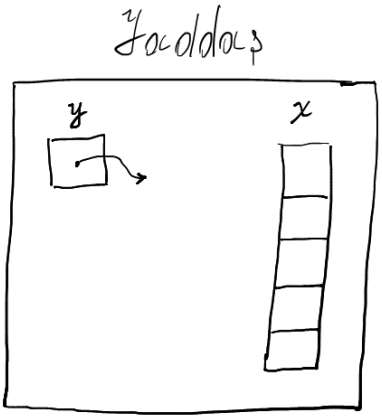
Eyni qayda ilə q göstəricisinin üzərinə 1 əlavə etsək, amma onun qiyməti 1 yox 4 ədəd artaraq (int tipinin ölçüsü qədər – 4 bayt) 3004 olar.

```
q = q + 1;
```



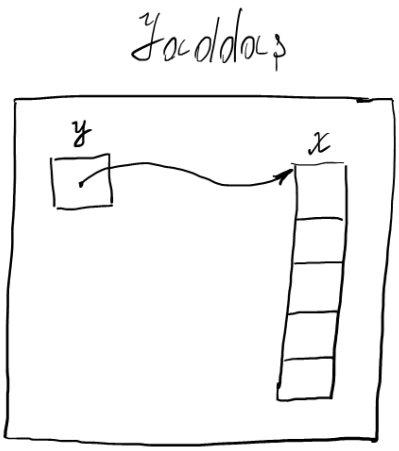
Yəni tipin ölçüsü nə qədərdirsə, göstəricini müəyyən ədəd artırıb – azaldanda, göstəricinin qiyməti həmin ədəd vurulsun tipin ölçüsü qədər dəyişir. Bu ona görədir ki, əgər verilmiş tipdən olan məlumatlar yaddaşda ardıcıl yerləşibsə, onda göstəricinin qiymətini artırıb-azaltmaqla növbəti məlumatın üzərinə sürüşmək mümkün olsun. Misal üçün cərgələrdə elementlər yaddaşda ardıcıl düzülür. Buna görə cərgənin hər-hansı elementinə istinad edən göstəricinin qiyətini artırıb-azaltsaq digər elementlərin üzərinə sürüşə bilərik. Fikrimizi kod ilə izah edək. Tutaq ki, int tipli 5 elementdən ibarət x cərgəsi və y göstəricisi verilib:

```
int   x[5], *y;
```



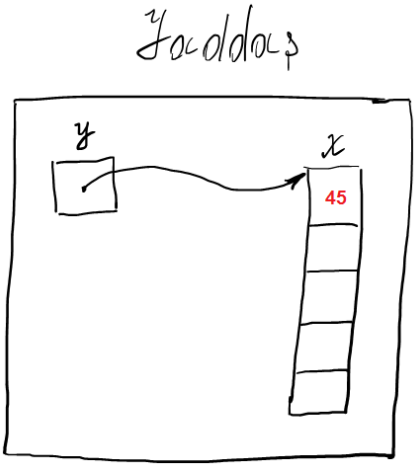
y göstəricisini x cərgəsinin ilk elementinin ünvanına mənimsədək:

```
y = x;
```



y göstəricisi ilə x cərgəsinin ilk elementinə 45 qiyməti mənimsədək:

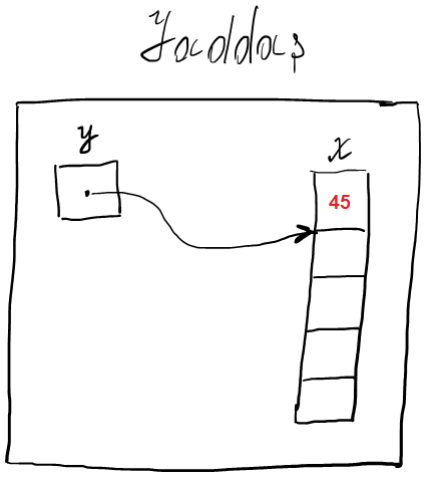
```
*y = 45;
```



y göstəricinin qiymətini 1 vahid artıraq.

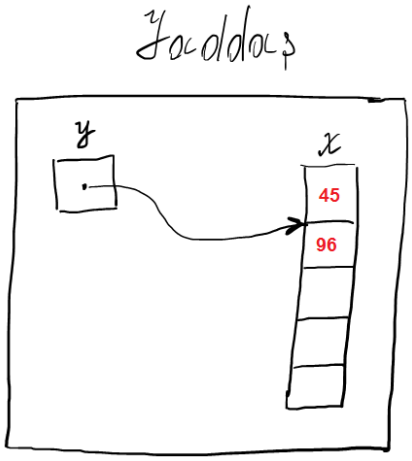
```
y++;
```

Nəticədə y göstəricisi x cərgəsinin ikinci elementi üzərinə sürüşər:



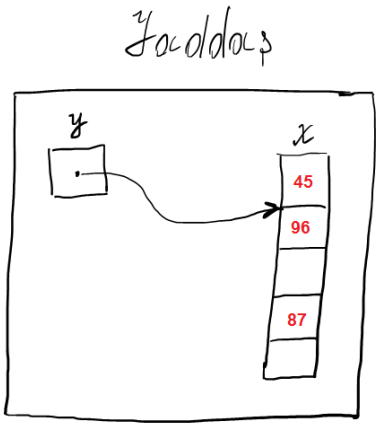
y göstəricisi ilə x cərgəsinin ikinci elementinə 96 qiyməti mənimsədək:

```
*y = 96;
```



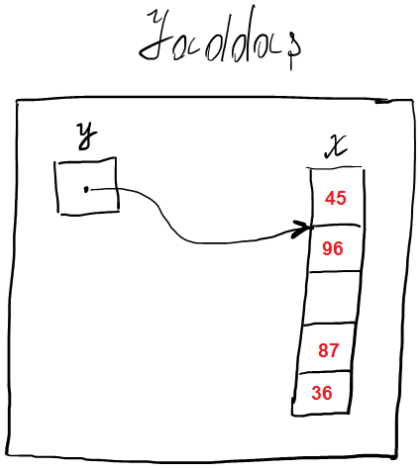
Qeyd eliyim ki, göstərici ilə cərgənin hansısa elementinə müraciət etmək üçün heç də göstəricini həmin elementin üzərinə sürüşdürməyə ehtiyac yoxdur. Elə tələb olunan element ilə göstəricinin istinad elədiyi elementlərin indeks fərqini göstəricinin üzərinə əlavə etməklə tələb olunan elementə qiymət mənimsətmək mümkündür. Misal üçün baxdığımız misalda göstərici cərgənin ikinci elementi üzərindədir. Aşağıdakı kod cərgənin 4-cü elementinə 87 qiyməti mənimsədər:

```
*(y+2) = 45;
```



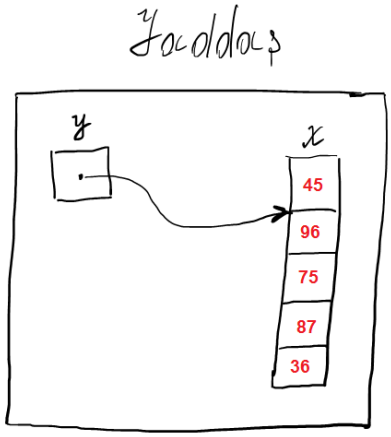
Cərgənin adı həmişə cərgənin ilk elementinə istinad edən göstərici olduğundan cərgənin elementlərinə indeks operatorundan (içində indeksi yazdığımız kvadrat mötərizələr - []) da əlavə yuxarıdakı qayda ilə istinad operatoru ilə də müraciət edə bilərik. Misal üçün cərgənin adından istifadə edərək istinad operatoru ilə x cərgəsinin sonuncu elementinə 36 qiyməti mənimsədək:

```
*(x+4) = 36;
```



Öz növbəsində isə indeks ([]) operatorundan istifadə etməklə, yəni tələb olunan elementin indeks nömrəsini kvadrat mötərizə arasında verməklə y göstəricisi vastəsilə də elementə qiymət mənimsədə bilərik. Misal üçün x cərgəsinin üçüncü elementinə 75 qiymətini y göstəricisi və indeks operatoru ilə aşağıdakı kimi mənimsədə bilərik:

```
y[2] = 75;
```



Gördüyümüz kimi C++ dilində cərgələr və göstəricilər arasında çox sıx bağlılıq var. Nümunə proqrama baxaq:

```
#include <iostream>

using namespace std;

int main (){

    int i, x[10], *y;

    y = x;

    for(i=0; i<10; ++i)
        *(y+i) = i;


    for(i=0; i<10; ++i)
         cout << *(y+i) << "  ";

    cout << "\n";


    for(i=0; i<10; ++i)
         cout << *(x+i) << "  ";

    cout << "\n";


    for(i=0; i<10; ++i)
         cout << x[i] << "  ";

    cout << "\n";


    for(i=0; i<10; ++i)
         cout << y[i] << "  ";
}
```

Nəticə:

```
0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  5  6  7  8  9
```

Qeyd: aşağıdakı mövzular bir qədər çətinliyi artırılmış mövzulardı. C++ dilini örgənməyə yeni başlayanlar bu mövzuların üzərindən keçə bilər.

8.5 İkiqat Göstəricilər

C++ dilində göstəricilərin özlərinə də göstərici təyin eləmək mümkündür. Buna ***ikiqat göstərici*** deyirlər – ***double pointers***. İkiqat göstəricilər aşağıdakı kimi elan olunur:

```
tip **goterici;
```

Gördüyümüz kimi adi göstəricidən fərqli olaraq ikiqat göstəricini elan edərkən onun adının əvvəlinə bir deyil, iki ulduz simvolu qoyuruq.

İkiqat göstəriciləri də, birqat göstəricilər kimi adi dəyişənlərlə eyni elan sətrində elan etmək olar.

Bəs ikiqat göstəricilərlə birqat göstəricilərin fərqi nədən ibarətdir?

Əgər birqat göstəricilər özlərində adi dəyişənlərin ünvanın yadda saxlayırdısa, ikiqat göstəricilər özlərində birqat göstəricilərin ünvanı yadda saxlayır.

Gəlin ikiqat göstəricilərə aid proqram nümunəsi ilə tanış olaq.

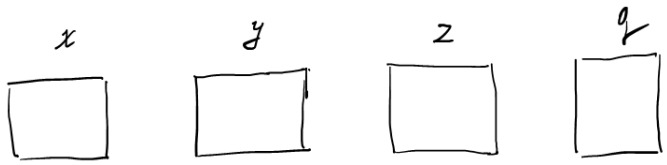
Nümunə:

```
#include <iostream>

using namespace std;

int main (){

    int x, y, *z, **q;

    //x ve y deyishenlerine muxtelif qiymetler menimsedek
    x = 9;
    y = 12;

    //z -te x-in unvanin menimsedek
    z = &x;

    //q -ye z-in unvanin menimsedek
```

```
    q = &z;

    //z gostericisi ile x-in qiymetin deyishek
    *z = 27;

    //q ikiqat gostericisi ile z-te y-in unvanin menimsedek
    *q = &y;

    //ikiqat istinad operatoru ile z ikiqat gostericisi ile
    // y-in qiymetin deyishek
    **q = 90;


}
```

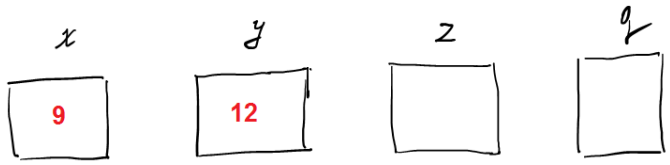
İzahı: int tipindən x və y dəyişənləri, z göstəicisi və q ikiqat göstəricisi elan edirik:

```
int x, y, *z, **q;
```



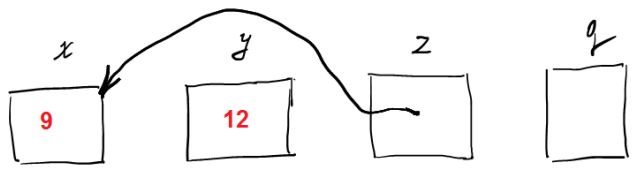
Daha sonra x və dəyişənlərinə qiymətlər mənimsədirik:

```
x = 9;
y = 12;
```



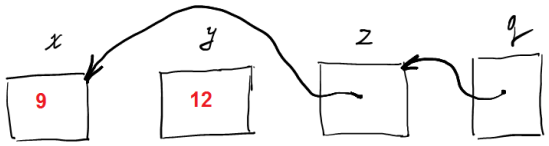
Növbəti sətirdə z göstəricisinə x dəyişəninin ünvanın mənimsədirik:

```
z = &x;
```



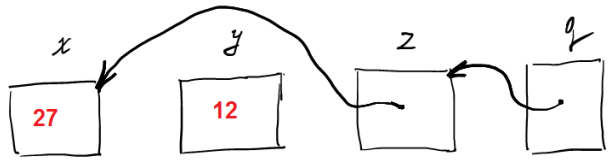
q ikiqat göstəricisinə isə z göstəricisinin ünvanın mənimsədirik:

```
q = &z;
```



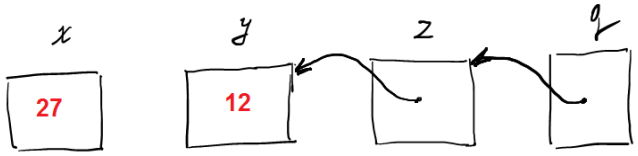
z göstəricisi özündə x –in ünvanın yadda saxladığından aşağıdakı kod x-in qiymətin dəyişir:

```
*z = 27;
```



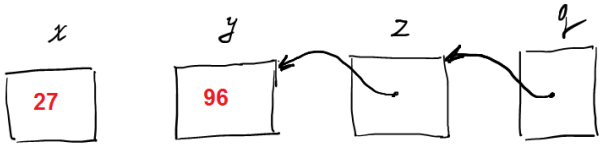
q ikiqat göstəricisi özündə z göstəricisinin ünvanın yadda saxladığından aşağıdakı kod z göstəricisinin qiymətini dəyişərək ona y dəyişəninin ünvanın mənimsədər6

```
*q = &y;
```



İkiqat istinad operatoru vastəsilə q ikiqat göstəricisindən istifadə edərək y-ə qiymət mənimsədək:

```
**q = 90;
```



8.6 Dinamik yaradılma

Göstəricilərə məxsus əsas özəlliklərdən biri də onlar vastəsilə dinamik olaraq, yəni proqram icra olunan zaman yaddaşda yer ayırmaq və istifadə etdikdən sonra sonra həmin ayrılmış yeri azad etmək olar. Bunun üçün C++ dilində müvafiq olaraq new və delete əmrlərindən istifadə olunur.

8.6.1 new əmri

new vastəsilə yaddaşda dinamik yer ayırmaq üçün və həmin yaddaş sahəsini verilmiş göstəriciyə mənimsətmək üçün aşağıdakı sintaksisdən istifadə edirlər:

göstərici = new tip;

Bu zaman qeyd olunan tipli dəyişən üçün yaddaşda yer ayrılacaq və göstəriciyə mənimsədiləcək.

New əmri ilə bir dəyişən üçün yox, cərgə üçün də yer ayıra bilərik. Bunun üçün tipdən sonra kvadrat mötərizələr içində dinamik yaradılan cərgənin elementlərinin sayını göstərməliyik:

göstərici = new tip[say];

Nümunə koda baxaq:

Nümunə:

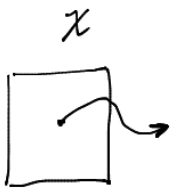
```
#include <iostream>

using namespace std;

int main (){


    int *x;

    x = new int;

    *x = 21;

    cout   << "x = "  << x  << "\n"
           << "*x = " << *x;

    delete x;

}
```

İcra:

```
x = 00558008
*x = 21
```

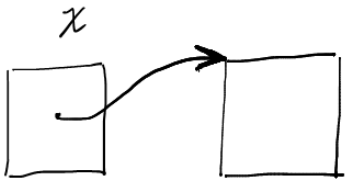
İzahı: Əvvəlcə int tipli x göstəricisi elan edirik:

```
int *x;
```



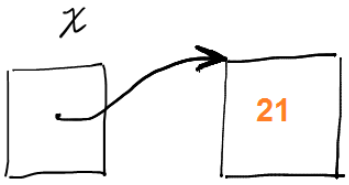
Daha sonra new operatoru ilə x göstəricisi üçün yaddaşda yer ayırırıq:

```
x = new int;
```



Növbəti kod x –in istinad elədiyi yerə 21 qiyməti yazır:

```
*x = 21;
```

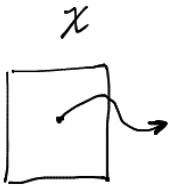


Daha sonra x –in istinad elədiyi ünvanı və həmin ünvandakı qiyməti çap edirik.

```
cout   << "x = "  << x  << "\n"
       << "*x = " << *x;
```

Sonda isə x göstəricisi üçün ayrılan yeri azad edirik:

```
delete x;
```



8.6.2 Dinamik cərgə yaratmaq

Qeyd elədik ki new operatoru ilə yaddaşda yalnız bir dəyişən üçün yox, tələb olunan sayda elementdən ibarət cərgə üçün də yer ayıra bilərik. Bu zaman elementlərin sayını tip –dən sonra kvadrat mötərizə içində göstərməliyik.

göstərici = new tip[say];

Dinamik cərgə üçün ayrılan yeri azad etmək üçün isə yazmalıyıq:

delete[] göstərici;

Burada delete əvəzinə delete[] yazmağımız adi dəyişən üçün deyil, cərgə üçün ayrılan yeri silməli olduğumuzu bildirir.

Nümunə koda baxaq:

```
#include <iostream>

using namespace std;

int main (){

      int i,*x;

      x = new int[5];

      x[0] = 23;
      x[1] = 12;
      x[2] = 34;
      x[3] = 96;
      x[4] = 78;

      for (i=0; i<5; ++i)
            cout << "x[" << i << "] = "  << x[i] << " \t" ;

      delete[] x;

}
```

Nəticə:

```
x[0] = 23       x[1] = 12       x[2] = 34       x[3] = 96       x[4] = 78
```

Burada biz `x = new int[5];` sətri ilə int tipli 5 elementdən ibarət cərgə yaradırıq və x göstəricisinə həmin cərgənin ilk elementinin ünvanın mənimsədirik. Daha sonra cərgənin elementlərinə müxtəlif qiymətlər mənimsədirik və onları çap edirik. Sonda isə əmri ilə `delete[] x;` yeni yaradılan cərgə üçün ayrılan yeri azad edirik.

8.7 İkiqat Göstəricilər və İkiölçülü Cərgələr

Göstəricilərlə Cərgələrin əlaqəsi ilə tanış olduq. İndi isə ikiqat göstəricilərlə, ikiölçülü cərgələrin əlaqəsini örgənək. Hər bir ikiölçülü cərgəyə həmin tipdən olan ikiqat göstərici kimi baxmaq olar. Tutaq ki, aşağıdakı kimi int tipli ikiölçülü x cərgəsi və ikiqat y göstəricisi elan etmişik.

```
int x[5][5], **y;
```

x və y hər ikisi ikiqat göstərici olduğundan yaza bilərik.

```
y = x;
```

Bu mənimsətmədən sonra `y[2][3]` ilə x cərgəsinin müvafiq elementinə müraciət edə bilərik (`x[2][3]`).

Çalışmalar:

Çalışma 1. int tipli x dəyişəni elan edən və onun ünvanını ekranda çap edən proqram qurun.

Çalışma 2. int tipli x dəyişəni və y göstəricisi elan edən proqram qurun. x dəyişəninə hər hansı qiymət, y göstəricisinə isə x-in ünvanın mənimsədin. y göstəricisindən istifadə edərək x dəyişəninin qiymətini və ünvanını çap edin.

Çalışma 3. int tipli x dəyişəni və y göstəricisi elan edən proqram qurun. x dəyişəninə hər hansı qiymət, y göstəricisinə isə x-in ünvanın mənimsədin. y göstəricisindən istifadə edərək x dəyişəninin qiymətini və ünvanını çap edin. y göstəricisindən istifadə edərək x dəyişəninə ayrı qiymət mənimsədin. x dəyişəninin yeni qiymətini çap edin.

Çalışma 4. int tipli x və z dəyişənləri və y göstəricisi elan edən proqram qurun. x və z dəyişənlərinə müxtəlif qiymətlər mənimsədin və onları çap edin. y göstəricisi ilə x dəyişəninə z dəyişəninin qiymətini mənimsədin. Hər iki dəyişənin qiymətlərini yenidən çap edin.

Çalışma 5. int tipli 5 elementdən ibarət x dəyişəni və y göstəricisi elan edən proqram qurun. x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədin. y göstəricisinə x cərgəsinin ilk elementinin ünvanını mənimsədin. y göstəricisindən istifadə edərək x cərgəsinin ilk elementinin ünvanını və qiymətini çap edin.

Çalışma 6. int tipli 5 elementdən ibarət x dəyişəni və y göstəricisi elan edən proqram qurun. x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədin. y göstəricisinə ardıcıl olaraq x cərgəsinin hər-bir elementinin ünvanını mənimsədin. y göstəricisindən istifadə edərək x cərgəsinin hər-bir elementinin ünvanını və qiymətini çap edin.

Çalışma 7. int tipli 5 elementdən ibarət x dəyişəni və y göstəricisi elan edən proqram qurun. y göstəricisindən istifadə etməklə x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədin və onları çap edin.

Çalışma 8. int tipli 5 elementdən ibarət x dəyişəni və y göstəricisi elan edən proqram qurun. x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədin. x cərgəsinin elementlərinin ünvanlarını və qiymətlərini çap edin. y göstəricisinə x cərgəsinin ikinci elementinin ünvanın mənimsədin. y göstəricisindən istifadə edərək x cərgəsinin ikinci elementinin ünvanını və qiymətini çap edin. y göstəricisinin qiymətini 1 vahid artırın. y göstəricisinin istinad elədiyi ünvanı və həmin ünvandakı qiyməti çap edin. Nəticələri x cərgəsinin 3-cü elementinin ünvanı və qiyməti ilə müqaisə edin.

Çalışma 9. int tipli 5 elementdən ibarət x dəyişəni və y göstəricisi elan edən proqram qurun. x cərgəsinin elementlərinə müxtəlif qiymətlər mənimsədin. x cərgəsinin elementlərinin ünvanlarını və qiymətlərini çap edin. y göstəricisinə x cərgəsinin dördüncü elementinin ünvanın mənimsədin. y göstəricisindən istifadə edərək x cərgəsinin dördüncü elementinin ünvanını və qiymətini çap edin. y göstəricisinin qiymətini 2 vahid azaldın. y göstəricisinin istinad elədiyi ünvanı və həmin ünvandakı qiyməti çap edin. Nəticələri x cərgəsinin 2-ci elementinin ünvanı və qiyməti ilə müqaisə edin.

§9 Funksiyalar

9.1 Funksiyaların tərtib olunması

Funksiyalar C++ proqralarının ayrılmaz hissəsidir. Kiçikhəcmli proqramlarda funksiyaların əhəmiyyəti elə də ciddi nəzərə çapmasa da, irihəcmli proqram layihələrinin tərtibi zamanı funksiyalar xüsusi əhəmiyyət daşıyır.

Funksiyaların ən əsas özəllikləri odur ki, onların vastəsilə proqram kodunu bir neçə kiçik hissəyə bölmək və hər bir hissəni ayrıca kodlaşdırmaq olar. Daha sonra ayrı-ayrı funksiyalar birləşərək yekun kodu əmələ gətirir.

9.1.1 Funksiyaya aid numunə

C++ dilində istənilən kodu funksiya şəklində tərtib edə bilərik. Baxacağımız ilk sadə nümunə funksiya iki ədədin cəmlənməsi üçün istifadə olunur.

```
int cem( int x, int y) {
	int z;
	z = x + y;
	return z;
}
```

C++ dilində bütün funksiyalar yuxarıda göstərdiyimiz nümunə funksiya ilə eyni qaydada tərtib olunur. Gəlin baxdığımız nümunə əsasında bu qaydalar ilə tanış olaq.

Hər bir funksiya elan sətrindən və kod hissəsindən ibarət olur. Baxdığımız funksiyanın elan sətir:

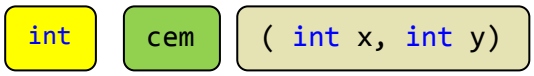
```
int cem( int x, int y)
```

, kod hissəsi isə

```
{
	int z;

	z = x + y;

	return z;
}
```

kimidir.

Elan sətri özü də 3 hissədən ibarət olur:



1) funksiyanın tipi
2) funksiyanın adı
3) funksiyanın parametrləri

9.1.2 Funksiyanın Tipi

C++ dilində hər-bir funksiya müəyyən bir əməliyyat icra etdikdən sonra aldığı qiyməti nəticə olaraq qaytara *bilər*. Bu zaman funksiyanın qaytarmalı olduğu nəticənin tipi funksiyanın elanı sətrində birinci olaraq qeyd edilir.

Baxdığımız nümunədə funksiyanın tipi olaraq **`int`** göstərilib. Bu o deməkdir ki, funksiya **`int`** tipli nəticə qaytarmalıdır. Eyni qayda ilə funksiya `double`, `char`, `bool` v.s. kimi standart tiplərdən, o cümlədən proqramçının özünün tərtib etdiyi yeni tiplərdən nəticə qaytara bilər.

Əgər funksiya sadəcə müəyyən əməliyyatları yerinə yetirmək üçün tərtib olunursa və heç bir nəticə qaytarması tələb olunmursa bu zaman funksiyanın tipi olaraq `void` – yəni boş tip yazılmalıdır.

9.1.3 Funksiyanın Adı

Funksiyanın elanı zamanı onun tipini qeyd etdikdən sonra funksiyanın adını yazmalıyıq. Funksiyalara ad verərkən dəyişənlərin adlandırılması qaydaları əsas tutulur(hərf ilə başlama, yalnız igilis əlifbası simvollarından istifadə v.s.). Çalışmaq lazımdır ki, funksiyalara verdiyimiz adlar onların gördüyü işə uyğun olsun. Baxdığımız nümunə funksiyaya elan sətrindən gördüyümüz kimi ***cem*** adı vermişik.

Yanlış adlar:

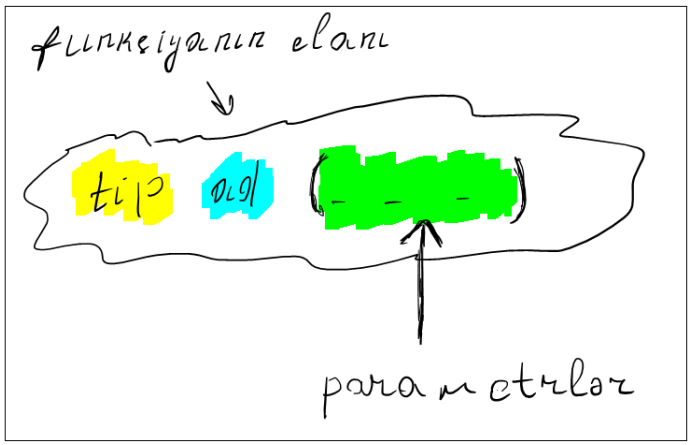
```
34_iki_ededin_cemi // reqem ile başlayır
enBoyuk%Sahe       // % simvolundan istifadə olunur
cergeni_Cap Et     // məsafə simvolu var
```

Düzgün adlar:

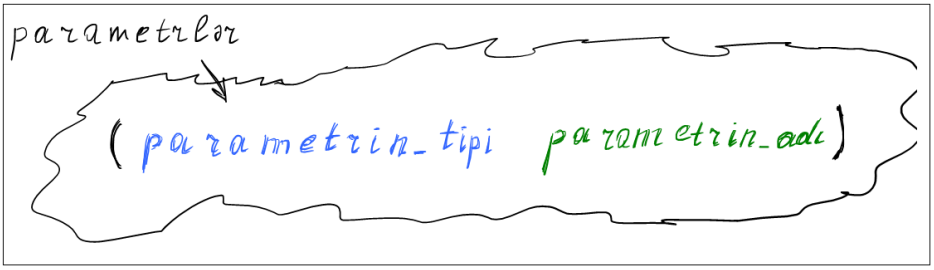
```
iki_ededin_cemi
enBoyuk%$ashe
cergeni_CapEt
```

9.1.4 Funksiya parametrləri

Funksiyanın elanı sətri ilə tanış olarkən qeyd etdik ki,funksiyanın adından sonra mötərizə daxilində onun parametrləri qeyd olunur.



Funksiyanın parametrləri funksiya çağırılan zaman ona ötürülməli olan məlumatları bildirir. Parametrlər siyahısına hər-hansı dəyişəni yerləşdirmək üçün əvvəlcə onun tipini, daha sonra adını yazmalıyıq.



Misal üçün əgər funksiyaya int tipli x dəyişənini parametr kimi ötürmə istəyiriksə bu zaman mötərizə arasına yazmalıyıq: `int x`

```
(int x)
```

Bəs funksiyaya birdən çox sayda parametr ötürə bilərikmi?
Bəli, hər bir parametrin tipini və adını qeyd etdikdən sonra vergül qoyub, növbəti parametri daxil edə bilərik. Misal üçün aşağıdakı **enb** adlı funksiya iki parametr qəbul edir: `double` tipli **max** və `char` tipli **s** parametrlərini:

```
void enb (double max, char s)
```

Əgər o `int` tipli **r** adlı 3-cü parametr də qəbul etsəydi onda elan aşağıdakı kimi olardı:

```
void g (double max, char s, int r)
```

Gördüyümüz kimi parametrlərin elanı dəyişənlərin elanına oxşasa da, dəyişənlərin elanından fərqli olaraq hərbir parametrin tipi mütləq *ayrıca* qeyd olunmalıdır. Hər bir parametr qeyd olunduqdan sonra (tipi və adı) vergül qoyub növbəti parametrə keçə bilərik.

Səhv: `( int x, y)`
Səhv: `( int x; int y)`

Düzgün: `( int x, int y)`

Ola bilər ki funksiya heç bir parametr qəbul etməsin. Bu zaman funksiyanın adından sonra içi boş mötərizələr qoymaq lazımdır - (). Parametrlər yoxdu deyib mötərizələri də yazmamaq sintaksis səhv sayılır.

Funksiyaların elanına aid bəzi çalışmalar həll edək.

9.1.5 C++ dilində funksiyaların elan olunmasına aid çalışmalar.

Çalışma 1. Heç bir nəticə qaytarmayan və heç bir parametr qəbul etməyən **ahmed** adlı funksiya elan edin.

Həlli: Funksiyanın adı ahmed alduğuna görə, ilk öncə onun adını qeyd edə bilərik:

ahmed

ahmed funksiyası heç bir nəticə qaytarmadığına görə tip olaraq `void` yazmalıyıq. Bilirik ki, funksiyanın qaytardığı tip adının qarşısına yazılır, ona görə belə olar:

```
void ahmed
```

Tip ilə adı yazdıq, qaldı parametrlər. Parametrləri yazmaq üçün addan sonra mötərizə qoyuruq:

```
void ahmed ()
```

Funksiya heç bir parametr qəbul etmədiyinə görə mötərizələrin arasına heçnə yazmırıq:

```
void ahmed ()
```

Çalışma 2. `int` tipli nəticə qaytaran və heç bir parametr qəbul etməyən **ali** adlı funksiya elan edin.

Həlli: `int ali ( )`

Çalışma 3. `double` tipli nəticə qaytaran və heç bir parametr qəbul etməyən f adlı funksiya elan edin.

Həlli: `double f ( )`

Çalışma 4. Heç bir nəticə qaytarmayan, `int` tipli x, `char` tipli s və `int` tipli q parametrləri qəbul edən enB adlı funksiya elan edin.

Həlli: `void enB ( int x, char s, int q )`

Funskyaların elanına aid müxtəlif nümunələr ilə tanış olduq, indi isə funksiyaların bədən hissəsi ilə məşğul olaq.

9.1.6 Funksiyanın bədəni

Funksiyanı elan etdikdən sonra onun bədən hissəsini tərtib etməliyik. Funksiyanın bədən hissəsində biz funksiyanın görməli olduğu işi kodlaşdırırıq. Burada indiyə kimi örgəndiyimiz istənilən proqram kodu qura bilərik, təkcə yeni funksiyaların elanından savayı. Yəni funksiya daxilində funksiya elan etmək olmaz.

Funksiyanın bədəni bütövlükdə { } – mötərizələri arasında yerləşir. Bu mötərizələr arasına heç nə yazmaya da bilərik, { } - şəklində. Bu halda funksiya heç-bir iş görməyəcək. Lakin istənilən halda, funksiya heç-bir iş görmədiyi halda belə, elandan sonra {} mötərizələrini mütləq qoymalıyıq. Əks halda kompilyator səhv mesajı verəcək.

Çalışmalar həlli ilə məşğul olaq.

Çalışma 4. Heç bir iş görməyən, işərisi boş olan funksiya bədəni tərtib edin.

Həlli:

{}

Çalışma 5. Ekranda `"Salam dunya"` sətrini çap edən funksiya bədəni tərtib edin.

Həlli:

```
{
	cout << "Salam dunya";
}
```

Çalışma 6. `int` tipli x dəyişəni elan edən və bu dəyişənə istifadəçinin daxil etdiyi qiyməti mənimsədən funksiya bədəni tərtib edin.

Həlli:

```
{
	int x;

	cout << "Zehmet olmasa her - hansi eded daxil edin\n";
	cin >> x;
}
```

9.1.7 Funksiyanın elanı ilə funksiya bədəninin birləşdirilməsi

Funksiyanın elanını bədəni ilə birləşdirdiyimiz zaman funksiya kodu tam şəkildə tərtib olunmuş hesab olunur.

Çalışma 7. Heç bir parametr qəbul etməyən və heç bir nəticə qaytarmayan, icra olunan zaman ekranda "Heey men icra olunuram..." sətrini çap edən test adlı funksiya tərtib edin.

Həlli. Çalışmanın tələblərinə əsasən funksiyanın elanı

```
void test()
```

şəklində, bədəni isə

```
{
	cout << "Heey men icra olunuram ...";
}
```

şəklində olar. Bu ikisini birləşdirsək, aşağıdakı kimi test funksiyası alınar:

```
void test() {
	cout << "Heey men icra olunuram ...";
}
```

9.2 Funksiyanın çağrılması

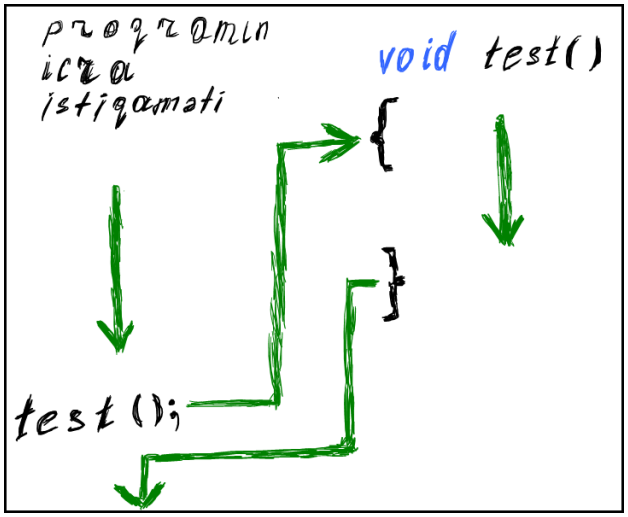
Funksiyanı tərtib etdikdən sonra biz ondan proqram daxilində istənilən yerdə, o cümlədən digər funksiyaların daxilində də istifadə edə bilərik. Bunun üçün funksiyanı "***çağırmaq***" tələb olunur. Funksiyanı çağırmaq olduqca sadədir!
İstənilən bir şeyin çağrılmasında olduğu kimi funksiyanı çağıran zaman da onun adından istifadə edirik. Daha dəqiq desək: funksiyanı çağırmaq üçün əvvəlcə onun adını yazırıq, daha sonra () mötərizələrini mütləq qoyuruq və çağırmanı ; nöqtəvergül ilə tamamlayırır. Bu qədər bəsit.

Misal üçün çalışma 7-də tərtib etdiyimiz **test** funksiyasını çağırmaq üçün yazmalıyıq:

```
 // test funksiyasının çağrılması
test();
```

Nəticədə nə baş verir?

Nəticədə icra olunma funksiya çağrılan yerdən funksiyanın bədəninə ötürülür. Funksiyanın bədən kodu icra olunduqdan sonra yenidən icraolunma funksiya funksiya çağırıldığı əvvəlki yerinə qayıdır və funksiyanın çağırılmasından sonra gələn əməliyyat icra olunur. Sxematik olaraq bunu aşağıdakı kimi təsvir edə bilərik:



Gəlin funksiyanı nə cür tərtib etməyi və onu çağırmağı proqram kodu ilə göstərək. Əvvəlcə C++ dilində bildiyimiz sadə bir proqram tərtib edək.

```
#include <iostream>

using namespace std;


//C++ dilinde sade proqram
int main() {

}
```

Bizə indiyə qədər keçdiyimiz dərslərdən çox yaxşı tanış olan bu proqram kodu əslində artıq anlaya bildiyimiz kimi özü də elə funksiyadır, həm də proqramın ən əsas funksiyası.
Bütün C++ proqramlarında main funksiyası olmalıdır və hər bir proqram icra olunmağa main funksiyasından başlayır. Daha doğrusu main funksiyasının açan mötərizəsindən **{** icra olunmağa başlayır və bağlayan mötərizəsi **}** ilə başa çatır.

Proqramımızı icra eləsək o heç bir iş görməz, çünki gördüyümüz kimi main –in **{ }** mötərizələri arasına heçnə yazmamışıq.

İndi proqramımıza yuxarıda tərtib etdiyimiz test funksiyasını əlavə edək.

```
#include <iostream>

using namespace std;
```

```
//test funksiyasi
void test() {

    cout << "Heey men icra olunuram ...";

}


//C++ dilinde sade proqram
int main() {

}
```

Bəs indi proqramı icra eləsək nə nəticə olar? Yenə heçnə, ona görə ki, proqrama yeni kod əlavə etməyimizə baxmayaraq, main –in içi yenə də boşdur və nəqədərki main –in içinə bir şey yazmamışıq, başqa yerə nə yazırıq yazaq fərq eləməz. Proqram heçnə icra eləməyəcək. O cümlədən proqrama yeni əlavə etdiyimiz test funksiyasın.

İndi isə gəlin test funksiyasın icra eliyək. Bilirik ki, bunun üçün onu çağırmalıyıq. test funksiyasın çağırmaq üçün main –in daxilinə test(); sətrini əlavə etməliyik, aşağıdakı kimi:

```
//C++ dilində sadə proqram
int main() {

      //test funksiyasi cagiraq
      test();

}
```

Tam şəkildə proqram kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

//test funksiyası
void test() {

    cout << "Heey men icra olunuram ...";

}


//C++ dilində sadə proqram
int main() {

    //test funksiyasi cagiraq
    test();

}
```

Bu proqramı icra eləsək əvvəlcə main icra olunmağa başlayacaq, main –test –i çağıracaq və test icra olunacaq. test funksiyası ekranda **"Heey men icra olunuram ..."** sətrini çap edəcək və main -ə qayıdacaq. Proqram başa çatacaq.

9.3 Funksiyanın nəticə qaytarması

Bilirik ki, funksiyanı çağırdığımız zaman o icra olunur və çağrıldığı yerə geri qayıdır. Bundan əlavə funksiyalar həm də gördükləri işin nəticəsini çağrıldıqları yerə geri qaytara bilərlər. Bunun üçün `return` operatorundan istifadə olunur. `return` operatorunun sintaksisi belədir:

```
return  nəticə;
```

Funksiya daxilində bu kod icra olunduqda `return` operatorunun qarşısında yazılan ***"nəticə"*** funksiya çağrılan yerə geri qaytarılır.

Gəlin yuxarıda tərtib etdiyimiz test funksiyasını elə dəyişək ki, o çağrıldığı yerə nəticə olaraq 4 ədədini qaytarsın.

Əvvəlcə test funksiyasının kodunu bir daha yadımıza salaq.

```
//test funksiyası
void test() {

	cout << "Heey men icra olunuram ...";

}
```

Gördüyümüz kimi bu kodda heç bir yerdə `return` əmrindən istifadə olunmur. Deməli bu halda test funksiyası heç bir nəticə qaytarmır. Tələb olunan nəticəni (4 ədədini) qaytarması üçün test funksiyasının sonuna

```
return  4;
```

sətrini əlavə etməliyik. test funksiyasının yeni şəkli aşağıdakı kimi olar:

```
//test funksiyası
void test() {

	cout << "Heey men icra olunuram ...";

	return  4;
}
```

Amma bu şəkildə proqramı kompilyasiya etsək aşağıdakı kimi xəta alınar:

```
1>src\prg.cpp(10): error C2562: 'test': 'void' function returning a value
1>src\prg.cpp(8): note: see declaration of 'test'
```

Kompilyasiya nəticəsində bu cür xəta mesajı almağımızın səbəbi odur ki, funksiyanın qaytardığı nəticənin tipi, onun elanında göstərilən tip ilə eyni deyil. Funksiyanın elanı sətrinə bir daha nəzər yetirək:

```
void test()
```

Yadımıza salaq ki, elan sətrində ilk söz – yəni `void` funksiyanın tipini, daha doğrusu funksiyanın qaytardığı nəticənin tipini göstərir. Yeni test funksiyası `int` tipli nəticə qaytardığından biz `void` əvəzinə `int` yazmalıyıq, aşağıdakı kimi:

```
int test()
```

Beləliklə icra olunan zaman 4 ədədini qaytaran test funksiyasının yekun kodu aşağıdakı kimi olar:

```
//test funksiyası
int test() {

	cout << "Heey men icra olunuram ...";

	return  4;
}
```

Tam proqram kodu isə aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

//test funksiyasi
int test() {

	cout << "Heey men icra olunuram ...\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	//test funksiyasi cagiraq
	test();
```

```
}
```

Əgər bu proqramı kompilyasiya və icra eləsək, onda ekranda `"Heey men icra olunuram ..."` sətri çap olunar.

Bəs funksiyanın qaytardığı nəticə necə oldu?

Baxdığımız nümunədə test funksiyası nəticə qaytarsa da, biz bu nəticədən istifadə eləmədik. Bu cür hallarda həmin nəticə sadəcə olaraq proqram tərəfindən inkar olunur. Funksiyanın nəticəsini müxtəlif cür istifadə edə bilərik, misal üçün hər-hansı dəyişənə mənimsədə bilərik. test funksiyasının qaytardığı nəticəni əldə etmək üçün funksiyanın tipinə uyğun hər-hansı bir dəyişən elan edib funksiyanı həmin dəyişənə mənimsətməliyik. Funksiyanın tipi `int` olduğuna görə əvvəlcə `int` tipindən har-hansı y dəyişəni elan edək.

```
int y;
```

Indi isə əsas məqam, test funksiyasının qaytardığı nəticəni y-ə mənimsətmək üçün yazmalıyıq:

```
y = test();
```

Bu sətir icra olunan zaman əvvəlcə test funksiyası çağırılır, yerinə yetirilir və qaytardığı nəticə (4 ədədi) y dəyişəninə mənimsədilir.
Bundan sonra y –in qiymətini çap edə bilərik. Kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

//test funksiyasi
int test() {

	cout << "Heey men icra olunuram ...\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	int y;

	//test funksiyasi cagiraq
	y = test();

	cout << "y -in qiymeti = " << y;
```

```
}
```

Nəticə:

```
Heey men icra olunuram ...
y -in qiymeti = 4
```

Nümunə test funksiyasında nəticə olaraq 4 ədədini qaytardıq, lakin C++ dili funksiyanın qaytardığı nəticəyə heç bir məhdudiyyət qoymur. Biz funksiyalardan tam, kəsr, simvol, göstərici və gələcəkdə örgənəcəyimiz sətir, strukt və sinif tiplərindən də nəticə qaytara bilərik.

Funksiyaların qaytardığı nəticədən istifadəyə aid proqram çalışmaları ilə məşğul bir qədər sonra ətraflı məşğul olacayıq. Lakin daha maraqlı proqramlar qura bilmək üçün əvvəlcə funksiya parametrlərindən istifadə etməyi örgənməliyik.

9.4 Funksiyaya parametr ötürülməsi

test funksiyası üzərində parametrlərin izahını davam edək. Test funksiyasını elə dəyişək ki, o `int` tipli x parametri qəbul eləsin. Parametrlərin elanı ilə yuxarıda tanış olmuşdur. Tələb olunan test funksiyasının elanı aşağıdakı kimi olar:

```
//test funksiyasi
int test( int x ) {

	cout << "Heey men icra olunuram ...\n";

	return  4;
}
```

test funksiyasını yeni formada proqramda yazaq:

```
#include <iostream>

using namespace std;

//test funksiyasi
int test( int x ) {

	cout << "Heey men icra olunuram ...\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	int y;
```

```
	//test funksiyasi cagiraq
	y = test();

	cout << "y -in qiymeti = " << y;
}
```

Əgər biz bu şəkildə test funksiyasını proqrama yerləşdirib, kompilyasiya etsək, aşağıdakı kimi kompilyasiya xətası alacayıq:

```
1>prg.cpp(12): error C2660: 'test': function does not take 0 arguments
1>prg.cpp(17): error C2084: function 'int test(int)' already has a body
1>  prg.cpp(602): note: see previous definition of 'test'
```

Bu cür xəta mesajı almağımızın səbəbi funksiyanı yanlış çağırmağımızdır. test funksiyasını proqramda hazırda bu şəkildə çağırırıq:

```
y = test();
```

Bu cür çağırış, test funksiyasının əvvəlki variantında doğru çağırış idi, belə ki, əvvəlki test funksiyası heç bir parametr qəbul etmirdi. Laki yeni test funksiyası əvvəlkindən fərqli olaraq artıq bir dənə int tipli parametr qəbul etdiyindən, funksiyanı çağıran zaman bu nəzərə alınmalıdır. Ümumiyyəltə isə funksiyanı çağıran zaman aşağıdakı qaydaya ciddi əməl olunmalıdır:

Qayda: Funksiyanın elanında neçə **parametr** göstərilibsə, funksiyanı çağıran zaman ona ötürlən **məlumatların** sayı və tipləri eyni olmalıdır.

test funksiyasının elanına bir daha nəzər yetirək:

```
int test( int x )
```

Yeni elana görə o artıq bir dənə tipi `int` olan parametr qəbul edir. test funksiyasını düzgün çağırmaq üçün biz onun mötərizələri arasına `int` tipli məlumat yazmalıyıq, misal üçün 25 ədədi, aşağıdakı kimi:

```
y = test(25);
```

Bu halda test funksiyasının çağırışı onun elanına uyğun gəlir. Diqqət yetirək və yadda saxlayaq ki, funksiyanı çağıran zaman ona ötürülən məlumat tipi elanda olduğu kimi parametrin qarşısında yazılmır, yalnız məlumatın özünü yazırıq. Yəni aşağıdakı kimi yazılış səhvdir:

```
y = test(int 25);
```

Yekun kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

//test funksiyasi
int test(int x) {

	cout << "Heey men icra olunuram ...\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	int y;

	//test funksiyasi cagiraq
	y = test(25);

	cout << "y -in qiymeti = " << y;
}
```

Biz test funksiyasını çağırdıq və ona parametr olaraq 25 ədədini ötürdük. Lakin bu 25 ədədinin başına nə iş gəldiyi barədə bir məlumatımız olmadı. Başqa sözlə desək, funksiyaya ötürülən parametrdən funksiya daxilində heç bir yerdə istifadə etmədik. Indi isə gəlin funksiyaya ötürülən məlumatın funksiya daxilində nə cür istifadə oluna bilməsi ilə tanış olaq.

9.5 Funksiya parametrindən istifadə

Funksiya parametrindən funksiya daxilində adi dəyişənlər kimi istifadə edə bilərik. Misal üçün test funksiyası daxilində x parametrinin qiymətini çap edə bilərik, aşağıdakı kimi:

```
cout << "x parametrinin qiymeti = " << x << "\n";
```

Tam kod bu şəkildə olar:

```
#include <iostream>

using namespace std;
//test funksiyasi
int test(int x) {
```

```
	cout << "Heey men icra olunuram ...\n";
	cout << "x parametrinin qiymeti = " << x << "\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	int y;

	//test funksiyasi cagiraq
	y = test(25);

	cout << "y -in qiymeti = " << y;
}
```

Bu proqramı icra eləsək, aşağıdakı kimi nəticə alarıq:

```
Heey men icra olunuram ...
x parametrinin qiymeti = 25
x -in qiymeti = 4
```

Biz proqram daxilində test funksiyasın istənilən dəfə çağıra və ona tam tipdən olan bir ədəd istənilən məlumat ötürə bilərik, aşağıdakı kimi:

```
	test(45);
	test(67);
	test(1);
```

Test funksiyası ona ona ötürülən məlumatı x parametrinin qiyməti olaraq qəbul edəcək və onun qiymətini çap edəcək. Yuxarıdakı sətirləri proqram kodunda nəzərə alsaq alaq:

```
#include <iostream>

using namespace std;

//test funksiyasi
int test(int x) {

	cout << "Heey men icra olunuram ... \n";

	cout << "x parametrinin qiymeti = " << x << "\n\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {
```

```
	test(45);
	test(67);
	test(1);

}
```

Bu proqramı icra eləsək, aşağıdakı kimi nəticə alarıq:

```
Heey men icra olunuram ...
x parametrinin qiymeti = 45

Heey men icra olunuram ...
x parametrinin qiymeti = 67

Heey men icra olunuram ...
x parametrinin qiymeti = 1
```

9.6 Dəyişənlərin funksiyaya məlumat kimi ötürülməsi

Yuxarıdakı nümunədə test funksiyasına 45, 67 v.s. ədədləri ötürüldü. Lakin test funksiyasına məlumat olaraq dəyişən də ötürə bilərik. Sadəcə ötürülən dəyişənin tipi funksiyanın müvafiq parametrinin tipi ilə eyni olmalıdır. Aşağıdakı nümunəyə baxaq.

```
#include <iostream>

using namespace std;

//test funksiyasi
int test(int x) {

	cout << "Heey men icra olunuram ... \n";

	cout << "x parametrinin qiymeti = " << x << "\n\n";

	return  4;
}


//C++ dilinde sade proqram
int main() {

	//int tipli a deyisheni elan edek
	int a;

	// a deyishenine qiymet menimsedek
	a = 17;

	//a deyishenin test funksiyasina oturek
	test(a);

}
```

Bu proqramı icra eləsək aşağıdakı nəticəni alarıq:

```
Heey men icra olunuram ...
x parametrinin qiymeti = 17
```

9.7 Funksiyalara aid çalışmalar

Çalışma 1. **sh** adlı elə funksiya tərtib edin ki, int tipli parametr qəbul eləsin və qəbul etdiyi məlumatı olduğu kimi nəticə olaraq qaytarsın.

Həlli. Kod aşağıdakı olar:

```
#include <iostream>

using namespace std;

int sh(int x) {

	return  x;
}

int main() {

	int y;

	y = sh(5);

	cout << "y = " << y ;
}
```

Nəticə:

```
y = 5
```

Çalışma 2. **sg** adlı elə funksiya tərtib edin ki, int tipli parametr qəbul eləsin və qəbul etdiyi məlumatın qiymətini 1 vahid artıraraq nəticə qaytarsın.

Həlli. Kod aşağıdakı olar:

```
#include <iostream>

int sh(int x) {

	return  x + 1;
}

int main() {

	int y;
```

```
	y = sh(5);

	cout << "y = " << y ;
}
```

Nəticə:

```
y = 6
```

Çalışma 3. **sr** adlı elə funksiya tərtib edin ki, int tipli iki parametr qəbul eləsin və qəbul etdiyi məlumatların qiymətlərinin cəmini nəticə olaraq qaytarsın.

Həlli. Kod aşağıdakı olar:

```
#include <iostream>

using namespace std;

int sr(int x, int z ) {

	return  x + z;
}

int main() {

	int y;

	y = sh(5, 12);

	cout << "y = " << y ;
}
```

Nəticə:

```
y = 17
```

Çalışma 4. Qəbul etdiyi int tipli iki parametrin hasilini nəticə olaraq qaytaran hsl adlı funksiya tərtib edin. hsl funksiyasından istifadə etməklə istifadəçinin daxil etdiyi iki ədədin hasilini hesablayıb çap edən proqram qurun.

Həlli. Kod aşağıdakı olar:

```
#include <iostream>

using namespace std;

int hsl(int x, int z) {

	return  x * z;
```

```
}

int main() {

	int x, y, z;

	cout << "Birinci ededi daxil edin \n";
	cin >> x;

	cout << "Ikinci ededi daxil edin \n";
	cin >> y;

	z = hsl(x, y);

	cout << x << " * " << y << " = " << z;
}
```

Nəticə:

```
Birinci ededi daxil edin
45
Ikinci ededi daxil edin
78
45 * 78 = 3510
```

Sual: Yuxarıdakı nümunədə biz hsl funksiyasının parametrləri olaraq x və y göstərdik. Daha sonra main funksiyasında da eyni adlı dəyişənlər elan elədik. Bəs eyni adlı dəyişəni proqramda müxtəlif yerlərdə elan etməyə icazə varmı?

Bu suala irəlidə örgənəcəyimiz Lokal və qlobal dəyişənlər mövzusunda cavab verəcəyik. Hələlik isə qısa olaraq qeyd edək ki, müxtəlif funksiyalar daxilində elan olunan dəyişənlər ***"bir-birini görmürlər"***. Ona görə müxtəlif funksiyaların daxilində eyni adlı dəyişənlər elan edə bilərik və onlar bir-birləri ilə konflikt yaratmaz.

Çalışma 5. Qəbul etdiyi iki ədəddən hansının böyük olduğunu müəyyən edən enb adlı funksiya tərtib edin. Bu funksiyadan istifadə etməklə istifadəçinin daxil etdiyi iki ədədin ən böyüyünü müəyyən edən proqram qurun.

Həlli. Əvvəlki dərslərimizdə bu cür çox çalışmalar etdiyimizdən detallar üzərində çox dayanmayacayıq. Əsas diqqətimizi məsələnin funksiyalar vastəsilə nə cür realizə olunmasına yönəldəcəyik. Nümunə kod aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int enb(int x, int y) {

	if (x > y)
```

```
            return x;
      else
            return y;

}

int main() {

      int x,y;

      cout << "Zehmet olmasa iki eded daxil edin \n";
      cin >> x >> y;

      cout << x << " ile " << y << " arasinda "
            << "en boyuyu " << enb(x, y) << " -dir";


}
```

Nümunə nəticə:

```
Zehmet olmasa iki eded daxil edin
34 56
34 ile 56 arasinda en boyuyu 56 -dir
```

İzah:

Əgər koda diqqət yetirsəniz biz enb funksiyasının qaytardığı nəticəni yadda saxlamaq üçün main –funksiyasında əlavə dəyişən elan etmədik. Birbaşa cout –da enb() funksiyasını çağırdıq.

Funksiyanı `cout` ilə çağırmaq olar? Əlbəttə - heç bir problem yoxdur. Funksiyanı harda istəsəniz çağıra bilərsiniz, if operatorunun şərtində, for operatorunun sayğaclara qiymət mənimsədən kodunda, mənimsətmə operatorunun sağ tərəfində, hətta riyazi ifadələrin içində, `cout` operatorunda, eləcə də tam sərbəst formada. Nəticədə funksiya icra olunacaq və qaytardığı nəticə (əgər qaytarırsa) funksiyanın işləndiyi yerdə onun əvəzinə yazılacaq. İrəlidəki çalışmalarda funksiyanın çağırılmasına aid müxtəlif nümunələrlə tanış olacayıq.

Çalışma 6. Verilmiş ədədin sadə olub olmadığını müəyyən edən sade adlı funksiya tərtib edin. sade funksiyasından istifadə edərək istifadəçinin daxil etdiyi 5 ədədin sadə ədəd olduğunu müəyyən edən proqram tərtib edin.

Həlli. Əvvəlcə gəlin verilmiş ədədin sadə olmasını müəyyən edən proqram tərtib edək. Bu məsələnin alqoritmik hissəsi bizlərə əvvəlki dərslərimizdən tanış olduğuna görə alqoritm üzərində çox dayanmayacayıq. Nümunə kod belə olar:

```
#include <iostream>

using namespace std;

int sade(int x) {

	int i;

	for (i = 2; i < x; ++i)
		if (x%i == 0)
			return 0;

	return 1;
}

int main() {

	int j,y;

	for (j = 0; j < 5; ++j) {

		cout << "Ededi daxil edin \n";
		cin >> y;

		if (sade(y) == 1)
			cout << y << " sade ededdir \n";
		else
			cout << y << " sade eded deyil \n";

	}

}
```

Nümunə nəticə:

```
Ededi daxil edin
23
23 sade ededdir
Ededi daxil edin
45
45 sade eded deyil
Ededi daxil edin
56
56 sade eded deyil
Ededi daxil edin
78
78 sade eded deyil
Ededi daxil edin
91
91 sade eded deyil
```

İzah: tərtib ediyimiz sade funksiyası qəbul etdiyi parametr sadə ədəd olduqda 1, əks halda 0 qiyməti qaytarır. Biz bunu main funksiyasında `if` operatorunun şərtində nəzərə alırıq, beləki aşağıdakı kod sade funksiyasının qaytardığı nəticəni 1 qiyməti ilə müqaisə edir.

```
	if (sade(y) == 1)
```

Gördüyümüz kimi burada biz funksiyanı `if` operatorunun şərti içərisində çağırırıq. Nəticədə sade funksiyası çağırılır, icra olunur, nəticə qaytarır və qaytardığı nəticə yazılır `if` operatorunun şərtində sade funksiyasının yerinə və şərt yoxlanılır.

Yuxarıdakı çalışmada tərtib etdiyimiz sade funksiyasını biraz da təkmilləşdirə bilərik. Belə ki, sadə funksiyası özü istifadəçidən ədədi daxil etməsini istəyə və sadəliyini müəyyənləşdirdikdən sonra müvafiq məlumat çap edər. Bu halda sade funksiyası öz üzərinə daha çox yük götürdüyündən main funksiyasının yükü xeyli azalar və o daha tez anlaşılan olar.

```
#include <iostream>

using namespace std;

void sade() {

	int i,x ;

	cout << "Ededi daxil edin \n";
	cin >> x;

	for (i = 2; i < x; ++i)
		if (x%i == 0) {
			cout << x << " sade eded deyil \n";
			break;
		}
		else {
			cout << x << " sade ededdir \n";
			break;
		}
}

int main() {

	int i;

	for (i = 0; i < 5; ++i)
		sade();

}
```

Nümunə nəticə

```
Ededi daxil edin
34
34 sade eded deyil
Ededi daxil edin
5
5 sade ededdir
```

```
Ededi daxil edin
6
6 sade eded deyil
Ededi daxil edin
67
67 sade ededdir
Ededi daxil edin
8
8 sade eded deyil
```

Proqramımızı daha da təkmilləşdirək, belə ki o, istifadəçinin daxil etdiyi iki ədəd arasında olan sadə ədədləri çap etsin.

Çalışma 7. Funksiya tərtib etməklə istifadəçinin daxil etdiyi iki ədəd arasında olan bütün sadə ədədləri çap edən proqram tərtib edin.

Həlli. İstifadəçinin daxil etdiyi iki ədəd arasında birincidən başlayaraq ikinciyə kimi dövr qurub, sayğacı sade() funksiyasına parametr kimi ötürməklə tələb olunan proqramı qura bilərik. Əvvəlcə nümunə kod ilə tanış olaq:

```
#include <iostream>

using namespace std;

void sade(int x) {

	int i;

	for (i = 2; i < x; ++i)
		if (x%i == 0)
			return;


	cout << x << "  ";

}

int main() {

	int i,x,y;

	cout << "Iki eded daxil edin \n";
	cin >> x >> y;

	for (i=x; i<y; ++i)
		sade(i);

}
```

9.8 Lokal və qlobal dəyişənlər

C++ dilində dəyişənlərin istifadəsi ilə bağlı bəzi qaydalar mövcuddur. Bunlardan bəziləri ilə biz artıq tanışıq. Dəyişənlərin adlandırılması, elan olnuduqdan sonra müraciət v.s. İndi isə biz dəyişənlərin proqramda hansı yerlərdə qüvvədə olmasını örgənəcəyik. Bunun üçün isə bizə tanış olan blok anlayışını bir daha yada salaq.

Tərif: C++ dilində açan və bağlayan fiqurlu mötərizələrdən **{ }** ibarət kod parçasına **blok** deyirlər.

Əvvəlki dərslərimizdən bilirik ki, dövr, şərt və ya digər hər-hansı mürəkkəb operatorların icra etdikləri əməliyyatların sayı birdən çox olduqda onları fiqurlu mötərizə - blok daxilinə yerləşdirmək lazımdır. İndi isə görürük ki, funksiya bədəninin özü də ayrıca bir blokdan ibarətdir.

Misal üçün tutaq ki, aşağıdakı kimi bir funksiya verilib:

```
void test() {


}
```

Gördüyümüz kimi bu funksiya tək bir blokdan ibarətdir. Şərti olaraq bu bloku A bloku adlandıraq.

```
void test() {
//A bloku


}
```

Tutaq ki, test funksiyası daxilində hər-hansı şərt operatoru yazmışıq ki, onun da daxilində blokdan istifadə edirik:

```
void test() {
//A bloku

	if (şərt) {


	}

}
```

if operatorunun blokunu şərti olaq B bloku adlandıraq.

```
void test() {//A bloku
```

```
        if (şərt) {//B bloku



        }//B blokunun sonu
}//A blokunun sonu
```

Baxılan kod üçün B loku A blokunun daxilində yerləşmiş olur. C++ dilində biz bu şəkildə iç-içə çoxlu blok yerləşdirə bilərik. İndi isə dəyişənlərin qüvvədə olma sahələri ilə bağlı önəmli bir qaydanı daxil edəcəyik.

Qayda: Hər bir dəyişənin qüvvədə olma sahəsi onun elan olunduğu blokun sərhədləri daxilindədir. Yəni heç bir dəyişənə onun elan olunduğu blokdan xaricdə müraciət edə bilmərik. Başqa sözlə blokdan xaricdə qalan kod hissəsi, blokun daxilində elan olunmuş dəyişənləri görmür. Yuxarıdakı nümunə üzərində bu deyilənlərə aydınlıq gətirməyə çalışaq.

Misal üçün tutaq ki, A bloku daxilində hər-hansı x dəyişəni elan etmişik:

```
void test() {
//A bloku

        int x;

        if (şərt) {
        //B bloku



        }

}
```

x dəyişəninin qüvvədə olduğu sahəni sarı fonda verək:

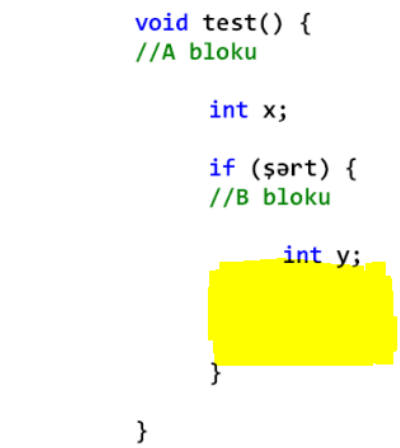
```
void test() {
//A bloku

        int x;

        if (şərt) {
        //B bloku



        }
}
```

Gördüyümüz kimi x dəyişəni A blokunun içərisində elan olunur və elan olunduğu yerdən başlayaraq, A blokunun sonuna kimi qüvvədədir. Bu o deməkdir ki həmin bu hissədə istənilən yerdən x dəyişəninə müraciət edə bilərik. Şəkildən göründüyü kimi, B bloku daxilindən də x dəyişəninə müraciət edə bilərik. x dəyişəni b bloku üçün qlobal sayılır, A bloku üçün isə lokal.

İndi isə gəlin B bloku daxilində int tipli hər-hansı y dəyişəni elan edək :

```
void test() {
//A bloku

	int x;

	if (şərt) {
	//B bloku

		int y;



	}

}
```

y dəyişəninin qüvvədə olduğu sahəni sarı fonda verək:



Gördüyümüz kimi y dəyişəninin qüvvədə olduğu sahə ancaq elan olunduğu yerdən, B blokunun sonuna kimidir. Bu o deməkdir ki, B bloku xaricində y dəyişəninə müraciət edə bilmərik. Əks halda kompilyator səhv mesajı verər:

```
void test() {
//A bloku

	int x;

	if (şərt) {
	//B bloku
```

```
            int y;

      }

      //y dəyişəninə müraciət edə bilmərik
      //kompilyasiya xətası
      cout << y;

}
```

Lakin B bloku daxilində x dəyişəninə müraciət edə bilərik, çünki, B bloku üçün x qlobal dəyişəndir:

```
void test() {
//A bloku

      int x;

      if (şərt) {
      //B bloku

            int y;

            // x dəyişəninə müraciət edə bilərik
            y = x;

      }

}
```

9.9 Funksiyanın qayıtması barədə ətraflı

Biz yuxarıda funksiyanın nəticə qaytarması barədə danışarkən `return` əmri ilə tanış olduq və qeyd elədik ki `return` əmri funksiyadan nəticə qaytarmaq üçün istifadə olunur. Lakin `return` əmri nəticə qaytarmaqla bərabər, həm də funksyanın işini yarımçıq dayandırıb geri qayıtmaq üçün də istifadə olunur. Yəni funksiyanın istənilən yerində `return` əmri icra olunsa, onda `return` əmrindən sonra gələn kod hissəsi ***icra olunmayacaq*** və funksiya dərhal geri qayıdacaq. Sadə nümunə əsasında bunu izah edək. Tutaq ki aşağıdakı kimi test funksiyası verilib:

```
void test(int x) {

      cout << "Setir 1 \n";

      cout << "Setir 2 \n";

      if (x > 5)
            return;

      cout << "Setir 3";
}
```

test funksiyasının bədən hissəsində əvvəlcə iki cout operatoru icra olunur. Daha sonra isə test funksiyasına ötürülən x parametrinin qiyməti 5 ilə müqaisə olunur. Əgər funksiyaya ötürülən qiymət 5 –dən böyük olarsa bu halda return əmri icra olunur və funksiya dərhal geri qayıdır, növbəti cout operatorunu icra etmədən. Tam proqram kodu və nümunə icra nəticələri aşağıda verilib:

```
#include <iostream>

using namespace std;

void test(int x) {

	cout << "Setir 1 \n";

	cout << "Setir 2 \n";

	if (x > 5)
		return;

	cout << "Setir 3";
}


int main() {

	int x;

	cout << "Her-hansi eded daxil edin\n";
	cin >> x;

	//istifadecinin daxil etdiyi ededi test funksiyasina
	//oturek
	test(x);

}
```

Nəticə 1

```
Her-hansi eded daxil edin
0
Setir 1
Setir 2
Setir 3
```

Nəticə 2

```
Her-hansi eded daxil edin
45
Setir 1
Setir 2
```

Əlavə onu da qeyd edək ki, funksiyanın tipinin void olmasına baxmayaraq biz return – dən istifadə elədik. Bu halda return hansısa nəticə qaytarmaq üçün deyil, sadəcə geri qayıtmaq üçün istifadə olunur.

9.10 Məlumatın qiymətə və ünvana görə ötürülməsi

C++ dilində məlumat funksiyaya iki üsulla ötürülə bilər: qiymətə və ünvana görə. Bu bölmədə hər iki üsul ötürülmə arasındakı fərqdən söhbət açacağıq. İndiyə kimi tərtib etdiyimiz funksiyalarda məlumatı yalnız qiymətə görə ötürürdük. Misal üçün aşağıdakı test1 funksiyasında `int` tipli x parametri qiymətə görə ötürülür.

```
void test1 (int x)
```

Ünvana görə ötürmədə isə funksiyanın elanında parametrin adının əvvəlinə ampersand - & işarəsi artırılır. Aşağıda `int` tipli x parametri test2 funksiyasına ünvana görə ötürülür.

```
void test2(int &x)
```

Maraqlı və əhəmiyyətli bir sual yaranır: bu iki ötürülmənin fərqi nədir?

Qiymətə görə ötürülmə zamanı funksiyaya məlumatın özü yox, nüsxəsi ötürülür. Bu zaman funksiya daxilində ona ötürülən parametr üzərindəki dəyişikliklər funksiya qayıtdıqdan sonra qüvvədən düşür. Çünki dəyişikliklər nüsxə üzərində aparılmış olur. Nümunəyə baxaq.

```
#include <iostream>

using namespace std;

//qiymete gore oturme
void test1( int x) {

	//funksiyaya oturulen qiymeti cap edek
	cout << "test1 -e oturuldu "<< x <<"\n";

	//parametrin qiymetini deyishek
	x = x * 2;

	//yeni qiymeti tekrar cap edek
	cout << "test1 -de oldu " << x << "\n";
}

int main()
{
	int x;
```

```
	x = 9;

	//x deyishenin qiymetini cap edek
	cout << "test1 -e oturulmeden evvel " << x << "\n";

	//test1 -i cagiraq ve x -i qiymete gore oturek
	test1(x);

	//test1 qayitdiqdan sonra x-i cap edek
	cout << "test1 -e oturulmeden sonra " << x << "\n";

}
```

Nəticə:

```
test1 -e oturulmeden evvel 9
test1 -e oturuldu 9
test1 -de oldu 18
test1 -e oturulmeden sonra 9
```

Proqram kodunun və icra nəticəsinin təhlili oxuculara sərbəst həvalə olunur.

Ünvana görə ötürmə zamanı isə funksiyaya məlumatın ünvanı ötürülür və funksiya daxilində parametr üzərində aparılan bütün dəyişikliklər funksiya qayıtdıqdan sonra qüvvədə qalır. Nümunə koda nəzər salaq.

```
#include <iostream>

using namespace std;

//unvana gore oturme
void test2(int &x) {

	//funksiyaya oturulen qiymeti cap edek
	cout << "test2 -e oturuldu " << x << "\n";

	//parametrin qiymetini deyishek
	x = x * 2;

	//yeni qiymeti tekrar cap edek
	cout << "test2 -de oldu " << x << "\n";
}

int main()
{
	int x;

	x = 9;
```

```
        //x deyishenin qiymetini cap edek
        cout << "test2 -e oturulmeden evvel " << x << "\n";

        //test2 -i cagiraq ve x -i unvana gore oturek
        test2(x);

        //test2 qayitdiqdan sonra x-i cap edek
        cout << "test2 -e oturulmeden sonra " << x << "\n";

}
```

Nəticə:

```
test2 -e oturulmeden evvel 9
test2 -e oturuldu 9
test2 -de oldu 18
test2 -e oturulmeden sonra 18
```

Yenə kod və nəticənin təhlili oxucunun öz öhdəsinə buraxılır. Gördüyümüz kimi ünvana görə ötürülmə zamanı funksiya daxilinə parametr üzərində edilən dəyişikliklər qüvvədə qalır.

9.10.1 Göstəricinin ötürülməsi

Sual: Funksiyaya parametr olaraq göstərici ötürə bilərikmi?

Əlbəttdə! C++ dilində istənilən tipdən olan göstəricini funksiyaya parametr olaraq ötürə bilərik və hətta funksiya özü də göstərici tipli nəticə qaytara bilər.

Parametri int tiplli göstərici olan nümunə test3 funksiyasının elanını nəzərdən keçirək.

```
void test3 ( int *x)
```

Göstəricilərlə yanaşı parametr sətrində digər dəyişənlər də elan edə bilərik:

```
void test4(int *x, int y)
```

Funksiya daxilində göstərici kimi elan olunmuş parametrlərə elə adi göstəricilər kimi müraciət edirik, aşağıdakı kimi:

```
#include <iostream>

using namespace std;

void test3(int *x) {
```

```
	cout << "x gostericisinin istinad etdiyi unvan "
		<< x << "\n";

	cout << "hemin unvanda yerleshen melumat "
		<< *x << "\n";
}

int main()
{
	//int tipli x deyisheni ve y gostericisi elan edek
	int x, *y;

	x = 15;

	//y -e x-in unvanin menimsedek
	y = &x;

	//test3 -u cagiraq ve y -i  oturek
	test3(y);
}
```

Nəticə:

```
x gostericisinin istinad etdiyi unvan 0000001C274EFC84
hemin unvanda yerleshen melumat 15
```

Sual: Bəs funksiya daxilində göstərici parametrin istinad elədiyi məlumatı dəyişsək, funksiya qayıtdıqdan sonra bu dəyişikliklər qüvvədə qalarmı?

Əlbəttə! Göstərici qiymət olaraq özündə istinad elədiyi məlumatın ünvanın saxladığına görə funksiya daxilində həmin göstərici parametrin istinad elədiyi ünvanda yerləşən məlumat üzərindəki bütün dəyişikliklər funksiya qayıtdıqdan sonra qüvvədə qalır. Nümunəyə baxaq:

```
#include <iostream>

using namespace std;

void test3(int *x) {

	cout << "x gostericisinin istinad etdiyi unvan "
		<< x << "\n";

	cout << "hemin unvanda yerleshen melumat "
		<< *x << "\n";

	//x gostericisinin istinad etdiyi melumati deyishek
	*x = 23;
```

```
}


int main()
{
	//int tipli x deyisheni ve y gostericisi elan edek
	int x, *y;

	x = 15;

	//y -e x-in unvanin menimsedek
	y = &x;

	cout << "x -in qiymeti funksiyadan evvel " << x << "\n";

	//test3 -u cagiraq ve y -i  oturek
	test3(y);

	cout << "x -in qiymeti funksiyadan sonra " << x << "\n";

}
```

Nəticə:

```
x -in qiymeti funksiyadan evvel 15
x gostericisinin istinad etdiyi unvan 0000004BF0BBFCF4
hemin unvanda yerleshen melumat 15
x -in qiymeti funksiyadan sonra 23
```

Gördüyümüz kimi funksiya daxilində gösətəricinin elan etdiyi məlumat üzərindəki aparılan dəyişikliklər funksiya qayıtdıqdan sonra qüvvədə qalır.

Sual: Bəs funksiya daxilində göstəricinin istinad elədiyi ünvanın özünü dəyişsək necə, funksiya qayıtdıqdan sonra bu dəyişiklik qüvvədə qalarmı?

Xeyr! Gəlin əvvəlcə müvafiq nümunə ilə tanış olaq ki, məsələnin nədən getdiyi daha da aydın olsun.

```
#include <iostream>

using namespace std;

void test3(int *x, int *y) {

	cout << "\n   test3 funksiyasi daxilinde \n\n";

	cout << "   x gostericisinin istinad etdiyi unvan "
		<< x << "\n";
```

```
	cout << "   hemin unvanda yerleshen melumat "
		<< *x << "\n";

	cout << "   y gostericisinin istinad etdiyi unvan "
	     << y << "\n";

	cout << "   hemin unvanda yerleshen melumat "
	     << *y << "\n";

	//x gostericisinin istinad etdiyi unvani deyishek
	cout << "   x gostericisinin istinad etdiyi unvani deyishirik \n";
	x = y;

	cout << "   x gostericisinin istinad etdiyi unvan "
	     << x << "\n";

	cout << "   hemin unvanda yerleshen melumat "
	     << *x << "\n";

	cout << "\n   test3 funksiyasi sonu \n\n";

}



int main()
{
	int x, y, *z, *r;

	x = 15;
	y = 26;

	z = &x;
	r = &y;

	cout << "z -in istinad etdiyi unvan funksiyadan evvel " << z << "\n";
	cout << "hemin unvandaki melumat funksiyadan evvel " << *z << "\n";

	test3(z, r);

	cout << "z -in istinad etdiyi unvan funksiyadan evvel " << z << "\n";
	cout << "hemin unvandaki melumat funksiyadan evvel " << *z << "\n";

}
```

Nəticə:

```
z -in istinad etdiyi unvan funksiyadan evvel 000000BCA64CFC04
hemin unvandaki melumat funksiyadan evvel 15

   test3 funksiyasi daxilinde

   x gostericisinin istinad etdiyi unvan 000000BCA64CFC04
```

```
   hemin unvanda yerleshen melumat 15
   y gostericisinin istinad etdiyi unvan 000000BCA64CFC24
   hemin unvanda yerleshen melumat 26
   x gostericisinin istinad etdiyi unvani deyishirik
   x gostericisinin istinad etdiyi unvan 000000BCA64CFC24
   hemin unvanda yerleshen melumat 26

   test3 funksiyasi sonu

z -in istinad etdiyi unvan funksiyadan evvel 000000BCA64CFC04
hemin unvandaki melumat funksiyadan evvel 15
```

Görürük ki, funksiya daxilində göstərici parametrin istinad etdiyi ünvanı dəyişməyimiz funksiya qayıtdıqdan sonra qüvvədə qalmır.

Sual: Bəs funksiyaya ötürülən göstərici parametrin istinad elədiyi ünvanı necə dəyişmək olar?

İkiqat göstəricidən istifadə etməklə.

9.10.2 Cərgələrin funksiyaya parametr kimi ötürülməsi

C++ dilində funksiyaya parametr olaraq cərgə ötürmək üçün funksiyanın elanı sətrində parametrin adından sonra kvadrat mötərizə qoyulur, aşağıdakı kimi:

```
void test ( int x[])
```

Bu elanda bildirilir ki, test funksiyası parametr olaraq int tipli cərgə qəbul edir. Cərgənin elementlərinin sayının göstərilib, göstərilməməsi vacib deyil. C++ bunu yoxlamır. Adətən cərgənin elementlərinin sayı funksiyaya əlavə parametr kimi ötürülür. Diqqət yetirməli məqam isə odur ki, funksiya çağrılanda cərgənin adını kvadrat mötərizəsiz verəcəyik, aşağıdak kimi:

```
int ededler[10];

test(ededler);
```

Yuxarıdakı kodda int tipli 10 elementdən ibarət ededler cərgəsi test funksiyasına parametr olaraq ötürülür. Nümunə proqram koduna baxaq:

```
#include <iostream>

using namespace std;

void enb(int x[]) {

      int i, max;
```

```
	max = x[0];

	for (i = 0; i < 10; ++i)
		if (x[i] > max)
			max = x[i];

	cout << "En boyuk: " << max << "\n";

}

int main() {

	int ededler[10], i;

	cout << "10 eded daxil edin\n";
	for (i = 0; i < 10; ++i)
		cin >> ededler[i];

	enb(ededler);

}
```

Nəticə:

```
10 eded daxil edin
112 5 67 8 3 55 786 4 43 52
En boyuk: 786
```

İzahı:

Əvvəlcə `int` tipli 10 elementdən ibarət **`ededler`** cərgəsi elan edirik:

```
int ededler[10], i;
```

Daha sonra **`ededler`** cərgəsinin elementlərinə istifadəçi tərəfindən qiymətlər mənimsədilir. **`ededler`** cərgəsini **`enb`** funksiyasına aşağıdakı kimi ötürürük:

```
enb(ededler);
```

Gördüyümüz kimi burada biz cərgənin adını kvadrat mötərizələri olmadan yazmışıq, sintaksis tələbi olaraq: funksiyanın elanında cərgənin adından sonra kvadrat mötərizə qoymalıyıq, funksiyanı çağıranda isə yox.

`enb` funksiyası qəbul etdiyi cərgənin ən böyük parametrini tapır və çap edir. Cərgənin ən böyük elementinin tapılması ilə cərgələr bölməsində ətraflı məşğul olmuşuq.

İndi isə **`enb`** funksiyasını elə dəyişək ki, o cərgənin ən böyük elementini çap etməsin, lakin nəticə olaraq qaytarsın. Bu zaman funksiyada aşağıdakı kimi cüzi dəyişikliklər

etməliyik. İlk olaraq **enb** funksiyası artıq nəticə qaytarmalı olduğundan, həm də `int` tipli nəticə qaytarmalı olduğundan onun elanında tipini `void` yox `int` etməliyik, aşağıdakı kimi:

```
int enb(int x[])
```

Daha sonra isə ən böyük element tapıldıqdan sonra onu çap etmək əvəzinə `return` əmri ilə aşağıdakı kimi qaytara bilərik:

```
return max;
```

Funksiyanın tam kodu belə olar:

```
int enb(int x[]) {

	int i, max;

	max = x[0];

	for (i = 0; i < 10; ++i)
		if (x[i] > max)
			max = x[i];

	return max;

}
```

Funksiyanın qaytardığı nəticəni yadda saxlamaq üçün əlavə hər-hansı dəyişən elan edib, misal üçün y dəyişəni, funksiyanın nəticəsin həmin dəyişənə mənimsədə bilərik:

```
y = enb(ededler);
```

Proqramın tam kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int enb(int x[]) {

	int i, max;

	max = x[0];

	for (i = 0; i < 10; ++i)
		if (x[i] > max)
			max = x[i];

	return max;

}
```

```
int main() {

	int ededler[10], i, y;

	cout << "10 eded daxil edin\n";
	for (i = 0; i < 10; ++i)
		cin >> ededler[i];

	y =   enb(ededler);

	cout << "En boyuk: " << y ;

}
```

Cərgəyə qiymət mənimsədilməsi üçün də əlavə daxilet adlı funksiya tərtib edə bilərik:

```
void daxilet(int x[]) {

	int i;

	cout << "10 eded daxil edin\n";

	for (i = 0; i < 10; ++i)
		cin >> x[i];
}
```

Proqramın daxilet funksiyasından istifadə edən versiyası aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

int enb(int x[]) {

	int i, max;

	max = x[0];

	for (i = 0; i < 10; ++i)
		if (x[i] > max)
			max = x[i];

	return max;

}

void daxilet(int x[]) {

	int i;

	cout << "10 eded daxil edin\n";

	for (i = 0; i < 10; ++i)
```

```
            cin >> x[i];
}

int main() {

      int ededler[10], y;

      daxilet(ededler);

      y =   enb(ededler);

      cout << "En boyuk: " << y ;

}
```

Adətən irihəcmli proqramlarda ümumi iş bir neçə hissəyə bölünür, hər bir hissənin yerinə yetirilməsi üçün müvafiq funksiya tərtib olunur. Daha sonra bu funksiyalar ayrı bir ümumi funksiyada birləşdirilir. Bu şəkildə proqram kodunu həm komanda şəklində tərtib etmək asan olur(ayrı-ayrı qruplar ancaq verilmiş funksiyanı tərtib edirlər), həm proqramda səhvləri təhlil eləmək asan olur(hər bir funksiyanı ayrılıqda test eləmək bütün kodu tam şəkildə test etməyə nisbətən asandır), həm də kodda dəyişiklik eləmək asan olur. Misal üçün tutuq ki bizdən 10 elementi olan int tipli cərgənin elementlərini artan sırada düzən proqram qurmaq tələb olunur. Bu cür proqramların tərtibi ilə cərgələr mövzusunda məşğul olmuşuq. Nümunə üçün aşağıdakı proqram kodunu göstərə bilərik:

```
#include <iostream>

using namespace std;

int main() {

      int x[10], y, z, i, j, k;

      //x -in elementlerini daxil edek
      cout << "x -in elementlerini daxil edin \n";
      for (i = 0; i<10; ++i)
            cin >> x[i];

      //ESAS DOVR
      for (j = 9; j >= 0; --j) {

            //en boyuk elementi ve onun indeksini tapaq
            y = x[0];
            k = 0;

            for (i = 1; i <= j; ++i) {
                  if (x[i] > y) {
                        y = x[i];
                        k = i;
                  }
            }
```

```
            //eger en boyuk element sonuncu deyilse
            //onu sona kocurek
            if (k != j) {
                  z = x[k];
                  x[k] = x[j];
                  x[j] = z;
            }

      }

      //cergeni cap edek
      cout << "x -in yeni duzumu\n";
      for (i = 0; i<10; ++i)
            cout << x[i] << " ";
}
```

Gördüyümüz kimi proqram kodu bütöv şəkildə main funksiyasında tərtib olunub. İlk baxışda kodun nə iş gördüyün anlamaq və kodda səhvləri tapmaq elə də asan görünmür. İndi isə eyni proqramın funksiyalardan istifadə olunan variantını tərtib edib, nəticəni müqaisə edək.

Proqramda əvvəlcə istifadəçi tərəfindən cərgəyə qiymətlər mənimsədilir, daha sonra cərgənin elementləri artan sırada düzülür, sonda isə hazır cərgə çap olunur. Eyni qayda ilə biz də bu işlərin hər birini görmək üçün ayrıca bir funksiya tərtib edib, daha sonra onları main –də çağıra bilərik. Cərgənin elementlərinə qiymətlər mənimsətmək üçün daxilet, elementləri artan sırada düzmək üçün artan, çap etmək üçün capet adlı funksiya tərtib edək.

Cərgənin elementlərinə qiymətlər mənimsətmək üçün daxilet funksiyasının nümunə kodu aşağıdakı kimi olar:

```
void daxilet(int x[]) {

      int i;

      cout << "10 eded daxil edin\n";

      for (i = 0; i < 10; ++i)
            cin >> x[i];
}
```

Cərgənin elementlərini artan sırada düzmək üçün artan funksiyasının nümunə kodu aşağıdakı kimi olar:

```
void artan(int x[]) {

      int i, j, y, k, z;

      for (j = 9; j >= 0; --j) {
```

```
		//en boyuk elementi ve onun indeksini tapaq
		y = x[0];
		k = 0;

		for (i = 1; i <= j; ++i) {
			if (x[i] > y) {
				y = x[i];
				k = i;
			}
		}

		//eger en boyuk element sonuncu deyilse
		//onu sona kocurek
		if (k != j) {
			z = x[k];
			x[k] = x[j];
			x[j] = z;
		}

	}

}
```

Cərgənin elementlərini çap etmək üçün capet funksiyasının nümunə kodu aşağıdakı kimi olar:

```
void capet(int x[]) {

	int i;

	cout << "Cergenin elementleri \n";

	for (i = 0; i < 10; ++i)
		cout << x[i] << " ";
}
```

Proqramın tam kodu isə aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

void daxilet(int x[]) {

	int i;

	cout << "10 eded daxil edin\n";

	for (i = 0; i < 10; ++i)
		cin >> x[i];
}


void artan(int x[]) {

	int i, j, y, k, z;
```

```
	for (j = 9; j >= 0; --j) {

		//en boyuk elementi ve onun indeksini tapaq
		y = x[0];
		k = 0;

		for (i = 1; i <= j; ++i) {
			if (x[i] > y) {
				y = x[i];
				k = i;
			}
		}

		//eger en boyuk element sonuncu deyilse
		//onu sona kocurek
		if (k != j) {
			z = x[k];
			x[k] = x[j];
			x[j] = z;
		}

	}

}

void capet(int x[]) {

	int i;

	cout << "Cergenin elementleri \n";

	for (i = 0; i < 10; ++i)
		cout << x[i] << " ";
}

int main() {

	int ededler[10];

	daxilet(ededler);

	artan(ededler);

	capet(ededler);

}
```

9.10.3 Funksiyalara ikiölçülü cərgələrin ötürülməsi

İkiölçülü cərgələr də funksiyalara eynilə birölçülü cərgələr kimi ötürülür, sadəcə funksiyanın elanında cərgənin adından sonra bir yox iki kvadrat mötərizə qoyulur, aşağıdakı kimi:

```
void test(int x[][])
```

Funksiyanı çağıranda isə heç bir kvadrat mötərizəsiz yazırıq cərgənin adın:

```
test(x);
```

9.11 Rekursiv funksiyalar

C++ dili fuknsiyanı proqramın istənilən yerindən, o cümlədən hər hansı digər funksiyanın daxilindən çağırmağa imkan verir. Bundan əlavə funksiyanı elə öz daxilindən də çağıra bilərik. Proqramlaşmada buna ***rekursiya*** deyirlər.

Misal üçün aşağıdakı kimi rekursiv bir funksiyaya baxaq:

```
void test() {
	cout << "Men rekursiv funksiyayam\n";
	test();
}
```

Baxdığımız nümunə test funksiyası əvvəlcə ekranda sətir çap edir, daha sonra öz daxilindən özünü çağırır. Nəticədə test funksiyası yenidən icra olunmağa başlayar. Bu isə öz növbəsində yenidən ekranda sətir çap edər və bir daha artıq 3-cü dəfə test funksiyası çağrılar. Yəni funksiyanın öz-özünə müraciət etməsi sonsuz sayda baş verər, aydın məsələdir ki, bu zaman proqramımız donacaq. Maraqlı oxuccular bunu yoxlaya bilərlər.

Bu nümunəni biz izah məqsədilə verdik. Tətbiqi proqramlar qurarkən rekursiv funksiyanın dayanma şərtini ciddi nəzərə almaq lazımdır, sonsuz çağrılmaların qarşısını almaq üçün. Adətən bunun üçün funksiya daxilində hansısa bir şərt qoyub, onun ödənib-odənmədiyini yoxlamaq lazımdır. Aşağıdakı nümundə rekursiv olaraq cərgənin elementlərini çap edirik:

```
#include <iostream>
using namespace std;
void cap_et_rekursiv(int x) {
	//basha catma sherti
```

```
	if (x > 10)
		return;

	cout << x << " ";

	cap_et_rekursiv(x+1);

}

int main() {

	cap_et_rekursiv(1);

}
```

Nəticə:

```
1 2 3 4 5 6 7 8 9 10
```

Izahı: Yuxarıdakı `cap_et_rekursiv` funksiyasında biz başa çatma şərti olaraq funksiyanın qəbul etdiyi parametrin 10-dan böyük olduğunu yoxlayırıq. Əgər şərt ödənmirsə onda onu çap edirik və da sonra funksiyanı yenidən çağıraraq parametrin qiymətini bir vahid artırıb funksiyaya ötürürük.

```
cap_et_rekursiv(x+1);
```

main –dən `cap_et_rekursiv` funksiyasını çağıran zaman 1 başlanğıc qiyməti ilə çağırırıq:

```
cap_et_rekursiv(1);
```

9.11.1 n faktorial

Rekursiv funksiyalarla bağlı məşhur nümunələrdən biri də n faktorialın hesablanmasıdır. n faktorial n! kimi işarə olunur və aşağıdakı kimi hesablanır:

```
0! = 1
n! = n*(n-1)!
```

Bu cür təyindən misal üçün 6! –lı aşağıdakı kimi hesablaya bilərik:

```
6! = 6*5!
5! = 5*4!
4! = 4*3!
3! = 3*2!
2! = 2*1!
```

```
1! = 1*0!
0! = 1
----
6! = 6*5*4*3*2*1
```

C++ dilində n faktorialı aşağıdakı nümunə proqram ilə hesablaya bilərik. Burada verilmiş ədədin faktorialını hesablamaq üçün rekursiv nfak funksiyasından istifadə olunur. Koda nəzər salaq:

```
#include <iostream>

using namespace std;

//n! faktoriali hesablamaq ucun proqram

int nfak(int n) {

	if (n == 0)
		return 1;

	return n * nfak(n - 1);

}



int main() {

	int i;

	for (i=0; i<8; ++i)
		cout << i <<"! = " << nfak(i) << "\n";

}
```

Nəticə:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
```

Çalışma 1. Elə funksiya tərtib edin ki, arqument olaraq iki tam ədəd qəbul edir, onların cəmini hesablayıb nəticə olaraq qaytarır. Bu funksiyadan istifadə etməklə proqram tərtib edin, hansı ki, istifadəçidən iki ədəd qəbul edir, sonra bu ədədləri parametr olaraq sizin tərtib etdiyiniz funksiyaya ötürür və nəticəni alaraq ekranda çap edir.

Çalışma 2. Elə funksiya tərtib edin ki, istifadəçidən birinci ədəd, hesablama əməliyyatı (+,-,*,/) və ikinci ədəd qəbul edir. Hesablama nəticəsini çap edir. İstifadəçinin daxil etdiyi hər 3 parametr funksyaya ötürülməlidir və yekun nəticə funksiyadan alınmalıdır.

Çalışma 3. Funksiyadan istifadə etməklə iki ədədin böyüyünü tapan proqram qurun.

Çalışma 4. Elə funksiya qurun ki parametr olaraq kəsr ədəd qəbul etsin və onun tam hissəsini nəticə olaraq qaytarsın. Tərtib etdiyiniz funksiyadan istifadə etməklə proqram qurun.

Çalışma 5. Elə funksiya tərtib edin ki, verilmiş ədədin sadə olduğunu müəyyən etsin. Həmin funksiyadan istifadə edərək 1 -dən 100 -ə qədər olan ədədlər arasında yerləşən sadə ədədləri çap edən proqram tərtib edin.

Çalışma 6. Funksiyadan istifadə etməklə verilmiş düzbucaqlının sahəsini hesablayan proqram tərtib edin.

Çalışma 7. Funksiyadan istifadə etməklə verilmiş kubun səthinin sahəsini hesablayan proqram tərtib edin.

Çalışma 8. Funksiyadan istifadə etməklə verilmiş radiuslu dairənin sahəsini hesablayan proqram tərtib edin.

§10 Sətirlər.

10.1 Sətirlər

C++ dilində iki cür sətrilərdən istifadə edə bilərik: simvollar cərgəsi şəklində elan olunan sətirlərdən və **string** sinfinə mənsub olan təkmilləşdirilimiş sətir obyektlərindən.

Ilk olaraq simvollar cərgəsi kimi elan olunan sətirlərlə tanış olaq.

10.1 C Sətirləri

Qeyd edim ki bu tip sətirlər əsasən C dilində istifadə olunduğundan və C++ dilinə də C dilindən keçdiyindən onları çox vaxt C sətirləri də adlandırırlar.

Mahiyyət etibarilə C sətirləri sadəcə char tipindən olan cərgələrdir.

```
char ad [10];
```

Yuxarıdakı nümunədə 10 simvoldan ibarət ad sətri elan etdik. Sətirlərə qiymət mənimsətməyin müxtəlif yolları var. Misal üçün elan vaxtı, aşağıdakı kimi:

```
char ad[10] = "Ahmed";
```

Əgər cout operatoru ilə ad sətrini çap etsək onda ekranda onun qiyməti yəni `"Ahmed"` sözü çap olunar(cüt dırnaq işarəsi olmadan). Nümunə koda baxaq:

```
#include <iostream>

using namespace std;

int main() {

	char ad[10] = "Ahmed";

	cout << ad;

}
```

Nəticə:

```
Ahmed
```

Sətirlərə cin operatoru ilə də qiymət mənimsədə bilərik. Nümunəyə baxaq:

```
#include <iostream>

using namespace std;

int main() {

	char ad[10];

	cout << "Zehmet olmasa adinizi daxil edin: \n";
	cin  >> ad;

	cout << "Salam " << ad;

}
```

Nəticə:

```
Zehmet olmasa adinizi daxil edin:

Ahmed

Salam Ahmed
```

Gördüyümüz kimi sətirlərə həm elan vaxtı cüt dırnaq arasında, həm də proqram icra olunarkən cin operatoru ilə qiymət mənimsədə bilərik.

Çalışma 1. İstifadəçinin daxil etdiyi sətrin ilk simvolunu ekranda çap edən proqram tərtib edin.

Həlli: Sətirlər char tipindən olan cərgə olduğuna görə cətri daxil etdikdən sonra onun ilk elementini çap edə bilərik (indeks 0).

```
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
	char str[80];


	cout << "Her hansi setir daxil edin\n";
	cin >> str;
```

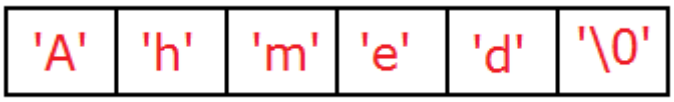
```
    cout << "Ilk simvol = " << str[0];
}
```

Nəticə:

```
Her hansi setir daxil edin
Telebe
Ilk simvol = T
```

10.1.1 Sətrin sonu simvolu

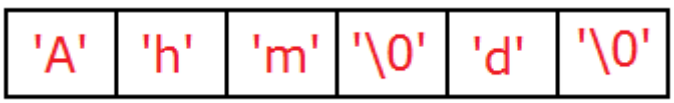
C sətirləri ilə ilk dəfə tanış olarkən nisbətən çətinlik yaradan məsələlərdən biri sətrin sonu simvoludur. Bu yalnız C sətirlərinə aid olan məsələdir. Sətrin sonu somvolu tək bir simvoldur və - `'\0'` kimi işarə olunur. Bu simvol adətən kompilyator tərəfindən sətirlərin sonuna artırılır, lakin proqramçılar da bu simvoldan tez-tez istifadə edirlər. Misal üçün yuxarıdakı ad sətrinin elanı zamanı ona qiymətini mənimsədərkən sətrin sonu simvolu ad sətrinin sonuna kompilyator tərəfindən artırılmış olar və yaddaşda ad sətri aşağıdakı kimi təsvir olunar:



Proqramçı tərəfindən sətrin sonu simvolu sətrin istənilən yerinə mənimsədilə bilər və həmin yerdən sonra gələn hissə artıq nəzərə alınmayacaq. Misal üçün Yuxarıdakı nümunədə əgər biz sətrin 3-cü simvolundan (Ahm) sonra gələn hissəni silmək istəyiriksə bu zaman 3-cü simvolun sonuna yəni 4-cü simvola sətrin sonu simvolunu mənimsətməliyik.

```
//ad setrinin 4-cu simvoluna(indeks 3) setrin sonu simvolunu menimsedek
ad[3] = '\0';
```

Yaddaşda vəziyyət bu cür olar:



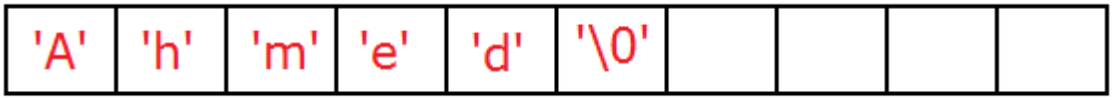
Bundan sonra əgər ad sətrini çap etsək yalnız sətrin sonu simvoluna qədər olan hissə çap oluncaq. Baxmayaraq ki, digər simvollar yaddaşdan silinməyəcək, onlar sadəcə yoxmuş kimi özlərini aparacaq. Nümunəyə baxaq:

```
#include <iostream>

using namespace std;

int main() {

	char ad[10] ;

	cout << "Zehmet olmasa adinizi daxil edin: \n";
	cin  >> ad;

	cout << "Sizin adiniz evvel :" << ad << "\n";

	//ad setrinin 4-cu simvoluna(indeks 3) setrin sonu simvolunu menimsedek
	ad[3] = '\0';

	cout << "Sizin adiniz sonra :" << ad;

}
```

Nəticə:

```
Zehmet olmasa adinizi daxil edin:
Ahmed
Sizin adiniz evvel :Ahmed
Sizin adiniz sonra :Ahm
```

Qeyd edək ki, sətrin ölçüsünü elan edərkən mütləq sətrin sonu simvolunu nəzərə almaq lazımdır. Misal üçün əgər sətrə 5 simvoldan ibarət bir söz yerləşdirmək istəyiriksə, deyək ki Ahmed sözün, bu zaman elan zamanı bir simvol da əlavə sətrin simvolu üçün nəzərə alıb ölçünü 6 simvol verməliyik. Əlbəttə ölçünü 6 simvoldan çox da verə bilərik. Bu zaman tələb olunan sayda simvollar sətrə yerləşdiriləcək, qalan artıq yerlər isə boş qalacaq. Yuxarıdakı nümunədə 10 simvoldan ibarət ad sətri elan etdik, lakin ona 5 simvoldan ibarət Ahmed sözünü yerləşdirdik. Sətrin sonu simvolunu da kompilyator əlavə etdi, növbəti 4 simvol yeri isə boş qaldı, aşağıdakı kimi.



Ümumiyyətlə sətrin ölçüsünü onda yerləşdirəcəyiniz ifadənin uzunluğundan bir qədər artıq götürməyinizin heç bir ziyanı olmaz, əksinə bu bir qədər ehtiyat tədbiri hesab olunur. Misal üçün əgər istifadəçinin soyadını yadda saxlamaq üçün sətir elan etməlisinizsə, onda təxmini fikirləşsək, ən uzun soyadı yadda saxlamaq üçün 20-30 simvoldan ibarət sətir kifayət edər. Amma siz rahatlıqla 256 və ya 512 v.s. ölçüdə sətirdən istifadə edə bilərsiniz. Bunu sizə heç kim nöqsan tutmaz, amma soyad yadda saxlamaq üçün elan etdiyiniz sətrin ölçüsün 10000 götürməyiniz artıq yaddaş israfçılığı deməkdir və proqramlarınızı təhvil verdiyiniz rəhbər proqramçılar bunu sizə nöqsan hesab edə bilərlər.

10.1.2 Sətir funksiyaları

C++ dilində C sətirləri üzərində əməliyyat aparmaq üçün bəzi kitabxana funksiyaları təyin olunub. Bunlardan istifadə etmək üçün `string.h` başlıq faylını proqramınıza əlavə etməlisiniz. Aşağıdakı cədvəldə bir neçə məşhur sətir funksiyası və onların qısa izahı verilir. Daha sonra isə bəzi funksiyalardan istifadəyə aid kod nümunələri ilə tanış olacayıq.

Standart sətir funksiyaları

strlen	Verilmiş sətrin uzunluğunu müəyyən edir
strcpy	Bir sətri digər sətrin üzərinə yazır
strncpy	Bir sətrin verilmiş ölçüdə simvollarını digər sətrin üzərinə yazır
strcat	Bir sətri digər sətrin sonuna əlavə edir
strncat	Bir sətrin verilmiş ölçüdə simvollarını digər sətrin sonuna əlavə edir
strcmp	Bir sətri digər sətir ilə müqaisə edir
strncmp	İki sətrin verilmiş uzunluqda hissəsini müqaisə edir
strchr	Verilmiş sətirdə verilmiş simvolun ilk rast gəldiyi yeri müəyyən edir
strtok	Verilmiş sətri verilmiş simvollarla parçalara ayırır

Gəlin bu funksiyalar ilə tanış olaq.

10.1.2.1 Strlen funksiyası

Sətir funksiyaları içində ən sadəsi strlen funksiyasıdır. strlen funksiyasının elanı aşağıdakı kimidir:

```
int strlen(char * str);
```

Qeyd: Əslində elan **size_t strlen(const char *** str**);** kimidir. Biz onu sadələşdirilmiş şəkildə təqdim edirik.

strlen funksiyası parametr olaraq sətir qəbul edir və nəticə olaraq qəbul etdiyi sətrin uzunluğunu qaytarır. Sətrin uzunluğu onun başlanğıcından sətrin sonu simvollarına qədər olan simvolların sayına bərabərdir. Sətrin sonu simvolu hesaba alınmır. Gördüyümüz kimi funksiyanın elanında sətir parametri `char` tipindən olan göstərici kimi elan olunub. Sadə proqram nümunəsinə baxaq:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char str[256];
	int k;

	cout << "Her-hansi setir daxil edin \n";
	cin >> str;

	// s setrinde olan simvollarin sayini k-ya menimsedek
	k = strlen(str);

	// setrin uzunlugunu cap edek
	cout << "setrin uzunlugu = " << k;

}
```

Nəticə:

```
Her-hansi setir daxil edin
continental
setrin uzunlugu = 11
```

10.1.2.2 strcpy funksiyası

strcpy funksiyasının elanı aşağıdakı kimidir:

```
char * strcpy(char * mənsəb, const char * mənbə);
```

strcpy funksiyası mənbə kimi göstərilən sətri mənsəb kimi göstərilən sətrin üzərinə köçürür sətrin sonu simvolunu sona əlavə etməklə və nəticə olaraq mənsəb sətrini qaytarır. Nümunəyə baxaq:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char s1[256], s2[256];

	cout << "Her-hansi setir daxil edin \n";
	cin >> s1;

	// s1 setrini s2-ye kocurek
	strcpy(s2, s1);

	// s2-ni cap edek
	cout << "s2 setri = " << s2;

}
```

Nəticə:

```
Her-hansi setir daxil edin
Program
s2 setri = Program
```

10.1.2.3 strncpy funksiyası

strncpy funksiyası da strcpy funksiyası kimidir, sadəcə mənbə sətrin yalnız göstərilən sayda simvollarını mənsəbin üzərinə köçürür və sətrin sonu simvolunu da **yazmır**. Strncpy funksiyasının elanı aşağıdakı kimidir:

```
char * strcpy(char * mənsəb, const char * mənbə, int say );
```

Nümunəyə baxaq:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char s1[256], s2[256];

	cout << "Her-hansi setir daxil edin \n";
	cin >> s1;
```

```
	// s1 setrini s2-ye kocurek
	strncpy(s2, s1, 5);

	// s2-ni cap edek
	cout << "s2 setri = " << s2;

}
```

Nəticə:

```
Her-hansi setir daxil edin
Program
s2 setri = Progr
```

10.1.2.4 strcat funksiyası

strcat funksiyası bir mənbə sətri mənsəb sətrin sonuna əlavə edir. Bu zaman mənsəb sətrin sonu simvolu mənbə sətrinin ilk simvolu tərəfindən silinir və hər iki sətrin birləşməsinin sonuna mənbə sətrin sonu simvolu yazılır. Elan aşağıdakı kimidir:

```
char * strcat(char * mənsəb, const char * mənbə);
```

Nümunəyə baxaq:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char s1[256], s2[256];

	cout << "Her-hansi setir daxil edin \n";
	cin >> s1;

	cout << "diger setir daxil edin \n";
	cin >> s2;

	// s2 -ni s1-in sonuna elave edek
	strcat(s1, s2);

	// s1-ni cap edek
	cout << "s1 setri = " << s1;

}
```

Nəticə:

```
Her-hansi setir daxil edin
Salam
```

```
diger setir daxil edin
Ahmed
s1 setri = SalamAhmed
```

10.1.2.5 strcmp funksiyası

strcmp funksiyası verilmiş iki sətrin bərabər olub-olmamasını yoxlayır. Elan aşağıdakı kimidir:

```
int strcmp(const char * str1, const char * str2);
```

Əgər verilmiş iki sətir bir-birinə bərabər olarsa onda strcmp funksiyası nəticə olaraq 0 qiyməti qaytarır. strcmp ilə iki sətrin bərabərliyini yoxlamaq üçün strcmp –nin nəticəsinin 0 qiyməti ilə müqaisə etməliyik, nümunəyə baxaq:

Nümunəyə baxaq:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char s1[256], s2[256];

	cout << "Her-hansi setir daxil edin \n";
	cin >> s1;

	cout << "diger setir daxil edin \n";
	cin >> s2;

	if (strcmp(s1, s2) == 0)
		cout << "Bu setirler beraberdir\n";
	else
		cout << "Bu setirler ferqlidir\n";

}
```

Nəticə:

```
Her-hansi setir daxil edin
Ahmed
diger setir daxil edin
Ahmedd
Bu setirler ferqlidir
```

10.1.2.6 strtok funksiyası

strtok funksiyası verilmiş sətri parçalara ayırır. Bunun üçün delimetrlər sətrində verilmiş ayırıcı simvollardan istifadə edir. Elan aşağıdakı kimidir:

```
char * strtok(char * str, const char * delimetrlər);
```

Misal üçün tutaq ki, str olaraq `"Atam bazardan alma, pomidor, erik aldi."` sətri verilib. Əgər delimetr olaraq `","` vergül götürsək onda strtok yuxarıdakı sətri aşağıdakı parçalara ayırar:

```
"Atam bazardan alma"
" pomidor"
" erik aldi."
```

Biraz sonra proqram kodu ilə bu nümunəni test edəcəyik. strtok funksiyası ilə sətri parçalamaq üçün əvvəlcə birinci dəfə strtok funksiyasını çağırmalıyıq. Bu zaman parametr olaraq parçalamaq istədiyimiz sətri və ayırma simvollarını veririk. strtok sətirdən ayırdığı ilk parçanı nəticə olaraq qaytarır. Yerdə qalan parçaları əldə etmək üçün isə strtok funksiyasını NULL qiyməti qaytarana qədər çağırmalıyıq. Hər dəfə çağıranda da birinci dəfədən başqa ilk parametr olaraq NULL daxil etməliyik. Düzdü bir qədər qarışıq kimi görünsə də, ümid eliyirəm kod nümunəsi bütün qaranlıqlara aydınlıq gətirər. Nümunə kod:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char str[80] = "Atam bazardan alma, pomidor, erik aldi.";
	char s[2] = ",";
	char *hisse;

	// birinci parcani elde edek
	hisse = strtok(str, s);

	// yerde qalan parcalari elde edek
	while (hisse != NULL)
	{
		cout << hisse << "\n";

		hisse = strtok(NULL, s);
	}

}
```

Nəticə:

```
Atam bazardan alma
```

```
pomidor
erik aldi.
```

Ümid eliyirəm ki, C sətirləri mövzusu sizə az-çox aydın oldu. İrəlidə C sətrilərinə nisbətən C++ dilinin daha təkmilləşdirilmiş string sətir obyektləri ilə tanış olcayıq. Fayllarla iş mövzusu ilə tanış olduqda sətirlər iş praktikamızı təkmilləşdirəcəyik.

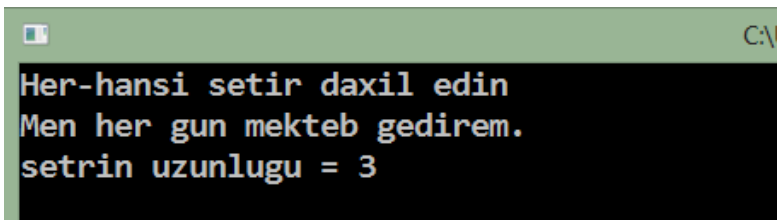
İndi isə sətirlərlə bağlı bir neçə çalışma həll edək.

10.1.2.7 getline funksiyası

Artıq bilirik ki, sətrin uzunluğunu müəyyən etmək üçün strlen funksiyasından isitfadə edirik. Nümunə kodda strlen funksiyası ilə continental sətrinin uzunluğunu müəyyən etdik. Gəlin eyni proqramı yenidən daxil edək lakin bu dəfə uzunluğunu hesablamaq istədiyimiz sətri bir qədər fərqli daxil edək.

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

	char str[256];
	int k;

	cout << "Her-hansi setir daxil edin \n";
	cin >> str;

	// s setrinde olan simvollarin sayini k-ya menimsedek
	k = strlen(str);

	// setrin uzunlugunu cap edek
	cout << "setrin uzunlugu = " << k;

}
```

Bu proqramı icra edək və uzunluğunu hesablamaq istədiyimiz sətri daxil edək:



Gördüyümüz kimi istifadəçi `"Men her gun mektebe gedirem."` sətrini daxil edib, hansı ki 28 simvoldan ibarətdir. Lakin strlen funksiyası düzgün nəticə hesablamayıb. Bunun səbəbi odur ki istifadəçi klaviturадan məlumat daxil edərkən C++ giriş sistemi məlumatı məsafə simvolları ilə hissələrə bölür və hər-bir hissəni ayrı bir məlumat kimi proqrama ötürür. Yuxarıdakı nümunədə proqrama 5 giriş məlumatı ötürülür:

```
"Men"
"her gun"
"mektebe"
"gedirem."
```

Proqram isə bunlardan yalnız birincisini qəbul edir cin operatoru vastəsilə. Əgər biz cin operatoru ilə məlumat qəbulunu davam etdirsək onda digər hissələri də ardıcıl olaraq qəbul edərik. Bu yerdə proqramın nəticəsinə diqqət yetirsək ilk hissənin yəni `"Men"` sözünün uzunluğunu çap etdiyini görərik. Bununla bağlı müxtəlif ekperimentlər aparmağınız tövsiyyə olunur(hər dəfə ilk sözün uzunluğunu dəyişib müxtəlif sətirlər daxil etməklə və yaxud cin operatorunu bir neçə dəfə çağırmaqla v.s.). Bizim üçün maraqlı olan C++ giriş sisteminin daxil olunan məlumatı parçalamadan – olduğu kimi proqrama ötürməsi qaydasını örgənməkdir. Bunun üçün **getline** funksiyasından istifadə etməliyik. getline funksiyasını indiyə kimi adət etmədiyimiz qaydada çağıracayıq. Sintaksis aşağıdakı kimidir:

```
getline(char *s, int uz);
```

birinci parametr məlumatı yerləşdirmək istədiyimiz sətri, ikinci parametr isə həmin sətrə yerləşdirmək istədiyimiz simvolların maksimal sayını bildiri. getline funkiyasını çağırarkən bir qədər obyekyönümlü metoddan istifadə olunur, belə ki, cin obyektinin funksiyası kimi çağırılır, aşağıdakı kimi:

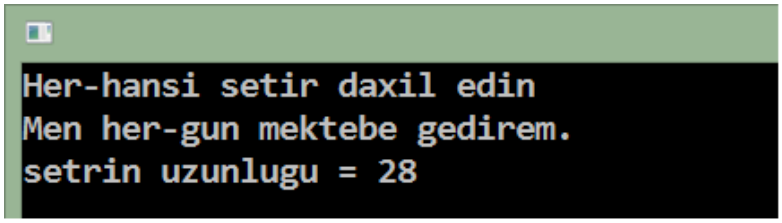
```
cin.getline(setir, uzunluq) ;
```

Diqqət yetirsəni burda cin obyekti ilə getline funksiyası arasına nöqtə işarəsi yazmışıq. Bu təsadüfi deyil, nöqtə işarəsinin mənası ilə irəlidə obyekt yönümlü proqralaşdırmanı örgənərkən tanış olacayıq.

Proqramın getline funksiyasından istifadə edən halı və müvafiq nəticəsi aşağıdakı kimi olar:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

    char str[256];
    int k;

    cout << "Her-hansi setir daxil edin \n";
    cin.getline(str, 256);

    // s setrinde olan simvollarin sayini k-ya menimsedek
    k = strlen(str);

    // setrin uzunlugunu cap edek
    cout << "setrin uzunlugu = " << k;

}
```

Əgər bu proqramı icra edib bir qədər əvvəl daxil etdiyimiz sətri təkrar daxil etsək, aşağıdakı nəticəni alarıq:



Gördüyümüz kimi bu dəfə biz istifadəçinin daxil etdiyi sətri tam olaraq qəbul etdik. Növbəti çalışmalarda həmin sətrin emalına aid çalışmalar edəcəyik.

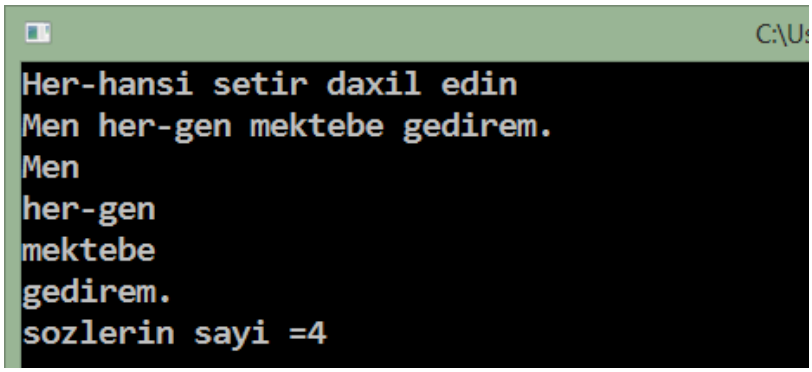
Çalışma. İstifadəçinin daxil etdiyi sətrin neçə sözdən ibarət olduğunu müəyyən edən proqram tərtib edin.

Həlli. Bunun üçün əvvəlcə getline ilə istifadəçinin daxil etdiyi sətri qəbul edirik. Daha sonra strtok ilə onu məsafə simvolları ilə hissələrə ayırıb sayı tapırıq. Kod aşağıdakı kimi olar:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {
```

```
    char setir[256], *soz;
    int say;

    cout << "Her-hansi setir daxil edin \n";
    cin.getline(setir, 256);

    say = 0;

    // birinci sozu elde edek
    soz = strtok(setir, " ");

    // yerde qalan parcalari elde edek
    while (soz != NULL)
    {
        cout <<soz<<"\n";
        soz = strtok(NULL, " ");
        say++;
    }

    // setrin uzunlugunu cap edek
    cout << "sozlerin sayi =" << say;

}
```

Əgər strtok funksiyasını başa düşmüsünüzsə onda bu kodu təhlil etmək sizə çətinlik yaratmamalıdır. Nümunə nəticə aşağıdakı kimi olar:



Yuxarıdakı çalışmanı şərtini bir qədər dəyişək.

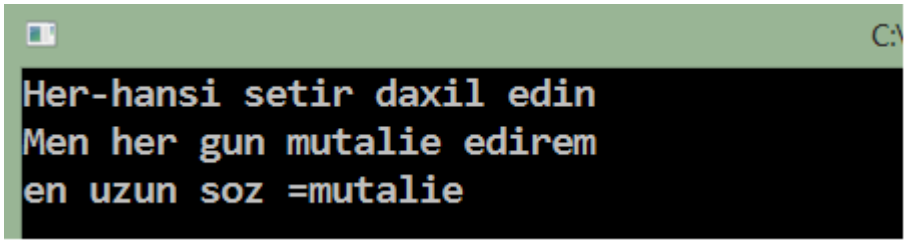
Çalışma. İstifadəçinin daxil etdiyi cümlədə ən uzun sözü çap edən proqram tərtib edin.

Həlli. Bunun üçün artıq bildiyimiz kimi əvvəlcə getline ilə istifadəçinin daxil etdiyi cümləni qəbul edirik. Burada mən sətir əvəzinə cümlə terminindən istifadə etdim. Hesab edirəm ki, bu bir anlaşılmazlığa səbəb olmaz. (Proqramlaşdırmada cümlə anlayışı mövcud deyil. Sadəcə bizim dilimizə daha münasib olduğuna görə özündə məsafə simvollarını saxlayan bir qədər böyük sətri cümlə kimi adlandırmaq qərarına gəldim.) Daha sonra qəbul etdiyimiz cümləni strtok ilə məsafə simvolları ilə hissələrə ayırırıq və ayırdığımız hər-bir hissənin(gəlin burada

dilimizə daha uyğun gələn **söz** terminindən istifadə edək) uzunluğunu tapırıq. Kod aşağıdakı kimi olar:

```
#include <iostream>
#include <string.h>

using namespace std;

int main() {

    char setir[256], enbsoz[50], *soz;
    int say, max;

    cout << "Her-hansi setir daxil edin \n";
    cin.getline(setir, 256);

    say = 0;
    max = 0;

    soz = strtok(setir, " ");
    enbsoz[0] = '\0';

    while (soz != NULL)
    {

        if (strlen(soz) > max){
            //eger cari soz max-dan uzundursa
            //onu enbsoz-e yazaq ve max-i yenileyek
            max = strlen(soz);
            strcpy(enbsoz, soz);
        }

        //novbeti sozu elde edek
        soz = strtok(NULL, " ");
    }

    // setrin uzunlugunu cap edek
    cout << "en uzun soz =" << enbsoz;

}
```

Nümunə nəticə:



10.2 string tipi

Yuxarıda biz C++ dilinə C dilindən keçən sətir tipləri ilə tanış olduq. C++ dilində çox geniş istifadə olunan digər sətir tipi isə **string** tipidir. **string** tipi ilə işləmək klassik sətirlərə nisbətən xeyli rahatdır. Bu tiplər obyekt yönümlü olub, klassik sətir funksiyalarının bir çoxunu özündə saxlamaqla yanaşı əlavə bir çox yeni funksiyalar təqdim edir. string tipindən sətri adi tiplərdən olduğu kimi elan edirik:

```
string str;
```

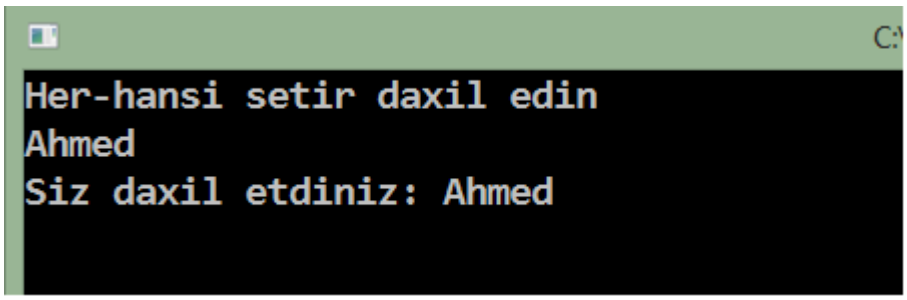
Gördüyümüz kimi bu elanda biz C sətirlərində olduğu kimi char tipli cərgə elan etmədik. Elandan sonra sətrə qiymət mənimsədə bilərik:

```
str = "Salam dunya\n";
```

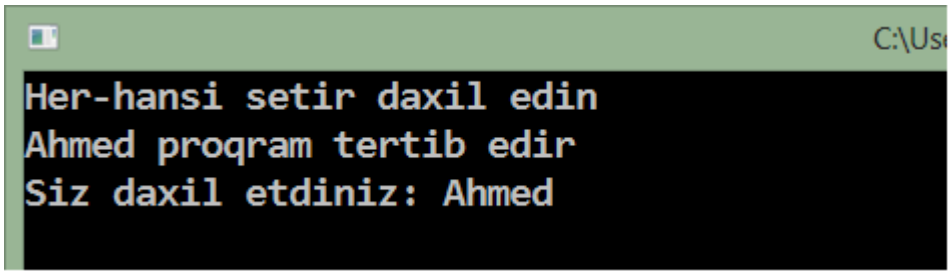
Daha sonra bu sətri cout ilə çap etsək ekranda müfaviq sətir çap olunar. Kod aşağıdakı kimi olar:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {

    string str;

    str = "Salam dunya\n";

    cout << str;

}
```

Bu tipdən istifadə etmək üçün **string** faylını proqrama əlavə etməliyik. Eyni qayda ilə **cin** ilə str sətrinə istifadəçinin daxil etdiyi qiyməti mənimsədə bilərik:



Bəs özündə məsafə simvolu saxlayan bir sətir daxil etsək necə, ancaq ilk sözü qəbul edəcək, yoxsa getline kimi tam sətri? Gəlin yoxlayaq, proqramı təkrar icra edək və bir neçə sözdən ibarət sətir daxil edək, məsafə simvolu ilə aralı:



Gördüyümüz kimi istifadəçinin daxil etdiyi məlumatı tam formada qəbul etmək üçün C sətrilərində olduğu kimi string sətirlərdə də getline funksiyasından istifadə etməliyik. Lakin string sətirləri üçün getline funksiyasından istifadə bir qədər fərqlənir. C sətirləri üçün istifadə etdiyimiz getline funksiyasının string sətirləri üçün istifadə edə bilmərik. string sətirləri üçün getline funksiyasını aşağıdakı kimi çağırmalıyıq:

```
getline(cin, str);
```

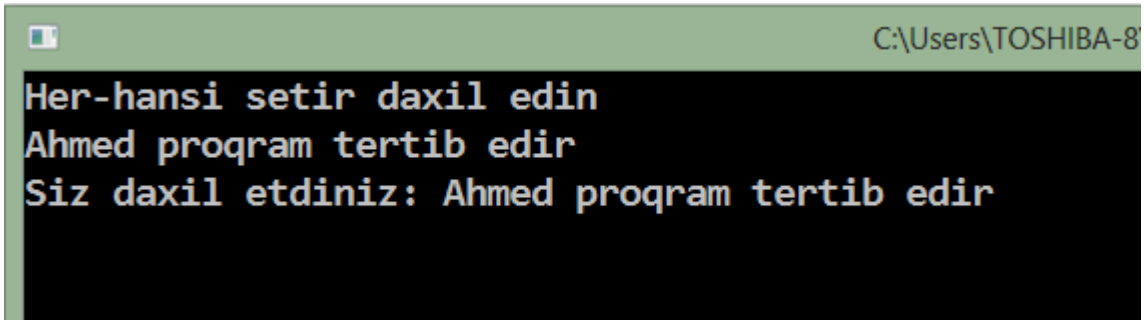
Burada ilk parametr məlumatın qəbul olunduğu mənbəni, ikinci parametr isə qbul olunan məlumatın yerləşdirilməli olduğu yeri bildirir. Yada salaq ki C sətirlərində getline funksiyasını aşağıdakı kimi çağırırdıq:

```
cin.getline(setir, 256);
```

Burada getline funksiyasının ilk parametri (setir) məlumatın yerləşdirilməli olduğu yeri, ikinci parametr isə (256) həmin yerdə yerləşdirilməli olan maksimal uzunluğu bildirirdi. Qeyd edim ki string tipləri ilə işləyən getline sətirləri müxtəlif mənbələrdən qəbul edə bilər. Faylları keçərkən bu məsələyə yenidən qayıdacayıq. İndi isə getline funksiyası ilə istifadəçinin daxil etdiyi sətri format etmədən tam şəkildə qəbul edən kod ilə tanış olaq:

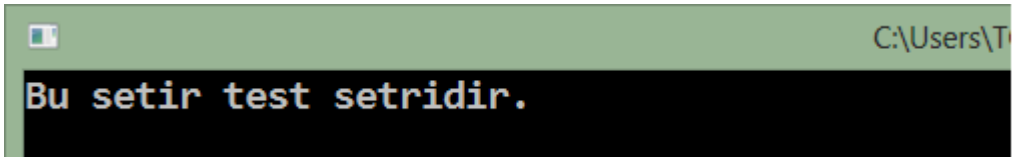
```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
```

```
int main() {

    string str;

    cout << "Her-hansi setir daxil edin\n";
    getline(cin, str);

    cout << "Siz daxil etdiniz: "
         << str
         << "\n";

}
```



string tipi bir neçə sətri üstəgəl operatoru ilə bir-birinə “birləşdirə” bilərik. Nümunə koda baxaq:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {

    string str1, str2, str3;

    str1 = "Bu setir ";
    str2 = "test setridir.";

    str3 = str1 + str2;

    cout << str3;


}
```



Ümumiyyətlə isə bütün bu əməliyyatları string sətirləri ilə işləmək üçün tərtib olunmuş funksiyalar ilə həll etmək olar. Bu funksiyaların bəziləri ilə tanış olaq.

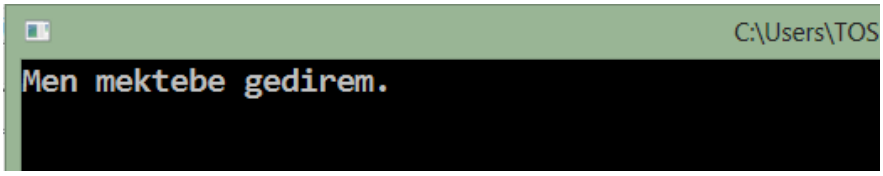
```
string& append (const string& str);

string& insert (size_t pos, const string& str);

string& replace (size_t pos,  size_t len,  const string& str);

const char* c_str() const;
```

10.2.1 append funksiyası

```
string& append (const string& str);
```

append funksiyası verilmiş sətrin sonuna parametr kimi ötürülən sətri əlavə edir. Nümunə koda baxaq:

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
  string str = "Men mektebe ";

  str.append("gedirem.");

  std::cout << str << '\n';

  return 0;
}
```

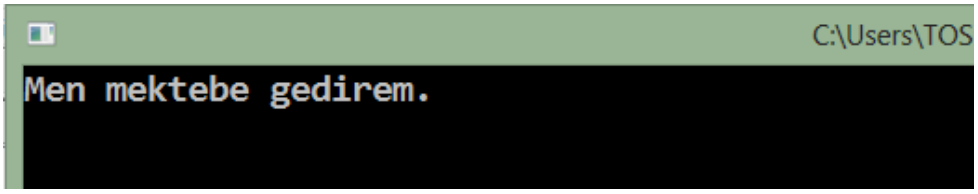


10.2.2 insert funksiyası

```
string& insert (size_t pos, const string& str);
```

insert funksiyası pos ünvanından(pos- uncu simvoldan dərhal sonra) başlayaraq parametr kimi ötürülən sətri verilmiş sətrin arasına yerləşdirir. Nümunə koda baxaq.

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
  string str = "Men gedirem.";

  str.insert(4, "mektebe ");

  cout << str << '\n';

  return 0;
}
```

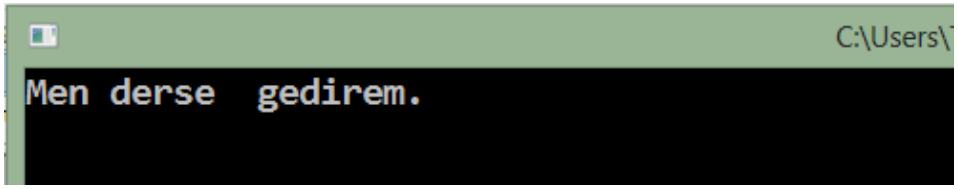


10.2.3 replace funksiyası

```
string& replace (size_t pos,  size_t len,  const string& str);
```

replace funksiyası verilmiş sətrin pos ünvanından başlayan və uzunluğu len olan hissəsini parametr kimi verilmiş sətirlə əvəz edir.

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
  string str = "Men mektebe gedirem.";

  str.replace(4, 7, "derse ");

  cout << str << '\n';

  return 0;
}
```



10.2.4 c_str funksiyası

```
const char* c_str() const;
```

c_str funksiyası verilmiş string tipli sətri C tipli sətrə çevirmək üçün istifadə olunur.

```
#include <iostream>
#include <cstring>
#include <string>

using namespace std;

int main ()
{
  string str ("Men mektebe gedirem.");

  char * cstr = new char [str.length()+1];
  strcpy (cstr, str.c_str());

  char * p = strtok (cstr," ");
  while (p!=0)
  {
    cout << p << '\n';
    p = strtok(NULL," ");
  }

  delete[] cstr;
  return 0;
}
```

Çalışmalar.

1. İstifadəçinin daxil etdiyi sətrin uzunluğunu çap edən proqram tərtib edin.

2. İstifadəçinin daxil etdiyi sətrin son 5 simvolunu ekranda çap edən proqram tərtib edin.

3. İstifadəçinin daxil etdiyi sətrin ilk 3 simvolu ilə son 5 simvolunu birləşdirib çap edən proqram tərtib edin.

4. El proqram qurun ki, istifadəçinin daxil etdiyi sətrin 5-ci simvolu ilə 15-ci simvolu arasında qalan hissəsini çap etsin.

5. Elə proqram tərtib edin ki, istifadəçidən 3 sətir qəbul etsin və bu sətirləri ardıcıl birləşdirərək tam sətir kimi çap etsin.

6.* Elə proqram tərtib edin ki, istifadəçidən 4 sətir qəbul etsin və bu sətirləri daxil olma sırasının əksi ardıcıllığında birləşdirərək tam sətir kimi çap etsin.

7.* Elə proqram tərtib edin ki, istifadəçidən 4 sətir qəbul etsin və bu sətirləri uzunluqlarının artma ardıcıllığı ilə alt-alta çap etsin.

8. Şifrə proqramı. İstifadəçinin daxil etdiyi şifrənin düzgünlüyünü yoxlayan proqram tərtib edin.

§11 Struct tiplər.

11.1 Struct tiplər

İndiyə qədər dəyişən elan edərkən biz standart tiplərdən istifadə edirdik. Bu bölmədə isə biz yeni tip yaratmağı örgənəcəyik. Əvvəl görək yeni tip yaratmaq nəyə lazımdır. Misal üçün tutaq ki bizdən proqramda hər- hansı qurğunun müxtəlif parametrlərini – adı, çəkisi, qiyməti v.s. yadda saxlamaq tələb olunur. Əlbəttə bunun üçün biz müvafiq tiplərdən müxtəlif dəyişənlər elan edə bilərik:

```
char ad[100];
int qiymet;
double ceki;
```

Bir qurğunun parametrlərini yadda saxlamaq üçün 3 müxtəlif dəyişən elan etdik. Ölçülər, istehsal tarixi v.s. kimi parametrləri də yadda saxlamaq tələb olunsaydı elan etməli olduğumuz dəyişənlərin sayı müvafiq surətdə artardı. Cərgələrdən istifadə etməklə müxtəlif sayda qurğu parametrlərin yadda saxlaya bilərdik. Lakin bütün bunlar bir növ kodda səliqəsizliyə gətirib çıxarır. Daha praktik həllə ehtiyac yaranır. C++ dilində bu cür həllər üçün yeni tiplər yaradırlar və tələb olunan parametrləri həmin tipdə birləşdirirlər, aşağıdakı kimi:

```
yeni tip {

  char ad[100];
  int qiymet;
  double ceki;

}
```

Yeni yartadığımız tipə istənilən ad verə bilərik, lakin çox vaxt təsvir etdiyimiz obyektə uyğun ad veririk. Misal üçün yuxarıdakı məsələnin həllinə uyğun yaradacağımız tipə `qurqu` adını verə bilərik. Yeni tipə verdiyimiz adın əvvəlinə sintaksis tələbi olaraq `struct` açar sözünü yazmalıyıq:

```
struct qurqu
```

Daha sonra fiqurlu mötərizələr və nöqtəvergül qoyuruq:

```
struct qurqu {

};
```

Yeni tipə aid olan bütün parametrlər və onların müvafiq tiplərini fiqurlu mötərizələr arasına yerləşdiririk:

```
struct qurqu {
	char ad[100];
	int qiymet;
	double ceki;
};
```

Beləliklə artıq proqramımızda `qurqu` adında yeni tip yaratmış olduq. Yeni yaratdığımız tipin daxilində elan olunan dəyişənlərə yeni tipin həddləri deyilir. Yuxarıdakı nümunədə yaratdığımız `qurqu` tipi `char[100]` tipli **ad**, `int` tipli **qiymet** və `double` tipli **ceki** həddlərindən ibarətdir.

Çalışma . C++ dilində `int` tipli **en** və **uz** adlı həddlərdən ibarət `duzb` adli yeni tip yaradın.

Həlli: Yeni tip yaratmaq üçün əvvəlcə `struct` açar sözünü yazırıq və daha sonra yaratmaq istədiyimiz tipin adını yazırıq:

```
struct duzb
```

Daha sonra fiqurlu mötərizələr yazıb nöqtəvergül qoyuruq.

```
struct duzb {

};
```

Indi qalır yeni yaratdığımız tipin həddlərini əlavə etmək.Yeni yaratdığımız `duzb` tipinin `int` tipli iki həddi var, **en** və **uz** adında. Onları da yerləşdirək fiqurlu mötərizələr içinə:

```
struct duzb {
	int en;
	int uz;
};
```

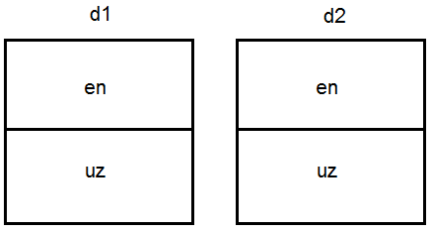
Vəssalam, yeni tipimiz hazırdır.

11.1.1 Yeni tipdən dəyişən elan etmək

Yeni tip yaratdıqdan sonra bu tipdən adi qaydada (əvvəlcə tipin adı, sonra dəyişənin adı) dəyişənlər və cərgələr elan edə bilərik. Misal üçün yuxarıda yaratdığımız `duzb` tipindən d1 və d2 adlı iki dəyişən elan edək:

```
duzb d1, d2;
```

Əslində biz yeni tip yaradanda yaddaşda heç bir yer ayrılmır. Sadəcə gələcək elanlar üçün hazırlıq görülür. Yaddaşda yer məhs yeni yaratdığımız tipdən dəyişən elan etdikdə ayrılır. Misal üçün yuxarıda `duzb` tipindən elan etdiyimiz **d1** və **d2** dəyişənləri üçün yaddaşda aşağıdakı kimi yer ayrılar:



Yeni tipdən yaradılan dəyişənə onun tərkibinə daxil olan bütün həddlərin yaddaşda tutduğu yerlərin cəmi qədər yer ayrılır. Yəni əgər int tipi 4 bayt yer tutsa onda d1 və d2 dəyişənlərinin hər birinə yaddaşda 8 bayt yer ayrılar. Yeni yaradılan tiplər özündə müxtəlif digər tiplərdən olan həddlər saxladığına görə adlanır strukt tiplər, bu tiplərdən elan olunan dəyişənlər adlanır obyekt . Bu andan etibarən biz də yeni yaratdığımız tiplərdən olan dəyişənləri obyekt adlandıracağıq. Yəni yuxarıdakı çalışmada `duzb` **strukt** tipindən **d1** və **d2** obyektləri yaradılıb.

11.1.2 Obyektin həddlərinə müraciət.

Obyektin həddələrindən adi dəyişənlərdən istifadə etdiyimiz kimi istifadə edirik. Obyektin istənilən bir həddinə müraciət etmək üçün əvvəlcə obyektin adını, daha sonra həddə müraciət operatoru ola nöqtə simvolunu və müraciət etmək istədiyimiz həddin adını yazırıq. Misal üçün yuxarıda elan etdiyimiz d1 obyektinin en həddinə müraciət etmək üçün yazmalıyıq:

```
d1.en
```

Eyni qayda ilə d1 obyektinin uz həddinə müraciət etmək üçün yazmalıyıq:

```
d1.uz
```

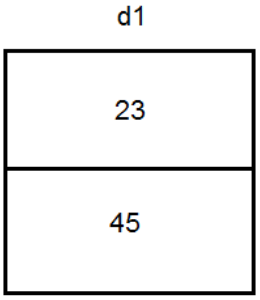
Obyektin həddlərinə müraciət etməyin qaydası ilə tanış olduqdan sonra artıq onlardan istifadəyə aid kod nümunələri ilə tanış olmaq olar. Misal üçün d1 obyektinin en həddinə 23 qiymətini mənimsədən kod aşağıdakı kimi olar:

```
d1.en = 23;
```

Eyni qayda ilə uz həddinə 45 qiyməti mənimsədən kod aşağıdakı kimi olar:

```
d1.en = 45;
```

Yaddaşa nəzər salaq:



Aşağıdakı nümunə proqramda yuxarıda elan olunan d1 və d2 obyektlərinin en və uz həddlərinə istifadəçi tərəfindən qiymətlər mənimsədilir və onların qiymətləri çap edilir.

```
#include <iostream>

using namespace std;

struct duzb {
	int en;
	int uz;
};

int main()
{
	duzb d1, d2;

	cout << "d1 obyektinin en heddinin qiymetini daxil edin \n";
	cin >> d1.en;

	cout << "d1 obyektinin uz heddinin qiymetini daxil edin \n";
	cin >> d1.uz;

	cout << "d2 obyektinin en heddinin qiymetini daxil edin \n";
	cin >> d2.en;

	cout << "d2 obyektinin uz heddinin qiymetini daxil edin \n";
	cin >> d2.uz;

	cout << "d1 obyektinin heddleri: \n";
	cout << "en = " << d1.en << " uz = " << d1.uz << "\n";

	cout << "d2 obyektinin heddleri: \n";
	cout << "en = " << d2.en << " uz = " << d2.uz << "\n";

}
```

Nəticə:

```
d1 obyektinin en heddinin qiymetini daxil edin
34
d1 obyektinin uz heddinin qiymetini daxil edin
56
d2 obyektinin en heddinin qiymetini daxil edin
78
d2 obyektinin uz heddinin qiymetini daxil edin
90
d1 obyektinin heddleri:
en = 34 uz = 56
d2 obyektinin heddleri:
en = 78 uz = 90
```

Aşağıdakı nümunədə `char`[25] tipindən olan ad həddi, `char`[25] tipindən olan soyad həddi və `int` tipli yash həddlərindən ibarət olar insan tipi yaradılır. İnsan tipindən Ahmed, Ali və Tofiq adlı obyektlər yaradılır. Obyektlərin həddlərinə qiymətlər mənimsədilir və onlar çap olunur.

```
#include <iostream>

using namespace std;

//insan adli yeni tip teyin edirik
struct insan {
	char ad[25];
	char soyad[25];
	int yash;
};

int main()
{
	//insan tipinden 3 dene deyishen elan edirik(obyekt yaradiqir)
	insan direktor, muhendis1, muhendis2;

	//direktorun melumatlari
	cout  << "direktor-un melumatlarini daxil edin \n"
	      << "ad, soyad, yash\n";
	cin   >> direktor.ad
	      >> direktor.soyad
	      >> direktor.yash;

	//muhendis1-in melumatlari
	cout  << "muhendis1-in melumatlarini daxil edin \n"
	      << "ad, soyad, yash\n";
	cin   >> muhendis1.ad
	      >> muhendis1.soyad
	      >> muhendis1.yash;

	//muhendis2-in melumatlari
	cout  << "muhendis2-in melumatlarini daxil edin \n"
	      << "ad, soyad, yash\n";
	cin   >> muhendis2.ad
	      >> muhendis2.soyad
	      >> muhendis2.yash;
```

```
    //melumatlari cap edek

    cout  << "direktor-un melumatlari: \n"
          << "ad \t: " << direktor.ad  << "\n"
          << "soyad \t: "   << direktor.soyad << "\n"
          << "yash \t:  "   << direktor.yash  << "\n\n";

    cout  << "muhendis1-in melumatlari: \n"
          << "ad \t: "      << muhendis1.ad   << "\n"
          << "soyad :\t: "  << muhendis1.soyad << "\n"
          << "yash \t: "    << muhendis1.yash << "\n\n";

    cout  << "muhendis2-in melumatlari: \n"
          << "ad \t: "      << muhendis2.ad   << "\n"
          << "soyad \t: "   << muhendis2.soyad << "\n"
          << "yash \t: "    << muhendis2.yash << "\n";

}
```

Nəticə:

```
direktor-un melumatlarini daxil edin
ad, soyad, yash
Ahmed Aliyev 45
muhendis1-in melumatlarini daxil edin
ad, soyad, yash
Rahib Mehdiyev 36
muhendis2-in melumatlarini daxil edin
ad, soyad, yash
Edalet Memmedov 52
direktor-un melumatlari:
ad      : Ahmed
soyad   : Aliyev
yash    : 45

muhendis1-in melumatlari:
ad      : Rahib
soyad   : Mehdiyev
yash    : 36

muhendis2-in melumatlari:
ad      : Edalet
soyad   : Memmedov
yash    : 52
```

Obyektlərlə tanışlıq üçün yaxşı nümunə olsa da, praktik cəhətdən bu nümunənin bəzi çatışmamazlıqları var. Eyni kod təkrar-təkrar müxtəlif obyektlərin həddlərini daxil etmək üçün istifadə olunur. Əslində funksiyalardan istifadə etməklə biz bu nümunəni xeyli dərəcədə həm başadüşülən, həm-də az həcmli edə bilərdik. Növbəti bölmələrdə biz bu məsələ üzərində çalışacayıq. Oxunaqlı və asan başadüşülən kod əldə etmənin qaydası ilə tanış olmaqla yanaşı, obyektlərin funksiyalara parametr kimi ötürülməsini örgənmiş olacayıq.

11.1.3 Obyektlərin funksiyalara ötürülməsi

Strukt tipli dəyişənlər adi dəyişənlər kimi funksiyalara ötürülə bilər. Bu zaman həm qiymətə, həm də ünvana görə ötürülmədən istifadə etmək olar. Obyektlərin funksiyaya parametr kimi ötürülməsinə misal göstərmək üçün yuxarıda tərtib etdiyimiz nümunəyə qayıdaq. Yeni bir cap_et() funksiyası təyin edək ki, arqument olaraq `insan` tipindən obyekt qəbul eləsin və bu obyektin həddlərini çap etsin. Funksiyanın elanı aşağıdakı kimi olar:

```
//insan tipinden olan deyishenlerin heddlerini cap eden funksiya
void cap_et(insan adam) {

	cout  << "ad \t: "    << adam.ad    << "\n"
	      << "soyad \t: " << adam.soyad << "\n"
	      << "yash \t:  " << adam.yash  << "\n\n";

}
```

cap_et funksiyasından istifadə etməklə proqramımızda derektor və mühəndislərin məlumatlarını çap edə bilərik. Bu zaman kod aşağıdakı kimi sadələşər:

```
#include <iostream>

using namespace std;

//insan adli yeni tip teyin edirik
struct insan {
	char ad[25];
	char soyad[25];
	int yash;
};

//insan tipinden olan deyishenlerin heddlerini cap eden funksiya
void cap_et(insan adam) {

	cout  << "ad \t: " << adam.ad << "\n"
	      << "soyad \t: " << adam.soyad << "\n"
	      << "yash \t:  " << adam.yash << "\n\n";

}

int main()
{
	//insan tipinden 3 dene deyishen elan edirik(obyekt yaradiqir)
	insan direktor, muhendis1, muhendis2;

	//direktorun melumatlari
	cout  << "direktor-un melumatlarini daxil edin \n"
	      << "ad, soyad, yash\n";
	cin   >> direktor.ad
	      >> direktor.soyad
	      >> direktor.yash;
```

```
	//muhendis1-in melumatlari
	cout   << "muhendis1-in melumatlarini daxil edin \n"
	       << "ad, soyad, yash\n";
	cin    >> muhendis1.ad
	       >> muhendis1.soyad
	         >> muhendis1.yash;

	//muhendis2-in melumatlari
	cout   << "muhendis2-in melumatlarini daxil edin \n"
	       << "ad, soyad, yash\n";
	cin    >> muhendis2.ad
	       >> muhendis2.soyad
	       >> muhendis2.yash;

	//melumatlari cap edek
	cout << "direktor-un melumatlari: \n";
	cap_et(direktor);

	cout << "muhendis1-in melumatlari: \n";
	cap_et(muhendis1);

	cout << "muhendis2-in melumatlari: \n";
	cap_et(muhendis2);

}
```

Gördüyümüz kimi bu kodda biz `direktor` , `muhendis1` və `muhendis2` obyektlərini `cap_et` funksiyasına qiymətə görə ötürürük. Lakin bizim proqramımızın daha da sadələşdirilməyə ehtiyacı var. Bunu necə edə bilərik. Obyektlərin həddlərini daxil etmək üçün daxil_et() adlı funksiya yaradaq. İlk baxışda biz insan tipli obyektin həddlərinə qiymət mənimsədən daxil_et() funksiyasını aşağıdakı kimi tərtib etməliyik:

```
void daxil_et(insan adam) {

	cin  >> adam.ad
	     >> adam.soyad
	     >> adam.yash ;

}
```

Lakin bu şəkildə daxil_et funksiyası ilə insan tipindən olan obyektlərin həddlərinə mənimsətdiyimiz qiymətlər funksiya qayıtdıqdan sonra qüvvədən düşmüş olur. Səbəb isə qiymətə görə ötürülmədir. Ona görə əgər biz istəyiriksə ki, funksiya qayıtdıqdan sonra ona ötürülən obyekt parametrinin həddlərinə mənimsətdiyimz qiymətlər itməsin, onda biz obyekti ünvana görə ötürməliyik. Ünvana görə ötürülmənin qaydası ilə funksiyalar mövzusunda tanış olmuşuq. Burada da eyni qayda ilə dəyişənin adının əvvəlinə & simvolu artırırıq, aşağıdakı kimi:

```
void daxil_et(insan &adam) {
```

```
	cin >> adam.ad
		>> adam.soyad
		>> adam.yash ;

}
```

Tam proqram kodu isə aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

//insan adli yeni tip teyin edirik
struct insan {
	char ad[25];
	char soyad[25];
	int yash;
};

//insan tipinden olan deyishenlerin heddlerini cap eden funksiya
void cap_et(insan adam) {

	cout  << "ad \t: " << adam.ad    << "\n"
		<< "soyad \t: " << adam.soyad << "\n"
		<< "yash \t: " << adam.yash  << "\n\n";

}

void daxil_et(insan &adam) {

	cin   >> adam.ad
		>> adam.soyad
		>> adam.yash ;

}

int main()
{
	//insan tipinden 3 dene deyishen elan edirik(obyekt yaradiqir)
	insan direktor, muhendis1, muhendis2;

	//direktorun melumatlari
	cout  << "direktor-un melumatlarini daxil edin \n"
		<< "ad, soyad, yash\n";
	daxil_et(direktor);

	//muhendis1-in melumatlari
	cout  << "muhendis1-in melumatlarini daxil edin \n"
		<< "ad, soyad, yash\n";
	daxil_et(muhendis1);

	//muhendis2-in melumatlari
	cout  << "muhendis2-in melumatlarini daxil edin \n"
		<< "ad, soyad, yash\n";
	daxil_et(muhendis2);
```

```
	//melumatlari cap edek

	cout << "direktor-un melumatlari: \n";
	cap_et(direktor);


	cout << "muhendis1-in melumatlari: \n";
	cap_et(muhendis1);

	cout << "muhendis2-in melumatlari: \n";
	cap_et(muhendis2);

}
```

11.2 İç-içə strukt tiplər

Strukt tip yaradarkən ancaq standart tiplərdən istifadə etdik. Bəs strukt tipin həddləri olaraq hər-hansı başqa strukt tipdən istifadə etmək olarmı? Əlbəttə. Bunu aşağıdakı nümunə üzərindən izah edək. Əvvəlcə özündə sətir tipindən olan şəhər, rayon , küçə və int tipindən olan mənzil həddlərini saxlayan ünvan adlı yeni strukt tip yaradaq:

```
//unvan tipi
struct unvan {
	char sheher[100];
	char rayon[100];
	char kuce[100];
	int menzil;
};
```

Daha sonra yeni ev adlı strukt tipi yaradaq. ev tipinin həddləri int tipli otaq_say, double tipli sahe və yuxarıda yaratdığımız unvan tipindən olan unv həddi olsun.

```
//ev adli yeni tip teyin edirik
struct ev {
	int otaq_say;
	double sahe; //olcu vahidi sot
	unvan unv;
};
```

Gördüyümüz kimi ev strukt tipinin unv adlı həddi özü də unvan adlı strukt tipindəndir. Bu cür təyin olunma, yəni bir strukt tipin digərinin içində təyin olunması iç-içə təyin olunmuş strukt tip adlanır. Bu şəkildə iç-içə istənilən ierarxik dərinlikli strukt tip təyin etmək olar.

Yeni yaratdığımız ev tipindən obyektlər elan edək və onların həddlərinə müraciət edək.

```
#include <iostream>

using namespace std;
```

```
//unvan tipi
struct unvan {

	char sheher[100];
	char rayon[100];
	char kuce[100];
	int menzil;
};



//ev adli yeni tip teyin edirik
struct ev {

	int otaq_say;
	double sahe; //olcu vahidi sot
	unvan unv;
};



int main()
{


	ev ev1;

	cout << "Zehmet olmasa ev1 - in heddlerini daxil edin:\n";

	cout << "Otaqlarin sayi: ";
	cin >> ev1.otaq_say;

	cout << "sahe: ";
	cin >> ev1.sahe;

	cout << "sheher: ";
	cin >> ev1.unv.sheher;

	cout << "rayon: ";
	cin >> ev1.unv.rayon;

	cout << "kuce: ";
	cin >> ev1.unv.kuce;

	cout << "menzil: ";
	cin >> ev1.unv.menzil;

	cout
		<< "\nev1-in heddleri\n\n"
		<< "otaqlarin sayi \t" << ev1.otaq_say << "\n"
		<< "sahe \t" << ev1.sahe << "\n"
		<< "unvan: \t"
		<< ev1.unv.sheher << " sheheri , "
		<< ev1.unv.rayon << " rayonu , "
		<< ev1.unv.kuce << " kucesi , menzil "
```

```
            << ev1.unv.menzil;

}
```

Nəticə:

```
Zehmet olmasa ev1 - in heddlerini daxil edin:
Otaqlarin sayi: 5
sahe: 3.4
sheher: Baki
rayon: Nizami
kuce: S.Eliyev
menzil: 12

ev1-in heddleri

otaqlarin sayi  5
sahe    3.4
```

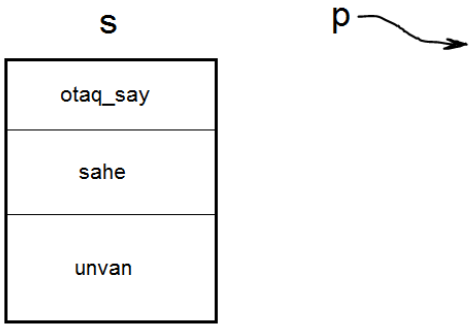
11.3 Strukt tipindən olan göstəricilər

Strukt tipindən də digər tiplərdən olduğu kimi adi dəyişənlərlə yanaşı göstəricilər elan edə bilərik. Bunun üçün eyni qaydadan istifadə etməliyik. Göstəricinin adının əvvəlinə ulduz - * işarəsi artırmalıyıq.

Yuxarıda elan etdiyimiz ev adlı strukt tipindən s obyekti və p göstəricisi elan edək və yaddaşdakı vəziyyəti təhlil edək.

```
ev s, *p;
```

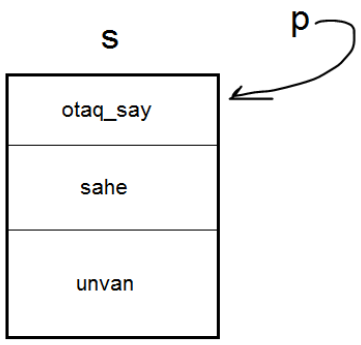
Yaddaşa nəzər salaq:



Gördüyümüz kimi s obyekti üçün yaddaşda həddlərini yadda saxlamaq üçün yer ayrılıb. p göstəricisi isə hələlik hara istinad elədiyi məlum deyil. Bu cür elandan sonra p gəstəricisinə s obyektinin ünvanın mənimsədə bilərik:

```
p = &s;
```

Nəticədə p göstəricisi s obyektinə istinad edər:



Bu şəkildə strukt tipdən olan göstəriciyə həmin tipdən olan obyektin ünvanın mənimsətdikdən sonra artıq gəstəricidən istifadə etməklə obyektin həddlərinə müraciət edə bilərik.

Yadımıza salaq ki, adi obyektlərin həddlərinə müraciət edərkən abyektin adı ilə, müraciət etmək istədiyimiz həddin arasına nöqtə işarəsi qoyurduq. Göstərici ilə müraciət etdiyimiz zaman göstərici ilə həddin arasına çıxaq və böyükdür işarəsini qoymalıyıq, bu şəkildə:

Göstərici->Hədd

Misal üçün s obyektinin otaq_say həddinə p göstəricisi ilə 5 qiyməti manimsətmək istəsək, aşağıdakı kimi yaza bilərik:

```
p->otaq_say = 5;
```

p göstəricisi ilə s obyektinin unv həddinə də müraciət etmək üçün eyni üsuldan istifadə edəcəyik:

```
p->unv
```

Lakin, s –in daxilində yerləşən unv- ın həddlərinə, misal üçün şəhər həddinə müraciət etmək üçün isə artıq nöqtə operatorundan istifadə etməliyik:

```
p->unv.sheher
```

Əgər biz ev tipinin daxilində unv həddinin unvan tipindən olan göstərici kimi elan etsəydik

```
struct ev {
	int otaq_say;
	double sahe;
	unvan *unv;
};
```

Onda sheher heddine müraciət etmək üçün yazmalı olardıq:

```
p->unv->sheher
```

11.4 Dinamik yaradılma

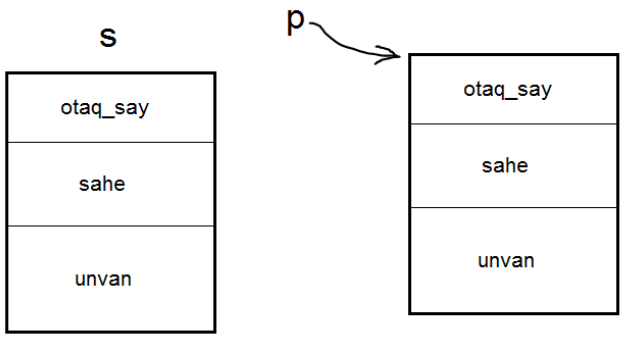
Strukt tipindən olan göstəricilərə dinamik olaraq yer ayıra bilərik. Bunun üçün `new` operatorundan istifadə olunur. Sintaksis aşağıdakı kimidir:

Göstərici = `new` strukt tipi;

Misal üçün yuxarıda elan etdiyimiz p göstəricisinə dinamik yer ayırmaq üçün yazmalıyıq:

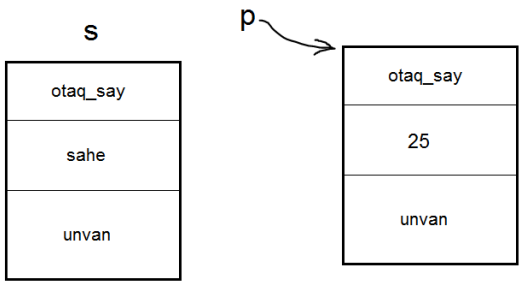
```
p = new ev;
```

Bu zaman yaddaşda ev tipindən olan dinamik obyekt yaradılacaq və p göstəricisi həmin obyektə istinad edəcək:



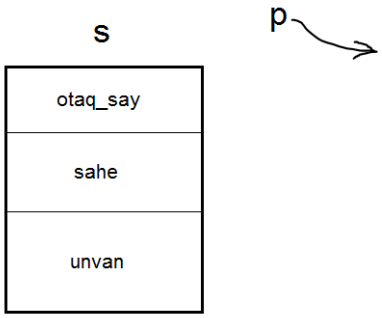
Bundan sonra p göstəricisi ilə dinamik yaradılmış obyektin həddlərinə müraciət edə bilərik:

```
p->otaq_say = 25;
```



Daha sonra proqramın artıq bu dinamik yaradılmış obyektə ehtiyac qalmayanda onun üçün ayrılmış yeri `delete` əmri ilə azad edə bilərik:

```
delete p;
```



III HİSSƏ

Obyekt Yönümlü Proqramlaşdırma

§12 Siniflər.

12.1 Siniflər

Bu mövzuda biz **Obyekt Yönümlü Proqramlaşdırma** (OYP) ilə tanış olacağıq. OYP bir çox proqramlaşdırma dilləri tərəfindən dəstəklənir. OYP kifayət qədər maraqlı proqramlaşdırma üsuludur, lakin arxitektura səviyyəsində dəstəklənmədiyinə görə bəzi performans itkilərinə səbəb olur(sürət və yaddaş baxımından). Yəni qeyri OYP proqramları(posedur yönümlü) OYP proqramlarına nisbətən daha sürətli olurlar və daha az yaddaşdan istifadə edirlər. Əlbəttə əgər siz oyun proqramı, kompilyator, anitivirus və ya hər-hansı digər resurs meyilli proqram tərtib etmirsinizsə bunun elə də fərqi olmur və siz rahatlıqla OYP –dən istifadə edə bilərsiniz.

Görək OYP nədir?

Univeristetdə OYP-ni keçəndə düzü mən heçnə başa düşməmişdim. Elə indiyə kimi də (prosedural yönümlü proqramçı olaraq) mənə tam çatmır ki, bu qədər əslində aydın olan məsələləri önə çəkmək nəyə lazımdır. Amma hər halda bu qaydalardan müasir proqramlaşdırmada istifadə olunur, ona görə biz də onları bilməliyik.

Adətən OYP kanseptini aşağıdakı bir neçə məhfumun birləşməsi kimi təsvir edirlər:

İnkapsulyasiya

Abstraksiya

Varislik

Polimorfizm

Əsas olanları hələlik bunlardı və bizdən də bunları başa düşmək tələb olunur. İzah da tam aydın olmaya bilər, çünki əminəm ki real həyata aid olan bu anlayışların proqramlaşdırmada uyğunluğunu tapmaq ilk dəfəyə elə də asan olmaya bilər. Amma kod izahında tətbiqi sizə aydın olmalıdır. İndi mən sizə bu anlayışların hər birinin qısa olaraq izahını verəcəm, lakin irəlidə artıq proqramlaşdırmada nə cür tətbiq olunmalarını konkret nümunələr üzərində nəzərdən keçirəcəyik.

İnkapsulyasi – İnkapsulyasiya dedikdə məlumat və bu məlumat üzərində yerinə yetirilməli olan əməliyyatların vahid tərkib formasında birləşməsi nəzərdə tutulur. Yəni verilmiş məlumatlar üzərində aparılmalı olan əməliyyatları tapmaq üzün başqa heç yeri axtarmalı deyilsən. Real həyatdan nümunə göstərsək misal üçün əvvəllər musiqiyə qulaq asmaq üçün elektromaqnit lentli kasseti yerləşdirirdik maqnitofona və dinləyirdik. Burada kasset və maqnitofon ayrı-ayrı idi. Lakin indi mobil telefonumuzda həm dinləmək istıdimiz musiqinin ikili faylı(mp3 ...) həm də həmin faylı oxuda biləcək proqram və səsləndirmə qurğusu var. Bu adlanır inkapsulyasiya. Bu cür misallar çox gətimək olar.

Abstaksiya – Abstraksiya işin detalların bizdən gizlədərək yalnız bizim istifadə etdiklərimizi göstərməkdir. Misal üçün mən telefonda hansısa nömrəni yığıb yes düyməsinə basıram. Zəng çalınır, amma yes düyməsin basdıqdan sonra hansı proseslərin getdiyi, nə cür getdiyi mənə aydın deyil. Bunlar hamısı alt fonda yerinə yetirilir, mənə ancaq lazımlı nəticə təqdim olunur.

Varislik – Varislik müəyyən tərtib olunmuş kodun bir daha təkrar yazılmasının qarşısını almaq üçündür. Deyək ki velosipedi ikinci dəfə kəşf etməmək məsələsinə oxşayır. Misal üçün samsung 4 çıxır, sonra samsung 5-in üzərində işləyəndə daha hər şeyi 0-dan başlamırlar, samsung 4-dən olan funksiyaları hazır götürüb daha da təkmilləşdirirlər.

Polimorfizm - Eyni bir obyektin vəziyyətdən asılı olaraq özünü müxtəlif cür aparmasıdır, çoxformalı deməkdir. Bunu eyni adlı əməliyyatın vəziyyətdən asılı olaraq müxtəlif formalarda icra olunması kimi başa düşmək lazımdır. Yəni tutaq işin adı nəqliyyat vastəsini idarə etməkdir. Tutaq ki, elə sürücü var ki, həm motosiklet, həm yük maşını, həm kater və eləcə də təyyarəni idarə edə bilir (Xarici kinolardakı agentlər kimi). Bu sürücüdə bu nəqliyyat vastələrinin hər birini idarəetmə bacarığı var. Lakin motoskilet idarəetmək lazım olanda o özünü motosiklet sürücüsü, təyyarə idarə etmək lazım olanda o özünü pilot v.s. kimi aparır. Yəni eyni bir iş – nəqliyyat vastəsinin idarə edilməsi işi müxtəlif formalarda yerinə yetirilir. Konkret hansı formanın seçilməsi isə vəziyyətə görə müəyyən olunur.

Keçək ilk sinfimizi yaratmağa.

12.2 İlk sinif

Biz qeyd elədik ki, OYP-də məlumatlar və bu məlumat üzərində icra olunan əməliyyatlar vahid tərkibdə birləşir və buna inkapsuyasiya deyilir. Həmin bu tərkib - Sinif adlanır. Siniflərin tərtibi əvvəlki başlıqda keçdiyimiz strukt tiplərə oxşayır. Sintaksis aşağıdakı kimidir:

```
class Sinfin_Aadı {


};
```

Sinif elan etmək üçün əvvəlcə class açar sözünü yazırıq, yeni yaratdığımız sinfə verdiyimiz adı, daha sonra isə fiqurlu mötərizə qoyub, nöqtəvergül ilə tamamlayırıq. Sinfə istədiyimiz adı verə bilərik. Sinfin daxilində onun həddləri elan olunur. Sinfin iki cür həddləri ola bilər: dəyişən həddləri və funksiya həddləri. Sinfin həddləri müraciət edilmə xassələrinə görə fərqlənirlər. Misal üçün elə həddlər ola bilər ki, onlara sinfindən xaricdə müraciət etmək olmaz, yalnız sinfin daxilinə görünməlidi. Elə həddlər (funksiya və məlumat) onlara proqramın istənilən yerindən, eləcə də digər siniflərdən

müraciət etmək olar. Elə həddlər ola bilər ki, varis siniflər onları görməməlidir v.s. Bütün bu tip müraciətlərə nəzarəti təmin eləmək üçün C++ -da sinfin müraciət tənzimləyicilərindən istifadə olunur. C++ dilində sinfin 3 müraciət tənzimləyiciləri var, bunlar aşağıdakılardır:

`public`

`private`

`protected`

Həddlər bu üç tənzimləyicidən birinə məxsus olmalıdır. Biz qısa olaraq ilk əvvəl istifadə edəcəyimiz `public` tənzimləyicisinin izahını verəcəyik. `private` və `protected` tənzimləyiciləri ilə bir qədər sonra tanış olacayıq.

`public` - ümumi

public kimi elan olunan həddlərə proqramın istənilən yerindən müraciət etmək olar. Eləcə də digər siniflərdən. Həddi public elan etmək üçün sinfin daxilində public yazıb qoşanöqtə qoyuruq. Daha sonra sinfin hamı tərəfindən müraciət olunmasına icazə verilən həddlərini yazırıq, aşağıdakı kimi:

```
class Sinfin_Adı {

public:
	//umumi heddler bura yazilir

};
```

Gəlin int tipli en və int tipli uzunluq adlı public həddlərdən ibarət olan duzbucaqli adli yeni sinif tipi yaradaq. Yuxarıdakı izaha əsasən duzbucaqli sinfinin elanı aşağıdakı kimi olar:

```
class duzbucaqli {

public:

	int en;
	int uzunluq;

};
```

12.3 Obyektlərin Yaradılması

Sinif yaratmaqla biz yeni bir tip yaratmış oluruq. Bu tipdən digər standart tiplərdən və struct tiplərdən olduğu kimi dəyişənlər elan edə bilərik. Struct tipində də olduğu kimi, sinif tipindən də elan etdiyimiz dəyişənləri obyekt adlandıracağıq.

Aşağıdakı kodda yeni yaratdığımız duzbucali adlı sinif tipindən d1 obyektini yradırıq.

```
	duzbucaqli d1;
```

Obyekti yaratdıqdan sonra onun həddlərinə müraciət edə bilərik.

12.3.1 Obyektin həddlərinə müraciət

Sinif tipinin obyektlərinin həddlərinə müraciət strukt tipi ilə eyni olduğundan burada onun təkrar izahını verməyəcəyik. d1 obyektinin en və uzunluq həddlərinə qiymətlər mənimsədən proqram tərtib edək:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:

	int en;
	int uzunluq;

};

int main()
{
	duzbucaqli d1;

	d1.en = 12;
	d1.uzunluq = 15;

	cout   << "duzbucaqlinin eni " << d1.en <<"\n"
	       << "duzbucaqlinin uzunluqu" << d1.uzunluq;

}
```

Nəticə:

```
duzbucaqlinin eni 12
duzbucaqlinin uzunluqu15
```

12.4 Sinfin Funksiya Həddləri

Sinif daxilində dəyişən həddləri ilə yanaşı funksiya həddlərini də tərtib edə bilərik. Sinfin funksiya həddləri bizim indiyə kimi tərtib etdiyimiz funksiyalardan heçnə ilə fərqlənmir. Sinfin funksiya həddlərinin elanın sinfin elanı daxilində verməliyik. Proqram kodunu isə həm sinfin elanı daxilində, həm də ondan ayrı tərtib edə bilərik. Aşağıda hər iki hal üçün nümunə göstərilir. Nümunə üçün isə gəlin duzbucali sinfinə sahe adlı public funksiya hədd əlavə edək. sahe funksiyası adından da göründüyü kimi düzbucaqlının sahəsini hesablayıb nəticə olaraq qaytarmalıdır. Kodu daxil edək, daha sonra izah hissəsini davam edərik.

```
class duzbucaqli {

public:

	int en;
	int uzunluq;

	int sahe() {

		return en*uzunluq;

	}

};
```

Bu halda fuksiya həddinin proqram kodu sinfin elanı daxilində verilib. Digər mümkün həll isə funksiyanın elanın sinfin daxilində verib, kod hissəsin sinifdən ayrı tərtib etməkdir. İkinci hal bu şəkildə olar:

```
class duzbucaqli {

public:

	int en;
	int uzunluq;

	int sahe();

};
```

Gördüyümüz kimi bu dəfə yalnız sahə funksiyanın elanın sinif daxilində verdik. Bundan sonra sahə funksiyasının kodun proqramın istənilən yerində, hətta digər mənbə kodunda tərtib edə bilərik. Adətən yeni sinif yaradarkən C++ proqramçıları sinfin elanın ayrı başlıq faylında, funksiya həddlərinin realizasiyasın isə ayrı mənbə C++ faylında yığırlar.

Sinfin funksiya həddinin kod hissəsi qeyd etdiyimiz kimi proqramın istənilən yerində tərtib edə bilərik(elandan sonra). Bunun üçün aşağıdakı qaydadan istifadə etməliyik. Əvvəlcə bütün funksiyalarda olduğu tipi göstərməliyik:

```
int
```

Daha sonra tərtib edəcəyimiz funksiyanın duzbucaqli sinfinin funksiya həddi olduğunu bildirmək üçün sinfin adını yazıb iki ardıcıl qoşanöqtə işarəsi qoyuruq

```
duzbucaqli::
```

Daha sonra isə adi bildiyimiz qaydada funksiyanı tərtib edirik:

```
sahe() {

		return en*uzunluq;
}
```

Bütün bunların hamısını birləşdirsək sinfin funksiya həddinin ikinci tərtib metodu alınar:

```
int duzbucaqli::sahe() {

		return en*uzunluq;
}
```

Biz hər iki həll metodundan istifadə edəcəyik.

12.4.1 Sinfin funksiya həddlərinə müraciət

Sinfin funksiya həddlərinə dəyişən həddlərinə olduğu kimi müraciət edə bilərik. Nümunə koda baxaq:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	int en;
	int uzunluq;
	int sahe();
};


int main()
{
	duzbucaqli d1;

	d1.en = 12;
	d1.uzunluq = 15;

	cout << "duzbucaqlinin eni " << d1.en << "\n"
		<< "duzbucaqlinin uzunluqu " << d1.uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << d1.sahe();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
```

```
}
```

Nəticə

```
duzbucaqlinin eni 12
duzbucaqlinin uzunluqu 15
duzbucaqlinin sahesi 180
```

Gördüyümüz kimi sahe funksiyasının kodunu main –dən sonra tərtib etmişik. Qeyd etdiyimiz kimi elanı sinif daxilində verdikdən sonra kodu istənilən yerdə tərtib edə bilərik.

Funksiya həddinin də necə tərtib olunması ilə tanış olduq. Nümunədən gördüyümüz kimi sahe funksiyası sinfin məlumat həddlərinə asanlıqla müraciət edə bildi. Bu zaman həddlərə müraciət edərkən onların adlarının əvvəlinə nöqtə qoyulmadı.

```
return en*uzunluq;
```

Sinfimizə yeni bir funksiya da əlavə edə bilərik, perimetr funksiyası. Sinfin tam kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	int en;
	int uzunluq;
	int sahe();
	int perimetr();
};


int main()
{
	duzbucaqli d1;

	d1.en = 12;
	d1.uzunluq = 15;

	cout << "duzbucaqlinin eni " << d1.en << "\n"
		<< "duzbucaqlinin uzunluqu " << d1.uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << d1.sahe() << "\n"
		<< "duzbucaqlinin perimetri " << d1.perimetr();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}
```

```
int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}
```

Nəticə

```
duzbucaqlinin eni 12
duzbucaqlinin uzunluqu 15
duzbucaqlinin sahesi 180
duzbucaqlinin perimetri 54
```

Sinfin sahə və perimetrini hesablayan funksiya həddlərini əlavə etdik. Lakin sinfin məlumatlarının çap olunması üçün əlavə funksiyada tərtib edə bilərik. Kod belə olar:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	int en;
	int uzunluq;
	int sahe();
	int perimetr();
	void cap_et();
};


int main()
{
	duzbucaqli d1;

	d1.en = 12;
	d1.uzunluq = 15;

	d1.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}

void duzbucaqli::cap_et() {

	cout << "duzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
}
```

Yeni yaradılmış cap_et funksiya həddini anlamaq oxucuya sərbəst həvalə olunur.

12.5 Private – Gizli həddlər

OYP texnologiyasına görə sinfin məlumat həddləri yalnız sinfin öz funksiyaları üçün əlçatan olmalıdır. Proqramın digər yerlərindən sinfin məlumat həddlərinə birbaşa müraciət olunması doğru sayılmır. Müraciət və dəyişiklik yalnız bunun üçün xüsusi nəzərdə tutulmuş funksiya həddləri vastəsilə həyata keçirilir.

Sinfin hər hansı məlumat və ya funksiya həddini proqramın digər hissələrindən gizlətmək üçün private tənzimləyicisindən istifadə olunur.

```
private:
```

Buna görə yuxarıda elan etdiyimiz duzbucaqli sinfini en və uzunluq həddlərini private tanzimləyici ilə elan etmək daha məqsədəuyğun olar.

```
class duzbucaqli {

public:
	int sahe();
	int perimetr();
	void cap_et();

private:
	int en;
	int uzunluq;
};
```

Lakin bu halda elan etdiyimiz obyektin en və uzunluq həddlərinə artıq əvvəlki qaydada main funksiyasından qiymətlər mənimsədə bilməyəcəyik:

```
	d1.en = 12;
	d1.uzunluq = 15;
```

Artıq bu həddlərə qiymətləri yalnız sinfin öz funksiyalarından mənimsətmək mümkündür. Bunun üçün terefler adlı yeni funksiya hədd tərtib edə bilərik. Nümunə məqsədilə gəlin onu tərtib edək və private həddlərə qiymətlər mənimsətməyin yolunu göstərək. Lakin əgər söhbət obyekt yaradılarkən onun məlumat həddlərinə başlanğıc

qiymət mənimsədilməsindən gedirsə, bu iş üçün kanstruktordan istifadə olunur. Kanstruktor barədə növbəti bölmədə danışacağıq.

Sinfin məlumat həddlərinə qiymət mənimsətmək üçün terefler funksiyasını sinfin daxilində aşağıdakı kimi elan edə bilərik:

```
	void terefler(int x, int y);
```

Realizasiyasını isə bu şəkildə:

```
void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}
```

terefler funksiyası sinfin öz funksiyası olduöuna görə sinfin həddlərinə müraciət edə bilər. Bütün ideya bundan ibarətdir. Tam proqram kodu və nümunə nəticə aşağıda verilib:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	int sahe();
	int perimetr();
	void cap_et();
	void terefler(int x, int y);

private:
	int en;
	int uzunluq;
};


int main()
{
	duzbucaqli d1;

	d1.terefler(42, 57);
	d1.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}

void duzbucaqli::cap_et() {
```

```
	cout << "duzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
}

void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}
```

Nəticə:

```
duzbucaqlinin eni 42
duzbucaqlinin uzunluqu 57
duzbucaqlinin sahesi 2394
duzbucaqlinin perimetri 198
```

Burada gördüyümüz kimi main funksiyasında obyekti elan etdikdən sonra onun həddlərinə qiymət mənimsətmək üçün terefler funksiya həddini aşağıdakı kimi çağırdıq:

```
	d1.terefler(42, 57);
```

12.6 Kanstruktor

Yuxarıdakı nümunə proqramda qeyd etdiyimiz kimi obyekti yaradarkən onun məlumat həddlərinə başlanğıc qiymətlər mənimsətmək üçün kanstruktordan istifadə olunur. Kanstruktor mahiyyət etibarilə sinfin funksiya həddidir, lakin digər funksiya həddlrindən bəzi xüsusiyyətlərinə və vəzifələrinə görə fərqlənir.

Birinci: kanstruktorun tipi olmur. Yəni kanstruktoru elan etdikdə tip olaraq heç nə yazmırıq, heç void də.

İkinci: kanstruktorun adı sinfin adı ilə eyni olmalıdır. Bu kompilyatora həmin funksiya həddinin məhs kanstruktor olmasını müəyyən etməyə imkan verir.

Üçüncü: kanstruktor obyekt yaradılarkən avomatik çağırılır, yəni digər funksiya həddləri kimi kanstruktoru da çağırmağa ehtiyac yoxdur. Onun məqsədi məhs obyekt yaradılarkən icra olunmaq və obyektin funksiya həddlərinə nəzərdə tutulan qiymətlər mənimsətməkdir.

Başlayaq duzbucaqli sinfinin kanstruktorunu qurmağa. Deyilənləri nəzərə aldıqda duzbucaqli sinfinin kanstruktorunu aşağıdakı kimi elan edə bilərik.

```
	duzbucaqli();
```

Kanstruktirun realizasiyasını isə aşağıdakı kimi tərtib edə bilərik:

```
duzbucaqli::duzbucaqli() {

	en = 1;
	uzunluq = 2;

}
```

Proqramın tam kodu aşağıdakı kimi olar:

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	duzbucaqli();
	int sahe();
	int perimetr();
	void cap_et();
	void terefler(int x, int y);

private:
	int en;
	int uzunluq;
};


int main()
{
	duzbucaqli d1;

	d1.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}

void duzbucaqli::cap_et() {

	cout << "duzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
```

```
}

void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}


duzbucaqli::duzbucaqli() {

	en = 1;
	uzunluq = 2;

}
```

Nəticə:

```
duzbucaqlinin eni 1
duzbucaqlinin uzunluqu 2
duzbucaqlinin sahesi 2
duzbucaqlinin perimetri 6
```

Burada biz kanstruktorda en və uzunluq həddlərinə susmaya görə 1 və 2 qiymətlərini vermişik. Gördüyümüz kimi kanstruktiru biz heç bir yerdə çağırmadıq. O, obyekt yaradılarkən avtomatik çağrılır. Kanstruktorla bağlı digər bir məsələ də ona parametr ötürməkdir. Kanstruktor da digər funksiyalar kimi parametr qəbul edə bilər və obyektin məlumat həddlərinə bu parametrləri başlanğıc qiymət olaraq mənimsədə bilər. Biz hal-hazırdakı tərtib etdiymiz kanstruktoru da dəyişə bilərik ki o parametr qəbul eləsin və bu parametrləri en və uzunluq həddlərinə mənimsətsin. Lakin C++ dili sinif daxilində bir neçə kanstruktor tərtib etməyə imkan verir. Yəni indiki kanstruktora dəyməyərək belə biz parametr qəbul edən əlavə yeni bir kanstruktor yarada bilərik. Yeni yaradacağımız kanstruktor iki parametr qəbul edəcək int tipli x və y. Bu parametrləri uyğun olaraq en və uzunluq həddlərinə mənimsədəcək.

Yeni yaradacağımız kanstruktorun elan və realizasiyası müvafiq olaraq aşağıdakı kimi olar:

```
duzbucaqli(int x, int y);


duzbucaqli::duzbucaqli(int x, int y) {

	en = x;
	uzunluq = y;

}
```

Tam kod isə aşağıdakı kimi olar:

```cpp
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	duzbucaqli();
	duzbucaqli(int x, int y);
	int sahe();
	int perimetr();
	void cap_et();
	void terefler(int x, int y);

private:
	int en;
	int uzunluq;
};


int main()
{
	duzbucaqli d1, d2(34, 55);

	d1.cap_et();
	d2.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}

void duzbucaqli::cap_et() {

	cout << "\n\nduzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
}

void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}


duzbucaqli::duzbucaqli() {

	en = 1;
	uzunluq = 2;

}
```

```
duzbucaqli::duzbucaqli(int x, int y) {

	en = x;
	uzunluq = y;

}
```

Nəticə:

```
duzbucaqlinin eni 1
duzbucaqlinin uzunluqu 2
duzbucaqlinin sahesi 2
duzbucaqlinin perimetri 6

duzbucaqlinin eni 34
duzbucaqlinin uzunluqu 55
duzbucaqlinin sahesi 1870
duzbucaqlinin perimetri 178
```

Maraqlı məsələ bu iki kanstruktorun çağırılması ilə bağlıdır. Kompilyator obyekt yaradılarkən hansı kanstruktorun çağırılacağın parametrlərə görə müəyyən edir. Biz duzbucaqli sinfindən d1 və d2 adlı iki obyekt yaradırıq.

```
	duzbucaqli d1, d2(34, 55);
```

Bu obyektlərin birincisini yaradarkən susmaya görə kanstruktor çağırılır. Lakin ikinci obyektin elanına diqqət yetirək. Onun adından sonra mötərizə daxilində parametrlərini vermişik. Bu qayda ilə biz kompilyatora çağırmaq istədiyimiz kanstruktor barədə məlumat vermiş oluruq və kampilyator da müvafiq kanstruktoru çağırıq.

12.7 Destruktor

Varislik bölməsinə keçmədən öncə siniflərlə bağlı bilməli olduğumuz daha bir məqam da var(Polimorfizm mövzusunu izah etməyi kitabın budəfəki buraxılılşında nəzərdə tutmamışıq). Bu da destruktordu. Destruktorun işi kanstruktorun gördüyü işin əksini görməkdir. Destruktor da kanstruktor kimi avtomatik çağrılır.

Destruktor obyekt, əgər dinamik obyektdirsə silinəndə çağırılır. Digər hallarda isə proqram icrasını başa çatdırdıqda çağrılır. Destruktorun məqsədi obyekt üçün ayrılmış resursları (əgər varsa) azad etməkdir.

Destruktor da kanstruktor kimi elan olunur, lakin adının əvvəlinə tilda işarəsi - ~ artırılır.

```
	~duzbucaqli();
```

Kodunu isə aşağıdakı kimi tərtib edə bilərik:

```
duzbucaqli::~duzbucaqli() {

	cout << "\n Obyekt yaddashdan silindi \n";

}
```

Destruktordan istifadə ilə tam koda və icra nəticəsinə baxaq.

```
#include <iostream>

using namespace std;

class duzbucaqli {

public:
	duzbucaqli();
	duzbucaqli(int x, int y);
	~duzbucaqli();
	int sahe();
	int perimetr();
	void cap_et();
	void terefler(int x, int y);

private:
	int en;
	int uzunluq;
};


int main()
{
	duzbucaqli d1, d2(34, 55);

	d1.cap_et();
	d2.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2*(en+uzunluq);
}

void duzbucaqli::cap_et() {

	cout << "\n\nduzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
}

void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}
```

```
duzbucaqli::duzbucaqli() {

	en = 1;
	uzunluq = 2;

}

duzbucaqli::duzbucaqli(int x, int y) {

	en = x;
	uzunluq = y;

}

duzbucaqli::~duzbucaqli() {

	cout << "\n Obyekt yaddashdan silindi \n";

}
```

Nəticə:

```
duzbucaqlinin eni 1
duzbucaqlinin uzunluqu 2
duzbucaqlinin sahesi 2
duzbucaqlinin perimetri 6

duzbucaqlinin eni 34
duzbucaqlinin uzunluqu 55
duzbucaqlinin sahesi 1870
duzbucaqlinin perimetri 178
 Obyekt yaddashdan silindi

 Obyekt yaddashdan silindi
```

12.8 Varislik

Varislik sinifdən sonra OYP-nin ən vacib ikinci prinsipidir. Varislik yeni sinfi yaratamaq üçün oxşar siniflərdən istifadə etməklə təmamilə başlanğıcdan yaradılmanın qarşısını alır. Misal üçün tutaq ki, biz proqramımızda **meyvə** sinfi təyin etmişik. **meyvə** sinfinin rəng, dad, qoxu v.s. kimi xassələrini özündə saxlayan həddlərini təyin etmişik. Əgər bir yeni **alma** sinfi təyin etmək istəyiriksə, bu zaman tamamilə başlanğıcdan yeni bir sinif yaratmağa ehtiyac yoxdur. Biz **alma** sinfini **meyvə** sinfinin varis sinfi kimi təyin etməklə rəng, qoxu v.s. bütün müyvələrə xas olan əlamətləri hazır əldə edə bilərik. Qalır sadəcə almaya spesifik olan xassələrin əlavə olunmasına. Misal üçün tərkibində dəmirin miqdarı v.s.

Bu zaman başlanğıc sinif adlanır əsas sinif, yeni yaradılan sinif isə adlanır varis sinif. Yəni bizim misalda əsas sinif olur – meyvə, varis sinif isə alma.

Hər hansı mövcud A sinfindən varsilik yolu ilə yeni B sinfi yaratmaq üçün aşağıdakı qaydadan istifadə olunur:

```
class B : public class A {


};
```

Yeni yaradılan sinif əcdad sinfin məlumat və funksiya həddlərinə sahib olur. Yəni əcdad sinifdə mövcud olan həddləri təkrar olaraq varis sinifdə elan etməyə ehtiyac yoxdur. Gəlin izahı konkret proqram nümunəsi üzərində verək.

Proqram nümunəsi üçün avtomobil ilə əlaqəli bir misala müraciət edəcəyik. Ümid eliyirəm ki, avtomobil ilə bağlı nümunə həm praktik, həm də maraqlı olar. İlk əvvəl bütün avtomobillər üçün oxşar xassələri özündə saxlayan **avto** adlı sinif yaradaq və bu sinifdən a1 adlı obyekt elan edək:

```
#include <iostream>
#include <string>

using namespace std;

class avto {

public:
	avto(string, int, int, int, int);
	void hereket_et();
	void dayan();
	void parametrleri_de();
protected:
	string nov;
	string model;
	int en;
	int uzunluq;
	int aqirliq;
	int max_suret;
};

int main()
{

	avto a1("her-hansi neq. vst.", 1000 , 1000, 1000, 100);

	a1.parametrleri_de();
	a1.hereket_et();
	a1.dayan();
}

avto::avto(string s, int x, int y, int z, int h) {

	model = s;
```

```
	en = x;
	uzunluq = y;
	aqirliq = z;
	max_suret = h;

}

void avto::hereket_et() {

	cout << "Men hereket edirem\n";

}

void avto::dayan() {

	cout << "Men dayaniram\n";

}

void avto::parametrleri_de() {

	cout << "\n\nNov: " << nov << "\t"
		<< "Model: " << model << "\n"
		<< "En: " << en << " sm\t"
		<< "Uzunluq: " << uzunluq << " sm\t"
		<< "Aqirliq: " << aqirliq << " kq\t"
		<< "Max_suret: " << max_suret << " km\\saat\n";
}
```

Nəticə:

```
Nov:    Model: her-hansi neq. vst.
En: 1000 sm     Uzunluq: 1000 sm        Aqirliq: 1000 kq        Max_suret: 100
km\saat
Men hereket edirem
Men dayaniram
```

Diqqət yetirsək avto sinfinin məlumat həddlərini `private` deyil `protected` elan etmişik. Bunun səbəbi odur ki, `private` kimi elan olunan həddləri gizli saxlanılır və hətta varis sinif də görə bilmir. `protected` həddləri yalnız varislər görə bilər. Buna görə varislik ilə ötürülməli olan həddləri adətən `protected` tənzimləyici ilə elan edirlər.

İndi isə varislik yolu ilə avto sinfindən public traktor adlı sinif yaradaq və traktor sinfindən tr adlı obyekt elan edək. Traktor sinfinin elanı aşağıdakı kimi olar:

```
class traktor : public avto {
public:
	traktor(string s, int x, int y, int z, int h);
	void yer_qaz();
};
```

Yeni yaratdığımız traktor sinfi iki yeni funksiya hədd əlavə etmişik. Bunlardan biri konstruktor, digəri yer_qaz() funksiya həddidir. Yeni kanstruktorda biz əcdad sinfin kanstruktorundan fərqli olaraq nov məlumat həddinə qiymət mənimsədirik:

```
traktor::traktor(string s, int x, int y, int z, int h) {

	model = s;
	en = x;
	uzunluq = y;
	aqirliq = z;
	max_suret = h;

	nov = "traktor";

}
```

Bu məlumat həddlərin hamısını traktor sinfi öz əcdadı olan avto sinfindən varislik yolu ilə əldə edir. Nümunədən görürük ki, nov həddindən başqa digər bütün həddlər əcdad sinfin kanstruktorunda da çağrılır. Ona görə əlavə kodlaşdımanın qarşısını almaq məqsədilə varis sinifin kanstruktorunu tərtib edərkən biz əcdad sinfin kanstruktorunu çağıra və mənimsədilmməsi gərəkən parametrləri ona arqument kimi ötürə bilərik. Varis sinfin funksiya həddlərindən əcdad sinfin funksiya həddlərini çağırmaq üçün əcdad sinfin adını yazıb ardıcıl iki qoşanöqtə qoyub çağırmaq istədiyimiz funksiya həddin adını yazırıq. Yeni qayda ilə traktor sinfinin kanstruktorunu aşağıdakı kimi çağıra bilərik:

```
traktor::traktor(string s, int x, int y, int z, int h) {

	avto::avto(s, x, y, z, h);

	nov = "traktor";

}
```

Sadə halda isə funksiya parametrləri mötərizəsindən sonra qoşanöqtə qoyub əcdad sinfin kanstruktorunu çağırmaqdır. Bizim üçün bir qədər öyrəşmədiyimiz sintaksis olsa da, maraqlı görünüşü ilə diqqəti cəlb edir və C++ kodlarını daha oxunaqlı edir.

C++ dilinin qurucuları bir qədər də irəli gedərək əcdad sinfin funksiya həddini çağırmağı daha da sadələşdirmişlər. Bunun üçün vasi sinfin funksiya həddində funksiya parametrləri mötərizəsindən sonra qoşanöqtə qoyub əcdad sinfin kanstruktorunu çağıra bilərik, aşağıdakı kimi:

```
traktor::traktor(string s, int x, int y, int z, int h) : avto(s, x, y, z, h) {

	nov = "traktor";

}
```

Bizim üçün bir qədər öyrəşmədiyimiz sintaksis olsa da, maraqlı görünüşü ilə diqqəti cəlb edir və C++ kodlarını daha oxunaqlı edir.

traktor sinfinin yer_qaz() funksiya həddini isə aşağıdakı kimi tərtib edəcəyik:

```
void traktor::yer_qaz() {

	cout << "Men yer qaziram\n";

}
```

avto sinfindən varislik yolu ilə traktor sinfi yaradan proqram kodu aşağıdakı kimi olar:

```
#include <iostream>
#include <string>

using namespace std;

class avto {

public:
	avto(string, int, int, int, int);
	void hereket_et();
	void dayan();
	void parametrleri_de();
protected:
	string nov;
	string model;
	int en;
	int uzunluq;
	int aqirliq;
	int max_suret;
};

class traktor : public avto {
public:
	traktor(string s, int x, int y, int z, int h);
	void yer_qaz();
};

int main()
{

	traktor tr("Aqrolux", 1969, 3315, 2160, 30);

	tr.parametrleri_de();
	tr.hereket_et();
	tr.dayan();
	tr.yer_qaz();

}

avto::avto(string s, int x, int y, int z, int h) {

	model = s;
	en = x;
	uzunluq = y;
	aqirliq = z;
	max_suret = h;
```

```
}

void avto::hereket_et() {

	cout << "Men hereket edirem\n";

}

void avto::dayan() {

	cout << "Men dayaniram\n";

}

void avto::parametrleri_de() {

	cout << "\n\nNov: " << nov << "\t"
		<< "Model: " << model << "\n"
		<< "En: " << en << " sm\t"
		<< "Uzunluq: " << uzunluq << " sm\t"
		<< "Aqirliq: " << aqirliq << " kq\t"
		<< "Max_suret: " << max_suret << " km\\saat\n";
}


traktor::traktor(string s, int x, int y, int z, int h) :avto(s, x, y, z, h) {

	nov = "traktor";

}

void traktor::yer_qaz() {

	cout << "Men yer qaziram\n";

}
```

Nəticə:

```
Nov: traktor    Model: Aqrolux
En: 1969 sm     Uzunluq: 3315 sm        Aqirliq: 2160 kq        Max_suret: 30
km\saat
Men hereket edirem
Men dayaniram
Men yer qaziram
```

Aşağıdakı kodda isə avto sinfindən varislik yolu ilə traktor sinfi ilə yanaşı kamaz sinfi də yaradılır:

```
#include <iostream>
#include <string>

using namespace std;

class avto {

public:
```

```
	avto(string, int, int, int, int);
	void hereket_et();
	void dayan();
	void parametrleri_de();
protected:
	string nov;
	string model;
	int en;
	int uzunluq;
	int aqirliq;
	int max_suret;
};

class traktor : public avto {
public:
	traktor(string s, int x, int y, int z, int h);
	void yer_qaz();
};

class kamaz : public avto {
public:
	kamaz(string s, int x, int y, int z, int h);
	void yuk_dashi();
};

int main()
{

	traktor tr("Aqrolux", 1969, 3315, 2160, 30);
	kamaz km("Iveco", 2205, 5257, 7834, 90);

	tr.parametrleri_de();
	tr.hereket_et();
	tr.dayan();
	tr.yer_qaz();

	km.parametrleri_de();
	km.hereket_et();
	km.dayan();
	km.yuk_dashi();

}

avto::avto(string s, int x, int y, int z, int h) {

	model = s;
	en = x;
	uzunluq = y;
	aqirliq = z;
	max_suret = h;

}

void avto::hereket_et() {

	cout << "Men hereket edirem\n";

}

void avto::dayan() {
```

```
	cout << "Men dayaniram\n";

}

void avto::parametrleri_de() {

	cout << "\n\nNov: " << nov << "\t"
		<< "Model: " << model << "\n"
		<< "En: " << en << " sm\t"
		<< "Uzunluq: " << uzunluq << " sm\t"
		<< "Aqirliq: " << aqirliq << " kq\t"
		<< "Max_suret: " << max_suret << " km\\saat\n";
}


traktor::traktor(string s, int x, int y, int z, int h) :avto(s, x, y, z, h) {

	nov = "traktor";

}

void traktor::yer_qaz() {

	cout << "Men yer qaziram\n";

}

kamaz::kamaz(string s, int x, int y, int z, int h) :avto(s, x, y, z, h) {

	nov = "kamaz";

}

void kamaz::yuk_dashi() {

	cout << "Men yuk dashiyiram\n";

}
```

Nəticə:

```
Nov: traktor    Model: Aqrolux
En: 1969 sm     Uzunluq: 3315 sm        Aqirliq: 2160 kq        Max_suret: 30
km\saat
Men hereket edirem
Men dayaniram
Men yer qaziram


Nov: kamaz      Model: Iveco
En: 2205 sm     Uzunluq: 5257 sm        Aqirliq: 7834 kq        Max_suret: 90
km\saat
Men hereket edirem
Men dayaniram
Men yuk dashiyiram
```

Çalışmalar.

Çalışma 1. Tam tipli en, uzunluq və hündürlük dəyişən həddlərindən ibarət Paralelpiped adlı sinif tərtib edin. Paralelpipedin yan səthinin sahəsini hesablamaq üçün Sahe(), həcmini hesablamaq üçün Hecm() funksya həddəlrini tərtib edin. Paralelpiped sinfindən parp adlı obyekt yaradın. Parp obyektinin en, uzunluq və hundurluk həddlərinə istifadəçinin daxil etdiyi qiymətləri mənimsədin. Parp obyektinin Sahe və Hecm funksiya həddlərini çağıraraq onun sahəsini və həcmini çap edin.

Çalışma 2. Cerge adlı sinif yaradın. Sinif dəyişən hədd olaraq özündə tam 10 elementdən ibarət x adlı cərgə saxlasın.Cerge sinfinin x cərgəsinin elemenlərini daxil etmək, çap etmək, artan və azalan sırada düzmək üçün müvafiq funksiya həddlərini tərtib edin. Cərgə sinfindən crg adlı obyekt yaradın. Crg obyektinin müvafiq funksiya həddlərindən istifadə edərək onun x cərgə həddinin elemenlərinə qiymətlər mənimsədin, onları artan və azalan sırada düzərək çap edin.

§13 Direktivlər.

C++ dili proqramlarda kompilyator üçün nəzərdə tutulmuş əmrlər və ya göstərişlərdən istifadə etmək üçün **direktiv** adlanan imkan təqdim edir. Adətən bu cür direktivlər hansısa faylı proqram koduna əlavə etmək, proqram kodunun müəyyən hissəsini kompilyasiya zamanı nəzərə almamaq, hər-hansı ifadəni digər bir ifadə ilə əvəzləmək v.s. kimi məqsədlər üçün istifadə olunur.

Direktivlər (bunlara həm də **preprosessor** direktivləri deyilir) əmr, təyin və şərti direktivlər olaraq növlərə bölünür. Bir qayda olaraq C++ dilində bütün direktivlərin əvvəlinə həştəq işarəsi – **#** artırılır. Direktivlər adətən bir sətri əhatə edir və sonlarına nöqtəvergül işarəsi qoyulmur.

13.1 Əmr direktivi

C++ dilində bir əmr ditektivi mövcuddur. Bu digər mənbə fayllarını proqrama əlavə etmək üçün istifadə olunan **`include`** direktividir. include derektivinin sintaksisi aşağıdakı kimidir:

```
#include <əlavə_ediməli_fayl>
```

və ya

```
#include "əlavə_ediməli_fay"
```

Əvvəlcə bütün direktivlərdə olduğu kimi **#** simvolunu yazırıq, daha sonra direktivi, baxdığımız halda include. Include sözündən sonra isə ya böyükdür-kiçikdir mötərizələri arasında, ya da cütdırnaq arasında proqrama əlavə etmək istədiyimiz faylın adını yazırıq. Birinci üsul adətən sistem başlıq fayllarını, ikinci üsul isə proqramçının özünün tərtib etdiyi faylları proqrama əlavə etmək üçün istifadə olunur. Birinci üsul ilə faylı proqrama əlavə etdikdə həmin fayl kompilyatorun susmaya görə təyin olunmuş başlıq faylları yerləşən qovluqda, ikinci üsulda isə proqram faylının özünün yerləşdiyi qovluqda axtarılır.

include direktivi ilə proqrama əlavə oluna fayllar başlıq faylları adlanır (header files).

Misal üçün biz öz proqramlarımızda həmişə `iostream` və ya `string` başlıq fayllarını **include** direktivi ilə proqrama əlavə edirdik. Bunlar standart fayllardır. Lakin biz özümüz də bu cür başlıq fayllar yaradıb include direktivi ilə proqramımıza əlavə edə bilərik. Adətən başlıq faylları kod tərtibi üçün deyil, hansısa dəyişənlərin elanı, struct və ya sinif tiplərin təyini, funksiyaları elanlarının yerləşdirilməsi v.s. üçün istifadə olunur.

Yeni bir fayl yaradaq və bu fayla **test.h** adını verək(başlıq fayllarının sonu adətən .h ilə qurtarır). test.h faylında int tipindən olan x dəyişəni elan edək:

```
int x;
```

Yadda saxlayaq və proqramımızın əsas koduna include direktivi ilə yeni yaratdığımız test.h faylını əlavə edək.

```
#include <iostream>
#include <string>
#include "test.h"

int main()
{



}
```

include direktivi ilə test.h faylını proqrama əlavə etdikdən sonra onda elan olunan dəyişən və tiplərdən istifadə edə bilərik.

```
#include <iostream>
#include <string>
#include "test.h"

int main()
{

	//x deyisheni test.h faylinda elan olunub
	x = 9;

}
```

Bu cür tərtib əlbəttə düzdür, sadəcə normal layihələrdə 10-larla başlıq faylından istifadə oluna bilər, hansılar ki öz növbəsində digər başlıq fayllarını əlavə edə bilərlər(Bəli, başlıq fayllarının özləri də include direktivi ilə digər başlıq fayllarını əlavə edə bilərlər). Bu cür çoxşaxəli və qarşılıqlı əlavəolunmlar zamanı isə rekursiv olaraq biri – digərini sonsuz əlavəetməyə nəzarət etmək praktik cəhətdən çox əlverişsizdir. Əgər A başlıq faylı B-ni, B-də öz növbəsində A-nı əlavə edirsə, onda onlar bir-birini sonsuz proqram koduna dayanmadan əlavə edəcəklər. Bunun qarşısını almaq üçün şərti direktivlərdən istifadə olunur. Şərti direktivlərin izahı ilə növbəti bölmədə tanış

olacayıq. Hələlik isə şərti direktivlərdən istifadə etməklə test.h başlıq faylının nə cür düzgün tətrtib olunmasını göstərək.

```
#ifndef TEST_H
#define TEST_H

int x;

#endif
```

Başlıq faylları şərt və təyin direktivlərindən istifadə etməklə bu şəkildə tərtib olunduqları zaman, hətta qarşılıqlı əlavəolunma zamanı proqram koduna rekursiv olaraq sonsuz əlavə olunmalarının qarşısı alınmış olur. İndi isə siniflərə aid veridyimiz proqram kodu nümunəsini başlıq faylından istifadə etməklə tərtib olunmuş nümunəsinə baxaq, harada ki, sinfin elanı ayrıca başlıq başlıq fayla yerləşdirilir və proqramın əsas koduna əlavə olunur. Nəzərə alaq ki, sinfin yalnız elanını başlıq fayla yerləşdiririk, funksiya həddlərinin kodu isə elə proqramın əsas kodunda qalır.

Aşağıdakı kimi **duzb.h** adlı başlıq faylı yaradaq və duzbucaqli sinfini elanını bu faylda yadda saxlayaq:

```
#ifndef DUZB_H
#define DUZB_H

class duzbucaqli {

public:
	duzbucaqli();
	duzbucaqli(int x, int y);
	int sahe();
	int perimetr();
	void cap_et();
	void terefler(int x, int y);

private:
	int en;
	int uzunluq;
};

#endif
```

duzb.h faylını proqrama əlavə edək və onun funksiya həddlərini tərtib edək:

```
#include <iostream>
#include "duzb.h"

using namespace std;

int main()
{
	duzbucaqli d1, d2(34, 55);
```

```
	d1.cap_et();
	d2.cap_et();

}

int duzbucaqli::sahe() {

	return en*uzunluq;
}

int duzbucaqli::perimetr() {

	return 2 * (en + uzunluq);
}

void duzbucaqli::cap_et() {

	cout << "\n\nduzbucaqlinin eni " << en << "\n"
		<< "duzbucaqlinin uzunluqu " << uzunluq << "\n"
		<< "duzbucaqlinin sahesi " << sahe() << "\n"
		<< "duzbucaqlinin perimetri " << perimetr();
}

void duzbucaqli::terefler(int x, int y) {

	en = x;
	uzunluq = y;
}


duzbucaqli::duzbucaqli() {

	en = 1;
	uzunluq = 2;

}

duzbucaqli::duzbucaqli(int x, int y) {

	en = x;
	uzunluq = y;

}
```

Əslində normal layihələr zamanı sinfin funksiya həddlərini ayrı bir CPP faylında yığırlar və ümumi proqram kodu ilə birlikdə kompilyasiya edirlər. Lakin bu üsul hər-bir proqramlaşdırma mühiti tərəfində fərqli realizə olunduğuna görə ümumi şəkildə nümunə göstərmək çətindir.

13.2 Təyin direktivləri

C++ dilində **define** və **undef** adlı təyin direktivləri mövcuddur. Birinci direktiv hər-hansı ifadəni təyin edir və ona qiymət mənimsədir. İkinsi isə əkasinə olaraq həmin ifadəni ləğv edir.

13.2.1 define direktivi

define direktivinin sintaksisi aşağıdakı kimidir:

```
#define Ifadə Qiymət
```

Bu direktivdən sonra proqram kodunda hər-bir İfadə sözünə rast gəlinən yerdə `Ifadə` sözü `Qiymət` ilə əvəz olunacaq. Misal üçün nümunəyə baxaq:

```
#include <iostream>

using namespace std;

#define SAY 10

int main()
{

	int i, x[SAY];

	for (i = 0; i < SAY; ++i)
		cin >> x[i];

}
```

Bundan sonra proqramın hər yerində SAY sözü `10` ilə əvəz olunacaq. Yəni yuxarıdakı proqramda `int` tipli 10 elemendən ibarət x cərgəsi elan olaunur və onun elementlərinə istifadəçi tərəfindən qiymətlər mənimsədilir.

Əlbəttə deyə bilərsiniz ki bunun mənası nədir, elə proqramı adi bildiyimiz qaydada tərtib etsəydik nə əksik olardı, aşağıdakı kimi:

```
#include <iostream>

using namespace std;

int main()
{

	int i, x[10];

	for (i = 0; i < 10; ++i)
		cin >> x[i];

}
```

Haqlısınız, amma bir məsələ var. Təsəvvür edin, tərtib etdiyiniz layihə 10 000 sətir koddan ibarətdir və bu kodda məhs x cərgəsinin ölçüsündən dəfələrlə istifadə

etmisiniz. Əgər bu ölçünü dəyişmək tələb olunsa onda proqramı sətir-sətir oxuyub hər yerdə x-in ölçüsünü (dövr operatorlarında) dəyişməlisiniz. Find/Replace –dən istifadə edə bilməzsiniz, çünki dəyişmək istədiniz ədəd başqa məqsədlər üçün də istifadə oluna bilər. Lakin direktivdən istifadə etsəniz, yalnız direktivin qiymətini dəyişməklə tələb olunan dəyişikliyi əldə etmiş olursuz, daha tez və daha dəqiq.

13.2.2 undef direktivi

undefine direktivi define direktivinin əksinə əvvəlcədən define ilə hər-hansı ifadəyə verilmiş qiyməti ləğv edir. Aşağıdakı kimi:

```
#undef Ifadə
```

Bu zaman define ilə -yə verilmiş qiymət ləğv olur. Əgər undef ilə ifadənin qiymətini ləğv etdikdən sonra yenidən ona müraciət etmək istəsək onda kompilyasiya xətası alarıq, aşağıdakı kimi:

```
#include <iostream>

using namespace std;

#define SAY 10

int main()
{

	int i, x[SAY];

	for (i = 0; i < SAY; ++i)
		cin >> x[i];

#undef SAY

	int y[SAY];
}
```

Əgər bu proqramı icra eləsək kompilyasiya səhvi alacayıq. Misal üçün Bu proqram MS Visual Studio –da kompilyasiya olunduqda aşağıdakı xəta alınır:

```
source.cpp(17): error C2065: 'SAY': undeclared identifier
```

13.3 Şərt direktivləri

Şərt direktivlərinin məqsədi proqram kodunun bu və ya digər hissəsini kompilyasiyadan kənar etməkdir. Proqramlaşdırmada buna çox vaxt ehtiyac yaranır. Tutaq ki, ilk öncə hansısa formada test məqsədilə alqoritmi tərtib edirsən. Daha sonra

məqbul həll variantını tapdıqdan sonra artıq keçici tərtib etdiyin test alqoritmi artıq gərəkli olmur. Belə olan halda həmin kodu tamamilə proqramın mənbə faylından silməmək üçün (gələcəkdə yenidən lazım ola biləcəyini nəzərə alaraq) şərti direktivlər vastəsilə kompilyasiyadan kənarlaşdırırlar. Fiziki olaraq kod proqramın mətn faylında olduğu kimi qalır, sadəcə kompilyasiya zamanı kompilyator həmin hissələri nəzərə almır.

C++ dilində aşağıdakı şərt direktivləri var:

#if

#ifdef

#ifndef

#else

#elif

#endif

Bu direktivlərdən if, ifdef və ifndef direktivləri özlərindən sonra bir ifadənin olmasını tələb edirlər, qalanları isə yuxarıda göstərilən şəkildə istifadə olunurlar. Misala baxaq:

```
#if ifadə

kod

#endif
```

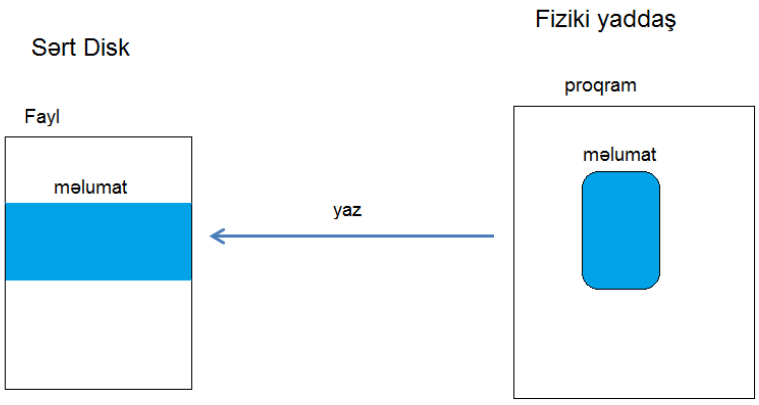
Hər bir if ilə başlayan direktiv mütləq bir #endif direktivi ilə bağlanmalıdır. Arada istənilən qədər bir-birinin içinə yerləşdirilmiş if/endif qoymaq olar. Yuxarıdakı misalda göstərilən kod if derektivinin qarşısındakı ifadə doğru qiymət aldığı zaman kompilyator tərəfindən nəzərə alınacaq.

ifdef və ifndef direktivləri isə define direktivi tərəfindən təyin olunmuş ifadələri arqument kimi qəbul edir.

§14 Fayllar.

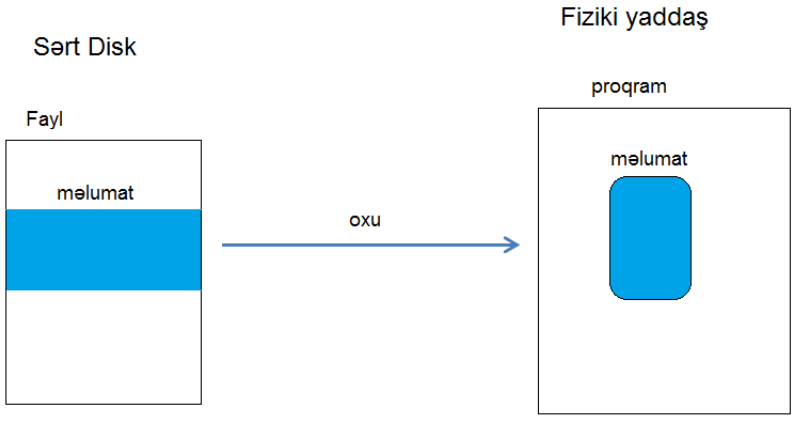
14.1 Fayllar

Biz proqramlarda hansısa məlumatı yadda saxlamaq üçün dəyişənlərdən istifadə edirik. Adətən bizə simvol, sətir və ya ədəd kimi sadə məluamtları yadda saxlamaq tələb olunur. Nisbətən irihəcmli və mürəkkəb struktura malik olan məlumatları yadda saxlamaq üçün isə cərgələrdən, obyektlərdən istifadə edirik. Bütün bunlar öz işlərini mükəmməl görür lakin, bir çatışmamazlıqdan başqa. Dəyişənlərdə yadda saxladığımız məlumat proqram icrasını bitirdikdən sonra yaddaşdan silinir. Əgər bizə bu məlumatları yadda saxlamaq tələb olunursa onda biz onu **"fayla yazmalıyıq".**



Fayla yazılşım məlumat daimi yadddaş qurğusunda yadda saxlandığından (Sərt disk, usb, CD/DVD disklər v.s.) onlar nəinki proqram bitdikdən sonra, hətta kompüter söndürüldükdə belə yaddaşdan silinmir və uzun müddət yaddaşda qalır (Yalnız yaddaş qurğusu fiziki olaraq zədələndiyi və ya öz istismar müddətini başa vurduğu müddətdən sonra fayl silinmiş olur, əlbəttə ki faylın səhvən və ya bilərəkdən əl ilə silinməsini də nəzərə almaqla).

Faylda saxlanılan məlumatı proqram daha sonra istədiyi vaxt oradan oxuya bilər.



C++ dilində məlumatlar fayla iki üsulla yazıla(oxuna) bilər:

Xam formada**,** yəni necə var elə, oluğu kimi, məlumat üzərində heç bir dəyişilik aparmadan

və

Emal olunmuş formada. Bu zaman məlumatlar olduğu kimi deyil, bir qədər dəyişikliyə uğrayaraq fayla yazılır(oxunur).

Bu yazıb-oxunma üsulları adətən müvafiq olaraq ikili və mətn üsulu kimi də adlandırılır. Bir çox hallarda mətn rejimindən sətir tipli məlumatları fayla yazıb-oxuyan zaman istifadə edirlər.

14.2 Stream - Axınlar

C++ dili fayl üzərinə oxuma-yazma əməliyyatı aparmaq üçün **stream** –lərdən istifadə edir. Stream sözü Azərbaycan dilində **axın** mənasını verir. Bu məlumatların bir mənbədən digər mənbəyə köçürülməsi üçün istifadə olunan xüsusi yaradılmış sinifdir. Bu sinfin bəzi obyektləri ilə biz artıq tanışıq: **cin** və **cout** obyektləri. Bu iki axın obyekti müvafiq olaraq məlumatı klaviaturadan proqrama və proqramdan ekrana ötürmək üçün istifadə olunurlar. Fayllarla işləmək üçün isə ayrıca tərtib olunmuş axın sinifləri mövcuddur. Bu siniflər müvafiq olaraq **ifstream**(fayldan oxumaq üçün), **ofstream**(fayla yazmaq üçün) və **fstream**(eyni zamanda həm oxuyub, həm də yazmaq üçün) siniflərdir.

ifstream	fayldan oxumaq üçün
ofstream	fayla yazmaq üçün

fstream	eyni zamanda həm oxuyub, həm də yazmaq üçün

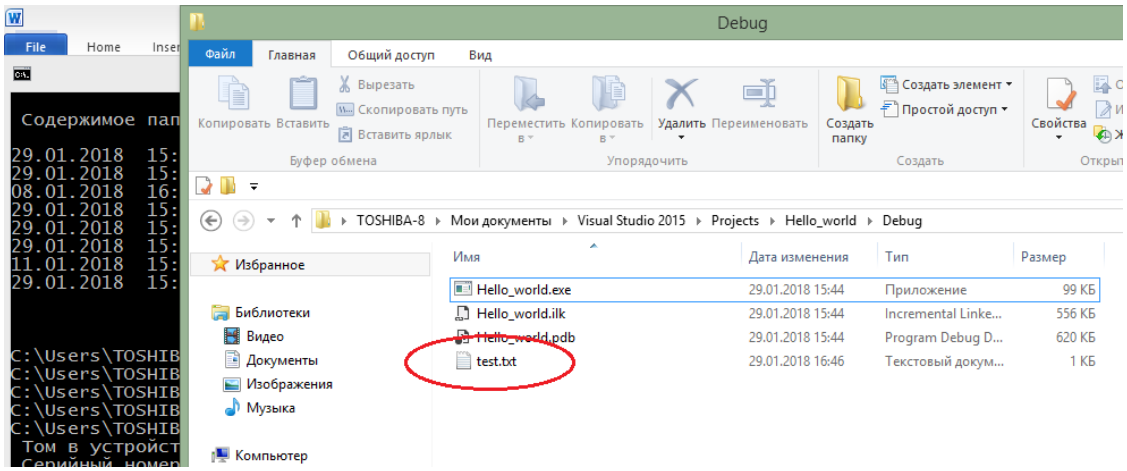
Bu siniflərdən istifadə edə bilmək üçün **fstream** başlıq faylını proqrama əlavə etmək lazımdır. Əvvəlcə bu axın siniflərindən obyekt elan etməliyik və bu obyekti **açmalıyıq**. Açılma zaman biz verilmiş axın obyektini fayl ilə əlaqələndiririk. Daha sonra isə axına əlavə et **<<** və axından götür **>>** operatorlarını köməyi ilə məlumatları fayla yazıb-oxuya bilərik.

Faylı açmaq üçün bu open funksiyasından istifadə edirik. Sadəcə open yazırıq və açmaq istədiyimiz faylın adını cütdırnaq işarəsi arasıda veririk.

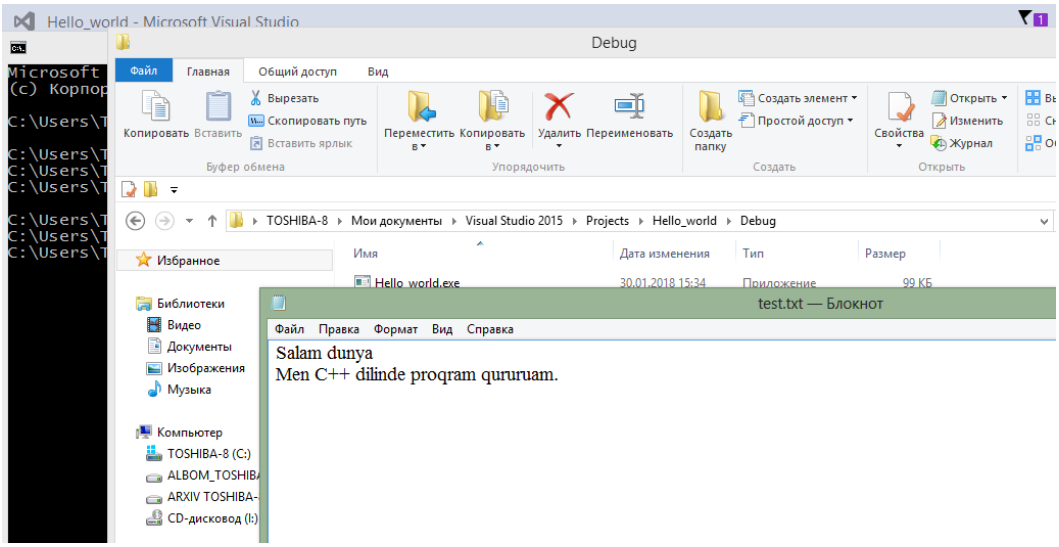
Aşağıdakı proqram hazırki qovluqda (proqram icra olunan qovluqda) `"test.txt"` adlı fayl yaradır və ona `"Salam dunya \n"` sətrini yerləşdirir.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {

	ofstream f_yaz;

	f_yaz.open("test.txt");

	f_yaz << "Salam dunya \n";
	f_yaz << "Men C++ dilinde proqram qururuam. \n";

	f_yaz.close();
}
```

Bu proqramı icra etsək hazırki qovluqda (proqram icra olunan qovluqda) `"test.txt"` adlı fayl yaranar.



Əgər bu test.txt faylını hər-hansı mətn redaktoru məsələn Notepad ilə açsaq orada Salam dunya sətrinin yazıldığını görərik:



İzahı: Əvvəlcə **ofstream** axın sinfindən **f_yaz** adlı obyekt elan edirik. Elan etdiyimiz obyektə adi bir dəyişən kimi istənilən ad verə bilərik.

```
ofstream f_yaz;
```

Daha sonra file obyektinin open funksiyası ilə test.txt faylını açırıq. Qeyd edək ki bu zaman əgər fayl mövcud deyilsə o open funksiyası tərəfindən yaradılmış olacaq.

```
f_yaz.open("test.txt");
```

Faylı açdıqdan sonra artıq axına əlavə et operatorunun **<<** köməyi ilə məlumatları test.txt faylına yaza bilərik:

```
f_yaz << "Salam dunya \n";
f_yaz << "Men C++ dilinde proqram qururuam. \n";
```

Fayl üzərində əməliyyatları bitirdikdən sonra **close**() funksiyası ilə onu bağlayırıq. Faylı bağlayan zaman axın obyektinin fayl ilə əlaqəsi kəsilir və proqram fayl üçün ayrılmış olan resursları azad edir.

14.3 Fayldan məlumatın oxunması

Fayldan məlumatın oxunması yazmaya oxşardır. Bu zaman ofstream deyil ifstream sinfindən obyekt elan etməliyik. Gəlin yuxarıdakı proqram ilə tərtib etdiyimiz test.txt faylını yazmaq üçün açaq və onu oxuyub məlumatları çap edək. Kod aşağıdakı kimi olar:

```
#include <iostream>
```

```
#include <fstream>
#include <string>

using namespace std;

int main() {

	ifstream f_oxu;

	f_oxu.open("test.txt");

	while (f_oxu) {

		string str;
		f_oxu >> str;
		cout << str << "\n";
	}

	f_oxu.close();

}
```

Nəticə:

```
Salam
dunya
Men
C++
dilinde
proqram
qururuam.
```

İzahı: Əvvəlcə proqram kodunu izah edək daha sonra nəticəni təhlil edərik. Fayldan məlumatı oxuduğumuza görə **ifstream** axınından istifadə edirik. **Ifstream** sinfindən **f_oxu** adlı axın obyekti elan edirik və open funksiyası ilə test.txt faylını oxumaq üçün açırıq.

```
	ifstream f_oxu;

	f_oxu.open("test.txt");
```

Daha sonra ühile operatoru ilə faylda olan məlumatları oxuyub çap edirik. Faylı açarkən fayl göstəricisi faylın əvvəlində dayanır. Hər dəfə axından götür **>>** operatoru ilə növbəti məlumatı oxuyan zaman fayl göstəricisi faylın sonuna doğru sürüşür. Faylın sonuna çatdıqda isə f_oxu obyekti 0 qiyməti qaytarır. Bu zaman dövr bitmiş olur.

```
while (f_oxu) {
```

```
        string str;
        f_oxu >> str;
        cout << str << "\n";
}
```

Fayldan oxumaq öçün aşağıdakı koddan istifadə edirik:

```
f_oxu >> str;
```

Bu zaman f_oxu axın obyekti fayldan növbəti oxuyaraq str string dəyişəninə yerləşdirir. Daha sonra f_oxu obyektinin str sətrinə yerləşdirdiyi məlumatı cout ilə ekranda çap edirik:

```
cout << str << "\n";
```

Qeyd edək ki, std string tipindən istifadə etdiyimizə görə string faylını proqrama əlavə etməliyik. string tipi ilə sətirlər bölməsində tanış olmuşuq(bax $9). İndi isə nəticəni təhlil edək. Gördüyümüz kimi faylan oxuduğumuz məlumat ilə fayla yazdığımız məlumat bir qədər fərqlənir:

Yazmışdıq:

```
        "Salam dunya \n"
        "Men C++ dilinde proqram qururuam. \n"
```

Oxuduq:

```
Salam
dunya
Men
C++
dilinde
proqram
qururuam.
```

Məsələ ondadır ki, oxuyan zaman axından götür **>>** operatoru fayldan oxunan məlumatı **format** edir. Yəni məsafə simvollarını nəzərə alaraq hissələrə ayırır. Əgər biz fayldan sətirləri bütöv halda oxumaq istəyiriksə onda paraqraf 9-dan bizə məlum olan **getline** funksiyasından istifa etməliyik, aşağıdakı kimi:

```
        getline(f_oxu, str);
```

Tam kod və nəticə bu şəkildə olacaq:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
```

```
int main() {

	ifstream f_oxu;

	f_oxu.open("test.txt");

	while (f_oxu) {

		string str;
		getline(f_oxu, str);
		cout << str << "\n";
	}

	f_oxu.close();

}
```

Nəticə:

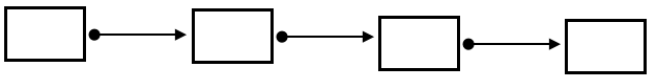
```
Salam dunya
Men C++ dilinde proqram qururuam.
```

III HİSSƏ

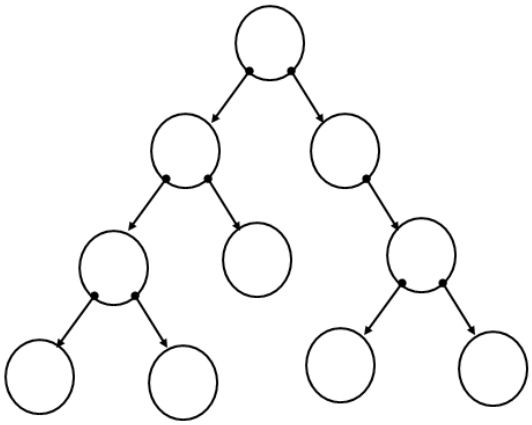
DINAMIK VERILƏNLƏR STRUKTURU

§15 Siyahılar.

Siyahı



Ağac



15.1 Siyahılar

Biz indiyə kimi çoxölçülü məlumatlarla işləyərkən cərgələrdən istifadə edirdik. Cərgələr bizə istənilən tipdən istənilən ölçüdə verilənlər adıcıllığı yaratmağa imkan verir. Lakin cərgələrin çox ciddi çatışmayan cəhətləri var. Biz cərgədəki elementlərin sayın yalnız cərgəni elan edəndə verə bilərik və proqramın icrası boyu bu ölçü dəyişilməzdir. Bundan əlavə cərgənin hansısa iki elementi arasına başqa bir element yerləşdirə bilmərik. Yada gərək həmin elementdən axıra kimi bütün elementləri bir vahid sürüşdürək, bu halda isə axırıncı elementi itiririk. Cərgələrin digət mənfi cəhəti odur ki, böyük ölçüdə elan etdiyimiz cərgənin istifadə olunmayan elementləri daima yaddaşda boş yer tutub yaddaşı israf edir.

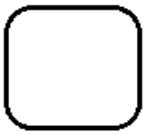
Dinamik verilənlər strukturu isə cırgıdın fərqli olaraq proqramın icrası boyu öz ölçüsünü dinamik olaraq istənilən cür artırıb – azalda bilər. İstənilən qədər özünə yeni element əlavə edə və silə bilər. Əvvəlcədən onda nə qədər element olacağıcını bilməyə ehtiyac yoxdur. Çatışmamazlığı isə odur ki, cərgələrdəki kimi istənilən elementdə birbaşa indeksinə görə müraciət eləmək olmur, gərək lap əvvəldən bütün elementəri bir-bir yoxlayıb tələb olunan elementin üzərinə keçəsən.

Bu paraqrafda C++ dilində dinamik verilənlər strukturlarının ən çox istifadə olunan nümunələrindən biri olan ***əlaqəli siyahılar*** ilə tanış olacayıq. Dinamik verilənlər strukturunun digər növbəti məşhur sinfi olan ***ağaclarla*** isə gələn paraqrafda tanış olacayıq. Qeyd edim ki, bu və növbəti mövzu ümumən proqramlaşdırmada çətinliyi artırılmış mövzular hesab olunur. Bu mövzuları ilk dəfəyə qavramaq çətin məsələdər. Ona görə təkrar – təkrar məsələyə yanaşmağı və mümkün qədər kağız qələmdən çox istifadə etməyi məsləhət görürəm, əlbəttə kompüter praktikasını da unutmamaq şərtilə.

15.2 Siyahı yaratmaq

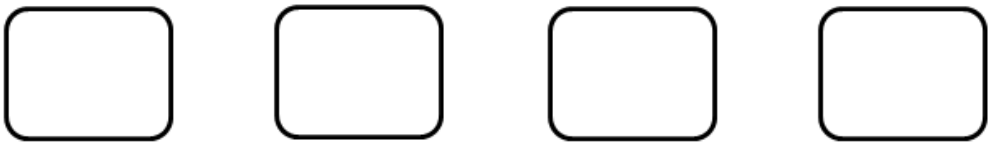
Siyahı yaratmaq üçün siyahı elementlərindən başlamaq lazımdır.

Siyahı elementi

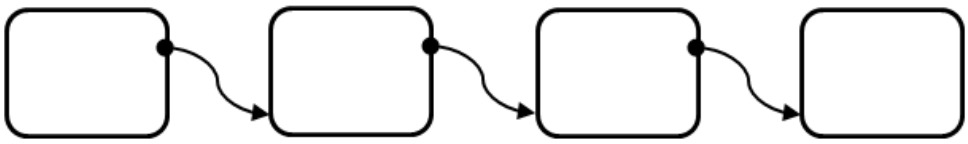


Siyahı boş da ola bilər, onda yalnız bir element də ola bilər və birdən çox sayda element ola bilər. Siyahı boş olanda və ya onda yalnız bir element olanda siyahının əsl mahiyyəti elə də hiss olunmur. Maraqlı hissə məhs siyahıya birdən çox sayda element əlavə etdikdə olur.

İki və daha artıq element



Bu elementlərdən siyahı yaratmaq üçün onları bir-biri ilə ardıcıl birləşdirməliyik.



Bunun üçün birləşdirici qoldan istifadə olunur.

Bütün bunlar siyahının dizayn cizgiləridi, indi isə siyahının C++ dilində nə cür realizə olunmasını örgənək.

Siayhı yaratmaq üçün əvvəlcə siyahının elementinin təsvir etməliyik. Bunun üçün struct tip yaradırıq

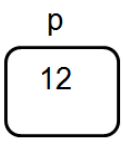
```
struct el{

};
```

Hələki yaratmaq istədiyimiz sadəcə siyahının elementIdir, siyahı filan yoxdur. Gördüyümüz kimi el adlı içi boş bir struct tipi təyin etmişik. Siayhının elementinin tipinə istənilən ad ver bilərik. Siyahı hansısa verilənləri saxlamaq üçün istifadə olunur. Bu verilənlər isə elementlərdə yerləşdirilir. Siyahının elementlərində istənilən tipli məlumat yerləşdirə bilərik, sadə standart tipli dəyişənlərdən tutmuş, mürəkkəb strukturlu obyeklərə kimi. Lakin sadəlik xətrinə gəlin biz tərtib edəcəyimiz siyahımızda məlumat olaraq yalnız tam tipdən olan bir dəyişən yadda saxlayaq. Bunu elan etdiyimiz el tipinin daxilinə yerləşdirməliyik.

```
struct el{

  int x;

};
```

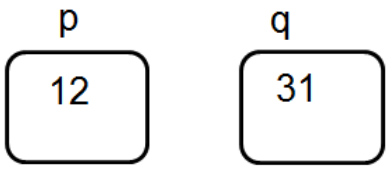
Misal üçün gəlin struct tipindən p adlı obyekt yaradaq və onun x həddinə 12 qiyməti mənimsədək.

```
el p;

p.x = 12;
```



Başqa bir q obyekti elan edib, x həddinə 31 qiyməti mənimsədək.

```
el q;

q.x = 12;
```



Məlumat hissəsini tamamladıqdan sonra əlaqə hissəsinə keçək. Bu əlaqəni qurmaq üçün, yəni siyahının iki elementini bir-biri ilə birləşdirmək üçün aşağıdakı açar metoddan istifadə olunur.

Qayda:

Siyahının elementini təsvir edən struct tipin daxilində həmin tipin özündən olan göstərici elan edirik.

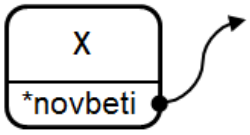
Aşağıdakı kimi:

```
struct el{
  int x;
 struct el *novbeti;
};
```

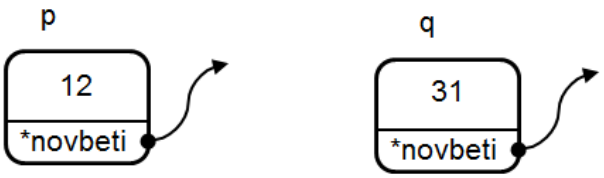
Yeni əlavə etdiyimiz həddi ayrıca yazaq

```
struct el *novbeti
```

Burada biz struct el tipinin daxilində həmin tipin özündən olan yəni struct el tipindən olan növbəti adlı göstərici hədd elan edirik. Əgər bu hədd göstərici olmasaydı, onda kompilyator səhv mesajı verərdi, çünki struct tipin daxilində həmin tipin özündən olan obyekt elan etmək olmaz, başqa sözlə rekursiv strukt tip elan edə bilmərik. Amma göstərici ilə problem yoxdur. Nəticədə struct el tipinə həmin tipdən olan obyektlərin ünvanın yadda saxlaya biləcək hədd əlavə etmiş olduq.



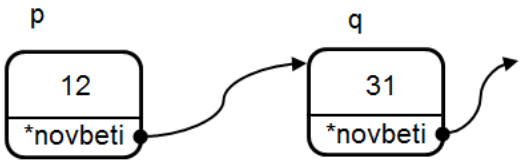
p və q obyektləri aşağıdakı kimi olar



Artıq bu iki elementi birləşdirməyə tam hazırıq. Bunun üçün p obyektinin novbeti həddini q obyektinin yaddaşdakı ünvanına mənimsətməliyik.

```
p.novbeti = &q;
```

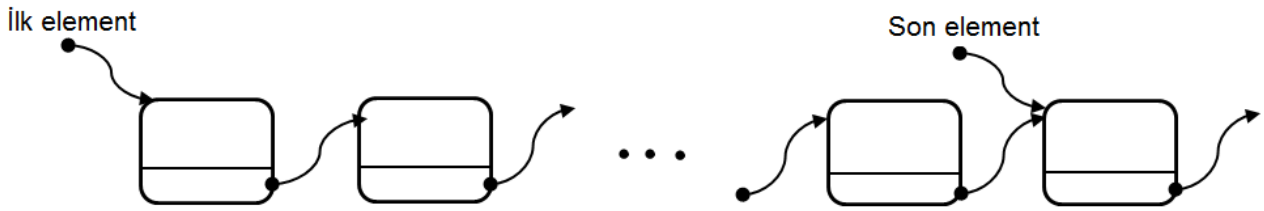
Nəticədə p obyektinin novbeti heddi q-ye istinad etmiş olar



Bu qayda ilə istənilən sayda obyekti bir-biri ilə əlaqələndirib siyahını istədiyimiz qədər böyüdə bilərik.

15.2.1 Siyahının ilk və son elementləri

Siyahıda istənilən sayda element ola bilər. Cərgədən fərqli olaraq siyahının elementlərinə indeksinə görə müraciət etmək mümkün deyil. Əksinə tələb olunan elementi tapmaq üçün ilk elementdən başlayaraq son elementə kimi bütün elementləri nəzərdən keçirməliyik. Bunun üçün hər-bir siyahının ilk və son elementlərinə istinadın bilinməsi zəruridir.



Bəzən siyahının son elementinə istinadı yadda saxlamırlar. Belə halda siyahının sonunu müəyyənləşdirmək üçün sonuncu elementin növbəti həddinin NULL olması şərtindən istifadə edirlər.

Ümumiyyətlə siyahının son elenetinin növbəti həddi həmişə NULL –a mənimsədilməlidir. Yalnız **qoşaistiqamətli** siyahılarda bu fərqli ola bilər.

15.2.2 Siyahının təyin olunması

Yuxarıda biz siyahının elementini təyin etdik. Bu bölmədə isə siyahının özünün nə cür yaradılmasını örgənəcəyik. Əlbəttə ki bunun üçün yuxarıda təyin etdiyimiz siyahı elementindən istifadə etməliyik. Yuxarıda təyin etdiyimiz siyahı elementinin tipinə bir daha nəzər salaq:

```
struct el{
  int x;
 struct el *novbeti;
};
```

Elementin tipini təyin etdikdən sonra siyahının tipini aşağıdakı kimi tətin edə bilərik:

```
struct siyahi {
    struct el *ilk;
    struct el *son;
};
```

siyahı adlı yeni struct tip yaratdıq hansı ki özündə struct el tipindən iki göstərici saxlayır, ilk və son. Artıq təxmin etdiyiniz kimi bu iki hədd müvafiq olaraq siyahının ilk və son elementlərinə istinad edəcək.

siyahının tipini təyin etdikdən sonra onda obyekt elan edə bilərk. struct siyahi tipindən syh adlı obyekt elan edək:

```
struct siyahi syh;
```

Bu elan ilə biz artıq siyahını yaratmış olduq. Sadəcə siyahımı hələ boşdur. Buna görə siyahının ilk və son həddlərini NULL –a mənimsədək.

```
syh.ilk = NULL;
```

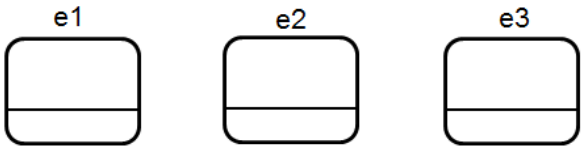
```
syh.son = NULL;
```

Artıq siyahımıza ilk elementi daxil edə bilərk.

15.2.3 Siyahıya elementlərin əlavə olunması

Tutaq ki, struct el tipindən el1, el2 və el3 adlı 3 obyekt elan etmişik

```
struct el el1, el2, el3;
```

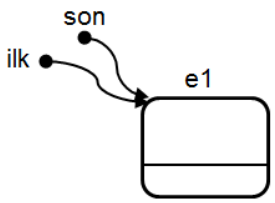


Sadəlik xətrinə ilk çalışmamızda dinamik obyektlərdən istifadə etməyəcəyik. Yəni siyahıya əlavə edəcəyimiz hər 3 element gördüyümüz kimi ***əvvəlcədən*** elan olunub, daha dinamik olaraq proqramın icrası zamanı yaradılmır. Elementlərin dinamik olaraq yaradılıb siyahıya əlavə oluması ilə bir qədər irəlidə məşğul olacayıq.

İlk olaraq siyahıya e1 elementini əlavə edək. Yada salaq ki, **struct siyahi** tipindən olan **syh** adlı siyahını bir az öncə yuxarıda yaratmışdıq və onun ilk və son həddlərinə NULL qiyməti mənimsətdik. Siyahıya ilk elementi əlavə etmək asandır, beləki bu halda siyahının həm ilk həm də son elementi eyni olur. Buna görə e1 –i siyahıya əlavə etmək üçün ilk və son göstəricilərini e1-in ünvanına mənimsətməyimiz kifayətdir.

```
syh->ilk = &e1;
```

```
syh->son = &e1;
```



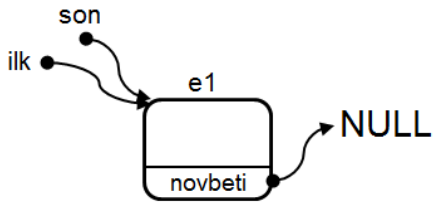
Siayhı dinamik struktur olduğuna görə biz ona element əlavə edərkən siyahının boş olub olmadığını yoxlamadan bilə bilmərik. Yuxarıdakı kodda e1 –i siyahıya əlavə edərkən biz heç bir yoxlama aparmadıq. Əgər siyahıda digər elementlər olsaydı yuxarıdakı kimi e1-i əlavə etdikdə həmin elementlər ilə rabitəni itirərdik. Buna görə qaydalara uyğun olaraq siyahıya yeni element əlavə edərkən onun boş olub-olmadığını yoxlamalıyıq. Bu isə çox sadədir. Siyahının ilk və son elementlərini NULL ilə müqaisə

edərək onun boş olduğunu müəyyən edə bilərik. e1 –in siyahıya düzgün şəkildə əlavə olunması aşağıdakı kimi olmalıdır.

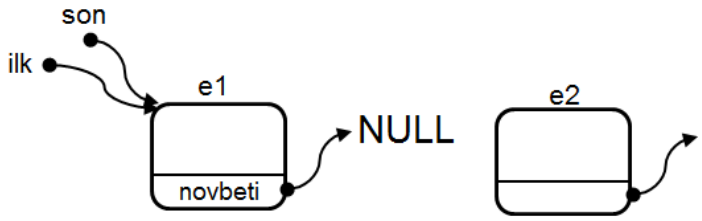
```
if (syh->ilk == NULL && syh->son == NULL){

    syh->ilk = &e1;
    syh->son = &e1;

}
```

Siyahıya element əlavə etdikdən sonra yeni əlavə olunan elementin növbəti həddini NULL-a mənimsətməliyik.

e1->novbeti = NULL;



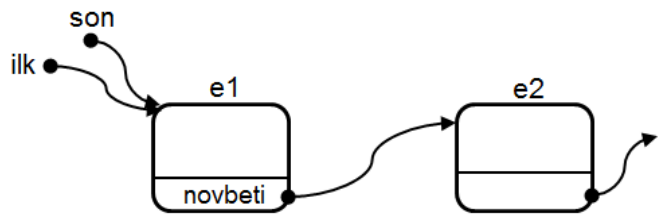
İlk elementi əlavə etdik, indi ikinci elementi əlavə edək.



İki element arasında əlaqənin nə cür yaradılmasından yuxarıda bəhs etmişik. Bir elementin növbeti heddinin digər elementin ünvanına mənimsədirik.

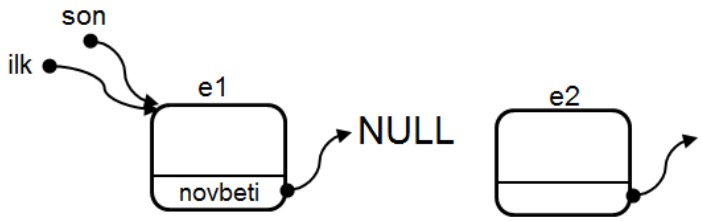
```
e1->novbeti = &e2;
```

Nəticədə e1 e2-yə birləşər



Bu əlbəttə izah üçün yarıyar, amma elementlər dinamik olaraq yaradılıb əlavə edilərkən biz e1, e2 kimi adlardan istifadə edə bilməyəcəyik. Bunun üçün istinad olaraq həmişə siyahının son elementindən istifadə etməliyik. Nəzərə alarıq ki son element həmişə sonuncu elementə istinad edir. Əlbəttə biz elementi yalnız siyahının sonuna

deyil, əvvəlinə və hətta istənilən iki elementi arasına da yerləşdirə bilərik. Lakin bu məsələlərlə bir qədər təcrübə əldə etdikdən sonra məşğul olacağıq. Hələlik isə elementləri siyahının sonuna artıracağıq. Beləliklə e2 elementini siyahının sonuna əlavə etmək üçün



düzgün qayda aşağıdakı kimi olar:

```
syh->son->novbeti  = &e2;
```

Burada biz iki dəfə ardıcıl göstəricinin həddə müraciət operatorundan , yəni -> operatorundan istifadə etdik. syh->son e1-ə istinad etdiyindən ondan sonra da -> işarəsi qoyub e1-in istənilən həddinə müraciət edə bilərik. Bu şəkildə

```
syh->son->
```

Bizə e1-in **novbəti** həddi lazım olduğundan biz aşağıdakı kimi yazdıq

```
syh->son->novbeti
```

və həmin həddi e2-nin ünvanına mənimsətdik

```
syh->son->novbeti  = &e2;
```

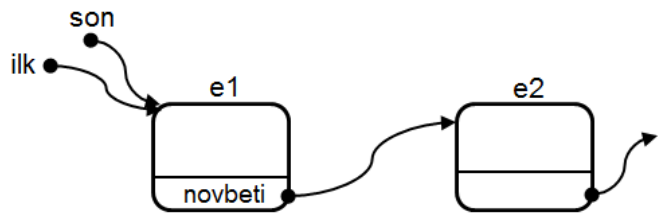
Bu kod elə

```
e1->novbeti = &e2;
```

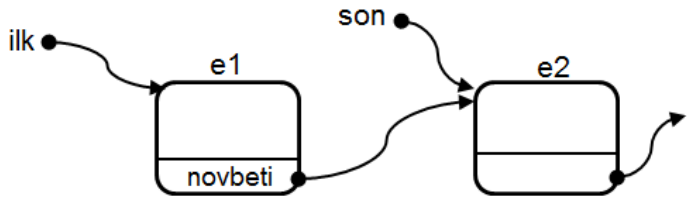
koduna ekvivalentdir. Düzgün kodu bir daha təkrar yazaq

```
syh->son->novbeti  = &e2;
```

Nəticədə siyahının son elementinin novbeti həddi e2-nin ünvanına mənimsənər.



Yuxarıda biz bu nəticəni artıq almışdıq, lakin indi daha düzgün qayda ilə elədik. Lakin bununla hər şey başa çatmır. Siyahıya yeni element əlavə olunduğuna görə biz siyahının son həddinin yeni əlavə olunan elementin ünvanına mənimsətməliyik. Unutmayaq ki siyahının son həddi həmişə siyahnın sonuncu elementinə istinad etməlidir.

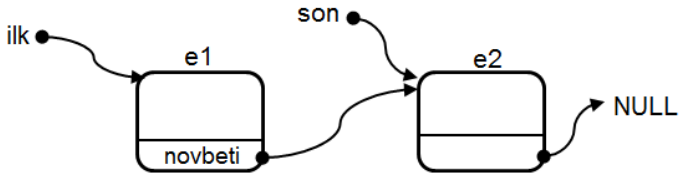


Bunun üçün isə aşağıdakı kod kifayətdir

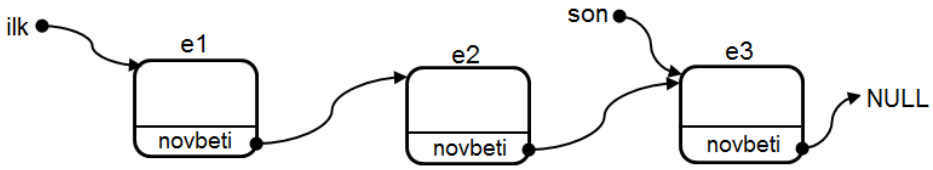
```
syh->son   = &e2;
```

Son olaraq isə siyahının sonuncu elementinin novbeti həddini NULL-a mənimsədirik.

```
syh->son->novbeti   = NULL;
```



Eyni qayda ilə e3 –də siyahımıza əlavə edə bilərik



Müvafiq proqram kodu aşağıdakı kimi olar

```
syh->son->novbeti   = &e3;

syh->son   = &e3;

syh->son->novbeti   = NULL;
```

Gördüyümüz kimi 3-cü elementin əlavə olunması kodu ikinci elementin əlavə olunması kimidir. Bütün digər elementlər də siyahıya bu qayda ilə əlavə olunur, yalnız ilk element fərqli şəkildə əlavə olunurdu. Buna görə də hər dəfə siyahıya element əlavə etdikdə onun boş olub-olmadığını yoxlayırıq, əgər boşdursa onda yuxarıda göstərilən qayda ilə birinci element əlavə olunur, əks halda indi tanış olduğumuz qayda ilə bir

sonrakı element əlavə olunur. Bütün bu deyilənləri kod şəklində aşağıdakı kimi təsvir edə bilərik.

15.2.4 Siyahıya elementin əlavə olunması

Tutaq ki hər-hansı elem_x elementini siyahıya əlavə etmək istəyirik, kod belə olar

```
//siyahinin bosh olmasini yoxlayiriq

if (syh->ilk == NULL && syh->son == NULL){

    //siyahi boshdur
    syh->ilk = &elem_x  ;
    syh->son = &elem_x  ;
    syh->son->novbeti  = NULL;
}
else
{
    //siyahida elementler var
    syh->son->novbeti  = &elem_x  ;
    syh->son  = &elem_x  ;
    syh->son->novbeti  = NULL;
}
```

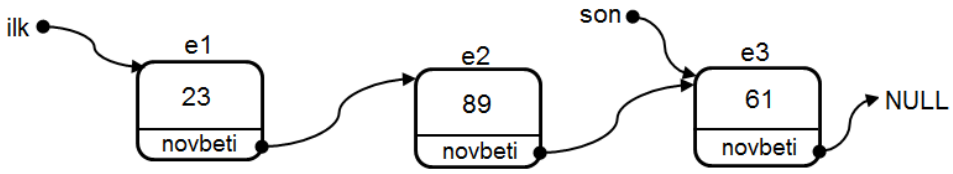
Əslində siyahıya elementin əlavə olunması tez-tez təkrarlanan əməliyyat olduğundan yuxarıdakı kodu proqram mətnində təkrarlamamaq üçün onu funksiya daxilinə yerləşdirək. Belə ki yeni elave_et adlı funksiya tərtib edək və hər-dəfə siyahıya yeni element əlavə etmək üçün bu funksiyanı çağıraq. elave_et funksiyası aşağıdakı kimi olar

```
void elave_et (struct siyahi *syh, struct el *eml){

    //siyahinin bosh olmasini yoxlayiriq
    if (syh->ilk == NULL && syh->son == NULL){

        //siyahi boshdur
        syh->ilk = &eml;
        syh->son = &eml;
        syh->son->novbeti  = NULL;
    }
    else
    {
        //siyahida elementler var
        syh->son->novbeti  = &eml;
        syh->son  = &eml;
        syh->son->novbeti  = NULL;
    }
}
```

Tərtib etdiyimiz elave_et funksiyasından istifadə etməklə siyahı yaradan və ona 3 element əlavə edən proqram kodu tərtib edə edək.

```
#include <iostream>

using namespace std;

struct el{
  int x;
 struct el *novbeti;
};

struct siyahi {
    struct el *ilk;
    struct el *son;
};
void elave_et (struct siyahi *syh, struct el *eml){

    //siyahinin bosh olmasini yoxlayiriq
    if (syh->ilk == NULL && syh->son == NULL){

        //siyahi boshdur
        syh->ilk = &eml;
        syh->son = &eml;
        syh->son->novbeti  = NULL;
    }
    else
    {
        //siyahida elementler var
        syh->son->novbeti  = &eml;
        syh->son  = &eml;
        syh->son->novbeti  = NULL;
    }
}

int main(){

    struct siyahi syh;

    syh.ilk = NULL;

    syh.son = NULL;

    struct el el1, el2, el3;

    //elementlerin melumat heddine muxtelif qiymetler menimsedek
    e1.x = 23;
    e2.x = 89;
    e3.x = 61;

    //elementleri siyahiya elave edek
    elave_et(syh, &e1);
    elave_et(syh, &e2);
    elave_et(syh, &e3);

}
```

Nəticədə aşağıdakı kimi siyahı alarıq.



Əgər bu proqramı kompilyasiya və icra eləsək heç bir nəticə hiss etməyəcəyik. Ən azından yaratdığımız siyahının elementlərini çap etsək bir qədər maraqlı olar. İndi gəlin bu məsələ ilə məşğul olaq.

15.3 Siyahının elementlərinə müraciət

Siyahıda nə qədər elementin olduğunu bilmirik, əlimizdə olan sadəcə siyahının ilk və son elementlərinə olan istinaddır. Biz isə hər bir elementə müraciət edib onun məlumat həddini çap etməliyik. Yuxarıdakı proqramda biz elementlərin məlumat həddlərinə heçnə mənimsətmədik. İndi tərtib edəcəyimiz kodda isə məlumat həddlərinə müxtəlif qiymətlər mənimsədərik.

Siyahının bütün elementlərini nəzərdən keçirməyin qaydası aşağıdakı kimidir:

Siyahının elementi tipindən hər-hansı göstərici elan ediri

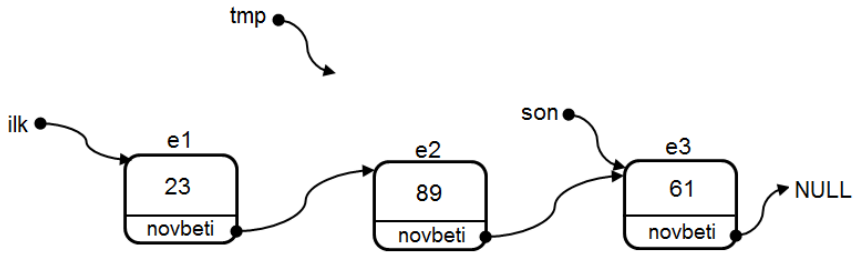
Bu göstəricini siyahının ilk həddinə mənimsədirik

Daha sonra həmin göstəricini ***bir-bir elementlərin üzərindən sürüşdürməklə*** sonuncu elementlə qədər davam edirik.

Gəlin yuxarıda tərtib etdiymiz siyahı üzərində bu dediklərimizi izah edək.

Əvvəlcə siyahının elementi tipində tmp adlı göstərici elan edək

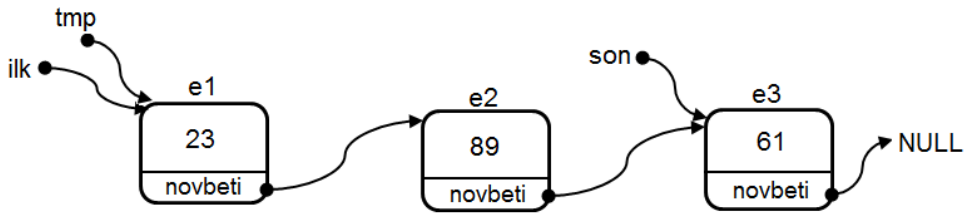
```
struct el  *tmp;
```



tmp-ni siyahının ilk elementinə mənimsədək

```
tmp = syh.ilk ;
```

Nəticədə tmp siyahının ilk elementinə istinad edər



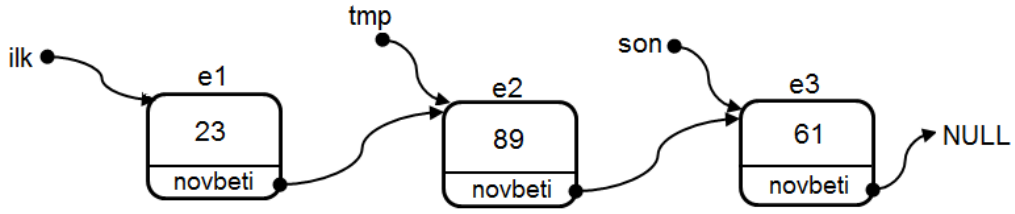
İndi tmp göstəricisindən istifadə etməklə ilk elementin x həddini çap edə bilərik

```
cout << tmp->x;
```

Bəs yuxarıda qeyd etdiyimiz kimi növbəti elementin üzərinə nə cür sürüşəcəyik? Bunun üçün elemetin novbeti heddindən istifadə edə bilərik. Siyahını tərtib edərkən (siyahının mahiyyəti bundan ibarətdir) hər elementin novbəti həddi sonrakı gələn elementə mənimsənir. Onda aşağıdakı qayda ilə tmp –dən ondan sonra gələn elementin üzərinə sürüşə bilərik

```
tmp = tmp->next;
```

Nəticədə tmp ikinci elementin üzərinə sürüşmüş olar



Əgər yenə eyni qayda ilə tmp-in x həddini çap etsək bu dəfə ikinci elementin x həddi çap olunar.

```
cout << tmp->x;
```

Bu şəkildə hər dəfə növbəti elemetin üzərinə sürüşməklə onun məlumat həddini çap edirik, ta ki sonuncu elementə qədər. Bəs harada dayanmalı olduğumuzu nə cür müəyyən edək? Bunun üçün şərti dövr daxilində tmp-ni NULL ilə müqaisə edə bilərik.

```
while( tmp != NULL ){

    cout << tmp->x;
    tmp = tmp->next;
}
```

Beləliklə yuxarıda yaratdğımız siyahının bütün elementlərini çap edən kod aşağıdakı kimi olar

```
struct el  *tmp;
tmp = syh.ilk ;

while( tmp != NULL ){

    cout << tmp->x;
    tmp = tmp->next;
}
```

Yenə səliqəli proqramlaşdırma ənənəsinə sadiq qalaraq siyahının elementlərini çap edən kod parçasını bir funksiya daxilinə yerləşdirək və hər dəfə siyahını çap etmək lazım gələndə bu funksiyadan istifadə edək. Funksiyamız aşağıdakı kimi olar

```
void   cap_et(struct siyahi *syh){

    struct el  *tmp;

    tmp = syh.ilk ;
    cout << "Siyahinin elementleri: \n";

    while( tmp != NULL ){
        cout << tmp->x;
        tmp = tmp->next;
    }

}
```

Bu funksiyamızı yuxarıda tərtib etdiyimiz siyahı yaradan proqramımıza əlavə etsək, 3 elementdən ibarət siyahı yaradan və onun elementlərini çap edən proqram kodu aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

struct el{
  int x;
 struct el *novbeti;
};

struct siyahi {
    struct el *ilk;
    struct el *son;
};
```

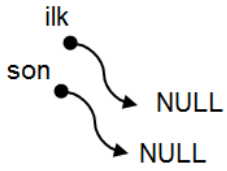
```
void elave_et (struct siyahi *syh, struct el *eml){

    //siyahinin bosh olmasini yoxlayiriq
    if (syh->ilk == NULL && syh->son == NULL){

        //siyahi boshdur
        syh->ilk = eml;
        syh->son = eml;
        syh->son->novbeti  = NULL;
    }
    else
    {
        //siyahida elementler var
        syh->son->novbeti  = eml;
        syh->son  = eml;
        syh->son->novbeti  = NULL;
    }
}

void   cap_et(struct siyahi *syh){

    struct el  *tmp;

    tmp = syh.ilk ;
    cout << "Siyahinin elementleri: \n";

    while( tmp != NULL ){
        cout << tmp->x;
        tmp = tmp->next;
    }
}

int main(){

    struct siyahi syh;

    syh.ilk = NULL;

    syh.son = NULL;

    struct el el1, el2, el3;

    //elementlerin melumat heddine muxtelif qiymetler menimsedek
    e1.x = 23;
    e2.x = 89;
    e3.x = 61;

    //elementleri siyahiya elave edek
    elave_et(&syh, &e1);
    elave_et(&syh, &e2);
    elave_et(&syh, &e3);
    cap_et(&syh);
}
```

15.4 Elementlərin dinamik əlavə olunması

Yuxarıdakı izahda sadəlik xətrinə siyahıya əvvəlcədən mövcud olan elementləri əlavə etdik cəmi 3 dənə. Əgər siyahıya 4-cü elementi əlavə etmək istəsəydik bunun üçün hazır elementimiz yox idi. Siyahıya yeni element əlavə etmək üçün dinamik yaradılmış obyektlərdən istifadə etsək onda istənilən qədər element əlavə edə bilərik. Bu bölmədə bunun nə cür yerinə yetirilməsini örgənəcəyik.

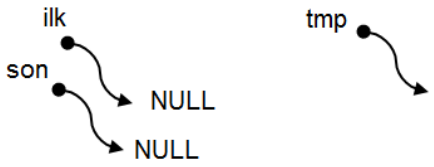
Tutaq ki siyahı elan etmişik və onun ilk və son həddlərini NULL –a mənimsətmişik

```
struct siyahi syh;

syh.ilk = NULL;
syh.son = NULL;
```



Siyahının elementi tipindən tmp adlı hər-hansı göstərici elan edək.

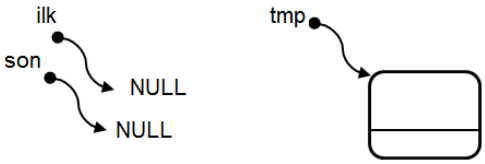
```
struct el  *tmp;
```



Daha sonra dinamik olaraq siyahıya əlavə etmək istədiyimiz ilk obyektimizi yaradaq

```
tmp  = new struct el;
```

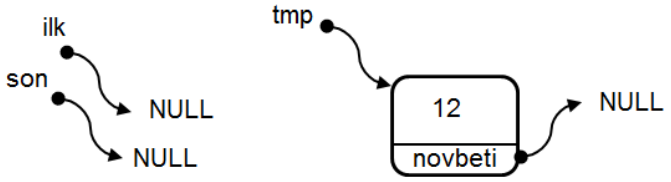
Ümid eliyirəm bu kodun izahı göstəricilər mövzusundan sizlərə aydındır. Bu zaman struct el tipindən yeni bir obyekt yaradılacaq və onun ünvanı tmp –yə mənimsədiləcək.



Yeni yaratdığımız elementin x həddinə hər-hansı qiymət, novbeti heddinə isə NULL mənimsədək.

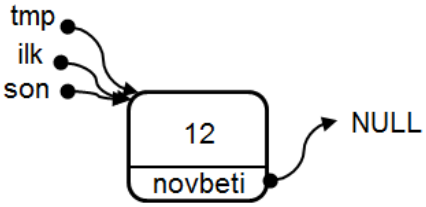
```
tmp->x = 12;
```

```
tmp->novbeti = NULL;
```



Indi yuxarıda izah etdyimiz qaydaya uyğun yeni yaratdığımız obyekti siyahıya əlavə edə bilərik

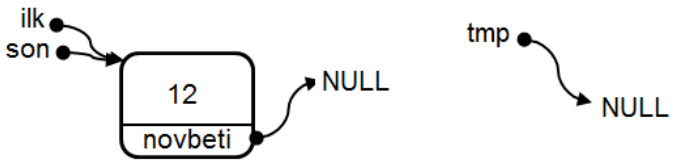
```
syh->ilk = tmp;
syh->son = tmp;
```



Yaratdığımız elementi siyahıya əlavə etdik. tmp –ni digər elementlərin yaradılması üçün istifadə edəcəyik, lakin hal-hazırda tmp siyahının elementinə istinad edir. Bu bir qədər düzgün sayılmır. Səhvən tmp-ni silə bilərik v.s. tmp öz işini hal-hazırda qurtarmışdır, ona görə onun siyahı ilə əlaqəsini kəsmək lazımdır.

```
tmp = NULL;
```

Nəticədə tmp-nin siyahı ilə əlaqəsi kəsilər.

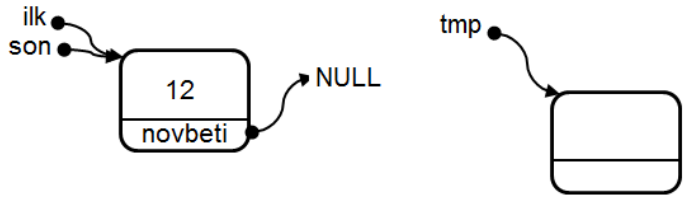


Əgər diqqət edirisinizsə biz siyahıya yeni element əlavə etdikdə onun boş olduğunu yoxlamadıq. Düzgün qayda aşağıdakı kimi olar.

```
if (syh->ilk == NULL && syh->son == NULL){

    syh->ilk = tmp;
    syh->son = tmp;
    syh->son->novbeti  = NULL;
}
```

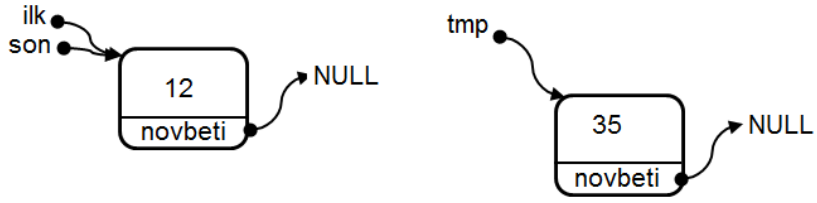
Eyni qayda ilə siyahıya ikinci element əlavə olunur. Dinamik olaraq yeni bir element yaradırıq(istədiyimiz qədər yarada bilərik).

tmp = new struct el;



yeni yaratdığımız elementin məlumat həddinə hər-hansı qiymət, novbeti həddinə NULL mənimsədirik

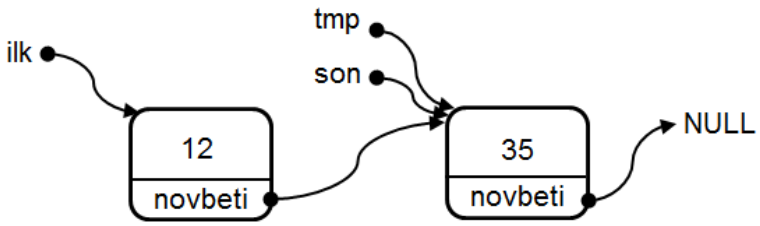
```
tmp->x = 35;

tmp->novbeti = NULL;
```



Yeni yaratdığımız elementi siyahıya əlavə edək, amma nəzərə alaq ki artıq siyahı boş deyil. Bu barədə ətraflı yuxarıda söhbət etmişik.

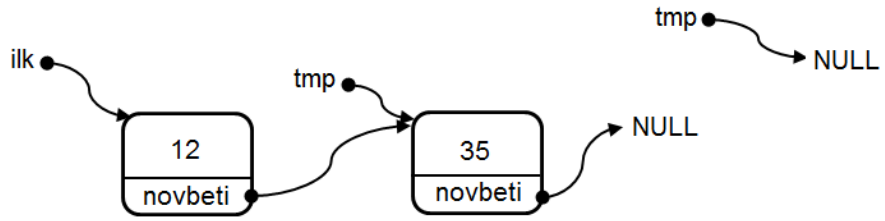
```
syh->son->novbeti  = tmp;
syh->son  = tmp;
syh->son->novbeti  = NULL;
```

Nəticədə ikinci element də siyahıya əlavə olunar



və tmp ilə siyahının əlaqəsini kəsirik

```
tmp = NULL;
```



Bu şəkildə siyahıya istənilən sayda element əlavə edə bilərik. Aşağıda siyahı yaradan və ona istifadəçinin daxil etdiyi 5 müxtəlif ədədi yerləşdirən proqram təqdim olunur.

```
#include <iostream>

using namespace std;

struct el {
	int x;
	struct el *novbeti;
};

struct siyahi {
	struct el *ilk;
	struct el *son;
};


void elave_et(struct siyahi *syh, int eded) {

	struct el *tmp;

	tmp = new struct el;
	tmp->x = eded;
	tmp->novbeti = NULL;

	//siyahinin bosh olmasini yoxlayiriq
	if (syh->ilk == NULL && syh->son == NULL) {
		//siyahi boshdur
		syh->ilk = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
	else
	{
		//siyahida elementler var
		syh->son->novbeti = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
}


void	cap_et(struct siyahi *syh) {

	struct el	*tmp;
```

```
	tmp = syh->ilk;

	cout << "Siyahinin elementleri : \n";

	while (tmp != NULL) {
		cout << tmp->x << " ";
		tmp = tmp->novbeti;
	}

	cout << "\n";
}


int main() {

	struct siyahi syh;
	int i, y;

	syh.ilk = NULL;
	syh.son = NULL;

	cout << "5 muxtelif eded daxil edin:\n";

	for (i = 0; i < 5; ++i) {
		cin >> y;
		elave_et(&syh, y);
	}

	cap_et(&syh);

}
```

Bu proqramı icra eləsək aşağıdakı nəticəni verər:

```
5 muxtelif eded daxil edin:
23
54
678
1
90
Siyahinin elementleri :
23 54 678 1 90
```

Çalışma. Yuxarıda tərtib olunan siyahıda verilimş elementi tapan proqram tərtib edin.

Siyahını tərtib eləməyi, ona element əlavə eləməyi başa düşdükdən sonra, onda verilmiş elemetin axtarılması elə də çətin olmamalıdır. Bunun üçün əvvəlcə siyahı elemetin tipindən göstərici elan edirik və bu göstəricini siyahının ilk elementinə mənimsədirik. Daha sonra **cap_et**() funksiyasında olduğu kimi elan etdiyimiz göstərici vastəsilə siyahının ilk elementindən son elementinə kimi iterasiya edib, tələb olunan

elementi (elementdə olan qiyməti) axtarırıq. Bunun üçün axtar() adlı aşağıdakı kimi funksiya tərtib edək

```
// siyahida verilmish ededin oldugunu yoxlayan funksiya
struct el * axtar(struct siyahi *syh, int eded) {

	struct el *tmp;

	tmp = syh->ilk;

	while (tmp != NULL) {

		if (tmp->x == eded)
			return tmp;

		tmp = tmp->novbeti;
	}

	return NULL;
}
```

axtar() funksiyası əgər axtarılan element siyahıda mövcuddursa onun ünvanın qaytarır, əks halda isə NULL qaytarır. axtar() funksiyasının qaytardığı nəticəni NULL ilə müqaisə edib tələb olunan elementin siyahıda olub-olmadığını müəyyən edə bilərik. Baxdığımız test proqramında siyahıda elementin tapılmasının elə də əhəmiyyəti hiss olunmadı. Gəlin bir qədər fərqli proqram ilə siyahıda axtarış həllinin əhəmiyyətini daha da düzgün anlamağa çalışaq. Bunun üçün siyahının element tipini bir qədər dəyişək, ona bəzi yeni həddlər əlavə edək, aşağıdakı kimi:

```
struct el {
	char ad[100];
	int yash;
	struct el *novbeti;
};
```

Burada biz siyahının elementində iki məlumat həddi sətir tipli ad və int tipli yash həddlərini əlavə etdik. Elementdə istənilən qədər məlumat həddi yerləşdirə bilərik. İni belə bir proqram quraq: siyahı yaradaq və ona istifadəçinin daxil etdiyi məlumatları yerləşdirək. Daha sonra istifadəçidən hər-hansı yaş daxil etməsini istəyək və yaşı istifadəçinin daxil etdiyi yaşdan yuxarı olanların adlarını çap edək. Tələb olunan elementləri tapmaq üçün yuxarıdakı axtar() funksiyasından istifadə edəcəyik, lakin onda bir qədər dəyişiklik aparacağıq. Proqram kodu aşağıdakı kimi olar.

```
#include <iostream>

using namespace std;

struct el {
```

```
	char ad[100];
	int yash;
	struct el *novbeti;
};

struct siyahi {
	struct el *ilk;
	struct el *son;
};


void elave_et(struct siyahi *syh, char *ad, int yash) {

	struct el *tmp;

	tmp = new struct el;
	tmp->yash = yash;
	strcpy(tmp->ad, ad);
	tmp->novbeti = NULL;

	//siyahinin bosh olmasini yoxlayiriq
	if (syh->ilk == NULL && syh->son == NULL) {
		//siyahi boshdur
		syh->ilk = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
	else
	{
		//siyahida elementler var
		syh->son->novbeti = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
}

//yashi verilmish hedden yuxari olan elementlerin ad -laini cap et
void axtar(struct siyahi *syh, int hedd) {

	struct el *tmp;

	tmp = syh->ilk;

	cout << "Yashi "<<hedd << " -den yuxari olanlar\n";

	while (tmp != NULL) {

		if (tmp->yash >= hedd)
			cout << tmp->ad << " ";

		tmp = tmp->novbeti;
	}

}
```

```
void   cap_et(struct siyahi *syh) {

	struct el  *tmp;
	tmp = syh->ilk;

	cout << "Siyahinin elementleri : \n";

	while (tmp != NULL) {
		cout << tmp->ad << " : " << tmp->yash<<" ";
		tmp = tmp->novbeti;
	}

	cout << "\n";
}


int main() {

	struct siyahi syh;
	char ad[100];
	int i, yash, hedd;

	syh.ilk = NULL;
	syh.son = NULL;

	cout << "5 dene ad ve yash daxil edin:\n";

	for (i = 0; i < 5; ++i) {
		cin >> ad >> yash;
		elave_et(&syh, ad,  yash);
	}

	cout << "Teleb olunan yash heddinin daxil edin\n";
	cin >> hedd;

	axtar(&syh, hedd);
}
```

Nəticə

```
5 dene ad ve yash daxil edin:
Ahmed 32
Ali 24
Tahir 41
Letif 16
Imran 50
Teleb olunan yash heddinin daxil edin
30
Yashi 30 -den yuxari olanlar
Ahmed Tahir Imran
```

15.5 Siyahıdan elementlərin silinməsi

İndi axtarışa nisbətən bir qədər çətin məsələ - siyahıdan elemetin silinməsi məsələsi ilə məşğul olaq. Bunun əvvəlcə proqramı daxil edəcəyik, daha sonra isə izah üzərindən siyahıdan elementin silinməsini izah edəcəyik. Aşağıdakı kimi proqram qurmaq istəyirik. İstifadəçi siyahıya bəzi elementlər daxil edir və daha sonra silinməli olan elementi daxil edir (elementin hər-hansı həddini). Daha sonra siyahıda axtarış aparıb tələb olunan element üzərinə gəlirik və onu siyahıdan silirik. Proqramı kodunu təqdim etdikdən və nümunə nəticə göstərdikdən sonra məhs bu elementin silinmə hissəsi üzərində addım-addım dayanaraq izah verəcəyik. Kod belə olar:

```
#include <iostream>

using namespace std;

struct el {
	char ad[100];
	int yash;
	struct el *novbeti;
};

struct siyahi {
	struct el *ilk;
	struct el *son;
};


void elave_et(struct siyahi *syh, char *ad, int yash) {

	struct el *tmp;

	tmp = new struct el;
	tmp->yash = yash;
	strcpy(tmp->ad, ad);
	tmp->novbeti = NULL;

	//siyahinin bosh olmasini yoxlayiriq
	if (syh->ilk == NULL && syh->son == NULL) {
		//siyahi boshdur
		syh->ilk = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
	else
	{
		//siyahida elementler var
		syh->son->novbeti = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
}

//yashi verilmish hedden yuxari olan elementlerin ad -laini cap et
```

```
struct el * axtar(struct siyahi *syh, int hedd) {

	struct el *tmp;

	tmp = syh->ilk;

	while (tmp != NULL) {

		if (tmp->yash == hedd)
			return tmp;

		tmp = tmp->novbeti;
	}

	return NULL;
}


void   cap_et(struct siyahi *syh) {

	struct el  *tmp;
	tmp = syh->ilk;

	cout << "Siyahinin elementleri : \n";

	while (tmp != NULL) {
		cout << tmp->ad << "  \t" << tmp->yash<<"  \n";
		tmp = tmp->novbeti;
	}

	cout << "\n";
}

void sil(struct siyahi *syh, struct el *elem) {

	struct el *tmp;

	cout << "Ashagidaki element siyahidan silinecek: \n"
		<< elem->ad << " " << elem->yash << "\n";

	//elementi sil
	if (elem == syh->ilk) {

		if (syh->ilk == syh->son) {
			syh->ilk = NULL;
			syh->son = NULL;
			delete elem;
		}
		else {
			syh->ilk = syh->ilk->novbeti;
			elem->novbeti = NULL;
			delete elem;
		}
	}
	else {
```

```
            tmp = syh->ilk;

            while (tmp->novbeti != elem)
                  tmp = tmp->novbeti;

            //tmp->novbeti == elem
            tmp->novbeti = elem->novbeti;
            elem->novbeti = NULL;
            delete elem;
      }

}


int main() {

      struct siyahi syh;
      char ad[100];
      int i, yash, hedd;

      syh.ilk = NULL;
      syh.son = NULL;

      cout << "5 dene ad ve yash daxil edin:\n";

      for (i = 0; i < 5; ++i) {
            cin >> ad >> yash;
            elave_et(&syh, ad,  yash);
      }

      cout << "Siyahidan silinmeli olan elementin yash heddini daxil edin\n";
      cin >> hedd;

      struct el *tmp;

      tmp = axtar(&syh, hedd);

      if (tmp == NULL)
            cout << "Siayhida yashi " << hedd << " olan element yoxdur\n";
      else
            sil(&syh, tmp);

      //siyahinin elementlerini cap edek
      cap_et(&syh);
}
```

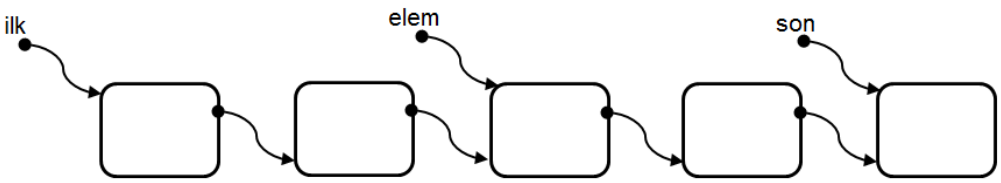
Nəticə

```
5 dene ad ve yash daxil edin:
Ahmed 23
Ali 31
Taleh 70
Ruslan 55
Ibish 27
Siyahidan silinmeli olan elementin yash heddini daxil edin
55
```

```
Ashagidaki element siyahidan silinecek:
Ruslan 55
Siyahinin elementleri :
Ahmed   23
Ali     31
Taleh   70
Ibish   27
```

Proqramın məntiqi bizə aydındır. Gəlin elə bizim üçün önəmli olan sil funksiyası üzərində dayanaq. sil() funksiyası iki parametr qəbul edir, siyahı və siyahının silinməli olan elementinə istinad.

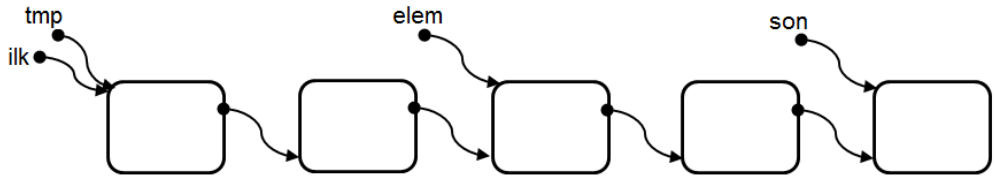
```
void sil(struct siyahi *syh, struct el *elem)
```



İlk elementin silinməsi sadə olduğuna görə siyahıda bir neçə element olduğu halı nəzərdən keçirək. Beləliklə siyahımız tərtib olunub və elem göstəricisi silinməli olan elementə istinad edir.

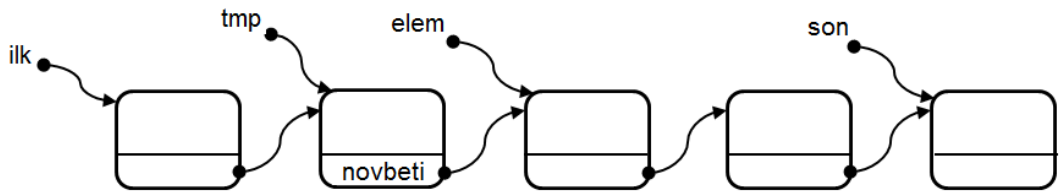
İlk öncə tmp adlı əlavə göstərici elan edib siyahının ilk elementinə məimsədirik.

```
tmp = syh->ilk;
```



Daha sonra dövrü ilə tmp-nin növbəti həddi elem –in üzərində dayanana qədər siyahı üzrə hərəkət edirik. Ühile dövrü `tmp->novbeti –nin` qiyməti elem -ə bərabər olanda dayanacaq.

```
while (tmp->novbeti != elem)
      tmp = tmp->novbeti;
```

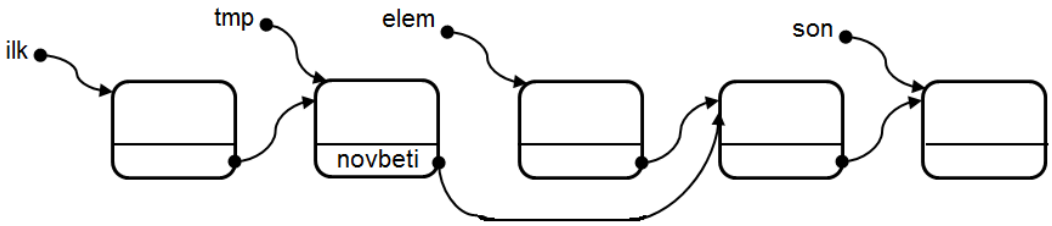


Bütün bunlar hamısı hazırlıq işləri idi. İndi siyahını silməyə tam hazırıq. Bunun üçün yerinə yetirilən kodları bir-bir nəzərdən keçirək və qrafik izləyək.

Əvvəlcə tmp-nin novbeti həddini hansı ki hal-hazırda elem -ə istinad edir elem-dən ayırıb siyahıda elemendən sonra gələn elementə birləşdiririk. Siyahıda elem –dən sonra gələn elemenin ünvanın isə elem –in novbeti həddindən əldə edirik.

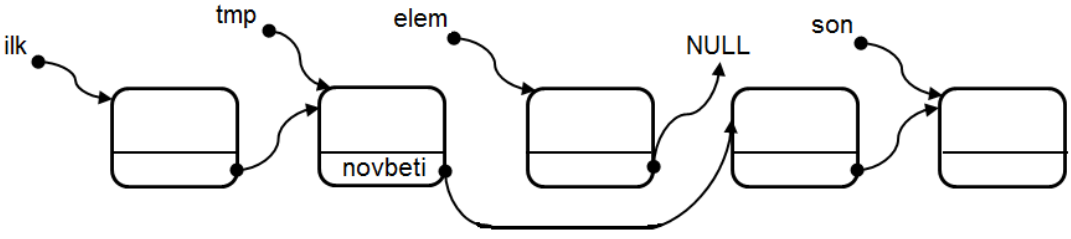
```
tmp->novbeti = elem->novbeti;
```

Nəticə belə olar



Məhs bu kod siyahının elementlərinin birləşmə ardıcıllığından elem –i ayırır, lakin silmək istədiyimiz elementin novbeti həddi hələ də siyahıya istinad edir ki, bu da təhlükəlidir. İlk əvvəl həmin əlaqəni kəsməliyik, hansı ki aşağıdakı kod bu işi görür.

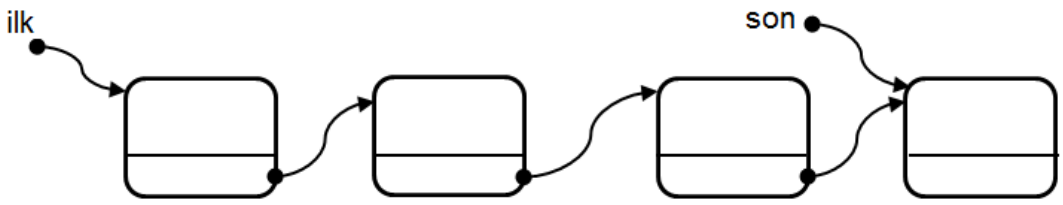
```
elem->novbeti = NULL;
```



Gördüyümüz kimi elem siyahıdan artıq təmamilə ayrılmışdır. Son olaraq siyahıdan kənarlaşdırdığımız elementi yaddaşdan silməklə həmin element üçün yaddaşda ayrılmış yeri də ləğv edirik

```
delete elem;
```

Nəticədə elem göstəricisinin istinad elədiyi element yaddaşdan silinər.

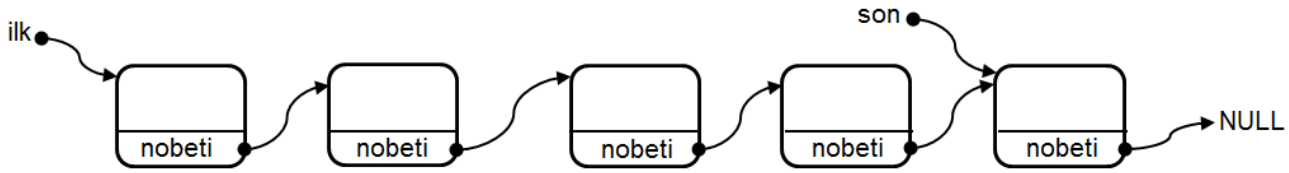


Əslində səliqəli proqramlaşdırma prinsipi baxımdan tmp –ni də NULL –a mənimsətməli idik.

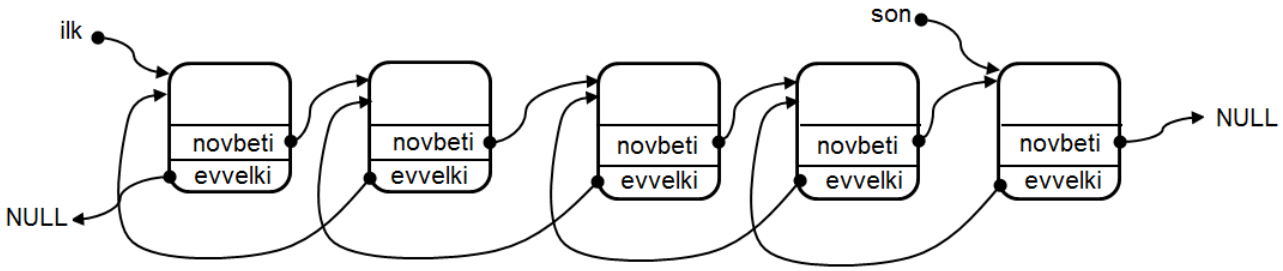
```
tmp = NULL;
```

15.6 Iki istiqamətli siyahılar

Yuxarıda tanış olduğumuz siyahı ***biristiqamətli*** siyahı adlanır. Çünki burada hər-bir element yalnız özündən sonrakı elementlə birləşir.



Siyahıların digər bir növü olan ikiistiqamətli siyahılarda isə hər element özündən həm sonra, həm də əvvəl gələn elementlə əlaqələnmiş olur.



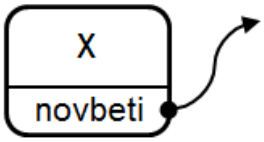
İndi isə biristiqamətli siyahılarda olduğu kimi ikiistiqamətli siyahıların da yaradılması, siyahıya yeni element əlavə olunması, axtarış və silinmə işlərinin nə cür yerinə yetirildiyi ilə tanış olaq.

15.6.1 İkiistiqamətli siyahının yaradılması

Əvvəlcə yadımıza salaq ki, bir istiqamətli siyahı üçün aşağıdakı kimi nümunə element tipi yaratmışdıq

```
struct el {
      int x;
      struct el *novbeti;
};
```

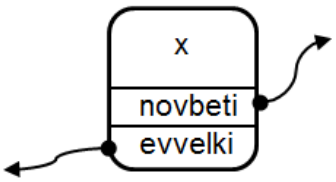
Burada elementin öz tipindən olan novbeti adlı göstərici həddən siyahıda bu elementdən sonra gələn elementə birləşmək üçün istifadə edirdik.



İkiistiqamətli siyahıda isə hər bir elementin həm özündən əvvəl gələn, həm də sonra gələn elementlə birləşə bilməsi üçün artıq iki dənə "***qolu***" olmalıdır. Başqa sözlə elementin öz tipinə olan göstərici həddən iki dənə elan etməliyik.

```
struct el {
	int x;
	struct el *novbeti;
	struct el *evvelki;

};
```

Burada adlandırma təbii ki, şərtidir və istənilən həddi istənilən elementə birləşdirə bilərik. Xatırlayaq ki birləşdirmək deyəndə müvafiq elementin ünvanın göstəriciyə mənimsətməyi nəzərdə tuturduq. Biz **novbeti** həddindən sonrakı, **evvelki** həddindən isə əvvəl gələn elementə birləşmək üçün istifadə edəcəyik.



Aşağıdakı nümunə kod ikiistiqamətli siyahıya 5 element daxil edir və siyahının elementlərini əvvəldən sona və sondan əvvələ doğru çap edir.

```
#include <iostream>

using namespace std;

struct el {
	int x;
	struct el *novbeti;
	struct el *evvelki;
};

struct siyahi {
	struct el *ilk;
	struct el *son;
```

```
};


void elave_et(struct siyahi *syh, int eded) {

	struct el *tmp;

	tmp = new struct el;
	tmp->x = eded;
	tmp->novbeti = NULL;
	tmp->evvelki = NULL;

	//siyahinin bosh olmasini yoxlayiriq
	if (syh->ilk == NULL && syh->son == NULL) {
		//siyahi boshdur
		syh->ilk = tmp;
		syh->son = tmp;
		syh->son->novbeti = NULL;
		syh->son->evvelki = NULL;
	}
	else
	{
		//siyahida elementler var
		syh->son->novbeti = tmp;
		tmp->evvelki = syh->son;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
}


void   cap_et_evvelden_sona(struct siyahi *syh) {

	struct el  *tmp;
	tmp = syh->ilk;

	cout << "Siyahinin elementleri evvelden sona: \n";

	while (tmp != NULL) {
		cout << tmp->x << " ";
		tmp = tmp->novbeti;
	}

	cout << "\n";
}

void   cap_et_sondan_evvele(struct siyahi *syh) {

	struct el  *tmp;
	tmp = syh->son;

	cout << "Siyahinin elementleri sondan evvele: \n";

	while (tmp != NULL) {
		cout << tmp->x << " ";
		tmp = tmp->evvelki;
```

```
	}

	cout << "\n";
}


int main() {

	struct siyahi syh;
	int i, y;

	syh.ilk = NULL;
	syh.son = NULL;

	cout << "5 muxtelif eded daxil edin:\n";

	for (i = 0; i < 5; ++i) {
		cin >> y;
		elave_et(&syh, y);
	}

	cap_et_evvelden_sona(&syh);
	cap_et_sondan_evvele(&syh);

}
```

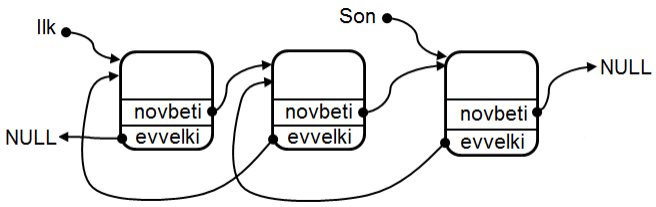
Nümunə nəticə:

```
5 muxtelif eded daxil edin:
2 45 90 -1 17
Siyahinin elementleri evvelden sona:
2 45 90 -1 17
Siyahinin elementleri sondan evvele:
17 -1 90 45 2
```

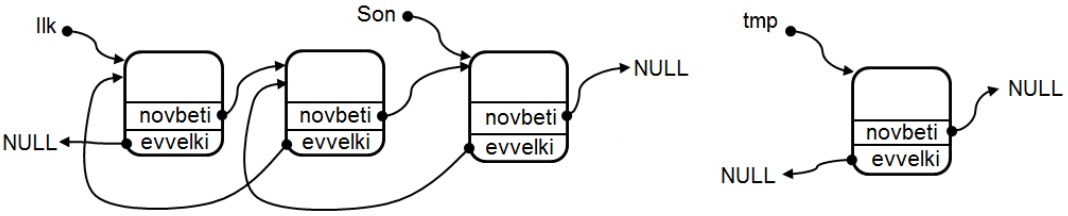
Əgər biristiqamətli siyahıları yaxşı başa düşmüsünüzsə onda ikiistiqamətli siyahılar üçün olan oxşar kodu da çətinlik çəkmədən başa düşə bilməlisiniz. Əks halda biristiqamətli siyahıları yenidən nəzərdən keçirməyiniz gərəkir. Bu kodda ən mühüm məqam yeni elementin siyahıya əlavə olunması hissəsidir və əlbəttə ki boş siyahıya yox, özündə müəyyən elementləri olan siyahıya. Yuxarıdakı proqramda dediyimiz prosedura uyğun kod kəsimi aşağıdakı kimidir:

```
	else
	{
		//siyahida elementler var
		syh->son->novbeti = tmp;
		tmp->evvelki = syh->son;
		syh->son = tmp;
		syh->son->novbeti = NULL;
	}
```

Məhs bu kod vastəsilə biz siyahının son elementinin novbeti həddini yeni əlavə olunan elementə mənimsədirik və yeni əlavə olunan elementin evvelki həddini siyahının son elementinə mənimsədirik. Gəlin bu kodu addım –addım qrafik təsvirlərin köməyilə nəzərdən keçirək(məsləhət görürəm ki siz də kağız-qələm götürüb birlikdə davam edəsiniz). Tutaq ki aşağıdakı kimi siyahımız var:



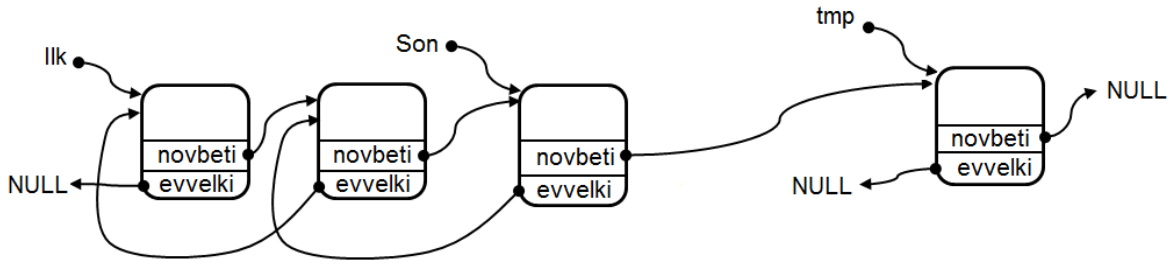
və biz bu siyahıya (sonuna) yeni element əlavə etmək istəyirik:



Bunun üçün yuxarıda göstərdiyimiz kodu sətir-sətir icra edək:

```
syh->son->novbeti = tmp;
```

Bu kodda siyahının son elementinin novbeti heddi siyahıya əlavə etmək istədiyimiz yeni elementə mənimsədilir.



Bu hissə biristiqamətli siyahılarda olduğu kimidir. İkinci sətrə baxaq:

```
tmp->evvelki = syh->son;
```

Bu zaman yeni əlavə olunan elementin evvelki həddi siyahının son elementinə mənimsədilir (özündən əvvəlki).

Əlavələr.

Əlavə A
ASCİİ cədvəli

ascii code	0	**NULL**	(Null character)
ascii code	1	**SOH**	(Start of Header)
ascii code	2	**STX**	(Start of Text)
ascii code	3	**ETX**	(End of Text)
ascii code	4	**EOT**	(End of Transmission)
ascii code	5	**ENQ**	(Enquiry)
ascii code	6	**ACK**	(Acknowledgement)
ascii code	7	**BEL**	(Bell)
ascii code	8	**BS**	(Backspace)
ascii code	9	**HT**	(Horizontal Tab)
ascii code	10	**LF**	(Line feed)
ascii code	11	**VT**	(Vertical Tab)
ascii code	12	**FF**	(Form feed)
ascii code	13	**CR**	(Carriage return)
ascii code	14	**SO**	(Shift Out)
ascii code	15	**SI**	(Shift In)
ascii code	16	**DLE**	(Data link escape)
ascii code	17	**DC1**	(Device control 1)
ascii code	18	**DC2**	(Device control 2)
ascii code	19	**DC3**	(Device control 3)
ascii code	20	**DC4**	(Device control 4)
ascii code	21	**NAK**	(Negative acknowledgement)
ascii code	22	**SYN**	(Synchronous idle)
ascii code	23	**ETB**	(End of transmission block)
ascii code	24	**CAN**	(Cancel)
ascii code	25	**EM**	(End of medium)
ascii code	26	**SUB**	(Substitute)
ascii code	27	**ESC**	(Escape)
ascii code	28	**FS**	(File separator)
ascii code	29	**GS**	(Group separator)
ascii code	30	**RS**	(Record separator)
ascii code	31	**US**	(Unit separator)
ascii code	32		(space)
ascii code	33	**!**	(exclamation mark)
ascii code	34	**"**	(Quotation mark)
ascii code	35	#	(Number sign)
ascii code	36	$	(Dollar sign)
ascii code	37	%	(Percent sign)
ascii code	38	&	(Ampersand)
ascii code	39	'	(Apostrophe)
ascii code	40	(	(round brackets or parentheses)
ascii code	41	)	(round brackets or parentheses)
ascii code	42	*	(Asterisk)
ascii code	43	+	(Plus sign)
ascii code	44	,	(Comma)

ascii code	45	-	(Hyphen)
ascii code	46	.	(Full stop , dot)
ascii code	47	/	(Slash)
ascii code	48	0	(number zero)
ascii code	49	1	(number one)
ascii code	50	2	(number two)
ascii code	51	3	(number three)
ascii code	52	4	(number four)
ascii code	53	5	(number five)
ascii code	54	6	(number six)
ascii code	55	7	(number seven)
ascii code	56	8	(number eight)
ascii code	57	9	(number nine)
ascii code	58	:	(Colon)
ascii code	59	;	(Semicolon)
ascii code	60	<	(Less-than sign)
ascii code	61	=	(Equals sign)
ascii code	62	>	(Greater-than sign ; Inequality)
ascii code	63	?	(Question mark)
ascii code	64	@	(At sign)
ascii code	65	A	(Capital A)
ascii code	66	B	(Capital B)
ascii code	67	C	(Capital C)
ascii code	68	D	(Capital D)
ascii code	69	E	(Capital E)
ascii code	70	F	(Capital F)
ascii code	71	G	(Capital G)
ascii code	72	H	(Capital H)
ascii code	73	I	(Capital I)
ascii code	74	J	(Capital J)
ascii code	75	K	(Capital K)
ascii code	76	L	(Capital L)
ascii code	77	M	(Capital M)
ascii code	78	N	(Capital N)
ascii code	79	O	(Capital O)
ascii code	80	P	(Capital P)
ascii code	81	Q	(Capital Q)
ascii code	82	R	(Capital R)
ascii code	83	S	(Capital S)
ascii code	84	T	(Capital T)
ascii code	85	U	(Capital U)
ascii code	86	V	(Capital V)
ascii code	87	W	(Capital W)
ascii code	88	X	(Capital X)
ascii code	89	Y	(Capital Y)
ascii code	90	Z	(Capital Z)
ascii code	91	[	(square brackets or box brackets)
ascii code	92	\	(Backslash)
ascii code	93	]	(square brackets or box brackets)
ascii code	94	^	(Caret or circumflex accent)
ascii code	95	_	(underscore, under strike, underbar or low line)
ascii code	96	`	(Grave accent)
ascii code	97	a	(Lowercase a)
ascii code	98	b	(Lowercase b)
ascii code	99	c	(Lowercase c)
ascii code	100	d	(Lowercase d)
ascii code	101	e	(Lowercase e)
ascii code	102	f	(Lowercase f)
ascii code	103	g	(Lowercase g)
ascii code	104	h	(Lowercase h)
ascii code	105	**i**	(Lowercase i)
ascii code	106	j	(Lowercase j)

ascii code	107	k	(Lowercase k)
ascii code	108	l	(Lowercase l)
ascii code	109	m	(Lowercase m)
ascii code	110	n	(Lowercase n)
ascii code	111	o	(Lowercase o)
ascii code	112	p	(Lowercase p)
ascii code	113	q	(Lowercase q)
ascii code	114	r	(Lowercase r)
ascii code	115	s	(Lowercase s)
ascii code	116	t	(Lowercase t)
ascii code	117	u	(Lowercase u)
ascii code	118	v	(Lowercase v)
ascii code	119	w	(Lowercase w)
ascii code	120	x	(Lowercase x)
ascii code	121	y	(Lowercase y)
ascii code	122	z	(Lowercase z)
ascii code	123	{	(curly brackets or braces)
ascii code	124 |		(vertical-bar, vbar, vertical line or vertical slash)
ascii code	125	}	(curly brackets or braces)
ascii code	126	~	(Tilde ; swung dash)
ascii code	127	DEL	(Delete)
ascii code	128	Ç	(Majuscule C-cedilla)
ascii code	129	ü	(letter "u" with umlaut or diaeresis ; "u-umlaut")
ascii code	130	é	(letter "e" with acute accent or "e-acute")
ascii code	131	â	(letter "a" with circumflex accent or "a-circumflex")
ascii code	132	ä	(letter "a" with umlaut or diaeresis ; "a-umlaut")
ascii code	133	à	(letter "a" with grave accent)
ascii code	134	å	(letter "a" with a ring)
ascii code	135	ç	(Minuscule c-cedilla)
ascii code	136	ê	(letter "e" with circumflex accent or "e-circumflex")
ascii code	137	ë	(letter "e" with umlaut or diaeresis ; "e-umlaut")
ascii code	138	è	(letter "e" with grave accent)
ascii code	139	ï	(letter "i" with umlaut or diaeresis ; "i-umlaut")
ascii code	140	î	(letter "i" with circumflex accent or "i-circumflex")
ascii code	141	ì	(letter "i" with grave accent)
ascii code	142	Ä	(letter "A" with umlaut or diaeresis ; "A-umlaut")
ascii code	143	Å	(letter "A" with a ring)
ascii code	144	É	(Capital letter "E" with acute accent or "E-acute")
ascii code	145	æ	(Latin diphthong "ae")
ascii code	146	Æ	(Latin diphthong "AE")
ascii code	147	ô	(letter "o" with circumflex accent or "o-circumflex")
ascii code	148	ö	(letter "o" with umlaut or diaeresis ; "o-umlaut")
ascii code	149	ò	(letter "o" with grave accent)
ascii code	150	û	(letter "u" with circumflex accent or "u-circumflex")
ascii code	151	ù	(letter "u" with grave accent)
ascii code	152	ÿ	(letter "y" with diaeresis)
ascii code	153	Ö	(letter "O" with umlaut or diaeresis ; "O-umlaut")
ascii code	154	Ü	(letter "U" with umlaut or diaeresis ; "U-umlaut")
ascii code	155	ø	(slashed zero or empty set)
ascii code	156	£	(Pound sign ; symbol for the pound sterling)
ascii code	157	Ø	(slashed zero or empty set)
ascii code	158	×	(multiplication sign)
ascii code	159	*ƒ*	(function sign ; f with hook sign ; florin sign)
ascii code	160	á	(letter "a" with acute accent or "a-acute")
ascii code	161	í	(letter "i" with acute accent or "i-acute")
ascii code	162	ó	(letter "o" with acute accent or "o-acute")
ascii code	163	ú	(letter "u" with acute accent or "u-acute")
ascii code	164	ñ	(letter "n" with tilde ; enye)
ascii code	165	Ñ	(letter "N" with tilde ; enye)
ascii code	166	ª	(feminine ordinal indicator)
ascii code	167	º	(masculine ordinal indicator)

ascii code	168	¿	(Inverted question marks)
ascii code	169	®	(Registered trademark symbol)
ascii code	170	¬	(Logical negation symbol)
ascii code	171	½	(One half)
ascii code	172	¼	(Quarter or one fourth)
ascii code	173	¡	(Inverted exclamation marks)
ascii code	174	«	(Guillemets or angle quotes)
ascii code	175	»	(Guillemets or angle quotes)
ascii code	176	▒	
ascii code	177	▒	
ascii code	178	▓	
ascii code	179	│	(Box drawing character)
ascii code	180	┤	(Box drawing character)
ascii code	181	Á	(Capital letter "A" with acute accent or "A-acute")
ascii code	182	Â	(letter "A" with circumflex accent or "A-circumflex")
ascii code	183	À	(letter "A" with grave accent)
ascii code	184	©	(Copyright symbol)
ascii code	185	╣	(Box drawing character)
ascii code	186	║	(Box drawing character)
ascii code	187	╗	(Box drawing character)
ascii code	188	╝	(Box drawing character)
ascii code	189	¢	(Cent symbol)
ascii code	190	¥	(YEN and YUAN sign)
ascii code	191	┐	(Box drawing character)
ascii code	192	└	(Box drawing character)
ascii code	193	┴	(Box drawing character)
ascii code	194	┬	(Box drawing character)
ascii code	195	├	(Box drawing character)
ascii code	196	─	(Box drawing character)
ascii code	197	┼	(Box drawing character)
ascii code	198	ã	(letter "a" with tilde or "a-tilde")
ascii code	199	Ã	(letter "A" with tilde or "A-tilde")
ascii code	200	╚	(Box drawing character)
ascii code	201	╔	(Box drawing character)
ascii code	202	╩	(Box drawing character)
ascii code	203	╦	(Box drawing character)
ascii code	204	╠	(Box drawing character)
ascii code	205	═	(Box drawing character)
ascii code	206	╬	(Box drawing character)
ascii code	207	¤	(generic currency sign)
ascii code	208	ð	(lowercase "eth")
ascii code	209	Ð	(Capital letter "Eth")
ascii code	210	Ê	(letter "E" with circumflex accent or "E-circumflex")
ascii code	211	Ë	(letter "E" with umlaut or diaeresis ; "E-umlaut")
ascii code	212	È	(letter "E" with grave accent)
ascii code	213	ı	(lowercase dot less i)
ascii code	214	Í	(Capital letter "I" with acute accent or "I-acute")
ascii code	215	Î	(letter "I" with circumflex accent or "I-circumflex")
ascii code	216	Ï	(letter "I" with umlaut or diaeresis ; "I-umlaut")
ascii code	217	┘	(Box drawing character)
ascii code	218	┌	(Box drawing character)
ascii code	219	█	(Block)
ascii code	220	▄	
ascii code	221	¦	(vertical broken bar)
ascii code	222	Ì	(letter "I" with grave accent)
ascii code	223	▀	
ascii code	224	Ó	(Capital letter "O" with acute accent or "O-acute")
ascii code	225	ß	(letter "Eszett" ; "scharfes S" or "sharp S")
ascii code	226	Ô	(letter "O" with circumflex accent or "O-circumflex")
ascii code	227	Ò	(letter "O" with grave accent)
ascii code	228	õ	(letter "o" with tilde or "o-tilde")
ascii code	229	Õ	(letter "O" with tilde or "O-tilde")

ascii code	230	µ	(Lowercase letter "Mu" ; micro sign or micron)
ascii code	231	þ	(capital letter "Thorn")
ascii code	232	Þ	(lowercase letter "thorn")
ascii code	233	Ú	(Capital letter "U" with acute accent or "U-acute")
ascii code	234	Û	(letter "U" with circumflex accent or "U-circumflex")
ascii code	235	Ù	(letter "U" with grave accent)
ascii code	236	ý	(letter "y" with acute accent)
ascii code	237	Ý	(Capital letter "Y" with acute accent)
ascii code	238	¯	(macron symbol)
ascii code	239	´	(Acute accent)
ascii code	240	¬	(Hyphen)
ascii code	241	±	(Plus-minus sign)
ascii code	242	_	(underline or underscore)
ascii code	243	¾	(three quarters)
ascii code	244	¶	(paragraph sign or pilcrow)
ascii code	245	§	(Section sign)
ascii code	246	÷	(The division sign ; Obelus)
ascii code	247	¸	(cedilla)
ascii code	248	°	(degree symbol)
ascii code	249	¨	(Diaeresis)
ascii code	250	•	(Interpunct or space dot)
ascii code	251	¹	(superscript one)
ascii code	252	³	(cube or superscript three)
ascii code	253	²	(Square or superscript two)
ascii code	254	■	(black square)
ascii code	255	**nbsp**	(non-breaking space or no-break space)