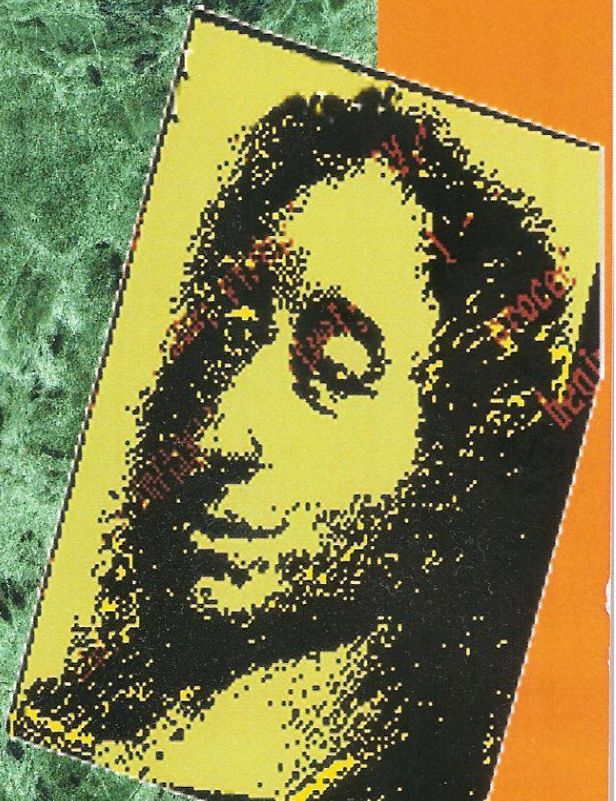
Z.T. Məhərrəmov

Turbo Pascal



next.pas

Atamın əziz xatirəsinə ithaf edirəm...

Z.T. Məhərrəmov

turbo PASCAL

BAKI – 2013

Rəy verənlər:

Texnika elmləri doktoru, professor **Əli Həsən oğlu Nağıyev**
Sumqayıt Dövlət Universiteti

Texnika elmləri namizədi, **Faiq Həsən oğlu Hacıyev**
Azərbaycan Dövlət Neft Akademiyası

Elmi redaktor:
Texnika elmləri namizədi, dosent **Babək Abdulla oğlu Abasov**
Heydər Əliyev adına AAHM

Məhərrəmov Z.T.
Turbo Pascal. Bakı: Nəşriyyat, 2013. – 212 s.

Kitab proqramlaşdırmanı öyrənmək üçün çox sadə və ən yaxşı vasitə olan Turbo Pascal dilinin öyrədilməsinə həsr edilmişdir. Bütün izahatlar zəngin praktiki misallarla müşayiət olunmuşdur. Bu misallar nəinki Turbo Pascal dilini, həm də proqramlaşdırmanın ümumi metod və prinsiplərini öyrənməyə imkan verir. Ona görə də kitab digər proqramlaşdırma dillərinin, o cümlədən Delphi dilinin asanlıqla öyrənilməsinə zəmin yaradır.

Kitab ilk növbədə proqramlaşdırmanı öyrənməyə başlayanlar və ali məktəb tələbələri üçün nəzərdə tutulmuşdur. Kitabdan magistrlər, doktorantlar və proqramlaşdırmanın tədrisi ilə məşğul olan müəllimlər də istifadə edə bilərlər.

M ü n d ə r i c a t

Birinci fəsil

TURBO PASCAL DİLİNİN İNTEQRALLAŞDIRILMIŞ MÜHİTİ

Bu fəsildə Pascal dilinin **Turbo Pascal 7.0, Borland Pascal 7.0, Turbo Pascal for Windows 1.5** və **Borland Pascal for Windows 7.0** versiyalarının inteqrallaşdırılmış mühiti ilə tanış olacaq, yeni proqramların yaradılması, mövcud proqramların açılması, onların sazlanması, kompilyasiyası və yerinə yetirilməsi və habelə inteqrallaşdırılmış mühitin idarə edilməsi ilə əlaqədar əmrləri öyrənəcəyik.

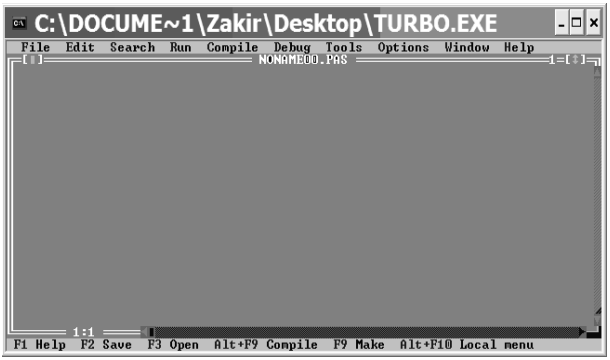
1.1. Turbo Pascal 7.0 dilinin pəncərəsi

Turbo Pascal dilində proqram bu dilin inteqrallaşdırılmış mühitində yaradılır. İnteqrallaşdırılmış mühit dilin kompil-yatorundan, proqram mətnini yığmaq və ona düzəlişlər etmək üçün redaktordan və sazlayıcıdan ibarətdir. İnteqrallaşdırılmış mühiti yükləmək üçün **Turbo.exe** proqramını yükləmək lazımdır. Bu zaman ekranda şəkil 1.1 – də göstərilmiş pəncərə açılacaqdır. Bu pəncərə yuxarı hissədə menyu sətrindən, mərkəzi hissədə işçi oblastdan və aşağı hissədə isə vəziyyətlər sətrindən ibarətdir.

Menyu sətri inteqrallaşdırılmış mühitin əmrlərindən ibarətdir. *File* (*Fayl*) menyusu yeni proqram yaratmaq,

mövcud proqramları çağırmaq, proqramı yadda saxlamaq və s. kimi əmrlərdən ibarətdir. *Edit* (*Düzəliş*) menyusunda proqrama düzəlişlər etməyə (köçürmək, kəsmək, əlavə etmək və s.) imkan verən əmrlər toplanmışdır. Proqramda olan səhvləri aşkar etmək, onu icra və kompilyasiya etmək üçün *Run* (*Yerinə yetirmək*) və *Compile* (*Kompilyasiya*) menyularından istifadə edilir. Bəzi hallarda menyu sətri görünməyəcəkdir. Belə hallarda menyu sətrini göstərmək üçün ***F10*** klavişini basmaq lazımdır.

İşçi oblastda proqram mətni yığılır. Proqram aşağıdakı qayda ilə yaradılır. İnteqrallaşdırılmış mühit yükləndikdən sonra, ekranda boş redaktor pəncərəsi açılmalıdır. Əgər bu pəncərə boş deyildirsə, onda yeni proqram mətni yaratmaq üçün *File/New* (*Fayl/Yeni fayl yaratmaq*) əmrini icra etmək lazımdır. Bu zaman pəncərənin yuxarı hissəsində `NONAME00.PAS` adlı yeni boş pəncərə açılacaqdır. Yeni proqram yaratmaq üçün inteqrallaşdırılmış mühit hər dəfə bu adı təklif edir. Təkidlə məsləhət görürük ki, bu adı dərhal dəyişəsiniz. Bunun üçün *File/Save As...* (*Fayl/Faylı necə*



Şəkil 1.1. Turbo Pascal 7.0 pəncərəsi

yadda saxlamaq...) əmrini icra edərək, yeni açılmış pəncərədə proqrama ad vermək lazımdır. Fayla ad verdikdə, avtomatik olaraq ona *.pas* genişlənmiş hissəsi (kitabın bütün sonrakı fəsillərində biz faylın genişlənmiş hissəsinə *faylın tipi*

deyəcəyik) mənimsədiləcəkdir və bu fayl kompilyatorun yerləşdiyi qovluqda yerləşəcəkdir.

Bundan sonra, yeni proqramın mətni yığılır. Bütün mətn redaktorlarında olduğu kimi, burada da kursordan soldakı simvolu pozmaq üçün *Backspace*, cari simvolu pozmaq üçün *Delete*, yeni sətrə keçmək üçün *Enter*, simvollar registrini dəyişmək üçün *Shift* və baş hərflər rejimini qoşmaq üçün *Caps Lock* klavişlərindən istifadə edilir. Mətnin yığılması zamanı *Edit* (*Düzəlişlər*) menyusunun *Cut* (*Shift+Del – Kəsmək*), *Copy* (*Ctrl+Ins – Köçürmək*), *Paste* (*Shift+Ins – Yerləşdirmək*) və *Clear* (*Ctrl+Del – Pozmaq*) əmrlərindən istifadə etməklə, mətnin yığılması prosesini yüngülləşdirmək olar. Həmçinin hər 10–15 sətirdən bir ***F2*** klavişini basmaqla yığılmış proqram mətnini yadda saxlamaq məsləhət görülür. Bununla da Siz, kompüterdə təsadüfi imtinaların baş verməsi səbəbindən, proqramınızı itirmək təhlükəsindən sığortalanacaqsınız.

İşçi oblastda həm də kompüterdə artıq mövcud olan pascal–proqramların mətninə baxmaq (*.pas* tipli) və onları kompilyasiya etmək olar. Bunun üçün *File/Open...* (*Fayl/Açmaq...*) əmrini icra edib və ya ***F3*** klavişini basıb, açılan pəncərədən faylı seçərək *Open* (*Açmaq*) düyməsini basmaq lazımdır.

Nəhayət, sonuncu vəziyyətlər sətrində, proqramçıya müəyyən klavişlər kombinasiyalarının funksiyaları xatırladılır və müəyyən əməllərin yerinə yetirilməsi haqqında məlumatlar təsvir olunur.

İnteqrallaşdırılmış mühitdən çıxmaq üçün *File/*Exit (*Fayl/Çıxmaq*) əmrini icra etmək və ya ***Alt+X*** klavişlərini basmaq lazımdır.

1.2. Proqramın sazlanması

İnteqrallaşdırılmış mühitdə proqram mətni yığıldıqdan sonra, proqramı sazlamaq lazımdır. Proqramın sazlanması

zamanı proqramda mövcud olan *sintaksis səhvlər* kompilyator tərəfindən aşkar edilir. Proqramı sazlamaq üçün *Compile/Make* əmrini icra etmək və ya ***F9*** klavişini basmaq kifayətdir. Əgər proqramda səhvlər olarsa, kursor həmin mövqedə yerləşəcək və kompilyatorun birinci sətrində həmin səhvin əlaməti haqqında məlumat peyda olacaqdır. Səhvlərin xarakteri haqqında dolğun məlumat almaq üçün inteqrallaşdırılmış mühitin məlumat sisteminə (*Help – Kömək*) müraciət etmək olar.

1.3. Proqramın yerinə yetirilməsi

Proqramda mövcud olan sintaksis səhvlər aradan qaldırıldıqdan sonra, son nəticələr almaq üçün, proqramı icra etmək lazımdır. Bunun üçün *Run* (*Yerinə yetirmək*) menyusundan *Run* (*Yerinə yetirmək*) əmrini icra etmək və ya ***Ctrl+F9*** klavişlərini basmaq lazımdır. Bu zaman ekrana *istifadəçi pəncərəsi* adlanan yeni pəncərə çıxacaqdır. Bu pəncərədə proqram dəyişənlərinə ilkin qiymətlər daxil edilir və nəticələr alınır. Proqramın icrası başa çatdıqdan dərhal sonra kompilyatora qayıdış baş verir, inteqrallaşdırılmış mühit ekrana qayıdır və istifadəçi pəncərəsinin üstünü örtür. Alınmış nəticələri təkrarən görmək və ya onları təhlil etmək üçün istifadəçi pəncərəsini yenidən ekranda göstərmək lazımdır. Kompilyatoru ekrandan müvəqqəti olaraq götürmək üçün *Debug* (*Sazlamaq*) menyusundan *User Screen* (*İstifadəçi pəncərəsi*) əmrini icra etmək və ya ***Alt+F5*** klavişlərini basmaq lazımdır. *Tərtib olunmuş proqram vasitəsilə də istifadəçi pəncərəsini ekranda göstərmək olar.* Bu məqsədlə proqramda sonuncu sətirdən əvvəlki sətirdə, yəni `end.` operatorundan əvvəl `readln` prosedurunu yazmaq lazımdır. Bu halda kompilyator proqrama ilkin verilənin daxil ediləcəyini gözləyir, lakin `readln` prosedurunda heç bir dəyişən yazılmadığı üçün, proqrama

heç bir qiymət daxil edilməyəcək və *Enter* klavişi basılana qədər istifadəçi pəncərəsi bağlanmayacaqdır.

Proqramın yerinə yetirilməsi zamanı da kompilyator tərəfindən səhvlər aşkar edilə bilər. Belə səhvlər *semantik* və ya *alqoritm xarakterli* olur, yəni dəyişənlərin aldığı qiymətlər yolverilə bilən həddi aşır, yolverilməz əməliyyatlar (sıfra bölmə, mənfi ədədin loqarifmlənməsi və ya ondan kvadrat kökalma və s.) baş verir və s. Bu səhvlərə *yerinə yetirmə vaxtının səhvləri* deyilir. Kompilyator yerinə yetirmə vaxtının səhvləri haqqında aşağıdakı məlumatı verir:

```
Run-time error <errnum> at <segment>:<offset>
```

Burada, **<errnum>**– *səhvin kodu*, **:<offset>** – *səhvin baş verdiyi yaddaşın ünvanıdır*.

Turbo Pascal dilinin inteqrallaşdırılmış mühiti proqramın addım–addım yerinə yetirilməsinə imkan verir. Bu zaman proqramın müxtəlif dəyişənlərinin aldığı real qiymətləri görmək mümkün olur ki, bu da proqramın təhlili üçün əvəzsiz məlumatdır. Proqramı belə seansda icra etmək üçün *Run/Trace into* (*Yerinə yetirmək/Sətirlərlə*) əmrini icra etmək və ya ***F7*** klavişini basmaq lazımdır. Bu seansda proqram sətirləri növbə ilə yerinə yetirilir və hər növbəti sətri yerinə yetirmək üçün ***F7*** klavişini basmaq lazımdır.

Proqramın yerinə yetirilməsi zamanı müşahidə (*Watch*) pəncərəsindən istifadə etmək faydalı ola bilər. *Debug* (*Sazlamaq*) menyusundan *Add Watch* (*Müşahidə üçün əlavə et*) əmrini icra etməklə və ya ***Ctrl+F7*** klavişlərini basmaqla bu pəncərəni ekranda göstərmək olar. Açılan pəncərədə, dəyişənin adını yazmaqla, onun aldığı cari qiyməti görmək olar. Müşahidə pəncərəsinə yeni dəyişənlər əlavə etmək üçün yenidən *Add Watch* əmrini icra etmək lazımdır.

Beləliklə, *Run/Trace into* (*Yerinə yetirmək/Sətirlərlə*) və *Debug/Add Watch* (*Sazlamaq/Müşahidə üçün əlavə et*)

əmrləri ilə düzgün işləməyən proqramlarda səhvləri aradan qaldırmaq asanlaşır və proqrama düzgün "diaqnoz" qoymaq mümkün olur.

1.4. Proqramın kompilyasiyası

Proqram sazlandıqdan sonra, onu kompilyasiya etmək lazımdır. Kompilyasiya nəticəsində proqramın mətni ikilik verilənlərdən və prosessorun təlimatlarından ibarət maşın kodlarına çevrilir. Kompüter yalnız maşın kodlarından ibarət proqramları icra edir. Proqram mətninin maşın kodlarına çevrilməsi prosesini *kompilyator* adlanan xüsusi proqram yerinə yetirir. Kompilyasiya zamanı `Uses` bölməsində göstərilmiş modullardakı alt proqramlar proqrama əlavə edilir, `Const` və `Var` bölmələrindəki dəyişən və sabitlər üçün yaddaş xanalarına adlar mənimsədilir və s.

Proqramı kompilyasiya etmək üçün *Compile* (*Kompilyasiya*) menyusundan *Compile* (*Kompilyasiya*) əmrini icra etmək və ya ***Alt+F9*** klavişlərini basmaq lazımdır. Proqram uğurla kompilyasiya olunduqdan sonra, ekrana *"Compile successful: press any key"* məlumatı çıxarılacaqdır. Bundan sonra, yeni bir fayl yaranacaqdır ki, bu faylın adı Sizin proqrama verdiyiniz adla eyni olacaq, lakin onun tipi ***.pas*** yox, ***.exe*** olacaqdır. Bu yeni *.exe* tipli faylda proqramın mətnini heç vaxt görə bilməyəcəksiniz və bu fayla düzəlişlər edə bilməyəcəksiniz. Çünki, bu fayl artıq *tam hazır proqram faylıdır* və Turbo Pascal kompilyatorunun mövcud olmasından asılı olmayaraq, o bütün kompüterlərdə yerinə yetirilə biləcəkdir.

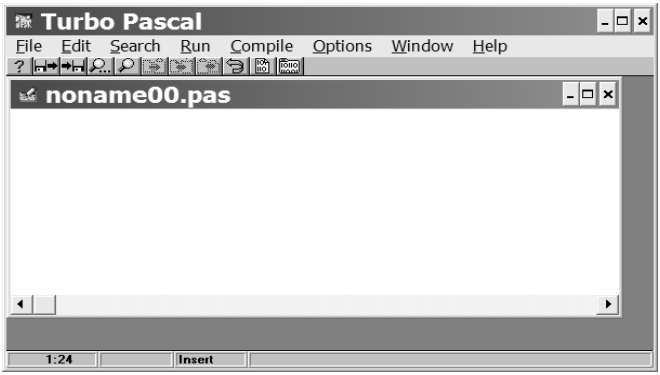
Proqramın yerinə yetirilməsini ***Ctrl+Break*** klavişlərini basmaqla dayandırmaq olar.

Qeyd. **Borland Pascal 7.0** inteqrallaşdırılmış mühitinin pəncərəsi Turbo Pascal 7.0 inteqrallaşdırılmış mühitinin pəncərəsi ilə tamamilə eynidir və yuxarıda nəzərdən

keçirdiyimiz bütün əmrlər Borland Pascal 7.0 inteqrallaşdırılmış mühitində eyni qayda ilə icra olunur.

1.5. Turbo Pascal for Windows 1.5 dilinin pəncərəsi

Turbo Pascal 7.0 və Borland Pascal 7.0 alqoritmik dilləri, əsasən, *Ms DOS* əməliyyat sistemində işləmək üçün nəzərdə tutulmuşdur. Bununla yanaşı, bu dillər həm də *Ms Windows* əməliyyat sistemində işləyə bilər. Lakin, *Ms DOS* əməliyyat sistemində bu dillər çox dəqiq işlədiyi halda, *Ms Windows* əməliyyat sistemində tam korrekt işləməyə bilər. Ona görə də, Pascal dilinin *Ms Windows* əməliyyat sistemində işləyə bilən **Turbo Pascal for Windows 1.5** (qısaca **– TPW 1.5**) – "*Turbo Pascal Windows üçün"* versiyası yaradılmışdır. TPW 1.5 dilinin inteqrallaşdırılmış mühitinin pəncərəsi şəkil 1.2 –



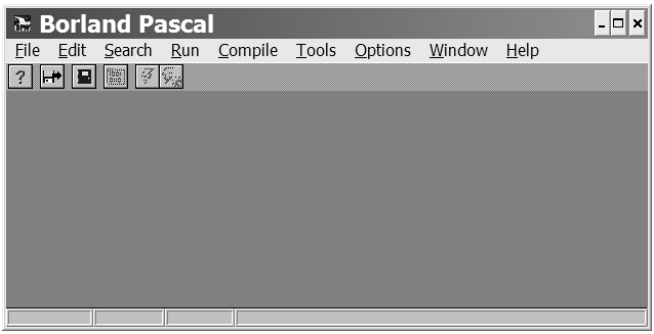
Şəkil 1.2. Turbo Pascal for Windows 1.5 (TPW 1.5) pəncərəsi

də göstərilmişdir. Yeni proqramın yaradılması, mövcud proqram faylının açılması, proqramın yadda saxlanması, sazlanması, yerinə yetirilməsi və kompilyasiyası və s. bu kimi əmrlər bu mühitdə də eyni qayda ilə yerinə yetirilir. Kəskin fərq ondan ibarətdir ki, bu mühitdə istifadəçi pəncərəsi ayrı bir pəncərə kimi açılır və o adi Windows pəncərələri kimi idarə olunur (zənnimcə, bu mühitin ən

böyük üstünlüyü hesab oluna bilər). TPW 1.5 dili ilə işlədikdə unutmayın ki, proqramda **`Uses Crt;`** əvəzinə **`Uses WinCrt;`** yazılmalıdır və `end.` operatorundan əvvəl `readln` prosedurunu yazmaq lazım deyil.

Turbo Pascal dilində qrafiklərin qurulması ilə əlaqədar yazılmış proqramlar TPW 1.5 mühitində işləməyəcəkdir.

Borland Pascal 7.0 dilinin də Windows versiyası hazırlanmışdır. Bu dil **Borland Pascal for Windows 7.0** (qısaca – **BPW 7.0**) adlanır və onun inteqrallaşdırılmış mühitinin pəncərəsi şəkil 1.3 – də göstərilmişdir. Şəkildən



Şəkil 1.3. Borland Pascal for Windows 7.0 pəncərəsi **(BPW 7.0)**

göründüyü kimi bu pəncərə TPW 1.5 pəncərəsindən çox az fərqlənir.

1.6. Delphi–də pascal–proqramın kompilyasiyası

Pascal dilində yazılmış proqramı Delphi mühitində də kompilyasiya etmək olar. Bunun üçün Delphi sistemini yüklədikdən sonra, *File*/*New* (*Fayl/Yeni fayl yaratmaq*) əmrini seçib, *Other* (*Digər fayllar*) əmri üzərində mausun sol düyməsini basmaq və açılan dialoq pəncərəsində *Console Application (Konsol əlavəsi)* piktoqramı üzərində mausun sol düyməsini yenidən iki dəfə basmaq lazımdır. Bu zaman kod

redaktoru pəncərəsində aşağıdakı proqram mətnindən ibarət karkas görəcəksiniz:

```
program Project2;

{$APPTYPE CONSOLE}

uses
    SysUtils;



Begin

  { TODO -oUser -cConsole Main : Insert
        code here }
  { Burada proqram sətirləri yazılır }

end.
```

Bu hazır karkasda Siz, istəsəniz, proqramın adını dəyişdirə bilərsiniz. `Uses` bölməsindən sonra `Var`, `Const`, `Type` bölmələrini əlavə edə bilərsiniz. `{$APPTYPE CONSOLE}` *direktivinə isə toxunmayın!* Nümunə üçün aşağıda $y=a+b$ ifadəsinin hesablanması proqramı yazılmışdır.

```
program delphi_pascal;

{$APPTYPE CONSOLE}

uses
 SysUtils;

var
   a,b,y : real;

begin
```

```
  Writeln('a,b dəyişənlərinin
         qiymətlərini daxil edin:');
  Readln(a,b);
  y:=a+b;
  Writeln(y);
  Readln;

end.
```

Proqramı kompilyasiya etmək üçün *Run/run* əmrini icra etmək və ya sadəcə olaraq ***F9*** klavişini basmaq lazımdır. Proqramın yadda saxlanması, mövcud proqramın çağrılması əməliyyatları *File* menyusunun əmrləri ilə yerinə yetirilir.

Turbo Pascal dilində modullarla və qrafiklərin qurulması ilə əlaqədar yazılmış proqramlar Delphi mühitində işləməyəcəkdir. Belə proqramların yaradılması kitabın ikinci hissəsində şərh olunmuşdur.

1.7. Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə edilən əmrlər

Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə edilən əmrlər cədvəl 1.1 – də göstərilmişdir.

Cədvəl 1.1. Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə olunan əmrlər

Əmrin adı	**Klavişlər**	**Əmrin yerinə yetirdiyi funksiya**
File/New		*Yeni proqram faylı yaratmaq*
File/Open...	**F3**	*Mövcud pascal–proqram faylını açmaq*
File/Save	**F2**	*Proqram mətnini yadda saxlamaq*
File/Save As...		*Proqramı yeni adla yadda saxlamaq*

File/Exit	**Alt+X**	*İnteqrallaşdırılmış mühitdən çıxmaq*
Edit/Cut	**Shift+Del**	*Seçilmiş proqram fraqmentini kəsib buferdə yerləşdirmək*
Edit/Copy	**Ctrl+Ins**	*Seçilmiş proqram fraqmentini buferə köçürmək*
Edit/Paste	**Shift+Ins**	*Buferdə olan proqram fraqmentini kursorla göstərilən mövqeyə yerləşdirmək*
Edit/Clear	**Ctrl+Del**	*Seçilmiş proqram fraqmentini pozmaq*
Compile/Make	**F9**	*Proqramı sazlamaq (səhvləri aşkar etmək)*
Run/Run	**Ctrl+F9**	*Proqramı yerinə yetirmək*
Compile/Compile	**Alt+F9**	*Proqramı kompilyasiya etmək*
Debug/User screen	**Alt+F5**	*İstifadəçi pəncərəsini ekranda göstərmək və ya gizlətmək*
Run/Trace into	**F7**	*Proqramı sətirlərlə yerinə yetirmək*
Run/Program Reset	**Ctrl+F2**	*Proqramın icrasından imtina etmək*
Debug/Add Watch	**Ctrl+F7**	*Dəyişənləri müşahidə pəncərəsinə əlavə etmək*
Ctrl+Break		*Proqramın icrasını dayandırmaq*
	F10	*Menyu sətrini aktivləşdirmək*

İ k i n c i f ə s i l

TURBO PASCAL DİLİNİN ƏLİFBASI VƏ STRUKTURU

Bu fəsildə, biz, Turbo Pascal dilinin əlifbası, lüğəti və əsas elementləri ilə tanış olacaq, proqramın strukturunu öyrənəcəyik. Dilin tərkibinə daxil olan işçi sözlərlə, dəyişən və sabitlərlə tanış olacaq, identifikatorlara qoyulan tələbatları, proqramın təsviretmə hissəsində yazılan operatorların strukturunu, prosedur və funksiyaların təsviri bölmələrini, şərhləri öyrənəcəyik və kompilyatorun direktivləri ilə tanış olacağıq.

2.1. Dilin əlifbası

Turbo Pascal dilinin əlifbasına aşağıdakı simvollar daxildir:

53 hərf – latın əlifbasının baş (**A – Z**) və kiçik (**a – z**) hərfləri və nəzərə çarpdırma işarəsi(**_**);

10 ərəb rəqəmləri: (**0 – 9**);

23 xüsusi simvol:

```
+  –  *  /  .  ,  :  ;  =  >  <  '
(  )  {  }  [  ]  #  $  ^  @ probel;
```

Xüsusi simvolların birləşməsindən aşağıdakı simvollar əmələ gəlir:

:= *–mənimsətmə;*
<> *–bərabər deyildir;*
.. *–qiymətlərin dəyişmə diapazonu;*
<= *–kiçik və ya bərabər;*
>= *–böyük və ya bərabər;*
{ } *–şərh simvolları;*
(* *və* ***)** *–***{** *və* **}** *fiqurlu mötərizələrin əvəzləyiciləri.*

2.1.1. Dilin lüğəti

İstənilən danışıq dili (Azərbaycan, rus, ingilis, fransız, alman və s.) simvollar, söz, söz birləşmələri və cümlələrdən ibarətdir. Proqramlaşdırma dillərində də buna analoji elementlər var. Bunlar simvollar, sözlər, ifadələr (söz birləşmələri) və operatorlardır (cümlələr).

Dilin simvollarından mətnlər əmələ gəlir. Hər hansı dilin əlifbası sonlu sayda simvollar çoxluğundan ibarətdir. Məsələn, danışıq dillərindən olan Azərbaycan dilinin əlifbası 32 hərfdən, rabitə texnikasında istifadə edilən Morze əlifbası 2 simvoldan ibarətdir. Azərbaycan dilində hər hansı cümlənin yazılışında Azərbaycan dilinin əlifbasının xüsusi simvolları olan durğu işarələrindən (nöqtə, vergül, nida işarəsi, sual işarəsi və s.), rəqəmlərdən, qısaldılmış sözlərdən (*ADR*, *BMT*, *ADNA* və s.) istifadə olunur. Bəzi texniki sahələrdə isə inteqral, cəm, hasil və s. kimi işarələrin hesabına simvolların sayı xeyli arta bilər.

İstənilən proqramlaşdırma dili də onun əlifbasını təşkil edən hərf və simvollardan əmələ gəlir. Hərflərin ayrılmaz ardıcıllığı sözlər əmələ gətirir və onlar bir–birindən ayırıcı ilə ayrılır. Sözlər arasında ayırıcı kimi probel, sətrin sonu simvolu, şərhlər, digər xüsusi simvollar və onların

birləşmələri istifadə oluna bilər. Sözlər özləri isə aşağıdakılara bölünür:

işçi sözlər və ya dilin söz ehtiyatları;
standart identifikatorlar;
istifadəçi identifikatorları.

İşçi sözlər və ya dilin söz ehtiyatları dilin əsas tərkib hissəsini təşkil edib, proqramda müəyyən məna və funksiyaları yerinə yetirir. Proqramda onların yazılışında istənilən təhrif kompilyator tərəfindən ciddi səhv kimi qəbul olunur. Proqramçı bu sözlərin mənasını da dəyişdirə bilməz. İşçi sözlərə misal olaraq `Unit`, `Goto`, `Begin`, `Uses`, `for` və s. göstərmək olar. Proqramda bu sözlər kompilyator tərəfindən ağ rəngli şriftlərlə yazılaraq digər sözlərdən fərqləndirilir.

Turbo Pascal dilindəki işçi sözlər aşağıdakılardır:

and	**goto**	**program**
array	**implementation**	**repeat**
asm	**if**	**record**
begin	**in**	**set**
case	**inherited**	**shl**
const	**inline**	**shr**
construct	**interface**	**string**
destructor	**label**	**then**
div	**library**	**to**
do	**mod**	**type**
downto	**nil**	**unit**
else	**not**	**until**
end	**object**	**uses**
exports	**of**	**var**
file	**or**	**while**
for	**packed**	**with**
function	**procedure**	**xor**

Standart identifikatorlar dilin əvvəlcədən müəyyən olunmuş aşağıdakı konstruksiyalarını işarə etmək üçündür:

verilənlərin tipləri;
sabitlər;
prosedur və funksiyalar.

Standart identifikatorlara misal olaraq `Pi`, `Sin`, `Integer`, `Delete`, `Insert` və s. göstərmək olar.

İstifadəçi identifikatorları nişanların, sabitlərin, dəyişənlərin, prosedur və funksiyaların adlarını, verilənlərin tiplərini işarə etmək üçün tətbiq olunur. Bu identifikatorlar proqramçılar tərəfindən müəyyən edilir və onlar aşağıdakı tələbatlara cavab verməlidir:

- identifikator hökmən latın hərfi ilə başlamaqla hərf və rəqəmlərdən ibarət olmalıdır;
- identifikatorlarda baş və kiçik hərflərdən istifadə oluna bilər, lakin kompilyator onları bir–birindən fərqləndirmir;
- nişanlar 0 – 9999 diapazonunda dəyişən işarəsiz tam ədədlərdən və ya latın hərfi ilə başlamaqla hərf və rəqəmlərdən tərtib edilir;
- proqramda iki identifikator arasında ən azı bir ayırıcı yazılmalıdır. Əgər identifikator söz birləşmələrindən ibarət olarsa, sözləri baş hərflərlə ayırmaq daha məqsədəuyğundur, məsələn, `ListBox`, `btnOpen`, `OnClick`, `LabelColor`, `ModalResult` və s.

İstənilən proqramın əsas elementləri dəyişənlər, sabitlər və operatorlardır.

Dəyişənlər proqramın yerinə yetirilməsi prosesində müxtəlif qiymətlər alan kəmiyyətlərə deyilir və onlar dəyişənlərin elan edilməsi bölməsində elan edilir. Dəyişənlərdən fərqli olaraq, *sabitlər* proqram boyu öz qiymətlərini dəyişmir və onların qiymətləri sabitlərin elan edilməsi bölməsində təyin olunur. Sabit və dəyişənlərə onların adları ilə müraciət olunur.

Ədədlər. Turbo Pascal dilində tam onluq ədəd, tam onaltılıq ədəd və həqiqi onluq ədədlərdən istifadə olunur. Həqiqi ədədlər iki müxtəlif yazılış formasında təsvir olunur: adi (qeyd olunmuş vergüllü) və tərtibli (sürüşkən vergüllü).

Qeyd olunmuş vergüllü formada həqiqi ədədlər *tam* və *kəsr* hissələrdən ibarət olur və bu hissələr arasında nöqtə işarəsi qoyulur. Məsələn, *7.81, –3.456, .378, 456., –.565, 65.0* və s. Bu yazılışlarda onluq nöqtə öz mövqeyini dəyişmədiyi üçün belə ədədlərə qeyd olunmuş vergüllü ədədlər deyilir.

Sürüşkən vergüllü formada isə həqiqi ədədlər əsası 10 olan qüvvətüstlü ədəd kimi təsvir olunur. 10 əsasının əvəzinə *E* işarəsindən istifadə olunur. Məsələn, *–179234.65E–05, 51.91e+05, 0.29678e+03*. Adi halda bu ədədlər uyğun olaraq *–1.7923465, 5191000* və *296.78* ədədlərinə bərabərdir. Sürüşkən vergüllü formada onluq nöqtə öz mövqeyini dəyişdiyi üçün bu ədədlərə sürüşkən vergüllü ədədlər deyilir. Məsələn, *625.45* ədədi sürüşkən vergüllü formada aşağıdakı formalarda yazıla bilər: *0.62545E+03, 62545.0E–02, 6.2545E+02* və s.

2.2. Proqramın strukturu

Proqramın mətni istənilən mövqedən başlayan sətirlərdən ibarətdir. Turbo Pascal dilində proqram sərlövhə və blokdan ibarət olur.

Sərlövhə proqramın başlanğıcında yerləşir, başqa sözlə, proqram sərlövhə sətri ilə başlayır və bu sətirdə

`Program` *name*;

yazılır. Burada, *name* – proqramın adıdır və bu ad proqramçı tərəfindən verilir.

Blok isə iki hissədən – təsviredici və icraedici hissələrdən ibarət olur.

Təsviredici hissədə proqramın elementləri təsvir olunur və ümumi halda bu hissə aşağıdakı bölmələrdən ibarət olur:

- *modulların qoşulması;*
- *nişanların elan edilməsi;*
- *sabitlərin elan edilməsi;*
- *verilənlərin tiplərinin təsvir edilməsi;*
- *dəyişənlərin elan edilməsi;*
- *prosedur və funksiyaların təsvir edilməsi.*

Hər bir bölmənin sonunda nöqtəli–vergül işarəsi qoyulmalıdır.

İcraedici hissədə, son nəticələr almaq üçün, müəyyən əməliyyatları yerinə yetirən operatorlar ardıcıllığı yazılır. Bu hissəyə *operatorlar bölməsi* də deyilir. Bölmə **`begin`** operatoru ilə başlayır və **`end.`** operatoru ilə qurtarır. Hər bir operatorun sonunda nöqtəli–vergül işarəsi qoyulmalıdır. Yalnız `end`. operatorundan əvvəlki operatorda nöqtəli–vergül işarəsini yazmamaq olar. `End` operatorundan sonra hökmən nöqtə işarəsi qoyulmalıdır: bu proqramın sonu əlamətidir. Proqramın strukturu şəkil 2.1 – də göstərilmişdir.

Ümumi halda, proqramın şəkil 2.1 – də göstərilən strukturunu operatorlar vasitəsilə aşağıdakı kimi göstərmək olar:

`Program` *proqramın adı*;

`Uses` *modulların siyahısı*;

`Label` *nişanların siyahısı*;

`Const` *sabitlərin siyahısı*;

`Type` *tiplərin təsviri*;

`Var` *dəyişənlərin elan edilməsi*;

*prosedurların təsviri***;**
*funksiyaların təsviri***;**

Begin

*operatorlar***;**

End.

Program ***name*****;**
Təsviredici hissə
Begin
İcraedici hissə
End.

Şəkil 2.1. Turbo Pascal dilinin strukturu

Konkret proqramda bu təsviredici və elanedici bölmələrdən hər hansı biri olmaya da bilər. Modulların qoşulması bölməsi müstəsna olmaqla, yerdə qalan bölmələr proqramda istənilən sayda, istənilən yerdə təkrar oluna bilər. Elanetmə və təsviretmələr elementlər istifadə olunana qədər yerinə yetirilməlidir. Bir sətirdə bir və ya bir neçə operator yazıla bilər.

Bu bölmələrə ayrı–ayrılıqda baxaq.

Modulların qoşulması bölməsi **Uses** işçi sözü ilə başlayır və onun ümumi forması belədir:

Uses *1–ci ad, 2–ci ad, . . . , n–ci ad***;**

Misal.

```
Uses Crt,WinCrt,Dos,
     Graph,Messages,Windows;
```

Bu bölmədə nəinki standart kitabxana modulları, habelə proqramçının yaratdığı istifadəçi modulları da (əgər varsa) göstərilməlidir.

Nişanların elan edilməsi bölməsi **`Label`** işçi sözü ilə başlayır və bu sözdən sonra proqramda istifadə olunan nişanların adları, aralarında vergül işarəsi qoymaqla, sadalanmalıdır. Nişan operatora verilən addır. Turbo Pascal dilində nişanların ədəd və simvol növü mövcuddur. Ədədi nişanlar 0 ilə 9999 aralığında tam onluq ədədlərlə ifadə olunur. Nişan kimi identifikatorlardan istifadə olunduqda identifikatorun yazılış qaydalarına riayət edilməlidir. Nişan operatordan iki nöqtə işarəsi (**:**) ilə ayrılır.

Bu bölmənin ümumi forması belədir:

`Label` *1–ci nişan, 2–ci nişan, . . . , n–ci nişan***;**

Misal.

```
Label M8,K3,n7,10,30;
```

Sabitlərin elan edilməsi bölməsi **`Const`** sözü ilə başlayır və bu bölmədə sabitlər identifikatorlarına onların aldıqları ədədi qiymətlər mənimsədilir.

Bu bölmənin ümumi forması belədir:

`Const`
1–ci identifikator **=** *qiymət***;**
2–ci identifikator **=** *qiymət***;**
. . .
n–ci identifikator **=** *qiymət***;**

İdentifikatorla onun aldığı ədədi qiymət arasında bərabərlik işarəsi qoyulur. Bir sətirdə, aralarında nöqtəli–vergül işarəsi yazılmaqla, bir neçə identifikator yazıla bilər. Hər sətrin sonunda da nöqtəli–vergül işarəsi qoyulmalıdır. İdentifikatorun tipi, avtomatik olaraq, sabitin tipi ilə eyni qəbul edilir.

Misal.

```
Const a=8.2; b= -1.25;
      c=998;
      line1='BAKI'; k='27';
```

Tiplərin təsviri bölməsi **`Type`** işçi sözü ilə başlayır və bu bölmədə verilənlərin istifadəçinin özü tərəfindən müəyyən etdiyi tiplər göstərilir. Bu bölmə məcburi deyildir. Tiplər dəyişənlərin elan edilməsi bölməsində qeyri–aşkar formada da təsvir oluna bilər. `Type` sözündən sonra tiplərin adları və onların təsvirləri yerləşir, onlar arasında bərabərlik işarəsi yazılır. Ad və təsvirdən sonra nöqtəli–vergül işarəsi yazılır. Bölmənin ümumi forması belədir:

`Type`
1–ci tip = *tipin təsviri*;
2–ci tip = *tipin təsviri*;
. . .
n–ci tip = *tipin təsviri*;

Misal.

```
Type
  B=(BAKI,London,Moskva,Paris);
  D=(May,Iyun,Iyul,Avqust);
  index=1..100;
  char2='a'..'z';
  massiv=array[1..20] of word;
  mont=1..12;
  list=array[1..10] of massiv;
```

Bu misala görə, proqramçı özünün təyin etdiyi aşağıdakı tipləri daxil edir: `B` və `D` – sadalanan tiplər, `index` – *1*–dən *100*–ə qədər tam ədədlər intervalı, `char2` – *a*–dan *z*–ə qədər 26 simvol, `massiv` – `word` tipli 20 elementdən ibarət massiv, `mont` – *1*–dən *12*–yə qədər tam ədədlər intervalı, `list` – 20∗10 sayda elementdən ibarət ikiölçülü massiv.

Sonralar bu tiplər hər hansı dəyişənlərin təsviri üçün istifadə oluna bilər.

Dəyişənlərin elan edilməsi bölməsi **`Var`** işçi sözü ilə başlayır. Bu bölmədə proqramda istifadə olunan bütün dəyişənlər, onların tipləri göstərilməklə, hökmən sadalanmalıdır. `Var` sözündən sonra aralarında nöqtəli–vergül işarəsi qoyulmaqla dəyişənlərin adları, iki nöqtə (**:**) işarəsi qoyulmaqla onların tipləri göstərilir. Bölmənin ümumi forması belədir:

`Var`
1–ci identifikator **:** *tip***;**
2–ci identifikator **:** *tip***;**
`...`
n–ci identifikator **:** *tip***;**

Misal.

```
Var
  l1,l2:word;
  s,z,r:integer;
  rr:extended;
  x,y:real;
```

İkinci misala baxaq. Bu misalda biz dəyişənlərin tiplərini `Type` bölməsində təyin etdiyimiz tiplərlə müəyyən edəcəyik.

Misal.

```
Var
  city:B;
  month:D;
  s1,s2:index;
  dbase:massiv;
  matris:list;
  calendar:mont;
 matris:list;
  calendar:mont;
```

Burada, `city` dəyişəni `B` tipli elan olunduğu üçün o, `BAKI`, `London`, `Moskva`, `Paris` qiymətlərini, `month` dəyişəni isə `B` tipli elan olunduğu üçün `May`, `Iyun`, `Iyul` və `Avqust` qiymətlərini alır. Yerdə qalan dəyişənlərin tipləri də analoji qayda ilə müəyyən olunur.

Dəyişənlərin elan edilməsi dəyişənlərin istifadə edilməsindən əvvəl yerinə yetirilməlidir. Qeyd edək ki, elanetmə ilə təsviretmə bir–birindən fərqlənir. Fərq ondan ibarətdir ki, elanetmə zamanı dəyişənlər üçün əsas yaddaşda yer ayrıldığı halda, təsviretmə zamanı belə yaddaş tələb olunmur. Lakin hər iki halda dəyişənlərə heç bir qiymət mənimsədilmir: bu qiymətlər dəyişənlər istifadə olunana qədər proqramçılar tərəfindən onlara mənimsədilir.

Prosedur və funksiyaların təsviri bölməsində proqramda istifadə olunan alt proqramların təsviri (əgər alt proqram varsa) verilməlidir. Alt proqramlar prosedur və funksiya tipli olur. Alt proqramlar təsvir olunduqda onların adları və parametrlərinin siyahısı göstərilməlidir.

Misal.

```
procedure binary(var x:longİnt);
procedure hesab(x0,y0,x1,y1,n:byte);
Procedure POrta(Bp:Massiv;L:integer;
          var k:integer;Ad:string);
Procedure Fact;
Funstion n(x:integer): integer;
Funstion sinh(x:real):real;
```

Operatorlar bölməsi **`begin`** operatoru ilə başlayır. Sonra isə proqramı icra edən operatorlar ardıcıllığı yazılır. Hər operatordan sonra nöqtəli–vergül işarəsi qoyulur. Bu bölmədə dilin istənilən operatoru istifadə oluna bilər. Bölmənin sonunda **`end`** operatoru yazılır və nöqtə qoyulur. Nöqtə işarəsi proqramın sonu əlamətidir. Bölmənin ümumi forması belədir:

```
Begin
  1–ci operator;
  2–ci operator;
  ...
  n–ci operator;
end.
```

2.3. Şərhlər

Şərhlər – proqramın daha yaxşı başa düşülməsi üçün, proqramın istənilən yerində yazıla bilən, izahedici mətndən ibarətdir. Şərhlər bir və ya bir neçə sətirdə yazıla bilər. Proqramda şərhləri **{** və **}** mötərizələri daxilində və ya onların ekvivalenti olan **(*** və ***)** işarə birləşmələri daxilində yazmaq lazımdır. Şərhlər proqramın yerinə yetirilməsinə heç bir təsir göstərmir və onun mətni kompilyator tərəfindən emal olunmur. Düşünülərək yazılan şərhlər proqramın başa düşülməsini asanlaşdırır. Şərhlərin başqa qeyri–standart tətbiqi də vardır. Belə ki, proqramın kompilyasiyası zamanı bəzi operatorları müvəqqəti olaraq proqramdan çıxarmaq lazım gələrsə, onları şərhlə əvəz etmək olar. Bu zaman həmin operatorlar proqramın mətnində icra olunmayan sətirlər kimi saxlanacaqdır.

Misal.

```
{ Biz Delphi dilini
öyrənirik. Delphi obyektyönlü
proqramlaşdırma
dilidir }


(* Laboratoriya işi №1
   Nyuton üsulu        *)

{sum:=sum+x; }


{ begin
```

```
for k:=1 to 10 do
a[k]:=x*k;
end; }
```

2.4. Kompilyatorun direktivləri

Direktivlər kompilyatorun iş rejimlərini idarə etmək üçün proqramın mətnində yazılır. Direktivlər də **{** və **}** mötərizələri daxilində yazılır, lakin onlar şərhlər deyildir. Direktivlərin qarşısında, fiqurlu mötərizə daxilində, **$** simvolu yazılır. Beləliklə, **{$...}** yazılışı həmişə kompilyatorun direktivini müəyyən edəcəkdir. Kompilyatorun direktivi ilə, məsələn, sətir tipli dəyişənlərin interpretasiya üsulu (**{$H+}**) verilə bilər.

Turbo Pascal dilində həqiqi tip ədədlər üzərində əməliyyatların yerinə yetirilməsi üçün kodun generasiyasının iki üsulu var:

- 80×87 soprosessoru olduqda (aparat üsulu);
- 80×87 soprosessoru olmadıqda (proqram üsulu) .

Bu üsulların seçilməsi kompilyatorun **{$N}** və **{$E}** direktivləri ilə həyata keçirilir.

Üçüncü fəsil

VERİLƏNLƏRİN TİPLƏRİ

Bu fəsildə Turbo Pascal dilinin tərkibinə daxil olan verilənlərin təsnifatına baxılacaq, sıralı və həqiqi tip verilənlər öyrəniləcəkdir. Tamədədli və həqiqi tiplərin növləri, onların dəyişmə diapazonları və yaddaşda təsviri şərh olunacaq, simvol və məntiqi tiplər izah olunacaqdır. Sadalanan və interval tiplərin proqramda təsviri və bu tiplərin strukturu göstəriləcəkdir. Verilənlərin struktur tiplərinə daxil olan massivlər, sətirlər, çoxluqlar, yazılar və faylların mahiyyəti, onların strukturu, və proqramda təsvirolunma qaydaları ətraflı izah olunacaq və bu izahatlar çoxlu misallar üzərində nümayiş etdiriləcəkdir.

3.1. Tiplərin təsnifatı

İstənilən proqramın əsas elementləri dəyişənlər, sabitlər və operatorlardır. Bu bölmədə biz dəyişən və sabitlərin tiplərini öyrənəcəyik.

Dəyişənlər proqramın yerinə yetirilməsi prosesində müxtəlif qiymətlər alan kəmiyyətlərə deyilir və onlar dəyişənlərin elan edilməsi bölməsində elan edilir. Dəyişənlərdən fərqli olaraq, *sabitlər* proqram boyu öz qiymətlərini dəyişmir və onların qiymətləri sabitlərin elan edilməsi bölməsində təyin olunur. Sabit və dəyişənlərə onların adları ilə müraciət olunur.

Turbo Pascal dilində verilənlərin çoxlu sayda tipləri mövcuddur: bu, şübhəsiz, dilin üstün cəhətidir. Verilənlərin

tiplərini öyrənmək üçün onları qruplaşdırmaq məqsədəuyğundur. Hər şeydən əvvəl, verilənlərin tipləri *sadə* və *struktur* tipli olur. *Sadə* tiplər öz növbəsində *sıralı* və *həqiqi* tiplərə bölünür. Turbo Pascal dilində müəyyən edilmiş tiplərin təsnifatı cədvəl 3.1 – də göstərilmişdir.

Cədvəl 3.1. Turbo Pascal dilində verilənlərin tiplərinn təsnifatı

Qrup	Alt qrup	Adı	İdentifikatoru
Sadə	***Sıralı***	***Qısa tamədədli***	`ShortInt`
		Bayt	`Byte`
		Söz	`Word`
		Tamədədli	`Integer`
		Uzun tamədədli	`LongInt`
		Simvol	`Char`
		Bul	`Boolean`
	Həqiqi	***Həqiqi***	`Real`
		Birqat dəqiqlikli	`Single`
		İkiqat dəqiqlikli	`Double`
		Yüksək dəqiqlikli	`Extended`
		Mürəkkəb	`Comp`
Sətir			`String`
Struktur		***Massiv***	`Array`
		Çoxluq	`Set`
		Fayl	`File`
		Yazı	`Record`
Göstərici			`Pointer`
Prosedur		***Prosedur***	`Procedure`
		Funksiya	`Function`
Obyekt			`Object`

Tiplər ona görə *sıralı* adlanır ki, əvvəla, dəyişənlərin aldığı qiymətlər sonlu sayda elementlərdən ibarət olur, digər tərəfdən, hər bir elementdən əvvəlki və sonrakı qiymətlər mövcud olur. Başqa sözlə, qonşu qiymətlər bir–birindən bir vahid fərqlənir. Belə ki, misal üçün, 15 tam ədədindən bir vahid çıxdıqda 14, ona bir vahid əlavə etdikdə isə 16 alınacaqdır. Başqa sözlə, sıralı tip verilənləri nömrələmək mümkündür.

Sıralı tiplərdən fərqli olaraq, *həqiqi* tiplər tam və kəsr hissələrdən ibarət olur, hətta ən məhdud diapazonda yerləşən ədədləri belə nömrələmək mümkün olmur.

3.2. Verilənlərin sadə tipləri

Bəzi sadə tiplər fiziki (əsas) və ümumi tiplərə bölünür. *Fiziki tiplərin* təməli dil yaradıldıqda qoyulur və kompüterin xüsusiyyətlərindən asılı olmur. *Ümumi tiplər* fiziki tiplərdən hər hansı birinə uyğun gəlir və kompilyator bu tipləri istifadə etdikdə daha səmərəli kod yaratdığı üçün onlara daha çox üstünlük verilir.

Bəzi sıralı tipləri proqramçı özü də yarada bilər, ona görə də belə tiplərə istifadəçi tipləri deyilir. İstifadəçi tiplərinə *sadalanan* və *interval* tiplər aiddir.

3.2.1. Tamədədli tiplər

Cədvəl 3.2 – də sadə sıralı tiplərin ala biləcəyi mümkün qiymətlər diapazonu verilmişdir.

Cədvəl 3.2. Tamədədli tiplər

Tipin adı (*işarəsi*)	**Dəyişmə diapazonu**	**Yaddaşda təsviri** (*işarə ilə birlikdə baytlarla*)
ShortInt	(–128) – 127	*1*
Integer	(–32 768) – 32 767	*2*
LongInt	(–2 147 483 648) – 2 147 483 647	*4*
Byte	0 – 255	*1*
Word	0 – 65 535	*2*

Tam ədədlərin qarşısında "+" və "–" işarələri yazıla bilər. Onlar onluq və onaltılıq say sistemlərində təsvir oluna bilər. Tam ədədləri onaltılıq say sistemində təsvir etmək üçün

ədədin qarşısında **$** işarəsi qoyulmalıdır. Belə ədədlər `$00000000 – $FFFFFFFF` diapazonunda yerləşməlidir.

Misal.

```
Var i,j: byte;
    n: word;
    x6,ss: integer;
```

3.2.2. Həqiqi tiplər

Həqiqi tiplər və onların ala biləcəyi mümkün qiymətlər diapazonu cədvəl 3.3 – də göstərilmişdir:

Cədvəl 3.3. Həqiqi tiplər

Tipin adı (*işarəsi*)	**Dəyişmə diapazonu**	**Yaddaşda təsviri** (*baytlarla*)
Real	$(-1{,}7\cdot10^{38}) – (-2{,}9\cdot10^{-39})$, $2{,}9\cdot10^{-39} – 1{,}7\cdot10^{38}$	*6*
Single	$(-3{,}4\cdot10^{38}) – (-1{,}5\cdot10^{-45})$, $1{,}5\cdot10^{-45} – 3{,}4\cdot10^{38}$	*4*
Double	$(-1{,}7\cdot10^{308}) – (-5\cdot10^{-324})$, $5\cdot10^{-324} – 1{,}7\cdot10^{308}$	*8*
Extended	$(-1{,}1\cdot10^{4932}) – (-1{,}9\cdot10^{-4951})$, $1{,}9\cdot10^{-4951} – 1{,}1\cdot10^{4932}$	*10*
Comp	$(-2^{63}+1) – (2^{63}-1)$	*8*

Həqiqi ədədlər, bütün dillərdə olduğu kimi, burada da qeyd olunmuş və sürüşkən vergüllü formalarda təsvir olunur. `Comp` tipi həqiqi tipə aid olsa da, $(-2^{63}+1) – (2^{63}-1)$ aralığında tam ədədləri təsvir edir. Bu tipli dəyişənə həqiqi tipli qiymətlər mənimsətdikdə o, avtomatik olaraq, yaxın tam ədədə qədər yuvarlaqlaşdırılır.

Misal.

```
Var a,b: real;
    d: double; x,s: extended;
```

3.2.3. Simvol tiplər

Simvol tipli dəyişən və sabitlər **ASCII** (*American Standart Code for Information Interchange*) kodunun simvollar çoxluğundan yalnız bir qiymət (işarə, hərf, rəqəm) ala bilər. Bu tip dəyişənlər **char** işçi sözü ilə müəyyənləşdirilir:

```
Var a,b,c: char;
```

Simvol tipli dəyişənlər sıralı tiplərə aid olmaqla, yalnız bir simvoldan ibarət olur və proqram daxilində onlara qiymətlər mənimsətmək olar:

```
a:='t';
b:='$';
c:='9';
```

Kompüterdə hər bir simvola 0 – 255 aralığından bir tam ədəd uyğun gəlir. Hər hansı simvolun kodunun qiymətini **Ord** funksiyasının köməyi ilə almaq mümkündür. Əks əməliyyat isə **Chr** funksiyası ilə yerinə yetirilir.

Misal.

```
Ord('A') = 65;
Ord('a') = 97;
Chr(65)= 'A';
Chr(100)= 'd';
```

Chr funksiyasını **#** işarəsi ilə də əvəz etmək olar:

```
#65='A';
#97='a'.
```

3.2.4. Məntiqi tiplər

Turbo Pascal dilində bir neçə məntiqi tipin olmasına baxmayaraq, proqramda **Boolean** tipini istifadə etmək

məqsədəuyğundur. Bu tip verilənlər yalnız iki qiymət ala bilər: *True* (Doğru) və *False* (Yalan). Məntiqi tiplər də sıralı tiplərə aiddir.

Misal.

```
Var a1,b2: boolean;
    d7,s44: boolean;
```

3.3. Sadalanan tiplər

Sadalanan tiplərdə verilənlərin qiymətləri birbaşa sadalanır. Qiymətlər bir–birindən vergül işarəsi ilə ayrılmaqla mötərizə daxilində yazılır. Sadalanan tiplərin ümumi forması belədir:

`Type` *tipin adı* **=** **(***1–ci ad, 2–ci ad, . . . , n–ci ad***);**

Misal.

```
Type  Color=(red,orange,yellow,blue);
Var   c1,c2: color;
      heyvanlar:(Fil,At,Aslan);
```

Bu misalda, `Color` tipi aşkar təsvir olunmuş və onun üçün qiymətlər kimi rənglər müəyyənləşdirilmişdir. `c1`, `c2` dəyişənləri bu sadalanan rənglərdən birini ala bilər: onlara digər qiymətlər vermək olmaz. İkinci tip anonim müəyyənləşdirilmişdir (adı yoxdur) və `heyvanlar` dəyişəni `Fil`, `At` və `Aslan` qiymətlərindən birini ala bilər. Qeyd edək ki, sadalanan tiplər sıralı tiplərə aiddir. Belə ki, `red`, `orange`, `yellow` və `blue` qiymətlərinin sıra nömrələri uyğun olaraq *0*, *1*, *2* və *3*–dür.

Sadalama proqramçıya verilənlərin yeni tiplərini təsvir etməyə imkan verir.

Sadalanan tipin bütün elementlərinin identifikatorları sabit kimi interpretasiya olunur. Bu identifikatorlar sətir sabitləri olmadığı üçün dırnaq işarəsi daxilində yazılmır. Qeyd etmək lazımdır ki, eyni identifikatorun müxtəlif

tiplərdə təsviri səhvdir. Məsələn:

```
program tekrar;
type
  Hafta_1=(Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  Hafta_2=(Mon,Tue,Wed,Thu,Fri);
begin
...
end.
```

Bu proqramı icra etsək, kompilyator *"Error: Duplicate identifier* (`Mon`)*"* məlumatı verəcəkdir.

Proqramda sadalanan tip dəyişəndən istifadə etdikdə aşağıdakıları nəzərə almaq lazımdır:

- sadalanan tiplər sıra tipinə aiddir;
- `Read`, `Readln`, `Write` və `Writeln` prosedurlarında arqument kimi sadalanan tipin qiymətlərindən istifadə etmək olmaz;
- sadalanan tiplərdə yalnız münasibət əməliyyatlarından istifadə etmək olar;
- sadalanan tip dəyişənlərdən mənimsətmə operatorlarında, massivin indekslərində və `for` operatorunun sərhədlərində istifadə etməyə icazə verilir.

3.4. İnterval tiplər

Dəyişənlərin tipləri interval tipləri ilə təsvir olunduqda dəyişmə intervalından istifadə olunur. Belə ki, bu dəyişmə intervalı dəyişənin sərhəd qiymətlərini göstərən iki sabitlə müəyyən edilir. İnterval tiplər yalnız sıralı tip dəyişənlərə verilə bilər, başqa sözlə, sərhəd qiymətləri həqiqi tiplər istisna olmaqla, istənilən sadə tiplər ola bilər. Lakin hər iki sabit eyni tipli olmalıdır. İnterval tiplərin ümumi forması belədir:

`Type` *tipin adı* **=** *1–ci sabit* **. .** *2–ci sabit***;**

Misal.

```
Const m=10; n=100;
Type otrezok=1..25;
     sem=m..n;
     simvol='a'..'z';

Var a,b:otrezok;
     a1,a2:sem;
     bukva:simvol;
```

`a` və `b` dəyişənləri `otrezok` tipli elan olunur və onlar 1–dən 25–ə qədər diapazonda qiymətlər ala bilər; `a1` və `a2` dəyişənləri `sem` tipli elan olunur və 10–dan 100–ə qədər qiymətlər ala bilər; `bukva` dəyişəni isə `simvol` tipli elan olunmaqla latın hərflərini ala bilər.

Misaldan və izahatdan göründüyü kimi, interval tiplər də sıralı tipə aiddir. Proqramın icrası zamanı interval tipli dəyişənlərin qiymətləri göstərilən sərhəd qiymətlərindən kənara çıxarsa, bu barədə məlumat verilməyəcək və onun qiyməti səhv olacaqdır.

3.5. Verilənlərin struktur tipləri

Cədvəl 3.1 – də göstərildiyi kimi, struktur tiplərə aşağıdakılar aiddir:

- *massivlər;*
- *sətirlər;*
- *çoxluqlar;*
- *yazılar;*
- *fayllar.*

Bu tiplərlə tanış olaq.

3.5.1. Massivlər

Massiv eyniadlı və eyni tipli indeksli elementlərin yığımına deyilir. Massivin elementləri, struktur tip də daxil olmaqla, istənilən tipli ola bilər, lakin eyni bir massivin elementləri eyni tipli olmalıdır. Massivlər, özlərinin adları və indeksləri ilə müəyyən olunur. Massivin indeksi onun elementlərinin sıra nömrəsini göstərir. İndekslərin sayı isə massivin ölçüsünü müəyyən edir. Əgər bir indeks yazılarsa, massiv birölçülü, iki indeks yazılarsa ikiölçülü və s. olur. Birölçülü massivlər riyaziyyatda vektorlara, ikiölçülü massivlər isə matrislərə uyğun gəlir. Çoxölçülü paralelepipedləri massivlərin həndəsi obrazı hesab etmək olar. Massivin indeksləri sıralı tip olmalıdır. Eyni bir massivin müxtəlif indeksləri müxtəlif tip ola bilər. Massivlərin indeksləri daha çox tamədədli tipli olur. İndekslər dəyişən və ifadələr də ola bilər. Massivə müraciət etmək üçün onun adını və kvadrat mötərizə daxilində yazılmış indekslərini göstərmək lazımdır, məsələn, `a[5,8]`, `s[16]`, `x[i,j]`, `b[m+n]` və s.

Massivlərin ölçüləri elanetmə bölməsində əvvəlcədən müəyyənləşdirilir. Proqramın yerinə yetirildiyi müddətdə onların ölçüləri dəyişmir. Belə massivlərin ümumi təsvir forması aşağıdakı kimidir:

Massivin adı **: Array [***indekslər***] of** *elementlərin tipi***;**

Burada, *indekslər* interval tip ilə göstərilir.

Misal.

```
Const max=1000;
Type a=array[1..5,1..8] of integer;
Var x,y:a;
   b:array[1..50] of real;
   c:array[1..40] of char;
   d:array['a'..'z'] of integer;
   z:array[1..max] of boolean;
```

`x` və `y` dəyişənləri 40 elementdən – 5 sətir və 8 sütundan ibarət ikiölçülü massiv olur; massivin elementləri `integer` – tam tiplidir. `b` dəyişəni 50 həqiqi tipli elementdən, `c` dəyişəni 40 simvol tipli elementdən, `d` dəyişəni 26 tam ədədlərdən, `z` dəyişəni isə 1000 məntiqi tipli elementlərdən ibarət massivlər kimi təyin olunur.

Massivlərin elementlərinə `const` operatoru vasitəsilə də qiymətlər vermək olar. Bu halda massivin elementləri adi mötərizə daxilində, bir–birindən vergüllə ayrılmaqla verilir. Çoxölçülü massivlərdə isə xarici adi mötərizə sol indeksə, daxili adi mötərizə isə növbəti indeksə və s. aid olur.

Misal.

• *həqiqi ədədlərdən ibarət birölçülü massiv*

```
const
    vektor : array[1..7] of real=(0.25,
    3.21,6.37,9.91,71.06,67.9,37.6);
```

• *tam ədədlərdən ibarət ikiölçülü massiv*

```
const
    m2 : array[1..3,1..4] of integer=
    ((1,2,3,4),(5,6,7,8),(9,10,11,12));
```

• *tam ədədlərdən ibarət üçölçülü massiv*

```
Const
  m3 : array[1..4, 1..3, 1..2] of Byte=
  (((1,2),(3,4),(5,6)),((7,8),(9,10),(11,12)),
  ((13,14),(15,16),(17,18)),((19,20),(21,22),
  (23,24)));
```

• *simvollardan ibarət birölçülü massiv*

```
const
    CharVect1 : array[1..6] ofchar=
    ('P','A','S','C','A','L');
```

və ya

```
    CharVect2 : array [1..6] of char='PASCAL'.
```

Bu misallardakı ikiölçülü `m2` və üçölçülü `m3` massivlərinin elementlərinə qiymətlərin mənimsədilməsinin interpretasiyası şəkil 3.1 – də göstərilmişdir.

```
m2(1,1)= 1  m2(1,2)= 2  m2(1,3)= 3  m2(1,4)= 4
m2(2,1)= 5  m2(2,2)= 6  m2(2,3)= 7  m2(2,4)= 8
m2(3,1)= 9  m2(3,2)=10  m2(3,3)=11  m2(3,4)=12

m3(1,1,1)= 1 m3(1,1,2)= 2 m3(1,2,1)= 3 m3(1,2,2)= 4 m3(1,3,1)= 5 m3(1,3,2)= 6
m3(2,1,1)= 7 m3(2,1,2)= 8 m3(2,2,1)= 9 m3(2,2,2)=10 m3(2,3,1)=11 m3(2,3,2)=12
m3(3,1,1)=13 m3(3,1,2)=14 m3(3,2,1)=15 m3(3,2,2)=16 m3(3,3,1)=17 m3(3,3,2)=18
m3(4,1,1)=19 m3(4,1,2)=20 m3(4,2,1)=21 m3(4,2,2)=22 m3(4,3,1)=23 m3(4,3,2)=24
```

Şəkil 3.1. Massivlərin elementlərinə qiymətlərin mənimsədilməsi

Massivləri tətbiq etdikdə aşağıdakı məhdudiyyətlərə əməl etmək lazımdır.

Massivlərin indeksləri, `LongInt` tipindən başqa, istənilən sıralı tip ola bilər. Massivin elementlərinin sayı

$$n = \frac{65520}{p}$$

düsturu ilə təyin olunur. Burada p – massivin elementlərinin baytlarla ölçüsüdür. $p=1$ olduqda massivin elementlərinin sayı 65520 olacaqdır. Bu düstur verilənlər seqmentinin ölçüsü ilə bağlıdır. Proqramda dəyişən və sabitlər bu seqmentdə saxlanır. Bu seqmentin ölçüsü 64 *Kb* və ya 65536 baytdır. Turbo Pascal dilində istənilən tip dəyişənin maksimal uzunluğu 65520 baytdır. Ona görə də, massivin ölçüsünü müəyyən etdikdə, bu məhdudiyyəti nəzərə almaq lazımdır. Məsələn,

```
Var
  a: array[1..200,1..100] of extended;
```

təsviri ilə `a` massivinin yaddaşda yerləşdirilməsi üçün, 200∗100∗10 bayt yaddaş tələb olunur ki, bu da 64 *Kb* – dan çoxdur (çünki, `Extended` tipli dəyişən yaddaşda 10 *bayt* yer

tutur). Bu Turbo Pascal dilinin nöqsanıdır və bu nöqsanı başqa üsullarla aradan qaldırmaq mümkündür.

3.5.2. Sətirlər

Sətirlər **`String`** tipi ilə müəyyənləşdirilir. `String` tipli sətirlər sətir simvollarından təşkil olunmuş birölçülü massivin xüsusi bir formasıdır və ümumi uzunluğu 255 – dən çox olmayan simvollar ardıcıllığından ibarətdir. Sətirlərin ümumi uzunluğu kvadrat mötərizədə göstərilir.

Misal.

```
var
  S1: string[12];
  S2: string[128];
  Smax: string;
```

Təsvirdə sətrin uzunluğu göstərilməzsə, onda susmaya görə 255 simvola bərabər maksimal uzunluq götürülür.

Sətirlər massivlərə uyğun gəldiyindən, sətrin istənilən simvoluna massivin elementi kimi müraciət etmək olar. Bunun üçün dəyişənin adının yanında, kvadrat mötərizə daxilində, simvolun nömrəsini göstərmək lazımdır. Massivin sıfırıncı elementi idarəedici element olmaqla, sətir tipli dəyişənin faktiki uzunluğunu göstərir. Əgər sətir dəyişəni

```
Var ad: string[4];
```

kimi elan olunarsa, `ad` dəyişəni 4 simvoldan çox qiymət ala bilməz, ona görə də proqramda

```
ad:='Ada';
```

yazılışı düzgün qəbul olunacaq, lakin

```
ad:='Pascal';
```

yazılışında isə `ad` dəyişəninin faktiki qiyməti `'Pascal'` yox, `'Pasc'` olacaqdır.

Əgər, proqramda `ad:='abcd';` yazsaq, onda sıfırıncı element `ad[0]` – sətirdə simvolların ümumi sayını göstərir, yəni `ord(ad[0])=4` olacaqdır. Massivin növbəti elementləri isə `ad[1]='a'`, `ad[2]='b'`, `ad[3]='c'` və `ad[4]='d'` olacaqdır. Göründüyü kimi, proqramçı sətir tipli dəyişənin istənilən simvoluna müraciət edə bilər.

Misal.

```
Var a,b:string;
 begin
   a:='Mən kitab oxuyuram';
   a[5]:='q';
   a[6]:='ə';
   a[7]:='z';
   a[8]:='e';
   a[9]:='t';
   b:=a;
end.
```

Bu proqrama əsasən `a` sətir dəyişəninin qiyməti `'Mən kitab oxuyuram'` olduğu halda, `b` dəyişəninin qiyməti `'Mən qəzet oxuyuram'` olacaqdır.

3.5.3. Çoxluqlar

Proqramlaşdırmada "çoxluq" tipi riyaziyyatdakı çoxluq anlayışına uyğun olaraq istifadə olunur. Fərq ondadır ki, Turbo Pascal dilində çoxluğun elementləri yalnız sıra tipli olmalıdır. Çoxluq əvvəlcədən müəyyən edilmiş qiymətlərdən seçilmiş elementlər yığımıdır. Bundan başqa, çoxluğun yuxarı və aşağı sərhədlərinin qiymətləri *0* ilə *255* intervalında olmalıdır. Buna görə də çoxluqda `ShortInt`, `Integer`, `LongInt` və `Word` tiplərindən istifadə etmək olmaz. Çoxluq

tipli dəyişənin aldığı qiymətlər kvadrat mötərizə daxilində göstərilir. Boş çoxluq **[]** kimi işarə olunur.

Çoxluq tipini təyin etmək üçün **set** və **of** işçi sözlərindən istifadə olunur və sonra bu çoxluğun elementləri göstərilir. Çoxluq tip dəyişənlərin ümumi təsvir forması belədir:

Set of *çoxluğun elementləri***;**

Çoxluq tipinin sabitlərinin hər bir komponenti, ya tipə uyğun ayrı sabit kimi, ya da bir–birindən **..** simvolları ilə ayrılan interval qiymətləri ilə təsvir olunur.

Misal.

```
type
  eded = set of 0..9;          {0–dan 9–a kimi
                                 rəqəmlər çoxluğu }

  simvol = set of '0'..'9'; {'0'–dan '9'–a kimi
                                 simvollar çoxluğu }

Const
  eded1  :eded = [0, 2, 4, 6, 8];
  eded2  :eded = [1..3, 5..7];
  simvol1:simvol=['0', '2', '4','6', '8'];
  simvol2:simvol =['0'..'3','5'..'7'];
  simvollar:Set of Char=['a'..'z','A'..'Z'];
```

Bu misalda, `type` bölməsində, iki çoxluq tipi müəyyənləşdirilmişdir: `eded` – *0*–dan *9*–a kimi rəqəmlər çoxluğu və `simvol` – *'0'*–dan *'9'*–a kimi simvollar çoxluğu. `Const` bölməsində `eded1` və `eded2` sabitləri `eded` tipli təyin olunmuş və onlara həmin çoxluqdan qiymətlər mənimsədilmişdir. Analoji qayda ilə `simvol1` və

`simvol2` sabitləri `simvol` tipli təyin olunmuş və onlara həmin çoxluqdan qiymətlər mənimsədilmişdir. `simvollar` sabiti isə birbaşa `Const` bölməsində çoxluq kimi təyin olunmuş və bu çoxluğun elementləri latın əlifbasının bütün hərflərindən ibarətdir.

Çoxluqlar üzərində əməliyyatlara "İfadələr" bölməsində baxacağıq.

3.5.4. Yazılar

Yazılar eyni bir suala aid olan müxtəlif verilənləri sadə üsulla birləşdirməyə imkan verir. Başqa sözlə, yazılar da massivlər kimi verilənlər yığımından ibarətdir, lakin massivlərdən fərqli olaraq, yazılar müxtəlif tipli verilənlərdən təşkil olunur. Massivin isə bütün elementləri həmişə eyni tipli olur. Yazılara misal olaraq işə qəbul olunan şəxsin anketini misal göstərmək olar. İşçi xüsusi blankda ad, ünvan, yaş, ailə vəziyyəti və s. kimi suallara cavab verməlidir. Aydındır ki, bu sualların bəzilərinə sözlərlə, bəzilərinə tam ədədlə, bəzilərinə isə məntiqi sabitlərlə (*hə – True, yox – False*) cavab vermək lazım gəlir. Əlbəttə, belə müxtəlif tipli verilənləri bir massivin elementləri kimi təsvir etmək qeyri–mümkündür. Digər tərəfdən, bu verilənlər nə qədər müxtəlif tipli olsalar da onlar bir nəfərə aiddir. Ona görə də belə verilənləri təsvir etmək üçün yazılardan istifadə edilir.

Müxtəlif verilənlər massivin elementləri adlandırıldığı halda, yazılar sahələrdən ibarət olur. Massivin ayrı–ayrı elementləri üzərində əməliyyatlar aparmaq mümkün olduğu kimi, yazıların da ayrı–ayrı sahələri üzərində əməliyyatlar aparmaq olar. Hər bir sahənin adı olur və yazı daxilində bu ad təkrarlana bilməz.

Yazılar iki növ olur: qeyd olunmuş hissəli və variantlı.

Qeyd olunmuş hissəli yazılar sonlu sayda sahələrdən ibarətdir. Onun elan edilməsinin ümumi forması belədir:

Record

1–ci sahənin adı **:** *sahənin tipi***;**
...
n–ci sahənin **:** *sahənin tipi***;**

end;

Yazının sahələrinə müraciət etmək üçün, aralarında nöqtə işarəsi qoymaqla, yazının adını və sahənin adını yazmaq lazımdır. Sahə üzərində onun tipinin yol verə biləcəyi əməliyyatları aparmaq olar.

Misal.

```
Var Persone: record
      Name:string;
      Address:string;
      Married:boolean;
      Salary:real;
    end;
  ...
  Persone.Name:='Abdullayev R.K.';
  Persone.Address:='Səməd Vurğun 31';
  Persone.Married:=True;
  Persone.Salary:=500;
```

Bu misalda, `Persone` yazının adıdır və onun dörd sahəsi müəyyənləşdirilmişdir: adı (`Name` – sətir tipli), ünvanı (`Address` – sətir tipli), ailə vəziyyəti (`Married` – məntiqi tipli) və əmək haqqı (`Salary` – həqiqi tipli). Proqramın icrası blokunda isə həmin sahələrə müvafiq qiymətlər mənimsədilmişdir. Sahələrə qiymətlər mənimsətmək üçün yazının öz adı göstərilməlidir, ona görə də proqramda, məsələn,

```
Persone.Address:='Səməd Vurğun 31';
```

yazılmışdır.

Misal.

```
type
 Complex = record
   re: real;
   im: real;
 end;
 Tarix = record
   il: integer;
   ay: 1..12;
   gun: 1..31;
 end;
```

Yazı tipi təsvir edildikdən sonra, bu tipin dəyişənləri və ya tipləşdirilmiş sabitləri verilə bilər. Yazı tipli sabitlərin təsvirində, yazının bütün sahələrinin qiymətləri ilə bərabər, onların identifikatorları da göstərilir. Yazı tipli tipləşdirilmiş sahələrdən istifadə etməyə icazə verilmir.

Yuxarıda təsvir edilmiş yazı tipindən sonra, aşağıdakı dəyişən və sabiti təyin etmək olar:

```
var
 X,Y,Z: complex;
 Tar: tarix;
const
 İngilt:tarix=(il: 1951; ay: 12; gün: 19);
```

Sahələrə müraciət etdikdə, hər dəfə yazının adını təkrarən yazmamaq üçün, **With** operatorundan istifadə olunur. Bu operatoru növbəti bölmələrdə öyrənəcəyik.

Variantlı yazılar da sonlu sayda sahələrdən ibarət olur, lakin yaddaşda sahələrin tutduğu yeri müxtəlif cür interpretasiya etməyə imkan verir. Yazının bütün variantları yaddaşın eyni bir hissəsində yerləşir və onlara müxtəlif adlarla müraciət etmək mümkün olur.

Variantlı yazının ümumi forması belədir:

Record
Case *əlamət* **:** *əlamətin tipi* **of**

1–ci variant **:** *variantın təsviri***;**
. . .
n–ci variant **:** *variantın təsviri***;**

end;

Qeyd olunmuş hissəli yazılar bir və ya bir neçə sahədən ibarət olur ki, hər bir sahənin adı və tipi onların təsvirində göstərilir. Bunu tələbələrin müvəffəqiyyətini əks etdirən yazıda göstərək.

Misal. Tələbələrin müvəffəqiyyətini əks etdirən yazı.

```
Type
 Str6 = String [6];
 Str20 = String [20];
 Sqiymat = record
    Aliriy:Byte;     { Ali riyaziyyat }
    Tarix :Byte;       { Tarix }
    İnform:Byte;     { İnformatika }
    Fizika:Byte;     { Fizika }
 end;

 Talaba = record
    Soyad:Str20;          { Soyadı}
    Ad   :Str20;          { Adı}
    Atasi:Str20;          { Atasını adı }
    İl   :Integer;        { Doğulduğu il }
    Unvan:Str20;          { Ünvan}
    Grup :Str6;           { Qrupun şifri }
    Qiymat:Sqiymat   { Sonuncu semestrin qiymətləri }
 end;
```

Bu yazıya baxdıqda görürük ki, buraya yalnız tələbənin bir semestrinin qiymətləri daxil edilmişdir. Hər semestrdə

fənlər dəyişdiyindən, bu yazıya bütün fənlər daxil edilməlidir. Bu isə lazımsız informasiyanın saxlanmasına gətirir. Bundan başqa, hər bir tələbə üçün yaddaşda bütün fənlər üçün yer ayrılır. Ona görə də belə hallarda variantlı yazılardan istifadə etmək məqsədəuyğundur. Bu yazıda da, qeyd olunmuş hissəli yazılarda olduğu kimi, bütün mümkün sahələr təsvir olunur. Amma, yaddaşda cari halda lazım olan variant üçün yer ayrılır. İki semestrin fənlərini nəzərə alsaq, yuxarıdakı yazının variantlı forması aşağıdakı kimi olar:

Misal. Variantlı yazı.

```
Type

 Str6 = String [6];
 Str20 = String [20];

 Sqiymat1 = record
  Aliriy1 :Byte;    { Ali riyaziyyat }
  Tarix   :Byte;    { Tarix }
  İnform1 :Byte;    { İnformatika }
  Fizika  :Byte;    { Fizika }
 end;

 Sqiymat2 = record
  Aliriy2 :Byte;    { Ali riyaziyyat }
  Electr  :Byte;    { Elektronika }
  İnform2 :Byte;    { İnformatika }
  İntexn  :Byte;    { İnformasiya texnologiyası }
 end;

 Talaba = record
  Soyad :Str20;     { Soyadı }
  Ad    :Str20;     { Adı }
  Atasi :Str20;     { Atasını adı }
  İl    :Integer;   { Doğulduğu il }
  Unvan :Str20;     { Ünvan }
```

```
    Grup   :Str6;       { Qrupun şifri }
```

{ *Yazının variant hissəsi* }

```
    case Semestr:Byte of

    1 : (Qiymat1:Sqiymat1);   { Birinci semestrin
                                qiymətləri }
    2 : (Qiymat2:Sqiymat2);   { İkinci semestrin
                                qiymətləri }

  end;
```

Gördüyümüz kimi, variantlı yazı iki hissədən ibarətdir. Birinci hissə qeyd olunmuş hissəli yazı, ikinci hissə isə bir neçə variantdan ibarət variant hissəsidir. Burada `Talaba` yazısının `Semestr` sahəsi iki – `Qiymat1` və `Qiymat2` kimi alternativ variantlardan təşkil olunmuşdur.

Qeyd. Yazı tip dəyişənlər mənimsətmə operatorlarında iştirak edə bilər, lakin onlar üzərində heç bir əməliyyatlar aparıla bilməz. Hesabi və digər əməliyyatlar yalnız yazı sahələri üzərində yerinə yetirilə bilər.

3.5.5. Fayllar

Fayl eyni tipli elementlərin xarici qurğularda – disklərdə tutduğu adlı yerə deyilir. Faylın diskdə nə qədər yer tutmasını əvvəlcədən göstərmək lazım deyildir. Diskdə yerləşən fayl üzərində hər hansı bir əməliyyat aparmaq üçün proqramda fayl dəyişənindən (məntiqi fayldan) istifadə olunur. Fayl dəyişəni proqramda təsvir olunduqdan sonra, əgər ona müraciət olunarsa, diskdəki faylla əlaqə yaranır, fayl üzərində əməliyyat aparılır və ona müvafiq dəyişikliklər edilir. Əməliyyat qurtardıqda faylla əlaqə kəsilir. Bundan sonra, fayl dəyişəni həmin tip başqa faylla əlaqə yarada bilər.

Elementlərin tiplərindən asılı olaraq üç növ fayl mövcuddur:

- *mətn tipli;*
- *tipləşdirilmiş;*
- *tipləşdirilməmiş.*

Mətn tipli fayllar müxtəlif uzunluqlu simvollar sətirlərindən ibarət olur və onların tipi **`Text`** işçi sözü ilə müəyyənləşdirilir. *Tipləşdirilmiş fayllar* proqramda göstərilən tipli (fayl tipi istisna olmaqla) elementlərdən ibarət olur və **`File of`** işçi sözü ilə müəyyənləşdirilir. *Tipləşdirilməmiş fayllar* isə tipləri göstərilməmiş elementlərdən ibarət olur və **`File`** işçi sözü ilə müəyyənləşdirilir. Fayl dəyişəninin tipi faylın elementlərinin tipinə uyğun olmalıdır.

Misal.

```
Var
Metn_fayli:Text;                    {Mətn tipli }
Tam_ededli_fayl:File of integer;{Tipləşdirilmiş –
                                    tamədədli }
Heqiqi_ededli_fayl:File of real;{Tipləşdirilmiş –
                                    həqiqi ədədli }
Tipsiz_fayl:File;                   {Tipləşdirilməmiş }.
```

Fayllarla əlaqədar əməliyyatları səkkizinci fəsildə öyrənəcəyik.

Qeyd. Pascal dilində mətn tipli fayllar **`Text`** sözü ilə təsvir olunur. Delphi–də isə bir sıra komponentlər `Text` xassəsinə malik olduğundan, çaşqınlıq yaranmaması üçün, mətn tipli faylların təsvirində **`TextFile`** və ya **`System.Text`** yazmaq lazımdır.

D ö r d ü n c ü f ə s i l

İFADƏLƏR

Bu fəsildə hesabi və məntiqi ifadələri, münasibət əməliyyatlarını öyrənəcəyik. Bu ifadələr üzərində yerinə yetirilən əməliyyatlar və onların yerinə yetirilmə ardıcıllığı şərh olunacaq, ənənəvi hesab əmlləri ilə yanaşı, tamədədli bölmə əməllərini, bitlər üzərində əməliyyatları öyrənəcəyik. İfadələrdə ən çox istifadə olunan funksiyaların Pascal dilində yazılışı, bir sıra hesab əməllərini icra edən standart funksiyalara müraciət, sətir ifadələri və onlar üzərində yerinə yetirilən əməliyyatlar, bu əməliyyatları yerinə yetirən standart funksiya və prosedurlar, ekranla işləmək üçün prosedurlar izah ediləcəkdir. Çoxluqlar üzərində əməliyyatlar öyrəniləcəkdir. Bütün bu məsələlər çoxlu misallar üzərində izah ediləcəkdir.

4.1. İfadələr üzərində əməliyyatlar

İfadələr proqramlaşdırmada ən çox istifadə olunan konstruksiyalardır. Riyaziyyatda ifadələr adətən düsturlarla təsvir olunduğu halda, proqramlaşdırmada hər hansı əməliyyatın təyini üçün istifadə edilir. İfadələr sabit, dəyişən, əməliyyat işarələri, adi mötərizələr və funksiyalardan təşkil olunur.

İstifadə olunan verilənlər və əməliyyatların tiplərindən asılı olaraq ifadələr *hesabi*, *məntiqi* və *sətir* tipli olur. Əməliyyatlar özləri isə aşağıdakı qruplara bölünür:

- *Hesabi əməliyyatlar:*
  +, –, *, /, div, mod
- *Münasibət əməliyyatları:*
  =, <>, <, >, <=, >=
- *Məntiqi (bul) əməliyyatlar:*
  not, and, or, xor
- *İnformasiya bitləri üzrə əməliyyatlar:*
  not, and, or, xor, shl, shr
- *Sətir əməliyyatı*
  + və ya **konkatenasiya**
- *Çoxluqlar üzrə əməliyyatlar:*
  +, –, *, in, <=, >=

Əməliyyatlar aşağıdakı ardıcıllıqla yerinə yetirilir:

1. **Not**
2. ***, /, div, mod, and, shl, shr**
3. **+, –, or, xor**
4. **=, <>, <, <=, >=, in**

İlk növbədə mötərizədaxili ifadələr hesablanır. Eyni prioritetli bir neçə ardıcıl əməliyyatlar soldan sağa istiqamətdə ardıcıl olaraq yerinə yetirilir.

4.2. Hesabi ifadələr

Hesabi ifadələrdə hamıya məlum olan ənənəvi hesab əməlləri ilə yanaşı, bir sıra xüsusi əməliyyatlar da tətbiq olunur. Bu əməliyyatların bəzilərini nəzərdən keçirək.

Div əməliyyatı iki tam tipli sabit və dəyişənləri tamədədli bölmə əməliyyatı adlanır.

Misal.

`39 div 4` = 9; `a=53` və `b=7` olduqda `a div b` = 7.

`Mod` əməliyyatı iki tam tipli sabit və dəyişənlərin tamədədli bölünmə qalığının tapılması əməliyyatı adlanır.

Misal.

`39 mod 4` = 3; `a=53` və `b=7` olduqda `a mod b` = 4.

Hər iki əməliyyat yalnız müsbət tam ədədlərə tətbiq olunur.

Tam tipli verilənlər üzərində daha mürəkkəb çevirmələr aparan əməliyyatlar da tətbiq olunur. Bunlar bitlər üzrə yerinə yetirilən **`shl`** – *sola sürüşdürmə* və **`shr`** – *sağa sürüşdürmə* əməliyyatlarıdır. Bu əməliyyatlar nəticəsində verilənlərin bitləri (mərtəbələri) sağa (`shr`) və ya sola (`shl`) sürüşdürülür; bu zaman artıq bitlər atılır, azad olan bitlər isə sıfırlarla doldurulur. Məsələn,

Misal.

```
00000111 shl 3 = 00111000;
00111000 shr 3 = 00000111.
```

Bu əməliyyatları işarəli tam ədədlərə tətbiq etdikdə yadda saxlamaq lazımdır ki, ədədin böyük mərtəbəsi onun işarəsini müəyyən edir. Ona görə də istənilən sağa sürüşdürmədə həmin bit sıfra çevriləcək, yəni alınan ədəd müsbət olacaqdır. Sola sürüşdürmədə isə nəticənin işarəsi ixtiyari ola bilər. Bu əməliyyatlar müsbət ədədlərə tətbiq olunduqda, `shl` əməliyyatının nəticəsi – verilmiş ədədi əsası 2 olan qüvvətüstlü kəmiyyətə vurmaqla, `shr` əməliyyatının nəticəsi isə bölməklə alınan nəticələrlə eyni olur. Başqa sözlə, yuxarıda yazılmış misallarla aşağıdakı əməliyyatlar eyni nəticəli olur:

`7 shl 3`, yəni $7*2^3=56$,
`56 shr 3`, yəni $56\ div\ 2^3=7$.

Qeyd edək ki, `shl` və `shr` əməliyyatları ilə hesablamalar daha tez yerinə yetirilir və daha optimal maşın kodları yaradılır.

Tam tipli verilənlər üzərində daha dörd *bit əməliyyatları* – **`not`, `and`, `or`** və **`xor`** yerinə yetirilə bilər. Xatırladırıq ki, bu əməliyyatlar verilənlərin bitləri üzərində yerinə yetirilir. Həmin əməliyyatlar cədvəl 4.1 – də göstərilmişdir.

Cədvəl 4.1. Bit əməliyyatları

Dəyişənlər		**Əməliyyatların nəticələri**				
a	**b**	**a and b**	**a or b**	**a xor b**	**not a**	**not b**
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Misal.

```
Not 01111011=10000100;
10111010 and 01100011=100010;
10111010 or 01100011=11111011;
10111010 xor 01100011=11011001;
48 and 36=32;
48 or 36=52;
48 xor 36=20.
```

Pascal dilində ifadələrdə hazır element kimi istifadə olunan əvvəlcədən hazırlanmış alt proqram – funksiyalar mövcuddur. Turbo Pascal dilində isə bunların sayı artırılmış və standart bir modulda yerləşdirilmişdir. Turbo Pascal proqramlarında sabit, dəyişən, prosedur və funksiyalardan istifadə edərkən, onların təyin olunduğu modullar təsvir olunmalıdır. İstifadəçi tərəfindən yaradılan modulların və **`System`** modulunun təsviri vacib deyil. Digər modullar hökmən təsvir edilməlidir. Funksiyalara müraciət edərkən onun adı, sonra isə adi mötərizədə funksiyanın arqumentləri göstərilməlidir. Arqumentlər birdən çox olduqda onlar bir–birindən vergüllə ayrılır.

Hesabi funksiyalar. Cədvəl 4.2 – də **System** modulunun tərkibinə daxil olan, sadə riyazi hesablamaları yerinə yetirən və ən çox istifadə edilən funksiyalar göstərilmişdir.

Hesabi ifadələri düzgün yazmaq üçün aşağıdakı qaydalara riayət etmək lazımdır:

– simvollar sətirdə bütün əməliyyat işarələri qoyulmaqla eyni səviyyədə yazılır;

– iki ardıcıl əməliyyat işarələrinə icazə verilmir. Məsələn, `A+-B` yazılışı səhvdir. Düzgün yazılış `A+(-B)` kimi olmalıdır.

Cədvəl 4.2 – dən göründüyü kimi, Turbo Pascal dilində qüvvətə yüksəltmə üçün əməliyyat və ya funksiya yoxdur. $y=x^n$ funksiyasının hesablanmasında aşağıdakılar məsləhət görülür: əgər n – tam ədəddirsə, onda qüvvətin hesablanmasında vurma əməliyyatından və ya `sqr` standart funksiyasından istifadə oluna bilər. Məsələn, $y=x^4$ funksiyası `y:=x*x*x*x` və ya `y:=sqr(x)*sqr(x)` kimi yazılır; böyük qüvvətlərdə vurma dövrlərdə hesablanır; əgər n – həqiqi ədəddirsə, onda $y=x^n=e^{nln(x)}$ riyazi düsturundan istifadə olunur ki, bu ifadə də Turbo Pascal dilində `y:=exp(n*ln(x))` kimi yazılır.

Cədvəl 4.2. Ən çox istifadə edilən funksiyalar

Riyazi funksiya	Proqramda yazılışı	Nəticənin tipi
$y=\|x\|$	`y:=abs(x)`	*x ilə eyni tipli*
$y=x^2$	`y:=sqr(X)`	*"_ _ _"*
$y=\sqrt{x}$	`y:=sqrt(X)`	*həqiqi*
y=arctgx	`y:=arctan(x)`	*"_ _ _"*
y=cosx	`y:=cos(x)`	*"_ _ _"*
y=sinx	`y:=sin(x)`	*"_ _ _"*
$y=e^x$	`y:=exp(x)`	*"_ _ _"*
y=lnx	`y:=ln(x)`	*"_ _ _"*
π ədədi	`Pi`	*"_ _ _"*

Misal. Turbo Pascal dilində bir neçə ifadənin yazılışına baxaq.

1. $y=ax+b$

```
y:=a*x+b;
```

2. $y=\frac{a+b}{c+d}\sin(x)$

```
y:=(a+b)/(c+d)*sin(x);
```

3. $y=\frac{a\ln x+b\sqrt{x}}{\cos x-x^2}$

```
y:=((a*ln(x)+b*sqrt(x))/
   (cos(x)-sqr(x));
```

4. $y=x^{\frac{1}{7}}+arctg\frac{x^2}{\sqrt{a}}$

```
y:=exp((1./7.)*ln(x))+
  arctan(sqr(x)/sqrt(a));
```

5. $y=\pi\cdot e^{ax+b}+\ln|x-a|$

```
y:=Pi*exp(a*x+b)+ln(
   abs(x-a));
```

6. $y=x^6$

```
y:=exp(6*ln(x));
```

7. $y=\frac{\cos^3(x-a)+\sin^2(x-a)^3}{\sqrt{|x-c^2z|}+\pi e^{\frac{a}{b}}}$

```
y:=(exp(3.0*ln(cos(x-a)))+sqr(sin(exp(3.0*
   ln(x-a)))))/(sqrt(abs(x-c*c*z))+pi*exp(a/b));
```

8. $y=\cos\varphi+\sin\gamma$

```
y:=cos(f)+sin(g);
```

və ya

```
y:=cos(fi)+sin(qamma));
```

4.3. Məntiqi ifadələr

Məntiqi ifadələrdə **and**, **or**, **xor** və **not** məntiqi əməliyyatlarından istifadə olunur ki, onlar üzərində

əməliyyatların yerinə yetirilmə qaydası cədvəl 4.3 – də göstərilmişdir.

Cədvəl 4.3. Məntiqi əməliyyatlar

Dəyişənlər		**Əməliyyatların nəticələri**				
a	**b**	**a and b**	**a or b**	**a xor b**	**not a**	**not b**
False	*False*	*False*	*False*	*False*	*True*	*True*
False	*True*	*False*	*True*	*True*	*True*	*False*
True	*False*	*False*	*True*	*True*	*False*	*True*
True	*True*	*True*	*True*	*False*	*False*	*False*

Misal. Turbo Pascal dilində bir neçə məntiqi ifadənin yazılışına baxaq.

Riyazi ifadə	***Turbo Pascal dilində yazılışı***
1. $y \geq \frac{a+b}{c+d}\sin(x)$	`y>=(a+b)/(c+d)*sin(x);`
2. $y \leq a\ln x$	`y<=a*ln(x);`
3. $x \leq a$ ***və ya*** $y \geq b$	`(x<=a) or (y>=b);`
4. $a < x \leq b$	`(x>a) and (x<=b);`
5. *y=a və x=c və z=d*	`(y=a) and (x=c) and (z=d);`

4.4. Münasibət əməliyyatları

Münasibət əməliyyatları tam, həqiqi, məntiqi, simvol, sətir tipli verilənlər və massivin elementləri üzərində yerinə yetirilə bilər. Münasibət əməliyyatlarının nəticəsi məntiqi sabitlərdir (*True*, *False*). Simvol və sətir tipli sabitlərin müqayisəsi **ASCII** simvollar koduna uyğun olaraq, soldan sağa istiqamətdə, simvollar üzrə həyata keçirilir. Hansı simvolun kodu böyükdürsə, həmin sətir böyük olur.

Turbo Pascal dilində bir neçə münasibət əməliyyatlarının yazılışına baxaq.

Misal.

İfadə	Nəticə
10>5	True
6=5	False
False<>True	True
'ADNA'< 'BDU'	True
'ALFA' > 'A'	True
'AAHM'='AAHM'	True
'AAHM '>'AAHM'	True
'A'>'a'	False

4.5. Sətir ifadələri

Sətir ifadələri üzərində əməliyyatların nəticələri simvollardan ibarət sətirlər olur. Sətirlər üzərində yalnız toplama əməli yerinə yetirilə bilər ki, bu əməliyyat nəticəsində sətirlərin birləşməsi (*konkatenasiya*) baş verir.

Misal.

```
S:='BAKI '+'AZƏRBAYCANIN '+'PAYTAXTIDIR';
```

Bu operatorun icrasından sonra, `S` sətir tipli dəyişənin qiyməti `'BAKI AZƏRBAYCANIN PAYTAXTIDIR'` olacaqdır.

Konkatenasiya əməliyyatı nəticəsində alınan sətir *255* simvoldan çox olmamalıdır. Əgər sətrin uzunluğu *255* – dən çox olarsa, onda *255* – ci simvoldan sonrakı simvollar atılır.

Bundan başqa, yuxarıda göstərdiyimiz kimi, sətirlər müqayisə də oluna bilər. Əgər sətirdə simvolların sayı bərabər, simvollar isə ekvivalent olarsa, onda sətirlər bərabər olur. Sətirlərin müqayisəsi hər simvol üzrə yerinə yetirilir və hansı simvolun kodu böyükdürsə, həmin sətir böyük olur.

Misal.

```
'BAKI'='BAKI';
'A' < 'a' {"A" – nın kodu 65, "a" – nın kodu isə 97 – dir};
```

```
'AZƏRBAYCAN' > 'BAKI'
```

`String` tipli sətirlərlə işləmək üçün Turbo Pascal dilində aşağıdakı funksiya və prosedurlardan istifadə olunur:

`Concat` (*s1, s2, ..., sN***); *funksiyası*** – *s1, s2, ..., sN* sətirlər ardıcıllığının *konkatenasiyasını* (birləşməsini) yerinə yetirir.

`Copy` (*st, index, count***); *funksiyası*** – verilmiş *st* sətrindən, *index* nömrəli simvoldan başlayaraq *count* sayda simvolların *surətini qaytarır*.

Misal.

```
SS:='Microsoft Word -2003';
NSS:= Copy(SS,11,4);
```

olarsa, nəticədə `NSS` sətir dəyişəninin qiyməti `NSS:='Word'` olacaqdır.

`Delete` (*st, index, count***); *proseduru*** – verilmiş *st* sətrindən, *index* nömrəli simvoldan başlayaraq *count* sayda *simvolları pozur*.

Misal.

```
SS:='Microsoft Word -2003';
Delete(SS,11,4);
```

olarsa, nəticədə `SS` sətir dəyişəninin qiyməti `SS:='Microsoft -2003'` olacaqdır.

`Insert` (*subst, st, index***); *proseduru*** – *subst* alt sətrini, *index* nömrəli simvoldan başlayaraq, *st* sətrində *yerləşdirir*.

Misal.

```
SS:='Microsoft -2003';
Insert('Word',SS,11);
```

olarsa, nəticədə `SS` sətir dəyişəninin qiyməti `SS:='Microsoft Word -2003'` olacaqdır.

Pos(*subst, st***);** ***funksiyası*** – *st* sətrinə daxil olan *subst* alt sətrinin *başlanğıc mövqeyini* müəyyən edir. Əgər *st* sətrində *subst* alt sətri mövcud olmazsa, onda `Pos` funksiyası sıfra bərabər olur.

Misal.

```
SS:='Microsoft Word -2003';
Pos('Word',SS);
```

olarsa, `Pos` funksiyasının nəticəsi `11` olacaqdır.

Length(*st***);** ***funksiyası*** – *st* sətrinin *uzunluğunu qaytarır*. `Length('Turbo Pascal')` funksiyasının qiyməti `12` olacaqdır.

Str (*x* [**:***m* [**:***n*], *st* **);** ***proseduru*** – istənilən *x* həqiqi və ya tam tipli ədədi, *st sətir simvollarına (tipinə) çevirir*. *m* və *n* parametrləri çevirmə formatını göstərir. Bu parametrlər yazılmaya da bilər. *m* parametri *x* həqiqi və ya tam tipli ədədinin simvol təsvirindəki ümumi sahənin uzunluğunu, *n* isə kəsr hissədəki simvolların sayını təyin edir. *x* yalnız həqiqi ədəd olduqda *m* və *n* parametrlərindən istifadə olunur.

Misal.

```
Const X:=2000;
Var s:string;
...
Str(x,s,err);
```

Nəticədə `x` sabitinin qiyməti olan 2000 ədədi simvollara çevrilərək `s` dəyişəninə mənimsədiləcəkdir, yəni `s='2000'` olacaqdır.

Val (*st, x, code***);** ***proseduru*** – *st* simvollar sətrini, *həqiqi və ya tamədədli x təsvirinə (tipinə) çevirir*. Əgər *code* – nin qiyməti sıfır olarsa, çevirmə müvəffəqiyyətlə başa çatır, əks halda, *st* sətrinin səhv olan simvolunun mövqeyinin nömrəsini qaytarır.

Misal.

```
Const s:='2000';
Var x:integer;
...
Val(s,x,err);
```

Nəticədə, s sabitinin simvollardan ibarət '2000' sətir qiyməti, 2000 ədədinə çevrilərək x dəyişəninə mənimsədiləcəkdir, yəni x=2000 olacaqdır. Hər iki misalda çevirmə düzgün yerinə yetirilərsə, err parametrinin qiyməti sıfra bərabər olacaqdır.

UPCASE (*ch***);** ***funksiyası*** *– kiçik latın hərflərini uyğun böyük hərflərə çevirir*, nəticə char tipli olur. UPCASE('t') funksiyasının qiyməti 'T' olacaqdır.

4.6. Standart funksiya və prosedurlar

Turbo Pascal dilində standart funksiya və prosedurlar müxtəlif əməliyyatları çox asanlıqla yerinə yetirmək üçün nəzərdə tutulmuşdur. Məsələn, simvolun tam ədədə, həqiqi ədədin tam ədədə çevrilməsi, həqiqi ədədlərin tam və kəsr hissələrinin tapılması və s. Bu funksiyalardan bəziləri aşağıda göstərilmişdir.

Chr(*x***);** – *ASCII kodunun simvola çevrilməsi.* Funksiyanın arqumenti 0..255 intervalında olmaqla, tam ədəd olmalıdır. Nəticədə bu koda uyğun simvol alınır. Məsələn, chr(97) ifadəsinin nəticəsi 'a' olacaqdır.

Ord(*x***);** – *istənilən sıra tipinin tam tipə çevrilməsi*. Funksiyanın arqumenti ixtiyari sıralı (məntiqi, simvol və sadalanan) tip ola bilər. Nəticədə longint tipli kəmiyyət alınır. Məsələn, Ord('a') ifadəsinin nəticəsi 97 olacaqdır.

Round(*x***);** – *x həqiqi ədədinin qiymətinin bu ədədə yaxın olan tam ədədə qədər yuvarlaqlaşdırılması.*

Funksiyanın arqumenti həqiqi, nəticə isə `longint` tipində olur. Məsələn, `Round(7.96)=8;` `Round(7.06)=7` olur.

`Trunc`(*x*)**;** – *x həqiqi ədədinin tam hissəsinin tapılması*. Funksiyanın arqumenti həqiqi, nəticə isə `longint` tipində olur. Məsələn, `Trunc(7.96)=7;` `Trunc(7.06)=7` olur.

`Int`(*x*)**;** – *tipini dəyişdirmədən x həqiqi ədədinin tam hissəsinin tapılması.* Məsələn, `İnt(75.96)=75.0;` `İnt(13457.88976)=13457.0` olur.

`Frac`(*x*)**;** – *x həqiqi ədədinin kəsr hissəsinin tapılması.* Məsələn, `Frac(7.96)=0.96;` `Frac(7.06)=0.06` olur.

Sıralı tip kəmiyyətlər üçün funksiyalar. Bu funksiyalar əvvəlki və ya sonrakı elementlərin tapılması, ədədin təkliyinin yoxlanılması və s. üçündür. Buraya aşağıdakı funksiyalar aiddir:

`Odd`(*x*)**;** – *x dəyişəninin təkliyinin yoxlanılması.* Funksiyanın arqumenti `longint` tipli, nəticə isə arqument tək olduqda – *True,* cüt olduqda isə *False* olur. Məsələn, `Odd(32)` funksiyasının nəticəsi *False,* `Odd(33)` funksiyasının nəticəsi isə *True* olacaqdır.

`Pred`(*x*)**;** – *x dəyişəninin əvvəlki qiymətinin təyini.* Funksiyanın arqumenti sıra tipli ixtiyari kəmiyyət, nəticə isə həmin tipli əvvəlki qiymətdir. Məsələn, `Pred(20)` funksiyasının nəticəsi `19`, `Pred('B')` funksiyasının nəticəsi isə `'A'` olacaqdır.

Succ(*x*); – *x dəyişəninin sonrakı qiymətinin təyini.* Funksiyanın arqumenti sıra tipli ixtiyari kəmiyyət, nəticə isə həmin tipli sonrakı qiymətdir. Məsələn, `Succ(2)` funksiyasının nəticəsi `3`, `Succ('B')` funksiyasının nəticəsi isə `'C'` olacaqdır.

Inc(*x*); – *x dəyişəninin qiymətini 1 vahid artırır.* Məsələn, əgər `a:=4` olarsa, `Inc(a)` icra olunduqdan sonra `a=5` olacaqdır.

Dec(*x*); – *x dəyişəninin qiymətini 1 vahid azaldır.* Məsələn, əgər `a:=4` olarsa, `dec(a)` icra olunduqdan sonra `a=3` olacaqdır.

Random; – *$0 \le s < 1$ aralığında təsadüfi s ədədi yaradır.*

Random(*x*); – *$0 \le s < x$ aralığında təsadüfi s ədədi yaradır.*

Ekranla işləmək üçün prosedurlar.

ClrScr;– *ekranı təmizləyir, onu fonun rəngi ilə rəngləyir və kursoru ekranın sol yuxarı küncünə yerləşdirir.*

GotoXY (*x*, *y*:Byte); – *kursoru ekranın x,y dəyişənləri ilə verilmiş mövqeyinə yerləşdirir. x* – sütunu, *y* isə sətri göstərir. Məsələn, `GotoXY(30,25)` o deməkdir ki, kursor *25*–ci sətrin *30*–cu mövqeyində yerləşəcəkdir.

WhereX; – *üfqi koordinat oxu üzrə kursorun cari x koordinatını müəyyən edir.*

WhereY; – *şaquli koordinat oxu üzrə kursorun cari y koordinatını müəyyən edir.*

4.7. Çoxluqlar üzərində əməliyyatlar

Çoxluqlar üzərində əməliyyatlar çoxluqlar nəzəriyyəsinin qaydalarına görə aparılır.

İki çoxluğun birləşməsi, yəni `A+B` əməliyyatının nəticəsi, həm `A` çoxluğunun, həm də `B` çoxluğunun bütün təkrarlanmayan elementlərindən ibarət `C` çoxluğudur.

İki çoxluğun fərqi, yəni `A-B` əməliyyatının nəticəsi, `A` çoxluğunun `B` çoxluğuna daxil olmayan elementlərindən ibarət `C` çoxluğudur.

İki çoxluğun kəsişməsi, yəni `A*B` əməliyyatının nəticəsi, `A` və `B` çoxluqlarının eyni elementlərindən təşkil olunmuş `C` çoxluğudur.

`A` və `B` çoxluqlarının elementləri eyni olduqda `A=B` əməliyyatının nəticəsi *True* və `A<>B` əməliyyatının nəticəsi *False* olur.

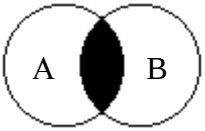
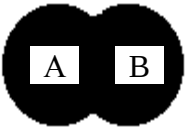
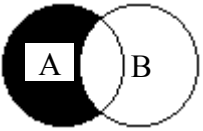
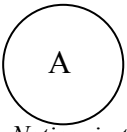
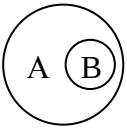
Əgər `A` çoxluğu `B` çoxluğunun alt çoxluğudursa, onda `A<=B` əməliyyatının nəticəsi *True* olur.

Əgər `A` çoxluğu `B` çoxluğunun bütün elementlərini özündə saxlayırsa, onda `A>=B` əməliyyatının nəticəsi *True* olur.

Əgər hər hansı *x* kəmiyyəti `A` çoxluğunun elementidirsə, onda `x in A` əməliyyatının nəticəsi *True* olur.

Cədvəl 4.4 – də çoxluqlar üzərində əməliyyatların həndəsi təsviri göstərilmişdir.

Cədvəl 4.4. Çoxluqlar üzərində əməliyyatların həndəsi təsviri

Riyazi işarəsi	**Proq-ramda işarəsi**	**Əməliyyat-lar**	**Həndəsi təsviri**
∩	`*`	*Kəsişmə*	 C=A∩B `C:=A*B` *Nəticənin tipi – çoxluq*
∪	`+`	*Birləşmə*	 C=A∪B `C:=A+B` *Nəticənin tipi – çoxluq*
\	`-`	*Fərq*	 C=A\B `C:=A-B` *Nəticənin tipi – çoxluq*
∈	`in`	*Aiddir və ya çoxluğun elementidir*	 *X∈A* `if X in A then` *Nəticənin tipi* – `boolean`
⊂ (⊆) ⊃ (⊇)	`<=` `>=`	*Alt çoxluqdur* *Alt çoxluğu var*	*Çoxluğun operandları*  *A⊃B [B⊃A]* `A>=B [B<=A]` *Nəticənin tipi* – `boolean`

Misal.

İfadə	Nəticə
`[1,2,3,4]+[3,4,5,6]`	`[1,2,3,4,5,6]`
`[1,2,3,4]-[3,4,5,6]`	`[1,2]`
`[1,2,3,4]*[3,4,5,6]`	`[3,4]`
`[1,2,3]=[1,2,3,4]`	`False`
`[1,2,3]<>[1,2,3,4]`	`True`
`[1,2,3]<=[1,2,3,4]`	`True`
`[1,2,3]>=[1,2,3,4]`	`False`
`4 in [3,4,5,6]`	`True`

Misal.

```
Type MonthDays=Set of 1..31;
Var color: set of (Red,Blue,White,Black);
    Day: MonthDays;
Begin
  ...
  Color:=[Blue];     { Color dəyişəninə Blue
                       qiyməti mənimsədilir }
  Color:=Color-[blue,red,white];{Color=[ ]
                                    olur }
  Color:=Color+[Black];  { Color=Black olur }

  Day:=[ ];            { Day dəyişəni boş çoxluqdur }

  Day:=Day-[1];           { Day=[ ]-[1]=[ ] }
  Day:=[2,4];
  Day:=Day+[5,12,1]; {Day=[2,4]+[5,12,1]=
                          [2,4,5,12,1] }

  Day:=Day-[1];      {Day=[2,4,5,12,1]-
                          [1]=[2,4,5,12] }
end;
```

Beşinci fəsil

OPERATORLAR

Bu fəsildə proqramlaşdırmanın ən vacib konstruksiyaları olan operatorlar bölməsi öyrəniləcəkdir. İlkin verilənləri proqrama daxil etmək və alınmış nəticələri ekranda təsvir etmək üçün verilənləri daxiletmə və xaricetmə prosedurlarını, mənimsətmə və keçid operatorlarını öyrənəcəyik. Strukturlaşdırılmış operatorlar qrupuna daxil olan tərkibli operator, şərti operator, seçim operatoru, üç növ dövr operatorları və `With` operatoru izah olunacaqdır. Bu operatorların mümkün yazılış konstruksiyalarına baxılacaq və onların tətbiqi ilə müxtəlif məsələlər həll ediləcəkdir.

5.1. Verilənləri daxiletmə və xaricetmə prosedurları

Turbo Pascal dilində standart daxiletmə – `Read`, `Readln` və standart xaricetmə isə `Write`, `Writeln` prosedurları vasitəsilə həyata keçirilir. Bu prosedurların ümumi forması belədir:

`Read` (*fayl dəyişəninin adı, daxil ediləcək dəyişənlərin siyahısı*)**;**
`Readln` (*fayl dəyişəninin adı, daxil ediləcək dəyişənlərin siyahısı*)**;**
`Write` (*fayl dəyişəninin adı, xaric ediləcək dəyişənlərin siyahısı*)**;**
`Writeln` (*fayl dəyişəninin adı, xaric ediləcək dəyişənlərin siyahısı*)**;**

`Read` və `readln`, `write` və `writeln` prosedurlarının yazılışlarındakı fərq ondan ibarətdir ki, `Read`, `write` prosedurları ilə informasiya bir sətirdə, `Readln`, `writeln`

prosedurları ilə isə informasiya yeni sətirdə *("read line", "write line")* təsvir olunur.

Fayl dəyişəni ilkin informasiyanın daxil ediləcəyi və ya nəticələrin saxlanacağı faylla əlaqə yaradan *məntiqi fayl dəyişənidir* və onun haqqında "Fayllar" bölməsində daha geniş bəhs edəcəyik. Daxiletmə prosedurları fayldan yalnız bir simvol oxuyaraq onu dəyişənə mənimsədir. Ədədlərin oxunması isə birinci probelə, tabulyasiya simvoluna, sətrin sonu işarəsinə və ya faylın sonu işarəsinə kimi həyata keçirilir. Bu dəyişən yazılmadıqda standart daxiletmə (*klaviaturadan*) və standart xaricetmə (*monitora*) yerinə yetirilir. `Writeln` proseduru mətn faylları üçün `write` prosedurunun genişlənmiş variantıdır. Parametrsiz `writeln` proseduru faylın sonuna yalnız sətrin sonu işarəsini yazır.

Standart daxiletmə əvvəlcədən təyin olunmuş, klaviatura ilə əlaqəli `Input` adlı mətn faylından yerinə yetirilir. Standart xaricetmə isə əvvəlcədən təyin olunmuş, monitorla əlaqəli `Output` adlı mətn faylından yerinə yetirilir. Susmaya görə daxiletmə üçün `Input`, xaricetmə üçün isə `Output` götürülür:

```
readln(Input, A, B);
writeln(Output, 'A=', A, 'B=', B);
```

Proqramda `Input` və `Output` sözlərini yazmamaq daha məqsədəuyğundur və bu zaman giriş–çıxış qurğuları kimi klaviatura və monitor başa düşülür, yəni informasiya klaviaturadan daxil edilir və monitora çıxarılır:

```
readln(A,B);
writeln('A=',A, 'B=',B);
```

Standart daxiletmə və xaricetmə prosedurlarından istifadə edərkən aşağıdakılar nəzərə alınmalıdır:

– `read` və `readln` prosedurları ilə yalnız tam, həqiqi, simvol və sətir tipli verilənlər daxil edilir;

– `write` və `writeln` prosedurları ilə yalnız tam, həqiqi, simvol, sətir və məntiqi tipli verilənlər monitora çıxarılır.

5.1.1. Verilənlərin ekrandan daxil edilməsi

Verilənləri ekrandan daxil etdikdə `Read` və `readln` prosedurlarının ümumi forması belə olur:

`Read` (*daxil ediləcək dəyişənlərin siyahısı*)**;**
`Readln` (*daxil ediləcək dəyişənlərin siyahısı*)**;**

Daxiletmədə `read` prosedurundan fərqli olaraq `readln` proseduru verilənlərin növbəti sətrin başlanğıcından oxunmasını həyata keçirir. Əgər proqramda sadəcə olaraq

```
Readln;
```

yazılarsa, onda proqramın yerinə yetirilməsi dayandırılır və proqramın işini davam etdirmək üçün *Enter* klavişini basmaq lazımdır. Daxiletmə prosedurunun bu formasından, adətən, Turbo Pascal dilinin inteqrallaşdırılmış mühitində istifadəçi pəncərəsinin ekranda görünməsini təmin etmək üçün istifadə olunur. Belə ki, bu məqsədlə sonuncu `end.` operatorundan əvvəl `readln` yazmaq lazımdır. İnteqrallaşdırılmış mühitin pəncərəsini yenidən ekranda göstərmək üçün *Enter* klavişini basmaq lazımdır.

Dəyişənlərin qiymətlərini klaviaturadan daxil etdikdə, onları aralarında probel işarəsi qoymaqla bir sətirdən və ya dəyişənlərin hər qiymətini bir sətirdən daxil etmək lazımdır (*Enter* klavişini basmaqla).

Misal. a=7,8 ; b=10,965; s= “BAKI” və k= –654 qiymətlərinin klaviaturadan daxil edilməsi.

```
program daxil_etme;
```

```
uses crt;
var a,b:real;
   k:integer;
   s:string[4];
begin
 read(s,a,b,k);
 writeln(a,b,k,s);
 Readln
end.
```

Bu proqrama müvafiq olaraq dəyişənlərin qiymətlərini klaviaturadan

```
BAKI   7.8   10.965   -654
```

kimi, və ya hər qiyməti bir sətirdə yazmaqla, daxil etmək lazımdır.

Sətir tipli dəyişənləri daxil etdikdə, `Read` proseduru, elanedici hissədə sətrin uzunluğu nə qədər göstərilmişdirsə, bir o qədər simvol oxuyur. Bu zaman probel ayırıcı rolunu oynamır. `Read` proseduru yeni sətrə keçidi yerinə yetirmir, bu əməliyyatı `readln` proseduru yerinə yetirir. Sonuncu `readln` proseduru inteqrallaşdırılmış mühitin istifadəçi pəncərəsini ekranda ləngitmək üçün yazılmışdır. *Enter* klavişini basdıqdan sonra, yenidən inteqrallaşdırılmış mühitə qayıdacaqsınız.

Misal.

```
program daxil_et;
uses crt;
var A:array [0..5] of Char;
    S1,S2,S3:string [10];
begin
 Read(A);
 Read(S1);
 Read(S2);
 Readln;
 Read(S3);
```

```
  Readln
end.
```

Bu halda, bir sətirdə `A` massivinin elementləri üçün 5 simvol, onun ardınca `S1` və `S2` sətir dəyişənlərinin hər biri üçün 10 simvol yazaraq *Enter* klavişini basdıqdan sonra, növbəti sətirdə `S3` sətir dəyişəni üçün 10 simvol daxil etmək lazımdır. Qeyd edək ki, massivlərin elementlərinin belə daxil edilməsi məqsədəuyğun deyildir və sonrakı bölmələrdə massivlərin elementlərinin daxil edilməsi üçün daha səmərəli üsullar göstərəcəyik.

5.1.2. Verilənlərin ekrana çıxarılması

Verilənləri ekrana çıxardıqda `Write` və `Writeln` prosedurlarının ümumi forması belə olur:

`Write` (*xaric ediləcək dəyişənlərin siyahısı* **:** m **:** n **);**
`Writeln` (*xaric ediləcək dəyişənlərin siyahısı* **:** m **:** n **);**

Yuxarıda qeyd etdiyimiz kimi, əgər fayl dəyişəni yazılmazsa, onda nəticələr ekrana çıxarılacaqdır. Nəticələri printerdə çap etmək üçün fayl dəyişəninin adı əvəzinə `'prn'` (“printer”) yazmaq lazımdır. Məsələn,

```
Writeln('prn',a,b,c);
```

Əgər ekrana çıxarılma prosedurları

`Write` (*xaric ediləcək dəyişənlərin siyahısı* **);**
`Writeln` (*xaric ediləcək dəyişənlərin siyahısı* **);**

kimi yazılarsa, onda həqiqi ədədlər həmişə sürüşkən vergüllü formatda təsvir olunacaqdır.

`Write` və `Writeln` prosedurlarının ümumi formalarındakı *m*– xaricedilmə sahəsinin ümumi uzunluğunu, *n* isə xaric edilən dəyişənin onluq hissəsindəki rəqəmlərin sayını göstərən parametrlərdir (*m*>*n*). *n* parametri, çap

olunacaq dəyişən yalnız həqiqi tipli olduqda göstərilir. *n* göstərildikdə ədəd qeyd olunmuş vergüllü formatda, göstərilmədikdə isə sürüşkən vergüllü formatda təsvir olunur. Tam tipli və sətir tipli verilənlər üçün *n* yazılmır.

Əgər format `a:m:0` kimi yazılarsa, onda ədədin kəsr hissəsi tam ədədə qədər yuvarlaqlaşdırılacaq və onluq nöqtə ekrana çıxarılmayacaqdır. Bu halda, məsələn, *a=1248.6* olarsa və format `a:6:0` kimi yazılarsa, onda bu ədəd, ekranda qarşısında iki probel işarəsi qoyulmaqla, yuvarlaqlaşdırılaraq *1249* kimi təsvir ediləcəkdir. Əgər *a=150* olduqda, format `a:5:0` kimi yazılarsa, onda ədədin qarşısında iki probel işarəsi qoyulacaqdır. *c=1248.45* ədədi üçün `c:10:2` formatı yazılarsa, onda kompüter bu ədədi, qarşısına üç probel əlavə etməklə olduğu kimi təsvir edəcəkdir; əgər səhv olaraq `c:2:8` formatı yazılarsa, onda ədəd *1248.45000000* kimi təsvir ediləcəkdir.

Əgər proqramda sadəcə olaraq

```
Writeln;
```

yazılarsa, onda bir boş sətir buraxılır.

Misal.

```
program ekrana_cixarma;
uses Crt;
const
 i: Integer = 12345;
 r: Real =  -123.1234567;
 c: Char = '$';
 b: Boolean = True;
 s:string='Müasir proqramlaşdırma dilləri';
begin
 ClrScr;
 writeln('formatsız çap');
 writeln(i, r, c, b, s);
 writeln;
 writeln('formatlı çap');
```

```
    Writeln(i:10, r:10:3, c:10, b:6, s:35);
    Writeln;
    writeln('Həqiqi ədədlərin qeyd
           olunmuş formatda təsviri');
    writeln(r:3:0);
    writeln(r:6:3);
    writeln(r:13:7);
    writeln(r:25:9);
    writeln;
    writeln('Həqiqi ədədlərin sürüşkən
           formatda təsviri');
    writeln(r:3);
    writeln(r:6);
    writeln(r:13);
    writeln(r:25);
    Readln
end.
```

Bu proqramın nəticəsi şəkil 5.1 – də göstərilmişdir.

5.2. Mənimsətmə operatoru

Mənimsətmə operatoru dilin əsas operatorudur. Bu operatorun ümumi forması belədir:

a:=b;

Burada, `b` – sabit, dəyişən, ifadə və massivin elementi, `a` isə dəyişən və ya massivin elementidir. **:=** işarəsi bərabərlik işarəsindən fərqlidir. Belə ki, bu operator icra olunduqda `b` ifadəsi hesablanır və onun nəticəsi `a` dəyişəninə mənimsədilir.

Misal.

```
x:=x+1;
a:=3.8;
b:=8*Pi/sin(x);
s:='TARİX';
```

```
y:=(a+b)/(c+d);
```

```
formatsiz cap
12345-1.2312345670E+02$TRUEMuasir programlashdirma dilleri

formatli cap
     12345  -123.123         $  TRUE    Muasir programlashdirma dilleri

Heqiqi ededlerin qeyd olunmush formatda tesviri
-123
-123.123
 -123.1234567
          -123.123456700

Heqiqi ededlerin surushken formatda tesviri
-1.2E+02
-1.2E+02
-1.231235E+02
       -1.2312345670E+02
```

Şəkil 5.1. Müxtəlif tip verilənlərin ekranda təsvir qaydaları

Misal. Verilmiş 4 rəqəmli tam ədədin rəqəmlərinin tərsinə düzülüşündən alınan ədədin tapılması.

```
program ters_duz;
uses crt;
const n=4658;
var m1qis,m1qal,m2qis:integer;
    m2qal,m3qis,m3qal:integer;
    m:integer;
begin
   m1qis:= n div 1000;
   m1qal:= n mod 1000;
   m2qis:= m1qal div 100;
   m2qal:= m1qal mod 100;
   m3qis:= m2qal div 10;
   m3qal:= m2qal mod 10;
   m:=m3qal*1000+m3qis*100+m2qis*10+m1qis;
   writeln('4 rəqəmli tam ədəd = ',n:4);
   writeln('tərsinə düzülmüş tam
           ədəd = ',m:4);
   readln
end.
```

Bu məsələnin alqoritmi belədir: verilmiş ədədi 1000 ədədinə tamədədli böldükdə (`div` əməliyyatı – `m1qis` dəyişəni) alınmış qismət axtarılan ədədin ən kiçik mərtəbəsi (4–cü rəqəmi – `m1qis` dəyişəni) olur. Bu bölmə nəticəsində alınan qalığı (`mod` əməliyyatı – `m1qal` dəyişəni) 100 ədədinə bölüb, alınmış qisməti 10–a vursaq, axtarılan ədədin 2–ci rəqəmini (`m2qis*10`) alarıq. Bu bölmə nəticəsində alınan qalığı 10 ədədinə bölüb, alınmış qisməti 100–ə vursaq, axtarılan ədədin 3–cü rəqəmini (`m3qis*100`) alarıq. Axtarılan ədədin birinci rəqəmi isə sonuncu bölmə əməliyyatından alınmış qalığı 1000–ə vurmaqla alınır (`m3qal*1000`). Nəhayət, sonda bu ədədləri toplamaq lazımdır:

```
m:=m3qal*1000+m3qis*100+m2qis*10+m1qis;
```

Proqramın icrasından sonra aşağıdakı nəticələr alınacaqdır:

```
4 rəqəmli tam ədəd = 4658
Tərsinə düzülmüş tam ədəd = 8564
```

5.3. Keçid operatoru

Bu operator, bütün dillərdə olduğu kimi, hesablama ardıcıllığını dəyişərək idarətməni hər hansı bir operatora vermək üçündür. Keçid operatorunun ümumi forması belədir:

`goto` *nişan*;

Burada *nişan* idarətməni qəbul edəcək operatorun nişanıdır. Xatırladaq ki, nişan ya identifikator, ya da 0 – 9999 diapazonunda dəyişən işarəsiz tam ədəd ola bilər və o `Label` operatoru ilə təsvir olunmalıdır. Nişanla operator arasında **:** işarəsi qoyulur.

Misal.

```
label nischan, 56;
...
...
```

```
    goto nischan;
...

56: y:=a;
...
nischan: x:=x+1;
```

Unutmayın ki, idarəetməni strukturlaşdırılmış operatorların daxilində yerləşən operatorlara vermək olmaz. Ümumiyyətlə, proqramlaşdırmada bu operatorun istifadə edilməsi məsləhət görülmür, çünki bu halda proqramın etibarlılığı, dayanıqlığı azalır. Strukturlaşdırılmış dillər bu operatordan istifadə etmədən də proqramlar tərtib etməyə imkan verir.

5.4. Boş operator

Boş operator sətirdə yalnız bir nöqtəli–vergül işarəsindən ibarətdir və proqramda operatorların yerləşə biləcəyi istənilən hissədə yazıla bilər. Boş operator nişanlana da bilər. Boş operator heç bir əməliyyat yerinə yetirmir və idarəetməni dövrün və ya tərkibli operatorun sonuna vermək üçün istifadə olunur.

5.5. Strukturlaşdırılmış operatorlar

Strukturlaşdırılmış operatorlar müəyyən qayda ilə digər operatorlardan, ifadələrdən və işçi sözlərdən yaradılır. Bu operatorlara aiddir:

- *tərkibli operator;*
- *şərti operator;*
- *seçim operatoru;*
- *dövr operatorları;*
- `With` *operatoru.*

5.5.1. Tərkibli operator

Tərkibli operator **begin** və **end** mötərizə operatorları daxilində yerləşən və bir–birindən nöqtəli–vergüllə ayrılan ixtiyari sayda operatorlar ardıcıllığından ibarət operatordur. Bu operatorun ümumi forması belədir:

```
Begin
  1–ci operator;
  ...
  n–ci operator;
end;
```

Operatorun tərkibinə daxil olan operatorların sayından asılı olmayaraq tərkibli operator bir operator kimi qəbul edilir. Bu operator o hallarda istifadə olunur ki, hər hansı bir operatorun konstruksiyasında yalnız bir operator yazmağa icazə verilir, lakin, məsələnin məntiqinə görə isə orada bir neçə operatorun yazılması tələb olunur. Tərkibli operator adətən dövrü və şərti keçid operatorlarında istifadə olunur. Məsələnin məntiqindən asılı olaraq `end` operatorundan sonra nöqtəli–vergül işarəsi yazılmaya da bilər (bu, adətən `if` operatorunda belə olur).

Misal.

```
if m<n then
 begin
   r:=m mod n;
   n:=m;
   m:=r;
 end { burada nöqtəli–vergül işarəsi yazılmır }
else
 begin
   r:=m div n
```

```
    m:=sqr(r);
    n:=r;
  end;
```

Tərkibli operatorlar bir–birinin daxilində də yerləşə bilər.

5.5.2. Şərti operator

Şərti operatorun ümumi forması belədir:

if *şərt* **then** *1–ci operator* **else** *2–ci operator***;**

Operator icra olunduqda məntiqi tipli şərt yoxlanır: onun nəticəsi *True* (doğru) olarsa, *1–ci operator*, *False* (yalan) olduqda isə *2–ci operator* icra olunur. Xüsusi halda, `else` sözü və *2–ci operator* olmaya da bilər. Hər iki operator tərkibli operator ola bilər.

Müxtəlif proqramlarda `if` operatorunun belə yazılış formasına da tez–tez rast gəlinir:

if *şərt* **then** *1–ci operator***;** *2–ci operator***;**

Burada faktiki olaraq iki operator yazılmışdır. Belə ki, nöqtəli–vergül işarəsi `if` operatorunu başa çatdırır və sonra yeni operator icra olunmağa başlayır. Ona görə də şərtin ödənib–ödənməməsindən asılı olmayaraq *2–ci opreator* həmişə icra olunacaqdır.

Şərti operatorlarda məntiqi əməliyyatlardan da istifadə etmək olar. Bu halda operatorun ümumi forması belə olacaqdır:

if *məntiqi ifadə* **then** *1–ci operator* **else** *2–ci operator***;**

Bu halda, operator icra olunduqda, məntiqi ifadənin nəticəsi *True* (doğru) olarsa, *1–ci operator*, *False* (yalan) olduqda isə *2–ci operator* icra olunur. Xüsusi halda, `else` sözü və *2–ci operator* olmaya da bilər. Hər iki operator tərkibli operator ola bilər.

Misal.

```
a:=49;
b:=25;
if a>b then y:=sqrt(a) else y:=sqrt(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=7` olacaqdır.

Misal.

```
a:=49;
b:=25;
if a<b then y:=sqrt(a) else y:=sqrt(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=5` olacaqdır.

Misal.

```
a:=49;
b:=5;
if a<b then y:=sqrt(a);y:=sqr(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=25` olacaqdır.

Misal.

```
a:=3;
b:=8;
if (a=3) or (b=4) then y:=sqr(a) else
y:=a+b;
writeln(y);
```

Bu proqram fraqmentində `a=3` *və ya* `b=4` (`(a=3) or (b=4)`) olarsa, onda $y=a^2$, digər hallarda isə $y=a+b$ hesablanır, baxdığımız hal üçün proqram fraqmentinin nəticəsi `y=9` olacaqdır.

Misal.

```
a:=3;
b:=8;
if (a=3) and (b=4) then y:=sqr(a) else y:=a+b;
```

```
writeln(y);
```

Bu proqram fraqmentində `a=3` *və* `b=4` (`(a=3) and (b=4)`) olarsa, onda $y=a^2$, digər hallarda isə $y=a+b$ hesablanır, baxdığımız hal üçün proqram fraqmentinin nəticəsi `y=11` olacaqdır.

Turbo Pascal dilində bir `if` operatoru daxilində bir neçə `if` operatoru yazmaq mümkündür. Bu isə çox mürəkkəb şərtləri yerinə yetirməyə imkan verir. Bu Pascal dilinin üstün cəhətlərindən biridir. Bir–birinin daxilində yerləşmiş `if` konstruksiyaları aşağıdakı kimidir:

if *1–ci şərt*
 then
  if *2–ci şərt*
   then
    2–ci operator
     else *1–ci operator***;**

`if` operatorunun belə yazılışı aşağıdakı iki mənada başa düşülə bilər:

1.
if *1–ci şərt* **then**
 begin
  if *2–ci şərt* **then** *2–ci operator*
  else *1–ci operator***;**
 end;

2.
if *1–ci şərt* **then**
 begin
  if *2–ci şərt* **then** *2–ci operator*
 end
  else *1–ci operator***;**

Pascal kompilyatoru birinci mənanı daha düzgün hesab edir. Belə ki, hər bir `else` sözünə ən yaxın `if` operatoru uyğun gəlir. Ümumiyyətlə, bir–birinin daxilində yerləşən `if`

operatorlarını qarışıq salmamaq üçün, onları `begin` və `end` mötərizə operatorları daxilində yazmaq məsləhət görülür.

Üç və daha çox budaqlanma yaratmaq üçün `if` operatorlarını bir–birinin daxilində yazmaq lazımdır. Bu halda `if` operatorunun aşağıdakı konstruksiyalarına icazə verilə bilər:

1.

```
if 1–ci şərt then
   if 2–ci şərt then
      if 3–cü şərt then
          ...
           if n–ci şərt then n–ci operator
                          else 1–ci operator;
```

2.

```
if 1–ci şərt then 1–ci operator
  else
    if 2–ci şərt then 2–ci operator
     else
       if 3–cü şərt then 3–cü operator
          ...
           else
             if n–ci şərt then n–ci operator;
```

Hər iki konstruksiyada `else` sözü özündən əvvəlki ən yaxın `if` operatoruna uyğun gəlir.

Misal. İki ədədin bölünməsindən alınan qisməti tapmalı.

```
program qismet;
uses crt;
label son;
var
 x,y,nat:integer;
begin
 write('Bölünəni daxil edin');
 readln(x);
 write('Böləni daxil edin');
 readln(y);
```

```
  if y=0 then
  begin
    write('Sıfra bölmə');
    goto son;
  end;
  nat:=x div y;
  writeln( 'Qismət =' , nat);
  son:  ;        { Boş operator }
  Readln
end.
```

Misal. $ax^2+bx+c=0$ kvadrat tənliyinin köklərinin tapılması.

```
program kvadr_tenlik;
uses crt;
var
  a,b,c,d,x,x1,x2:real;
begin
  writeln(' a,b,c  -ni daxil edin');
  read (a,b,c);
  d:=sqr(b) -4*a*c;
  if d>0 then
  begin
    x1:=((-b+sqrt(d))/(2*a);
    x2:=((-b-sqrt(d))/(2*a);
    writeln('x1=',x1,'x2 =',x2);
  end    { Burada ; işarəsi yazmaq olmaz }
  else
  if d=0 then
  begin
    x:=-b/(2*a);
    writeln('köklər eynidir','x1=x2=',x);
  end    { Burada ; işarəsi yazmaq olmaz }
  else
  writeln(' köklər xəyalidir ');
  Readln
end.
```

Bu proqramda kvadrat tənliyin həlli alqoritmi, əslində, yalnız bir `if` operatoru ilə yerinə yetirilmişdir. Belə ki, əvvəlcə `d>0` şərti yoxlanır, şərt ödəndikdə, tənliyin hər iki kökü hesablanır və nəticə ekrana çıxarılır, əks halda `d=0` şərti yoxlanır və şərt ödəndikdə, tənliyin kökü hesablanır və köklərin eyni olması haqqında məlumat ekrana çıxarılır. Şərt ödənmədikdə isə sonuncu hal baş verir, yəni tənliyin həqiqi kökləri mövcud olmur və bu barədə məlumat ekranda təsvir olunur. Kvadrat tənliyin həlli alqoritmini hər üç şərti ayrı–ayrılıqda yoxlamaqla da yerinə yetirmək olar. Bu halda `if` operatorunu üç dəfə yazmaq lazımdır. Alqoritmin bu variantda proqramını belə yaza bilərik:

```
program kvadr_tenlik;
uses crt;
var
 a,b,c,d,x,x1,x2:real;
begin
 writeln(' a,b,c –ni daxil edin');
 read (a,b,c);
 d:=sqr(b) –4*a*c;
 if d>0 then
 begin
  x1:= (( –b+sqrt(d))/(2*a);
  x2:= (( –b–sqrt(d))/(2*a);
  writeln('x1=',x1,'x2 =',x2);
 end;
 if d=0 then
 begin
  x:= –b/(2*a);
  writeln(‘köklər eynidir’,’x1=x2=’,x);
 end;
 if d<0 then writeln('köklər xəyalidir');
 Readln
end.
```

5.5.3. Seçim operatoru

Variantların sayı çox olduqda `if` operatorundan deyil, **`Case`** seçim operatorundan istifadə etmək daha əlverişli olur. Bu operatorun ümumi forması belədir:

`Case` *selektor–ifadə* **`of`**

1–ci variant **:** *1–ci operator***;**
. . .
n–ci variant **:** *n–ci operator***;**

`else` *operator***;**
`end`;

Selektor–ifadə sıralı tip olmalıdır. Hər bir variant sabitlərdən və ondan iki nöqtə (**:**) işarəsi ilə ayrılan operatordan ibarətdir. Sabitlər bir–birindən vergüllə ayrılan ixtiyari sayda qiymətlərlə və ya qiymətlərin dəyişmə diapazonları ilə təsvir olunur. Dəyişmə diapazonları arasında "**. .**" işarəsi yazılır. Sabitlərin tipi *selektor–ifadənin* tipinə uyğun olmalıdır və onları `Label` operatoru ilə təsvir etmək lazım deyildir.

Operator belə icra olunur. Əvvəlcə *selektor–ifadə* hesablanır və onun qiyməti variantlardakı sabitlərlə müqayisə edilir. Əgər selektorun qiyməti sabitlərin hər hansı birinə bərabər olarsa və ya göstərilmiş diapazona daxil olarsa, onda həmin varianta uyğun operator icra olunur və operator öz işini dayandırır. Əgər *selektorun* qiyməti heç bir sabitlə üst–üstə düşməzsə, onda `else` sözündən sonrakı operator (əgər varsa) icra olunur.

Misal.

```
program secim_operat;
uses crt;
var
```

```
  abituriyent:string;
  bal:word;
begin
  read(bal);
  Case bal of
    700         : abituriyent:= '1-ci yer';
    670,680,690: abituriyent:= '2-ci yer';
    300..650    : abituriyent:= 'Tələbə';
    200..299    : abituriyent:= 'Musabiqə';
  else
  abituriyent:='Keçmir'
  end;
  writeln('abituriyent=', abituriyent);
  Readln
end.
```

Bu proqrama görə `bal` dəyişəninin (selektor–ifadə) qiyməti 700 olduqda abituriyentə 1–ci yer, 670, 680 və ya 690 olduqda 2–ci yer verilir; bal dəyişəninin qiyməti 300–dən 650–yə kimi istənilən tam ədəd aldıqda o, tələbə olur, 200–dən 299–a kimi istənilən tam ədəd aldıqda o, müsabiqəyə göndərilir və nəhayət bütün digər qiymətlərdə, o qəbul olmur (əlbəttə, bu bir misaldır və Tələbə qəbulu üzrə dövlət komissiyasının qaydalarına heç bir aidiyyəti yoxdur).

Misal.

```
program secim;
uses crt;
var
  simvol:char;
begin
  readln(simvol);
  case simvol of
    '0'..'9'   : writeln('Rəqəm');
    'a'..'z'   : writeln('Kiçik hərf');
    'A'..'Z'   : writeln('Böyük hərf');
  else writeln('Başqa simvol')
  end;
  Readln
```

```
end.
```

Bu proqram daxil edilən simvolun tipini müəyyən edir.

Misal.

```
program taqim;
uses crt;
var
 apolet:word;
begin
 readln(apolet);
 case apolet of
  1000..1025 : writeln('1 -ci taqım');
  1026..1050 : writeln('2 -ci taqım');
  1051..1071 : writeln('3 -cü taqım');
  1072..1100 : writeln('4 -cü taqım');
  1101..1130 : writeln('5 -ci taqım');
 end;
 Readln
end.
```

Bu proqram yaxa nömrələrinə görə kursantların hansı taqıma mənsub olmasını müəyyənləşdirir.

5.5.4. Dövr operatorları

Dövr operatorları müəyyən operatorlar qrupunu dəfələrlə yerinə yetirmək üçün istifadə edilir. Təkrar olunan operatorlar qrupu dövrün *gövdəsini* təşkil edir. Üç növ dövr operatorları vardır:

- *parametrli;*
- *ilkin şərtli;*
- *son şərtli.*

Əgər dövrlərin sayı əvvəlcədən məlumdursa, onda *parametrli*, əks halda *ilkin* və ya *son şərtli* dövr operatorları tətbiq olunur.

Goto operatorunu və ya parametrsiz **Break** prosedurunu istifadə etməklə, dövrün işini istənilən zaman dayandırmaq olar.

Continue prosedurunu tətbiq etməklə isə növbəti dövrün işini dayandırıb idarəetməni dövrün sonuna vermək olar.

Dövr operatorları bir–birinin daxilində də yerləşə bilər.

5.5.4.1. Parametrli dövr operatoru

Parametrli dövr operatorunun iki forması mövcuddur:

for *parametr* **:=** *1–ci ifadə* **to** *2–ci ifadə* **do** *operator*;

və

for *parametr* **:=** *1–ci ifadə* **downto** *2–ci ifadə* **do** *operator*;

Burada, *parametr* dövrün parametridir və o, sıralı tip istənilən dəyişən ola bilər. *1–ci ifadə* və *2–ci ifadə* dövrün parametrinin aldığı başlanğıc və son qiymətlərdir; bu ifadələrin tipi parametrin tipinə uyğun olmalıdır. `do` sözündən sonrakı *operator* isə dövrün gövdəsində təkrar olunacaq operatordur. Bu operator tərkibli operator da ola bilər.

Parametrli dövr operatoru belə icra olunur. Dövrün parametri özünün başlanğıc qiymətini alır və `do` sözündən sonra yerləşən operator yerinə yetirilir. Bu proses, dövrün parametrinin qiyməti hər dəfə 1 vahid artırılmaqla, dövrün parametrinin sonuncu qiymətinə kimi təkrar olunur. Operatorun ikinci formasında isə dövrün parametrinin dəyişmə addımı (–1)–ə bərabər olur və dövrün parametrinin qiyməti ardıcıl olaraq *2–ci ifadəyə* qədər azalır.

Misal. 100–ə qədər müsbət tam ədədlərin cəminin hesablanması.

```
Program tam_eded_cemi;
Uses crt;
Const n=100;
var i,cem:integer;
```

```
begin
 cem:=0;
 for i:=1 to n do cem:=cem+i;
 writeln('100-ə qədər tam
        ədədlərin cəmi=',cem:4);
 readln
end.
```

Misal. 10–a qədər müsbət tam ədədlərin kvadratlarını, kvadrat köklərini və natural loqarifmlərini hesablayıb, nəticələri cədvəl şəklində ekrana çıxarın.

```
Program cedvel;
Uses crt;
Const n=10;
var i:integer;
    x,y,z:real;
begin
 writeln;
 writeln('Kvadratlar, köklər
  və loqarifmlər cədvəli');
 writeln;
 writeln('ədəd','  kvadrat',
 '  kvadrat kök', '  loqarifm');
 writeln;
 for i:=1 to n do
 begin
  x:=sqr(i);
  y:=sqrt(i);
  z:=ln(i);
  writeln(i:2,'    ',x:6:2,
  y:9:2,'      ',z:10:8);
 end;
 readln
end.
```

Proqramın nəticəsi şəkil 5.2 – də göstərilmişdir.

Dövr operatorları bir–birinin daxilində də yerləşə bilər. Bu halda əvvəlcə xarici dövr icra olunur, onun ardınca daxili dövr icra olunmağa başlayır. Daxili dövr tamamilə icra

olunduqdan sonra, yenidən xarici dövr icra olunmağa başlayır və s.

Misal.

```
for n:=1 to 3 do    { xarici dövr }
for k:=10 to 20 do  { daxili dövr }
begin
  lines1[n,k]:=k;
  lines2[n,k]:=n;
end;
```

```
Kvadratlar, kokler ve loqarifmler cedveli

eded  kvadrat  kvadrat kok  loqarifm

 1      1.00      1.00       0.00000000
 2      4.00      1.41       0.69314718
 3      9.00      1.73       1.09861229
 4     16.00      2.00       1.38629436
 5     25.00      2.24       1.60943791
 6     36.00      2.45       1.79175947
 7     49.00      2.65       1.94591015
 8     64.00      2.83       2.07944154
 9     81.00      3.00       2.19722458
10    100.00      3.16       2.30258509
```

Şəkil 5.2. Funksiyanın cədvəlləşdirilməsi

Bu misalda birinci `for` operatoru *xarici dövr,* ikinci `for` operatoru isə *daxili dövr* adlanır. Əvvəlcə `n=1` olur, sonra daxili dövr icra olunur, yəni `k` ardıcıl olaraq 10,11,...,20 qiymətlərini alır və hər dəfə `lines1` və `lines2` massivlərinin elementləri hesablanır. `k=20` olduqda daxili dövr sonuncu dəfə hesablanır və bundan sonra, yenidən xarici dövr icra olunmağa başlayır, yəni `n=2` olur, yenidən daxili dövr icra olunur və s. Beləliklə, `lines1` və `lines2` massivlərinin elementləri 3∗20=60 dəfə hesablanır, yəni dövrlər sayı 3∗20=60 olur: `n`–in hər bir qiymətində (`n=1,2,3`) `k` parametri 10–dan 20–yə kimi qiymət alır.

Massivlərin elementlərini daxil etmək üçün adətən parametrli dövr operatorundan istifadə edilir. Birölçülü massivlərin elementlərini daxil etmək üçün bu operator

```
for i:=1 to n do readln(a[i]);
```

kimi, ikiölçülü massivlərin elementlərini daxil etmək üçün isə

```
for i:=1 to n do
for j:=1 to m readln(a[i,j]);
```

kimi yazılır.

Qeyd. Dövrün parametrini dövrün gövdəsində dəyişdirmək olmaz.

5.5.4.2. İlkin şərtli dövr operatoru

İlkin şərtli dövrlərin strukturu şəkil 5.3 – də göstərilmişdir. Belə dövrlərdə əvvəlcə şərt yoxlanır, şərt ödəndikdə dövr icra olunur, əks halda dövr yerinə yetirilmir. Bu operatorun ümumi forması belədir:

`While` *şərt* **`do`** *operator*;

İlkin şərtli dövr operatorunu o zaman tətbiq etmək məqsədəuyğundur ki, dövrlərin sayı əvvəlcədən məlum olmur və dövr, ümumiyyətlə, yerinə yetirilməyə də bilər. Bu operator icra olunduqda əvvəlcə məntiqi tipli şərt yoxlanır və onun nəticəsi *True* (doğru) olarsa, dövrün gövdəsi təkrarlanır. Şərt *False* (yalan) qiyməti alan kimi dövrün yerinə yetirilməsi dayandırılır. Dövrün gövdəsində şərtə təsir edilməlidir ki, dövr sonsuz təkrar olunmasın.

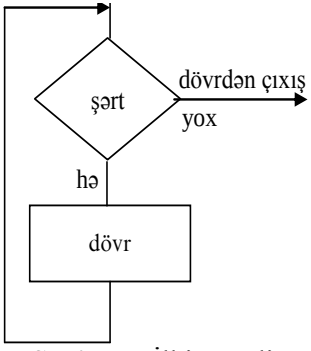
Əgər operator

`While` `True` **`do`** *operator*;

şəklində yazılarsa, onda dövr sonsuz təkrarlanacaqdır.

Misal.

```
Program İlkin_devr;
Uses crt;
var i:integer;
    sum:real;
begin
 sum:=0;
 i:=1;
 while i<=100 do
  begin
  sum:=sum+sqrt(i);
  i:=i+1;
  end;
end.
```



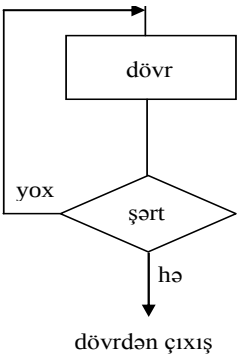
Şəkil 5.3. İlkin şərtli dövrlərin strukturu

Burada, `i` dəyişəninə əvvəlcə 1 qiyməti verilir və o, 100–dən kiçik olduğu üçün, dövrün gövdəsi hesablanır, sonra `i`–nin qiyməti 1 vahid artırılaraq dövr yenidən təkrarlanır. Bu proses `i` >100 şərti ödənənə qədər davam edir və proqram 1–dən 100–ə kimi tam ədədlərin kvadrat kökləri cəmini hesablayır.

5.5.4.3. Son şərtli dövr operatoru

Son şərtli dövrlərin strukturu şəkil 5.4 – də göstərilmişdir. Belə dövrlərdə əvvəlcə dövr yerinə yetirilir, sonra şərt yoxlanılır. Şərt ödənmədikdə, dövr yenidən icra olunur, şərt ödəndikdə isə dövr öz işini dayandırır.

İlkin şərtli dövr operatorundan fərqli olaraq, son şərtli dövr operatorunda şərt dövrün gövdəsindən sonra yoxlandığı üçün, dövrün gövdəsi hökmən bir dəfə yerinə yetirilir. Bu operatorun ümumi forması belədir:



Şəkil 5.4. Son şərtli dövrlərin strukturu

```
Repeat
  1–ci operator;
  ...
  n–ci operator;
until şərt;
```

`repeat` və `until` sözləri arasında yerləşən operatorlar dövrün gövdəsini təşkil edir. Dövrün gövdəsi bir dəfə yerinə yetirildikdən sonra, məntiqi tipli şərt yoxlanır və onun nəticəsi *False* (yalan) olduqda, dövrün gövdəsi yenidən yerinə yetirilir. Şərt *True* (doğru) qiyməti alan kimi, dövrün yerinə yetirilməsi dayandırılır. Bu operatorda da dövrün sonsuz təkrar olunmaması üçün dövrün gövdəsində şərtə təsirlər edilməlidir.

Əgər operatorun sonuncu sətri

```
until False;
```

şəklində yazılarsa, onda dövr sonsuz təkrarlanacaqdır.

Misal. İlkin şərtli dövr operatorunda həll etdiyimiz misalı son şərtli dövr operatoru vasitəsilə həll edək.

```
Program Son_devr;
Uses crt;
var i:integer;
    sum:real;
begin
 sum:=0;
 i:=1;
 Repeat
  sum:=sum+sqrt(i);
  i:=i+1;
 Until i>100;
end.
```

5.5.4.4. Daxilolma operatoru

Daxilolma operatoru adətən tərkibli adların, o cümlədən, yazıların sahələrinə daha asan daxil olmaq üçün istifadə edilir. Bilirik ki, yazı sahələrinə müraciət etmək üçün yazının öz adını və ondan nöqtə ilə ayrılan yazı sahəsinin adını göstərmək lazımdır. Daxilolma operatoru ilə isə yazı sahəsinə birbaşa müraciət etmək olar. Daxilolma operatorunun ümumi forması belədir:

`With` *obyektin adı* **`do`** *operator*;

Qeyd olunmuş yazıları öyrəndikdə araşdırdığımız misalı bu operatorun köməyi ilə həll edək.

Misal.

```
With Persone do
 begin
  Name:='Abdullayev R.K.';
  Address:='Səməd Vurğun küçəsi,31';
  Married:=True;
  Salary:=500;
 end;
```

Göründüyü kimi, `Persone` yazısının sahələrinə birbaşa müraciət edilmişdir.

Altıncı fəsil

HESABLAMA PROSESLƏRİNİN PROQRAMLAŞDIRILMASI

Bu fəsildə xətti, budaqlanan və dövrü hesablama proseslərinin proqramlaşdırılmasına, simvol və sətirlərin emalına aid ən müxtəlif xarakterli məsələlər proqramlaşdırılacaqdır. Seçilmiş məsələlər proqramlaşdırma üçün səciyyəvi xarakterli məsələlər olduğundan, belə məsələlərin proqramlaşdırılma texnikasının öyrənilməsi gələcəkdə daha mürəkkəb məsələləri proqramlaşdırmaq üçün zəruri vərdişlər əldə etməyə imkan verəcəkdir. Mürəkkəb məsələlərin proqramları əsasən belə proqram konstruksiyalarından ibarət olur.

6.1. Xətti və budaqlanan hesablama proseslərinin proqramlaşdırılması

Misal. v başlanğıc sürəti və α bucağı altında havaya atılmış cismin hündürlüyü

$$H = \frac{v^2}{2g} Sin^2(\alpha)$$

düsturu ilə hesablanır. Hündürlüyü hesablamaq üçün proqram yazın.

İlkin verilənlər kimi v=10 *m/san* və α=20 *dərəcə* qəbul edilmişdir. Proqramlaşdırmada triqonometrik funksiyaların

arqumentləri dərəcə ilə deyil, radianla verilməlidir. Ona görə də proqram özü bucağı radiana çevirəcəkdir (`Alfa_rad` dəyişəni).

```
Program Hundur;
Uses Crt;
Const g=9.8;
      v=10;
      Alfa=20;
Var Alfa_rad,h:real;
begin
 ClrScr;
 Alfa_rad:=Alfa*pi/180;
 H:=sqr(V*sin(Alfa_rad))/(2*g);
 Writeln('Alfa=',Alfa,'dərəcə və V=',V,
 'm/s olduqda h=',H,'m olur');
 readln
end.
```

Misal.

$a = e^{x^2+b}$, $b = \sqrt{1+x^2}$ olduqda

$y = \sin(x) + ax^2 + b\sqrt{|x+1|}$

funksiyasını hesablayın.

```
Program funksiya;
uses crt;
var a,b,x,y:real;
begin
 readln(x);
 a:=exp(x*x+b);
 b:=sqrt(1+sqr(x));
 y:=sin(x)+a*x*x+b*sqrt(abs(x+1));
 write(' ':5,'x=',x:5:2,' ':3,'y=',y:5:2);
 readln
end.
```

Misal.

$$y=\begin{cases}(x-b)/|x-0{,}79|+b^2 & , \quad 0{,}9\le(x-0{,}79)\\ \sqrt{\left|(x-b)/\left[(x-a)^2+b\right]\right|}+ab & , \quad 0{,}6\le(x-0{,}79)<0{,}9\\ \ln(x+ab) & , \quad x-0{,}79<0{,}6\end{cases}$$

hesablayın.

Burada,

$$a=e^{x+4} \quad , \quad b=\sqrt{ax}\,.$$

```
Program budaq1;
uses crt;
var a,b,x,y,z:real;
begin

 readln(x);
 a:=exp(x+4);
 b:=sqrt(a*x);
 z:=x-0.79;

 if z>=0.9 then y:=(x-b)/abs(z)+sqr(b);

 if (z>=0.6) and (z<0.9) then
 y:=sqrt(abs((x-b)/((sqr(x-a)+b))))+a*b;

 if z<0.6 then y:=ln(x+a*b);
 write(' ':5,'z=',z:5:2,' ':3,'y=',y:5:2);

 readln

end.
```

Bu məsələdə `x-0.79` ifadəsi bir neçə dəfə hesablanmalıdır. Proqramlaşdırmada çalışmaq lazımdır ki, kompüterin yaddaşından və sürətindən israfçılıqla istifadə

edilməsin, ona görə də tərtib etdiyimiz proqramda bu ifadə bir dəfə hesablanaraq `z` dəyişəninə mənimsədilmişdir.

Misal.

$$y=\begin{cases} ax^2+bx+c, & x=1 \\ ax+b, & x=2 \\ bx^2+c, & x=3 \\ cx, & x=4 \\ (ax+b)/(cx+a), & x=5 \\ e^{cx}, & x=6 \quad olduqda \\ \ln\left|ax^2+bx+c\right|, & diger\ hallarda \end{cases}$$

funksiyasını hesablayın.

Burada, $a=\sin^2 x$, $b=\cos x+\sin x$, $c=e^{ax+b}$.

Bu məsələdə variantların sayı çox olduğu üçün `if` operatorundan istifadə etmək əlverişli deyildir. Ona görə də `case` operatorunu tətbiq edəcəyik.

```
Program funks;
uses crt;
var a,b,c,y:real;
    x:byte;
begin
 readln(x);
 a:=sqr(sin(x));
 b:=cos(x)+sin(x);
 c:=exp(a*x+b);

 Case x of
  1: y:=a*x*x+b+x+c;
  2: y:=a*x+b;
  3: y:=b*x*x+c;
  4: y:=c*x;
  5: y:=(a*x+b)/(c*x+a);
```

```
      6: y:=exp(c*x);
      else
      y:=ln(abs(a*x*x+b*x+c));
    end;

    writeln(' ':5,'x=',x:5,' ':3,'y=',y:5:2);
    readln
  end.
```

Misal. Klaviaturadan daxil edilən *n* tam ədədinin ($0 \le n \le 15$) onaltılıq say sisteminə çevrilməsi.

```
program cevirme;
uses crt;
var
  n : integer;
  ch : char;

begin

  write('n=');
  readln(n);
  if (n>0) and (n<=15) then

  begin

    if n<10 then
    ch:=chr(ord('0')+n)
    else
    ch:=chr(ord('A')+n-10);
    writeln('n=',ch);

  end

    else
      writeln('Səhvdir');
  readln

end.
```

Proqramın alqoritmi simvolların **ASCII** kodlarına əsaslanır. **ASCII** simvollar cədvəlinə əsasən 0, 1, 2, . . . , 9 ədədlərinin kodları uyğun olaraq 48,49, . . .,57–dir. Ona görə də proqramda, `n<10` olduqda, `ch:=chr(ord('0')+n)` yazılmışdır, yəni, məsələn, *n=4* üçün `ch=chr`*(48+4)*`=chr`*(52)=4* olacaqdır. *4* isə onaltılıq say sistemində də *4* kimi yazılır. `n>10` olduqda, `ch:=chr(ord('A')+n-10)` yazılmışdır, yəni, məsələn, *n=13* üçün `ch=chr`*(97+13–10)*`=chr`*(100)* olacaqdır ki, 100 ədədinə də uyğun kod `chr`*(100)=D* olacaqdır, *13* ədədi isə onaltılıq say sistemində *D* kimi yazılır.

6.2. Dövrü hesablama proseslərinin proqramlaşdırılması

Proqramlaşdırma texnologiyasında bir sıra hesablama konstruksiyalarından istifadə edilir ki, bu konstruksiyalar hansı proqramlaşdırma dilinin tətbiq edilməsindən asılı olmayaraq, demək olar ki, eyni struktura malik olur. Belə konstruksiyalardan ən vacibləri cəmləmə, hasil və sayğac alqoritmləridir. Bu alqoritmlərə ayrılıqda baxaq.

Cəmləmə alqoritmi. Cəmləmə əməliyyatını yerinə yetirmək üçün, əvvəlcə cəmi yadda saxlayacaq hər hansı bir dəyişənə sıfır qiyməti mənimsədilir, sonra isə dövr təşkil olunaraq həmin dəyişənin üzərinə cəmlənəcək dəyişən və ya massivin elementləri əlavə edilir:

```
...
s:=0;
For i:=1 to n do
s:=s+a[i];
...
```

Dövrün birinci addımında, yəni `i=1` olduqda, `s` adlı xanada cəmin başlanğıc qiyməti – `a[1]` yerləşəcək. İkinci addımda, `i=2` olduqda, `s` xanasında artıq `a[1]` qiyməti

olduğu üçün, onun üzərinə `a[2]` əlavə olunacaqdır. Beləliklə, ikinci addımda `s` xanasında `a[1]+a[2]` cəmi olacaqdır. `n` – ci addımda isə massivin bütün elementləri cəmlənəcəkdir.

Hasil alqoritmi. Cəmləmə alqoritminə analoji olaraq, hasili yadda saxlayacaq hər hansı bir dəyişənə vahid qiyməti mənimsədilir, sonra isə dövr təşkil olunaraq hasili tapılacaq dəyişən və ya massivin elementləri həmin dəyişənə vurulur.

```
...
h:=1;
For i:=1 to n do
h:=h*a[i];
...
```

Bu alqoritmin iş prinsipi cəmləmə alqoritminin işləmə prinsipi ilə eynidir.

Sayğacların yaradılması. Sayğac proqramlaşdırmada ən vacib konstruksiyadır. Demək olar ki, elə bir proqram yoxdur ki, orada sayğaclardan istifadə edilməsin. Sayğac vasitəsilə bu və ya digər elementlərin sayı tapılır. Sayğacların strukturu belədir:

```
...
k:=0;
for i:=1 to n do
  if a[i]>0 then k:=k+1;
...
```

və ya

```
...
k:=0;
for i:=1 to n do
  if a[i]>0 then inc(k);
...
```

Bu sayğacla `a[i]` massivinin müsbət elementlərinin sayı (`k`) tapılır. Mənfi elementlərin sayını hesablayan sayğac belədir:

```
...
k:=0;
for i:=1 to n do
  if a[i]<0 then k:=k+1;
...
```

`a ≤ x[i] ≤ b` aralığında yerləşən elementlərin sayını hesablayan sayğac belədir:

```
...
k:=0;
for i:=1 to n do
  if (x[i] >=a) and (x[i] <=b) then
inc(k);
...
```

Misal. $S = \sum_{i=1}^{n} \frac{1}{i^2}$ cəminin hesablanması.

```
program cem;
uses crt;
var
 i,n:word;
 t,add,cem:real;
begin
 write('n –i daxil edin n = ');
 readln(n);
 cem:=0;
 for i:= 1 to n do
 begin
  t:=1.0/i;
  add:=sqr(t);
  cem:=cem+add;
 end;
 writeln('cəm = ', cem);
 write('enter klavişini basin:');
 readln;
end.
```

Cəmləmə əməliyyatını yerinə yetirmək üçün dövr təşkil etməzdən əvvəl, hər hansı bir dəyişənə sıfır qiyməti mənimsədərək (`cem:=0;`) dövrün daxilində onun üzərinə cəmlənəcək parametr əlavə edilmişdir (`cem:=cem+add;`).

Misal. $\sum_{i=1}^{1000}\frac{1}{n^5}$ cəmini düz və əks istiqamətlərdə hesablayıb, onların fərqini tapın.

```
program duz_eks_cem;
uses crt;
var x,duzcem,ekscem :real;
    k :word;
begin
 clrscr;
       { düz istiqamətdə cəmləmə }
 duzcem:=0.0;
 for k:=1 to 1000 do
 begin
  x:=k;
  duzcem:=duzcem+1.0/(exp(5.0*ln(x)));
 end;
       { əks istiqamətdə cəmləmə }
 ekscem:=0.0;
 for k:=1000 downto 1 do
 begin
  x:=k;
  ekscem:=ekscem+1.0/(exp(5.0*ln(x)));
 end;
 Writeln('irəli cəm     =',duzcem);
 writeln('geriyə cəm    =',ekscem);
 writeln('cəmlərin fərqi=',duzcem-ekscem);
 readln
end.
```

Bu proqramdan çox qəribə bir nəticə çıxır: məlumdur ki, toplananların yerini dəyişdikdə cəm dəyişmir, lakin, bu proqramdan alınan nəticəyə baxdıqda görürük ki,

hesablamanı düz və əks istiqamətlərdə apardıqda cəm dəyişir. Məsələ ondadır ki, kompüter eyni tərtibli dəyişənləri topladıqda daha kiçik xətalara yol verir. Ona görə də bu misalın həllində əks istiqamətdə alınan cavab daha düzgündür. Çünki, məsələn, *n=1,2,3* qiymətlərində $1+1/2^5+1/3^5$ toplananları bir–birindən çox fərqlənir. Lakin, məsələn, *n=1000, 999, 998* qiymətlərində $1/1000^5+1/999^5+1/998^5$ toplananları isə bir–birindən çox fərqlənmir.

Misal. *F(x)=x/(1+x)* funksiyasını cədvəlləşdirin.

```
program cedvel;
uses wincrt;
var x,f:real;
    k:word;
begin
 clrscr;
 x:=0.0;
 writeln('f(x)=x/(1+x)
 funksiyasının qiymətlər cədvəli');
 writeln;
 Writeln('sıra N','x':10,'f(x)':20);
 writeln;
 for k:=0 to 50 do
  begin
    f:=x/(1.0+x);
    writeln(' ',k,' ',x:12:4, f:20:10);
    x:=x+0.1;
    if k mod 10=9 then readln;
  end;
 readln
end.
```

Bu proqram hər 10 sətirdən sonra dayanır. Bunu `if k mod 10=9 then readln;` sətri təmin edir. Parametrsiz `readln;` proseduru *Enter* klavişinin basılmasını gözləyir və

bu klaviş basıldıqda hesablama davam etdirilir. Proqramın nəticəsinin bir hissəsi şəkil 6.1 – də göstərilmişdir:

Misal. *n* sayda tam ədədlər içərisindən 3 ədədinə tam bölünməyən ədədləri, onların sayı və cəmini tapmalı.

```
Program uche_bolme;
Uses wincrt;
Const n=300;
var k,i:word;
    s:longint;
begin
 writeln;
 K:=0;s:=0;
 For i:=1 to n do
 If not (i mod 3=0)
 then
  Begin
    Inc(k);
    s:=s+i;
    Write (' ',i:3);
    if k mod 10 = 9 then writeln;
  end;
 writeln;
 writeln;
 writeln('  3-ə bölünməyən
        ədədlərin sayı=',k:4);
 writeln('  3-ə bölünməyən
        ədədlərin cəmi=',s:10);
 readln
end.
```

f(x)=x/(1+x) funksiyasinin qiymetler cedveli

sira N	x	f(x)
0	0.0000	0.0000000000
1	0.1000	0.0909090909
2	0.2000	0.1666666667
3	0.3000	0.2307692308
4	0.4000	0.2857142857
5	0.5000	0.3333333333
6	0.6000	0.3750000000
7	0.7000	0.4117647059
8	0.8000	0.4444444444
9	0.9000	0.4736842105

Şəkil 6.1. Funksiyanın cədvəlləşdirilməsi

Nümunə üçün `n=300` qəbul edilmişdir. Proqram belə işləyir. 3 ədədinə tam bölünməyən ədədlərin cəmini tapmaq üçün `s` dəyişəninə, ədədlərin sayını tapmaq üçün isə `k` dəyişəninə sıfır qiymətləri mənimsədilmişdir. Sonra, dövr təşkil olunaraq, ədədlərin 3–ə bölünməsindən alınan qalıq tapılır. Əgər qalıq sıfra bərabər deyildirsə, yəni `if not(i mod 3=0)`, onda `k` sayğacının üzərinə vahid, `s` dəyişəninin

üzərinə isə həmin ədəd əlavə olunur. İkinci `if` operatoru hər

```
  1   2   4   5   7   8  10  11  13
 14  16  17  19  20  22  23  25  26  28
 29  31  32  34  35  37  38  40  41  43
 44  46  47  49  50  52  53  55  56  58
 59  61  62  64  65  67  68  70  71  73
 74  76  77  79  80  82  83  85  86  88
 89  91  92  94  95  97  98 100 101 103
104 106 107 109 110 112 113 115 116 118
119 121 122 124 125 127 128 130 131 133
134 136 137 139 140 142 143 145 146 148
149 151 152 154 155 157 158 160 161 163
164 166 167 169 170 172 173 175 176 178
179 181 182 184 185 187 188 190 191 193
194 196 197 199 200 202 203 205 206 208
209 211 212 214 215 217 218 220 221 223
224 226 227 229 230 232 233 235 236 238
239 241 242 244 245 247 248 250 251 253
254 256 257 259 260 262 263 265 266 268
269 271 272 274 275 277 278 280 281 283
284 286 287 289 290 292 293 295 296 298
299

 3-e bolunmeyen ededlerin sayi= 200
 3-e bolunmeyen ededlerin cemi=     30000
```

Şəkil 6.2. Üç ədədinə bölünməyən ədədlər

10 ədəddən bir yeni sətrə keçmək üçün yazılmışdır. Proqramın nəticəsi şəkil 6.2 – də göstərilmişdir.

6.3. Birölçülü massivlər üzərində əməliyyatlar

Massivlərin elementlərini daxil etmək üçün sadəcə olaraq `read` və ya `readln` prosedurundan istifadə etmək mümkün olmur. Çünki bu prosedurla hər dəyişənə yalnız bir qiymət daxil etmək olar, massivlərin isə elementləri çoxdur. Ona görə də massivlərin elementlərini daxil etmək üçün adətən parametrli dövr operatorundan istifadə edilir:

```
for i:=1 to n do readln(a[i]);
```

Misal. *A(15)* massivinin elementlərini ekrandan daxil edin. *B(15)* massivinin elementlərini elə qiymətləndirin ki,

cüt nömrəli elementlər *A*–nın uyğun elementlərindən 3 dəfə çox, tək nömrələr isə 5 vahid az olsun.

```
program tek_cut_element;
uses crt;
const n=15;
var
   A,B: array [1..n] of real;
   i:byte;
begin
 writeln('Massivin elementlərini
         daxil edin');
 for i:= 1 to n do readln(A[i]);
 clrscr;
 for i:= 1 to n do
 if Odd(i) then B[i]:=A[i]+5
 else B[i]:=A[i]*3;
 for i:= 1 to n do
 writeln(B[i]);
 readln
end.
```

Təqdim olunmuş bu proqramda tək və cüt nömrəli elementləri müəyyən etmək üçün `Odd` funksiyasından istifadə edilmişdir. Xatırlayaq ki, arqumenti tək ədəd olduqda, bu funksiyanın nəticəsi *True*, cüt ədəd olduqda isə *False* olur.

Misal. *n* sayda elementdən ibarət massivin elementlərinin cəminin hesablanması.

```
program mas_sem;
uses crt;
const n=25;
var
   a:array[1..n] of real;
   s:real;
   i:integer;
begin
 s:=0;
```

```
 i:=1;
 while i<=n do
 begin
  s:=s+a[i];
  i:=i+1;
 end;
 writeln('Cəm =', s);
 readln
end.
```

Bu proqramla, artıq bizə məlum olan alqoritm üzrə cəmləmə əməliyyatı yerinə yetirilmişdir, fərq ondadır ki, burada ilkin şərtli dövr operatorundan istifadə olunmuşdur.

Misal. Tam ədədlərdən təşkil olunmuş massivin birinci mənfi elementinin axtarılması.

```
program menfi_el;
uses crt;
const n=15;
      yes:boolean = False;
var
 mas:array[1..n] of integer;
 i:byte;
begin
 writeln('Massivin elementlərini
           daxil edin');
 for i:= 1 to n do
 begin
  write('mas[',i,']=');
  readln(mas[i]);
 end;
 for i:= 1 to n do
 begin
  if mas[i]>=0 then continue;
  writeln('Birinci mənfi element=',
         mas[i], 'Nömrə= ',i);
  yes:=True;
  break;
```

```
    end;
    if not yes then
    writeln('Mənfi element yoxdur');
    readln
  end.
```

Bu proqramda, dövr daxilində massivin elementlərinin müsbət olması yoxlanır: əgər element müsbətdirsə, onda `continue` proseduru ilə növbəti elementin yoxlanmasına keçilir. Mənfi element rast gəldikdə isə `yes` dəyişəninə *True* qiyməti mənimsədilməklə, `break` proseduru ilə dövrün işi dayandırılır. Əgər `yes` dəyişəninin qiyməti *False* olarsa, onda proqram massivin mənfi elementinin olmaması haqqında məlumat verir (`if not yes then writeln('Mənfi element yoxdur');`).

Misal. *X(n)* massivinin ən böyük elementinin tapılması.

```
program max_element;
uses crt;
const n=20;
var
   x:array[1..n] of real;
   i: integer; M: real;
begin
 writeln('Massivin elementlərini
          daxil edin');
 for i:= 1 to n do
  begin
   write('X[',i,']=');
   readln(X[i]);
  end;
 M:=X[1];
 for i:= 2 to n do
 if M < X[i] then M:=X[i];
 writeln('Massivin ən böyük
         elementi = ', M);
 readln
end.
```

Massivin ən böyük elementinin tapılması alqoritmi belədir: massivin birinci elementi hər hansı bir dəyişənə mənimsədilir (`M:=X[1];`) və dövr daxilində bu element növbəti elementlərlə müqayisə edilir. Hansı element böyük olarsa, o, element həmin dəyişənə mənimsədilir (`if M<X[i] then M:=X[i];`). Bu proses ən böyük element tapılana qədər davam etdirilir.

Kompüter texnologiyasında və proqramlaşdırma texnikasında ən çox istifadə edilən alqoritmlərdən biri massivlərin elementlərinin nizamlanması (elementlərin artma və ya azalma sırası ilə düzülməsi) alqoritmidir. Elementlərin nizamlanması üçün bir neçə metod mövcuddur.

Mövcud metodlardan biri belədir. Verilmiş ilkin ədədlərdən ən kiçiyi tapılır və o yeni massivin birinci mövqeyinə yazılır, ilkin massivdən isə həmin ədəd yox edilir. İlkin massivin yerdə qalan elementləri içərisindən ən kiçiyi tapılır, yeni massivin ikinci mövqeyinə yazılmaqla, ilkin massivdən həmin ədəd yox edilir. Növbəti elementlər üçün bu prosesi davam etdirməklə, sonuncu ən böyük ədədi alacağıq ki, bu ədəd yeni massivin sonuncu elementi olacaqdır. Belə alqoritmdə biz faktiki olaraq eyniölçülü iki massivdən – ilkin və yeni massivlərdən istifadə edirik. Proqramlaşdırmada isə belə israfçılığa yol vermək olmaz (massiv çox böyük ölçülü ola bilər). İndi bu alqoritmi təkmilləşdirək. Xətti nizamlama adlanan aşağıdakı alqoritmi tərtib edək.

1. a_i (i = 1, . . . , n) massivinin ən kiçik elementini taparaq onu a_1 elementinə yazaq. Bunun üçün a_1 elementini verilmiş ilkin massivin bütün növbəti elementləri ilə müqayisə edək. Əgər hər hansı element a_1–dən kiçik olarsa, onda onların yerlərini dəyişərək müqayisəetməni sonuncu elementə qədər davam etdirmək lazımdır. Belə müqayisə zamanı, əgər hər hansı element a_1 –dən böyük və ya ona bərabər olarsa, onda elementin yerini dəyişmədən növbəti elementlə müqayisəni davam etdirmək lazımdır.

2. Birinci mövqedəki elementi nəzərdən atmaqla, göstərilən prosesi ikinci mövqedən təkrar etmək lazımdır. Başqa sözlə, yerdə qalan elementlər içərisindən göstərilən üsulla ən kiçik elementi tapıb onu a_2 elementinə yazmaq lazımdır.

3. Bu əməlləri, axırıncı element müstəsna olmaqla, bütün elementlər üçün icra etmək lazımdır.

Bu alqoritmin proqramını aşağıdakı kimi yaza bilərik.

```
program xetti_nizam;
uses crt;
const n=4;
var i:word;
    a:array[1..n] of integer;
    yeni,x:integer;
begin
  for i:=1 to n do read(a[i]);
  for i:=1 to n-1 do
  for yeni:=i+1 to n do
  if a[i]>a[yeni] then
   begin
     x:=a[i];
     a[i]:=a[yeni];
     a[yeni]:=x;
   end;
  for i:=1 to n do
   begin
     Write(a[i]:5,' ':3);
     if i=10 then write(#10#13);
   end;
  readln
end.
```

Bu proqramla ekrana hər sətirdə 10 qiymət çıxarılır. Sətirdən–sətrə keçidi

```
if i=10 then write(#10#13);
```

operatoru təmin edir. Burada **#10** və **#13** kodları uyğun olaraq *sətrin dəyişdirilməsi* və yazı makinasının *karetkasının qaytarılması* idarəedici simvollarıdır, başqa sözlə, **#10#13** kodları ilə yeni sətrə keçilir.

Tərtib etdiyimiz bu proqram massivin elementlərini yalnız artma sırası ilə düzür. Massivin elementlərini azalma sırası ilə düzmək üçün `if` operatorundakı "`>`" işarəsini "`<`" işarəsi ilə əvəz etmək lazımdır. Lakin proqramı elə tərtib etmək olar ki, o elementlərin artma və ya azalma sırası ilə düzülməsini özü seçsin. Həmin proqram belə olacaqdır:

```
program xetti_nizam;
uses crt;
const n=4;
var i:word;
    a:array[1..n] of integer;
    yeni,x:integer;
    istiqamet:string;
begin
 write('Artma yoxsa azalma?');
 read(istiqamet);
 for i:=1 to n do readln(a[i]);
 if istiqamet='artma'
 then
{ artma sırası ilə düzülüş }
   begin
    for i:=1 to n-1 do
    for yeni:=i+1 to n do
    if a[i]>a[yeni] then
     begin
       x:=a[i];
       a[i]:=a[yeni];
       a[yeni]:=x;
     end;
   end   { ; işarəsi qoymaq olmaz }
 else
{ azalma sırası ilə düzülüş }
```

```
    begin
      for i:=1 to n-1 do
      for yeni:=i+1 to n do
      if a[i]<a[yeni] then
       begin
         x:=a[i];
         a[i]:=a[yeni];
         a[yeni]:=x;
       end;
    end;
    for i:=1 to n do
     begin
       Write(a[i]:5,' ':3);
       if i=10 then write(#10#13);
     end;
  readln
end.
```

Bu proqrama sətir tipli `istiqamet` dəyişəni daxil edilmişdir ki, ona `artma` qiyməti daxil etdikdə massivin elementləri artma sırası ilə, ixtiyari fərqli qiymət daxil etdikdə isə azalma sırası ilə düzüləcəkdir.

Daha bir metoda baxaq. Bu metod "qabarcıq" metodu adlanır. Bu alqoritmdə ən kiçik ədəd elementlər içərisindən yuxarıya qalxır, sanki "üzür", ən böyük elementlər isə aşağıya hərəkət edir, sanki "batır". Metoda ona görə "qabarcıq" adı verilmişdir ki, suda hava qabarcıqlarının hərəkətinə analoji olaraq, massivin kiçik elementləri massivin başlanğıcına ("suyun səthinə") qalxır. Bu metodun mahiyyəti belədir. Məlumdur ki, massivin elementləri artma sırası ilə düzülmüşdürsə, onda hər bir element özündən sonrakı elementdən kiçik olacaqdır. Bu sadə fakt "qabarcıq" metodunun əsas ideyasını təşkil edir. Belə ki, məhz bu qayda massivlərin elementlərini yeni üsulla müqayisə etməyə əsas verir, yəni kifayətdir ki, yalnız bir cüt qonşu ədədləri müqayisə edək. Əgər xətti nizamlama alqoritmində a_1 elementini yerdə qalan bütün elementlərlə müqayisə etmək lazım gəlirdisə, bu alqoritmdə a_1 elementini yalnız a_2 ilə

müqayisə etmək kifayətdir. Əgər $a_1<a_2$ olarsa, onda a_2 elementini a_3 ilə, a_3 elementini a_4 ilə və s. müqayisə etmək lazımdır. Belə müqayisə zamanı hər hansı element birbaşa özündən sonra gələn elementdən böyük olarsa, onda onların yerlərini dəyişməklə müqayisəetməni sonadək davam etdirmək lazımdır. Lakin, yerdəyişmə zamanı elə ola bilər ki, a_3 elementi a_2–dən kiçik olsun. Bu isə o deməkdir ki, növbəti dövr başa çatdıqdan sonra, massivin başlanğıcına qayıdaraq elementlərin cüt–cüt müqayisəetmə prosesini təkrarlamaq lazımdır. Beləliklə, "qabarcıq" metodunun alqoritmi belə olacaqdır. Massivin qonşu elementlərini, yəni a_1 elementini a_2 ilə, a_2 elementini a_3 ilə, a_3 elementini a_4 ilə və nəhayət a_{n-1} elementini a_n ilə müqayisə etməklə, müqayisəetmə prosesini o qədər təkrar etmək lazımdır ki, elementlərin yeri artıq dəyişməsin.

İlkin vəziyyətdə massivlərin elementləri müəyyən dərəcədə nizamlanmış olduqda "qabarcıq" metodu daha səmərəli olur. Ona görə də bu metodu ilk növbədə əvvəlcədən qismən nizamlanmış massivlər üçün tətbiq etmək daha sərfəlidir. Ümumi halda isə hər iki metod eyni dərəcədə səmərəlidir. "Qabarcıq" alqoritmində dövrlərin sayı əvvəlcədən məlum olmur, xətti nizamlama alqoritmində isə dövrlər sayı həmişə *n–1* sayda olur.

Bu alqoritmin isə proqramını aşağıdakı kimi yaza bilərik.

```
program qabarciq_nizam;
uses crt;
const n=5;
var i:word;
    a:array[1..n] of integer;
    x:integer;
    sort:boolean;
begin
  for i:=1 to n do read(a[i]);
  Repeat
    sort:=FALSE;
```

```
    for i:=1 to n-1 do
    if a[i]>a[i+1] then
     begin
       x:=a[i];
       a[i]:=a[i+1];
       a[i+1]:=x;
       sort:=TRUE;
     end;
  UNTIL sort=FALSE;
  for i:=1 to n do
   begin
     Write(a[i]:5,' ':3);
     if i=10 then write(#10#13);
   end;
  readln
end.
```

Burada, məntiqi tipli `sort` dəyişəninə `False` (yalan) qiyməti mənimsədilir. Dövr daxilində (`for`) massivin elementlərinin yerdəyişməsi baş verərsə, `sort` dəyişəninin qiyməti `True` (doğru) olur və növbəti elementləri müqayisə etmək üçün son şərtli dövr icra olunmağa başlayır. Massivin elementlərinin yeri dəyişmədikdə isə bu dəyişənin qiyməti `False` olaraq qalır, bu isə massivin elementlərinin nizamlanması deməkdir, ona görə də son şərtli dövr öz işini başa çatdırır.

Misal. *n* sayda təsadüfi ədədlər yaradaraq onların [*a*,*b*] parçasına daxil olanlarının sayını tapmalı; bütün təsadüfi ədədləri və [*a*,*b*] parçasına daxil olan ədədləri ekrana çıxarmalı.

```
program TES_EDED;
uses crt;
const a=10;b=30;
      n=20;
var x : array[1..n] of integer;
    i,k :byte;
```

```
begin
  k:=0;
  Writeln('Təsadüfi ədədlər generatorunun',
            #10#13,'yaratdığı bütün
            təsadüfi ədədlər:');
  Randomize;
  for i:=1 to n do
    begin
      x[i]:= random(50);
      write(x[i]:4);
    end;
  writeln(#10#13);
  writeln('intervala daxil olan ədədlər:');
  for i:=1 to n do
    begin
      if(x[i]>=a) and (x[i]<=b) then
      begin
        inc(k);
        write('   ',x[i]:2);
      end;
    end;
  writeln(#10#13);
  writeln('intervala daxil olan
          ədədlərin sayı',k:3);
end.
```

Proqramın əsas məqsədi təsadüfi ədədlərin yaradılmasını nümayiş etdirməkdən ibarətdir. Təsadüfi ədədləri yaratmaq üçün `random(50);` funksiyasından istifadə edilmişdir. Bu funksiya $0 \le x \le 50$ aralığında təsadüfi ədədlər yaradır. Lakin, bu funksiyaya müraciət etməzdən əvvəl, `Randomize;` prosedurunu çağırmaq lazımdır. Bu prosedur təsadüfi ədədlər generatorunu aktivləşdirir. Siz, proqramı dəfələrlə icra etdikdə hər dəfə tamamilə müxtəlif nəticələr alacaqsınız. Buna səbəb `Randomize;` prosedurudur. Əgər bu proseduru pozsanız, proqram həmişə eyni nəticələr verəcəkdir. Bu proqramın bir fraqmentdə həlli şəkil 6.3 – də göstərilmişdir.

6.4. Matrislər üzərində əməliyyatlar

Bildiyimiz kimi, matrislər ikiölçülü massivlərlə ifadə olunur. İkiölçülü massivlərin elementlərini daxil etmək üçün bir–birinin daxilində yerləşən ikiqat dövr təşkil etmək lazımdır. Dövr operatoru kimi adətən parametrli dövr operatoru istifadə edilir. Bu dövr operatorunun birincisinin

```
Tesadufi ededler  generatorunun
yaratdiqi butun tesadufi ededler:
19  6 48 38 10 37 29 10 34  1  7 37 14 31 12 45  1 41 18  1

intervala daxil olan ededler:
 19   10   29   10   14   12   18

intervala daxil olan ededlerin sayi  7
```

Şəkil 6.3. Təsadüfi ədədlər

parametri kimi massivin birinci indeksi, yəni matrisin sətirlərinin nömrələri, ikincisinin parametri kimi massivin ikinci indeksi, yəni matrisin sütunlarının nömrələri istifadə edilir. Daxili dövrün gövdəsində isə `read` və ya `readln` proseduru yazılır:

```
for i:=1 to n do
for j:=1 to m readln(a[i,j]);
```

Belə daxiletmə zamanı massivin hansı elementinin daxil edilməsini görmək mümkün olmur və adətən mexaniki səhvlərə yol verilir. Aşağıdakı proqram konstruksiyası isə massivin elementlərini sətirbəsətir daxil etməyə imkan verir:

```
for i:=1 to n do
begin
 write(i,'-ci sətir
       elementlərini daxil edin:');
 for j:=1 to m do read(a[i,j]);
end;
```

Belə daxiletmə zamanı proqram ekrana `1-ci sətir elementlərini daxil edin:` məlumatı verir Buna cavab olaraq massivin birinci sətir elementləri, aralarında bir probel simvolu qoyulmaqla, bir sətirdə yığılır və sətrin sonunda *Enter* klavişi basılır. Bundan sonra, növbəti sətirlərin daxil edilməsi tələb olunduqda, eyni qayda ilə növbəti sətirlərin elementlərini daxil etmək lazımdır.

Matrisin elementlərini sətir və sütunlar şəklində ekrana çıxarmaq üçün aşağıdakı proqram konstruksiyasından istifadə etmək daha məqsədəuyğundur:

```
for i:=1 to n do
begin
 for j:=1 to m do write(c[i,j],'  ');
 writeln;
end;
```

Massivlər üzərində əməliyyatlara aid misallara baxaq.

Misal. *A(n,m)* massivinin sətir elementlərinin hasilləri cəminin hesablanması.

```
program element_hasil_cemi;
uses crt;
const n=3;m=2;
type
   matr=array[1..n,1..m] of real;
var
   A: matr;
   i,j:integer;
   z,s:real;
begin
 for i:=1 to n do
   begin
     write(i,'-ci sətir elementlərini
           daxil edin:');
     for j:=1 to m do read(a[i,j]);
```

```
    end;
  clrscr;
  S:=0.0;
  for i:=1 to n do
   begin
     Z:=1;
     for j:=1 to m do Z:=Z*A[i,j];
     S:=S+Z;
   end;
  writeln('Nəticə = ',S);
  readln
end.
```

İlkin verilənləri daxil etdikdə, proqram ekrana `1-ci sətir elementlərini daxil edin:` məlumatı verir Buna cavab olaraq massivin birinci sətir elementləri, aralarında bir probel simvolu qoyulmaqla, bir sətirdə yığılır və sətrin sonunda *Enter* klavişi basılır. Bundan sonra, növbəti sətirlərin daxil edilməsi tələb olunacaqdır. Massivin sətir elementlərinin hasilini tapmaq üçün, dövr təşkil etməzdən əvvəl, hər hansı bir dəyişənə vahid qiyməti mənimsədərək (`Z:=1;`) dövrün daxilində onu hasili tapılacaq parametrə (bizim misalda massivin elementlərinə) vurmaq lazımdır (`Z:=Z*A[i,j];`). Xüsusi diqqət yetirin ki, `Z:=1;` operatoru iki dövr operatorunun arasında yazılmışdır. Bu ona görə belə edilmişdir ki, matrisin bütün elementlərinin hasilini deyil, sətir elementlərinin hasilini tapmaq və onları cəmləmək lazımdır. Əgər `Z:=1;` operatoru `S:=0.0;` operatorundan ya əvvəl, ya da sonra yazılsaydı, onda matrisin bütün elementlərinin hasilləri cəmi hesablanardı. Cəmləmə isə, həmişə olduğu kimi, dövrün daxilində məlum `S:=S+Z;` alqoritmi üzrə yerinə yetirilmişdir.

Misal. *X(n,m)* matrisinin müsbət elementlərinin kvadratları cəminin mənfi elementlərin sayına olan nisbətini tapın.

```
program element;
```

```
uses crt;
const n=2;m=2;
var
   X:array[1..n,1..m]of real;
   i,j,k:byte;
   cem,nisbet:real;
begin
  for i:=1 to n do
   begin
    write(i,'-ci sətir elementlərini
           daxil edin:');
    for j:=1 to m do read(x[i,j]);
   end;
  clrscr;
  cem:=0.0;
  k:=0;
  for i:=1 to n do
  for j:=1 to m do
  if x[i,j]>0
   then cem:=cem+x[i,j] { müsbət elementlərin cəmi }
   else k:=k+1;         { mənfi elementlərin sayı }
  if k=0 then
   begin
    writeln('Mənfi element yoxdur');
    exit;
   end;
  nisbet:=cem/k;
  writeln('Cavab=',nisbet);
  readln
end.
```

Misal. Paskal üçbucağının qurulması.

Paskal üçbucağı *n* sayda sətirdə yerləşən *n* sayda natural ədədlərdən tərtib olunur və bu üçbucağın yan tərəflərindəki ədədlər 1–ə bərabər olur. Hər bir üçbucağın oturacağındakı ədədlər isə özündən yuxarıdakı sətirdə ona ən yaxın olan iki ədədin cəminə bərabərdir. *n* sayda sətirdən ibarət Paskal üçbucağını qurmalı.

```
program Pascal_uchbucaqi;
uses crt;
const n=10;
var i,k,j:byte;
    s:string;
    a:array[1..n,1..n] of integer;
begin
 writeln;
 a[1,1]:=1;
 for k:=2 to n do
 for i:=1 to k do
 if (i=1) or (i=k) then a[k,i]:=1
 else a[k,i]:=a[k-1,i-1]+a[k-1,i];
 s:=' ';
 for i:=1 to n do s:=s+'  ';
 for i:=1 to n do
  begin
    write(s);
    for j:=1 to i do write(' ',a[i,j]:3);
    writeln;
    delete (s,1,2);
  end;
 readln
end.
```

Proqramın nəticəsi şəkil 6.4 – də göstərilmişdir.

Misal. Matrisin transponirə edilməsi.

```
program matrisin_transp_edilmasi;
uses crt;
var a:array[1..3,1..4] of real;
    b:array[1..4,1..3] of real;
    i,j,k:word;
begin
 for i:=1 to n do
  begin
    write(i,'-ci sətir elementlərini
           daxil edin:');
    for j:=1 to m do read(a[i,j]);
```

```
    end;
  for j:=1 to 4 do
    begin
      for i:=1 to 3 do
      begin
        b[j,i]:=a[i,j];
        write(b[j,i]:4:2,'  ');
      end;
      writeln;
    end;
  readln
end.
```

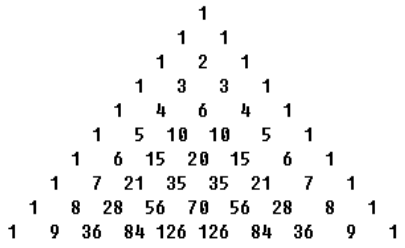
Əgər proqrama

$$A=\begin{pmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\end{pmatrix}$$

matrisi daxil edilərsə, nəticədə

$$B=\begin{pmatrix}1.00 & 5.00 & 9.00\\ 2.00 & 6.00 & 10.00\\ 3.00 & 7.00 & 11.00\\ 4.00 & 8.00 & 12.00\end{pmatrix}$$

matrisi alınacaqdır.



Şəkil 6.4. Paskal üçbucağı

Misal. Matrislərin vurulması.

```
program matrisin_vurulmasi;
uses crt;
var a:array[1..3,1..4] of real;
    b:array[1..4,1..3] of real;
    c:array[1..3,1..3] of real;
    i,j,k:word;
    s:real;
```

```
begin
 for i:=1 to 3 do
  begin
   write(i,'-ci sətir elementlərini
          daxil edin:');
   for j:=1 to 4 do read(a[i,j]);
  end;
 for i:=1 to 4 do
  begin
   write(i,'-ci sətir elementlərini
          daxil edin:');
   for j:=1 to 3 do read(b[i,j]);
  end;
 for k:=1 to 3 do
  begin
   for i:=1 to 3 do
   begin
    s:=0;
    for j:=1 to 4 do
    s:=s+a[i,j]*b[j,k];
    c[i,k]:=s;
   end;
  end;
 for i:=1 to 3 do
  begin
   for j:=1 to 3 do write(c[i,j],'  ');
   writeln;
  end;
 readln
end.
```

Əgər proqrama

$$A=\begin{pmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\end{pmatrix}$$ və

$$B=\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$ matrisləri daxil edilərsə, nəticədə

$$C=\begin{pmatrix} 70 & 80 & 90 \\ 158 & 164 & 210 \\ 246 & 288 & 330 \end{pmatrix}$$ alınacaqdır.

Matrisin baş və köməkçi diaqonal elementləri. Fərz edək ki,

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n-1} & a_{2n} \\ a_{31} & a_{32} & \ldots & a_{3n-1} & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \ldots & a_{nn-1} & a_{nn} \end{pmatrix}$$

matrisi verilmişdir. Bu matrisin ixtiyari elementinə `a[i,j]` kimi müraciət etmək olar. Burada, i ($i=1,2,\ldots,n$) – sətirlərin, j ($j=1,2,\ldots,n$) isə sütunların nömrəsidir.

$a_{11}, a_{22}, \ldots, a_{n-1n-1}, a_{nn}$

elementlərinə matrisin *baş diaqonal* elementləri,

$a_{1n}, a_{2n-1}, \ldots, a_{n-12}, a_{n1}$

elementlərinə isə *köməkçi diaqonal* elementləri deyilir. Məsələn, yuxarıdakı misalın *C* matrisində *70, 164, 330* ədədləri matrisin baş diaqonal elementləri, *90, 164, 264* ədədləri isə köməkçi diaqonal elementləridir. Göründüyü kimi, matrisin baş diaqonal elementlərinin indeksləri eynidir. Ona görə də bu elementləri tapmaq üçün dövr yaradıb, dövrün daxilində eyni indekslər yazmaq kifayətdir. Məsələn,

a(n,n) matrisinin baş diaqonal elementlərini tapmaq üçün proqramda

```
for i:=1 to n do b[i]:=a[i,i];
```

yazmaq kifayətdir. Burada `b[i]` matrisin baş diaqonal elementlərindən ibarət massivdir.

Matrisin köməkçi diaqonal elementlərini tapmaq üçün *i* və *j* indeksləri arasında asılılıq tapmaq lazımdır. Birinci köməkçi diaqonal elementi a_{1n} elementidir, yəni $i=1$ olduqda, $j=n$ olur. İkinci $a_{2\,n-1}$ elementi üçün $i=2$, $j=n-1$, üçüncü $a_{3\,n-2}$ elementi üçün $i=3$, $j=n-2$ və nəhayət *n*–ci a_{n1} elementi üçün $i=n$, $j=1$ olur. Bu asılılığı ümumiləşdirsək, köməkçi diaqonal elementlərinin sətir və sütunları arasında

$i=n-j+1$ və ya $j=n-i+1$

asılılığını alarıq. $i>n-j+1$ olduqda elementlər köməkçi diaqonaldan aşağıda, $i<n-j+1$ olduqda isə elementlər köməkçi diaqonaldan yuxarıda yerləşir. İndi isə bu elementlərin tapılmasına aid misallara baxaq.

Misal. Matrisin baş və köməkçi diaqonal elementlərinin tapılması.

```
program diaqonal_el;
uses crt;
const n=5;
var a:array[1..n,1..n] of integer;
    b,k:array[1..n] of integer;
    i,j:byte;
begin
  for i:=1 to n do
    begin
      write(i,'-ci sətir elementlərini
            daxil edin:');
      for j:=1 to n do read(a[i,j]);
    end;

    { baş diaqonal elementləri }
  for i:=1 to n do b[i]:=a[i,i];
```

```
    { köməkçi diaqonal elementləri }
  for i:=1 to n do
  for j:=n downto n-i+1 do k[i]:=a[i,j];

    { elementlərin çap edilməsi }
  writeln(' baş diaqonal elementləri ');
  for i:=1 to n do write(b[i],'  ');
  writeln;
  writeln('köməkçi diaqonal elementləri:');
  for i:=1 to n do write(k[i],'  ');
  readln
end.
```

Daxil edilmiş matris və proqramın nəticəsi şəkil 6.5 – də göstərilmişdir.

```
1-ci setir elementlerini daxil edin:10 20 30 1 2
2-ci setir elementlerini daxil edin:40 50 60 3 4
3-ci setir elementlerini daxil edin:70 80 90 5 6
4-ci setir elementlerini daxil edin:5 6 7 8 9
5-ci setir elementlerini daxil edin:11 22 33 44 55
bash diaqonal elementleri:
10  50  90  8  55
komekchi diaqonal elementleri:
2  3  90  6  11
```

Şəkil 6.5. Matrisin baş və köməkçi diaqonal elementlərinin tapılması

Misal. Matrisin köməkçi diaqonaldan yuxarıdakı və aşağıdakı elementlərinin tapılması.

```
program diaqonal_el;
uses crt;
const n=5;
var a,yu,ya:array[1..n,1..n] of integer;
    b,k:array[1..n] of integer;
    i,j:byte;
begin
  for i:=1 to n do
```

```
    begin
      write(i,'-ci sətir elementlərini
             daxil edin:');
      for j:=1 to n do read(a[i,j]);
    end;

    { köməkçi diaqonaldan yuxarıdakı elementlər }
  for i:=1 to n do
  for j:=1 to n-i do Yu[i,j]:=a[i,j];
  writeln(' köməkçi diaqonaldan
          yuxarıdakı elementlər:');
  for i:=1 to n do
  for j:=1 to n-i do  write(Yu[i,j],'  ');
  writeln;

  { köməkçi diaqonaldan aşağıdakı elementlər }
  for i:=2 to n do
  for j:=n-i+2 to n do Ya[i,j]:=a[i,j];
  writeln('köməkçi diaqonaldan aşağıdakı
           elementlər:');
  for i:=2 to n do
  for j:=n-i+2 to n do write(Ya[i,j],'  ');
  readln
end.
```

Daxil edilmiş matris və proqramın nəticəsi şəkil 6.6 – da göstərilmişdir.

```
1-ci setir elementlerini daxil edin:10 20 30 1 2
2-ci setir elementlerini daxil edin:40 50 60 3 4
3-ci setir elementlerini daxil edin:70 80 90 5 6
4-ci setir elementlerini daxil edin:5 6 7 8 9
5-ci setir elementlerini daxil edin:11 22 33 44 55
komekchi diaqonaldan yuxaridaki elementler:
10  20  30  1  40  50  60  70  80  5
komekchi diaqonaldan ashaqidaki elementler:
4  5  6  7  8  9  22  33  44  55
```

Şəkil 6.6. Matrisin köməkçi diaqonaldan yuxarıdakı və aşağıdakı elementlərinin tapılması

6.5. Simvol və sətirlər üzərində əməllər

Əvvəlki bölmələrdə biz simvol və sətirlər üzərində yerinə yetirilən əməllərlə, prosedur və funksiyalarla tanış olduq. İndi isə simvol və sətirlərin praktiki tətbiqi ilə məsələlər həll edək.

Misal. *Z* – dən *A* – ya kimi hərflərin ekrana çıxarılması.

```
program Z_A;
uses crt;
var i:char;
begin
 for i:='Z' downto 'A' do
 write(i);
 readln
end.
```

Bu proqram `Z` – dən `A` – ya kimi hərfləri ekranda göstərir.

Misal. Klaviaturadan simvolların daxil edilməsi.

```
Program testread;
uses crt;
var ch:char;
begin
 Writeln('Simvolu daxil et:'); readln(ch);
 ClrScr;
 Writeln(ch,'  simvolu daxil edildi.');
 readln
end.
```

Misal.

```
Program kod;
uses crt;
var ch:char;
begin
 Writeln('Proqram simvolların
```

```
            kodlarını tapmaq üçündür');
  Writeln('Proqramdan çıxmaq
            üçün Ctrl+Break
            klavişlərini bas');
  Writeln;
  Writeln;
    repeat;
      Write('Növbəti klaviş:');
      ch:=ReadKey;
      Writeln('  ord(',ch,')=',Ord(ch))
    Until False;
  readln
end.
```

Bu proqramda `ReadKey` funksiyası tətbiq edilmişdir ki, bu funksiya kursorun yerini dəyişdirmədən simvolu daxil etməyə imkan verir. Simvolu daxil etdikdə, bu funksiya, kursorun yerini dəyişdirmir, ona görə də daxil edilmiş simvolun yerində digər simvol təsvir olunur. Bundan başqa, son şərtli dövr operatorunda `Until False;` konstruksiyası tətbiq edildiyi üçün, proqram sonsuz dövr edəcəkdir. Proqramın işini dayandırmaq üçün ***Ctrl+Break*** klavişlərini birgə basmaq lazımdır.

Misal.

```
Program testread_key;
uses crt;
var ch:char;
begin
  Writeln('Kiçik hərfləri və ya çıxmaq
         üçün z hərfini daxil et:');
    repeat;
      ch:=readKey;
      Write(UpCase(ch));
    Until ch='z';
  readln
end.
```

Bu proqram işləyərkən ekrandan yalnız kiçik hərfləri daxil etmək lazımdır. Proqramın işi kiçik hərfləri baş hərflərə (`UpCase(ch)`) çevirməkdən ibarətdir.

Misal.

```
Program shifreleme;
uses crt;
var ch:char;
begin
 Writeln('Kiçik hərfləri və ya çıxmaq
        üçün Z hərfini daxil et:');
  repeat;
   ch:=ReadKey;
   Write(Char(Ord(ch)+1));
  Until ch='z';
 readln
end.
```

Bu proqramda şifrələmə əməliyyatı aparılır. Klaviaturadan hər hansı bir klavişi daxil etdikdə ekranda tamamilə başqa simvol təsvir olunur (`Write(Char(Ord(ch)+1));`). Bu proqram əsasında adınızı şifrələyən proqramın yazılmasını özünüzə həvalə edirik.

Misal. Daxil edilən simvollar çoxluğunun Azərbaycan dilində ilin ayına uyğunluğunu yoxlamaq. Sadəlik üçün, bu misalda, yalnız böyük hərflərə baxacağıq.

```
program aylar;
uses crt;
const
    Ay: array [1..12] of String [10] =
    ('YANVAR','FEVRAL',
    'MART','APREL','MAY',
    'İYUN','İYUL','AVQUST','SENTYABR',
    'OKTYABR','NOYABR','DEKABR');
```

```
var
   Str: string[10];
   i: integer;
begin
  writeln('BÖYÜK HƏRFLƏRLƏ AYIN
          ADINI DAXİL EDİN');
  Readln(Str);
  for i:= 1 to 12 do
  if Str = Ay[i] then
  writeln(' Ayın adı düzdür ')
  else
  writeln('Bu adda ay yoxdur');
  readln
end.
```

Misal. `Str` və `Val` prosedurlarının tətbiqi.

```
program setir_ve_qiymet;
uses crt;
var i,errcode:integer;
    S:string;
    a:integer;
begin
  read(A);
  Str(A,S);
  writeln('Sətir qiyməti =',S);
  Val(S,i,errcode);
  if errcode<>0 then
  writeln(errcode,'mövqeyində     daxiletmə
səhvi')
  else
  Writeln('Ədədi qiymət =',i);
  readln
end.
```

Bu proqramda, tam tipli `A` dəyişəninə ədədi qiymət daxil edilir, lakin o, `Str(A,S);` proseduru ilə `S` sətir tipinə çevrilir. `Val(S,i,errcode);` proseduru ilə isə əks çevirmə yerinə yetirilir: `S` sətir tipi `i` tam tipinə çevrilir.

Misal. "Kalkulyator" proqramı.

```
program kalkulyator;
uses crt;
var emeliyyat:char;
    y:real;
    a,b:real;
begin
 repeat
  writeln('Ədədləri daxil edin');
  write('a=');read(a);
  write('b=');read(b);
  writeln('Əməliyyat simvolunu
          daxil edin', 'və ya çıxmaq
          üçün z klavişini basın');
  emeliyyat:=readKey;
   case emeliyyat of
     '+': y:=a+b;
     '-': y:=a-b;
     '*': y:=a*b;
     '/': y:=a/b;
    else
     exit;
    end;
  writeln(y);
 until emeliyyat='z';
 readln
end.
```

Bu proqramla iki ədəd üzərində sadə hesab əməliyyatları yerinə yetirilir. Proqrama ixtiyari iki ədəd daxil etdikdən sonra, əməliyyat işarəsinin (+ toplama, – çıxma, * vurma və ya / bölmə) daxil edilməsi tələb olunur. Proqramın işi iki halda dayandırılır: `z` klavişini basdıqda və bu əməliyyat işarələrindən fərqli ixtiyari simvol daxil etdikdə `exit` proseduru ilə.

Yeddinci fəsil

ALT PROQRAMLAR

Bu fəsildə alt proqramlar haqqında ümumi məlumatlar veriləcək, lokal və qlobal dəyişənlər, formal və faktik parametrlər araşdırılacaqdır. Prosedur, funksiya və rekursiv tipli alt proqramların strukturunu və əsas proqramdan onlara müraciət qaydalarını öyrənəcəyik. Modulların strukturu ilə tanış olacaq və modulların yaradılması qaydalarını mənimsəyəcəyik. Alt proqramların tətbiqi ilə məsələlər həll ediləcəkdir.

7.1. Alt proqramlar haqqında ümumi məlumatlar

Proqramların tərtibində yaxşı proqramlaşdırma üslubu qaydalarına əməl etmək vacibdir. Yeri gəlmişkən bu qaydaların bir neçəsini xatırlayaq:

- `Program` sözündə (sərlövhədə) müəyyən məna bildirən yığcam proqram adları yazmaq;
- Sabit və dəyişənlərin adlarında müəyyən məna bildirən identifikatorlar istifadə etmək;
- Proqram bloklarının, dəyişən və sabitlərin mahiyyətini dərk etmək üçün qısa şərhlər yazmaq;
- Abzas və boş sətirlər əlavə etməklə proqramı strukturlaşdırmaq.

Bu və digər qaydaların tətbiqi asan başa düşülən proqramlar tərtib etməyə imkan verir.

Ən yaxşı proqramlaşdırma üslublarından biri də prosedur və funksiya tipli alt proqramların əsas proqramlara əlavə edilməsidir.

Alt proqram məntiqi bitkin və xüsusi qayda ilə tərtib edilmiş operatorlar qrupundan ibarətdir. Proqramın müxtəlif hissələrindən alt proqrama dəfələrlə müraciət etmək olar.

Strukturuna görə alt proqram əsas proqrama demək olar ki, tam analojidir. Alt proqram da sərlövhə və blokdan ibarətdir, lakin *alt proqramda modulların qoşulması bölməsi* (`Uses`) *olmur.*

Alt proqramla iş iki əsas mərhələdən ibarətdir:

- alt proqramın təsviri;
- alt proqramın çağrılması.

İstənilən alt proqram əvvəlcə təsvir olunmalıdır. Təsviretmə zamanı alt proqramın adı, parametrlərin siyahısı və onun yerinə yetirəcəyi əməliyyatları icra edən operatorlar ardıcıllığı yazılır. Alt proqramı çağırdıqda isə yalnız onun adı və alt proqrama ötürüləcək faktik parametrlərin siyahısı göstərilir.

Turbo Pascal dilinin müxtəlif modullarında çoxlu standart alt proqramlar vardır ki, onlara müraciət etmək üçün onları təsvir etməyə ehtiyac yoxdur. Belə alt proqramların bir neçəsi ilə biz ifadələr bölməsində tanış olduq. Bundan başqa, proqramçı özü də alt proqram yarada bilər ki, ona istifadəçi alt proqramı deyilir. İstifadəçi alt proqramları isə hökmən təsvir olunmalıdır.

Alt proqramlar *prosedur* və *funksiyalara* bölünür. Bunların bir sıra ümumi cəhətləri ilə yanaşı, fərqli xüsusiyyətləri də vardır. Bu fərqlər əsasən aşağıdakılardır:

- funksiya əsas proqrama bir qayda olaraq yalnız bir qiymət ötürür, məsələn, `Sin(x)`. Prosedur isə alt proqrama müxtəlif qiymətlər, məsələn, massiv ötürə bilər;
- funksiyanın sərlövhəsində onun proqrama ötürəcəyi qiymətin tipi (`real`, `integer` və s.) göstərilir. Məsələn, `function Sin(x:real):real;` Prosedurun sərlövhəsində isə buna ehtiyac yoxdur;
- funksiyanın gövdəsində ən azı bir mənimsətmə operatoru olmalıdır ki, onun sol tərəfində funksiyanın adı yazılmalıdır, yəni funksiyanın adına hökmən qiymət mənimsədilməlidir;
- funksiyaya müraciət etdikdə, onu ifadələrdə identifikator kimi istifadə etmək mümkün olduğu halda, prosedur ifadənin bir hissəsi ola bilməz.

Proqramla alt proqram arasında mübadilə dəyişənlər vasitəsilə yerinə yetirilir. Dəyişənlər qlobal və lokal dəyişənlərə bölünür. Əgər sabit və dəyişənlər əsas proqramda elan olunmuşdursa, belə dəyişənlərə *qlobal dəyişənlər* deyilir. Bu dəyişənlərə, kompilyasiya mərhələsində, verilənlər seqmentində yaddaşda yer ayrıldığı üçün, həmin dəyişənləri proqramın istənilən hissəsində, o cümlədən, alt proqramlarda da istifadə etmək olar (bu halda deyirlər ki, həmin dəyişənlər proqramın hər yerindən görünür).

Əgər sabit və dəyişənlər alt proqramda elan olunmuşdursa, onlara *lokal dəyişənlər* deyilir. Qlobal dəyişənlər alt proqramda təsvir olunduqda onlar lokal dəyişənlər olur və bu dəyişənlər yalnız onların elan olunduqları alt proqramlardan görünür. Lokal dəyişənlər üçün də yaddaşda – verilənlər seqmentində – yer ayrılır. Qlobal və lokal dəyişənlərin adları eyni olsa belə, onlar arasında heç bir əlaqə olmur. Dərhal qeyd edək ki, çalışın alt proqramlarda qlobal dəyişənlərdən istifadə etməyəsiniz. Bunun bir neçə səbəbi vardır. Səbəblərdən biri odur ki, qlobal dəyişənləri istifadə edən alt proqram az universal olur və onu digər proqramlarda istifadə etmək çətinləşir. İkinci səbəb isə, belə

alt proqramlardan istifadə etdikdə, çətin aşkar olunan səhvlərin artması ehtimalı ilə bağlıdır.

Əsas proqramla alt proqramın qarşılıqlı əlaqəsi, adətən, parametrlər vasitəsilə yerinə yetirilir. *Parametrlər* (*formal* parametrlər) alt proqramın elementləridir və alt proqramın yerinə yetirdiyi alqoritmin təsvirində istifadə edilir.

Arqumentlər (*faktik* parametrlər) alt proqramı çağıran proqramın elementidir. Alt proqrama müraciət etdikdə formal parametrlər faktik parametrlərlə əvəz olunur. Bu zaman formal və faktik parametrlər tipləri, yerləşmə ardıcıllığı və saylarına görə bir–birinə uyğun gəlməlidir. Formal və faktik parametrlər nəinki, müxtəlif adlı ola bilər, hətta çalışmaq lazımdır ki, bu elə belə də olsun.

Bəs prosedurlar necə çağrılır və onlara bu parametrlər necə ötürülür? Çox sadə. Əvvəlcə bu parametrlər stekə qəbul olunur. *Stek* müvəqqəti və ya lokal dəyişənlərin yadda saxlanması üçün yaddaş oblastıdır. Tutaq ki, prosedurun iki parametri vardır. Əvvəlcə birinci parametr, sonra isə ikinci parametr stekə ötürülür və prosedur çağrılır. Yerinə yetirilməyə başlamazdan əvvəl, prosedur həmin parametrləri stekdən əks istiqamətdə çıxarır.

Əgər alt proqramda digər prosedur və funksiyalar varsa, onda dəyişənlərin görünmə oblastı həmin prosedur və funksiyalara da aid olacaq, bu şərtlə ki, onlar eyniadlı olsun.

Alt proqramın işini dayandırmaq üçün `Exit` prosedurundan istifadə etmək olar ki, bu prosedur idarəetməni əsas proqrama verir.

İstənilən növ alt proqramları nəinki əsas proqramdan, həm də digər proqramlardan, parametrlərin eyni və ya müxtəlif qiymətlərində, istənilən qədər çağırmaq olar. Alt proqramları əsas proqramlardan kənarda da kompilyasiya etmək olar.

İndi isə, alt proqramlar haqqında bu vacib, ümumi bilikləri öyrəndikdən sonra, prosedur və funksiyaları öyrənək.

7.2. Prosedurlar

Prosedur sərlövhə və proqram blokundan fərqlənməyən blokdan ibarət olur. Sərlövhə **procedure** sözündən, prosedurun adından və onun yanında mötərizə daxilində yazılmış formal parametrlərin siyahısından ibarət olur. Prosedurun ümumi forması belədir:

Procedure *ad (formal parametrlər)*;
lokal parametr, prosedur və funksiyaların təsviri və təyini bölmələri;

begin

1–ci operator;
. . .
n–ci operator;

end;

Formal parametrlərin qarşısında `Var`, `Const` sözləri yazıla bilər. Bunların mahiyyətini bir az sonra izah edəcəyik. Formal parametrlərin adından sonra, iki nöqtə işarəsi ilə ayrılmaqla, onun tipi göstərilə bilər. Eyni tipli parametrlər olarsa, onlar arasında vergül işarəsi qoyulmalıdır. Formal parametrlər olmaya da bilər.

Misal.

```
procedure format(hour,min,sec,hund:word);
procedure ChangeStr(Var Source:string;
                    const char1, char2:char);
procedure zaman;
```

Prosedura müraciət üçün prosedurun adını və mötərizə daxilində faktik parametrləri göstərmək lazımdır.

İndi isə prosedura aid misallara baxaq.

Misal. $y = \sqrt{n!} + (m!)^2 + (n-m)!$ hesablayın.

Göründüyü kimi, üç müxtəlif dəyişənin faktorialını hesablamaq lazımdır. Ona görə də biz *n!* hesablanması üçün alt proqram tərtib edərək sonrakı hallarda ona müraciət edəcəyik.

```
Program fact_sum;
uses crt;
Var
 y1,y2,y3:integer;
 n,m:integer;
 y,t1,t2:real;

{ Alt proqram }
procedure fact(l:integer; Var p:integer);
Var
 i:integer;
begin
 p:=1;
 if l<0 then
 begin
  p:=1;
  Exit;
 end;
for i:=1 to l do p:=p*i;
end;

{ Əsas proqram }
begin
 readln(n,m);
 fact(n,y1); t1:=sqrt(y1);
 fact(m,y2); t2:=sqr(y2);
 fact(n-m,y3);
 y:=t1+t2+y3;
 writeln(y);
 readln
end.
```

Alt proqramda *n!* hesablanması üçün proqram yazılmışdır. Burada `l` və `p` alt proqramın formal parametrləridir. `l` faktorialı hesablanacaq parametr, `p` isə alt proqramın hesablayacağı nəticədir. Məlumdur ki, mənfi ədədin faktorialı mövcud deyil, ona görə də `l<0` olduqda, faktorial vahidə bərabər götürülür (`p:=1;`) və idarəetmə əsas proqrama ötürülür (`Exit;` proseduru), əks halda `l` dəyişəninin faktorialı hesablanır (`for i:=1 to l do p:=p*i;`). Əsas proqramda isə alt proqrama üç dəfə müraciət olunmuşdur. Hər dəfə müraciət zamanı əsas proqramdan alt proqrama, faktik parametrlər kimi, uyğun olaraq `n`, `m` və `n-m` dəyişənləri ötürülür, alt proqramdan isə əsas proqrama, nəticə kimi, uyğun olaraq `y1`, `y2` və `y3` dəyişənləri qaytarılır.

Misal. $y = [thx + th(x+a)^2] \cdot \sqrt{th(x-a)^2}$ hesablayın.

Hiperbolik tangensi hesablamaq üçün aşağıdakı düsturu tətbiq edək:

$$th\,x = \frac{e^{2x}-1}{e^{2x}+1}.$$

```
program Hyp_tang;
uses crt;
var x,a,y1,y2,y,r,q:real;

{ Alt proqram }
Procedure htan;
var
   s:real;
begin
  s:=exp(2*q);
  r:=(s-1)/(s+1);
end;

  { Əsas proqram }
begin
```

```
  read(x,a);
  q:=x;
  htan;
  y1:=r;
  q:=sqr(x+a);
  htan;
  y2:=r;
  q:=sqr(x-a);
  htan;
  y:=(y1+y2)*sqrt(r);
  writeln('y=',y);
  readln
end.
```

Bu proqramda parametrsiz `htan` prosedur tipli alt proqramından istifadə edilmişdir. Bu alt proqram yalnız hiperbolik tangens funksiyasını hesablayır. Əsas proqramda isə ona 3 dəfə müraciət olunmuşdur: birinci dəfə faktik parametr kimi bu prosedura x, ikinci dəfə $(x+a)^2$, üçüncü dəfə isə $(x-a)^2$ arqumentləri ötürülür. Hər müraciətdən sonra isə alt proqramın nəticələri uyğun olaraq `y1`, `y2` və `sqrt(r)` kimi əsas proqrama ötürülür.

7.3. Funksiyalar

Funksiyalar da sərlövhə və blokdan ibarətdir.

Sərlövhə – **`function`** sözündən, funksiyanın adından və onun yanında, mötərizə daxilində yazılmış, vacib olmayan, formal parametrlərin siyahısından və onların tiplərindən ibarət olur. Sərlövhədə funksiyanın özünün də tipi göstərilir.

Funksiyanın *bloku* prosedurun blokuna analojidir, lakin, funksiyanın gövdəsində onun adına hökmən qiymət (ifadə) mənimsədilməlidir. Funksiyanın gövdəsi `begin` sözü ilə

başlayır və `end;` ilə qurtarır. Beləliklə, funksiyaların ümumi forması belədir:

Function F ($q_1 : t_1$; $q_2 : t_2$; . . . ; $q_n : t_n$) **:** T**;**

lokal parametr və alt proqramların təsviri və təyini;

begin

1–ci operator;
. . .
n–ci operator;
F:=*ifadə*;

end;

Burada, **F** – funksiyanın adı, q_i – formal parametrlərin adları, t_i – formal parametrlərin tipləri, T isə funksiyanın özünün tipidir.

Misal.

```
function Sin(x:real):real;
function Pi:real;
function Random(x:word):word;
```

Funksiyaya müraciət mənimsətmə operatorunun sağ tərəfində funksiyanın adını və mötərizə daxilində faktik parametrləri göstərməklə həyata keçirilir, məsələn, `Sin(y)`, `Pi`, `Random(a)` və s. Faktik və formal parametrlər saylarına, tiplərinə və yerləşmə ardıcıllığına görə uzlaşmalıdır. Funksiyaya müraciət olunduqda faktik parametrlərin qiyməti uyğun mövqedə duran formal parametrlərə mənimsədilir. Sonra isə funksiyanın gövdəsini təşkil edən operatorlar yerinə yetirilir və son nəticə funksiyanın adına mənimsədilir. Əgər belə mənimsətmə bir neçə dəfə olarsa, onda ən sonuncu qiymət əsas proqrama ötürülür.

Misal. $y = \sqrt{n!} + (m!)^2 + (n-m)!$ hesablayın.

Prosedur bölməsində həll etdiyimiz bu məsələni funksiya alt proqramı ilə həll edək.

```
Program Fact_sum;
Uses crt;
var y:real;
    n,m:integer;

{ Alt proqram }
function fact(l:integer):integer;
var p,i:integer;
begin
 p:=1;
 if l<=0 then
 begin
   fact:=1;
   Exit;
 end;
 for i:=1 to l do p:=p*i;
 fact:=p;
end;

{ Əsas proqram }
begin
 readln(n,m);
 y:=sqrt(fact(n))+sqr(fact(m))+fact(n-m);
 write(y);
 readln
end.
```

Burada funksiya tipli `fact` alt proqramının formal parametri faktorialı hesablanacaq `l` arqumentidir. Prosedur tipli alt proqramda olduğu kimi, burada da `l` arqumentinin işarəsi yoxlanır və əgər o mənfidirsə, `fact:=1` qəbul olunur və alt proqram öz işini dayandırır (`Exit;` proseduru). `l` arqumentinin qiyməti müsbət olduqda isə alt proqramda onun

faktorialı hesablanaraq funksiyanın adına mənimsədilir (`fact:=p;`). Əsas proqramda, `y`–in hesablanması düsturunda, birbaşa `fact` funksiyasına müraciət olunur və hər müraciət zamanı bu funksiyaya faktik parametrlər kimi, `n`, `m` və `n-m` qiymətləri ötürülür.

Misal. *F(x)=x/(1+x)* funksiyasını cədvəlləşdirin.

Biz bu məsələnin proqramını əvvəlki bölmələrdə tərtib etmişdik. İndi bu məsələni alt proqram vasitəsilə həll edək.

```
program cedvel;
uses crt;
var x:real;
    k:word;

{ function tipli alt proqram }
function F(x:real):real;
begin
 F:=x/(1.0+x);
end;

{ Əsas proqram }
begin
 clrscr;
 x:=0.0;
 writeln('f(x)=x/(1+x)
        funksiyasının qiymətlər cədvəli');
 writeln;
 Writeln('sira N','x':10,'f(x)':20);
 writeln;
 for k:=0 to 50 do
  begin
   writeln(' ',k,' ',x:12:4, F(x):20:10);
   x:=x+0.1;
   if k mod 10=9 then readln;
  end;
 readln
end.
```

Misal. $y=\sum_{i=1}^{10}a_i+\sum_{i-1}^{15}a_i\cdot b_i+\sum_{i=1}^{20}(a_i-b_i)^2$ hesablayın.

```
program Alt_proq;
uses crt;
Type mas=array[1..20] of real;
var a,b,c,c1:mas;
    i:integer;
    y:real;
```

{ `function` *tipli alt proqram* }

```
function cem(k:integer;z:mas):real;
var y:integer;
    s:real;
begin
 s:=0.0;
 for y:=1 to k do
 s:=s+z[y];
 cem:=s;
end;
```

{ *Əsas proqram* }

```
begin
 for i:=1 to 20 do read(a[i]);
 for i:=1 to 20 do read(b[i]);
 for i:=1 to 15 do c1[i]:=a[i]*b[i];
 for i:=1 to 20 do c[i]:=sqr(a[i] -b[i]);
 y:=cem(10,a)+cem(15,c1)+cem(20,c);
 write(' ':10,'y=',y:10:2);
 readln
end.
```

7.4. Rekursiv alt proqramlar

Əgər alt proqram özü özünü çağırırsa, belə proqramlara *rekursiv* alt proqramlar deyilir.

Misal. *y=n!* hesablamaq üçün rekursiv alt proqram yazaq.

```
function fact(n:integer):integer;
```

```
begin
  if n<=0 then fact:=1 else fact:=n*fact(n-1);
end;
```

`fact` funksiyası `n` qiymətini alaraq onun faktorialını hesablayır. Əgər `n<=0` olarsa, onda faktorial vahidə bərabər olur. Əks halda, `n`–in qiymətini bir vahid azaldaraq (`fact(n-1)`) özü özünə müraciət edir. Bu proses `n=1` olana qədər davam edir.

7.5. Alt proqramlarda parametr və arqumentlər

Parametrlər alt proqramın elementləridir və onun alqoritminin təsvirində istifadə olunur. Arqumentlər isə alt proqram çağrıldıqda göstərilir və proqram icra olunduqda onlar parametrləri əvəz edirlər. Parametrlər istənilən tip ola bilər. Parametrlərin aşağıdakı tipləri mövcuddur:

- *parametr–qiymətlər;*
- *parametr–sabitlər;*
- *parametr–dəyişənlər;*
- *tipləşdirilməmiş sabit və dəyişənlər.*

Əgər alt proqramın sərlövhəsində, formal parametrlərin adları qarşısında, hər hansı bir işçi söz (`Var`, `Const`) yazılmazsa, onda həmin formal parametrlər *parametr–qiymətlər* adlanır.

Misal.

```
procedure par_qiy (x : real; y : integer;
                                s : string);
function vaxt (first, second : word) : boolean;
```

Parametr–qiymət kimi dəyişən, sabit və ya riyazi ifadə istifadə oluna bilər. Alt proqram çağrılmazdan əvvəl riyazi ifadə hesablanır, alınmış nəticə yaddaşa köçürülür və yalnız bundan sonra alt proqrama ötürülür. Eyni proses parametr–

qiymət sabit və ya dəyişən olduqda da baş verir. Beləliklə, faktik parametrin qiyməti eyni vaxtda iki yerdə yadda saxlanır: qiymətin əsli və onun surəti. Alt proqramda parametr–qiymət dəyişə bilər, lakin bu dəyişikliklər alt proqramlar yerinə yetirildikdə faktik parametrlərin qiymətlərinə təsir etməyəcəkdir, başqa sözlə, əsas proqrama ötürülməyəcəkdir.

Proqramın sərlövhəsində formal parametrlərin qarşısında `Const` sözü yazılarsa, həmin formal parametr *parametr–sabit* adlanır.

Misal.

```
Procedure sabit(Const x,y : integer);
```

Parametr–sabit kimi dəyişən, sabit və riyazi ifadə istifadə oluna bilər. Alt proqramın gövdəsində parametr–sabiti dəyişdirmək olmaz. Hansı parametrlərin ki, alt proqramda dəyişdirilməsi arzu olunmazdır, həmin parametrləri parametr–sabit kimi göstərmək lazımdır. Bundan başqa, sətir və struktur tipli parametr–sabitlər üçün kompilyator daha səmərəli kod yaradır.

Əgər alt proqramın sərlövhəsində, formal parametrlərin qarşısında, `Var` sözü yazılarsa, onda bu parametr *parametr–dəyişən* olacaqdır. Bu parametrə *Var–parametr* də deyirlər.

Misal.

```
Procedure takt(Var param:real;
               Var d,f:integer);
```

Parametr–dəyişən o hallarda istifadə olunur ki, alt proqramdan əsas proqrama qiymət ötürülməlidir. Parametr–dəyişən tətbiq olunduqda faktik qiymət kimi riyazi ifadə istifadə etmək olmaz. Parametr–dəyişənlərin tətbiqinin üstün cəhəti ondadır ki, böyük ölçülü parametrlərin ötürülməsi zamanı proqram sürətlə işləyir və parametrin nəticəsi, avtomatik olaraq, əsas proqrama ötürülür. Bununla bərabər, bütün formal parametrlərin parametr–dəyişənlər kimi elan

olunması məqsədəuyğun hesab olunmur. Çünki, bu əvvəla, faktik parametrlərə riyazi ifadələrin verilməsinə imkan vermir, digər tərəfdən formal parametrin alt proqramda qiyməti təsadüfən itirilə bilər. Parametrlər nə qədər az olarsa, qlobal dəyişənlər də bir o qədər az olar ki, bu da proqramın sadə və asan başa düşülən olmasına gətirər. Elə bu səbəbdən də funksiyalarda parametr–qiymətlərin istifadə edilməsi məsləhət görülmür.

Alt proqramların parametrləri *tipləşdirilməmiş sabit və dəyişənlər* də ola bilər. Bu halda alt proqramın sərlövhəsində parametr–sabit və parametr dəyişənlərin tipləri göstərilmir

Misal.

```
Procedure tipsiz(Var f1; Const t1);
Function tip (Var c2):real;
```

Formal parametrlərə verilən faktik parametrlər istənilən tip ola bilər və bunu proqramçı özü müstəqil müəyyənləşdirir.

7.6. Modullar

Modul anlayışı ilk dəfə *Ada* proqramlaşdırma dilinə daxil edilərək, paket adlanırdı. Niklaus Virt tərəfindən yaradılmış *Pascal* dilinin ilk versiyalarında modul olmamışdır. Amma bir qədər sonra, *Ada* dilində abstrakt tiplərin və paketlərin inkişafı ilə əlaqədar olaraq, Turbo Pascal dilinə modul daxil edilmişdir.

Modul informasiyanın gizlədilməsi ("*information hiding*") prinsipini əsas götürərək, proqramların yaradılmasında istifadə olunur. Turbo Pascal dilində modullar prosedur, funksiya və obyekt kitabxanalarının yaradılmasında istifadə olunur. Modulun köməyi ilə böyük proqramlar nisbətən kiçik proqram fraqmentlərinə parçalanır.

Modullar da proqramlar kimi kompilyasiya olunur, lakin onlardan fərqli olaraq sərbəst icra oluna bilmir.

Modullar iki qrupa bölünür:

- *standart modullar;*

- *istifadəçi modulları.*

Standart modullar Turbo Pascal dilində əvvəlcədən hazırlanmış modullardır. Bu modullardan proqramlarda kompilyasiya olunmuş halda istifadə olunur. Kitabxanada aşağıdakı standart modullar mövcuddur:

- **`System`** –əsas kitabxana;
- **`String`** –ASCIIZ –sətirlərinin emalı üçün kitabxana;
- **`Crt`** –konsol ilə işləmək üçün kitabxana;
- **`Graph`** –qrafiki kitabxana;
- **`Dos`** –Ms DOS sisteminin imkanlarından istifadə üçün kitabxana;
- **`WinDos`** –ASCIIZ–sətirlərini nəzərə almaqla Ms DOS sisteminin imkanlarından istifadə üçün kitabxana;
- **`Overlay`** –overley strukturun təşkili;
- **`Printer`** –printerlə iş;
- **`Turbo3`** –Turbo Pascal 3.0 proqramları ilə əlaqə;
- **`Grap3`** –Turbo Pascal 3.0 qrafikləri ilə əlaqə;

`CRT` **(***Cathod radio tube – elektron–şüa borusu*) modulu ekranı mətn rejimində idarə edən alt proqramlardan ibarətdir. Modul displeyin 8 iş rejimi sabitlərindən, 17 rəng sabitlərindən, 4 funksiya və 16 prosedurdan ibarətdir. Məsələn, `ClrScr`, `GotoXY`, `Delay`, `ReadKey`, `Keypressed` və s. prosedurlar bu modulun alt proqramlarıdır. Bu modulun qoşulması bütün giriş–çıxış əməliyyatlarını sürətləndirir. Ona görə də, hətta bu modulun alt proqramlarını istifadə etmədikdə də, onun qoşulması məsləhət görülür.

Qeyd. Pascal dilinin **Turbo Pascal for Windows 1.5** versiyası ilə işlədikdə **`CRT`** əvəzinə **`WinCrt`** yazmaq lazımdır.

İstifadəçi modulları proqramçı tərəfindən yaradılan modullardır. Bu modullar kompilyasiya olunub, sazlandıqdan sonra proqramlarda istifadə edilə bilər.

Modullar aşağıdakı hissələrdən ibarətdir:

- *modulun sərlövhəsi;*
- *modulun interfeysi;*
- *reallaşdırma bölməsi;*
- *inisiallaşdırma bölməsi.*

Modulun sərlövhəsi. Modulun sərlövhəsi **`unit`** işçi sözündən və modulun adından ibarətdir. Məsələn, **`unit`** `Modul2.`

Modullar da adi pascal–proqramları kimi inteqrallaşdırılmış mühitdə yığılır. Modulu yadda saxladıqda isə fayla verilən ad hökmən modulun adı ilə (`unit` sözündən sonrakı adla) eyni olmalıdır və bu fayl da *.pas* tipli olacaqdır.

Modulun interfeysi. Bu bölmədə modulun digər istifadəçi və standart modullarla, həmçinin əsas proqramla qarşılıqlı əlaqəsi təsvir olunur. Başqa sözlə, modulun "xarici aləmlə" qarşılıqlı əlaqəsidir. Modulun interfeysi **`interface`** sözü ilə başlayır və aşağıdakı hissələrdən ibarət ola bilər:

- *istifadə olunan modulların təsviri bölməsi;*
- *sabitlərin təsviri bölməsi;*
- *tiplərin təsviri bölməsi;*
- *dəyişənlərin təsviri bölməsi;*
- *prosedur və funksiyaların təsviri bölməsi.*

Reallaşdırma bölməsi (icraedici hissə). Bu bölmədə cari modulun reallaşdırılması təsvir olunur, başqa sözlə, modulun "daxili mətbəxidir" və digər modul və proqramların buradan istifadəsi mümkün deyildir. Reallaşdırma bölməsi **`implementation`** sözü ilə başlayır və inisiallaşdırma bölməsinin başlanğıcı və ya **`end.`** sözü ilə qurtarır.

Bu bölmə aşağıdakı hissələrdən ibarət ola bilər:

- *nişanların təsviri bölməsi;*
- *istifadə olunan modulların təsviri bölməsi;*
- *sabitlərin təsviri bölməsi;*
- *tiplərin təsviri bölməsi;*
- *dəyişənlərin təsviri bölməsi;*
- *prosedur və funksiyaların təsviri bölməsi.*

İnisiallaşdırma bölməsi. Bir çox hallarda, modula müraciətdən əvvəl, onun inisiallaşdırılması, məsələn, `Assign` prosedurunun köməyi ilə bəzi fayllarla əlaqə yaratmaq, hər hansı dəyişənin inisiallaşdırılması və s. həyata keçirilməlidir. Bütün bu əməliyyatları inisiallaşdırma bölməsi həyata keçirir. Bölmə `begin` və `end` sözləri arasındakı icra olunan operatorlardan təşkil olunur. İnisiallaşdırma operatorları tələb olunmursa, `begin` sözü yazılmır.

Modulda əksər hallarda inisiallaşdırma bölməsi olmur, adətən bu bölmədə hər hansı faylla əlaqə yaratmaq üçün kodlar yazılır. Əgər bu bölmə olarsa, o, `begin` və `end`**.** operatorları arasında yerləşdirilir.

Modulun interfeysində təsvir olunan sabit, dəyişən, prosedur və funksiyalardan əsas proqramda istifadə etmək üçün `uses` işçi sözündən istifadə olunur. Bu təsvirdən sonra, əsas proqramda interfeysdə göstərilən modullardan istifadə etmək mümkündür.

Modul köməkçi obyekt olduğu üçün, onu *Run* (***Ctrl+F9***) əmri ilə icra etmək olmaz. O, yalnız proqram və alt proqramların yaradılmasında iştirak edə bilər.

Modul iki mərhələ üzrə yaradılır. Birinci mərhələdə ayrı bir faylda modulun mətni yığılmalıdır. Bu faylın adı modulun adındakı birinci 8 simvolla eyni olmaqla, *.pas* tipinə malik olmalıdır. Modul və proqram hər ikisi eyni bir qovluqda yerləşməlidir.

Hər bir modul **TPU** (*Turbo Pascal Unit*) adlanan xüsusi kitabxanada yerləşməlidir. Modulu **TPU** kitabxanasına yerləşdirmək üçün onu ***F9*** və ya ***Ctrl+F9*** əmri ilə kompilyasiya etmək lazımdır. Bundan sonra, modulun adından ibarət *.pas* tipli fayl yaranacaqdır. Bu faylın *.pas* tipli adi proqram faylından fərqi ondadır ki, burada informasiya maşın kodları ilə təsvir olunduğu üçün, onu oxumaq və başa düşmək mümkün olmur. Sonradan düzəlişlər etmək məqsədilə *.pas* tipli faylı pozmaq məsləhət görülmür.

Misal. Hiperbolik funksiyaların hesablanması üçün modul yaradaq. Bu funksiyaların düsturlarını yada salaq:

$$\sinh(x)=\frac{e^{x}-e^{-x}}{2};\quad \cosh(x)=\frac{e^{x}+e^{-x}}{2};$$

$$\tanh(x)=\frac{\sinh(x)}{\cosh(x)}.$$

```
{$N+}
Unit hyp_fun;

interface

type
    deqiq=Extended;
    function sinh(x:deqiq):deqiq;
    function cosh(x:deqiq):deqiq;
    function tanh(x:deqiq):deqiq;

implementation

var t:deqiq;

function sinh(x:deqiq):deqiq;
begin
   t:=exp(x);
   sinh:=0.5*(t -1./t);
end;
```

```
function cosh(x:deqiq):deqiq;
begin
 t:=exp(x);
 cosh:=0.5*(t+1./t);
end;

function tanh(x:deqiq):deqiq;
begin
 t:=exp(2.0*x);
 tanh:=(t -1.0)/(t+1.0);
end;

end.
```

Bu modulu ***F9*** və ya ***Ctrl+F9*** əmri ilə kompilyasiya etdikdən sonra, onu `hyp_fun.pas` adı ilə yadda saxlayın. İndi bu moduldan əsas proqramda istifadə edək.

```
{$N+}
program test_hyperbolic_fun;
Uses CRT,hyp_fun;  {hyp_fun hökmən yazılmalıdır }
begin
 clrScr;
 Writeln('Sinh(0.5)=',sinh(0.5));
 Writeln('Cosh(-0.5)=',cosh(0.5));
 Writeln('Tanh(0.5)=',tanh(1.5));
 readln
end.
```

Bu misalda, alt proqramlara və modullara heç bir aidiyyəti olmayan **`{$N+}`** direktivindən istifadə olunmuşdur. `{$N+}` direktivi ona görə yazılmışdır ki, burada yüksək dəqiqlikli `Extended` tipindən istifadə edilmişdir (əks halda kompilyator həmin tipi tanımır).

S ə k k i z i n c i f ə s i l

FAYLLAR

Bu fəsildə proqramın nəticələrinin faylda yadda saxlanılması və ilkin verilənlərin fayllardan oxunması qaydalarını öyrənəcəyik. Mətn faylları, tipləşdirilmiş fayllar və tipləşdirilməmiş fayllar ayrı–ayrılıqda öyrəniləcək, onların müqayisəli təhlili veriləcək, bu fayllarla işləmək üçün zəruri funksiya və prosedurlar nəzərdən keçiriləcəkdir. Qovluq və fayllarla iş üçün ümumi vasitələr şərh olunacaq və fəslin sonunda fayllarla işləmək üçün çoxlu praktiki məsələlər həll ediləcəkdir.

8.1. Fayllar haqqında ümumi məlumatlar

İndiyədək yazdığımız proqramlarda ilkin verilənləri həmişə klaviaturadan daxil edirdik və proqramdan çıxan kimi onun nəticələri ekrandan itirdi. Fayllardan istifadə etdikdə biz, proqrama ilkin verilənləri klaviaturadan deyil, fayllardan daxil edə bilərik və nəticələri fayllarda saxlaya bilərik. Bu isə imkan verir ki, alınmış nəticələri sonradan təhlil edək və ya onları digər proqramlarda, verilənlər bazalarında istifadə edək. Bildiyimiz kimi, fayl xarici yaddaşda adlandırılmış hər hansı bir sahədir. Bu faylın fiziki mövcudluğudur ki, bu da fiziki fayl adlanır. Fiziki faylın strukturu informasiya daşıyıcılarında (disk və ya disket) ardıcıl baytlardan ibarətdir, yəni aşağıdakı kimidir:

bayt	*bayt*	*bayt*	. . .	*bayt*	*bayt*

Digər tərəfdən, fayl proqramlaşdırmada istifadə olunan çoxlu sayda verilənlər strukturundan biridir. Bu halda *məntiqi fayl* anlayışından istifadə olunur. Yəni proqram tərtibində faylın mövcudluğu haqqında bizdə yalnız məntiqi təsəvvür var. Proqramlarda məntiqi fayllar *fayl dəyişəni ilə* təsvir olunur.

Məntiqi faylın strukturu proqramda faylın başa düşülmə üsuludur. Daha dəqiq desək, faylın fiziki strukturuna baxmaq üçün olan "şablon" və ya "pəncərə"dir. Proqramlaşdırma dilində belə "şablonlara" verilənlərin tipi uyğundur. Turbo Pascal dilində bir neçə "şablonun" təsviri aşağıdakı göstərilmişdir:

`file of Byte`

bayt	*bayt*	*bayt*	. . .	*bayt*	**Eof**

`file of Char`

simvolun kodu	*simvolun kodu*	*simvolun kodu*	. . .	*simvolun kodu*	**Eof**

`file of Integer`

işarəli tam	*işarəli tam*	*işarəli tam*	. . .	*işarəli tam*	**Eof**

`file of P`

bayt	*simvolun kodu*	*işarəli tam*	. . .	*bayt*	*simvo-lun kodu*	*işarəli tam*	**Eof**

Sonuncu təsvirdə `P` yazı tipli dəyişəndir və onun strukturu belədir:

```
type
   P=record
     a: Byte;
     b: Char;
     c: Integer;
   end;
```

Məntiqi faylın strukturu massivə oxşayır. Amma, fayl və massivlərin bəzi fərqləri mövcuddur. Massivlər operativ yaddaşda təşkil olunur və onların elementlərinin sayı adətən əvvəlcədən məlum olur və nömrələnir. Faylda isə proqramın işi zamanı elementlərin sayı dəyişir və o, xarici informasiya daşıyıcılarında yerləşir. Faylın elementlərinin sayı hər an dəyişə bilər. Amma, faylın sonuna **ASCII** kodu *26* (***Ctrl+Z***) olan xüsusi **Eof** simvolu əlavə edilir. Bunlardan başqa, faylın uzunluğunu müəyyən etmək və ya fayllar üzərində digər əməliyyatları yerinə yetirmək üçün, xüsusi standart prosedur və funksiyalardan istifadə olunur.

Üçüncü fəsildə qeyd etdiyimiz kimi, Turbo Pascal dilində 3 növ fayl vardır:

- *Mətn faylları;*
- *Tipləşdirilmiş fayllar;*
- *Tipləşdirilməmiş fayllar.*

Qeyd etmək lazımdır ki, tipləşdirilmiş və tipləşdirilməmiş faylların məzmununa həm ardıcıl həm də birbaşa müraciət üsulu mümkün olduğu halda, mətn fayllarına yalnız ardıcıl müraciət etmək olar.

8.2. Faylların təyini, açılması və bağlanması

Fayl tipinin təyini üçün **`Text`**, **`file`** və **`of`** işçi sözlərindən istifadə olunur. Bu işçi sözlərdən sonra faylın tipi göstərilir. Faylların ümumi təsvir forması belədir:

Mətn faylları

`Var` *fayl dəyişəni* **`: Text;`**

Tipləşdirilmiş fayllar

`Var` *fayl dəyişəni* **`: file of`** *faylın tipi***`;`**

Tipləşdirilməmiş fayllar

`Var` *fayl dəyişəni* **`: file;`**

Burada, *fayl dəyişəni* fiziki faylla əlaqə yaradan məntiqi fayl dəyişəni, *faylın tipi* isə fiziki faylın tipidir. Mətn

fayllarında istənilən tip ədədlər, simvol və mətnlər saxlana bilər və sətirlər müxtəlif uzunluqlu ola bilər. Tipləşdirilmiş fayllar tam, həqiqi, ikiqat dəqiqlikli və s. verilənlərdən ibarət (lakin eyni bir faylın məzmunu yalnız eyni tipli və eyni uzunluqlu olmalıdır), tipləşdirilməmiş fayllar isə qarışıq tipli verilənlərdən ibarət olur.

Misal.

```
Var metn:file of text;
    f_tam : file of Integer;
    gir_fayl: file of real;
    son_fayl: file;
```

İnformasiya daşıyıcısında yerləşən fiziki faylla işləmək üçün əvvəlcə bu faylla əlaqə yaratmaq lazımdır. Bunun üçün fayl dəyişənindən istifadə edilir. Fiziki faylla əlaqə **`Assign`** proseduru vasitəsilə yaradılır. Bu prosedurun ümumi forması belədir:

`Assign` (*fayl dəyişəni, faylın ünvanı*)**;**

Misal.

```
Assign( f, 'Kafedra.dat' );
```

Bu operatorla, `f` *fayl dəyişəni* aktiv diskin cari qovluğunda yerləşən `Kafedra.dat` fiziki faylı ilə əlaqə yaradır. Faylın tam adını və ona müraciət yolunu göstərməklə prosedurun cari qurğudan asılılığını aradan qaldırmaq olar:

```
Fayl:='d:\AAHM\Kafedra.dat';
Assign(f,Fayl);
```

Fayla yolun ümumi uzunluğu 79–a qədər simvoldan ibarət ola bilər.

Növbəti mərhələdə, informasiyanı fayldan oxumaq və ya fayla yazmaq üçün, bu fayl hökmən açılmalıdır. İnformasiyanı oxumaq üçün fayl **`Reset`** proseduru ilə açılır. Bu prosedurun ümumi forması belədir:

`Reset` (*fayl dəyişəni*)**;**

Fayla informasiyanı yazmaq üçün o, **Rewrite** proseduru ilə açılır. Bu prosedurun ümumi forması belədir:

Rewrite (*fayl dəyişəni*)

İstənilən məqsədlə açılmış fayl sonda hökmən bağlanmalıdır. Faylın bağlanması **Close** proseduru ilə həyata keçirilir. Bu prosedurun ümumi forması belədir:

Close (*fayl dəyişəni*)**;**

`Reset` proseduru *fayl dəyişəni* ilə əlaqədə olan mövcud fiziki faylı açır. Əgər fiziki fayl mətn faylıdırsa, onda onun elementlərini yalnız ardıcıl müraciətlə oxumaq mümkündür. Əgər fiziki fayl tipləşdirilmiş fayldırsa, onda o, həm ardıcıl, həm də birbaşa müraciətlə oxunub–yazıla bilər.

Əgər göstərilən adlı fiziki fayl yoxdursa, onda prosedurun icrasında səhv meydana çıxır. Kompilyatorun **{$I–}** direktivini daxil edərək, `IOResult` funksiyasının köməyi ilə, faylın açılması əməliyyatının nəticəsini təhlil etmək mümkündür. Bu funksiyanın qiyməti *0* olarsa, deməli əməliyyat müvəffəqiyyətlə sona çatmışdır, əks halda səhv var.

`Rewrite` proseduru *fayl dəyişəni* ilə əlaqədə olan yeni fiziki fayl yaradır. Əgər bu adda fiziki fayl mövcuddursa, onda həmin faylın məzmunu pozulacaq və yeni fayl yaradılacaqdır.

`Close` proseduru faylı bağlayır. Yadda saxlayın ki, hər bir açılmış fayl hökmən bağlanmalıdır.

Fayllarla işləyərkən **Eof (***fayl dəyişəni***);** funksiyasından da istifadə olunur. Əgər cari göstərici faylın sonuncu elementinin mövqeyində yerləşərsə və ya fayl boş olarsa, onda **Eof (***fayl dəyişəni***);** funksiyasının qiyməti *True,* əks halda *False* olur.

Misal. *Massiv.dat* faylının elementlərinin cəminin hesablanması.

```
program fayl_oxumaq;
uses crt;
```

```
var
   f : file of Integer;
   X : Integer;
   S : Longint;
begin
  ClrScr;
  {$I -}
  Assign (f,'Massiv.dat');{faylla əlaqə yaradılır }
  Reset (f);   { fayl açılır }
  {$I+}
  if IOResult <> 0 then
  begin
    writeln('Faylın açılması səhvdir ');
    Halt(1);
  end;
  S:= 0;
  while not Eof(f) do
  begin
    Read(f,X); { fayl oxunur }
    S:=S+X;
  end;
  writeln('Faylın elementlərinin cəmi=',S);
  Close (f); { fayl bağlanır }
end.
```

Bu proqramda, `assign` proseduru məzmunu tam ədədlər olan `Massiv.dat` faylı ilə əlaqə yaradır, `reset` proseduru ilə fayl açılır və bundan sonra, faylın bütün elementlərini oxumaq üçün ilkin şərtli dövr təşkil olunur. Faylın məzmunu `Read(f,X);` proseduru ilə oxunub, tam tipli `X` dəyişəninə mənimsədilir. Fayldakı informasiyanın miqdarı əvvəlcədən məlum olmadığı üçün dövr faylın sonu əlaməti rast gələnəcən təkrar olunur (`while not Eof(f) do`). Faylın açılmasında problem yarandıqda bu barədə məlumat verilir, `Halt` proseduru proqramın icrasını dayandırır və idarəetməni əməliyyat sisteminə ötürür.

8.3. Mətn faylları

Mətn fayllarını təsvir etmək üçün əvvəldən təyin olunmuş **Text** tipindən istifadə olunur:

Var *fayl dəyişəni* : **Text;**

Misal.

```
var
 MatnFile:Text;
 A,b:Text;
```

Mətn faylları praktiki olaraq ən çox tətbiq edilən fayllardır. Bu tip faylların üstün cəhəti ondan ibarətdir ki, burada verilənlər maşın kodları ilə deyil (tipləşdirilmiş və tipləşdirilməmiş fayllarda olduğu kimi), adi **ASCII** kodları ilə təsvir olunur. Belə ki, mətn fayllarında istənilən tip ədədlər, simvol və mətnlər saxlana bilər. Digər fayllardan fərqli olaraq, mətn faylları müxtəlif uzunluqlu sətirlərdən ibarət olur. Hər bir sətrin sonunda **#13**–karetkanın qaytarılması və **#10**–sətrin sonu simvolları olur. Sətrin hər bir simvoluna, birinci simvoldan başlayaraq, yalnız ardıcıl müdaxilə oluna bilər. Fərz edək ki, `f_ilk` faylının 4 –cü elementini `x` dəyişəninə mənimsətmək lazımdır. Bunun üçün əvvəlki üç elementi "boş–boşuna" oxumaq lazımdır, yəni:

```
Reset(f_ilk); { f_ilk faylını açır və kursoru
                1–ci elementin üzərinə yerləşdirir }
For i:=1 to 4 do
Read(f_ilk, x);
Reset(f_ilk); { kursoru yenidən 1–ci
                elementin üzərinə yerləşdirir }
```

Bu halda `x` dəyişəninə yalnız dördüncü sətirdəki qiymət mənimsədiləcəkdir.

Mətn tipli fayl yaratdıqda hər bir sətrin sonunda *Enter* klavişini basmaq lazımdır. Bu zaman sətrin sonuna **#13#10** kodları və ya xüsusi **EOLN** (*end of line*) əlaməti əlavə edilir.

Hər bir faylın sonu əlaməti olmalıdır, bu zaman **EOF** (*end of file*) əlaməti yaranır. Faylın sonu əlaməti ***Ctrl+Z*** klavişlərini birgə basmaqla yaradılır.

Mətn fayllarından oxunma və fayla yazma əməliyyatları standart `Read`, `Readln`, `Write` və `Writeln` prosedurları ilə yerinə yetirilir və onların ümumi forması belədir:

`Read` (*fayl dəyişəni, dəyişənlərin siyahısı*)**;**
`Write` (*fayl dəyişəni, dəyişənlərin siyahısı*)**;**
`Readln` (*fayl dəyişəni, dəyişənlərin siyahısı*)**;**
`Writeln` (*fayl dəyişəni, dəyişənlərin siyahısı*)**;**

`Read` və `Readln` prosedurları ilə informasiya fayldan oxunaraq *dəyişənlərin siyahısında* göstərilmiş dəyişənlərə mənimsədilir. `Write` və `Writeln` prosedurlarında isə *dəyişənlərin siyahısında* göstərilmiş dəyişənlərin qiymətləri fayla yazılır.

Misal.

```
Read(f,A,B);
Write(g,'A=',A,'B=',B);
Readln(f,C,D);
Writeln(g,'C=',C,'D=',D);
```

Burada, `f` və `g` uyğun olaraq oxunan və yazılan fayllarla əlaqə yaradan fayl dəyişənləridir.

Sonrakı izahatları isə aşağıdakı misal üzərində davam etdirək.

Misal. *f(x)=x/(1+x)* funksiyasının cədvəlləşdirilməsi.

```
program text_file;
uses CRT;
var x:real;k:word;
    out_file:text;
function F(x:real):real;
 begin
    F:=x/(1.+x);
 end;
```

```
begin
 assign(out_file,'d:\user\qarayev\table.dat');
 rewrite(out_file);
 x:=0.0;
 writeln(out_file,'F(x)=x/(1+x)
 funksiyasının cədvəl qiymətləti');
 writeln(out_file);
 writeln(out_file,'x':10,'F(x)':25);
 writeln(out_file);
 for k:=0 to 50 do
 begin
  writeln(out_file,x:9:3,F(x):25:10);
  x:=x+0.1;
  if k mod 10=9 then writeln(out_file);
 end;
 close(out_file);
end.
```

Burada, `out_file` dəyişəni `Text` tipli mətn faylı dəyişəni kimi elan edilir. `Assign` proseduru bu dəyişəni `D:` diskindəki `user\qarayev` qovluğunda yerləşən `table.dat` faylı ilə əlaqələndirir. Əgər faylı proqramın yerləşdiyi qovluqda yaratmaq lazımdırsa, onda fayla yolu yazmamaq olar. `Rewrite` proseduru informasiya yazmaq üçün həmin faylı açır. Əgər diskdə eyniadlı fayl artıq mövcud olarsa, onda həmin faylın məzmunu pozulacaq, oraya yeni informasiya yazılacaqdır. Fayl mövcud olmazsa, onda o, yaradılacaqdır. Fayl açıldıqdan sonra, birinci dörd `writeln` prosedurları ilə fayla mətn və boş sətirlər yazılır. *x/(1+x)* funksiyasının qiymətləri isə fayla dövr operatoru daxilində yazılır. Bu zaman hər 10 sətirdən bir faylda bir boş sətir buraxılır. Fayla informasiya yazıldıqdan sonra, fayl `close(out_file)` proseduru ilə bağlanır.

Mətn faylları üçün əlavə olaraq aşağıdakı prosedur və funksiyalardan istifadə etməyə icazə verilir:

- **Append** –faylın sonuna elementləri əlavə etmək üçün mövcud faylı açır;

- **Flush** –faylın cari ölçüsünü qaytarır;
- **Readln** –`Read` proseduru kimi işləyir. Əlavə olaraq cari sətirdə qalan bütün simvolları buraxaraq göstəricini mətn faylının növbəti sətrinə yerləşdirir;
- **SeekEof** –mətn faylı üçün **Eof** vəziyyətini qaytarır;
- **SeekEoln** –mətn faylı üçün Eoln vəziyyətini qaytarır;
- **SetTextBuf** –mətn faylı üçün daxiletmə–xaricetmə buferi təyin edir;
- **Writeln** –`Write` proseduru kimi işləyir. Əlavə olaraq mətn faylına **Eoln** –"sətrin sonu" işarəsini yazır.

8.4. Tipləşdirilmiş fayllar

Tipləşdirilmiş fayllar belə təsvir olunur:

Var *fayl dəyişəni* **: file of** *faylın tipi***;**

Burada, *fayl dəyişəni* tipləşdirilmiş fayl dəyişəni, *faylın tipi* isə fayl tipi istisna olmaqla, istənilən tip ola bilər (`integer`, `longint`, `real`, `double`, `extended`, *massiv, yazı* və s.).

Misal.

```
type
    Nomr = file of Integer;
    Simv = file of 'A'..'Z';

Var     x: file of real;
    N: nomr;
    S: simv;
```

Tipləşdirilmiş faylların bütün elementləri eyni tipli olmaqla bərabər, fayl tipindən başqa, ixtiyari tipli ola bilər.

Ədəd tipli verilənləri yadda saxlamaq üçün tipləşdirilmiş fayllardan istifadə etmək əlverişlidir. Çünki, bu halda informasiya daha yığcam şəkildə yadda saxlanır. Tipləşdirilmiş fayllara ardıcıl və birbaşa müraciət etmək mümkündür. Qeyd etmək lazımdır ki, tipləşdirilmiş faylların elementləri həmişə sıfırdan başlayaraq nömrələnir.

Tipləşdirilmiş fayllardan informasiyanı oxumaq yalnız `Read` proseduru ilə, fayla informasiyanı yazmaq isə `Write` proseduru ilə yerinə yetirilir. Bu prosedurların ümumi forması aşağıdakı kimidir:

`Read` (*fayl dəyişəninin adı, dəyişənlərin siyahısı*)**;**
`Write` (*fayl dəyişəninin adı, dəyişənlərin siyahısı*)**;**

Tipləşdirilmiş fayllarla əməliyyatlarda aşağıdakı prosedur və funksiyalar nəzərdə tutulmuşdur:

- **`FilePos`** –göstəricinin fayldakı cari mövqeyinin nömrəsini qaytarır;
- **`FileSize`** –faylın cari ölçüsünü (elementlərinin sayını) qaytarır;
- **`Seek`** –göstəricinin fayldakı cari mövqeyini verilmiş nömrəli elementə dəyişdirir;
- **`Truncate`** –faylın ölçüsünü göstəricinin cari mövqeyinə qədər qısaldır. Faylın cari mövqedən sonrakı bütün elementləri silinir.

Misal. Mətn faylları ilə tipləşdirilmiş faylları müqayisə etmək üçün aşağıdakı proqrama baxaq. Bu proqramda *Sin(x)* funksiyası *0,001* addımı ilə cədvəlləşdirilir və alınmış *1000* ədəd qiymət həm mətn faylına, həm də tipləşdirilmiş fayla yazılır.

```
{$N+}
program file_of_extended;
uses CRT;
var extfile:file of extended;
    textfile:text;
```

```
      x,y:extended;
      i:word;
begin
    { Mətn faylının yaradılması }
  assign(textfile,'table.txt');
  rewrite(textfile);
  x:=0.0;
  for i:=1 to 1000 do
  begin
    y:=sin(x);
    writeln(textfile,y);
    x:=x+0.001;
  end;
  close(textfile);
    { Tipləşdirilmiş faylın yaradılması }
  assign(extfile,'table.ext');
  rewrite(extfile);
  x:=0.0;
  for i:=1 to 1000 do
  begin
    y:=sin(x);
    write(extfile,y);  {writeln yazıla bilməz}
    x:=x+0.001;
  end;
  close(extfile);
end.
```

Proqram icra olunduqdan sonra, proqramın yerləşdiyi qovluqda `table.txt` və `table.ext` faylları yaranacaqdır. `table.txt` faylı mətn faylı, `table.ext` faylı isə tipləşdirilmiş fayldır. Baxmayaraq ki, hər iki faylın məzmunu eynidir, onlar arasında kəskin fərqlər vardır. Əgər mətn faylının məzmununa baxsaq, görərik ki, burada ədədlər bir sütun üzrə yerləşmişdir. Tipləşdirilmiş faylda isə xaotik yerləşmiş simvollar yığımı görəcəyik. Əgər bu faylların ölçülərini müqayisə etsək, görəcəyik ki, `table.ext` faylı ~10 kilobayt, `table.txt` faylı isə ~25 kilobayt həcmə

malikdir. Bu ona görə belədir ki, `Extended` tipli dəyişənin uzunluğu 10 baytdır, ona görə də faylda 1000 həqiqi ədəd 10000 bayt (~10 kilobayt) yer tutur. Lakin, `Extended` tipli dəyişəni mətn sətirləri kimi yadda saxladıqda, bu sətir 23 simvoldan (*17 rəqəm*, üstəgəl *6* simvoldan ibarət *dərəcə,* yəni $E \pm nnnn$) və üstəgəl *sətrin sonu* və *karetkanın qaytarılması* simvollarından (cəmi 25 simvol) ibarət olduğu üçün, fayl ~25 kilobayt olacaqdır. Elə bu səbəbdən də rəqəm tipli verilənləri `Extended` tipli fayllarda yadda saxlamaq daha məqsədəuyğundur.

Tipləşdirilmiş faylda istənilən elementə müraciət etmək üçün

Seek (**Var** *fayl dəyişəni*; *n*:**LongInt**);

proseduru tətbiq edilir. Burada *n* müraciət ediləcək komponentin sıra nömrəsidir (1–ci komponentin nömrəsi sıfırdan başlayır).

Tipləşdirilmiş faylın ölçüsünü müəyyən etmək üçün

FileSize (**Var** *fayl dəyişəni*):**LongInt**;

funksiyasından istifadə edilir. Bu funksiya *fayl dəyişəni* ilə əlaqələndirilmiş faylın uzunluğunu müəyyən edir, əgər fayl boşdursa, sıfır qiyməti qaytarır. `Seek` və `FileSize` funksiyaları mətn faylları üçün tətbiq edilə bilməz.

8.5. Tipləşdirilməmiş fayllar

Tipləşdirilməmiş fayllarda müxtəlif tipli dəyişənlərin qiymətləri saxlanır. Belə faylları *ikilik fayllar* da adlandırırlar, çünki bu faylların məzmunu ikilik ədədlərdən ibarət olur. Tipləşdirilməmiş fayllarda onun tipi və strukturundan asılı olmayaraq istənilən elementə birbaşa müraciət etmək olur. Tipləşdirilməmiş faylları təsvir edərkən yalnız **file** işçi sözündən istifadə olunur və onun ümumi forması belədir:

Var *fayl_dəyişəni* **: File;**

Misal.

```
var Y: file;
```

Tipləşdirilmiş fayllar üçün nəzərdə tutulmuş bütün standart alt proqramlar həm də tipləşdirilməmiş fayllar üçün istifadə oluna bilər. Lakin, bu zaman `Read` və `Write` prosedurları müstəsnalıq təşkil edir. Belə ki, tipləşdirilməmiş fayllar üçün `Read` və `Write` prosedurlarının əvəzinə **BlockRead** və **BlockWrite** prosedurlarından istifadə olunur. Bu prosedurların ümumi forması belədir:

BlockRead (**var** *fayl dəyişəni* **: file;**
var *Buf*; *Count* **: word**; *Result* **: word**);

BlockWrite (**var** *fayl dəyişəni* **: file;**
var *Buf*; *Count* **: word**; *Result* **: word**);

Burada, *Buf* – bufer dəyişəni, *Count* – oxunacaq və ya yazılacaq baytların miqdarı, *Result* parametri isə həqiqətən oxunmuş və ya yazılmış informasiyanın faktiki qiymətidir. *Result* parametrinin qiyməti prosedur tərəfindən müəyyən edilir. Digər parametrləri isə proqramçı müəyyən etməlidir. *Buf* parametri fayla yazılacaq və ya fayldan oxunacaq informasiyadır və o, ümumi həcmi *n*Count* olan massivdir (*n* – `Reset` prosedurundakı parametrdir). *Buf* parametri ilə ötürülən informasiyanın maksimal həcmi (*n*Count*) 65520 baytı aşa bilməz.

Tipləşdirilməmiş fayllarla işlədikdə `Reset` və `Rewrite` prosedurlarının ümumi forması belə olur:

Reset (**var** *fayl dəyişəni*; *n* **: Word**);
Rewrite (**var** *fayl dəyişəni*; *n* **: Word**);

Burada, *n* əlavə parametrdir və onu yazmadıqda yazının ölçüsü susmaya görə 128 bayta bərabər götürülür, lakin *n* parametrinin yerində 1 yazmaq məsləhət görülür. Onun digər qiymətlərində yazılar fayla tam yazılmaya bilər. Tipləşdirilməmiş faylların yazılması və oxunmasını başa düşmək üçün disketlərin quruluşu ilə tanış olaq.

Disklərdə informasiya konsentrik yollarda yazılır. Bu yollar sektorlar adlanan ayrı–ayrı bərabər hissələrə bölünür. Belə sektorların hər birinin ölçüsü 512 bayta bərabərdir. Qeyd edək ki, yalnız eyni yollar üçün bu sektorların fəza uzunluqları eynidir. Yol mərkəzdən nə qədər uzaqda yerləşərsə, sektorun uzunluğu bir o qədər böyük olacaqdır. Bu onunla izah olunur ki, disk sabit bucaq sürəti ilə fırlandıqda elementin xətti sürəti onun yerləşdiyi yolun radiusu ilə mütənasibdir. İstənilən fayl klaster adlanan eyni miqdarda informasiyalardan ibarətdir. Diskin bir dövrü ərzində klasterə informasiya yazmaq və ya oxumaq olar. Ona görə də, əgər klasterin ölçüsü informasiyanın miqdarı ilə eyni olarsa, onda verilənlərin köçürülməsi ən yüksək sürətlə baş verir. Diskin klasteri 2 qarışıq sektordan ibarət olduğu üçün onun ölçüsü 1024 baytdır. Vinçesterin klasteri 4 və ya 8 qarışıq sektordan ibarət olduğu üçün onun ölçüsü uyğun olaraq 2048 və 4096 bayt olur.

8.6. Qovluq və fayllarla iş üçün ümumi vasitələr

Qovluq və fayllarla işləmək üçün **`System`** modulunun tərkibinə çoxlu prosedur və funksiyalar daxil edilmişdir. Bu prosedur və funksiyalara müraciət etmək üçün `uses` bölməsinə **`System`** modulunu qoşmaq vacib deyildir. Bu prosedur və funksiyaları çağırmaq həmişə mümkündür.

Qovluqlarla iş prosedurları. Qovluqlarla iş üçün **`System`** moduluna Ms DOS sisteminin analoji əmrləri olan **`ChDir`**, **`MkDir`**, **`RmDir`**, **`GetDir`** prosedurları daxil

edilmişdir. Bu prosedurlardan istifadə etməklə proqram tərtib edək.

Misal.

```
program karaloqla_ish;
uses crt;
var
   S: String;
begin
 Clrscr;

 { D: diskində cari qovluğu təyin et }
 ChDir('D:\');

 { Cari disk və qovluğu göstərmək }
 GetDir(0,S);
 Writeln('Cari disk və qovluq:', S);

 {Kat_azal alt qovluğunun yaradılması }
 MkDir('Kat_azal');

 { Kat_azal alt qovluğuna keçid}
 ChDir('Kat_azal');

 { Cari disk və qovluğu göstərmək }
 GetDir(0,S);
 Writeln('Cari disk və qovluq:', S);

 { D: diskində cari qovluğu təyin et }
 ChDir('\');

 { Kat_azal alt qovluğunun silinməsi }
 RmDir('Kat_azal');

 { Cari disk və qovluğu göstərmək }
 GetDir(0,S);
 Writeln('Cari disk və qovluq:', S);
end.
```

Faylların adının dəyişdirilməsi və pozulması. **`Rename`** prosedurundan fiziki faylların adının dəyişdirilməsində, **`Erase`** prosedurundan isə faylların pozulmasında istifadə olunur. Qeyd edək ki, bu prosedurlar hər hansı fiziki faylla əlaqəli olan məntiqi dəyişənlər üçün yerinə yetirilir. Aşağıdakı proqramda əvvəl `Ilkfile.Dat` faylının adı dəyişərək `Deifile.Dat` olur, sonra isə bu fayl pozulur.

Misal.

```
program fayl_ad;
uses crt;
var
   f: file;
begin
 Assign(f, 'Ilkfile.Dat');
 { Faylın adının dəyişdirilməsi }
 Rename(f, 'Deifile.Dat');
 Erase(f)
 Writeln('Cari disk və qovluq:', S);
end.
```

Fayllarla iş üçün prosedur və funksiyalar. Aşağıda təsvir olunan prosedur və funksiyalardan istənilən növ fayllarda istifadə etmək olar.

`GetFTime` proseduru. Faylın yaradılma və ya sonuncu yeniləşdirmə vaxtını qaytarır. Müraciət forması aşağıdakı kimidir:

`GetFTime` (*fayl dəyişəni, vaxt*)**;**

Burada, *fayl dəyişəni* – fayl dəyişəni kimi proqramda elan olunan identifikator, *vaxt* – isə `Longint` tipində olan dəyişəndir.

`SetFTime` proseduru. Faylın yaradılma və ya sonuncu yeniləşdirmə vaxtını təyin edir. Müraciət forması belədir:

`SetFTime` (*fayl dəyişəni, vaxt*)**;**

Burada, *vaxt* – yığcam formatdadır.

Qeyd edək ki, **DOS.TPU** modulunda aşağıdakı kimi DateTime tipi elan olunmuşdur:

```
type
   DateTime = record

     year :Word;  { 19XX formatlı il }
     month:Word;  { ay 1..12 }
     day  :Word;  { gün 1..31 }
     hour :Word;  { saat 0..23 }
     min :Word;   { dəqiqə 0..59 }
     sec :Word;   { saniyə 0..59 }

   end;
```

Yığcam formatdakı vaxtı Longint tipli dəyişənə çevirmək üçün aşağıdakı prosedurdan istifadə olunur:

```
PackTime(var T: DateTime;
         var Time:Longint);
```

Longint tipli vaxtı geniş formatlı tipli dəyişənə çevirmək üçün aşağıdakı prosedurdan istifadə olunur:

```
UnpackTime(var Time:Longint;
           var T: DateTime);
```

FExpand funksiyası. Faylın adına tam spesifikasiya, yəni disk, qovluq və faylın tipi əlavə edir. Müraciət forması belədir:

Fexpand (*faylın adı*)**;**

Burada, *faylın adı* sətir tipli ifadədir.

Fsearch funksiyası. Qovluqlar siyahısından faylın axtarışını təşkil edir. Müraciət forması belədir:

Fsearch (*faylın adı, qovluqların siyahısı*)**;**

Qeyd edək ki, faylın adında onun yolu da göstərilə bilər.

FindFirst proseduru. Göstərilən qovluqda qeyd olunan fayllardan birincisinin atributunu qaytarır. Müraciət forması:

FindFirst (*sətir tipli ifadə, atribut, ad*)**;**

Burada, *sətir tipli ifadə* – fayl və ya fayllar qrupunu, *atribut* – `Byte` tipli ifadədir. Fayl atributları **DOS.TPU** modulunda aşağıdakı kimi elan olunur:

```
const
   ReadOnly  = $01;   { yalnız oxumaq üçün }
   Hidden    = $02;   { gizlədilmiş fayl }
   SysFile   = $04;   { sistem fayl ı}
   VolumeID  = $08;   { diskin identifikatoru }
   Directory = $10;   { alt qovluğun adı }
   Archive   = $20;   { arxiv faylı }
   AnyFile   = $3F;   { istənilən fayl }
```

Qeyd edək ki, bu baytdakı bitlərin kombinasiyasından istifadə etməklə, müxtəlif variantları göstərmək olar. Məsələn, `$06` – bütün gizli və sistem fayllarını seçir.

`FindFirst` prosedurunun yerinə yetirilməsi nəticəsində `SearChrec` tipli dəyişən qaytarılır. Bu tip **DOS.TPU** modulunda aşağıdakı kimi təyin olunur:

```
type
   SearChrec = record
     Fill : array [1..21] of Byte;
     Attr : Byte;
     Time : LongInt;
     Size : LongInt;
     Name : String [12]
   end;
```

Burada, `Attr` – faylın atributu, `Time` – faylın yaradılmasının və ya sonuncu yeniləşdirməsinin yığcam formatdakı

vaxtı, `Size` – faylın baytlarla uzunluğu, `Name` – faylın adı və tipidir.

`FindNext` proseduru. Göstərilən qovluqdakı növbəti faylın adını qaytarır. Müraciət forması:

`FindNext` (*növbəti fayl*)**;**

Misal. `FindFirst` və `FindNext` prosedurlarından istifadə üsulları.

```
program fayl_siyahi;
uses crt,dos;
var
   S:SearchRec;
begin
 FindFirst('*.pas',AnyFile,S);
 while DosError = 0 do
 begin
  with S do
  writeln(Name:12,Size:12);
  FindNext(S);
 end
end.
```

Proqram cari qovluqdakı bütün *.pas* tipli faylların siyahısını ekrana çıxarır.

`GetFAttr` proseduru. Faylın atributunu – adını qaytarır. Müraciət forması:

`GetFAttr` (*fayl dəyişəni, atribut*)**;**

Burada, *atribut* `Word` tipli dəyişəndir.

`SetFAttr` proseduru. Faylın atributunun təyininə imkan verir. Müraciət forması:

`SetFAttr` (*fayl dəyişəni, atribut*)**;**

`DiskFree` funksiyası. Göstərilən diskdə boş sahənin baytlarla ölçüsünü qaytarır. Müraciət forması:

`DiskFree`(*disk*)**:**`LongInt`**;**

Funksiyaya müraciətdə `Byte` tipli *disk* dəyişəni diskin nömrəsini təyin edir.

`DiskSize` funksiyası. Göstərilən diskin baytlarla tam ölçüsünü qaytarır. Müraciət forması:

```
DiskSize (disk):LongInt;
```

Mövcud olmayan disk göstərilərsə, onda funksiya (–1) qiymətini qaytarır.

8.7. Fayllarla praktiki iş

Misal. Simvolları `Char` tipli fayla yazmalı və bu simvolların **ASCII** kodlarını ekrana çıxarmalı.

```
program Char_tipli_fayl;
uses crt;
var
   FC : file of Char;
   FB : file of Byte;
   Ch : Char;
   B  : Byte;
Begin
 ClrScr;
 { Fayl yaradılır }
 Assign (FC, 'D:\Test.dat');
 Rewrite (FC);
 for Ch := '0' to '9' do Write (FC, Ch);
 for Ch := 'A' to 'K' do Write (FC, Ch);
 Close (FC);
 { Fayl oxunur}
 Assign (FB, 'D:\Test.dat');
 Reset (FB);
  while not Eof(FB) do
   begin
    Read (FB,b);
    Write (b:8);
   end;
```

```
  Close (FB);
end.
```

Bu proqramda, əvvəl `Char` tipli fayla bir sıra simvollar yazılır, sonra bu `Byte` tipli fayl kimi açılır. Nəticədə bu fayla yazılan simvolların **ASCII** kodu çap olunur.

Misal. Faylın `A:` disketindən `D:` diskinə köçürülməsi (tipləşdirilməmiş fayl).

```
Program N_T_File;
uses crt;
const
    Razmer=1024; { Disk klasterinin ölçüsü }
    Ilk_file='A:\ilk_file' ;
    Son_file='d:\son_file' ;
var
    ilk, son:File;
    Bufer : array[1.. Razmer] of Byte;
    Rezultat:word;
begin
 Assign(ilk, ilk_file);
 Reset(ilk, Razmer);   { fayl oxunmaq üçün açılır }
 Assign(son, son_file);
 Rewrite(son, Razmer);    { fayl yazılmaq üçün açılır }
 While not EOF(ilk) do
  Begin
  {ilk faylından informasiya porsiyasını buferə oxuyuruq }
   BlockRead(ilk,Bufer,1,Rezultat);

  {son faylına buferdən fayl porsiyası yazırıq }
   BlockWrite(son,Bufer,1);
  end;
 Close(ilk);
 Close(son);
end.
```

Misal. Mətn faylındakı rəqəmlərin və latın hərflərinin ayrılıqda sayının tapılması.

```
program Rqem_herf;
uses crt;
type
    Coxlug = set of Char;
const
    Nomr : Coxlug = ['0'..'9'];
    Harf : Coxlug = ['a'..'z', 'A'..'Z'];
var
   Snomr,
   Sharf : Word;
   Fp    : Text;
   Ch    : Char;
begin
 ClrScr;
 Assign (Fp, 'D:\NIZAMI.TXT');
 Reset (Fp); { fayl oxunmaq üçün açılır }
 Snomr:=0;Sharf:=0;
 while not eof (Fp) do
  begin
    Read(Fp, Ch);
    if Ch in Nomr then inc(Snomr);
    if Ch in Harf then inc(Sharf);
  end;
 close(Fp);
 writeln('Rəqəmlərin sayı = ',Snomr:5);
 writeln('Hərflərin sayı', Sharf : 5);
end.
```

Bu proqramı icra etməzdən əvvəl, *Notepad* (*Bloknot*) mətn redaktoru vasitəsilə `Nizami.txt` adlı fayl yaratmaq lazımdır (çünki, Turbo Pascal yalnız **ASCII** simvollarını tanıyır). Proqramda iki çoxluq tipi müəyyənləşdirilmişdir: `Nomr` adlı `['0'..'9']` rəqəm formalı simvollar çoxluğu və `Harf` adlı `['a'..'z', 'A'..'Z']` kiçik və baş hərflər çoxluğu. `fp` məntiqi fayl dəyişəni `Nizami.txt` faylı ilə əlaqə yaradır. Faylın bütün simvollarını oxumaq üçün ilkin şərtli dövr operatorunda şərt kimi faylın sonu əlaməti yazılmışdır. Hər dövrdə fayl

oxunaraq onun simvolları simvol tipli `ch` dəyişəninə mənimsədilir. Bu dəyişənin aldığı qiymətlərin `Nomr` və `Harf` çoxluğuna mənsub olması (`in` əməliyyatı) yoxlanır. Əgər simvol çoxluğa mənsubdursa, rəqəm və hərf sayğaclarının üzərinə vahid əlavə olunur, yəni (`if Ch in Nomr then inc(Snomr);` və `if Ch in Harf then inc(Sharf);`).

Misal. Mətn faylındakı bəzi simvolların sayının tapılması.

```
program Simvollar_sayi;
uses crt;
var
   gir_fayl:text;
   s: string;
   i,vergul,cumle,probel,setir:integer;
begin
 clrscr;
 vergul:=0; cumle:=0;
 probel:=0; setir:=0;
 Assign (gir_fayl,'D:\NIZAMI.TXT');
 Reset (gir_fayl);
 while not eof(gir_fayl) do
  begin
   Readln(gir_fayl, s);
   for i:=1 to Length(s) do
    begin
      case s[i] of
        ',': inc(vergul); { Vergüllərin sayı }
        '.': inc(cumle);  { Cümlələrin sayı }
        ' ': inc(probel); { Probellərin sayı }
      end;
    end;
   inc(setir); { Sətirlərin sayı }
  end;
 close(gir_fayl);
 GotoXY(1,3);
```

```
    Writeln('Vergullərin sayı :', vergul);
    Writeln('Cumlələrin sayı  :', cumle);
    Writeln('Probellərin sayı :', probel);
    Writeln('Sətirlərin sayı  :', setir);
    Writeln(s);
  end.
```

Bu proqramda, `assign` proseduru ilə `Nizami.txt` adlı fayl ilə əlaqə yaradılır, `reset` proseduru ilə fayl açılır və bundan sonra, faylın bütün elementlərini oxumaq üçün ilkin şərtli dövr təşkil olunur. Fayldakı informasiyanın miqdarı əvvəlcədən məlum olmadığı üçün, dövr faylın sonu əlaməti rast gələnəcən təkrar olunur (`while not Eof(gir_fayl) do`). Fayldan informasiya sətirbəsətir oxunur və sətirdəki hər bir simvolu təhlil etmək üçün `for` operatoru vasitəsilə yenidən dövr təşkil olunur. Dövr sətirlərdəki simvolların sayı qədər təkrar olunur (`for i:=1 to Length(s) do`). Burada, sətirlərin uzunluğunu `Length(s)` funksiyası hesablayır. `Case` operatoru vasitəsilə simvolların vergül, nöqtə və ya probelə uyğunluğu yoxlanır və sayğac vasitəsilə onların sayı tapılır. Sətirlərin sayı isə ilkin şərtli dövrün gövdəsində tapılır (`inc(setir);`).

Misal. *A(4,4)* matrisinin bütün müsbət elementlərinin və onların indekslərinin tapılması və nəticələrin faylda yadda saxlanılması.

```
program Musbet_element;
uses crt;
var a:array[1..4,1..4] of real;
    n,i,j:word;
    netice:text;
begin
 for i:=1 to 4 do
  begin
    write(i,'-ci sətir elementlərini
           daxil edin:');
```

```
      for j:=1 to 4 do readln(a[i,j]);
    end;
  clrscr;
  Assign(netice,'musbet_el');
  rewrite(netice); { fayl yazılmaq üçün açılır }
  writeln(netice,'sətir':6,'sütun':6,
          ' musbət elementlər':16);
  for i:=1 to 4 do
  for j:=1 to 4 do
  if a[i,j]>=0 then
  writeln(netice,i:6,j:6,' ':12,a[i,j]:8:4);
  close(netice);
  write('Proqramin sonu');
end.
```

Burada, *A(4,4)* matrisinin elementləri klaviaturadan daxil edilir, proqramın nəticələri isə `musbet_el` adlı fayl yaradılaraq orada saxlanır. Bu fayl pascal kompilyatorunun yerləşdiyi qovluqda yaradılacaqdır (çünki, proqramda faylın hansı qovluqda yaradılması üçün xüsusi yol göstərilməmişdir). `Assign` proseduru vasitəsilə `musbet_el` adlı fayl ilə əlaqə yaradılır, `rewrite(netice);` proseduru ilə fayl yazılmaq üçün açılır və ikiqat dövr daxilində müsbət elementlər tapılaraq `writeln( netice,i:6,j:6,' ':12, a[i,j]:8:4);` proseduru ilə fayla yazılır. Sonda fayl bağlanır (`close(netice);`).

Misal. *A(5,5)* matrisinin hər bir sətir elementlərini cəmləyib, nəticələri faylda saxlayın.

```
program set_cem;
uses crt;
var a:array[1..5,1..5] of real;
    i,j:word;
    s:File of real;
    s1:real;
begin
```

```
    for i:=1 to 5 do
     begin
       write(i,'-ci sətir elementlərini
              daxil edin:');
       for j:=1 to 5 do readln(a[i,j]);
     end;
    clrscr;
    Assign(s,'son');
    rewrite(s);
    for i:=1 to 5 do
     begin
       s1:=0.;
       for j:=1 to 5 do
       s1:=s1+a[i,j];
       write(s,s1);
     end;
    close(s);
    write('Proqramın sonu');
  end.
```

Bu proqramın iş prinsipi əvvəlki proqramın iş prinsipinə tamamilə analojidir.

Doqquzuncu fəsil

QRAFİKLƏRİN PROQRAMLAŞDIRILMASI

Bu fəsildə Turbo Pascal dilinin qrafik imkanları araşdırılacaq, ekranın mətn və qrafik rejimlərini, qrafik koordinat sistemini, qrafik rejimin qoşulması və ondan çıxış qaydalarını öyrənəcəyik. **Graph** moduluna daxil olan qrafik rejimin əsas sabitləri və alt proqramları, təsviri ekrana çıxarma prosedurları, qrafik primitivlərin qurulması və mətnin ekrana çıxarılması prosedurları müfəssəl izah olunacaqdır. Qrafik təsvirlərin qurulması üçün çox zəngin və maraqlı məsələlər proqramlaşdırılacaqdır.

9.1. Mətn və qrafik rejimlər

Qrafik proqramlaşdırmanın əsaslarını öyrənməzdən əvvəl, qrafik rejimin nə olduğunu araşdıraq. Məlumdur ki, informasiyanın təsvir edilməsi üçün ən çox monitordan istifadə edilir. Monitor (ekran) iki rejimdə işləyir: mətn və qrafik rejimlər. *Mətn rejimində* ekran 25∗80=2000 sayda mövqelərdən ibarət olur. Buna mətn rejimində *ekranın yolvermə qabiliyyəti* deyilir. Bu halda ekranda təsvir edilən ən kiçik obyekt *simvol* olur. Həm də bu halda monitorun rəng imkanları çox məhdud olur və bu rejimdə əsasən mətnlər təsvir edilir, lakin bəzi qrafik təsvirlər də ekrana çıxarıla bilər. Belə təsvirlər isə çox keyfiyyətsiz və primitiv olacaqdır.

Qrafik təsvirlərlə işlədikdə proqramçının sərəncamında olan rənglər yığımı böyük əhəmiyyətə malikdir. Rənglər

yığımı *rənglər palitrasını* təşkil edir. Mümkün rənglərin miqdarı isə displey və videoadapterin imkanlarından və habelə videorejimdən asılıdır. Proqramlaşdırmada adətən üç rəngin – *qırmızı* (*Red*), *yaşıl* (*Green*) və *göy* (*Blue*) rənglərin qarışığından istifadə edilir ki, buna da *RGB təsvir sxemi* deyilir. *RGB* sxemin istifadə edilməsi elektron – şüa borusunun konstruktiv xüsusiyyətlərindən asılıdır. Elektron – şüa borusunda hər bir qrafik – nöqtə yuxarıda göstərilən rənglərdən ibarət olur. Hər bir əsas rəng isə 256 intensivliyə malik olduğundan, rəng və çalarların ümumi sayı 256∗256∗256= 16 777 216 olacaqdır. Lakin bütün qurğular, o cümlədən, şırnaqlı printerlər bu qədər rəngi təsvir etmək qabiliyyətinə malik deyildir. Ona görə də təsvirin çap edilməsi üçün avtomatik olaraq printerin imkan verdiyi rənglər seçilir. Analoji qayda şrift simvollarına da aiddir.

Monitoru qrafik rejimə keçirmək üçün xüsusi inisializasiya əməliyyatı yerinə yetirmək lazımdır. Bu əməliyyatdan sonra, **`Graph`** modulunun standart alt proqramlarının köməyi ilə, istənilən rəngli, istənilən qrafik çəkmək olar. Qrafik rejimdə ekran nöqtələrdən ibarət olur ki, bunlara da *piksellər* (*"picture element"*) deyilir. Simvolla müqayisədə piksel çox kiçikdir. Qrafik rejimdə ekran, məsələn, *EGA tipli* adapter 640∗350 (350 *sətir*, hər sətirdə 640 *piksel*) sayda piksellərdən ibarət olur. Buna qrafik rejimdə *ekranın yolvermə qabiliyyəti* deyilir. Pikselin parametrlərini (rəng, parlaqlıq) yadda saxlamaq üçün videoyaddaş və ya videobuferdən istifadə edilir. Şəklin fasiləsiz təsviri üçün onu ən azı 25–30 hers tezliklə təkrar etmək lazımdır. Bundan kiçik tezliklərdə xoşagəlməz və yorucu təsvirlər alınır.

Təsviri yadda saxlamaq və təkrar etmək üçün kompüterin daxilində xüsusi plata vardır ki, ona *videokart* deyilir. Müasir videokartlar şəkli 100 və daha artıq hers tezliklə təkrarlayır.

Sətirlərin və sətirlərdə piksellərin sayı adapterdən və onun rejimlərindən asılıdır. Hər bir adapterin altıya qədər rejimləri ola bilər. Bu rejimlər bir–birindən ekranın yolvermə

qabiliyyəti və mümkün rənglər yığımı ilə fərqlənir. Adapterlərin adları (identifikatorlar) və onların rejimləri **`Graph`** modulunun sabitlər (`Const`) bölməsində göstərilir. Bəzi adapterlərin maksimal yolvermə qabiliyyətinə baxaq (mötərizədə adapterin rejimlərinin sayı göstərilmişdir):

- *EGA (Enhanced Graphics Adapter)* – 640∗350 (2);
- *VGA (Video Graphics Array)* – 640∗480 (4);
- *İBM 8514* – 1024∗768 (2);
- *SVGA (super –VGA)* – 1024∗768;
- *SVGA (super–VGA)* – 1280∗1024.

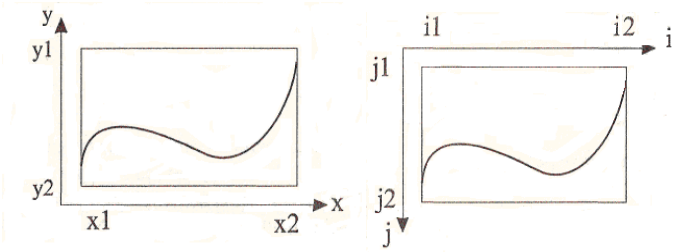
Monitor və adapter cansız qurğulardır. Onları idarə etmək üçün *drayver* adlanan xüsusi proqramlardan istifadə edilir. Məhz drayver sətirlərin, piksellərin sayını və s. müəyyən edir. Videoadapterlərin bütün drayverləri **Borland** firmasının yaratdığı `.bgi` (*Borland Graphics Interface*) tipli fayllarda yerləşir, məsələn `cga.bgi`, `egavga.bgi` və s.

9.2. Qrafik koordinat sistemi

Qrafik proqramlaşdırmada əsas çətinlik ondan ibarətdir ki, adi halda qrafiklər kağız üzərindəki dekart koordinat sistemində layihələndirilir, kompüterdə isə həmin qrafiklər monitorun qrafik koordinat sistemində təsvir edilməlidir. Monitorun koordinat sistemində isə koordinat başlanğıcı, yəni (*0,0*) nöqtəsi kimi, ekranın sol yuxarı küncü qəbul edilir (şəkil 9.1).

Hər hansı *y= f(x)* funksiyasını kağız üzərində qurmaq üçün *(X1,X2)∗(Y1,Y2)* ölçülü düzbucaqlı seçilir. Əsas problem bu funksiyanı ekranda çəkdikdə yaranır. Belə ki, bu halda *(X1,X2)∗(Y1,Y2)* düzbucaqlısını *(I1,I2)∗(J1,J2)* ekran düzbucaqlısına çevirmək lazım gəlir. Belə çevirmə üçün aşağıdakı düsturlardan istifadə edilə bilər:

$$\frac{x-X1}{X2-X1}=\frac{i-I1}{I2-I1},\quad \frac{y-Y1}{Y2-Y1}=\frac{j-J2}{J1-J2}.$$



Şəkil 9.1. Adi və ekran koordinat sistemləri

Koordinatları çevirmək üçün bu düsturları aşağıdakı funksiya tipli alt proqramlar şəklində tərtib etmək olar.

```
Function II(x:real):integer;
{ X oxu üzrə çevirmə }
Begin
  II:=I1+Trunc((X-X1)*(I2-I1)/(X2-X1));
End;

Funstion JJ(y:real):integer;
{ Y oxu üzrə çevirmə }
Begin
  JJ:=J2+Trunc((Y-Y1)*(J1-J2)/(Y2-Y1));
End;
```

Ekran koordinatları tam ədədlər olmalıdır, ona görə də baxılan proqramlarda `Trunc` funksiyası tətbiq edilmişdir.

Proqramlaşdırma zamanı hər bir pikselin ünvanına da müraciət etmək və onları rəngləmək olar. Ən sadə qrafik rejimdə 256 rəng, xətti modelli videoyaddaş və ekranın 320∗200 yolvermə qabiliyyəti istifadə olunur. 320∗200 yolvermə qabiliyyətində ekranın buferinin ölçüsü 320∗200∗1=64000 bayt olmalıdır. Mümkün rənglərin sayı 256 olduğu üçün, pikselin rəngi haqqında informasiyadan ibarət hər bir piksel üçün 1 bayt kifayət edir. Xətti modelli

videoyaddaşın piksellərinin ünvanları cədvəl 9.1 – də göstərilmişdir.

Cədvəl 9.1. Xətti modelli videoyaddaşın piksellərinin ünvanları

	x=0	**x=1**	**…**	**x=318**	**x=319**
y=0	0	2	…	318	319
y=1	320	321	…	638	639
…	…	…	…	…	…
y=199	63681	63682	…	63998	63999

Bu cədvələ uyğun olaraq qrafik koordinatda hər bir pikselin ünvanını belə tapa bilərik:

```
Ünvan:=320*y+x;
```

9.3. Qrafik rejimin qoşulması və ondan çıxış

Qrafik rejimdə işləmək üçün bir neçə üsul mövcuddur.

1. Turbo Pascal dilinin inteqrallaşdırılmış mühitini qrafik rejimə kökləmək lazımdır. Bunun üçün mühitin *Options* menyusundan *Directiories* əmrini icra edib, açılan pəncərədə *Tab* klavişinin köməyi ilə, *Unit Directiories* rejimi qarşısında **`Graph.tpu`** faylına yolu göstərmək lazımdır (bu faylda **`Graph`** modulu yerləşir), məsələn, `D:\BP\BGİ`.

2. Turbo Pascal dilinin inteqrallaşdırılmış mühitini kökləmədən, sadəcə olaraq, **`Graph.tpu`** faylını **turbo.exe** faylının yerləşdiyi qovluğa köçürmək lazımdır.

Hər iki üsulda ekranı qrafik rejimə keçirmək üçün proqramın `Uses` bölməsinə **`Graph`** və **`Graphs`** modullarını əlavə etmək, yəni

```
Uses CRT, Graph, Graphs;
```

sətrini, icraedici hissədə isə **`Open_Graph;`** prosedurunu yazmaq lazımdır.

3. Qrafik rejimə keçdikdə, proqram videoadapterin növünü müəyyən etməlidir. Proqramda videoadapterin növünü aşkar göstərmək olar və ya uyğun parametrlərin qiymətlərini proqram özü seçə bilər. Videoadapterin növünü aşkar şəkildə göstərdikdə

`İnitGraph` (*gd, gm,* `'C:\TP\BGI'` **);**

prosedurunu çağırmaq lazımdır. Burada *gd* və *gm* adları sərbəst seçilmişdir və Siz ixtiyari adlar yaza bilərsiniz. *gd* dəyişəni videoadapterin növünü müəyyən edir və proqramın icraedici hissəsində ona qiymət mənimsədilməlidir. Belə qiymət kimi videoadapterin adı və ya onun kodundan ibarət sabitlər istifadə oluna bilər. Belə sabitlərdən bir neçəsi cədvəl 9.2 – də göstərilmişdir.

Cədvəl 9.2. Videoadapterin sabitləri

Sabitlər	**Qiymətlər**
CGA	1
MCGA	2
EGA	2
EGA64	4
EGAMono	5
HercMono	7
ATT400	8
VGA	9
PC3270	10

Videoadapterin növünü avtomatik olaraq müəyyənləşdirmək üçün proqramda

`gd:=Detected;` və ya `gd:=0;`

yazmaq lazımdır. `gd` dəyişəninin qiyməti müəyyənləşdirildikdən sonra, *gm* dəyişəninə qiymət müəyyənləşdirilməlidir. Bu dəyişən videoadapterin qrafik rejimini müəyyən

edir. *gm* dəyişəninin ala biləcəyi mümkün qiymətlərdən bir neçəsi cədvəl 9.3 – də göstərilmişdir.

Cədvəl 9.3. Videoadapterin qrafik rejimləri

Sabitlər	Qiymətlər	Qrafik rejimin təsviri
EGALo	0	*640∗200, 16 rəng, 4 səhifə*
EGAHi	1	*640∗350, 16 rəng, 2 səhifə*
EGA64Lo	0	*640∗200, 16 rəng, 1 səhifə*
EGA64Hi	1	*640∗350, 4 rəng, 1 səhifə*
HercMonoHi	0	*720∗348 nöqtə, 2 səhifə*
VGALo	0	*640∗200, 16 rəng, 4 səhifə*
VGAMed	1	*640∗350, 16 rəng, 2 səhifə*
VGAHi	2	*640∗480, 16 rəng, 1 səhifə*
IBM8514Lo	0	*640∗480 nöqtə, 256 rəng*
IBM8514Hi	1	*1024∗768 nöqtə, 256 rəng*

Bu üsulda **`Open_Graph;`** prosedurunu yazmaq lazım deyil. Qrafik rejimlə işlədikdə səhv baş verərsə, **`İnitGraph`** proseduru sıfırdan fərqli nəticə – *səhv kodu* yaradır. Bu səhv haqqında məlumatı **`GraphResult`** funksiyası verir. Bunun üçün proqrama aşağıdakı proqram fraqmentini əlavə etmək lazımdır (sonrakı misallarda bu fraqment nümayiş etdiriləcəkdir):

```
uses Crt, Graph;
var Gd, Gm: Integer;
begin
  Gd := Detect;
  InitGraph(Gd,Gm,'c:\bp\bgi');
  if GraphResult <> grOk then
    begin
      Writeln('Qrafik rejimdə
               səhv aşkar edildi');
      Halt;
    end;
  ...
end.
```

Bu proqram fraqmentində

```
if GraphResult <> grOk then
```

sətri əvəzinə

```
if GraphResult <> 0 then
```

sətrini də yaza bilərik.

GraphResult funksiyası 15 məlumat verir. Onlardan üçünü göstərək:

- `GrOK=0` –*qrafik rejim uğurla yerinə yetirilmişdir;*
- `GrNoİnitGraph=-1` –*qrafik rejim müəyyən edilməmişdir;*
- `GrFileNotFound=-3` –*qrafik drayver tapılmamışdır.*

Ekranı qrafik rejimdən çıxarmaq üçün proqramda **CloseGraph;** prosedurunu çağırmaq lazımdır.

RestoreCrtMode; və **SetGraphMode;** prosedurları ilə qrafik rejimi bağlamadan, mətn rejiminə və əksinə keçmək olar.

9.4. Qrafik rejimin əsas sabitləri və alt proqramları

9.4.1. Qrafik rejimin əsas sabitləri

Qrafik təsvirləri tətbiq etmək üçün **Graph** kitabxanasında *50*–dən çox prosedur və funksiya mövcuddur. Əvvəlcə, mövcud olan *16* sabiti sadalayaq (cədvəl 9.4). Bu sabitləri rəqəm və ya söz formasında istifadə etmək olar. Əyanilik üçün sözlərdən istifadə etmək daha məqsədəuyğundur.

Qrafikləri çəkdikdə elə etmək lazımdır ki, bir monitordan digərinə keçdikdə onların həndəsi ölçüləri pozulmasın. Bu məqsədlə proqramda

```
GetMaxX:İnteger;
```

və

```
GetMaxY:İnteger;
```

Cədvəl 9.4. Əsas rəng sabitləri

Proqramda rəngin adına müraciət	Proqramda rəngin adına kodla müraciət	Rəngin adı
`Black`	0	*Qara*
`Blue`	1	*Göy*
`Green`	2	*Yaşıl*
`Cyan`	3	*Mavi*
`Red`	4	*Qırmızı*
`Magenta`	5	*Bənövşəyi*
`Brown`	6	*Qəhvəyi*
`Light Grey`	7	*Açıq boz*
`Dark Grey`	8	*Tünd boz*
`Light Blue`	9	*Açıq göy*
`Light Green`	10	*Açıq yaşıl*
`Light Cyan`	11	*Açıq mavi*
`Light Red`	12	*Çəhrayı*
`Light Magenta`	13	*Açıq qırmızı*
`Yellow`	14	*Sarı*
`White`	15	*Ağ*

funksiyalarından istifadə etmək lazımdır. Bu funksiyalar monitorun maksimal koordinatlarını müəyyən edir. Belə ki, monitorun koordinat başlanğıcı sol yuxarı küncdə ((*0,0*) nöqtəsi) olduğu halda, sonu ekranın sağ aşağı küncündə, yəni (`GetMaxX,GetMaxY`) nöqtəsində olur.

Təsviri ekrana çıxarmaq üçün xüsusi prosedurlar nəzərdə tutulmuşdur.

9.4.2. Təsviri ekrana çıxarma prosedurları

`ClearDevice;` – qrafik ekranı təmizləyir, fonun rəngini müəyyən edir və cari mövqe kimi (*0,0*) nöqtəsini təyin edir.

SetColor (*reng* **: word);** – qələmin rəngini *reng* rəngli edir (məsələn, `SetColor(Red);` və ya `SetColor(4);` – qırmızı rəng).

SetBkColor (*reng*: **word);** – fonun rəngini *reng* rəngli edir (məsələn, `SetBkColor(Green);` və ya `SetBkColor(2);` – yaşıl rəng).

Graph modulunun əsas hissəsi nöqtə, düz xətt parçaları, qövs, çevrə və s. çəkmək üçün prosedurlardan ibarətdir. Bu qrafik elementlərə *qrafik primitivlər* deyilir. Qrafik primitivləri çəkmək üçün aşağıdakı prosedurlar tətbiq edilir :

PutPiksel (*x*, *y* **: İnteger;** *reng*:**word);** – (*x*, *y*) koordinatlı nöqtədə pikseli *reng* rəngi ilə rəngləyir.

Line (*x1*, *y1*, *x2*, *y2* **: İnteger);** – cari rəngli rəqəmlə (*x1*, *y1*) başlanğıc və (*x2*, *y2*) – son nöqtələrdən keçən düz xətt parçası çəkir.

Circle (*x*, *y* **: İnteger;** *Radius*:**word);** – cari rəngli qələmlə mərkəzi (*x*, *y*) nöqtəsində yerləşən *Radius* – radiuslu çevrə çəkir.

Ellipse (*x*, *y* **: İnteger;** *Bash_bucaq*, *Son_bucaq*, *X_Radius*, *Y_Radius* **: word);**

proseduru cari rəngli qələmlə ellips çəkir. Burada, *Bash_bucaq* və *Son_bucaq* –ellips sektorunun başlanğıc və son bucaqlarıdır; (*x*, *y*)– ellipsin mərkəzidir.

Rectangle (*x1*, *y1*, *x2*, *y2*:**İnteger);** – cari rəngli qələmlə (*x1*, *y1*) – (*x2*, *y2*) diaqonallı düzbucaqlı çəkir.

Bar (*x1*, *y1*, *x2*, *y2* **: İnteger);** – sol yuxarı küncü (*x1*, *y1*), sağ aşağı küncü isə *(x2,y2)* nöqtəsində olan düzbucaqlı çəkir. Bu prosedurun `Rectangle` prosedurundan fərqi ondadır ki, çəkilən düzbucaqlının konturları olmur və düzbucaqlının daxilindəki sahə isə əvvəlcədən müəyyən edilmiş rənglə doldurulur.

FillEllipse(*x,y*:**İnteger**;*X_Radius,Y_Radius*:**Word**); – mərkəzi (*x, y*) nöqtəsində olan ellips çəkir və onun daxili sahəsi əvvəlcədən müəyyən edilmiş rənglə doldurulur.

Sektor (*x, y* : **İnteger**; *Bash_Bucaq, Son_Bucaq, X_Radius, Y_Radius* : **Word**);

proseduru cari rəngli qələmlə ellips sektoru çəkir və onun daxili sahəsi əvvəlcədən müəyyən edilmiş rənglə doldurulur.

Qapalı həndəsi fiqurların rənglə doldurulması aşağıdakı prosedurlar vasitəsilə yerinə yetirilir :

SetFillStyle(*Doldurma*, *reng* : **Word**); – `Bar`, `FillEllipse`, `Sektor` və s. prosedurları ilə çəkilmiş fiqurları *reng* rəngi ilə doldurur. *Doldurma* parametri doldurma tərzini müəyyən edir. Doldurma tərzini müəyyən edən sabitlərin adları və kodları cədvəl 9.5 – də göstərilmişdir.

Cədvəl 9.5. Həndəsi fiqurların doldurulma tərzləri sabitləri

Sabitin adı	Sabitin kodu	Həndəsi təsviri
`EmptyFill`	0	*Fonun rəngi ilə tam doldurma*
`SolidFill`	1	*Verilmiş rənglə tam doldurma*
`LineFill`	2	*Üfqi xətlərlə doldurma*
`LtSlashFill`	3	*Diaqonal xətlərlə doldurma (///)*
`SlashFill`	4	*Qalın diaqonal xətlərlə doldurma (///)*
`BkSlashFill`	5	*Əksinə qalın diaqonal xətlərlə doldurma (\\\)*
`LtBkSlashFill`	6	*Əksinə diaqonal xətlərlə doldurma (\\\)*
`HatchFill`	7	*Tor şəkilli doldurma*
`XhatchFill`	8	*Maili tor şəkilli doldurma*
`İnterleaveFill`	9	*Növbələşən xətlə doldurma*
`WideDotFill`	10	*Seyrək yerləşən nöqtələr*
`CloseDotFill`	11	*Sıx yerləşən nöqtələr*
`UserFill`	12	*İstifadəçinin müəyyən etdiyi üslub*

FloodFillStyle (*x*, *y* **: İnteger;** *reng_xett* **: Word);** – *reng_xett* rəngli xətti ilə əhatələnmiş (*x*,*y*) nöqtəsi ətrafında yerləşən bütün sahəni `SetFillStyle` şablonu ilə doldurulur.

Qeyd. Qrafik rejimdə `Write` və `Writeln` prosedurları korrekt işləməyə bilər. Onların əvəzinə xüsusi mətni ekrana çıxarma prosedurları tətbiq edilməlidir.

9.4.3. Mətni ekrana çıxarma prosedurları

Mətn rejimində ekranda mətni təsvir etmək üçün adətən 8∗8 pikselli matrisdən istifadə edilir. Belə matrisin tutduğu yer *işarə tutumu* adlanır. Yuxarıda qeyd olunduğu kimi, ən çox *80* simvoldan ibarət *25* sətirlik, yəni 80∗25=2000 yolvermə qabiliyyətinə malik ekrandan istifadə olunur. Bununla bərabər, 8∗14, 8∗16, 9∗14, 9∗16 işarə tutumlu matrisə və 40∗25, 80∗43, 80∗50 yolvermə qabiliyyətinə malik ekranlar da mövcuddur.

Qrafik rejimdə ekrana mətn çıxarmaq üçün aşağıdakı prosedurlardan istifadə edilir:

OutTextXY(*x*, *y* **: İnteger;** *Text* **: string**); – (*x*,*y*) nöqtəsindən başlayaraq *Text* mətnini ekrana çıxarır.

SetTextStyle(*font*,*hj*, *size*)**;** – ekranda mətnin yerləşməsini idarə edir. Burada *font* parametri vektor şriftin adını, *hj* parametri mətnin üfqi və ya şaquli istiqamətdə yerləşməsini, *size* parametri isə şriftin ölçüsünü (miqyas əmsalı) müəyyən edir. *font* parametrinin ala biləcəyi mümkün sabitlər cədvəl 9.6 – da göstərilmişdir.

Əgər proqram cədvəldə göstərilən şriftləri tapa bilməzsə, onda səhv baş vermir, sadəcə olaraq susmaya görə təyin olunmuş şrift istifadə olunur. Bu prosedurun ikinci parametri *hj* iki qiymət ala bilər:

- `HorizDir` –mətn *üfqi* (adi) istiqamətdə yerləşir;
- `VertDir` –mətn *şaquli* istiqamətdə yerləşir.

Nəhayət, prosedurun üçüncü parametri – *size* şriftin miqyas əmsalıdır, məsələn, rastr şriftlər üçün *1* miqyas əmsalı 8∗8 pikselli matrisə, *2* əmsalı isə 16∗16 pikselli matrisə uyğun gəlir.

Cədvəl 9.6. Turbo Pascal dilinin vektor şriftləri

Sabitin adı	Şriftin kodu	Şriftin yerləşdiyi fayl
`TriplexFont`	1	*trip.chr*
`SmallFont`	2	*litt.chr*
`SansSerifFont`	3	*sans.chr*
`GothicFont`	4	*goth.chr*

`SetTextJustufy` (*horj*, *verj*)**;** – ekranda mətni üfqi (birinci parametr) və şaquli (ikinci parametr) istiqamətlərdə düzləndirir. Düzləndirmə formaları cədvəl 9.7 – də göstərilmişdir.

Cədvəl 9.7. `SetTextJustufy` prosedurunda düzləndirmə formaları

Üfqi istiqamətdə		
Sabitin adı	**Kodu**	**Düzləndirmə**
`LeftText`	0	*Sola tərəf*
`CenterText`	1	*Mərkəzə görə*
`RightText`	2	*Sağa tərəf*
Şaquli istiqamətdə		
`BottomText`	0	*Aşağıya tərəf*
`CenterText`	1	*Mərkəzə görə*
`TopText`	2	*Yuxarıya tərəf*

`ClearDevice;` – cari göstəricini ilkin vəziyyətə, yəni (*0,0*) koordinatlı nöqtəyə yerləşdirir və ekranı fonun rəngi ilə rəngləyərək onu təmizləyir.

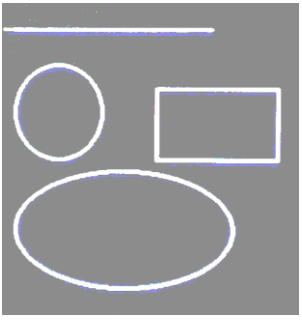
`RestoreCrtMode;` – sistemi qrafik rejimdən əvvəlki mətn rejiminə qaytarır.

9.5. Qrafiklərin qurulmasına aid misallar

Yuxarıda öyrəndiyimiz prosedur və funksiyaları tətbiq etməklə proqramlar tərtib edək.

Misal.

```
program fiqurlar;
uses Crt, Graph,graphs;
begin
 clrscr;
 Open_graph;
 line(10,50,200,50);
 Circle(60,120,40);
 Rectangle(150,100,260,160);
 Ellipse(120,220,0,360,100,50);
 Readln;
 CloseGraph;
end.
```



Şəkil 9.2. Qrafik primitivlər

Burada `line` proseduru ilə (10,50) və (200,50) nöqtələrini birləşdirən düz xətt, `Circle` proseduru ilə mərkəzi (60,120) nöqtəsində olan 40 piksel radiuslu çevrə, `Rectangle` proseduru ilə diaqonallarının koordinatları (150,100) və (260,160) olan düzbucaqlı və `Ellipse` proseduru ilə mərkəzi (120,220) nöqtəsində olan, başlanğıc bucağı 0, radiusu 100 və son bucağı 360 olan (yəni qapalı), radiusu 50 piksel olan ellips çəkilir (şəkil 9.2). Əgər `Ellipse` prosedurunda başlanğıc və son bucaqları, uyğun olaraq, məsələn, 130 və 400 götürsəniz, qapalı olmayan oval alacaqsınız; əgər başlanğıc və son radiusları 100 və 150 götürsəniz, dartılmış (şaquli) ellips alacaqsınız; əgər

başlanğıc və son radiusları eyni, məsələn, 100 götürsəniz, onda çevrə çəkiləcəkdir. Ümumiyyətlə, təqdim olunan bütün qrafik proqramlarda, parametrlərə müxtəlif qiymətlər verərək onların necə dəyişməsini müşahidə etmək lazımdır.

Misal.

```
program fiqur_metn;
uses Crt,Graph,graphs;
begin
 Open_graph;
 line(10,50,200,50);
 Circle(60,120,40);
 Rectangle(150,100,260,160);
 Ellipse(120,220,0,360,100,50);
 OutTextXY(83,220,'TURBO PASCAL');
 Readln;
 CloseGraph;
end.
```

Bu proqramda, yuxarıdakı misalda çəkilmiş həndəsi fiqurlar təkrar olunur, fərq ondadır ki, ellipsin daxilində `TURBO PASCAL` mətni yazılacaqdır.

Misal.

```
Program ucbucaq1;
uses Crt, Graph, Graphs;
begin

 Open_graph;
 line(300,100,450,450);
 line(300,100,150,450);
 line(150,450,450,450);
 Readln;
 CloseGraph;

end.
```

Bu proqramda, `line` proseduru ilə 3 düz xətt çəkilir. Lakin, koordinatlar elə seçilmişdir ki, bu düz xətlər üçbucaq əmələ gətirir. Bu proqramı belə də yaza bilərik:

```
Program ucbucaq2;
uses Crt, Graph;
var Gd, Gm: Integer;
begin
 Gd := Detect;
 InitGraph(Gd, Gm, 'c:\bp\bgi');
 if GraphResult <> grOk then
  begin
   Writeln('Səhv aşkar edildi');
   Halt(1);
  end;
 line(300,100,450,450);
 line(300,100,150,450);
 line(150,450,450,450);
 Readln;
 CloseGraph;
end.
```

Nəticə nöqteyi–nəzərindən bu proqramın yuxarıdakı proqramdan heç bir fərqi yoxdur. Lakin, burada qrafik rejimin qoşulması üçün yuxarıda qeyd etdiyimiz üçüncü üsuldan istifadə edilmişdir, yəni `uses` operatorunda **`graphs`** modulu qoşulmamış, onun əvəzinə proqramın icra hissəsində `InitGraph` proseduru istifadə edilmişdir.

Misal. Üçbucaq və çevrələrin çəkilməsi.

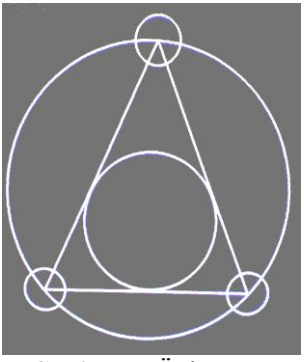
```
program ucbucaq_cevre1;
uses Crt, Graph, Graphs;
begin
 Open_Graph;
 Circle(310,70,30);
 line(310,70,450,400);
 line(310,70,150,400);
 Circle(150,400,30);
 Circle(304,303,95);
```

```
  Circle(300,270,200);
  line(150,400,450,400);
  Circle(450,400,30);
  Readln;
  CloseGraph;
end.
```

Bu proqramla düz xətlər və çevrələr çəkilir. Prosedurların parametrlərinin qiymətləri elə seçilmişdir ki, düz xətlər üçbucaq əmələ gətirir və bu üçbucağın daxilinə, xaricinə və mərkəzləri üçbucağın təpə nöqtələrində yerləşən çevrələr çəkilir (şəkil 9.3.).

Bu proqramı belə də yaza bilərik:

```
program ucbucaq_cevre2;
uses Crt,Graph;
var Gd,Gm: Integer;
begin
  Gd := Detect;
  InitGraph(Gd, Gm, 'c:\bp\bgi');
  if GraphResult <> 0 then
    begin
      Writeln('Qrafik rejimdə səhv!');
      Halt(1);
    end;
  Circle(310,70,30);
  line(310,70,450,400);
  line(310,70,150,400);
  Circle(150,400,30);
  Circle(304,303,95);
  Circle(300,270,200);
  line(150,400,450,400);
  Circle(450,400,30);
  Readln;
  CloseGraph;
end.
```



Şəkil 9.3. Üçbucaq və çevrələrin qurulması

Biz indiyədək həndəsi primitivləri çəkdikdə onların parametrlərinə konkret

qiymətlər verirdik. Bu parametrlər dövr altında müxtəlif qiymətlər alan dəyişənlər də ola bilər. Məhz bu halda, həndəsi fiqurların ölçüləri dəyişdiyindən, daha maraqlı təsvirlər alınır (həndəsi fiqurlar dinamiki dəyişir). Bu andan etibarən tərtib edəcəyimiz bütün proqramlarda belə xarakterli məsələlər əhatə olunacaqdır. Həmin məsələlərdə şərh işarələri (**{** və **}**) daxilində yazılmış operatorlar və prosedurlar görəcəksiniz. Proqramları yerinə yetirdikdə həmin şərh simvollarını pozub (ona analoji sətirləri isə, əksinə, şərh simvolları daxilinə alıb) proqramın nəticələrinin necə dəyişməsini müşahidə etməyi təkidlə tövsiyə edirik, bu halda proqramı daha dərindən qavraya biləcəksiniz.

Sonrakı misalların əksəriyyətində `Delay` prosedurundan istifadə edilmişdir. Onun ümumi forması belədir:

`Delay`(*time*)**;**

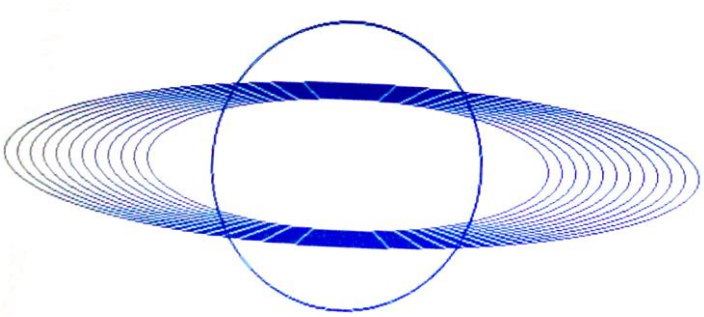
Bu prosedur proqramın icrasını *time* parametri ilə göstərilmiş millisaniyələr qədər ləngidir. *time* parametrinə qiymət verdikdə kompüterinizin sürətini nəzərə alın: əgər kompüteriniz köhnədirsə, bu parametrə çox böyük qiymətlər verməyin.

Misal. Çevrə və onun ətrafına konsentrik ellipslərin çəkilməsi.

```
program peyk;
uses Crt, Graph,graphs;
var i,t:word;
begin
 clrscr;
 Open_graph;
 circle(300,220,100);
 circle(300,220,101);
 t:=0;
 for i:=1 to 12 do
  begin
    t:=i+5;
    {delay(50000);}
```

```
      ellipse(300,220,0,360,90+10*t,40+t);
      {Ellipse(300,220,130,410,90+10*t,40+t);}
    end;
  Readln;
  CloseGraph;
end.
```

Proqramın nəticəsi sanki Saturn planetini və onun peyklərini xatırladır (şəkil 9.4). `for` operatorunda *12* əvəzinə digər ədədlər yazmaqla – ellipslərin sayını, `t`–nin ifadəsində addımı dəyişməklə ellipslər arasındakı məsafəni dəyişdirə bilərsiniz.
`{Ellipse(300,220,130,410,90+10*t,40+t);}` prosedurunu qoşsanız ellipslər qapalı olmayacaq (sanki, planetin arxa hissəsi görünməyəcəkdir). Və nəhayət, şəklin çəkilməsi prosesini müşahidə etmək istəyirsinizsə, `delay` prosedurunu qoşun.



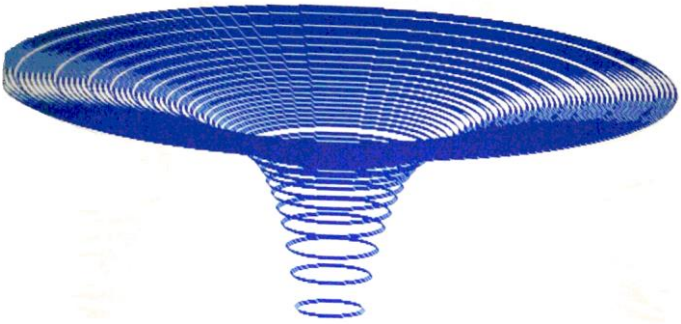
Şəkil 9.4. "Saturn" planeti

Misal. "Qara dəlik" təsviri.

```
program burulqan;
uses Crt,Graph,
     graphs;
var i,t:word;
     x,y:integer;
begin
```

```
    clrscr;
    Open_graph;
    setBkColor(15);
    for i:=60 downto 5 do
     begin
       t:=i+1;
       y:=trunc(400/t);
       x:=5*t;
       SetColor(1);
       {Delay(50000);}
       Ellipse(320,200+3*y,0,360,x,trunc(x/5));
       ellipse(321,201+3*y,0,360,x,trunc(x/5));
       ellipse(322,202+3*y,0,360,x,trunc(x/5));
     end;
    Readln;
    CloseGraph;
  end.
```

Bu proqramın nəticəsi gur çaylarda əmələ gələn burulğanı xatırladır, əslində isə bu təsvir təbiət hadisəsi olan "qara dəlik" təsviridir (şəkil 9.5). Bu proqramda biz `SetBkColor(15);` və `SetColor(1);` prosedurlarını tətbiq etdik. Birinci prosedur ekranı (fonu) `15` rəngi ilə, yəni ağ rənglə rəngləyir, ikinci prosedur isə `1` rəngi ilə, yəni göy rənglə təsviri çəkir (qələmin rəngi). `15` və `1` ədədlərinin əvəzinə uyğun olaraq `white` və `blue` yaza bilərik.

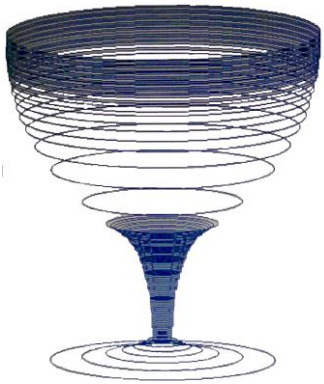


Şəkil 9.5. "Qara dəlik"

Misal. Kola üçün qədəh təsviri.

```
program qedeh;
uses Crt,
    Graph,graphs;
var i,t:word;
    x,y,st:word;
begin
 clrscr;
 Open_graph;
 {setBkColor(11);}
 {SetColor(4); }

   { qədəh hissəsi }
 for i:=200 downto 2 do
  begin
   x:=Trunc(100*cos(pi/i));
   y:=Trunc(100*sin(pi/i));
   {Delay(50000); }
   Ellipse(220,65+trunc(2.35*y),
   0,360,2*x,trunc(x/3));
  end;

 { qədəhin ayaq hissəsi }
 st:=y;
 for i:=10 to 75 do
  begin
   t:=5+i;
   x:= trunc(35/(t/15));
   {Delay(50000); }
   Ellipse(220,65+2*(st+t),0,360,2*x,
                        trunc(x/3));
  end;
 st:=st+i;

 { qədəhin oturacaq hissəsi }
 for i:=2 to 10 do
  begin
   x:=trunc(2.5*exp(i/3));
```

```
      {Delay(50000); }
      Ellipse(220,65+2*(st+2*i),0,360,2*x,
                          trunc(x/3));
    end;
  Readln;
  CloseGraph;
end.
```



Şəkil 9.6. Kola üçün qədəh

Proqramın nəticəsi şəkil 9.6 – da göstərilmişdir. Qədəhin şüşə divarlarının çəkilməsində *sinx* və *cosx* funksiyalarından istifadə edilmişdir. Qədəhin oturacaq hissəsində ellipslərin radiuslarının kiçik qiymətlərdən böyük sürətlə artması üçün eksponensial funksiya tətbiq edilmişdir. Dövrlərin parametrlərinin qiymətlərini, ellipslərin parametrlərini, rəngləri dəyişdirməklə və `Delay` prosedurunu qoşmaqla, Siz, çox maraqlı təsvirlər izləyə biləcəksiniz.

Misal. Müxtəlif tərzlərdə mətnin ekrana çıxarılması.

```
program metn;
uses Crt, Graph,graphs;
var ad:string;
begin
 clrscr;
 gotoxy(5,5);
 writeln('Mətni daxil
  edin:');read(ad);
 Open_graph;
 settextstyle(2,VertDir,15);
 setcolor(2);
 OutTextXY(10,10,ad);
 settextstyle(4,HorizDir,55);
 setcolor(3);
 OutTextXY(100,100,ad);
```

```
  Readln;
  CloseGraph;
end.
```

Proqram icra olunduqda mətnin daxil edilməsini tələb edir. Siz, mətn daxil etdikdən sonra, həmin mətn ekranda iki formada təsvir olunacaqdır. Birinci formada ekranın (10,10) nöqtəsində `SmallFont` şrifti ilə ölçüsü 15 punkt olan yaşıl rəngli, şaquli istiqamətdə mətn təsvir olunacaqdır.

Bunu `settextstyle(2,VertDir,15);` proseduru həyata keçirir (burada `2` ədədi `SmallFont` şriftinin kodudur). İkinci formada ekranın (100,100) nöqtəsində `GothicFont` şrifti ilə ölçüsü `55` punkt olan mavi rəngli, üfqi istiqamətdə mətn təsvir olunacaqdır. Bunu `settextstyle(4,HorizDir, 55);` proseduru həyata keçirir (burada `4` ədədi `GothicFont` şriftinin kodudur). Mətnin ekrana çıxarılmasını `OutTextXY` proseduru yerinə yetirir. Mətnin rəngini isə `setcolor` proseduru müəyyənləşdirir.

Misal. Ekran qoruma proqramı.

```
program ekran_qoruma1;
uses crt,graph,graphs;
var x,y:integer;
    a:string;
begin
 clrscr;
 gotoxy(5,5);
 write('Mətni
       daxil edin:');
 read(a);
 Open_graph;
 Randomize;
  repeat
   x:=random(800);
   y:=random(600);
```

```
      settextstyle(random(6),
      random(2),random(10));
      if (x+textheight(a)<500) and
      (y+textwidth(a)<400) then
       begin
         setcolor(random(15)+1);
         outtextxy(x,y,a);
         delay(60000);
         cleardevice;
       end;
    until keypressed;
  readln;
  closegraph;
end.
```

Bu proqram ekranı qoruma funksiyasını həyata keçirir. Daxil edilmiş mətn ekranın ixtiyari təsadüfi hissəsində təsadüfi ölçülü, təsadüfi rəng və təsadüfi istiqamətlərdə təsvir olunur. Proqramda **TextHeight** və **TextWidth** prosedurlarından istifadə edilmişdir ki, bu prosedurlar uyğun olaraq, piksellərlə, sətrin *hündürlüyü* və *enini* müəyyən edir. Bu proqramda **Crt** modulundan `Keypressed` proseduru da istifadə edilmişdir. İstənilən klaviş basıldıqda bu prosedur `True` qiyməti qaytarır. Ona görə də ixtiyari klavişi basdıqda dövrdən çıxış baş verəcək və proqram öz işini dayandıracaqdır. `Random` funksiyası ilə təsadüfi ədədlər yaradılır. Bu ədədlər ekran koordinatlarını müəyyənləşdirir. `if` operatoru ilə mətnin ekranın hüdudlarından kənara çıxmamasına nəzarət edilir. `Settextstyle` prosedurundakı `Random` funksiyası isə şriftin adına, mətnin istiqamətinə və şriftin ölçüsünə uyğun təsadüfi ədədlər yaradır. Son şərtli dövr operatoru təkrar olunduqca, `cleardevice;` proseduru hər dəfə ekranı təmizləyir. Əgər bu proseduru pozsanız, müəyyən müddətdən sonra ekran mətnlərlə tam dolacaqdır. İxtiyari klavişi basdıqda proqramdan çıxış baş verir.

Bu proqramı belə də yaza bilərik:

```
program ekran_qoruma2;
uses crt,graph;
const
    drive:integer=EGAHI;
    mode:integer=EGA;
    path='c:\bp\bgi';
var i,x,y:integer;
    a:string;
begin
 clrscr;
 gotoxy(5,5);
 writeln('Mətni daxil edin:');
 readln(a);
 İnitGraph(drive,mode,path);
 Randomize;
  repeat
  x:=random(800);
  y:=random(600);
  settextstyle(random(6),
  random(2),random(10));
  if (x+textheight(a)<500) and
  (y+textwidth(a)<400) then
   begin
     setcolor(random(15)+1);
     outtextxy(x,y,a);
     delay(50000);
     cleardevice;
   end;
  until keypressed;
 readln;
 closegraph;
end.
```

Misal. ”Hörümçək toru” təsviri.

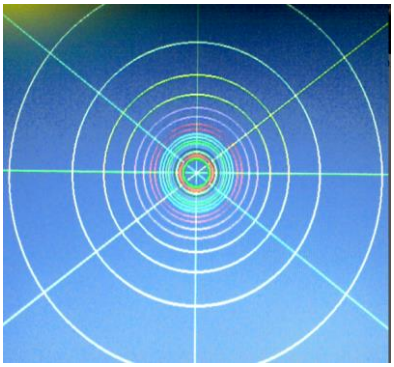
```
program Horumchek_Toru;
uses crt,graph,graphs;
var
   i:word;
```

```
begin
 open_graph;
 SetBkColor(blue);
 SetColor(LightCyan);
 Line(0,0,GetMaxX,GetMaxY);
 Delay(1000);
 SetColor(Yellow);
 Line(0,GetMaxY,GetMaxX,0);
 Delay(1000);
 SetColor(Lightgreen);
 Line(0,GetMaxY div 2,
 GetMaxX,GetMaxY div 2);
 Delay(1000);
 SetColor(Lightgray);
 Line(GetMaxX div 2,0,
 GetMaxX div 2, GetMaxY);
 Delay(1000);
 SetColor(Lightred);
 for i:=2 to 20 do
  begin
    SetColor(16-i div 2);
    Circle(GetMaxX div 2,
    GetMaxY div 2, GetMaxY div i);
    Delay(5000-15*i);
  end;
 Readln;
 closegraph;
end.
```

Bu proqram koordinat sistemini qurur və koordinat başlanğıcından keçən düz xətlər və mərkəzi koordinat başlanğıcında olan konsentrik çevrələr çəkir ki, bu da hörümçək torunu xatırladır. Proqramda `GetMaxX` və `GetMaxY` funksiyalarından istifadə edilmişdir. Bu funksiyalar, proqramı digər monitorlu kompüterdə işlətdikdə, qrafikin çəkilməsində həndəsi ölçülərlə əlaqədar baş verə biləcək problemin əmələ gəlməsinin qarşısını alır. Proqramın nəticəsi səkil 9.7 – də göstərilmişdir. `for` operatorunun

parametrlərini dəyişməklə, Siz, müxtəlif təsvirlər müşahidə edəcəksiniz.

Misal. Serpinski "xalısının" qurulması.



Şəkil 9.7. "Hörümçək toru" təsviri

Serpinski "xalısının" qurulması üçün əvvəlcə tərəfi vahidə bərabər olan kvadrat götürülür, sonra kvadratın hər tərəfi üç bərabər hissəyə, kvadratların özləri isə tərəfi 1/3 olan 9 bərabər hissəyə bölünür. Alınan həndəsi təsvirdən mərkəzi kvadrat çıxarılır. Daha sonra yaranmış 8 kvadratın hər biri üzərində belə bölgü əməliyyatı ardıcıl təkrar olunur. Belə təsvirlərə *fraktallar* deyilir. Fraktallar kompüter qrafikasında geniş tətbiq olunur. Şəkilçəkmə redaktorları vasitəsilə hər hansı bir təsviri çəkdikdə təsvir yaddaşda böyük yer tutur. Həmin təsviri fraktal kimi yaratdıqda isə əməliyyat iterasiyalı alqoritm üzrə yerinə yetirildiyi üçün, çox az yaddaş tələb olunur. Fraktalların qurulma prinsipi ondan ibarətdir ki, obyekt həndəsi ölçülərini kiçildir və yeni koordinatlarla çəkilir.

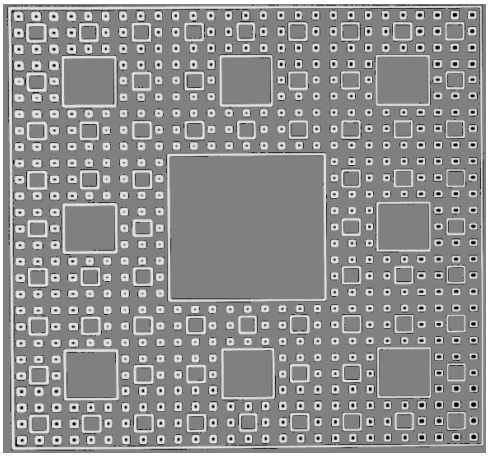
```
Program Serpin_kv;
Uses CRT,Graph;
Var
   gd, gm:Integer;
   x1,y1,x2,y2,x3,y3:real;


procedure serp(x1,y1,x2,y2:real;
                n:integer);
var
   x1n,y1n,x2n,y2n:real;
begin
```

```
    if  n > 0  then
    begin
      x1n:=2*x1/3+x2/3;
      x2n:=x1/3+2*x2/3;
      y1n:=2*y1/3+y2/3;
      y2n:=y1/3+2*y2/3;
      rectangle(round(x1n),
      round(y1n),round(x2n),
      round(y2n));
      delay(3000);
      serp(x1,y1,x1n,y1n,n-1);
      delay(5000);
      serp(x1n,y1,x2n,y1n,n-1);
      delay(3000);
      serp(x2n,y1,x2,y1n,n-1);
      delay(3000);
      serp(x1,y1n,x1n,y2n,n-1);
      delay(3000);
      serp(x2n,y1n,x2,y2n,n-1);
      delay(3000);
      serp(x1,y2n,x1n,y2,n-1);
      delay(3000);
      serp(x1n,y2n,x2n,y2,n-1);
      delay(3000);
      serp(x2n,y2n,x2,y2,n-1);
    end;
  end;


  Begin
    gd:=detect;
    InitGraph(gd,gm,'c:\bp\bgi');
    rectangle(50,50,400,400);
    Serp(50, 50,400,400,4);
    ReadLn;
    CloseGraph;
  End.
```

Burada, `Serp` adlı *rekursiv* alt proqram tərtib edilmiş və əsas proqramdan ona müraciət olunduqda, faktik arqumentlər

kimi, kvadratların koordinatları və dövrlər sayı (4) ötürülmüşdür. Kvadratlar `rectangle` proseduru ilə çəkilir. Dövrlər sayını artırdıqda kvadratların ölçüləri kiçiləcək, sayları isə artacaqdır.

Proqramın nəticəsi şəkil 9.8 – də göstərilmişdir.



Şəkil 9.8. Serpinski "xalısı"

Misal. Üçbucaqlardan ibarət Serpinski "xalısının" qurulması.

```
Program Serp_ucbucaq;
Uses CRT,Graph;
Var
   gd,gm:Integer;
Const
iter=5;

Procedure tr(x1,y1,x2,y2,x3,y3:Real);
Begin
Line(round(x1),round(y1),round(x2),round(y2));
Line(round(x2),round(y2),round(x3),round(y3));
Line(round(x3),round(y3),round(x1),round(y1));
End;

Procedure draw(x1,y1,x2,y2,x3,y3:Real;
```

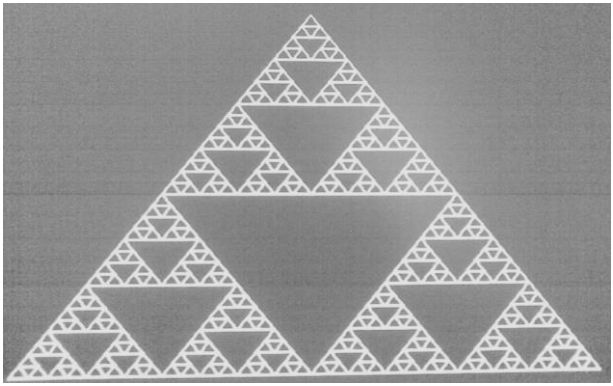
```
                    n:Integer);
Var
   x1n,y1n,x2n,y2n,x3n,y3n:Real;
Begin
  If  n > 0  then
    Begin
      x1n:=(x1+x2)/2;
      y1n:=(y1+y2)/2;
      x2n:=(x2+x3)/2;
      y2n:=(y2+y3)/2;
      x3n:=(x3+x1)/2;
      y3n:=(y3+y1)/2;
      delay(30000);
      tr(x1n,y1n,x2n,y2n,x3n,y3n);
      delay(3000);
      draw(x1,y1,x1n,y1n,x3n,y3n,n-1);
      delay(3000);
      draw(x2,y2,x1n,y1n,x2n,y2n,n-1);
      delay(3000);
      draw(x3,y3,x2n,y2n,x3n,y3n,n-1);
    End;
End;

Begin
  gd =Detect;
  InitGraph(gd,gm,'');
  tr(320,80,600,400,40,400);
  draw(320,80,600,400,40,400,iter);
  Readln;
  CloseGraph;
End.
```

Bu proqram əvvəlki proqrama tamamilə analojidir, yalnız kvadratlar əvəzinə üçbucaqların qurulması ilə ondan fərqlənir. Proqramın nəticəsi şəkil 9.9 – da göstərilmişdir.

Misal. Üçbucaqlardan ibarət Serpinski "xalısının" nöqtələrlə qurulması.

```
Program Serp_ucbucaq2R;
Uses CRT,Graph;
```

```
Var
   gd,gm:Integer;
   l,x,y:Real;
```



Şəkil 9.9. Üçbucaqlardan ibarət Serpinski "xalısı"

```
Begin
  gd:=Detect;
  InitGraph(gd,gm,'c:\bp\bgi');
  x:=0; y:=0;
  Randomize;
  While not Keypressed Do
    Begin
      l:=2/3*pi*random(3);
      x:=x/2+cos(l);
      y:=y/2+sin(l);
      PutPixel(320+Round(x*110),255+
                  Round(y*110),15);
    End;
  Readln;
  CloseGraph;
End.
```

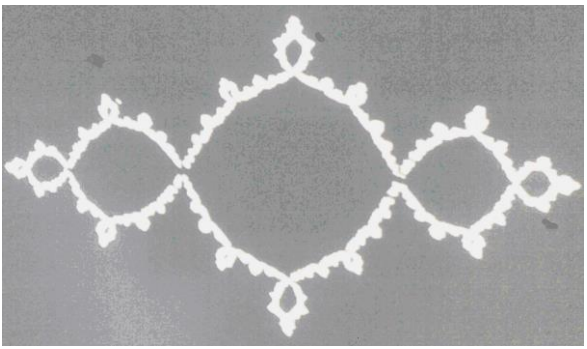
Bu proqramda ilkin şərtli dövr təşkil olunur, dövrün yerinə yetirilmə şərti "*heç bir klaviş basılmadıqda*" (`While not Keypressed Do`) şərtidir. Dövr daxilində `x` və `y` koordinatları sinus və kosinus funksiyaları vasitəsilə

hesablanır və onlar yuvarlaqlaşdırılaraq (`Round` funksiyası) `PutPixel` prosedurunun arqumentləri kimi istifadə edilir. Bu prosedur ağ rənglə (`15` kodu) nöqtələr çəkir.

Misal. Naxışlı fraktallar.

```
program serp_ornament;
uses crt,graph,graphs;
var d,rt:integer;
    i:longint;
    wx,wy,cx,cy,x,y,m,n,r,theta:real;
begin
 Open_graph;
 x:=GetMaxX; { ekranın eni }
 y:=GetMaxY; { ekranın hündürlüyü }
 for i:=1 to 100000 do { iterasiyaların sayı }
  begin
    cx:=-1;cy:=0; { görünüş sabitləri }
    wx:=x-cx;
    wy:=y-cy;
    if wx>0 then theta:=arctan(wy/wx);
    if wx<0 then theta:=3.14159+arctan(wy/wx);
    if wx=0 then theta:=1.57079; {pi/2}
    theta:=theta/2;
    r:=sqrt(wx*wx+wy*wy);
    if Random<0.5
     then
       r:=sqrt(r)
     else
       r:=-sqrt(r);
    x:=r*cos(theta);
    y:=r*sin(theta);
    m:=-5+(x)*100+GetMaxX/2;
    n:=(y)*100+GetMaxY/2;
    PutPixel(trunc(m),trunc(n),White);
  end;
 readln;
 Closegraph;
```

```
end.
```



Şəkil 9.10. Naxışlı fraktallar

Proqramın nəticəsi şəkil 9.10 – da göstərilmişdir. Bu proqramın da prinsipi Serpinski "xalısının" qurulması prinsipi ilə eynidir. Burada, kvadrat və ya üçbucaq əvəzinə qeyri–müəyyən formalı fiqur götürülmüşdür ki, o da ölçülərini kiçildib koordinatlarını dəyişdikdə maraqlı naxış əmələ gətirir. Burada da `x` və `y` koordinatları sinus və kosinus funksiyaları vasitəsilə hesablanır. Naxış ağ rəngli (`White`) nöqtələr vasitəsilə (`PutPixel` proseduru) qurulur. Nöqtələrin koordinatlarının tam ədədlər olması üçün `trunc` funksiyasından istifadə edilmişdir. Əgər naxışın qurulması prosesini izləmək istəsəniz, onda `PutPixel` prosedurundan əvvəlki sətrə `delay` prosedurunu əlavə edin, lakin onun parametrinə çox böyük qiymətlər verməyin. Çünki, dövr 100000 dəfə təkrar olunur.

ƏDƏBİYYAT

1. Məhərrəmov Z.T. Pascal–dan Delphi–yə. Ali məktəb tələbələri üçün dərs vəsaiti. – Bakı: "Təhsil" NPM, 2010. – 412 s.
2. Vəliyev Ə.A., Məhərrəmov Z.T., Əbilov Y.Ə. Delphi: nəzəriyyə və təcrübə. Ali məktəb tələbələri üçün dərs vəsaiti. – Bakı: Çaşıoğlu, 2004. – 336 s.
3. Məhərrəmov Z.T. Alqoritm və onun təsvir üsulları. Bakı, 2006.– 30 s.
4. Kərimov S.Q., Həbibullayev S.B., İbrahimzadə T.İ. İnformatika. Ali məktəb tələbələri üçün dərslik. – Bakı: 2010. – 434 s.
5. Vəliyev N.N. Turbo Pascal Windows üçün. Bakı: 2003. – 142 s.
6. Немнюгин С.А. Turbo Pascal. – СПб.: Питер, 2002. – 496 с.
7. Федоренко Ю. Алгоритмы и программы на Turbo Pascal. Учебный курс. – СПб.: Питер, 2001. – 240 с.