Algoritmi

Algoritem je postopek za rešitev nekega problema.
Ko pišemo algoritem, po katerem bomo izdelali program za računalnik, mora imeti naslednje lastnosti:

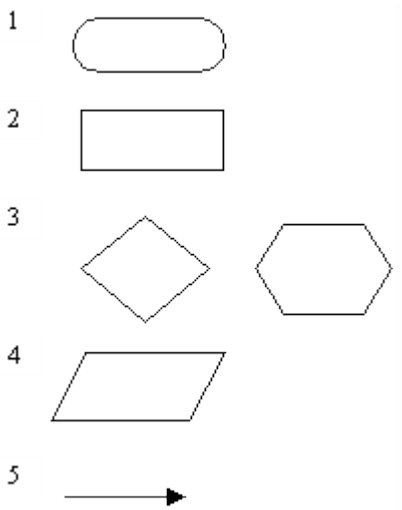
- sestavljen iz zaporedja korakov
- ustavljiv (pri različnih kombinacijah vhodnih podatkov)
- nedvoumen
- splošen (rešuje čim več podobnih problemov).

Algoritme lahko predstavimo v naslednjih oblikah:

- v psevdo naravnem jeziku (mešanica naravnega in formalnega jezika, npr. enačb)
- z diagramom poteka
- s programom

Diagram poteka je sestavljen iz blokov in povezav med njimi. Največ uporabljani simboli so:

1. začetni ali končni blok
2. prireditveni blok
3. odločitveni blok
4. vhodni ali izhodni blok
5. povezava



Izpisi
V izpisu lahko določimo,koliko mest naj izpis zasede.
Primer izpisa celega števila na 5 mest:

```
Writeln(x:5);
```

Če je število daljše od 5 mest, se izpiše na nujno potrebno število mest.
Pri izpisu realnega števila lahko določimo število vseh mest (eno mesto zasede decimalna pika) in število decimalnih mest.
Primer izpisa realnega števila na 5 mest, dve mesti od petih bosta za decimalno piko.

```
Writeln(x:5:2);
```

Če bo število daljše od 5 mest, bo izpisano na nujno potrebnem številu mest.

Spremenljivke
Spremenljivko si lahko predstavljamo kot škatlo, v katero spravljamo različne vrednosti. Različni podatki zahtevajo različne škatle. Vsako spremenljivko moramo deklarirati (določiti ime in rezervirati prostor v pomnilniku). Od tipa spremenljivke je odvisno, koliko prostora (kako veliko škatlo) bomo potrebovali.

Tip	**Velikost v zlogih**	**Interval**	**Natančnost**
Shortint	1	-128..127	cela števila
Integer	2	-32768..32767	cela števila
Longint	4	-2147483648..2147483647	cela števila
Byte	1	0..255	cela števila
Word	2	0..65535	cela števila
Boolean	1	True or False	celi števili 0,1
Real	6	2.9e-39..1.7e38 (cca 1E-38 to 1E+38)	11-12
Single	4	1.5e-45..3.4e38	7-8
Double	8	5.0e-324..1.7e308	15-16
Char	1	1 znak	znaki
String	256 ali manj	255 znakov	besedila

Konstante
Konstanta je sinonim za neko vrednost. Je nespremenljiva.

Prireditveni stavek
Računalnik izračuna vrednost izraza na desni in jo vpiše v spremenljivko na levi.
```
a:=b+c;
```

*Stavek **IF***

Stavek **IF** pripada odločitvenim konstruktom programskega jezika Pascal.

Zgradba stavka:

```
 IF pogoj THEN stavek;
```
pogoj je izraz ali primerjava, ki jo računalnik ovrednoti s TRUE ali FALSE. Če je vrednost pogoja TRUE, se stavek izvrši.
Primer:
```
   3 < 4        TRUE
   I < 6        TRUE, če je v I vrednost < 6, sicer FALSE.
   NOT(7 < 9)   FALSE
```
Pogoj je lahko sestavljen:
```
IF (odgovor = 'D') OR (odgovor= 'd') THEN stavek;
```
Pogoj je lahko spremenljivka tipa boolean:
```
bu := (odgovor = 'D') OR (odgovor = 'd');
IF bu = TRUE THEN stavek;
```
Če želimo, da se ob izpolnjenem pogoju izvrši več stavkov, dodamo besedi BEGIN in END.
```
IF I > J
  THEN
    BEGIN
      Zacasna := I;
      I := J;
      J := Zacasna;
    END;
```
Brez BEGIN in END se izvrši le prvi stavek:
```
IF I > J
  THEN
    Zacasna := I;
    I := J;
    J := Zacasna;
```
je enako:
```
IF I > J THEN Zacasna := I;
I := J;
J := Zacasna;
```

Stavek **IF** ima lahko tudi obliko:
```
IF pogoj THEN Stavek1 ELSE Stavek2;
Če je pogoj TRUE, se izvrši Stavek1, če je FALSE pa Stavek2
```

Stavek **WHILE**

Stavek **WHILE** pripada skupini pascalskih stavkov, ki omogočajo izvajanje ponavljanj, podobno kot stavka **for** in **repeat-until**.

Zgradba stavka:

```
WHILE pogoj DO stavek;
```

Če naj se znotraj zanke izvede več stavkov, dobi stavek WHILE obliko:

```
WHILE pogoj DO
  BEGIN
    Stavek1;
    Stavek2;
     .
     .
     .
    Stavekn;
  END;
```

Stavek WHILE se *izvaja*, dokler je pogoj izpolnjen. Če pogoj prvič ni izpolnjen, se ne izvrši niti enkrat.

Pogoj je lahko sestavljen:

```
WHILE (Vrednost1 < 5) AND (Vrednost2 > 0) DO Stavek1;
```

Stavek **REPEAT-UNTIL**

Stavek ***REPEAT-UNTIL*** pripada skupini pascalskih stavkov, ki omogočajo izvajanje ponavljanj, podobno kot stavka **while** in **for**.

Zgradba stavka:

```
REPEAT

  stavek1;

  stavek2;
    .

    .

    .

  stavekn

UNTIL Pogoj;
```

Zanka se *preneha* izvajati, ko postane **Pogoj** TRUE. Če ima **Pogoj** že pred začetkom izvajanja zanke vrednost TRUE, se zanka vseeno enkrat izvrši.

*Stavek **FOR***

Stavek **FOR** pripada skupini pascalskih stavkov, ki omogočajo izvajanje ponavljanj, podobno kot stavka **while** in **repeat-until**. Ponavadi ga uporabimo v primeru, ko nam je število ponovitev vnaprej poznano.

Zgradba stavka:

```
For KS:=zacetna_vred To koncna_vred do Stavek1;
```

KS predstavlja kontrolno spremenljivko zanke, ki med izvajanjem zavzame vse vrednosti med **zac_vred** in **kon_vred**. Pri vsaki ponovitvi se vrednost kontrolne spremenljivke samodejno poveča za 1. **KS** je zato lahko le številnega tipa (*izmed osnovnih pascalskih podatkovnih tipov torej **Integer, Char ali Boolean***).
Če poenostavimo: stavek **Stavek1** v zanki se izvrši za vsako vrednost kontrolne spremenljivke med **zac_vred** in **kon_vred**.
Če naj se znotraj zanke izvede več stavkov, dobi stavek **FOR** obliko:

```
For KS:=zac_vred To kon_vred do
    begin
      Stavek1;
      Stavek2;
    end.
```

Če je končna vrednost manjša od začetne, se zanka ne izvrši. Če sta enaki, se izvrši enkrat.

Stavek **FOR** lahko privzame tudi naslednjo obliko:

```
For KS:=koncna_vred DownTo zacetna_vred do Stavek1;
```

Razlika je le tem, da se med izvajanjem te oblike zanke vrednost kontrolne spremenljivke pri vsaki ponovitvi avtomatično zmanjša za 1.

Primerjava med zankami

Ponavljanje nam omogočajo trije stavki:

- stavek FOR uporabimo, če je število ponovitev zanke znano
- stavek WHILE uporabimo, če naj se zanka izvaja, dokler je izpolnjen določen pogoj. Pogoj preverjamo na začetku zanke, zato je možno, da se zanka nikoli ne izvrši.
- stavek REPEAT UNTIL uporabimo, če naj se zanka izvaja, dokler ne bo izpolnjen določen pogoj. Pogoj preverjamo na koncu zanke, zato se zanka izvrši vsaj enkrat.

*Stavek **CASE***

Stavek **CASE** pripada odločitvenim konstruktom programskega jezika Pascal. Njegova prednost pred stavkom **IF** je v tem, da omogoča razvejitev programa v več kot le dve veji.

Zgradba stavka:

```
Case vrednost Of
  konst_vred_1 : Stavek1;
  konst_vred_2 : Stavek2;
           .
           .
End;
```

vrednost predstavlja kontrolno spremenljivko stavka. Lahko je le številnega tipa (izmed osnovnih pascalskih podatkovnih tipov torej Integer, Char ali Boolean). Kot **vrednost** lahko nastopa spremenjivka, rezultat logičnega ali rezultat aritmetičnega izraza. **konst_vred_x** predstavlja konstantno vrednost, seznam konstantnih vrednosti, ločenih z vejico ali interval številnih vrednosti:

```
 Case A Of
    1     : Stavek1;
    2,3,5 : Stavek2;
    7..9  : Stavek3
 End;
```

Delovanje danega primera si razložimo na naslednji način :
V primeru, da je **A** enako 1, izvedi stavek ***Stavek1***. Če je **A** enako 2 ali 3 ali 5, izvedi stavek ***Stavek2***, in v primeru, da je **A** element intervala [7,9], izvedi programski stavek ***Stavek3***.

Pri izvajanju stavka CASE iz predhodnega primera se pri vrednosti **A**=33 ne izvede noben stavek. Če bi želeli izvesti stavek ***StavekN*** v primeru, ko vrednost **A** ne doseže nobenega izmed naštetih kriterijev, bi morali uporabiti razširjeno obliko stavka :

```
Case vrednost Of
    konst_vred_1 : Stavek1;
    konst_vred_2 : Stavek2;
           .
           .
  else
    StavekN
End;
```

1. Knjižnica Crt in zanke v pascalu

V knjižnici Crt so vgrajeni podprogrami, spremenljivke in konstante za delo z zaslonom, tipkovnico in zvokom. Uporabo knjižnice napovemo na začetku programa:

```
Uses Crt;
```

Opis pogosto uporabljanih ukazov

`ClrScr;`	Izbriše vsebino aktivnega okna in postavi kazalec v levi zgornji vogal okna.
`GotoXY(X,Y : Byte)`	Postavi kazalec na X- , Y- lego trenutnega okna (levi zgornji vogal okna predstavlja lego (1,1)).
`WhereX : Byte;`	Vrne X-koordinato trenutnega položaja kazalca v oknu.
`WhereY : Byte;`	Vrne Y-koordinato trenutnega položaja kazalca v oknu.
`Delay(MS : Word);`	Prekine izvajanje programa za MS-milisekund.
`Sound(Hz : Word);`	Sproži zvok s frekvenco Hz.
`NoSound;`	Ustavi predvajanje zvoka.
`KeyPressed: Boolean;`	Preverja ali je bila tipka na tikovnici pritisnjena; vsakokrat, ko je tipka pritisnjena, vrne vrednost True, sicer vrne vrednost False.
`ReadKey: Char;`	Prebere znak s tipkovnice.
`TextColor(Color : Byte);`	Nastavi barvo znakov.
`TextBackground(Color:Byte);`	Nastavi barvo ozadja.

2. Procedure in funkcije

Da bi si olajšali delo pri pisanju obširnih programov, delimo problem na manjše podprobleme, dokler ne pridemo do osnovnih podproblemov, ki jih je sorazmerno enostavno rešiti s programom.Za osnovne podprobleme napišemo ustrezne podprograme. Glavni program sestavimo iz klicev ustrezih podprogramov. V pascalu se podprogrami delijo na **procedure** in **funkcije**.Funkcije uporabljamo takrat, ko nam podprogram vrne katerokoli vrednost enostavnega tipa (npr. celo število, realno število, znak, ...). Če nameravamo s podprogramom izvesti zaporedje korakov ali vrniti sestavljeno vrednost, napišemo proceduro.

Oblika deklaracije (podanosti) funkcije

```
Function Ime_funkcije(parametri) : Tip_funcije;
```

Oblika deklaracije procedure

```
Procedure Ime_procedure(parametri);
```

Parametre posredujemo podprogramom:

- **po vrednosti** (pred parametrom ni besedice *var*), morebitne spremembe vrednosti parametra ostanejo znotraj podprograma,
- **po naslovu** (pred parametrom je besedica *var*), vse spremembe vrednosti parametra se poznajo tudi v programu (podprogramu), ki je poklical ta podprogram.

Procedure in funkcije imajo lahko tudi lokalne spremenljivke. Znotraj procedur in funkcij lahko napišemo tudi vgnezdene procedure in funkcije.

3. Nizi v pascalu (stringi)

Nizi vsebujejo poljubno kombinacijo alfanumeričnih in numeričnih znakov. Za deklariranje nizov uporabljamo tip **String**. Obstajata 2 načina deklaracije spremenljivk za nize:

```
Var S1 : String[n];
```

n ima vrednost iz intervala [1..255];
v S1 lahko vpišemo največ n znakov

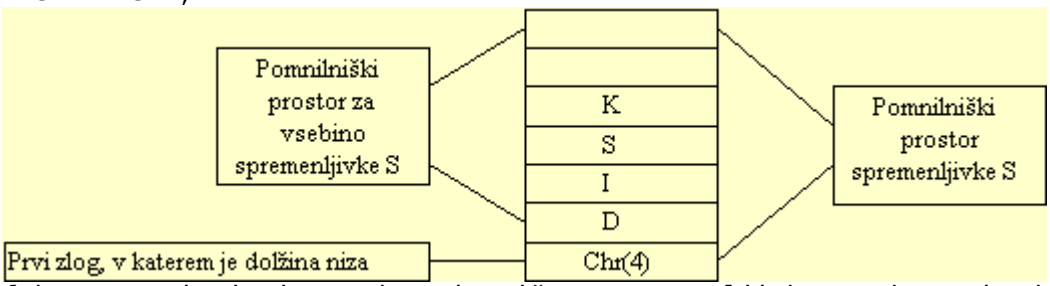
ali

```
Var S2 : String;
```

v S2 lahko vpišemo največ 255 znakov

Ne glede na način deklaracije, ima vsaka spremenljivka na začetku še en zlog, v katerem je zapisano trenutno število znakov v nizu (dolžina niza). Primer:

```
Var S : String[6];
Begin
  S:='DISK';
```




```
{Ukaz za prirejanje vrednosti vpiše v spremenljivko vrednost in dolžino
niza.}
Write(S);  {Izpiše 'DISK'.}
{ Glede na trenutno dolžino, izpiše vsebino niza.}
End.
```

Do posameznih znakov v nizu lahko pridemo tudi s pomočjo pozicije znaka v nizu.
Primeri:

```
Write(S[1]); {Izpiše prvo črko niza, to je D.}
Write(Ord(S[0])); {Izpiše dolžino niza, to je 4.}
S[1]:='P'; {Prvo črko niza nastavimo na P, dolžina ostane
nespremenjena.}
Write(S); {Ker je dolžina niza nespremenjena, izpiše PISK.}
S[0]:=Chr(2); {Dolžino niza nastavimo na 2.}
Write(S); { Ker je dolžina niza 2,izpiše PI.}
S:='P'; { Nizu priredimo vrednost P, dolžina se nastavi na 1.}
Write(S); { Ker je dolžina niza 1, izpiše P.}
```

Če pri delu z nizi uporabljmo ime spremenljivke, se ukaz nanaša na celotni niz. Če je potrebno, ukaz samodejno popravi 0-ti zlog niza. (Npr.: Ukaz **ReadLn**(S) prebere vsebino in nastavi dolžino niza.). Če pri delu z nizi uporabljamo lege posameznih znakov, se dolžina niza ne popravi samodejno. Kadar želimo vpisati v niz več znakov, kot je dožina spremenljivke, pascal odvečne znake odreže.

Primer:

```
S:='DISKETA'; {Po izvedbi tega ukaza bo vsebina spremenljivke S le
'DISKET', dolžina niza pa bo 6.}
```

Opis vgrajenih podprogramov za delo z nizi

Length(S : String) : Integer;	Vrne trenutno dolžino niza; enak učinek dosežemo, če uporabljamo Ord (S[0]);.
Copy(S:String;Index:Byte;Count:Integer):String;	Iz niza S vrne podniz, ki se začne na poziciji Index in ima Count znakov.
Pos(SubStr : String; S : String) : Byte;	Poišče, kje se v nizu S prvič pojavi podniz SubStr. Če S ne vsebuje podniza Substr, vrne funkcija 0, sicer pa vrne mesto prvega zapisa SubStr v nizu S.
Delete(Var S:String; Index:Byte; Count:Integer);	Iz niza S izbriše na poziciji Index Count znakov in samodejno popravi dolžino niza S.
Insert(Source:String;Var S:String;Index:Integer);	V niz S vrine na pozicijo Index podniz Source in samodejno popravi dolžino niza S.
Concat(S1, S2, ... Sn : String) : String;	Združi zaporedje nizov v en niz. Za združevanje nizov lahko uporabljamo tudi operator +.

4. Enodimenzijske in večdimenzijske tabele

Spremenljivke, ki so deklarirane kot tabele, so sestavljene iz elementov. Do posameznega elementa tabele pridemo s pomočjo lege (indeksa) elementa. Zgornja meja za velikost tabele je 64 K.

Enodimenzijsko tabelo deklariramo kot

```
Var T1 : Array [Index1 .. Indexn] Of Tip_elementa;
```

Index1 predstavlja lego prvega elementa tabele, Indexn pa lego njenega zadnjega elementa. Za indekse lahko uporabljamo cela števila in znake. Tip_elementa predstavlja podatkovni tip elementa tabele. Tipi elementov tabele so poljubni.

Dvodimenzijsko tabelo deklariramo kot

```
Var T2 : Array [Index11 .. Index1n, Index21 .. Index2n] Of
Tip_elementa;
```

Index11 predstavlja spodnjo mejo za prvo dimenzijo tabele, Index1n pa zgornjo mejo, Index21 predstavlja spodnjo mejo za drugo dimenzijo tabele in Index2n zgornjo mejo.

Do elementov tabele pridemo s pomočjo lege (indeksa) elementa tabele. Pri enodimenzijskih tabelah določa lego elementa en indeks, pri dvodimenzijskih tabelah določata lego elementa dva indeksa, itn. Pri n dimenzijskih tabelah določa lego n indeksov. Pri delu s tabelami uporabljamo največkrat zanko **For**. Za n dimenzijske tabele uporabljamo n vgnezdenih zank **For.** Pri tem uporabimo indeks zanke tudi kot indeks elementa tabele. Primer:

```
{Del programa za izračun vsote lihih vrednosti tabele stotih celih
števil.}
Type Tabela100 = Array [1..100] Of Integer;
{Deklaracija podatkovnega tipa za tabelo 100 celih števil.}
Var T : Tabela100; {Deklaracija potrebnih spremenljivk.}
    Index : Byte;
    Vsota : Integer;
Begin
.......
Vsota:=0; {Inicializacija vsote.}
For Index:=1 to 100 Do {Index uporabimo kot indeks zanke.}
  If Odd(T[Index]) Then {Index uporabimo kot indeks elementa tabele.}
    Inc(Vsota, T[Index]);
....
End.
```

5. Naštevni in intervalni tipi podatkov

Pri deklaraciji podatkovnega tipa lahko naštejemo tudi nabor vrednosti za ta podatkovni tip. Temu pravimo **naštevni podatkovni tip**. Zgornja meja je 255 različnih vrednosti.

Deklaracija naštevnega tipa

```
Type Nastevni_tip = (Vrednost1, Vrednost2, ..., Vrednostn);
```

Primer

```
Type Barva = (Bela, Rumena, Rdeca, Zelena, Modra);
```

Če se, pri deklaraciji podatkovnega tipa sklicujemo na del (interval) vrednosti nekega, že znanega podatkovnega tipa, potem govorimo o **intervalnem podatkovnem tipu**.

Deklaracija intervalnega tipa

```
Type Intervalni_tip = Spodnja_vrednost .. Zgornja_vrednost;
```

Primeri:

```
Type Ocena = 1..5; {Sklicujemo se na vgrajeni tip Integer.}
     Moja_barva = Bela .. Rdeca;
     {Sklicujemo se na prej definirani tip Barva.}
```

Vgrajeni podprogrami za delo z naštevnim in intervalnim podatkovnim tipom

Inc(Var X:Nastevni_tip);	Spremenljivki X poveča vrednost za 1 (priredi ji vrednost njenega naslednika).
Dec(Var X:Nastevni_tip);	Spremenljivki X zmanjša vrednost za 1 (priredi ji vrednost njenega predhodnika).
Succ(X:Nastevni_tip):Nastevni_tip;	Vrne vrednost naslednika vrednosti spremenljivke X.
Pred(X:Nastevni_tip):Nastevni_tip;	Vrne vrednost predhodnika vrednosti spremenljivke X.
Ord(X:Nastevni_tip):LongInt;	Vrne zaporedno številko vrednosti spremenljivke X. (Vrednosti so številčene od 0 naprej!)
Ime_tipa(N:Byte):Nastevni_tip;	Podprogram vrne vrednost naštevnega tipa na legi N.

Primeri:

```
Type Barva = (Bela, Rumena, Rdeca, Zelena, Modra);
Var B : Barva;
Begin
  B:=Zelena; {B dobi vrednost Zelena.}
```

```
  Write(Ord(B)); {Izpiše 3.}
  Inc(B); {B dobi vrednost Modra.}
  B:=Barva(1); {B dobi vrednost Rumena.}
  B:=Pred(B); {B dobi vrednost Bela.}
  B:=Pred(B); {Napaka, ker Bela nima definiranega predhodnika!}
  For B:= Bela to Zelena Do
{Spremenljivke naštevnega tipa lahko uporabljamo tudi kot indekse zank.}

    Write(Ord(B)); {Izpiše 0 1 2 3.}
End.
```

Posebnosti dela s spremenljivkami naštevnega tipa:

- Za izpisovanje vrednosti ne moremo uporabljati ukaza **WriteLn**. Tako je npr. ukaz **WriteLn**(B); napačen.

Največkrat si pomagamo tako, da za izpisovanje vrednosti spremenljivk naštevnega tipa napišemo pomožni podprogram. Primer:

```
Procedure Izpisi_barvo( X : Barva);
Begin
  Case X Of
    Bela   : WriteLn('Bela');
    Rumena : WriteLn('Rumena');
    Rdeca  : WriteLn('Rdeca');
    Zelena : WriteLn('Zelena');
    Modra  : WriteLn('Modra');
  End;
End;
```

- Za branje vrednosti ne moremo uporabljati ukaza **ReadLn**. Ukaz je napačen: **ReadLn**(B);

Pomagamo si s pomožnim podprogramom, v katerem preberemo zaporedno številko želene vrednosti in jo prestavimo v ustrezno vrednost naštevnega tipa. Primer:

```
Procedure Preberi_barvo( Var X : Barva);
Var N : Byte;
Begin
  ReadLn(N); {Pri branju številke je smiselno vključiti nadzor za
preverjanje meja vnesene vrednosti.}
  X:= Barva(N);
End;
```

- Spremenljivke naštevnega tipa največkrat uporabljamo za indekse zank in tabel.

6. Urejanje podatkov

Vsi algoritmi za urejanje (sortiranje) podatkov slonijo na eni od treh osnovnih tehnik:

Izbiranju, vstavljanju, premenami

Urejanje z izbiranjem

Pri urejanju z izbiranjem na vsakem koraku izberemo v neurejenem delu tabele najmanjši element in ga zamenjamo z i-tim elementom. Če ima tabela n elementov, je treba ta postopek ponoviti **n-1**-krat. Zato se indeks i spreminja od 1 do n-1.

```
Type Tabela = Array [1..100] Of Integer;
Procedure Sort_izbiranje(Var X : Tabela);
Var I,J,K,Pom : Integer;
Begin
  For I:= 1 to 99 Do{ Postopek ponovimo 99-krat. }
  Begin
   K:=I; Pom:=X[I];{ Zapomnimo si vrednost in indeks mesta na katero
bomo postavili najmanjši element neurejenega dela tabele.}
    For J:=I+1 to 100 Do { Pregledamo neurejeni del tabele;}
      If X[J]<Pom Then     { če najdemo element, ki je manjši od
trenutnega na I-tem mestu, si zapomnimo vrednost in indeks manjšega.}
      Begin
        Pom:=X[J];
        K:=J;
      End;
    X[K]:=X[I]; {Zamenjamo I-ti element z najmanjšim iz neurejenega dela
tabele.}
    X[I]:=Pom;
  End;
End;
```

Urejanje z vstavljanjem

Pri urejanju z vstavljanjem vstavimo na vsakem koraku i-ti element na pravo pozicijo v urejenem delu tabele. Če ima tabela n elementov, je treba ta postopek treba ponoviti **n-1**-krat. Zato se indeks i spreminja od 2 do n

```
Procedure Sort_vstavljanje(Var X : Tabela);
Var I,J,Pom : Integer;
```

```
Begin
  For I:=2 to 100 Do  {I-ti element bomo vstavili na pravo mesto v
urejenem delu tabele. }
  Begin
    Pom:=X[I]; J:=I-1;{ Zapomnimo si vrednost elementa, ki ga vstavljamo
in nastavimo pogoj za konec vstavljanja (stražarja). }
    X[0]:=Pom;
    While Pom<X[J] Do { Dokler ne najdemo vrednosti, manjše od elementa,
ki ga pogrezamo, prestavljamo vrednosti v urejenem delu tabele. }
    Begin
      X[J+1]:=X[J];
      Dec(J);
    End;
    X[J+1]:=Pom; { Element vstavimo na pravo mesto v urejenem delu
tabele. }
  End;
End;
```

Urejanje s premenami

Pri urejanju s premenami na vsakem koraku premaknemo en element na pravo mesto, ostale pa pogreznemo eno ali več mest bliže končnemu. Postopek ponovimo **n-1**-krat.

```
Procedure Sort_premene(Var X : Tabela);
Var I,J,Pom:Integer;
Begin
  For I:=2 to 100 Do { Pogrezanje ponovimo n-1-krat. }
   For J:=100 downto I Do { Elemente od zadnjega do i-tega pogrezamo proti končnemu
mestu. }
    If X[J-1]>X[J] Then
    Begin
     Pom:=X[J-1]; { Premene opravljamo sproti. }
     X[J-1]:=X[J];
     X[J]:=Pom;
    End;
End;
```

7. Zapisi v pascalu

Če želimo uporabljati spremenljivke, sestavljene iz več delov, pri čemer so posamezni deli spremenljivke različnih podatkovnih tipov, uporabljamo zapise (record).

Deklaracija zapisa

```
Type Zapis = Record
       Podatek1 : Tip1;
       Podatek2 : Tip2;
```

```
      ....
      Podatekn : Tipn;
    End;
Var X : Zapis;
```

Do posameznih komponent sestavljene spremenljivke X pridemo tako, da s piko ločimo ime spremenljivke od komponente.

Npr. **X.Podatek1 := Vrednost1;**

Dovoljena je tudi vgnezdena uporaba zapisov. Primer: Osebo opišemo z naslednjimi podatki: imenom (niz 10 znakov), priimkom (niz 20 znakov) in datumom rojstva (dan, mesec, leto).

```
Type TNiz10 = String10;
     TNiz20 = String20;
     TDatum  = Record
        Dan   : 1..31;
        Mesec : 1..12;
        Leto  : 1900..2000;
     End;
     TOseba = Record
        Ime     : TNiz10;
        Priimek : TNiz20;
        Rojen   : TDatum;
     End;
Var X : TOseba;
Begin
....
  X.Ime:='Jani'; X.Priimek:='Lakota';
  X.Rojen.Dan:=15; X.Rojen.Mesec:=5; X.Rojen.Leto:=1980;
...
End.
```

Če je struktura komponent v zapisu lahko odvisna od vrednosti drugih komponent v zapisu, uporabimo deklaracijo record - case.

Deklaracija zapisa z odvisnimi komponentami

```
Type Zapis = Record
        Podatek1 : Tip1;
        Podatek2 : Tip2;
```

```
        Case Podatek3 : Tip3 Of
          Vrednost1 : (Pod41:Tip41; Pod42:Tip42; ...);
          Vrednost2 : (Pod51:Tip51; Pod52:Tip52; ...);
     End;
Var X : Zapis;
```

Primer: Študenta opišemo z imenom in priimkom. Če študent redno študira (komponenta Redni_študij = TRUE), imamo še podatka o letniku in višini štipendije. Če pa študent študira ob delu (komponenta Redni_študij = FALSE), imamo še podatek o imenu podjetja, iz katerega prihaja.

```
Type TNiz10 = String[10];
     TNiz20 = String[20];
     TStudent = Record
       Ime     : TNiz10;
       Priimek : TNiz20;
       Case Redni_studij : Boolean Of
         TRUE : ( Letnik : Byte; Stipendija : Real);
         FALSE: ( Podjetje : TNiz20);
     End;

Var X,Y : TStudent; {Čeprav sta spremenljivki X in Y istega tipa, bosta
vsebovali različne komponente!}

Begin
  X.Ime := 'Matej'; X.Priimek:='Lenitis'; X.Redni_Studij:=FALSE;
  X.Podjetje:='KRTAČA';
  Y.Ime := 'Miha'; Y.Priimek:='Pridnež;'; Y.Redni_Studij:=TRUE;
  Y.Letnik:=4; Y.Stipendija:=13500;
End.
```

8. Datoteke

Pascal omogoča uporabo treh tipov datotek.

- **Tipizirane datoteke** uporabljamo, ko je datoteka urejena po zapisih in je znan sestav zapisa datoteke.
- **Tekstovne datoteke** uporabljamo, ko so podatki urejeni po vrsticah. Tekstovne datoteke so navadne ASCII datoteke. Njihovo vsebino si lahko ogledamo tudi iz operacijskega sistema z ukazom type.
- **Netipizirane datoteke** uporabljamo, ko za delovanje programa ni pomemben sestav datoteke (ali je sestavljena iz zapisov ali iz vrstic). Netipizirane datoteke največkrat uporabljamo takrat, ko je treba napisati program za kopiranje datoteke, primerjavo datotek, krčenje (zgoščevanje) datoteke, arhiviranje datotek,...

Pri delu z datotekami upoštevamo standardno zaporedje postopkov:
1. Datotečni spremenljivki priredimo zunanjo datoteko.
2. Datoteko odpremo.
3. Beremo ali zapisujemo podatke; pri tem praviloma uporabljamo zanko **wihle**.

4. Datoteko zapremo.

Deklaracija tipizirane datoteke

```
Type TDatoteka = File Of Tip_zapisa;
```

Primeri:

```
Type Datoteka_celih_stevil = File Of Integer;
    Zapis = Record
      Besedilo : String;
      Koda : Word;
    End;
    Datoteka_zapisov = File Of Zapis;
```

Vgrajeni podprogrami za delo s tipiziranimi datotekami

Assign(Var F:TDatoteka; Ime:String);	Datotečni spremenljivki F priredi ime zunanje datoteke Ime.
Reset(Var F : TDatoteka);	Odpre datoteko in postavi datotečni kazalec na začetek datoteke. Če datoteka ni, pride do napake.
Rewrite(Var F : TDatoteka);	Ustvari novo (prazno) datoteko in postavi datotečni kazalec na začetek datoteke. Če datoteka obstaja, izbriše obstoječo vsebino.
Read(Var F:TDatoteka;X:Zapis);	Zapis iz datoteke prebere v spremenljivko X in premakne datotečni kazalec na naslednji zapis.
Write(Var F:TDatoteka;X:Zapis);	Zapis X zapiše v datoteko in premakne datotečni kazalec za en zapis naprej.
Seek(Var F:TDatoteka; N:Longint);	Datotečni kazalec postavi na N-ti zapis datoteke. (Prvi zapis datoteke ima zaporedno številko 0!)
FilePos(Var F : TDatoteka):Longint;	Vrne zaporedno številko zapisa, na katerega kaže datotečni kazalec F.
FileSize(Var F : TDatoteka):Longint;	Vrne število zapisov v datoteki.
Eof(Var F : TDatoteka) : Boolean;	Če je datotečni kazalec na koncu datoteke, vrne vrednost TRUE, sicer vrednost FALSE.
Close(Var F : TDatoteka);	Zapre datoteko, na katero kaže datotečni kazalec F.

Primer : Napišite program, s katerim ustvarite datoteko celih števil STEVILA in vanjo vpišete prvih 10 večkratnikov števila 4. Potem izpišite vsebino datoteke na zaslon. Poskusite na zaslon izpisati vsebino datoteke tudi z DOS-ovim ukazom *type*. Pokazali se bodo nenavadni znaki.

```
Type TDatoteka = File Of Integer;
```

```
Var F : TDatoteka;
    I,Stevilo : Integer;
```

Begin
 {Datotečni spremenljivki F priredimo zunanjo datoteko STEVILA in odpremo datoteko.}
 Assign(F,'STEVILA');
 Rewrite(F);
 For I:= 1 **to** 10 **Do** {Desetkrat ponovimo:}
 Begin
  Stevilo:=I*4; {izračunamo i-ti večkratnik števila 4,}
  Write(F,Stevilo); {večkratnik zapišemo v datoteko.}
 End;
 Seek(F,0); {Postavimo se na začetek datoteke.}
 While Not(**Eof**(F)) **Do** {Ponavljamo, dokler ne pridemo do konca datoteke:}
 Begin
  Read(F,Stevilo); {preberemo število,}
  WriteLn(Stevilo);{prebrano število izpišemo na zaslon.}
 End;
 Close(F); {Zapremo datoteko.}
End.

Deklaracija tekstovne datoteke

```
Type Tekstovna_Datoteka = Text;
```

Tekstovne datoteke nimajo neposrednega dostopa do vrstic, zato ne moremo uporabljati ukazov, kot so Seek, FilePos, ...

Vgrajeni podprogrami za delo s tekstovnimi datotekami

Assign(Var F:Text; Ime:String);	Datotečni spremenljivki F priredi ime zunanje datoteke Ime.
Reset(Var F : Text);	Odpre datoteko samo za branje in postavi datotečni kazalec na začetek datoteke. Če datoteka ni, pride do napake.

Rewrite(Var F : Text);	Odpre novo (prazno) datoteko samo za zapisovnje, datotečni kazalec pa postavi na začetek datoteke. Če datoteka je, izbriše obstoječo vsebino.
Append(Var F : Text);	Odpre datoteko samo za zapisovanje, datotečni kazalec pa postavi na konec datoteke. Če datoteka ni, pride do napake.
Read(Var F:Text;X:Tip1; Y:Tip2;...);	Iz datoteke F prebere eno ali več vrednosti v spremenljivke X, Y, ... in jih oblikuje, glede na tip spremenljivk.
ReadLn(Var F:Text;X:Tip1;Y:Tip2;..);	Iz datoteke F prebere eno ali več vrednosti v spremenljivke X, Y, ... in jih oblikuje, glede na tip spremenljivk. Po končanem branju premakne datotečni kazalec F na začetek naslednje vrstice.
Write(Var F:Text;X:Tip1; Y:Tip2;..);	Zapiše eno ali več vrednosti v datoteko, na katero kaže datotečni kazalec F. Obenem premika datotečni kazalec naprej, in sicer za ustrezno število mest.
WriteLn(Var F:Text;X:Tip1; Y:Tip2;..);	Zapiše eno ali več vrednosti v datoteko, na katero kaže datotečni kazalec F. Po končanem zapisovanju premakne datotečni kazalec na začetek naslednje vrstice datoteke.
EoLn(Var F : Text) : Boolean;	Če je datotečni kazalec na koncu vrstice, vrne vrednost TRUE, sicer vrne vrednost FALSE.
Eof(Var F : Text) : Boolean;	Če se datotečni kazalec nahaja na koncu datoteke, vrne vrednost TRUE, sicer vrne vrednost FALSE.
Close(Var F : Tdatoteka);	Zapre datoteko, na katero kaže datotečni kazalec F.

Deklaracija netipizirane datoteke

```
Type Netipizirana_Datoteka = File;
```

Z netipiziranimi datotekami delamo najvarneje tako, da jih obravnavamo kot zaporedje zlogov. To dosežemo tako, da damo drugemu parametru vrednost 1 (npr. Reset(F,1)).

Vgrajeni podprogrami za delo z netipiziranimi datotekami

Assign(Var F:File; Ime:String);	Datotečni spremenljivki F priredi ime zunanje datoteke Ime.
Reset(Var F : File; RecSize : Word);	Odpre datoteko, datotečni kazalec pa postavi na začetek datoteke. Če datoteka ni, pride do napake. Z vrednostjo RecSize določimo velikost zapisa v zlogih.
Rewrite(Var F : File; RecSize:Word);	Odpre novo (prazno) datoteko, datotečni kazalec pa postavi na začetek datoteke. Če datoteka je, izbriše obstoječo vsebino. RecSize pomeni velikost zapisa v zlogih.

BlockRead(Var F:File; Var X:Tip1; Count:Word; Var Result:Word);	Iz datoteke F prebere Count * RecSize zlogov v spremenljivko X. Število v celoti uspešno prebranih zapisov se zapiše v spremenljivko Result. Če je Count * RecSize > SizeOf(X), se v pomnilnik vpišejo vrednosti tudi v spremenljivke, ki sledijo spremenljivki X.
BlockWrite(Var F:File; Var X:Tip1; Count:Word; Var Result:Word);	V datoteko F zapiše iz spremenljivke X Count *RecSize zlogov. Število v celoti uspešno zapisanih zapisov se zapiše v spremenljivko Result. Če je SizeOf(X) = RecSize in Count>1, bo v datoteko zapisana še vsebina spremenljivk, ki v pomnilniku sledijo spremenljivki X.
Eof(Var F : File) : Boolean;	Če je datotečni kazalec na koncu datoteke, vrne vrednost TRUE, sicer vrne vrednost FALSE.
Close(Var F : File);	Zapre datoteko, na katero kaže datotečni kazalec F.

Iz ukazne vrstice lahko damo programom parametre. Pri obravnavi parametrov si pomagamo z vgrajenimi podprogrami:

- **ParamCount** - vrne število podanih parametrov
- **ParamStr (n)** - vrne vsebino n-tega parametra

9. Rekurzije

Ponavljajoče se postopke lahko realiziramo z iteracijo ali rekurzijo. Bistvo iterativnega načina so zanke. Bistvo rekurzije pa je v tem, da izvajamo postopek na vedno manjšem naboru podatkov. Ker pri rekurziji podprogram kliče samega sebe, se vedno začne s pogojem (if), ki ustavi rekurzijo.

Primerjava iterativnega in rekurzivnega podprograma za izpis celih števil iz intervala [Sp..Zg].

```
{
Iterativni postopek
Nastavi Indeks na 5.
Dokler je Indeks<=15, ponavljaj:
  izpiši Indeks,
  povečaj Indeks za 1.
}
Procedure Iteracija(Sp,Zg : Byte);
Var Index : Byte;
Begin
  For Index:=Sp to Zg Do
```

```
    WriteLn(Index);
End;
```

{
 Rekurzivni postopek
 Če je Indeks<=15,
  izpiši Indeks,
  izpiši ostala števila od Indeks+1 do 15.
}

Procedure Rekurzija(Sp,Zg : Byte);
Begin
 If Sp<=Zg **Then** {Če spodnja meja še ni prišla do zgornje, izpiši število in izpiši še ostala števila iz intervala [sp+1..zg].}
 Begin
  WriteLn(Sp);
  Rekurzija(Sp+1,Zg); {Ponavljanje dosežemo tako, da podprogram pokliče samega sebe, le da so parametri nekoliko spremenjeni.}
 End;
End;
Begin
 Iteracija(5,15);
 Rekurzija(5,15);
End.

10. Množice

Pascal omogoča deklariranje spremenljivk, ki so množice. Pri deklariranju tovrstnih spremenljivk smo omejeni z velikostjo domene množice, ki ima lahko največ 256 različnih znakov. Za domeno množice največkrat uporabljamo podatkovni tip Byte, Char ali uporabniško definirane naštevne tipe.

Deklaracija množice

```
Type Mnozica = Set Of Tip _elementa_mnozice;
```

Primeri

```
Type    Male_crke = 'a'..'z';
        Mnozica_malih_crk = Set Of Male_crke;
        Barve = (Bela, Rumena, Rdeca, Zelena, Modra);
        Mnozica_barv = Set Of Barve;
Var C : Male_crke;
    B : Barve;
```

- Spremenljivki, ki je množica, priredimo vrednost tako, da elemente podamo v oglatih oklepajih. Primeri:

```
C:=['m','p'];{Množici C priredimo elementa 'm' in 'p'.}
C:=[];{Množica C naj bo prazna.}
```

```
B:=[Bela];{Množici B priredimo element Bela.}
B:=[Modra,Zelena];{Množici B priredimo elementa Modra in Zelena.}
```

Operatorji za delo z množicami

+	Unija množic
-	Razlika množic
*****	Presek množic.
in	Element množice.
=	Ugotavljanje enakosti množic.
<>	Ugotavljanje neenakosti množic.
<=	Ugotavljanje podmnožice.

- Elemente množice izpisujemo ponavadi tako, da nastavimo zanko, katere indeks se spreminja od začetne do končne vrednosti domene množice. Če je vrednost indeksa v množici, to vrednost izpišemo. Primer:

```
For Index:='a' to 'z' Do
  If Index In C Then
    WriteLn(Index);
```

11. Kazalci

Do vsebine pomnilniškega mesta lahko pridemo s pomočjo imena spremenljivke, ki je na tem mestu, ali s pomočjo kazalčne spremenljivke, ki kaže na neko pomnilniško mesto.

Deklaracija kazalčnega tipa

```
Type Kazalec = ^Podatkovni_tip;
```

Velikost spremenljivke je odvisna od podatkovnega tipa, uporabljenega pri deklaraciji spremenljivke Npr. spremenljivke tipa Char in Byte zavzamejo v pomnilniku 1 zlog, spremenljivke tipa Word in Integer zavzamejo v pomnilniku 2 zloga, ... Vsaka kazalčna spremenljivka zavzame natanko **4 zloge**, **ne glede na velikost strukture, na katero kaže.** V dveh zlogih je segment pomnilniškega mesta, v dveh zlogih pa odmik od začetka segmenta (ofset lokacije), na katerega kaže kazalčna spremenljivka.

Operatorji za pridobivanje pomnilniškega naslova

@	Vrne naslov spremenljivke S.
Addr(S)	Vrne naslov spremenljivke S.
Seg(S)	Vrne naslov spremenljivke S.

Ofs(S)	Vrne naslov spremenljivke S..
Ptr(Seg(S),Ofs(S))	Vrne naslov spremenljivke S.

Dostop do vsebine, na katero kaže kazalčna spremenljivka
Kazalcna_spremenljivka^

Primeri:
Var Ch1, Ch2 : **Char**;
PCh : **^Char;**
PInteger **: ^Integer;**
PBoolean **: ^Boolean;**
Begin
Ch1:='A'; Ch2:=Chr(1);
PCh:=**Addr**(Ch1); {PCh kaže na spremenljivko Ch1.}
PBoolean:=**Addr**(Ch2); {PBoolean kaže na spremenljivko Ch2.}
WriteLn(PCh^,PBoolean^); {Izpis : **A** (ker je v Ch1 'A') in **TRUE** (ker je v Ch2 1).}
Inc(PCh^); {Poveča vsebino, na katero kaže PCh, in s tem vrednost spremenljivke Ch1.}
PInteger:=**Ptr**(**Seg**(Ch1),**Ofs**(Ch1)); {PInteger kaže na Ch1.}
WriteLn(Ch1, PInteger^); {Izpis:
B (ker smo s PCh povečali vrednost spremenljivke Ch1)
322 (ker PInteger kaže na dvozložni tip Integer, se pri izračunu vrednosti upoštevata lokaciji spremenljivk Ch1 /spodnji zlog/ in Ch2 /zgornji zlog/).}
Dec(PBoolean^); {Zmanjša vsebino na katero kaže PBoolean, spremenljivko Ch2 torej postavi na 0.}
WriteLn(PInteger^); {Izpis: **66** (ker smo Ch2 /zgornji zlog/ zmanjšali na 0, se vrednost celega števila izračuna le na osnovi vrednosti spodnjega zloga /Ch1/).}
End.

Operatorji za delo s kazalci

Inc(Var Kazalec)	Vsebino odmika od začetka segmenta kazalčne spremenljivke poveča za velikost sestave (strukture), na katero kaže kazalčna spremenljivka.
Dec(Var Kazalec)	Vsebino odmika od začetka segmenta kazalčne spremenljivke zmanjša za velikost sestave, na katero kaže kazalčna spremenljivka.
=	Če kazalčni spremenljivki kažeta na isto pomnilniško mesto, vrne vrednost TRUE, sicer vrne vrednost FALSE.
<>	**Not**=

Primeri:
```
Var Ch1, Ch2 : Char;
```

```
    PCh : ^Char;
    PInteger : ^Integer;
    PBoolean : ^Boolean;
Begin
  Ch1:=Chr(0); Ch2:='A';
  PCh:=Addr(Ch1); {PCh kaže na spremenljivko Ch1.}
  PBoolean:=Addr(Ch2); {PBoolean kaže na spremenljivko Ch2.}
  Dec(PBoolean); {Zmanjša ofset, na katerega kaže PBoolean za 1;
PBoolean zdaj kaže na Ch1.}
  Inc(PCh); {Poveča ofset, na katerega kaže PCh za 1, PCh zdaj kaže na
Ch2.}
  Inc(PCh^); {Poveča vsebino, na katero kaže PCh, torej poveča vrednost
spremenljivke Ch2 za 1.}
  WriteLn(PCh^,PBoolean^); {Izpis : B (ker je v Ch2 'B') in FALSE (ker
je v Ch1 0).}
End.
```

Na kopici lahko realiziramo **dinamične podatkovne strukture**, kot so seznami in drevesa.

Ukazi za delo s kopico (Heapom)

New(Var Kazalec)	• Kazalčni spremenljivki priredi trenutno vrednost HeapPtr-ja. • Na kopici rezervira prostor za velikost sestave, na katero kaže kazalčna spremenljivka. • HeapPtr poveča za velikost sestave, na katero kaže kazalčna spremenljivka.
Dispose(Var Kazalec)	Sprosti prostor na kopici, na katerega kaže kazalčna spremenljivka **Kazalec**.

12. Prekinitve v pascalu

Vse DOS in BIOS funkcije so dostopne s pomočjo ustreznih programskih prekinitev. Običajni način dostopa do funkcij operacijskega sistema DOS je klic prekinitve 21H Osnovni mehanizem za dostop do BIOS in DOS funkcij je:

- registre napolnimo z ustreznimi vrednostmi
- generirmo programska prekinitev
- če obstaja, preverimo dobljen rezultat.

Klic prekinitve iz Turbo pascala

```
Intr(InterruptNumber : Byte, Regs : Registers);
```

ali

```
MsDos(Regs : Registers);
```

Procedura MsDos kliče vedno prekinitev 21H (klic DOS funkcij).
Podatkovni tip Registers je opisan v knjižnici (unit) DOS.

Primer: podprogram, ki preveri ali je instalirana miška.

Številka prekinitve = 33H
Številka funkcije = 00H
Vrnjene vrednosti v registrih:
AX = status (FFFFH ali 0000H)
BX = število gumbov miške

```
Function Miska_instalirana : Boolean;
Begin
  {V AX register vpišemo številko funkcije.}
  Regs.AX:=$0000;
  {Pokličemo prekinitev števika 33H (delo z miško).}
  Intr($33,Regs);
  {Miška je instalirana, če je register AX=$FFFF. }
  Miska_instalirana:= Regs.AX = $FFFF;
End;
```