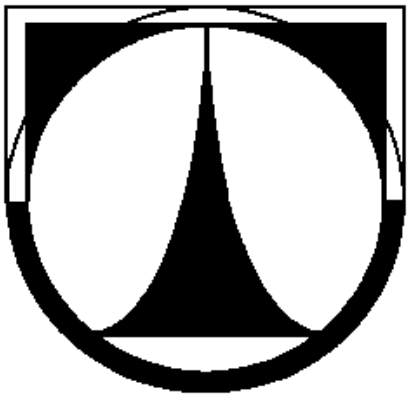
Technická Univerzita v Liberci
Fakulta strojní **Katedra výrobních systémů a automatizace**

prof. Ing. Miroslav Olehla, CSc.

Počítače a programování - Pascal

Computers & programming - Pascal

Studijní podklady



Liberec 2017

Lektoroval: doc. Ing. Libor Tůma, CSc.

ISBN 978-80-7494-340-9

Předmluva:

Předkládaný učební text je určen studentům 1. ročníku fakulty strojní v Liberci jako **pomůcka a doplněk** přednášek při studiu předmětu "Počítače a programování". Hlavní důraz je kladen na praktické zvládnutí programovacího jazyka TURBO PASCAL. Snahou je provést skripta ve formě studentům co nejbližší, a jsou proto opatřena značným počtem příkladů a obrázků obrazovky tak, jak se při práci na počítači zobrazí.
Tato skripta navazují a předpokládají prostudování skript *Počítače a programování VD*, TU LIBEREC 2015, ve kterých lze získat nutné základní znalost a přehled o zápisu vývojových programů a jedno - duchých programů.
Děkuji Ing. Miroslavu Vavrouškovi za pečlivou kontrolu uvedených skrit.

OBSAH:

Úvod

Programovací jazyk Pascal byl navržen v roce 1971 profesorem N.Wirthem z Curyšské university s cílem vytvořit programovací jazyk splňující zásady strukturovaného programování, což umožňuje psát programy systematicky, přehledně a dokonce provést ověření správnosti algoritmu. Autor Pascalu se snažil vytvořit jazyk vhodný pro systematickou výuku programování, založený na jasných a srozumitelných konstrukcích a umožnit jednoduchou a efektivní realizaci překladačů tohoto jazyka na současných výpočetních systémech. Dokladem splnění těchto cílů je stále trvající popularita Pascalu mezi studenty i mezi programátory a dnes již existuje značné množství jeho implementací.

Ve skriptech je popsána v současné době nejužívanější verze Pascal 7.0 fy Borland. Od standardního Pascalu se liší především způsobem komunikace uživatele s počítačem. Dále jsou zavedeny rozšířené specifikace konstant, typu proměnných, standardních a uživatelských modulů - unit. Moduly umožňují práci s okénky, grafickými příkazy a odstraňují i nedostatek starších verzí, omezení cílového kódu na 64 KB, čímž bylo Turbo vyřazeno z použití při větších projektech. Oproti standardnímu Pascalu nejsou do Borland zahrnuta konformní pole, příkazy get a put a některé další, méně významné příkazy. Text nebo příklady označené symbolem  , jsou určeny zvídavějším studentům.

1. PROGRAMOVÁNÍ V PASCALU

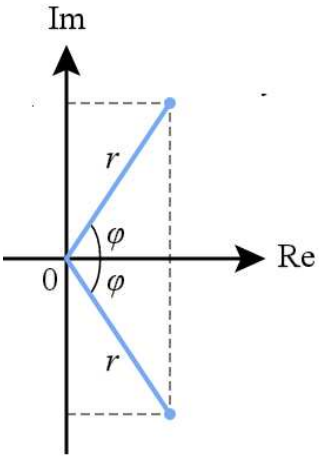
1.1 Jak začneme ?

Nejdříve je nutno vytvořit zdrojový text programu. Pro jednoduché úlohy jej můžeme vytvořit přímo zápisem do počítače, pro složitější příklady jej zapíšeme raději nejprve na papír.

Sestavme program pro řešení kořenů kvadratické rovnice.
Pro kvadratickou rovnici $ax^2 + bx + c = 0$ s reálnými koeficienty, kde $a \neq 0$ a pokud má nezáporný diskriminant D, řešení provedeme podle vztahu $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, kde $D = b^2 - 4ac$.
V případě, že je diskriminant záporný, kořeny rovnice získáme ze vztahu $x_{1,2} = \frac{-b \pm \sqrt{|D|}}{2a}$ nebo $x_{1,2} = \frac{-b \pm \sqrt{-D}}{2a}$, kde reálný kořen je násobný $x_{1,2} = \frac{-b}{2a}$ a imaginární kořeny jsou komplexně sdužené $x_1 = \frac{+\sqrt{-D}}{2a}$ a $x_2 = \frac{-\sqrt{-D}}{2a}$, což vidíme ze znamének u imaginaráních složek. [1]

```
{výpočet kořenu kvadratické rovnice}
var X1R,X1I,X2R,X2I,A,B,C,D:real;
begin
write('Zadej koeficienty A B C :');
read(A,B,C);
if A=0 then begin
            X1R:=-C/B;
            writeln('jednoduchý kořen' ,X1R:10:4)
           end
      else begin
            D:=sqr(B)-4*A*C;
            if D>=0 then {výpočet reálných kořenu}
                    begin
                      X1R:=(-B+sqrt(D))/(2*A);
                      X2R:=(-B-sqrt(D))/(2*A);
                      X1I:=0;
                      X2I:=0
           end
```

[1] Znázornění komplexního čísla a čísla k němu komplexně sdruženého v komplexní rovině.



```
                else {výpočet komplexních kořenu}
                  begin
                    X1R:=-B/(2*A);
                    X1I:=sqrt(abs(D))/(2*A);
                    X2R:=X1R;
                    X2I:=-X1I
                  end;
            writeln('kořeny kvadratické rovnice:');
            writeln(' ',X1R:10:4,' +i * ',X1I:10:4);
            writeln(' ',X2R:10:4,' +i * ',X2I:10:4)
            end
end.
```

```
Zadej koeficienty A B C: 1  -3  -10
Koreny kvadraticke rovnice:
  5.0000 + i*   0.0000
 -2.0000 + i*   0.0000
```

[A)]**Příklad pro pobavení:**
Sestavme program, který přečtený znak změní o zvolenou pozici v následující tabulce. Posun v našem příkladě volíme 1. Jako tabulku lze použít například kód ASCII, tedy znak A (ordinální číslo 65) změníme na znak B (ordinální číslo 66) atd. *Ord* je funkce pro určení pozice znaku v tabulce, *char* vybere znak na určené pozici v tabulce.

Znak	…	A	B	C	D	E	F	G	H	…
Pozice v tab. - ord(znak)	…	65	66	67	68	69	70	71	72	…
Posunutá pozice - ord(znak)+1	…	66	67	68	69	70	71	72	73	…
Nový znak - char(posunutá)	…	B	C	D	E	F	G	H	I	…

Princip této techniky lze využít po náležité úpravě například ke kryptografii. Například volbu posunu možno provést pomocí náhodných čísel, zvolit vlastní tabulku atd.

```
var znak_plus, znak:char;
begin
  read(znak);
  znak_plus:=char(ord(znak)+1); {zvolený posun je 1}
  writeln(znak_plus);
end.
```

```
Data:
A
Výsledek:
B
```

[A)] Kryptografie je věda zabývající se šifrováním. Cílem šifrování je upravit zprávu k neporozumění, tedy aby zprávě rozuměl pouze její příjemce. Na WWW lze získat velké množství literatury.

ENIGMA je z literatury a filmů známý přenosný mechanismus, používaný k šifrování a dešifrování tajných údajů. Využíval se od počátku dvacátých let dvacátého století, pro šifrování zpráv ve druhé světové válce.

1.1.1 Zápis programu

Jazyk Pascal patří mezi jazyky s blokovou strukturou programu. Základem této struktury je blok, který obsahuje deklarace a příkazy a které tvoří rámec platnosti deklarací v ní uvedených. Jednotlivé deklarace se v bloku sdružují do několika úseků - deklarace návěští, konstant, typů, proměnných a procedur a funkcí. Jednotlivé deklarační úseky nejsou povinné a vyskytují se pouze, je-li potřeba v tomto bloku deklarovat objekty příslušného druhu.

Příkazovou částí bloku se popisuje vlastní výpočet. Tato část bloku má tvar složeného příkazu, tj. je to posloupnost příkazu oddělených středníkem a uzavřená mezi omezovače begin a end.

Program je blok, jemuž může předcházet hlavička programu a který je zakončen tečkou. Pro zápis programu není předepsán pevný formát. Jediné omezení je, že ve zdrojovém programu nelze rozdělit mezerou či novým řádkem lexikální element (tj. speciální symbol včetně rezervovaných slov, identifikátor, konstantu či návěští) a dále, že každé dva po sobě jdoucí lexikální elementy je třeba oddělit pomocí oddělovačů lexikálních elementů, např. alespoň jednou mezerou nebo novým řádkem.

☞ **ANO**

```
var B,A: real;
'PRAHA'
A:=3.1415;
```

☞ **NE**

```
      varB,A: real;
      'PRA
            HA'
A:=3.1415;
```

Volný formát programu umožňuje psát zdrojový text tak, aby byl pro vás co nejpřehlednější. Některé možné formáty ilustrují čitelnost či nečitelnost programu.

a) ☹

```
var A,B: integer; begin read(A); read (B); write(A,B) end.
```

b) ☺

```
var A,B: integer;
begin
    read (A);
    read (B);
    write (A,B)
end.
```

Pro zvýšení srozumitelnosti je vhodné uvádět komentáře uzavřené mezi složené závorky

```
var A,B: integer;
  begin
  {program pro ilustraci komentáře}
  end.
```

Zdrojový text programu v jazyce Pascal vždy obsahuje následující části:

- deklarační část. Před použitím proměnné, návěští, funkce či procedury, případně konstanty je nutno tyto blíže specifikovat. Specifikaci je možno provést pomocí standardních identifikátorů typu nebo si je specifikuje programátor sám.
- příkazová část. V této části je již možno zapsat příkazy ke čtení, tisku, aritmetické či logické příkazy, podmíněné příkazy, atd. V příkaze je možno využít jen ty proměnné, návěští, funkce, atd., které byly specifikovány v deklarační části a používat jen ty operace, které daný programovací jazyk připouští. Uveďme jednoduchý příklad zápisu programu v Pascalu, který řeší daný problém a objasní strukturu jazyka.

```
Program SUMA (input, output);
```

Hlavička programu s uvedením jména programu SUMA, *input* značí vstupní textový soubor, *output* značí výstupní textový soubor. Hlavička programu v Turbo Pascalu nemusí být uváděna.

```
{deklarační část}

var   I, N: integer;
      A, SUMA: real;
```

Komentář uzavřen do složených závorek. Objasňuje akci v programu.
Deklarace proměnných *I,N,A,SUMA* pomocí standardních identifikátorů typu *integer* a *real. I,N* jsou celočíselné proměnné, *SUMA, A* jsou reálné proměnné.

```
{příkazová část}
begin

read (input, N);

SUMA := 0.0;

for I := 1 to N do
begin

read (input, A);
SUMA := abs (A) + SUMA



end;
write (output, ´SUMA:´,SUMA)



end.
```

Komentář.
Začátek příkazové části označen klíčovým slovem **begin.**
Přiřazení hodnoty celočíselné proměnné *N* příkazem vstupu. *Read* je standardní procedura pro vstup z textového souboru *input*.
Přiřazení reálné konstanty na reálnou proměnnou *SUMA*. Dvouznakový symbol := značí přiřazení.
Příkaz opakování. Hodnota *I*:= 1,2,..,*N*.
for, **to**, **do** jsou klíčová slova.
Začátek složeného příkazu označen klíčovým slovem begin.
Čtení na proměnnou *A*.
Aritmetický výraz pro součet obsahu proměnných *A*, kde + značí operaci sčítání pomocí speciálního symbolu +, *abs* označuje standardní identifikátor funkce pro výpočet absolutní hodnoty argumentu *A*.
Konec složeného příkazu.
Příkaz pro tisk řetězce *SUMA*: a obsahu proměnné *SUMA*. *Write* je standardní procedura pro výstup do textového souboru *output*.
Konec příkazové části, klíčové slovo **end** musí být ukončeno speciálním symbolem tečka.

Poznámka:
- U příkazu *read* a *write* se jméno textového souboru *input, output* zpravidla neudává.
- Klíčová,rezerovovaná slova jsou uvedena tučně,standardní identifikátory typů, funkcí a procedur jsou uvedeny kurzivou. Při praktickém zápisu se ovšem toto rozlišení neprovádí, rovněž zápis malými či velkými písmeny se nerozlišuje.
- Před **end** se může, ale nemusí zapisovat středník.

1.1.2 Základy programování

Sestavení algoritmu řešení úlohy na počítači je základní a rozhodující tvůrčí činnost při programování. Algoritmizace úlohy tvoří hlavní, nikoliv však jedinou etapu programování. Před vlastní algoritmizací úlohy, ať již vědeckotechnické nebo ekonomické, je třeba úlohu matematicky formulovat, vytvořit vhodný model či zvolit vhodnou numerickou metodu. Po sestavení algoritmu je pak třeba přepsat algoritmus do zvoleného programovacího jazyka. Na prohřešky proti pravidlům pro zápis programu nás upozorní překladač. Logické chyby, které způsobí jiný průběh výpočtu, než jaký požadujeme, je třeba hledat ověřením programu, tzv. laděním. V této fázi porovnáváme výsledky ověřovacích výpočtů s výsledky známými a odstraňujeme příčiny neshod úpravami algoritmu a programu.

Věnujme nyní pozornost algoritmizaci jednoduché úlohy, sečtení několika čísel. Stanovme si za úkol určit součet pěti čísel, např. 3+2+1+7+4. Je zřejmé, že v případě určení součtu jiných čísel je nutné výpočetní postup sčítání opakovat.

Program pro součet pěti číselných konstant:

```
{příkazová část}
begin
  write(output,3+2+1+7+4)          {zobrazení výsledku 3+2+1+7+4}
end.
```

Výsledek součtu často potřebujeme k dalšímu výpočtu. V tomto případě je nutno deklarovat použitou proměnnou - její jméno a její typ.

```
{deklarační část}
var SUMA:real;                     {deklarace proměnné SUMA jako reálná}
{příkazová část}
begin
  SUMA:=3+2+1+7+4;                 {součet 3+2+1+7+4 uložíme na SUMA}
 write(output,'SUMA JE: ',SUMA)    {zobrazíme výsledek, obsah SUMA}
end.
```

V případě, že bychom tento postup prováděli opakovaně s různými čísly, je vhodnější provádět sumaci s proměnnými, např. *A+B+C+D+E*. Místo konkrétních čísel jsme použili proměnné (podobně jako v matematice),jimž musí být před použitím přiřazeny hodnoty. Proměnnou chápeme jako paměťové místo,které má své jméno a do něhož lze uložit hodnotu.Uložit hodnoty můžeme přiřazením

```
{deklarační část}
var A,B,C,D,E,SUMA:real;
{příkazová část}
begin
 A:=3;
 B:=2;
 C:=1;
 D:=7;
 E:=4;
 SUMA:=A+B+C+D+E;                  {součet A,B,C,D,E na SUMA}
 write(output,'suma je: ',SUMA);
end.
```

nebo obecněji lze hodnoty na proměnné uložit čtením odpovídajících dat z klávesnice

```
{deklarační část}
var A,B,C,D,E,SUMA:real;
{příkazová část}
begin
  read(input,A,B,C,D,E);           {čtení požadovaných hodnot z klávesnice}
  SUMA:=A+B+C+D+E;
  write(output,'suma je: ',SUMA)   {zobrazení výsledku na monitoru}
end.
```

Pomocí dříve uvedeného postupu lze sečíst pětici jakýchkoliv čísel. V případě, že je požadováno sečíst jiný počet čísel, bylo by nutné program vždy přepsat. Je však možno sestavit obecnější program pro sečtení *n* čísel $s: = a_1 + a_2 + ... + a_n = \sum_{i=1}^{n} a_i$

```
{deklarační část}
var I,N:integer;                    {deklarace I,N typu celočíselná}
   A,SUMA:real;                     {deklarace A,SUMA typu reálná}
{příkazová část}
begin
 read(input,N);                     {čtení na N}
 SUMA:=0.0;                         {přiřazení nuly na SUMA}
 for I:=1 to N do                   {příkaz opakovaní}
 begin                              {začátek složeného příkazu}
  read(input,A);                    {čtení na A}
  SUMA:=SUMA+A                      {sečtení obsahu SUMA a A}
 end;                               {konec složeného příkazu}
write(output,'SUMA: ',SUMA)         {tisk výsledku}
end.
```

Na takto jednoduchém příkladě lze ilustrovat rovněž použití indexů. Indexované proměnné a jejich deklarace budou probrány v dalším textu, proto jen uveďme jejich výhodu a nevýhodu pro uvedený příklad.

1) program zabírá větší rozměr paměti. Místo čtení na jednu proměnnou *a*, se čte na proměnné a_1, a_2, ..., a_n. Je-li *n* příliš velké, je použití tohoto způsobu nemožné.
2) program je pro začátečníky pochopitelnější, pracuje se se všemi zavedenými proměnnými.

Poznámka:

Výpočet se provádí podle tzv. rekurentního vztahu. Jde o úlohy, které se týkají výpočtu *n*-tého členu posloupnosti pomocí vztahu mezi *i*-tým členem posloupnosti a jeho předchůdci, které platí pro obecné *i* a jsou doplněny o definici počátečních členů posloupnosti.

Nechť s_i je součet prvních *i* členů řady *a*, tj. $s_i = a_1 + a_2 + ... + a_i$. Položíme-li $s_o = 0$, pak hodnoty s_i tvoří řadu definovanou rekurentně takto

$$S_0 = 0$$
$$S_i = a_i + S_{i-1} \quad \text{pro } i > 0$$

```
{deklarační část}
var A:array[1..100] of real;        {deklarujeme A jako pole A[1],A[2],..,A[N] reálných
      N,I:integer;                  proměnných}
        S:real;                     {deklarace promenných N a I jako celočíselné}
{příkazová část}                    {deklarace proměnné S jako  reálná}
begin
      read(N);
      for I:=1 to N do              {čtení na proměnnou N}
         read(A[I]);                {příkaz opakovaní, cykl pro I=1 až N}
          S:=0;                     {čtení A[1],A[2],..,A[N]}
          for I:=1 to N do          {přiřazení nuly na proměnnou S}
         S:=S+A[I];                 {nový příkaz cyklu}
       writeln('SUMA:',S)           {součet 0+A[1]+A[2]+..+A[N]}
end.                                {zobrazení výsledku}
```

U tohoto jednoduchého příkladu lze ukázat rovněž využití koncové značky. Uvažujme např., že není znám počet členů řady, ale víme, že posledním členem řady (nezapočítává se do zpracovávané řady) bude číslo -9999 (toto číslo se v řadě jinak nevyskytuje).

```
{deklarační čast}
var A,S:real;                    {deklarace A,S jako typ realné}
label L1;                        {deklarace náveští}
{příkazová část}
begin
S:=0;                            {přiřazení nuly na proměnnou S}
L1:                              {označení řádku navěštím}
read(A);                         {čtení na proměnnou A}
if A<>-9999 then                 {jestliže A se nerovná -9999 vykonej}
        begin                    {začátek složeného příkazu}
        S:=S+A;                  {sečti obsah S a A a výsledek ulož na S
        goto L1                  {skok na řadek s návěštím L1}
        end                      {konec složeného příkazu}
write('SUMA:',S)                 {zobrazení výsledku}
end.
```

Tento zápis není vhodný, používáme příkaz skoku. Lépe je použít následující zápis

```
{deklarační část}
var A,S:real;                    {deklarace A,S jako realné}
{příkazová část}
begin
   S:=0;                         {přiřazení nuly na S}
   read(A);                      {čtení na A}
   while A<>-9999 do             {pokud A se nerovná -9999 pak vykonej}
   begin                         {začátek složeného příkazu}
         S:=S+A;                 {sečti obsah S a A a výsledek ulož na S}
         read(A);                {čti A}
    end;                         {konec složeného příkazu}
    write('SUMA:',S)             {zobrazení výsledku}
end.
```

V mnoha případech je použití indexů chybné. Uveďme příklad takového postupu na příkladu pro výpočet druhé odmocniny z čísla $x>0$ s přesností $\in$.

Podle Newtonovy metody je druhá odmocnina z nezáporného čísla x definovaná rekurentním vztahem

$$y_o = 1$$

$$y_i = \frac{1}{2}\left[\frac{x}{y_{i-1}} + y_{i-1}\right] \quad pro \quad i = 1,2,...$$

Výpočet ukončíme, je-li splněn vztah $\left|y_i - y_{i-1}\right| < \varepsilon$

```
{deklarační část}
var X:real;                                  {deklarace X}
I:integer;                                   {deklarace I}
Y:array[1..100] of real;                     {deklarace Y jako pole 1 až  100}

 {příkazová část}
begin
    read(X);                                 {čtení na X}
    I:=1;                                    {1 přiřadíme na I}
    Y[I]:=1;                                 {1 přiřadíme na Y[1}}
    repeat                                   {opakuj}
        I:=I+1;                              {I+1 přiřadíme na I}
        Y[I]:=(X/Y[I-1]+Y[I-1])/2;           {(X/Y[I-1]+Y[I-1])/2 přiřadíme na Y[I]}
            until abs(Y[I]-Y[I-1])<1e-6;             {až do splnění podminky, že
     abs(Y[I]-Y[I-1])<1*10^-6}
    write(X,Y[I])                            {zobrazení výsledku}
end.
```

Takto sestavený algoritmus není správný. Je sice přesným vyjádřením rekurentních vztahů, ale indexování proměnných y_1, y_2, ... je chybné. Konvergence $|y_i - y_{i-1}| < \varepsilon$ může nastat po 10 krocích, ale také po 1000 krocích. Je proto nutné tyto typy algoritmů zapisovat bez použití indexovaných proměnných, tedy ve tvaru:

```
{deklarační část}
var X,Y1,Y2:real;
    I:integer;

{príkazová část}
begin
    read(X);
    Y2:=1;
    repeat
        Y1:=Y2;
        Y2:=(X/Y2+Y2)/2;
    until abs(Y2-Y1)<1e-6;
     write(X,Y2)
end.
```

Programovací jazyk Pascal umožňuje využívat další typy proměnných (ne pouze celočíselné a reálné) a další příkazy (ne pouze přiřazovací, podmíněné a cykly). Tyto příkazy a proměnné budou postupně vysvětleny v následujících kapitolách.

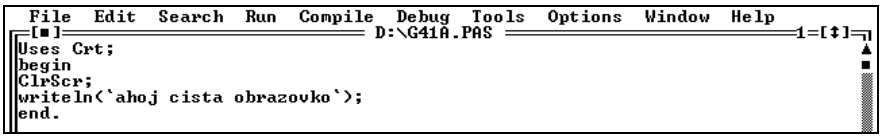
2. DOPORUČENÍ PRO TVORBU PROGRAMŮ

Uveďme několik základních příkazů, které je vhodné využívat při zápisu i jednoduchých programů. S podrobným popisem významu uvedených příkazů je možno se seznámit až v dalším textu.

V žádném případě se nenechte "otrávit" čtením této kapitoly. Všechny příkazy jsou popsány v dalších kapitolách. Příklady slouží jen k ilustraci možných problémů při tvorbě programů v Pascalu a k získání návyků při jejich zápisu.

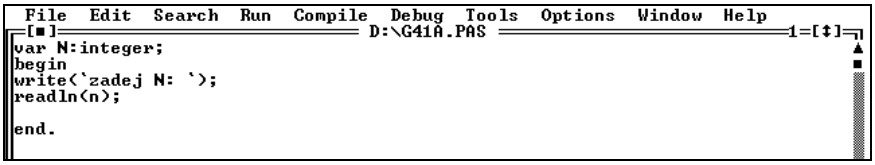
2.1 Mazání obrazovky

Před zobrazením výsledků našeho programu je vhodné smazat výsledky předchozích programů, zobrazených na uživatelské obrazovce např. následovně.



2.2 Čtení dat

Program nemusí vždy vykonávat akce, pro které byl připraven (samozřejmě nedokonalostí tvůrce programu). Pro čtení dat je z tohoto důvodu, ale i z důvodu lepší přehlednosti vhodné zobrazit upozornění, co chceme vlastně číst.

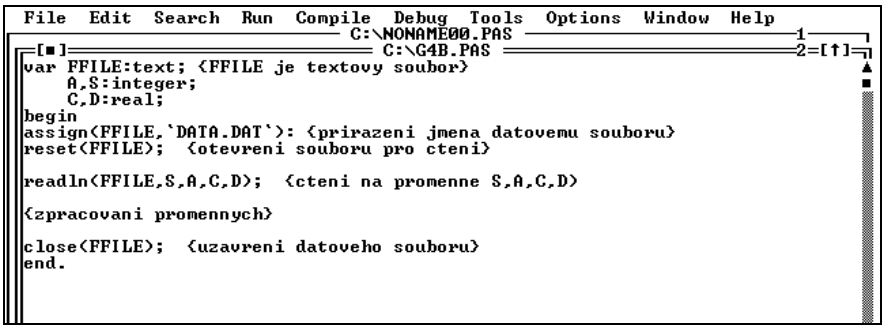


Čtení souboru dat - přesměrování vstupu

Málokdy se nám podaří odladit program napoprvé. Pak je ovšem nutné po opravě chyby v programu a jeho spuštění zapisovat vstupní data znovu a znovu. U Pascalu je možno číst data z předem připraveného datového souboru, čímž odpadne časté opakování "vyklepávání" již dříve vytvořených dat.

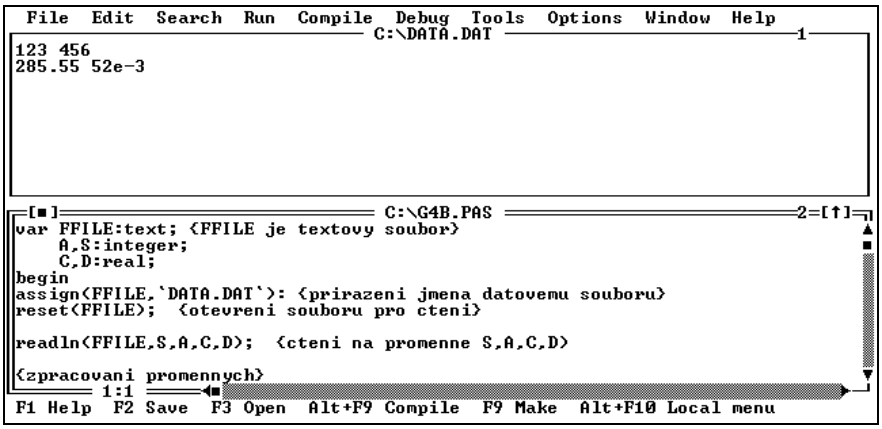
Je tím rovněž odstraněna nutnost psaní nových dat, byla-li zapsána chybná data a byla odeslána stlačením klávesy Enter.

Čtení datového souboru lze provést posloupností příkazů



Vytvoření datového souboru provedeme následovně:

Volíme příkaz File/New a do editačního okna zapíšeme požadovaná data. Uložíme soubor pomocí File/Save As pod zvoleným jménem, např. DATA.DAT.



2.3 Proměnná jako pole

V předchozím příkladě byla uvedena možnost čtení dat z předem připraveného datového souboru. Uveďme praktický příklad, ilustrující další možnosti využití tohoto postupu i pro jiné případy.

Sestavme program pro určení střední hodnoty a rozptylu pro naměřené hodnoty x_i ,i=1 až n. Pro výpočet střední hodnoty a rozptylu se používá například známý vztah

$$\bar{x} = \sum_{i=1}^{n} x_i / n$$

$$s^2 = \sum_{i=1}^{n} (x_i - \bar{x})^2 \ / (n-1)$$

Tento algoritmus využívá k řešení matematické definice střední hodnoty a rozptylu a je přesný. Je však vhodný v případě, že všechny hodnoty x_i jsou uloženy v operační paměti. Jestliže toto není splněno, je nutno číst dvakrát ať již z klávesnice (při větším počtu dat si toto rychle uvědomíme), nebo čtením z předem připraveného datového souboru na disku. Uveďme příklad - spíše odstrašující - pro dvojí čtení dat x_i, i=1 až n z klávesnice.

```
{výpočet střední hodnoty a rozptylu  pro dvoji čtení}
uses crt;
var N,I:integer;
   X,X1,S2:real;
begin
clrscr;
writeln('zadej počet hodnot N a hodnoty X(i), i=1 až N ');
read(N);
{výpočet střední hodnoty}
  X1:=0;
  for I:=1 to N do
  begin
    read(X);
    X1:=X1+X
  end;
  X1:=X1/N;
{výpočet rozptylu}
  S2:=0;
  writeln('zadej znovu hodnoty X(i), i=1 až N');
  for I:=1 to N do
  begin
    read(X);
    S2:=S2+(X-X1)*(X-X1)
  end;
  S2:=S2/(N-1);

{tisk výsledných hodnot}
writeln('střední hodnota: ',X1,' rozptyl: ',S2);
end.
```

Program pro čtení dat uložených na disku či disketě je možno sestavit následově:

```
{ výpočet střední hodnoty a rozptylu pro čtení z disku}
uses Crt;
var N,I:integer;
    X,X1,S2:real;
    F:text; {F – textovy soubor}
begin
ClrScr;
assign(F,'C:\DATA.DAT'); {F – prirazeni jmena datovemu souboru }
reset(F); {otevření a nastaveni cteni na pocatek soubor}
readln(F,N); {čtení dat z disku}
X1:=0;
for I:=1 to N do
  begin
   read(F,X); {čtení dat z disku}
   X1:=X1+X;
  end;
X1:=X1/N;
{nové čtení dat z disku}
reset(F); { nastaveni cteni na pocatek soubor}
S2:=0;
for I:=1 to N do
  begin
  read(F,X); {čtení dat z disku}
  S2:=S2+(X-X1)*(X-X1);
  end;
S2:=S2/(N-1);
{tisk výsledku}
writeln('střední hodnota: ',X1);
writeln('rozptyl : ',S2);
close(F); {uzavreni datoveho souboru }
end.
```

Uveďme další techniku naprogramování tohoto problému.

Hodnoty x_i, i=1 až n lze uložit do paměti pomocí pole. Se specifikací proměnných tohoto typu a operace s nimi budou popsány detailně v následujících kapitolách. Jejich použití je však snadné, jak je zřejmé z následujícího programu. Je nutno si však uvědomit, že se tím zvyšuje spotřeba paměti o n-1 položek.

```
{výpočet střední hodnoty a rozptylu pro malé soubory }
uses crt;
var N,I:integer;
    X:array [1..200] of real;
    X1,S2:real;
begin
clrscr;
writeln('zadej počet hodnot N a hodnoty X(i), i=1 až N ');
read(N);
{čteni hodnot X[i], i=1 až N}
for I:=1 to N do
   read(X[I]);
{výpočet střední hodnoty}
```

```
  X1:=0;
  for I:=1 to N do
    X1:=X1+X[I];
  X1:=X1/N;
{výpočet rozptylu}
  S2:=0;
  for I:=1 to N do
    S2:=S2+(X[I]-X1)*(X[I]-X1);
  S2:=S2/(N-1);
{tisk výsledných hodnot}
writeln('střední hodnota: ',X1:10:2,' rozptyl: ',S2:10:2);
end.
```

Existuje ještě další možnost, jak snížit spotřebu paměti a neprovádět čtení hodnot x_i, i=1 až n dvakrát. Lze nejdříve vypočítat $\sum x_i$ a $\sum x_i^2$ a pak určit střední hodnotu a rozptyl pomocí vztahu [2]

$$\bar{x} = \sum_{i=1}^{n} x_i / n$$

$$s^2 = \frac{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}{n - 1}$$

```
{výpočet střední hodnoty a rozptylu současně}
uses crt;
var N,I:integer;
  X,X1,S2:real;
begin
clrscr;
writeln('zadej počet hodnot N a hodnoty X(i), i=1 až N ');
read(N);
X1:=0;
S2:=0;
for I:=1 to N do
begin
  read(X);
  X1:=X1+X;
  S2:=S2+X*X;
 end;
X1:=X1/N;
S2:=(S2-N*X1*X1)/(N-1);
writeln('střední hodnota: ',X1:10:2,' rozptyl: ',S2:10:2);
end.
```

```
5
1 2 5 8 12
     5.60     20.30

5
1 2 5 8 12
     5.60     20.30
```

[2] Tento vzorec je málo znám, že je správný dokazuje porovnání výsledů předchozího a tohoto programu.

3. PROGRAMOVACÍ JAZYK TURBO PASCAL 7.0

3.1 Základní pojmy

Poznámka: Brysovým typem písma jsou v určitých místech vždy označeny příklady zápisu. Tento způsob by měl umožnit rychlejší orientaci pro daný tématický okruh.U těchto příkladů je vždy nutno doplnit "atd."

3.1.1 Lexikální elementy

Nejmenšími jednotkami zdrojového programu jsou lexikální elementy, které dělíme na

- speciální symboly

☞ **+ - _ <**

a klíčová, rezervovaná slova,která jsou rovněž speciální symboly

☞ **begin end array**

- identifikátory

☞ **SUMA X**

- číselné konstanty

☞ **1 1.0 1E1**

- řetězcové a znakové konstanty

☞ **'PASCAL' 'A' '1'**

- návěští.

☞ **KONEC: ZNOVU**:

- komentáře

☞ **{ to je ono }**

V případě, že dva sousední lexikální elementy jsou klíčové slovo, identifikátor, konstanta či číslo, musí být odděleny alespoň jedním separátorem, tj. mezerou, koncem řádku (eoln) nebo komentářem. Poznamenejme,že i některé speciální symboly slouží jako oddělovač, například ;.

Abeceda jazyka

Konstrukce lexikálních elementů je vytvářena z přípustných symbolů, které tvoří abecedu jazyka Pascal. Abecedu tvoří znaky

A až Z a a až z anglické abecedy pro písmena
0 až 9 pro decimální číslice
0 až 9 a A až F pro hexadecimální číslice

Speciální symboly

☞ **+ ***

Kromě uvedených základních symbolů se používají speciální symboly
jednoznakové + - * / = < > [ ] . , ( ) : ; @ ^ { } $ # _
dvouznakové <= >= <> := ..

3.1.2 Klíčová - Rezervovaná slova

☞ **var**

Jsou to slova, která nesmí být použita v jiném významu. Slouží k implementaci jednotlivých příkazů, definic, jako operátory, atd.

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

Tabulka 1: Klíčová - rezervova slova

3.1.3 Identifikátory

☞ **SUMA**

Slouží k označení konstant, typů, proměnných, funkcí, procedur, programů, programových jednotek, návěští a položek záznamu. Identifikátor může být tvořen písmeny, číslicemi a _ (podtržítko), prvním znakem musí být písmeno a délka je omezena na 63 znaků. Jako identifikátor nesmí být použito rezervované slovo. Velká a malá písmena nejsou rozlišována.
Příklad:

X1 SUMA X_1

2A chybný identifikátor, začíná číslicí.[3]

Každý identifikátor musí být vždy nejdříve definován a teprve potom může být použít. Znamená to tedy, že se musí každý identifikátor vyskytnout nejdříve v deklaraci, kde definujeme význam identifikátoru. Vhodnou volbou identifikátorů se snažíme zvýšit čitelnost a srozumitelnost programu. Identifikátory tedy volíme tak, aby co nejvíce vystihovaly význam toho, co vyjadřují, např. pro sumu volíme SUMA , pro maximum MAX atd.

Význam některých identifikátorů je předdefinován. Jedná se o standardní procedury a funkce, které budou postupně popsány v dalších kapitolách.

Platnost definic standardních identifikátorů procedur a funkcí lze však zastínit a dát jim jiný význam v deklarační části.

3.1.4 Konstanty

Číselné konstanty

☞ **1.1 1**

Občas je zapotřebí v programu použít hodnotu, která je známá již při psaní programu, tedy před jeho provedením. Je tedy možné uvádět příslušné hodnoty přímo v místě použití. Přímý zápis hodnoty v programu se nazývá literál. Přímou konstantou může být číselná konstanta, znak či konstanta řatězec.

Číselné konstanty mohou být decimální celočíselné nebo hexadecimální celočíselné (označené $) či decimální reálné. U reálných čísel je možno použít zápis s desetinnou tečkou nebo semilogaritmický

[3] Možnost omylu 2*A

(písmeno E nahrazuje "krát 10 na"). Celočíselné a reálné konstantě (mantise i exponentu) může předcházet znaménko + či -.

Příklad:

1 -1 +1	číslo typu celočíselného, decimálního
$1 $FFF	číslo typu celočíselného, hexadecimálního
1.0 +1.0 -1.0	číslo typu reálného, decimálního
1E0 -1E-1 +1E+1	číslo typu reálného, decimálního v semilogaritmickém tvaru

Řetězcové konstanty

☞ **'to je řetězec'**

Jsou posloupnosti znaků zapsaných mezi apostrofy. Jedna řetězová konstanta musí být zapsána na jednom řádku, chceme-li zapsat do řetězce znaků apostrof, pak jeho zápis musíme zdvojit. Odpovídající znak z ASCII kódu je možno zapsat číslem z intervalu 0 až 255, kterému předchází symbol #. Tento zápis musí být proveden mimo apostrofy.

Příklad:
'PASCAL' nebo #80#65#83#67#65#76

Znaková konstanta

☞ **'A'**

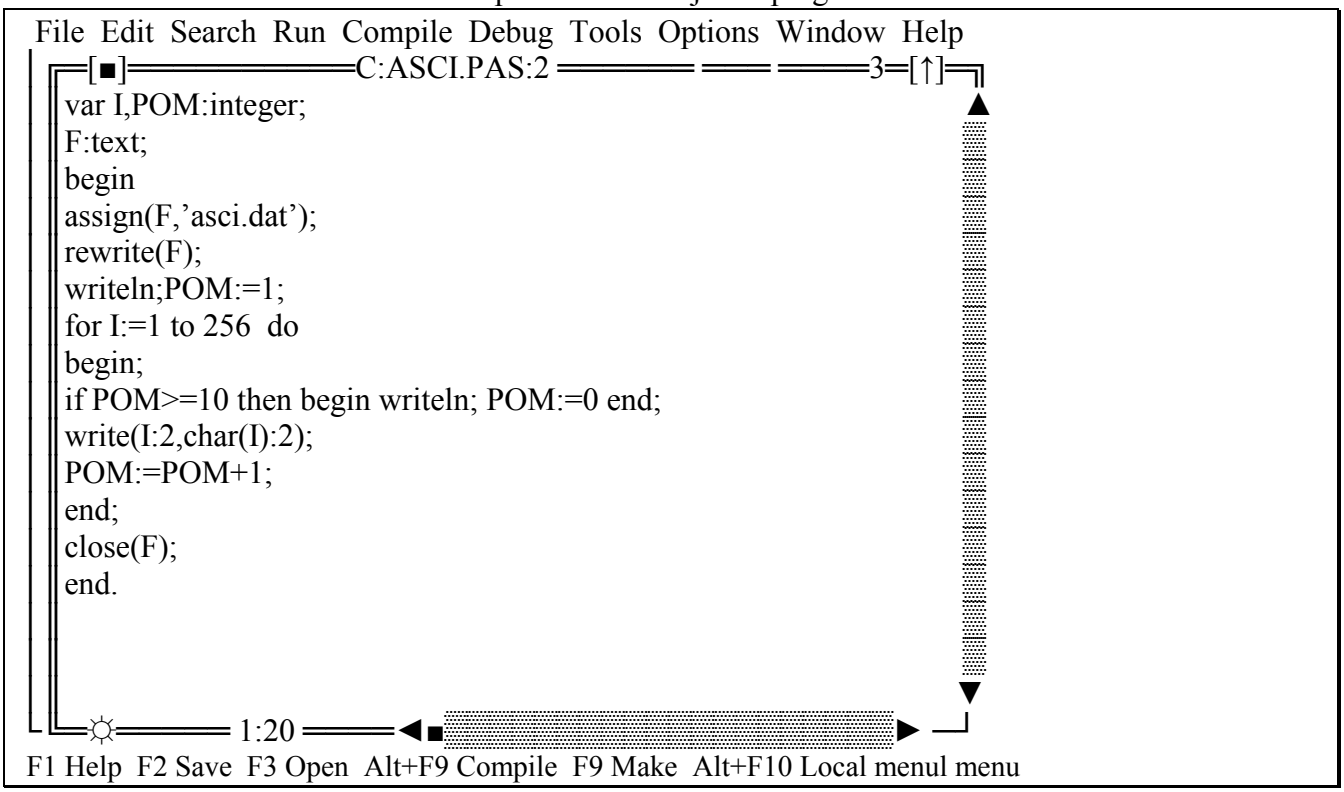
Znakovou konstantou je znak zapsaný mezi apostrofy nebo odpovídajícím kódem ASCII, kterému předchází symbol #.

Příklad:
'A' nebo #65 nebo #$41

	0	1	2	3	4	5	6	7	8	9
1	1f			cr						
2										
3				!	“	#	$	%	&	´
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	/	]	^	_	\	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Tabulka 2: Část ASCII kód (American Standard Cod for Information Interchange).

Zbrazit celou tabulku tohoto kódu lze pomocí následujícího programu:



3.1.5 Návěští

☞ **KONEC :**

Umožňuje přechod na daný příkaz v libovolném místě programu pomocí příkazu skoku (**goto** návěští). Návěštím může být číslo v rozsahu 0 do 9999 nebo identifikátor. Příkaz, na který je proveden skok, je opatřen odpovídajícím návěštím a dvojtečkou. Je však nutno zdůraznit, že využívání nepodmíněného skoku **goto** je nevhodné. Pascal jako strukturovaný programovací jazyk poskytuje lepší možnosti pro vyřešení různého větvení v programu. Použití příkazu **goto** činí ve většině případů program méně čitelný.

Příklad: ☞ **ANO**

```
……..
 if A<=0 then begin
                    X:=1;
                    Y:=2
                end
          else begin
                    X:=10;
                    Y:=20
                end;
 writeln(X,Y);
 ……
```

☞ **NE**

```
   …….
if A>0 then goto L1;
X:=1;
Y:=2;
goto L2;
L1:X:=10;
Y:=20;
L2:writeln(X,Y);
……..
```

3.1.6 Komentář

☞ **{ Komentář }**

Chceme-li do programu zařadit text, který má sloužit pouze k popisu významu objektů a konstrukce programu, případně k jejich vztahu k řešenému problému, možno jej vysvětlit pomocí komentáře. Komentář nemá operační účinek, jeho vypuštěním se realizace programu nezmění. Zařazení komentáře se doporučuje, zvyšuje čitelnost a přehlednost programu. Komentář se zapisuje do závorek { }, prvním znakem komentáře nesmí být $ (vyhrazeno pro direktivu překladače).
Příklad:{výpočet plochy} nebo (* komentář *). Alternativně lze místo složených závorek použít i dvojice symbolů (* a *).

3.2 Organizace programu - programové bloky

Program v Pascalu se dělí na programové bloky a do bloku mohou být vnořeny další bloky, procedury a funkce které se liší pouze hlavičkou. Další struktura je u všech bloků stejná. Programové bloky se skládají z deklarační části a příkazové části. Deklarační část může obsahovat

- uses (standardní a uživatelské moduly)
☞ **uses Crt;**
- deklarace návěští
☞ **label KONEC;**
- definice konstant
☞ **const MIN=1;**
- deklarace konstant s udaným typem.
☞ **const MAX : integer = 10;**
- definice typů
☞ **type XT=1;**
- definice proměnných
☞ **var x:1;**
- deklarace procedur a funkcí
☞ **procedure P;**
begin.....end

Klíčové slovo **uses** uvádí seznam jednotek, které má překladač spojit s překládaným programem.

3.2.1 Deklarace návěští

☞ **Label KONEC;**

Před použitím návěští v příkazové části je třeba definovat toto návěští v deklarační části, např. **label** 10, 20, 50, KONEC;

Příkaz skoku se používá k přímému předání řízení z jednoho místa programu na jiné. Použití tohoto příkazu se nedoporučuje, neboť se ve většině případů použití sníží čitelnost programu. Použití skoku lze přirovnat ke čtení knihy, kdy se neustále odkazuje na přečtení kapitoly umístěné před nebo za čtenou kapitolou. Pokud již příkaz použijeme, musí platit, že

- příkaz skoku nepředává řízení dovnitř strukturovaného příkazu
- v úseku deklarace návěští musí být deklarována právě ta návěští, která jsou použita jako označení příkazů v příkazové části bloku.

3.2.2 Definice konstant

☞ **1.0 const A=1 const B:integer=1**

Jednou z definic konstant je, že je datovým objektem, jehož hodnotu nelze během provádění měnit. U Pascalu je však lépe použít obecnější definici, že konstanta je symbol, který zastupuje určitou konstantu. U Pascalu rozlišujeme tři druhy konstant - literály (přímé konstanty), definovanou konstantu a definovanou konstantu s udáním typu.

- přímá konstanta se definuje zápisem konstanty do výrazu, např. 3 3.1415 -285E3

- [4]definovanou konstantu definujeme zápisem klíčového slova **const** a seznamem konstant s udanou hodnotou, např.

[4] Využití této konstanty je například při změně formátu tištěné stránky, tedy například pro počet řádků na stránce.

```
const MIN = 10;
      MAX = 100;
      ROZSAH = (MAX-MIN) div 2;
```

U definované konstanty MIN, MAX a ROZSAH nelze již ve výpočtové části programu změnit jejich hodnotu, změna může být provedena pouze v deklarační části.

- konstanty s udaným typem definujeme následovně

```
const MIN: integer = 10;
 MAX: integer = 100;
```

U těchto konstant lze v programu měnit jejich obsah, jsou tedy využitelné jako identifikátory proměnných se zadanou počáteční hodnotou. Tato konstanta nemůže být použita při definici jiné konstanty či typu. Chybný je např. zápis

```
const MIN: integer = 10;
      MAX: integer = 100;
type X = array [MIN..MAX] of real; {chybný zápis}
```

Příklad deklarační části:

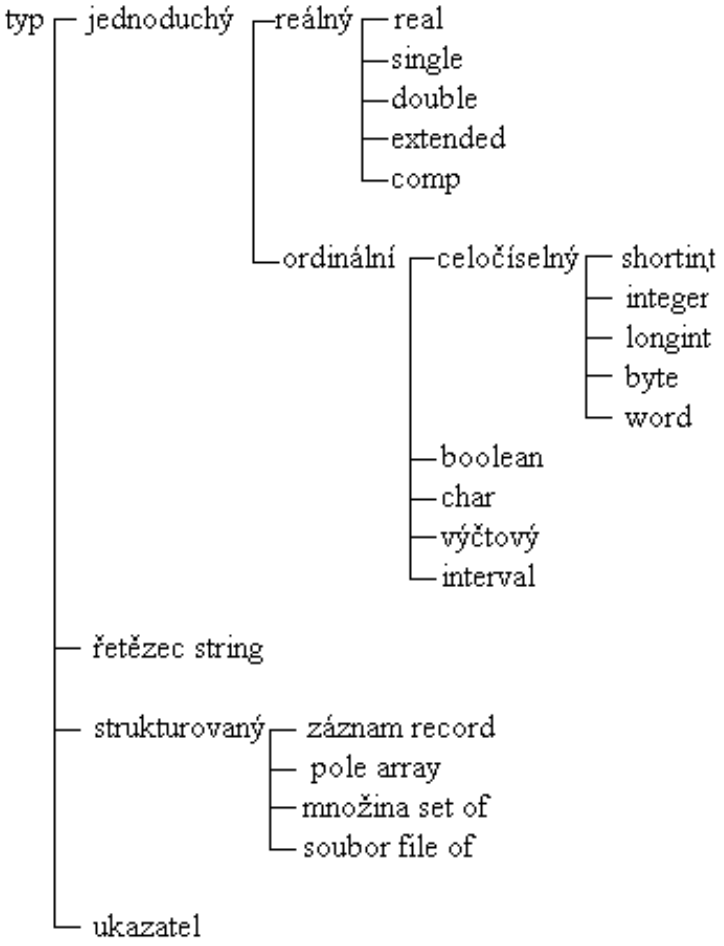
```
Uses Crt;
const MIN_HODNOTA=-100;
      MAX_HODNOTA=100;{nebo =-MIN_HODNOTA}
      ZNAK='X';
      RETEZ='ABCDE';
      MIN1=10;
label KONEC,ZACATEK,L100;
```

V uvedeném příkladě je deklarován standardní modul Crt, definovány číselné konstanty MIN_HODNOTA, MAX_HODNOTA, znaková konstanta ZNAK, řetězcová konstanta 'ABCD'. Dále jsou definovány číselné konstanty s uvedeným typem MIN1 a MAX1. Pro příkaz skoku lze použít návěstí KONEC, ZACATEK a L100.

3.2.3 Definice typů

Údajové typy

Akce programu se uskutečňuje na datových objektech, které mohou nabývat různé hodnoty a tyto hodnoty mohou být různého typu. Každá konstanta, proměnná, výraz nebo funkce může nabývat jen hodnotu jednoho typu a tento údaj o typu vede k následujícímu rozdělení údajového typu



- jednoduché jsou typy reálný, celočíselný, *Boolean* a *char*. K jejich definici používáme odpovídající standardní identifikátory typu.
- deklaraci typu řetězec provádíme pomocí rezervovaného slova a podobně, pomocí rezervovaných slov si programátor definuje strukturované typy.
- jednoduché typy interval a výčtový typ si musí definovat programátor sám.

U ordinálních typů existuje vždy následník a předchůdce (kromě poslední a první hodnoty) a jeho ordinální číslo.

3.3 Deklarace proměnných a definice konstant

☞ **var A: integer;**

Proměnná je datový objekt, jehož hodnotu lze v průběhu výpočtu měnit, na rozdíl od konstanty typu literál a definované konstanty. Proměnné slouží k ukládání vstupních a výstupních dat, různých výsledků výpočtu, atd. Každá proměnná je určitého typu, čímž jsou specifikovány přípustné hodnoty a reprezentace v paměti počítače. Abychom mohli v programu pracovat s proměnnou, je nejdříve nutno proměnnou deklarovat. Pomocí deklarace se v paměti vymezí místo pro ukládání hodnot určitého typu a velikost nutná k uložení dané hodnoty závisí na určeném typu. Na toto místo v paměti se odvoláváme pomocí idenfifikátoru proměnné, který nám zastupuje adresu místa či úseku vymezeného v paměti pro danou hodnotu.

Deklaraci proměnné provádíme pomocí klíčového slova var.

3.3.1 Celočíselný datový typ

☞ **var I:integer;**

Tento typ je používán pro celá čísla s různou délkou zobrazení.

typ	rozsah	délka
byte	0..255	1 byte
shortint	-128..127	1
word	0..65535	2
integer	-32 768..32 767	2
longint	-214743648..214744483647	4

Dále je povoleno použití hodnot typu integer v hexadecimálním tvaru vložením znaku $ před vlastním vyjádřením hodnoty.

Příklad:
var A, ALFA: *integer;*
DELT,A111:*longint;*

Definice konstanty byla uvedena v předchozí kapitole.

3.3.2 Reálný datový typ

☞ **var x:real;**

Používá se pro vyjádření reálných hodnot s různou délkou zobrazení

typ	rozsah	platné číslo	délka
real	$2.9*10\text{-}^{39}$ až $1.7*10^{38}$	11-12	6 byte
single	$1.5*10^{-45}$ až $3.4*10^{38}$	7-8	4
double	$5.0*10^{-324}$ až $1.7*10^{308}$	14-16	8
extended	$6.0*10^{-4932}$ až $1.1*10^{4932}$	19-20	10
comp	$-2^{63}+1$ až $2^{63}-1$	19-20	8

Poznámka:

Typ *comp* umožňuje zpracovávat pouze celá čísla do maximálního počtu 19 míst. Typ *single, double, exteded* a *comp* je možno použít buď při přítomnosti matematického koprocesoru 8087, nebo při jeho programové emulaci (nastavuje se pomocí programových direktiv {$N+} a {$E+}).

Příklad:
var ALFA, BETA: *real*;
Definice nepřímé konstanty má tvar
const X = 1E5/45.1; {definovaná konstanta}
T : *real* = 5.1; {konstanta s udaným typem}

3.3.3 Datový typ znak

☞ **var z:char;**

Je to datový typ o velikosti 1 byte a uchovává jeden znak. Zapisuje se v apostrofech nebo je uveden číslem (dekadicky nebo hexidecimálně), které odpovídá ASCII reprezentaci znaku. Například 'A' je totéž, jako #65 či #$41.

Příklad:
var ZN: *char;*
Definice nepřímé konstanty má tvar
const T = 'V';
T1: *char* = 'z';

3.3.4 Datový typ řetězec

☞ **var y:string;**

Následuje-li několik jednotlivých znaků bezprostředně za sebou, vytvoříme tzv. řetězec, který může obsahovat maximálně 255 znaků. V paměti je řetězec uložen na tolika bytech, kolik znaků řetězec obsahuje, plus jedna. V nultém bytu řetězce je uložena skutečná délka řetězce, ale jako ASCII kód, např. pro počet znaků 5 je na nultém byte uložena hodnota #5. Chceme-li použít tuto hodnotu jako číselnou, nutno zapsat *ord*(#5), nebo použít standardní funkcí *length* (argument). Když deklarujeme proměnnou typu řetězec na určitou délku a skutečný obsah řetězce je menší, paměť zabírají pouze naplněné znaky. Proto můžeme v programu provést deklaraci typu řetězec buď s uvedením maximální délky nebo bez uvedené délky. S řetězci lze pracovat jako s polem znaků, nultý byte obsahuje údaj, vyjadřující skutečnou délku řetězce.

Příklad:
```
var X:string [5];
begin
  X:='123';]
  writeln(ord(X[0]),X[1],X[2],X[3])
end.
```

Tento program zobrazí výsledky ve tvaru (funkce *ord*(X[0]) udává skutečnou délku řetězce):
3123

Příklad:
```
var XYZ1: string [255];      {nebo pouze string;}
      A,B : string [5];
      ZN : string [1];        { nebo char;}
```

Definice nepřímé konstanty má tvar
```
const ZPA = 'CHYBA';
    ER = 'POZOR: ' + ZPA
    XZ : string [4] = 'VSST';
```

3.3.5 Datový typ Boolean

☞ **var B:boolean;**

Tento typ může nabývat hodnotu *true* nebo *false* a deklarujeme jej například takto

var T1, T2: *boolean;*
Definice nepřímé konstanty
const F1 = *false*;
FA: *boolean* = *false*;

3.3.6 Typy definované programátorem

Kromě již uvedených standardních typů existují prostředky na definování jednoduchých typů, strukturovaných typů a typu ukazatel. Tyto si však definuje programátor sám, viz následující kapitoly.

Definice typů v programu

☞ **type A = real ;**

Použít deklaraci typu lze například následovně
type T = 1..100; {úsek definice typu interval}
var A,B:T; {úsek deklarace proměnných daného typu}
Lze použít i následující deklaraci
type TI = *integer*; {zavedení identifikátoru typu TI, označující}
var I1,I2,I3:TI; {totožný typ jako standardní identifikátor *integer*}

Výčtový typ

☞ **var BARVA:(CERVENY, CERNY)**

Při psaní programu se můžeme setkat s potřebou zobrazit určitý údaj, který nabývá pouze malého počtu různých hodnot. Například hodnoty dnů v týdnu je možno zakódovat pomocí celých čísel 0 až 6, avšak lepší přehled dává výčtový typ. Například deklarací

var DEN:(PONDELI,UTERY,STREDA,CTVRTEK,PATEK,SOBOTA,NEDELE)

je zaveden identifikátor DEN a identifikátory konstant PONDELI,UTERY až NEDELE. Každý výčtový typ je ordinálním typem.

Typy interval

☞ **var I:1..3**

Definování typu pomocí intervalu je založené na využití již definovaného ordinálního typu tím, že množinu hodnot omezíme na hodnoty určitého intervalu, jejichž hranice uvádíme v definici. Interval je tedy typ, který specifikuje souvislou podmnožinu hodnot nějakého ordinálního typu. Specifikovat je nutno první a poslední hodnotu intervalu, obě hodnoty uvedené v popisu musí být stejného typu a první hodnota musí být menší nebo rovna druhé hodnotě. Stanovením intervalu se omezuje množina přípustných hodnot dané proměnné např.

```
var  PISMENO_V: 'A' ..'Z';              {interval velkých písmen A,B,C,až Z}
     PISMENO_M: 'a' ..'z';              {interval malých písmen a,b,c,až z}
     INDEX: 1..100;                     {interval celých čísel 1 až 100}
     PRAC_DEN:PONDELI..PATEK            {interval z předem definovaného výčtového typu}
     MAX_INT:-maxint..maxint[5]         {interval celých čísel s využitím standardního
                                         identifikátoru konstanty maxint}
```

[5] *Maxint* určí maximální číslo typu integer tj. 32767 (16 bitů=2 byte). Minimální se určí jako záporné číslo maximálního čísla. Ve skutečnosti můžeme použít minimální číslo -32 768, pozor, vzhledem k zobrazení integer čísla je záporné číslo o 1 větší.

Kladné maximální číslo $0*2^{15} + 1*2^{14} +\ldots+1*2^5 +1*2^4 +1*2^3 +1*2^2 +1*2^1 +1*2^0$

Záporné maximální číslo $-1*2^{15} + 0*2^{14} +\ldots+0*2^5 +0*2^4 +0*2^3 +0*2^2 +0*2^1 +0*2^0$

číslo –1 $-1*2^{15} + 1*2^{14} +\ldots+1*2^5 +1*2^4 +1*2^3 +1*2^2 +1*2^1 +1*2^0$

číslo +1 $0*2^{15} + 0*2^{14} +\ldots+0*2^5 +0*2^4 +0*2^3 +0*2^2 +0*2^1 +1*2^0$

3.4 Příkazová část

☞
```
begin
A:= 1
end.
```

Příkazová část se skládá z příkazů, které však mohou obsahovat výrazy. Pod pojmem výraz rozumíme předpis pro získání hodnoty. Výraz se skládá z operátorů a operandů. Jako operand může být použita konstanta, proměnná a zápis funkce. Operátory dělíme na aritmetické, relační, logické, bitové, řetězcové, adresové a množinové. Dále operátory dělíme na binární (obsahují dva operandy) a unární (s jedním operandem). Vyhodnocování výrazu probíhá podle priorit, při shodné prioritě binárních operátorů zleva doprava. Prioritu lze upravit pomocí kulatých závorek, kdy se nejdříve vyhodnotí výraz v nejvnitřejší závorce, opět zleva doprava.

Priorita operátorů od nejvyššího k nejnižšímu

not
* / **div mod and shl shr**
+ **- or xor**
= <> >= <= > < **in**

3.4.1 Aritmetické operátory

☞ **A + A**

Následující tabulka ukazuje možnosti pro operandy typu integer a real a typ výsledku.

P	Q	P + * -	P/Q	P **mod** či **div** Q
integer	integer	integer	real	integer
integer	real	real	real	-
real	integer	real	real	-
real	real	real	real	-

kde + značí sčítání - odčítání, * násobení, / dělení,
div celočíselné dělení, **mod** zbytek po celočíselném dělení.

Příklad:
div a **mod** lze použít následovně

7 **div** 6 = 1
1 **div** 2 = 0

4 **div** 2 = 2
7 **mod** 6 = 1
1 **mod** 2 = 1
4 **mod** 2 = 0

Příklad:

```
var I,J,K: integer;
   A,B,C: real;
{předpokládejme I:=1; J:=10; K:=100; A:=5.0; B:=10.0; C:=20.0}
```

výraz

$a+\frac{b}{c}+a$ čili A + B/C + A dává výsledek 10.5 typu *real*

$\frac{a+b}{c+a}$ čili (A+B)/(C+A) dává výsledek 0.6 typu *real*

$\frac{1}{2}a$ čili 1/2 * A dává výsledek 2.5 typu *real*

1 **div** 2*A dává výsledek 0.0 typu *real*

Jak provedeme mocnění ?

U většíny programovacích jazyků je mocnění definováno např. pomocí zápisu a^b = A ↑ B (Basic) či A ** B (Fortran). V Pascalu je definována pouze funkce pro A^2 = SQR (A) (viz. standardní funkce). Pro A^n, kde n je buď velké číslo (větší než 2) typu integer nebo číslo typu real, je nutno použít zápis

$$e^{n(\ln(A))}$$

Když využijeme standardní funkce *exp* a *ln*, viz. kapitola standardní aritmetické funkce, pak výraz má tvar

$$\exp(N * \ln(A))$$

3.4.2 Logické operátory

☞ **TRUE or FALSE**

Vyhodnocení logických operací **and** (logický součin), **or** (logický součet), **xor** (exklusivní součet) je dáno následující tabulkou. Uvedené operátory umožňují kombinovat logické výrazy s proměnnými typu boolean a relačními výrazy.

P	Q	P **and** Q	P **or** Q	P **xor** Q
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

Logická negace:

not *true* = *false*
not *false* = *true*

Příklad:
var A1, B1, C1: *boolean;* {A1: = *true;* B1: = *false;* C1: = *false;*}
výraz A1 **and** B1 **or not** C1 dává výsledek *true*
A1 **or** (3>1) dává výsledek *true*
A1 **or** 3>l je chybný zápis (je chápán jako (A1 or 3) > 1)

3.4.3 Bitové operátory

☞ **1 and 0**

Logickým operátorům se podobají bitově orientované operátory, nejsou však totožné. Pomocí bitově orientovaných operátorů lze provádět následující operace na úrovni bitů:

not	provádí logický doplněk na každém bitu (změní 0 na 1 a 1 na 0)
and	provádí logický součin
or	provádí logický součet
xor	provádí exkluzivní součet na každém příslušném páru bitů
shl, shr	provádí posuv bitů vlevo resp. vpravo a bity vpravo, resp. vlevo doplňuje 0

Bitově orientované operace lze provádět na operandech typu integer, výsledek operace je opět typu integer.

Příklad:
1 **shl** 2 dává výsledek 4 neboť $(0001)_2$ se změní na $(0100)_2$ [6]
2 **shr** 1 dává výsledek 1 neboť $(0010)_2$ se změní na $(0001)_2$
6 **and** 5 dává výsledek 4 neboť $(0110)_2$ and $(0101)_2$ je $(0100)_2$
6 **or** 5 dává výsledek 7 neboť $(0110)_2$ or $(0101)_2$ je $(0111)_2$

3.4.4 Relační operátory

A > = 1

Vzájemné vztahy mezi hodnotami ordinálních typů a řetězci lze vyjádřit pomocí relačních operátorů.

>	větší
>=	větší nebo rovno
<	menší
<=	menší nebo rovno
=	rovno
<>	nerovno

Jako operandy je možno použít konstantu, proměnnou a výraz. Porovnat lze hodnoty stejného typu, ale porovnávat lze rovněž hodnoty *integer* a *real*, **string** a *char*. Při porovnání **string** a *char* je kriteriem uspořádání znaků v ASCII kódu. U ordinálních typů jsou rozhodující pro porovnání ordinální hodnoty, tedy např. u výčtového typu je rozhodující pořadí identifikátorů v popisu typu.

3.4.5 Řetězcové operátory

☞ **'A' + 'XYZ'**

Při práci s řetězci lze použít operátor +, výsledkem je zřetězení dvou řetězců. Ke zřetězení lze použít libovolný řetězový typ (včetně *char*), je-li výsledný řetězec delší než 255 znaků, nadbytečné znaky se ignorují, například 'PRAHA'+ 'LIBEREC' dá výsledek 'PRAHALIBEREC'

---

[6] $(0001)_2 = +0*2^3 +0*2^2 +0*2^1 +1*2^0 = 1_{10}$
$(0100)_2 = +0*2^3 +1*2^2 +0*2^1 +0*2^0 = 4_{10}$
index 2 resp. 10 znamená dvojkovou resp. desítkovou soustavu.

3.5 Standardní funkce a procedury

Operandem ve výrazu může být i zápis funkce, tj. předpis pro výpočet hodnoty zvolené funkce. Tento předpis si definuje programátor, avšak pro často používané funkce je tento předpis již předdefinován pomocí standardních funkcí (v modulu Unit SYSTEM).

3.5.1 Standardní funkce

☞ **sin (ALFA)**

Standardní aritmetické funkce

abs (X)	absolutní hodnota argumentu X
arctan(X)	arkustagent argumentu X
cos (X)	cosinus argumentu X v radiánech
exp (X)	exponencionála argumentu X
frac (X)	desetinná část argumentu (*frac* (123.456)=0.456)
int (X)	celá část argumentu (*int* (123.456) = 123.0)
ln (X)	logaritmus argumentu s X
pi	3.1415926535897932385 - pozor, jediná standardní funkce nemá závorky
sin (X)	sinus argumentu X v radiánech
sqr (X)	druhá mocnina argumentu X
sqrt (X)	druhá odmocnina z argumentu X

Poznámka:

Pro *abs*, *sqr* s argumentem typu *integer* je výsledek typu *integer* a s argumentem typu *real* je výsledk typu *real*.

Pro *sin, cos, arctan, ln, exp, sqrt* s argumentem typu *integer* či *real* je výsledek typu *real*

Transformační funkce:

trunc (X)	celočíselná část reálné hodnoty X
round (X)	zaokrouhlená reálná hodnota X na nejbližší celé číslo, t.j. celá část výrazu X ± 0,5, + pro X > 0, - pro X < 0 [7]

Příklad:

trunc (1.1) = 1	*round* (1.1) = 1
trunc (1.9) = 1	*round* (1.9) = 2
trunc (-1.1)= -1	*round* (-1.4)= -1
trunc (-1.9)= -1	*round* (-1.9)= -2

Ordinální funkce:

ord (X)	ordinální číslo argumentu
pred (X)	předchůdce v ordinálním typu
succ (X)	následník v ordinálním typu
chr (X)	znak, jehož ordinální číslo je X
high(X)	nejvyšší hodnota z rozsahu argumentu
low(X)	nejnižší hodnota z rozsahu argumentu

Pro *High*(X), *Low*(X) argument X musí být ordinálního typu, pole nebo řetězec. U ordinálního typu *High* vrací nejvyšší hodnotu a *Low* nejnižší hodnotu z rozsahu argumentu. U pole je vrácena nejvyšší a nejnižší hodnota indexu pole a pro řetězec se vrací délka řetězce a nula, viz následující příklad.

[7] Zaokrouhlit lze nejen desetiny, ale také setiny, tisíciny atd., například zápisem X ± 0,05, + pro X > 0, - pro X < 0.

Příklad:

```
var A: array [1..5] of integer;
   C: string [5];
   C1:string;
begin
   writeln (High (A), Low (A));
   writeln (High (C), Low (C));
   writeln (High (C1), Low (C1))
end.
```

```
Výsledek:
5   1
5   0
255   0
```

Pro *ord* platí, že *ord* (X) je pořadové číslo znaku X v dané množině znaků, přičemž pořadové číslo začíná nulou.

Ordinálním číslem hodnoty celočíselného typu je dané číslo (*ord* (1)=1), pro *char* je dáno kódem *ord* ('1') = 49, *ord* ('A')=65 či *ord* (#65)=65 v ASCII kódu). Pro *boolean* je *ord* (*false*) = 0, *ord* (*true*) = 1.

Pro *chr* platí, že *chr*(I) je znak,kterého ordinální hodnotou je I, například pro char(65) obdržíme A.

Vzájemný vztah funkcí *ord* a *char* je

chr (*ord* (X)) = X, *ord* (*chr* (I)) = I.

Pro funkce *succ* a *pred* u celočíselných typů platí
succ (I) odpovídá výrazu I + 1,
pred (I) odpovídá výrazu I - 1.
Pro typ *char* platí
succ ('B') = 'C'
pred ('B') = 'A'
Pro typ boolean platí *pred* (*true*) = *false*, *succ* (*false*) = *true.*

Predikáty:

Jsou to funkce, jejichž výsledkem je typ *boolean*.

odd (X)	*true*, je-li X liché celé číslo
eoln (X)	*true*, je-li při čtení nalezen konec řádku.

Smíšené funkce:

hi (X)	hodnotou je horní část argumentu X (t.j.osm horních bitů argumentu X)
lo (X)	hodnotou je dolní část argumentu X (tj. osm dolních bitů argumentu X)
random (X)	generování pseudonáhodných čísel v rozsahu 0 < = generované číslo < X. Není-li X zadáno, pak generuji čísla v rozsahu 0 < = generované číslo < 1 (*random* (1)). Argument musí být i*nteger* nebo *word*. Nastavení generátoru na počátek možno provést pomocí procedury randomize, nebo dosazením žádané hodnoty do proměnné RandSeed.

Poznamenejme, že funkci *hi* lze nahradit pomocí funkce *shr*, tedy jako X *shr* 8 a funkci *lo* jako ((X *shl* 8) *shr* 8).

Příklad:

```
var X:word;
begin
X:=$FFFF; {t.j. 65 535}
writeln(Lo(X));
writeln(Hi(X));
end.
```

```
Výsledek:
255
255
```

Řetězcové funkce:

length (S)	vrací délku řetězce uloženou v argumentu S (je tedy obdobou *ord* (s[0]).
copy (S,I1,I2)	vrací řetězec o délce I2 počínaje pozicí I1 z řetězce v S.
pos (S1, S2)	vyhledá pozici prvního znaku řetězce S1 v řetězci S2.
Concat(S1,S2,.)	vrací spojené řetězce uložené na S1, S2, atd. Je tedy obdobou S1 + S2 + ...

Příklady pro řetězové funkce

```
var S:string;
begin
  S:='12345';
  writeln(length(S),ord(S[0]));
end.
```

```
Výsledek:
55
```

```
var S:string;
begin
  S:='12345';
  writeln(copy(S,2,3));
end.
```

```
Výsledek:
234
```

```
var S:string;
begin
  S:='12345';
  writeln(pos('3',S));
end.
```

```
Výsledek:
3
```

```
var S:string;
begin
  S:='12345';
  writeln(concat('ABC',S,' ','ABC'+S));
end.
```

```
Výsledek:
ABC12345          ABC12345
```

Příklad:
Pro zadaný datum ve tvaru dd.mm.rrrr proveďme jeho reverzi, tedy do tvaru rrrr.mm.dd.

```
var TZ,TR:string[10];
begin
read(TZ);
TR:=copy(TZ,7,4);
TR:=TR+'.';
TR:=TR+copy(TZ,4,2);
TR:=TR+'.';
TR:=TR+copy(TZ,1,2);
writeln(TR);
end.
```

DATA:

```
01.12.1994
```

Výsledky:

```
1994.12.01
```

3.5.2 Standardní procedury

☞ **delete (S, 3, 3)**

Příkazy pro vykonání určitého algoritmu mohou být sestaveny jako samostatný blok (podprogram). Některé procedury, tak jako funkce jsou předdefinovány a procedura se volá jen svým jménem s uvedením použitých skutečných parametrů.

Řídící procedury:

break používá se uvnitř cyklu for, while repeat pro ukončení cyklu. Je ekvivalentní příkazu skoku na koncový příkaz cyklu.
exit v hlavním programu způsobí ukončení programu, v uživatelské proceduře ukončení této procedury a návrat do programu, kde byla volána
halt ukončení výpočtu a návrat do operačního systému - integrovaného prostředí Turbo Pascalu.
runerror obdoba *halt*, generuje se navíc chyba v běžném řádku.

Ordinální procedury:

dec (X,N) značí snížení hodnoty ordinálního typu X o N (N je typu *integer*). Není-li N uvedeno, chápe se jako N = 1. *Dec* (X) odpovídá X: = X - 1; *dec*(X,N) odpovídá X:=X-N.
inc (X,N) značí zvýšení hodnoty ordinálního typu X o N. Význam N je obdobný jako v proceduře *dec*, místo odečítání je uvažováno přičítání.

Řetězcové procedury:

delete (S,I1,I2) z řetězce S vyloučí I2 znaků počínaje znakem na pozici I1
insert (S1,S2,I) do řetězce S2 na pozici I připojí řetězec S1
str (X:I1:I2,S) převede celočíselné X resp X:I1 nebo real resp X:I1:I2 na řetězec S.Význam I2 je shodný s příkazem *write*
val (S,X,I) převede řetězec S na X,celočíselného typu nebo *real*. I obsahuje nulu, nebyl-li nalezen nepřípustný znak, jinak obsahuje pozici nedovoleného znaku.

Příklady pro řetězcové procedury:

```
var S:string;
begin
  S:='abcdef';
  delete(S,3,3);
  writeln(S);
end.
```

```
Výsledek:
abf
```

```
var S:string;
begin
  S:='abef';
  insert('cd',S,3);
  writeln(S);
end.
```

```
Výsledek:
abcdef
```

```
var S:string;
begin
  str(1234,S);
  writeln(S);
end.
```

```
Výsledek:
1234
```

```
var X:real;
   I:integer;
begin
   val('123.4',X,I);
   writeln(X);
   writeln(I);
end.
```

```
Výsledek:
 1.2340000000E+02
0
```

Procedury pro práci s adresáři:

ChDir (S)	aktivní adresář je přepsán na nový, zadaný v řetězci S, včetně aktivního disku - je-li zadán.
RmDir (S)	odstraní podadresář, definovaný cestou v řetězci S.
GetDir (X,S)	zapíše aktivní adresář na řetězec S. Disková jednotka je v X, kde disk A zapíšeme jako 0, B jako 1, C jako 2, atd.
MkDir(S)	vytvoří na disku podadresář. Cestu k novému podadresáři definujeme v řetězci S.

Poznámka:

Při zápisu konstant jsou povoleny pouze následující dříve uvedené standardní funkce *abs, chr, hi, lo, length, odd, ord, pred, round, succ, trunc* a rovněž funkce *ptr, sizeof a swap,* které nejsou ve skriptech popsány.

3.6 Příkazy

Příkazem nazýváme předpis pro vykonání určité algoritmické akce. Příkazy rozdělujeme na
jednoduché - ty, které v sobě neobsahují jiný příkaz
strukturované - obsahují v sobě další příkazy, které mohou být znovu strukturované.

Jednoduché příkazy se dělí na
- přiřazovací příkaz
☞ x: = 1
- příkaz pro volání procedur
☞ read (x)
- příkaz pro nepodmíněný skok
☞ goto L1
- prázdný příkaz
☞ begin end

Strukturované příkazy dělíme na
- složený příkaz
☞ begin x:=1; y:=1 end
- podmíněný příkaz **if** a **case**
☞ if I > 1 then x:=1
- příkaz cyklu **for, while, repeat**
☞ for I:=1 to N do příkaz
- příkaz **with**
☞ with OSOBA do příkaz

3.6.1 Přiřazovací příkazy

☞ **X:= sin(X) + 1.1 + X**

Provedením přiřazovacího příkazu se proměnné uvedené vlevo od := přiřadí hodnota výrazu uvedeného za :=. Každá proměnná ve výrazu na pravé straně příkazu musí mít definovanou hodnotu před provedením příkazu. Proměnná na levé straně příkazu je definována provedením příkazu. Této proměnné lze přiřadit jen takovou hodnotu, která patří do množiny přípustných hodnot specifikované typem proměnné.

Obecný tvar příkazu

P:=V

je správný pouze tehdy, když hodnota výrazu V je kompatibilní vzhledem k přiřazení s typem identifikátoru proměnné P. Požadovaný vztah mezi přiřazovanou hodnotou V a typem proměnné P je dán následující tabulkou

Typ P	Typ V
celočíselný	celočíselný
reálný	reálný nebo celočíselný
boolean	*boolean* nebo relační výraz
string	string nebo *char*
char	*char*

Příklad:
Uvažujme A typu reálná, I celočíselná, B *boolean*, ZN *char*.Pak příkazy

A: = 1.0 povoleno
A: = 1 povoleno (konstanta 1 se převede na 1.0)
I: = 1 povoleno

I: = 1.0 nepovoleno (nutno použít standardní funkce *trunc* či *round*)
ZN: = 'A' povoleno
B: = A<8 povoleno

3.6.2 Příkazy vstupu a výstupu

Pro komunikaci mezi uživatelem a počítačem slouží data, která mají formu posloupnosti znaků členěných na řádky. Nositeli vstupních a výstupních dat jsou soubory dat. Úplný popis vlastnosti souborů je uveden v kapitole 5.10.4. V této kapitole je uvedena pouze nejčastěji používaná forma pro zpracování standardního vstupního textového souboru *input* a standardního výstupního textového souboru *output.* Přiřazení souborů *input* a *output* se provede automaticky na začátku provádění programu.

Vstup ze souboru *input*

☞ **read (x,y,z)**

Jednotlivé údaje se ze souboru *input* čtou pomocí příkazu *read* ve tvaru
read (proměnná) či *read* (seznam proměnných oddělených čárkou)
readln (proměnná) či *readln* (seznam proměnných)

Proměnná typu:
- *char*. Přečte se jedinný znak a jeho hodnota se přiřadí proměnné. Přečte-li se oddělovač řádků, uloží ASCII kód #10 a #13, přečte-li se mezera, uloží se kód #32.
- **string**. Přečte se tolik znaků, kolik je předepsáno v deklaraci. Není-li v deklaraci specifikován rozsah proměnné, nutno jako oddělovač užít i <= (nový řádek) a příkaz *readln*.
-celočíselná. Přečte se posloupnost znaků tvořících dekadický zápis celého čísla s přípustným znaménkem oddělovače, tj. mezery a znaky <= před posloupností znaků jsou ignorovány.
- reálné. Přečte posloupnost znaků tvořících dekadický zápis čísla (ve tvaru s desetinnou tečkou nebo v semilogaritmickém tvaru). Oddělovače před posloupností znaků jsou ignorovány.

Příkaz *readln* způsobí, že zbytek čteného řádku se vynechá, případné další čtení probíhá již od počátku nového řádku. *readln* (X) má stejný účinek jako *read* (X); *readln*. Čtení čísel typu celočíselného a reálného nečiní problémy.

Příklad:

```
var A:real;
    I:integer;
begin
  read(I,A);
  {nebo
    read(I);read(A);
  nebo
    readln(I,A);}
end.
```

```
Vstupní data:
1 11.1
nebo
1
11.1
```

Příklad:

```
var A,B,C,D:integer;
begin
  read(A,B,C,D);
  {nebo
    read(A);read(B); read(C);read(D);
  nebo
    readln(A,B,C,D);
   ne readln(A);readln(B); readln(C);readln(D);}
  end.
```

Vstupní data:

```
1 2 3 4
nebo
1
2
3
4
nebo
1 3
3 4
```

Příklad:

```
var A,B,C,D:integer;
begin
  readln(A);readln(B); readln(C);readln(D);
end
  end.
```

Vstupní data:

```
1
2
3
4
ne 1 2 3 4
nebo ne
1
2 3
4
```

Poznámka:
Pro data
1 2 3 4
5
6
7
bude výsledek A=1, B=5,C=6,D=7. V prvním řádku dat budou data 2 3 4 ignorována.

Čtení znaků a řetězců nečiní problémy, je-li u řetězu uvedena délka.

Příklad:

```
var ZN1,ZN2:char;
   ZNSTRING:string[3];
begin
  read(ZN1,ZN2,ZNSTRING);
  {nebo
 readln(ZN1,ZN2,ZNSTRING);}
end.
```

```
Vstupní data:
ABXYZ
```

```
ne však
A
B
XYZ
```

```
var ZN1,ZN2:char;
   ZNSTRING:string[3];
begin
  readln(ZN1);readln(ZN2);readln(ZNSTRING);
end.
```

```
Vstupní data:
A
B
XYZ
```

```
ne však
ABXYZ
```

Problém může nastat při čtení řetězce bez udané délky.

```
var ZN1,ZN2,ZNSTRING:string;
begin
  readln(ZN1);readln(ZN2);readln(ZNSTRING);
  {ne však readln(ZN1,ZN2,ZNSTRING}
end.
```

```
Vstupní data:
A
B
XYZ
```

```
ne však
ABXYZ
```

Další problém při čtení nastane při čtení na proměnnou typu *integer* resp. *real* a *char* resp. **string**. Nelze například použít zápis *read* (A,ZN); ze souboru *input* ve tvaru 111A nebo 111 A nebo 111<= A, kde A je *integer (real)* a ZN **char** (**string**).

Pro tvar 111A je hlášena chyba 'invalid numeric format', pro 111 A je na proměnnou ZN uložena mezera, tj. hodnota #32, ne znak A, obdobně pro 111<=A je uložena hodnota #10 a pak #13.

Správný zápis má tvar
read (A, ZN, ZN) pro soubor *input* ve tvaru 111 A
read (A, ZN, ZN, ZN) pro soubor *input* 111<= A.

Výstup do souboru *output*

☞ **write (X,Y,Z)**

Jednotlivé údaje se do souboru *output* zapisují pomocí příkazu *write* , jehož základní tvar je
write (parametr) nebo *write* (seznam parametrů oddělených čárkou)
writeln (parametr) nebo *writeln* (seznam parametrů).
Parametry uvedené v závorce za identifikátorem standardní procedury *write* mohou mít tvar

- vystupující údaj typu *char*, **string**, reálný, celočíselný, *boolean*
- vystupující údaj: počet znaků
- vystupující údaj: počet znaků:počet desetinných míst (jen pro reálný typ).

Není-li počet znaků určen, pak je počet stanoven konkrétní implementací. Je-li v parametru určen větší počet znaků, než je nutné, doplní se vnější representace vystupující hodnoty zleva mezerami. Je-li v parametru určen menší počet znaků, než je nutné, vystoupí minimální počet znaků, nutných k zobrazení údaje. Je-li u reálného typu zadaný v parametru počet desetinných míst, vystoupí reprezentace desetinného čísla v pevné řádové čárce, jinak v semilogaritmickém tvaru. Hodnota *true* či *false* vystoupí jako řetězec.

Příklad:

```
var     I: integer;
        R: real;
        ZN: char;
        ZNSTRING: string [3];
        B: boolean;
```

Uvažujme hodnoty I:=-1; R:=10.0; ZN:='R'; ZSTRING:='ABC'; B:=*true*;

Příkaz:	Vytvořená vnější reprezentace
write (I)	-1
write (I:5)	-1
write (I:1)	-1
write (R) *2	1.0000000000E+00
write (R:9)	1.00E+01
write (R:5) *1	1.0E+01
write (R:2:1)	10.0
write (R:5:2)	10.00
write (R:20:13) *3	10.00000000000
write (ZN)	R
write (ZN:4)	R
write (ZSTRING)	ABC
write (ZSTRING:5)	ABC
write (ZSTRING:1)	ABC
write (B)	TRUE
write (B:5)	TRUE
write (B:1)	TRUE

write (**not** B:1) FALSE

Poznámka:

U reálných hodnot je minimální parametr počet znaků 8, je-li menší, dosadí se 8(*1), není-li uveden, pak 17(*2). Počet desetinných míst je maximální 11, je-li větší, dosadí se jen 11(*3).

Ukončení vytvářeného řádku, tj. zápis oddělovače řádku do výstupní posloupnosti znaků provádí příkaz *writeln,* který může být bez parametrů nebo s parametry. Pro předchozí příklad uvažujme zápis ve tvaru

Příkaz	Vytvořená vnější reprezentace
writeln (I:4,R:5:2);	-110.10
writeln (ZN:1, ZSTRING:5);	A ABC
writeln;	<=
write (B:4);	TRUE

Příklad:

```
var a,b,c,d:integer;
v,x,y,z:real;
begin
a:=1;b:=2;c:=3;d:=4;
v:=1.1;x:=2.2;y:=3.3;z:=4.4;
writeln(a,b,c,d);
writeln(a:2,b:2,c:2,d:2);
writeln(a:2,b:2);
writeln(c:2,d:2);
writeln(v:5:2,x:5:1);
writeln(y:5:2,z:5:1);
writeln(v,x,y,z);
writeln(v:5:2,x:5:2,y:5:1,z:5:0);
end.
```

Výsledky:

```
1234 [8]                pro writeln(a,b,c,d)
 1 2 3 4                pro writeln(a:2,b:2,c:2,d:2)
 1 2                    pro writeln(a:2,b:2)
 3 4                    pro writeln(c:2,d:2)
 1.10  2.2              pro writeln(v:5:2,x:5:1)
 3.30  4.4              pro writeln(y:5:2,z:5:1)
 1.10000000000036E+0000 2.20000000000073E+0000 3.29999999999927E+0000 4.40000000
000146E+0000            pro writeln(v,x,y,z)
 1.10 2.20  3.3    4    pro writeln(v:5:2,x:5:2,y:5:1,z:5:0)
```

[8] Pro typ integer je nuto zapsat počet požadovaných znaků, mezera mezi čísly není uvedena.

Příklad: Porovnejme možnosti tisku matice (tabulky).

```
var I,J,N,M:integer;
    A:array[1..10,1..10] of integer;
begin
    writeln;
    writeln('zadej pocet radku, pocet sloupcu');
    readln(N,M);{N - pocet radku, M - pocet sloupcu}
    writeln('zadej prvky matice');
    for I:=1 to N do
    for J:=1 to M do
    read(A[I,J]);

    writeln;
    writeln('tisk vysledne matice jako radek');
    for I:=1 to N do
    for J:=1 to M do
    write(A[I,J]:3);

    writeln;  writeln;
    writeln('tisk vysledne matice jako sloupec');
    for I:=1 to N do
    for J:=1 to M do
    writeln(A[I,J]:3);

    writeln;
    writeln('tisk radku matice na novy radek');
    for I:=1 to N do
    begin
    writeln;
    for J:=1 to M do
    write(A[I,J]:3)
    end;
end.
```

Data:

```
zadej pocet radku, pocet sloupcu
2 3
zadej prvky matice
1 2 3 4 5 6
nebo lépe
1 2 3
4 5 6
```

Výsledek:

```
tisk vysledne matice jako radek
1 2  3  4  5  6

tisk vysledne matice jako sloupec
 1
 2
 3
 4
 5
 6

tisk radku matice na novy radek

 1  2  3
 4  5  6
```

3.6.3 Strukturované příkazy

Složený příkaz

☞ **begin .... end**

Tímto příkazem vytváříme z několika příkazů jeden příkaz tím, že příkazy zapíšeme mezi **begin** a **end.**

```
begin
  read (A,B,C);
  X1: = (sqr (B) - sqrt (4*A*C))/(2*A);
  X2: = (sqr (B) + sqrt (4*A*C))/(2*A);
  writeln ('X1:', X1:10:5, 'X2:', X2:10:5)
end
```

Podmíněný příkaz

☞ **if A then B:=10**

Tento příkaz umožňuje větvení v programu pomocí **if** či **case.** Příkaz **if** má tvar

a) **if** B **then** P;

b) **if** B **then** P1 **else** P2;

V prvním případě, má-li boolovský výraz B hodnotu *true,* provede se příkaz P, v opačném případě se neprovede nic. Ve druhém případě, má-li B hodnotu *true,* provede se příkaz P1, v opačném případě příkaz P2. Před klíčovým slovem **else** nesmí být uveden středník. Příkazy P,P1,P2 mohou být libovolné příkazy.

Příklad:

1. **if** A > 3 **then** X:= 1;
2. **if** A > 3 **then begin** X:= 1; Y:= 2 **end;**
3. **if** A > 3 **then** X:=**1 else** X: = 2;

Pokud neuvedeme složený příkaz, dostaneme program se zcela jiným významem, například pro předchozí příklad č. 2.

```
if A > 3 then X:= 1;
Y:= 2;
```

Za **then** či **else** může následovat opět příkaz **if**. Konstrukce **if** B1 **then if** B2 **then** P1 **else** P2 se chápe následovně

if B1 **then begin** if B2 **then** P1 **else** P2 **end;**

Příklad: Sestavme program, který počet členů řady na proměnnou N a přečte a vytiskne prvky řady A, viz obdobný příklad u cyklu **for**, **while** a **repeat**

```
var A,N,I:integer;
label L;
begin
      read(N);
       I:=1;
       L:read(A);
       write(A:5);
       I=I+1;
       if I>=N goto L;
end.
```

Příklad:

Sestavme program, který při stlačení klávesy A či a, a pak Enter vypíše zprávu -KLÁVESA BYLA či NEBYLA STISKNUTA.

```
var X:char;
begin
readln(X);
if X='a' then writeln('stisknuta klavesa')
       else
if X='A' then writeln('stisknuta klavesa')
       else writeln('nestisknuta klavesa');
end.
```

```
{nebo}
   if (X=>`a`) or (X=`A`) then
  writeln (`stisknuta klavesa`)
   else writeln (`nestisknuta klavesa`)
```

Příkaz case

☞

```
Case V of
'1':X:=1;
'2':X:=2
    end
```

Tento příkaz umožňuje předepsat větvení výpočtu do několika alternativ v závislosti na hodnotě výrazu ordinálního typu.

Příkaz **case** má tvar

case V **of**
  K1 : P1;
  K2 : P2
end

nebo

case V **of**
  K1 : P1;
  K2 : P2
else
  PN
end

kde V je výraz ordinálního typu, K1,K2 jsou konstanty stejného typu jako výraz V a P1,P2,PN jsou příkazy.

Příklad pro určení sudých a lichých čísel v intervalu 0 až 9.

```
case I of                          {I typu integer}
0,2,4,6,8:writeln ('SUDA');
1,3,5,7,9:writeln ('LICHA')
else
writeln ('zaporne nebo > 9')
end.
```
[9]

Poznámka:

Uvedený příklad lze získat také v nevhodném a nepřesném tvaru (co záporné nebo > 9 ?)

```
If I=0 then write ('suda'); If I=1 then write ('licha');
If I=2 then write ('suda'); If I=3 then write ('licha');
If I=4 then write ('suda'); If I=5 then write ('licha');
If I=6 then write ('suda'); If I=7 then write ('licha');
If I=8 then write ('suda'); If I=9 then write ('licha');
```

Nebo rovněž v nevhodném tvaru:

```
If I=0 then write ('suda') else If I=1 then write ('licha') else If I=2 then write ('suda') else
If I=3 then write ('licha') else If I=4 then write ('suda') else If I=5 then write ('licha') else
If I=6 then write ('suda') else If I=7 then write ('licha') else If I=8 then write ('suda') else
If I=9 then write ('licha') else writeln ('zaporne nebo > 9')
```

Příklad pro určení velkých a malých písmen a upozornění, že daná písmena nejsou v daných datech:

```
case I of                          {I typu char}
'A'..'Z':writeln ('velka pismena ');
'a'..'z':writeln ('mala pismena ');
else
writeln ('specialni znaky')
end.
```

Příkazy cyklu

Tyto příkazy umožňují předepsat opakování příkazu několikrát. Počet opakování můžeme předepsat zapsáním počtu opakování pomocí parametru cyklu nebo zadáním podmínky.

a) Příkaz **for**

☞ **for I:= 1 to N do příkaz**

V programu je často potřeba předepsat opakování provádění některého příkazu, přičemž počet provedení tohoto příkazu je již znám. Příkaz má tvar

for I: = V1 **to** V2 **do** příkaz {krok je +1}

nebo

for I: = V2 **downto** V1 **do** příkaz {krok je -1 }

kde příkaz může být libovolný příkaz, který však nesmí měnit hodnotu řídící proměnné, v našem případě hodnota I,

[9] Pozor, **end** u tohoto příkazu nemá **begin**

I je identifikátor proměnné ordinálního typu,

V1, V2 jsou počáteční a koncová hodnota, tyto výrazy musí být ordinálního typu. Počítají se pouze před prvním provedením příkazu tvořícího tělo programu a během cyklu je není možno měnit. Je-li V1 > V2 (pro **to**) či V1<V2 (pro **downto**), cykl se neprovede.

Příklad:

```
for I:= 1 to 10 do
begin A:= I+1;
      write (A)
end
```

Příkaz **for** nemá vlastní ukončení cyklu a je proto nutné nazapomínat na složený příkaz. Rozdíl mezi následujícími příkazy je zřejmý

1) **for** I:=1 **to** 10 **do**
   read (A [I]);
   write (A [I]); { ???? }
2) **for** I:=1 **to** 10 **do**
   begin
   read (A [I]);
   write (A [I])
   end.

Příklad: Sestavme program, který počet členů řady na proměnnou N a přečte a vytiskne prvky řady A, viz obdobný příklad u podmíněného příkazu **if** a cyklu **while** a **repeat**

```
var A,N,I:integer;
begin
      read(N);
       for I:=1 to N do
       begin
       read(A);
       write(A:5);
       end;
end.
```

Příklad: Určete max. číslo z řady čísel *a*.

```
        var MAX:real;
        A:array [1..100] of real;
        I,N:integer;
begin
       MAX:=-9999; read(N);
       for I:=1 to N do
       begin
        read(A[I]);
        if A[I]>MAX then MAX:=A[I];
       end;
       writeln('maximalni prvek:          ',MAX);
end.
```

nebo:

```
        var MAX:real;
            A:array [1..100] of real;
   I,N:integer;
begin
     read(N);  read(A[1]);  MAX:=A[1];
        for I:=2 to N do
        begin
          read(A[I]);
          if A[I]>MAX then MAX:=A[I];
        end;
        writeln('maximalni prvek:           ',MAX);
end.
```

nebo:

```
        var MAX:real;
            A: real;
          I,N:integer;
begin
     read(N);  read(MAX);
        for I:=2 to N do
        begin
          read(A);
          if A>MAX then MAX:=A;
        end;
        writeln('maximalni prvek:           ',MAX);
end.
```

b) Příkaz **repeat**

Příkaz má tvar

repeat
příkazy
until B

V daném příkaze se vykonávají příkazy mezi **repeat** a **until**, pak se vyhodnotí boolovský výraz B. Je-li hodnota B *true*, cykl se ukončí, v opačném případě se proces opakuje**.** Posloupnost příkazů musí B vhodně modifikovat, aby existoval výstup ze smyčky.

Příklad: Sestavme program, který požaduje opakované čtení na proměnnou N, pokud není přečtena hodnota 0<N<=10.

```
        Uses crt;
        var N:integer;
        begin
        clrscr;
        repeat
        write('zadej N>0 a N<=10 ');
        read(N);
        until (N>0) and (N<=10);
        end.
```

Příklad: Sestavme program, který počet členů řady na proměnnou N a přečte a vytiskne prvky řady A, viz obdobný příklad u podmíněného příkazu **if** a cyklu **for** a **while**

```
var A,N,I:integer;
begin
readln(N);
I:=1;
repeat
read(A);
write(A:5);
I:=I+1;
until I>N-1;
end.
```

Příklad**:** Určete max. číslo z řady čísel a.

```
var MAX:real;
   A: real;

begin
read(MAX);   {nebo MAX:=-99999;}
repeat
  read(A);
  if A>MAX then MAX:=A;
  until  (A:=-99999);
writeln('maximalni prvek:                ',MAX);
end.
```

nebo:

```
var MAX:real;
   A: real;
   F:text;
begin
assign(F,'d:\DATA.DAT');
reset(F);
read(F,MAX);
repeat
  read(F,A);
  if A>MAX then MAX:=A;
  until  eof(F);
writeln('maximalni prvek:                ',MAX);
 close(F)
end.
```

Nebo pro známé N:

```
var MAX:real;
   A: real;
   I,N:integer;
    F:text;
  begin
```

```
assign(F,'d:\DATA.DAT');
reset(F);
read(F,N);
read(F,MAX); I:=2;
        repeat
        read(F,A);
        if A>MAX then MAX:=A;
        I:=I+1;
        until (I>N);
  end.
```

c) Příkaz **while**

☞ **while B do příkaz**

Příkaz má tvar
while B **do**
libovolný příkaz

Nejprve se vyhodnotí boolovský výraz B a je-li jeho hodnota *true*, provede se příkaz uvedený za **do** a tato činnost se opakuje. V opačném případě se provádění opakování ukončí. Je-li hodnota B *false* již na počátku provádění, pak příkaz uvedený za **do** se neprovede ani jednou (na rozdíl od příkazu **repeat**, který se provede alespoň jednou).

Příklad: Sestavme program, který počet členů řady na proměnnou N a přečte a vytiskne prvky řady A, viz obdobný příklad u podmíněného příkazu **if** a cyklu **for** a **repeat**

```
var A,N,I:integer;
begin
       read(N);
        I:=1;
        while I<N do
        begin
        read(A);
        write(A:5);
        I:=I+1;
        end;
end.
```

Příklad: Určete max. číslo z řady čísel *a*.

```
var MAX:real;
   A: real;
   I,N:integer;
   F:text;
begin
        assign(F,'d:\DATA.DAT');
        reset(F);
         read(F, MAX);
        while not eof(F) do
        begin
          read(F,A);
          if A>MAX then MAX:=A;
```

```
        end;
        writeln('maximalni prvek:               ',MAX);
         close(F)
end.
```

Příklad**:** Určeme počet výskytů zvoleného znaku v souboru DATA.DAT. Vyhledávaný znak přečteme z klávesnice na proměnnou ZNHL. Znaky ze souboru DATA.DAT čteme na proměnnou ZN. Výsledný počet vyhledávaného znaku uložíme na proměnnou S.

```
        var F:text;
           ZN,ZNHL:char;
           S:integer;
        begin
        assign(F,'DATA.DAT');
        reset(F);
        S:=0;
        write('zadej hledany znak');
        readln(input,ZNHL);
        while not eof(F) do
        begin
        read(F,ZN);
        if ZNHL=ZN then S:=S+1;
        end;
        write(output,'pocet znaku ',ZNHL,' je ',S);
        close(F);
        readln(input);
        end.
```

```
        Data z klávesnice (soubor input):
        Zadej hledany znak
        1
```

```
        Data v souboru DATA.DAT:
        23Aasd f1234
        a s 33 411
```

```
        Výsledky:
        pocet znaku 1 je 3
```

Poznámka:

Při použití příkazů **repeat** a **while** je nebezpečí vytvoření nekonečného provádění příkazů při chybném zadání podmínky pro ukončení cyklu. V praktických příkladech je tedy nutno v těle cyklu měnit cílovou podmínku.

3.7 Operace pro jednoduché typy deklarované programátorem

Deklarace jednoduchých standardních typů byla popsána v předchozích odstavcích. Uveďme operace pro jednoduché typy deklarované programátorem - výčtový typ a typ interval.

3.7.1 Výčtový typ

☞ **STŘEDA > ČTVRTEK**

Pro výčtové typy definované v programu jsou definovány pouze relační operace, nejsou pro ně zavedeny vstupní a výstupní konverze pro příkazy read a write. Pro výstup je možno použít příkazu **case**, nebo lze pro vstup a výstup pracovat s jejich ordinálními čísly.

Výčtový typ je ordinální typ, lze tedy použít standardní funkce *ord, succ, pred.*

ord (PONDELI) = 0; *ord* (UTERY) = 1

PONDELI < UTERY; UTERY < STREDA

succ (PONDELI) = UTERY; *succ* (NEDELE) není definováno

pred (UTERY) = PONDELI; *pred* (PONDELI) není definováno

Příklad:

Sestavme program pro výpočet průměrného týdenního zisku a maximálního zisku v daný den.

```
type DEN=(PONDELI,UTERY,STREDA,CTVRTEK,PATEK,SOBOTA,NEDELE);
var ZISK,PRUMER_ZISK,MAX_ZISK:real;
    I,DENMAX: DEN;
begin
PRUMER_ZISK:=0;
MAX_ZISK:=0;
writeln('zadej zisky v pondeli az nedele');
for I:=PONDELI to NEDELE do
begin
  read(ZISK);
  PRUMER_ZISK:=PRUMER_ZISK+ZISK;
  if ZISK > MAX_ZISK then begin MAX_ZISK:=ZISK; DENMAX:=I end
end;
PRUMER_ZISK:=PRUMER_ZISK/(ord(NEDELE)+1);
writeln('prumerny zisk:',PRUMER_ZISK:10:2,' Kc.');
write('maximalni zisk: ',MAX_ZISK:5:2,' Kc. v ');
case DENMAX  of
    PONDELI:writeln('pondeli');
    UTERY:writeln('utery');
    STREDA:writeln('stredu');
    CTVRTEK:writeln('ctvrtek');
    PATEK:writeln('patek');
    SOBOTA:writeln('sobotu');
    NEDELE:writeln('nedeli')
    end
end.
```

```
Data:
5500
 333
 884
 100
```

```
2990
2881
6333
```

```
Výsledky:
prumerny zisk: 2717.29 Kc
maximalni zisk: 6333 Kc v nedeli
```

3.7.2 Typ interval

☞ **var A: array [1..500] of char;**[10]

Jak již bylo prve uvedeno, interval je typ, který specifikuje neprázdnou souvislou podmnožinu hodnot nějakého ordinálního typu.

Stanovením intervalu jako typu proměnné se omezuje množina přípustných hodnot dané proměnné. Při přiřazení lze přiřazovacím příkazem definovat pouze takovou hodnotu proměnné, která je kompatibilní vzhledem k přiřazení s typem přiřazované proměnné.

U proměnných typu interval, celočíselné a znak se při čtení kontroluje, zda hodnota, která má být čtením přiřazena, patří do daného intervalu.

Typy interval mohou být použity rovněž jako typy formálních parametrů procedur a funkcí. Tímto způsobem můžeme vyjádřit případné omezení přípustné množiny hodnot určitého parametru.

Příklad:

```
var MAXINTZ: 0..maxint;
   MAXINTK: -maxint..-1;
MAXINTZ:= MAXINTK; {příkaz je evidentně nesprávný}
```

Příklad:

```
var I, N: integer ;
      A: array [1..5] of real;
begin
read (N);
for I:=1 to N do
read (A [I]);
end.
```

V případě, že je přečtena hodnota N mimo interval 0..10, je ohlášena chyba 201: Překročení rozsahu. V případě, že není zapnut příkaz Option-Compiler-Range Checking, není tato chyba kontrolována.

```
File  Edit  Search  Run  Compile  Debug  Tools  Options  Window  Help

    Code generation
      [X] Force far calls          [X] Word align data
      [ ] Overlays allowed         [ ] 286 instructions
     Runtime errors               Syntax options
      [X] Range checking           [X] Strict var-strings
      [X] Stack checking           [ ] Complete boolean eval
      [X] I/O checking             [X] Extended syntax
```

[10] Pozor, nepřípustný je zápis A: array [1..N] of real;

```
[X] Overflow checking      [ ] Typed @ operator
[ ] Open parameters
Debugging
[X] Debug information      Numeric processing
[X] Local symbols          [X] 8087/80287
[X] Emulation
```

Příklad: Sestavme program, který požaduje opakované čtení na proměnnou N, pokud není přečtena hodnota 0<N<=10.

```
Uses crt;
var N:integer;
begin
clrscr;
repeat
write('zadej N>0 a N<=10 ');
read(N);
until (N>0) and (N<=10);
end.
```

Nyní něco pro zvýdavější:

3.7.3 Typová změna proměnné a výrazu

Typová změna umožňuje jednak obejít v potřebných situacích poměrně přísná pravidla typové kompatibility Pascalu, jednak umožňuje provádět různá kouzla s paměťovým prostorem alokovaným proměnným různých typů.

Následující příklad ukazuje možnost zkrátit řetězec o daný počet znaků (obdoba procedury **delete**).

```
var S : string;
   N : integer;

BEGIN
   S:='1234567890';
   read(N);
   if N<length(S) then byte(S[0]):=byte(S[0])-N;
   {nebo}
   {if N<ord(S[0]) then S[0]:=char(ord(S[0])-N);}
   writeln(S);
END.
```

```
Výsledek:
5
12345
```

Pro N=5 je řetězec 1234567890 zkrácen o 5 znaků, tedy na řetězec 12345.

Typ výrazu může být změněn na jiný typ pomocí typové změny. Typová změna výrazu je chápána jako funkce a nemůže se tedy objevit na levé straně přiřazovacího příkazu (na rozdíl od typové změny proměnné).

Obecný tvar TYP(VÝRAZ), kde TYP A VÝRAZ musí být ordinálního typu a hodnota VÝRAZ je převedena na hodnotu odpovídajícího typu TYP. Následující příklady ilustrují využití typové změny výrazu.

```
var A:array[1..3] of char;
I1,I2,I3:byte;
begin
  A[1]:=1;A[2]:=2; A[3]:=4;
  I1:=byte(A[1]);
  I2:=byte(A[2]);
  I3:=byte(A[3]):
writeln(I1:3,I2:3,I3:3)
end.
```

```
Výsledek:
  49 50 52
```

```
var A:array[1..3] of byte;
I1,I2,I3:char;
begin
  A[1]:=1;
  A[2]:=2;
  A[3]:=4;
  I1:=byte(A[1]);
  I2:=byte(A[2]);
  I3:=byte(A[3]):
writeln(I1:3,I2:3,I3:3)
end.
```

```
Výsledek:
❶ ♥ ♦
```

Uvedené postupy umožní vytvořit tabulku ASCII kódu, viz následující program

```
uses crt;
var A : array[0..255] of char;
   I : array[0..255] of byte;
   J,ISTR:integer;
BEGIN
ISTR:=0;
clrscr;
for J:=0 to 255 do
begin
   A[J]:=char(J);
   I[J]:=byte(A[J]);
   write(A[J]:3,I[J]:4);
   if ISTR=10 then begin ISTR:=0;writeln;end;
   ISTR:=ISTR+1;
END;
end.
```

Zajímavý je rovněž následující program

```
type   TDNY=(pondeli,utery,streda,ctvrtek,patek,sobota,nedele);
       var I:integer;            {nebo I:char;}
          DNY:TDNY;
       begin
         DNY:=TDNY;
         I:=integer(DNY);        {nebo I:=char(DNY);}
         writeln(I:3)
       end.
```

Výsledek:

```
3                    nebo    ♥
```

3.8 Zápis jednoduchých programů

V této kapitole jsou uvedeny jednoduché příklady pro ilustraci využití příkazů, které byly dosud probrány.

3.8.1 Příklad: Program pro výpočet obvodu a obsahu kruhu.

```
var R,OBSAH,OBVOD : real;
begin
write('Zadej polomer kruhu R = ');
readln(R);
OBSAH:=PI* R * R;  { R*R možno nahradit SQR(R), PI je Ludolfovo číslo }
OBVOD:=2*PI* R;
writeln('Obsah: ', OBSAH:5:2);
writeln('Obvod: ', OBVOD:5:2);
end.
```

Data:
Zadej polomer kruhu r =
1[11]

Výsledek:
Obsah: 3.14
Obvod: 6.28

[11] Data je vhodné zadávat tak, aby kontrola výsledků byla velmi jednoduchá

3.8.2 Příklad: Určeme zda zvolené celé číslo x leží v intervalu a<=x<=b

```
var A,B,X:integer;
     label L1;
begin
L1:write('zadej dolni a horní mez: ');
readln(A,B);
if A>=B then
               begin
                  writeln('dolní mez je mensi nebo rovno horní mez');
                  goto L1;
                end
             else
                 begin
                  write('Zadej X: ');
                  readln(X);
                  if (X>=A) and (X<=B) then writeln('cislo ', X:3, ' je v daném intervalu')
                  else writeln('cislo ' 'neni v intervalu');
                 end;
end.
```

```
Data:
zadej dolni a horní mez: 10 20
zadej X: 15
```

```
Výsledek:
cislo 15 je v daném intervalu
```

3.8.3 Příklad: Určení počtu symbolů.

Určeme počet malých písmen, velkých písmen, číslic a speciálních znaků v datovém souboru DATA.DAT.

```
var I:char;
   MALA,VELKA,SPECIAL,CISL:integer;
   F:text;
begin
assign(F,'a:\DATA.DAT');
reset(F);
MALA:=0;
VELKA:=0;
SPECIAL:=0;
CISL:=0;
while not eof(F) do
begin
  read(f,I);
  case I of
    'A'..'Z':VELKA:=VELKA+1;
    'a'..'z':MALA:=MALA+1;
    '0'..'9':CISL:=CISL+1;
    else SPECIAL:=SPECIAL+1
  end
end;
writeln('velka:          ',VELKA);
writeln('mala:           ',MALA);
writeln('ostatni:        ',CISL);
writeln('special:        ',SPECIAL);
readln
end.
```

```
Data:
123 ABC def
+1.0;
```

```
Výsledek:
velka:      3
mala:       3
cislice:    5
ostatni:    7
```

3.8.4 Příklad: Výpočet kořenů kvadratické rovnice.

Sestavme program pro výpočet kořenů kvadratické rovnice

$$a\,x^2 + bx + c = 0$$

Kořeny určeme dle vztahu

$$x_{1,2} = (-b + -\sqrt{d}\,) / (2a); d = b^2 - 4ac$$

```
{vypocet korenu kvadraticke rovnice}
var X1R,X1I,X2R,X2I,A,B,C,D:real;
begin
write('Zadej koeficienty A B C :');
read(A,B,C);
if A=0 then begin
            X1R:=-C/B;
            writeln('jednoduchy koren' ,X1R:10:4)
           end
      else begin
            D:=sqr(B)-4*A*C;
            if D>=0 then {vypocet realnych korenu}
                     begin
                       X1R:=(-B+sqrt(D))/(2*A);
                       X2R:=(-B-sqrt(D))/(2*A);
                       X1I:=0;
                       X2I:=0
                     end
                   else{vypocet komplexnich korenu}
                     begin
                       X1R:=-B/(2*A);
                       X1I:=sqrt(abs(D))/(2*A);
                       X2R:=X1R;
                       X2I:=-X1I
                     end;
            writeln('koreny kvadraticke rovnice:');
            writeln(' ',X1R:10:4,' +i * ',X1I:10:4);
            writeln(' ',X2R:10:4,' +i * ',X2I:10:4)
            end
end.
```

Výsledek:

```
Zadej koeficienty A B C: 1  -3  -10
Koreny kvadraticke rovnice:
 5.0000 + i*   0.0000
-2.0000 + i*   0.0000
```

3.8.5 Příklad: Určení počtu bodů v kvadrantech.

Sestavme program pro určení počtu bodů ležících ve kvadrantech 1 až 4, pro body zadané souřadnicemi (x, y), počet bodů je n. Předpokládejme, že nenastane případ, že x resp. y je rovno nule.

```
uses crt;
var N,S1,S2,S3,S4,I:integer;
      X,Y:real;
begin
clrscr;
write('zadej pocet souradnic: ');
read(N);
S1:=0;S2:=0;S3:=0;S4:=0;
write('zadej souradnice X,Y: ');
for I:=1 to N do
begin
    read(X,Y);
    if (X>0) and (Y>0) then S1:=S1+1
    else if (X<0) and (Y>0) then S2:=S2+1
    else if (X<0) and (Y<0) then S3:=S3+1
    else if (X>0) and (Y<0) then S4:=S4+1
    else writeln('X or Y = 0')
end;
writeln('pocet bodu v kvadrantu: ');
writeln('1:',S1:3,' 2:',S2:3,' 3:',S3:3,' 4:',S4:3)
end.
```

```
výsledek:
zadej pocet souradnic: 5
zadej souradnice: 1 1 2 2 -1 2 -2 3 -5 -5

pocet bodu v kvadrantu:
1: 2    2: 1    3: 1    4: 1
```

3.8.6 Příklad: Vyhledání maxima a minima v řadě čísel.

Sestavme program pro vyhledání maximálního a minimálního prvku z řady čísel X_i, i = 1,2,.. n.

```
uses Crt;
var N,I:integer;
   MIN,MAX,X:real;
begin  ClrScr;
      write ('zadej pocet clenu rady N: ');
      read (N);
      write ('zadej N clenu rady: ');
      read (MIN);
      MAX:=MIN;
      for I:=2 to N do
      begin read(X);
            if X>MAX then MAX:=X;
            if X<MIN then MIN:=X
{nebo  if X>MAX then MAX:=X else MIN:=X}
      end;
      writeln ('minimum: ',MIN:10:3);
      writeln ('maximum: ',MAX:10:3)
end.
```

```
výsledek:
zadej pocet clenu rady N: 5
zadej N clenu rady: 2.1 333.2 3.22 99.3 5.88
minimum: 2.100
maximum: 333.200
```

Obdobný program lze zapsat také v následujícím tvaru, který ilustruje možnost chyby v programu obtížně k odhalení. Chyba se projeví, je-li první prvek v řadě minimální.

```
var N,I:integer;
MAX,MIN:real;
A:array[1..100] of real;
begin
writeln;
writeln('zadej pocet prvku v rade');
read(N);
writeln('zadej prvky rady');
for I:=1 to N do
read(A[I]);
MAX:=-9999; MIN:=9999;
for I:=1 to N do
if A[I]>MAX  then MAX:=A[I] else MIN:=A[I];
{možnost opravy  if X>MAX then MAX:=X;  if X<MIN then MIN:=X}
write('MAX= ',MAX:10:3,'   MIN=', MIN:10:3);
end.
```

Data a výsledky:

```
zadej pocet prvku v rade
5
zadej prvky rady
5
1 2 3 4 5
MAX=      5.000   MIN=     4.000

zadej pocet prvku v rade
5
zadej prvky rady
1 5 4 3 2
MAX=      5.000   MIN=     2.000

zadej pocet prvku v rade
5
zadej prvky rady
1 3 2 4 5
MAX=      5.000   MIN=     2.000

zadej pocet prvku v rade
5
zadej prvky rady
2 3 1 4 5
MAX=      5.000   MIN=     1.000
```

3.8.7 Příklad: Součet lichých čísel z řady.

Sestavme program pro sečtení všech lichých čísel z řady celých kladných čísel X_i, i = 1,2,.., n.

```
uses Crt;
var X,N,SUMA:integer;
begin   ClrScr;
      write('zadej N: ');
      read(N);
      SUMA:=0;
      write('zadej ',N,' celych cisel: ');
      for I:=1 to N do
      begin read(X);
        if (X mod 2) <> 0 then SUMA:=SUMA+X
       {nebo if odd(X)  then SUMA:=SUMA+X
            if (X div 2)*2 <> X then SUMA:=SUMA+X}
      end;
      writeln('Suma lichych cisel: ',SUMA:3);
end.
```

```
výsledek:
zadej N: 5
zadej N celych cisel: 1  2  3  4  5

suma lichych cisel:    9
```

3.8.8 Příklad: Součet lichých čísel s koncovou značkou.

Sestavme program pro sečtení všech lichých čísel z řady celých kladných čísel X. Počet čísel není znám, řada čísel je ukončena koncovým znakem 9999. Předpokládá se alespoň jedno číslo v řadě.

```
uses Crt;
var X,SUMA:integer;
begin  ClrScr;
     SUMA:=0;
     writeln('zadej cela cisela - koncime s 9999 ');
     read(X);
     repeat if odd(X) then SUMA:=SUMA+X;
          read(X)
     until X=9999;
     {nebo
     read(X);
     while x<>9999 do
     begin if odd(X) then SUMA:=SUMA+X;
          read (X)
     end;}
     writeln('suma lichych cisel:',SUMA:7)
end.
```

```
Výsledky:
zadej celya cisela - koncime s 9999: 1 2 3 4 5
suma lichych cisel:       9
```

3.8.9 Příklad: Výpočet výsledku výrazu.

Sestavme program pro určení výsledku výrazu

$1 + X/1! + X^2/2! + ... + X^n/n!$

Při řešení je výhodné využít platnosti

$X^2/2!=X/1*X/2$; $X^3/3!=X^2/2!*X/3$; $X^4/4!=X^3/3!*X/4$; atd.

```
var SUMA,POM,X:real;
   N,I:integer;
begin SUMA:=1;
    POM:=1;
    write('zadej X:');
    read(X);
    repeat write ('zadej N:');
    read(N)
    until (N>=1);
    {vypocet bud}
    I:=1;
    while I<=N do
    begin POM:=POM*X/I;
        SUMA:=SUMA+POM;
        I:=I+1
    end;
    writeln('suma: ',SUMA ,' pro x: ', X, ' n:', N:2);
    {nebo}
    SUMA:=1;
    POM:=1;
    for I:=1 to N do
    begin POM:=POM*X/I;
        SUMA:=SUMA+POM
    end;
    writeln('suma: ',SUMA ,' pro x: ', X, ' n:', N:2);
    {nebo}
    SUMA:=1;
    POM:=1;
    I:=1;
    repeat
    POM:=POM*X/I;
    SUMA:=SUMA+POM;
    I:=I+1;
    until I>N;
    writeln('suma: ',SUMA ,' pro x: ', X, ' n:', N:2);
end.
```

```
Výsledky:
zadej X:1
zadej N:2
suma: 2.5000000000e+00 pro x: 1.0000000000 n: 2
```

3.8.10 Příklad: Sestavme program pro setřídění řady čísel

Setřídit řadu čísel lze mnoha metodami, mezi jednoduší patří metoda transpozice prvků – metoda bublání. Třídění probíhá porovnáním sousedních prvků, tj a_j s a_{j+1} , j=1,2,..,n-1. Je-li prvek $a_j < a_{j+1}$, provedeme vzájemné přemístění těchto prvků.

```
var I,J,N:integer;
POM:real;
A:array[1..100] of real;
begin
writeln('zadej pocet prvku v rade');
read(N);
writeln('zadej prvky rady');
for I:=1 to N do
read(A[I]);
for I:=1 to N do
for J:= 1 to N-1 do
if A[J]<A[J+1] then    {pro třídění opačné - od nejmenšího if A[J]>A[J+1] then }
begin    {přerovnání prvků řady}
POM:=A[J];
A[J]:=A[J+1];
A[J+1]:=POM;
end;
write('setridena rada');
for I:=1 to N do
write(A[I]:10:2);
end.
```

```
zadej pocet prvku v rade
5
```

```
zadej prvky rady
1 5 4 3 2
```

```
setridena rada      5.00      4.00      3.00      2.00      1.00
```

Příklad: Výčetka platidel.

Sestavme program, který určí počty platidel pro zadanou mzdu tzv. výčetka platidel.

```
uses crt;
var A,B,CISLO,I,M,N,P,V : integer;
   ret,mez : string;
   K,L,J   : array[1..20] of integer;
   F       : text;
BEGIN
   clrscr;
   {zadani existujicich druhu platidel}
   M:=10;
   K[1]:=2000; K[2]:=1000;  K[3]:=500;  K[4]:=200;
   K[5]:=100;   K[6]:=50;   K[7]:=10;
   K[8]:=5;    K[9]:=2;    K[10]:=1;
   {hlavicka vycetky}
   for I:=1 to M do      {nulovani souctu platidel}
   begin
     L[I]:=0;
     J[I]:=0;
   end;
   A:=0;
   FillChar(ret,80,'-'); {retezce mezer a '-'}
   FillChar(mez,80,' ');
   mez[0]:=#14;          {nastaveni delky retezce mezer}
   writeln('osobni   mzda',mez,'pocet odpovidajicich platidel');
   mez[0]:=#12;          {zmena delky retezce mezer}
   write('cislo',mez);
   for I:=1 to M do write(K[I]:5);
   writeln;
   ret[0]:=#67;          {nastaveni delky retezce podtrzeni}
   writeln(ret);
   {cteni vet v souboru a vypocet poctu platidel}
   assign(F,'DATA23.DAT');
   reset(F);
   readln(F,N);          {precten pocet zamestnancu}
   for P:=1 to N do
   begin
     read(F,CISLO,V);
     write(CISLO:4,V:9); {tisk osobniho cisla zamestnance}
                         {a jeho mzdy}
     if V>0 then         {kontrola nejvyssi zobrazitelne}
                         {hodnoty}
     begin
       A:=A+V;           {soucet vsech mezd}
       for I:=1 to M do     {plat porovnam s platidly}
       begin              {od 1000 Kc do 1 Kc}
          J[I]:=V div K[I];{urcim pocet platidel}
          V:=V-j[i]*K[I];  {vybrana platidla odcitam od mzdy}
          L[I]:=L[I]+J[I]; {suma ruznych platidel pro}
                           {vsechny zamestnance}
       end;
```

```
      mez[0]:=#4;
      write(mez);
      for I:=1 to M do
        write(J[I]:5);  {tisk nutnych platidel}
                        {pro zamestnance}
      writeln;
    end else writeln('ma moc, zmente integer na longint');
  end;
  {celkove soucty a kontrolni soucet}
  B:=0;
  for I:=1 to M do      {vypocet kontrolniho souctu}
    B:=B+L[I]*K[I];
  if A=B then           {porovnani kontr,souctu se}
                        {souctem vsech mezd}
  begin
    writeln(ret);       {podtrzeni}
    write('soucet');
    write(A:7);
    mez[0]:=#4;
    write(mez);
    for I:=1 to M do
      write(L[I]:5);
    writeln;
    mez[0]:=#44;
    write(mez,'kontrolni soucet',B:7);
    readln;
  end else writeln('nesouhlasi kontrolni soucty');
end.
```

```
Data:
5
1  1111
2  2222
3  3333
4  4444
5  5555
```

Vysledek:

```
osobni   mzda            pocet odpovidajicich platidel
cislo          2000  1000  500  200   100   50   10    5    2    1
-------------------------------------------------------------------
  1    1111       1    0    0    1    0    0    1    0    0    1
  2    2222       2    0    1    0    0    1    0    0    1    0
  3    3333       3    0    1    1    0    1    1    0    1    1
  4    4444       4    0    2    0    0    2    0    0    2    0
  5    5555       5    1    0    0    1    0    0    1    0    0
-------------------------------------------------------------------
soucet  16665      15    1    4    2    1    4    2    1    4    2
                                  kontrolni soucet  16665
```

3.9 Deklarace uživatelských procedur a funkcí

Jazyk PASCAL umožňuje strukturalizovat program nejen na úrovni základních řídících konstrukcí, ale i na úrovni uzavřených programových celků. Umožňuje tyto celky pojmenovat a potom toto jméno používat pro aktivaci pojmenovaného celku v tom místě programu, kde je vyžadován odpovídající výpočet. Je možno deklarovat dva druhy podprogramů - procedury a funkce. Abychom mohli podprogram používat, je třeba jej nejdříve deklarovat. Procedury a funkce jsou součástí úseku definic a deklarací deklarační části programu. Mezi deklarační část a příkazovou část programu lze vložit libovolné množství deklarací procedur a funkcí. Procedura či funkce může mít svou vlastní deklarační část, veškeré prvky deklarované uvnitř procedury či funkce mají lokální charakter. Deklarace uvedené v bloku však mají globální charakter. Deklarace uživatelské procedury má následující strukturu:

```
procedure identifikátor procedury (formální parametry);
deklarační část (label, const, type, var, uživatelské procedury a funkce)
begin
příkazová část;
end
```

Deklarace uživatelské funkce má následující strukturu

```
identifikátor funkce (formální parametry):specifikace  identifikátoru funkce;
lokální deklarační část
begin
příkazová část;
end
```

Globální proměnné

Globální proměnné, které jsou zavedeny v deklarační části hlavního programu existují po celou dobu provádění programu.

Jaký je rozdíl mezi procedurou a funkcí?

Funkce a procedura

Funkci i proceduru voláme jejím jménem,

- pro proceduru je jméno pouze prostředkem pro její volání,
- u funkce je nositelem výsledné hodnoty.

```
{procedura bez parametru                krok:      hodnota   promennych:}
var A,B:integer;{globální proměnné}
procedure P0;
{deklarace procedury}
begin
  writeln('p1 ',A:5,B:5);                2.        A:=1  B:=2
  A:=A+10; B:=B+20;
  writeln('p2 ',A:5,B:5)                 3.        A:=11 B:=22
end; {konec deklarace procedury}
{příkazová část hlavního programu}
begin
   A:=1;B:=2;
   writeln('pr1',A:5,B:5);               1.        A = 1  B = 2
   {volani procedury}
   P0;                                   4.        A = 11 B = 22
   writeln('pr2',A:5,B:5);
end. {konec programu}
```

```
výsledky:
pr1    1      2
p1     1      2
p2     11     22
pr2    11     22
```

Příklad:

```
{funkce bez parametru}
var X:real
function TANG:real;
begin
  TANG:=sin(X)/cos(X)
end;

begin
  write('zadej argument X:');
  read(X);
  writeln('tangens',X:10:3,' = ',TANG:10:3)
end.
```

Lokální proměnné

Lokální proměnné zavedené lokálními deklaracemi nejsou přístupné z vnějšku procedury či funkce. Nejčastěji se lokálními deklaracemi zavádějí lokální proměnné, které slouží k uložení mezivýsledků v proceduře. Lokální proměnné na rozdíl od globálních proměnných, existují pouze po dobu provádění procedury či funkce, ve které jsou uvedeny. Je-li procedura vyvolána během provádění programu několikrát, její lokální proměnná vzniká pokaždé znovu a na začátku každého volání mají nedefinovanou hodnotu. Nelze tedy očekávat, že na začátku volání procedury by její lokální proměnné měly stejné hodnoty jako na konci předchozího volání.

Příklad:

```
{procedura s lokalnimi promennymi}
procedure P0;
begin
var I:integer;
{promenna I je lokalni}
begin
for I:=1 to 3 do
  write(I);
end;
begin
   P0;
end.
```

```
Výsledky:
123
```

Komunikace procedury s hlavním programem (či jinou procedurou) prostřednictvím globálních proměnných, představuje speciální případ. Chceme-li však proceduru či funkci využívat vícekrát pro nestejně označené proměnné, je vhodnější používat formální a skutečné parametry.

Formální parametry

Parametry procedury se definují v hlavičce procedury seznamem formálních parametrů. Jednotlivé specifikace se oddělují středníkem. Podle druhu rozlišujeme parametry volané hodnotou a parametry volané odkazem – referencí.

Příklad hlavičky procedury s parametry:

```
procedure P (A,B: integer; var X,Y: real);
```

Tato hlavička obsahuje parametry

A,B - formální parametry typu integer volané hodnotou

X,Y - formální parametry typu real volané odkazem.

Formální parametry volané hodnotou

Formální parametr volaný hodnotou představuje v těle procedury lokální proměnnou, které je na začátku provedení procedury přiřazena hodnota skutečného parametru a vlastní realizace příkazu procedury se vykoná s touto lokální proměnnou. Pro parametry nahrazené hodnotou je možno do procedury hodnoty přiřadit,hodnota skutečného parametru (tj.mimo proceduru) se nemění. Přípustným skutečným parametrem může tedy být libovolný výraz, jehož hodnota je kompatibilní vzhledem k přiřazení s typem formálního parametru.

Formální parametry volané odkazem

Formální parametr volaný odkazem představuje v těle procedury vždy tu konkrétní proměnnou, která je určena skutečným parametrem a jejíž adresa umístění v paměti se vypočte na začátku provedení procedury. Tyto parametry lze použít i na odevzdání vstupní hodnoty proceduře, ale skutečný parametr není chráněný vůči změně při realizaci příkazu procedury. Skutečným parametrem může být jen proměnná, jejíž typ je totožný s typem formálního parametru.

Parametry volané odkazem používáme tedy tam, kde potřebujeme předat změny hodnot formálních parametrů v proceduře do hlavního programu. Tento způsob výměny má menší nároky na paměť než volání hodnotou.

Skutečné parametry

Pro volání procedury s parametrem slouží příkaz procedury, v němž je za identifikátorem procedury uveden seznam skutečných parametrů. Jednotlivé skutečné parametry se oddělují čárkou a jejich počet musí souhlasit s počtem formálních parametrů. Příklad volání předchozí procedury s parametry:

```
P(3,4,XR,YR);
```

Příkaz volání obsahuje parametry

3,4 - skutečné parametry volané hodnotou (mohou to být konstanty, proměnné či výrazy).

XR, YR - skutečné parametry volané odkazem (může být pouze proměnná odpovídajícího typu).

Příklad: {parametry nahrazene hodnotou krok: hodnota: }

```
  var A,B:integer;
procedure  P1(X,Y:integer);
  begin
    writeln('p1 ',X:5,Y:5);          2.    A:=1  B:=2
    X:=X+10; Y:=Y+20;
    writeln('p2 ',X:5,Y:5)           3     A=11  B=22
  end
begin
   A:=1;B:=2;
   writeln('pr1',A:5,B:5);           1.    A:=1   B:=2
   P1(A,B);
   writeln('pr2',A:5,B:5)            4.    A:=1   B:=2
end.
```

Výsledky:

```
pr1     1      2
p1      1      2
p2     11     22
pr2     1      2
```

Příklad:

```
{parametry nahrazene referenci          krok:       hodnota:}
var A,B:integer;

procedure P2(var X,Y:integer);
begin
   writeln('p1 ',X:5,Y:5);                  2.    A:=1   B:=2
   X:=X+10; Y:=Y+20;
   writeln('p2 ',X:5,Y:5)                   3.    A:=11  B:=22
end;

begin
   A:=1;B:=2;
   writeln('pr1',A:5,B:5);                  1.    A:=1    B:=2
   p2(A,B);
   writeln('pr2',A:5,B:5);                  4.    A:=1    B:=2
end.
```

Výsledky:

```
pr1     1      2
p1      1      2
p2     11     22
pr2     11     22
```

V bloku procedury nebo funkce mohou být deklarovány nejen lokální proměnné, konstanty a návětší, ale též lokální procedury a funkce. Úroveň vnoření takovýchto procedur a funkcí přitom není omezena.

```
{vnoreni deklaraci procedur}
var A,B:integer; {A,B globalni promenne}

procedure P1;
var A:integer; {A lokalni v procedure P1 zastini globalni A }

  procedure P11; { globalni A}
  var B:integer; {B lokalni v P11 zastini globalni B }
  begin            {A,B nemají definované hodnoty }
     writeln('P11');
      A:=10;
      B:=10;
     writeln('P11',A:5,B:5)
  end;

begin
  writeln('P1 ',B:5); {A nema definovanou hodnotu, B je globální }
  P11;
  writeln('P1 ',A:5,B:5) {A převzato z procedury P11, B je globální }
end;

procedure P2;
var C:integer; {C lokalni v procedure P2}
begin
  C:=100;
  writeln('P2 ',A:5,B:5,C:5) {A, B jsou globální }
end;

 begin
    A:=1; {C neni definovano}
    B:=2;
    writeln('ZK ',A:5,B:5);
    P1;
    P2;
    writeln('ZK ',A:5,B:5)
end.
```

```
Výsledky:
ZK    1     2
P1   ?  2
P11 ?    ?
P11     10    10
P1      10    2
P2      1     2     100
ZK    1     2
```

Vedlejší efekt

V těle procedur či funkcí je povoleno přiřadit hodnotu globální proměnné. Takovéto přiřazení se nazývá vedlejší efekt procedury či funkce a z hlediska bezpečnosti představuje značné nebezpečí. Vedlejší efekt je skryt v těle procedury či funkce a není zřejmý z příkazu procedury či zápisu funkce a snižuje se tím srozumitelnost programu a mohou se stát zdrojem chyb.

Pokud procedura či funkce mění hodnoty globálních proměnných uvnitř procedury či funkce, můžeme se setkat např. s následujícím překvapením:

```
uses Crt;
var A,X:integer;

function FCE(X:integer):integer;
begin
FCE:=A+X;
A:=A+10;
end;

begin
ClrScr;
A:=10;writeln(FCE(10)+FCE(A));
A:=10;writeln(FCE(A)+FCE(10));
A:=10;writeln(FCE(10)+FCE(10));
A:=10;writeln(FCE(A)+FCE(A));
end.
```

```
Výsledky:
50
60
50
60
```

Rekurzívní volání

Při volání procedury či funkce se provedou příkazy, které jsou uvedeny v příkazové části procedury či funkce. Pokud se v příkazové části vyskytne volání sama sebe, provádí se procedura či funkce rekurzivně.

Příklad rekurzivního volání:

```
{vypocet faktorialu rekurzivni funkci}
var N:integer;
function FAKT(N:integer):integer;
begin
if N=0 then FAKT:=1 else FAKT:=N*FAKT(N-1)
end;
 begin
   write('zadej N: ');
   read(N);
   if N<0 then writeln('faktorial neexistuje')
     else writeln('faktorial:',N:10,'=',FAKT(N):5)
end.
```

Příklad:

Pomocí rekurzivní procedury proveďme reverzi znaků končících mezerou, například změňme 12345 na 54321.

```
procedure REVERZE;
var ZN:char;
begin read(ZN);
   if ZN<>' ' then REVERZE;
   write(ZN)
end;

begin
writeln;
REVERZE
end.
```

Výsledky:
12345
 54321

Poznámka: Rekurzivní řešení je z hlediska rychlosti výpočtu a paměťové náročnosti únosné pro velmi jednoduché případy. Pro výpočty složitější, například výpočet Fibonacciho čísla je rekurzivní výpočet neefektivní.

Forward, external, inline

Místo bloku procedury či funkce můžeme za hlavičkou procedury či funkce zapsat deklaraci **forward, external, inline.**

Deklarace **forward** říká překladači, že vlastní definice procedury následuje po této deklaraci. Vlastní deklarace procedury nebo funkce pak používá stejný identifikátor procedury či funkce, vynechá však seznam parametrů. Tento způsob se využívá při deklaraci procedury či funkce, která je použita dříve v některé proceduře či funkci. Velmi často se příkazu **forward** můžeme vyhnout pouze přehozením pořadí definice procedur a funkcí.

Příklad:

```
{pouziti forward}
var SUM:integer;

   procedure VSTUP(var X,Y:integer);forward;
procedure VYPOCET(X,Y:integer;var S:integer);
begin
  VSTUP(X,Y);
  S:=X*Y;
  writeln(X:5,Y:5,S:5)
end;

procedure VSTUP;
begin
  write('zadej X,Y ');
  read(X,Y)
end;
begin VYPOCET(1,2,SUM) end.
```

Příklad:{bez pouziti forward}

```
var SUM:integer;

procedure VSTUP(var X,Y:integer);
begin
   write('zadej X,Y ');
   read(X,Y)
end;

procedure VYPOCET(X,Y:integer;var S:integer);
begin
   VSTUP(X,Y);
   S:=X*Y;
   writeln(X:5,Y:5,S:5)
end;
begin
   VYPOCET(1,2,SUM)
end.
```

Deklarace **external** umožňuje připojení procedur a funkcí v jazyce symbolických adres a deklarace **inline** umožňuje průběžně vkládat do textu programy ve strojovém jazyce.

💣 Procedura či funkce v roli parametrů

Typ procedura nebo funkce se může použít v roli parametrů. Můžeme tedy deklarovat procedury nebo funkce, které mají jako parametr jinou proceduru nebo funkci. Využití funkce v roli parametru je zřejmé z následujícího příkladu. Procedura či funkce v roli parametru musí splňovat požadavek kompatibility a musí se překládat s volbou překladače {$F+}.

Příklad:

```
type FCE=function(X,Y:integer):integer;
{$F+}

function SOUCET(X,Y:integer):integer;
begin   SOUCET:=X+Y  end;

function ROZDIL(X,Y:integer):integer;
begin
  ROZDIL:=X-Y
end;

procedure VYPOCET(X,Y:integer;F:FCE);
begin
   writeln(X:5,Y:5,F(X,Y):5)
end;

{$F-}
begin   writeln;
    VYPOCET(1,2,SOUCET);
    VYPOCET(1,2,ROZDIL)
end.
```

Výsledky:

```
  1    2    3
  1    2   -1
```

3.10 Strukturované typy

V předchozích kapitolách byly vysvětleny jednoduché typy.
Pro úplnost je zopakujeme:
standardní typy - celočíselné, reálné, logické, znakové
- řetězec
definované programátorem - výčtový typ, typ interval.
V dalším textu bude věnována pozornost strukturovaným typům, které dělíme na:
- typ pole
- typ záznam
- typ množina
- typ soubor.

S typy proměných pole a soubor jsme se rámcově setkali v dřívějším textu, nyní rozšíříme možnosti použití uvedených proměných.

3.10.1 Typ pole

☞ **var X: array [1..10] of char;**

Pole je homogenní datová struktura skládající se ze složek stejného typu, které se rozlišují pomocí indexu, který musí být ordinálního typu.

Popis typu

Popis typu má základní tvar
array [typ indexu] **of** typ složky

Příklad:

```
var VEKTOR: array [1..3] of integer;
   MATICE: array [1..2,1..3] of real;
   {nebo MATICE: array [1..2] of array [1..3] of real;}
   RETEZEC: array ['I'..'K'] of char;
```

Jednotlivé složky uvedených proměnných zpřístupňujeme zápisem
VEKTOR [1], VEKTOR [2], VEKTOR [3] nabývají hodnotu typu *integer*
MATICE [1,1],MATICE [1,2],MATICE [1,3] nabývají hodnotu typu *real*
MATICE [2,1],MATICE [2,2],MATICE [2,3]
nebo rovněž
MATICE [1][1], MATICE [1][2], atd.
RETEZEC ['I'], RETEZEC ['J'], RETEZEC ['K'] nabývají hodnotu typu *char*

Vektor lze zobrazit následovně

VEKTOR [1] *integer*	VEKTOR [2] *integer*	VEKTOR [3] *integer*

Obdobným způsobem lze zobrazit matici

MATICE [1,1] *real*	MATICE [1,2] *real*	MATICE [1,3] *real*
MATICE [2,1] *real*	MATICE [2,2] *real*	MATICE [2,3] *real*

Označení složek proměných typu pole, tedy zpřístupnění složek, lze uskutečnit pomocí indexu, čímž obdržíme t.zv. indexovou proměnnou. Index musí mít hodnotu, která je stejného typu jako je typ indexu - nejčastěji interval z typu *integer* nebo *char*.

Obecně však může být použit libovolný ordinální typ (přesněji, musí být kompatibilní vzhledem k přiřazení).

Na typ složek se neklade žádné omezení, může to být jednoduchý typ nebo strukturovaný typ.

Příklad:
VEKTOR [2]- složka proměnné VEKTOR je typu *integer* a index je dán konstanta typu *integer*.
VEKTOR [1+I-K]- složka proměnné VEKTOR je typu *integer* a index je dán výsledkem výrazu 1+I-K typu *integer*.

Příkaz přiřazení

Hodnoty proměnných typu pole se obvykle definují a mění postupně tak, že se přiřazují hodnoty jednotlivým složkám.

Příklad:

```
VEKTOR[1]:=10;
VEKTOR[1-I+K]:=10+K*(K-I);
read(VEKTOR[3]);
```

Přiřazovacím příkazem je však možné definovat i všechny specifikované složky proměnné typu pole. Příkaz přiřazení má pak tvar

P:= V,

kde

P a V musí být stejného typu. Lze například použít přiřazení (při deklaraci **var** A,B:**array** [1..3] **of** *real*; a za předpokladu přiřazení hodnot na B)

A:= B, což je ekvivalentní s příkazem A[1]: = B[1];
A[2]: = B[2];
A[3]: = B[3];

Přiřazení hodnoty proměnné typu pole jako celku je možno provést na všech úrovních. Chceme-li nulovat matici, je možno vynulovat první řádek po složkách a další již použitím vynulovaného řádku.

```
var MATICE:array[1..10,1..10] of real;
   N,M,I,J:integer;
begin
write('zadej pocet radku a pocet sloupcu: ');
readln(N,M);
{nulovani matice}
{vynulovani prvniho radku}
for I:=1 to M do
  MATICE[1,I]:=0;
{prirazeni prvniho radku na dalsi radky}
for I:=2 to N do
  MATICE[I]:=MATICE[1];
{tisk matice}
for I:=1 to N do
begin
  writeln;
  for J:=1 to M do
  write(MATICE[I,J]);
end;
end.
```

Program lze samozřejmě napsat i v následujícím tvaru

```
var MATICE:array[1..10,1..10] of real;
   N,M,I,J:integer;
begin
write('zadej pocet radku a pocet sloupcu: ');
readln(N,M);
writeln('zadej prvky matice ');
{nulovani matice}
for I:=1 to N do
for J:=1 to M do
  MATICE[I,J]):=0;
{tisk matice}
for I:=1 to N do
begin
  writeln;
  for J:=1 to M do
  write(MATICE[I,J]);
end;
end.
```

Podobně lze zapsat program pro výměnu řádků v matici. V tomto případě je nutno zavést pomocnou proměnnou, která je téhož typu, jako jsou řádky matice, v našem příkladě např. proměnná POM_RADEK.

Výměnu I-této a J-tého řádku pak provedeme pomocí příkazů

```
POM_RADEK:=MATICE[I];
MATICE[I]:=MATICE[J];
MATICE[J]:=POM_RADEK;
```

```
type TRADEK=array[1..10] of real;
    TMATICE=array[1..10] of TRADEK;
var MATICE:TMATICE;
   POM_RADEK:TRADEK;
   IZ1,IZ2,N,M,I,J:integer;
begin
write('zadej pocet radku a pocet sloupcu: ');
readln(N,M);
writeln('zadej prvky matice ');
for I:=1 to N do
for J:=1 to M do
 read(MATICE[I,J]);
write('zadej cisla radku pro vymenu: ');
readln(IZ1,IZ2);
POM_RADEK:=MATICE[IZ2];
MATICE[IZ2]:=MATICE[IZ1];
MATICE[IZ1]:=POM_RADEK;
for I:=1 to N do  begin   writeln;
  for J:=1 to M do
    write(MATICE[I,J]);
end;
end.
```

Příklad je ekvivalentní zápisu programu ve tvaru následujícího programu

```
tvar MATICE:array[1..10] of array[1..10] of real;
   POM:array [1..10] of real;
   IZ1,IZ2,N,M,I,J:integer;
begin
write('zadej pocet radku a pocet sloupcu: ');
readln(N,M);
writeln('zadej prvky matice ');
for I:=1 to N do
for J:=1 to M do
  read(MATICE[I,J]);
write('zadej cisla radku pro vymenu: ');
readln(IZ1,IZ2);
for J:=1 to M do
begin
 POM[J]:=MATICE[IZ2,J];
 MATICE[IZ2,J]:=MATICE[IZ1,J];
 MATICE[IZ1,J]:=POM[J];
end;
for I:=1 to N do
begin
 writeln;
 for J:=1 to M do
 write(MATICE[I,J]);
end;
end.
```

Definice proměnné typu pole nám umožní uložit do paměti jednotlivé prvky z řady čísel či prvky matice, atd. Vhodnost použití tohoto typu je zřejmá například při řešení střední hodnoty a rozptylu pro X_i, i= 1,2 .., N, jestliže použijeme pro řešení již známé vztahy

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_I - \overline{x})^2$$

Příklad je uveden v kapitole 2.3

Příklad: Sestavme program pro určení maximálního a minimálního prvku z řady x_i, i = 1,2,...,n.

```
  var N,I:integer;
     MIN,MAX:real;
     X:array [1..100] of real;
     begin write('zadej pocet clenu rady N: ');
          read(N);
          write('zadej n clenu rady: ');
          read(X[1]);
          MIN:=X[1];
          MAX:=MIN;
          for I:=2 to N do
          read(X[I]);
          for I:=1 to N do
```

```
            begin if X[I]>MAX then MAX:=X[I];

    if X[I]<MIN then MIN:=X[I]
            end;
      writeln('minimum: ',MIN:10:3);
      writeln('maximum: ',MAX:10:3)
  end.
```

Výsledek:

```
zadej pocet clenu rady N: 5
zadej N clenu rady: 2.1 333.2 3.22 99.3 5.88
minimum: 2.100
maximum: 333.200
```

Příklad: Sestavme program pro určení maximálního prvku ležícího na diagonále matice **A** rozměru *n.n*.

```
var A:array[1..10,1..10] of real;
   I,J,N:integer;
   MAX:real;
begin write('zadej rozmer matice: ');
     read(N);
     writeln('zadej prvky matice:');
     for I:=1 to N do
     for J:=1 to N do
       read(A[I,J]);
     MAX:=A[1,1];
     for I:=2 to N do
       if A[I,I]>MAX then MAX:=A[I,I];
     writeln('max. prvek na diagonale:',MAX:10:2);
end.
```

Výsledek:

```
zadej rozmer matice: 3
zadej prvky matice:
11.22 123.45 22.22
91.22 77.00 39.91
21.10 2.01 66.81
max. prvek na diagonale: 77.00
```

U procedur a funkcí musí být formální parametry typu pole stejného typu jako skutečné parametry. Definice je jasná z následujícího příkladu.

Příklad: Sestavme program pro součet matice **C** = **A** + **B**. Čtení matic **A,B** a součet matic **C** = **A** + **B** proveďme pomocí procedur CTI a SUM.

```
type TA=array [1..10,1..10] of real;
var A,B,C:TA;
  I,J,N:integer;

procedure CTI(var X:TA;N:integer);
var I,J:integer;
begin
writeln('zadej prvky matice oddelene mezerou ci novym
radkem');
 for I:=1 to N do
 for J:=1 to N do
 read(X[I,J])
end;

  procedure SUM(var X,Y,Z:TA;N:integer);
  var I,J:integer;
  begin
   for I:=1 to N do
   for J:=1 to N do
   Z[I,J]:=X[I,J]+Y[I,J]
  end;

begin
    repeat
      write('zadej rozmer ctvercove matice 10>=n>1: ');
      read(N);
    until (N>1) and (N<=10);
    CTI(A,N);
    CTI(B,N);
    SUM(A,B,C,N);
    write('soucet matic');
    for I:=1 to N do
    begin writeln;
        for J:=1 to N do
        write(C[I,J]:10:2);
    end

end.
```

Výsledky

```
zadej rozmer ctvercove matice 10>n>0: 3
zadej prvky matice oddelene mezerou ci novym radkem
10 20 30
-1 -2 -3
 1 2 3
```

```
zadej prvky matice oddelene mezerou ci novym radkem

-1 -2 -3
10 20 30
 8 16 24
soucet matic:
     9.00 18.00 27.00
     9.00 18.00 27.00
     9.00 18.00 27.00
```

💣 Definování hodnot je možno provést pomocí definice konstant s udaným typem, například následovně

1. **const** ZN: **array** [0..5] **of** *char* = ('A', 'B', 'C', 'D', 'E', 'F'); nebo ZN: **array** [0..5] **of** *char* = 'ABCDEF';

2. **type** TM = **array** [0..1, 0..1] of *integer*;
const M:TM = ((0,1), (2,3));
čímž přiřadíme M[0,0]:= 0; M[0,1]:= 1;
M[1,0]:2; M[1,1]:= 3;

3. **type** TDEN = (PO,UT,ST,CT,PA,SO,NE);
TYDEN = array [TDEN] **of string;**
const DEN: TYDEN = ('PO','UT','ST','CT','PA','SO','NE');
čímž přiřadíme DEN [PO]:= 'PO';
DEN [UT] = 'UT';
---

4. **const** MIN = 0;
MAX = 100;
type A = **array** [MIN..MAX] **of** *integer;*
Nelze však použít zápis
const MIN: *integer* = 0;
MAX: *integer* = 100;
type A = **array** [MIN..MAX] **of** *integer;*
neboť v tomto případě jsou MIN a MAX konstanty s udaným typem.

3.10.2 Typ záznam

☞ **var X: record**

A:integer;
B:real
end

Záznam je nehomogenní datová struktura, skládající se z určitého počtu pojmenovaných složek, které mohou být různého typu a které nazýváme položky záznamu. Každá položka je specifikována identifikátorem s udaným typem.

Záznam může mít v jednoduchém případě tvar

PŘÍJMENÍ	JMÉNO	RODNÉ ČÍSLO	PLAT
string	string	string	real

a deklarace má tvar

```
var OSOBA: record
        PRIJMENI: string [10];
        JMENO: string [10];
        RODNE_CISLO: string [8];
        PLAT: real
        end;
```

Přístup k jednotlivým složkám proměnných typu záznam se provádí pomocí selektoru záznamu, jehož zápis obsahuje označení proměnné typu záznam a identifikátor příslušné položky oddělené tečkou. Pro uvedenou deklaraci lze zapsat OSOBA.PRIJMENI OSOBA.JMENO
OSOBA.RODNE_CISLO OSOBA.PLAT

Záznam může obsahovat na místě položky další záznam

PŘÍJMENÍ	JMÉNO	NAROZEN			PLAT
string	string	DEN	MĚSÍC	ROK	real
		1..31	1..12	1900..2000	

deklarace má tvar

```
var OSOBA: record
        PRIJMENI: string;
        JMENO : string;
        NAROZEN : record
                DEN:1..31;
                MESIC:1..12;
                ROK:1900..2000
                end;
        PLAT: real
        end;
```

Pro uvedenou deklaraci lze zapsat (tj. zpřístupnit jednotlivé položky záznamu)
OSOBA.PRIJMENI OSOBA.JMENO OSOBA.NAROZEN.DEN
OSOBA.NAROZEN.MESIC OSOBA.NAROZEN.ROK OSOBA.PLAT

Příkaz with

Používáme-li některou položku záznamu několikrát nebo několik položek téhož záznamu, lze použít zkrácený zápis pomocí **with**. Pro uvedené případy lze použít zápis

read (OSOBA.JMENO, OSOBA.PRIJMENI, OSOBA.PLAT);
nebo lépe
with OSOBA **do** *read* (JMENO,PRIJMENI,PLAT);
with připouští uvedení několika proměnných typu záznam buď ve tvaru

```
with A do            nebo      with A,B  do
 with B do                     příkaz
 příkaz                      end
 end
end
```

Příklad: Pro zadaný textový soubor DATA.DAT určeme maximální plat, jméno a příjmení pracovníka, který má tento plat. POZOR, forma dat je nutno dodržovat, v našem případě 5 znaků pro jméno, 5 znaků pro příjmení, číslo pro plat může být oddělelo mezerami nebo bez mezer.

```
uses Crt;
var OSOBA:record
     PRIJMENI,JMENO:string[5];
     PLAT:real;
     end;
     F:text;
     MAX_PLAT:real;
     MAX_PRIJMENI,MAX_JMENO:string[5];

begin
ClrScr;
assign(F,'c:\DATA.DAT');
reset(F);
MAX_PLAT:=0;
while not eof(F) do
with OSOBA do
begin
readln(F,JMENO,PRIJMENI,PLAT);
if PLAT>MAX_PLAT then
    begin
    MAX_PLAT:=PLAT;
    MAX_PRIJMENI:=PRIJMENI;
    MAX_JMENO:=JMENO;
    end;
end;
writeln('Maximalni plat ',MAX_PLAT:7:2,
       ' ma ',MAX_JMENO,MAX_PRIJMENI);
readln;
end.
```

```
Datový soubor DATA.DAT má tvar:
jm1  pr1xx3000
jm2  pr2  5000
jm3xxpr3xx 2000
jm4xxpr123456
```

```
Výsledky:
Maximalni plat 5000.00 ma jm2 pr2
```

S potížemi se můžeme setkat v případě, že je uvedeno nejdříve číslo a pak text.

```
uses Crt;
var  OSOBA : record
           PRIJMENI,JMENO : string[5];
             PLAT,CISLO:real;
             end;
BEGIN
   ClrScr;
   with OSOBA do
   begin
```

```
    read(CISLO,JMENO, PRIJMENI,PLAT);
    write(CISLO,JMENO,PRIJMENI,PLAT);
   end;
END.
```

```
data1:
333 aaaabbbbb123
Vysledek:
 3.33E2 aaaabbbbb 1.23E2
data2:
333aaaaabbbbb123
```

```
Vysledek:
Error 106: Invalid numeric format.
```

Řešení je možné rovněž následujícím postupem

```
uses Crt;
var  OSOBA : record
          PRIJMENI,JMENO : string[5];
                zn:string[1];
           PLAT,CISLO:real;
           end;
BEGIN
   ClrScr;
   with OSOBA do
   begin
   read(CISLO,zn,JMENO, PRIJMENI,PLAT);
         write(CISLO,zn,JMENO,PRIJMENI,PLAT);
   end;
end.
```

```
data:
333 aaaaabbbbb123
Vysledek:
3.33E2 aaaaabbbbb 1.23E2
```

Záznam může být použit také jako prvek pole. Použití je zřejmé z následujícího příkladu pro určení amplitudy komplexního čísla pro známé reálné a imaginární složky $X_{Re}(i)$, $X_{Im}(i)$, i=1,N.

Tvar záznamu:

```
XRE[1]  XIM[1]
XRE[2]   XIM[2]
XRE[3]  XIM[3]
------       ------
XRE[50]  XIM[50]
```

Příklad:

```
{amplituda komplexniho cisla}
type T=record REX:real;
            IMX:real
      end;
var X:array[1..50] of T;
   I,N:integer;
   C:array [1..50] of real;
begin
read(N);
for I:=1 to N do
begin read(X[I].REX,X[I].IMX);
     C[I]:=sqrt(sqr(X[I].REX)+sqr(X[I].IMX));
     writeln(X[I].REX:12:2,X[I].IMX:12:2,
           C[i]:12:2)
end;

{nebo jinak}
for I:=1 to N do
begin with X[I] do
           begin read(REX,IMX);
               C[I]:=sqrt(sqr(REX)+sqr(IMX));
               writeln(REX:12:2,IMX:12:2,C[I]:12:2)
           end
end
end.
```

Výsledky:

```
1
3 4
      3.00    4.00    5.00
3 4
      3.00    4.00    5.00
```

Definování hodnot je možné pomocí definice konstant s udaným typem například ve tvaru

```
type TDEN = record
          DNY:1..31;
          MESICE:1..12;
          ROKY:1..2000;
          end
const DEN:TDEN = (DNY:2; MESICE:3; ROKY:1992);
```

čímž přiřadíme

```
DEN.DNY:=2;
DEN.MESICE:=3;
DEN.ROKY:=1992;
```

Variantní záznam

V některých případech je užitečný tzv. variantní záznam, kdy považujeme několik typů za různé varianty téhož typu. Uvažujme záznam ve tvaru

<table>
<tr><td rowspan="2">A</td><td rowspan="2">P</td><td>B1</td><td>B2</td><td></td></tr>
<tr><td>C1</td><td>C2</td><td>C3</td></tr>
</table>

kdy po položce A a tzv. rozlišovací položce následuje buď B1, B2 nebo C1, C2, C3.

Popis typu má tvar

```
var VETA: record
        A: string [5];
        case P:O..1 of
        O: (B1:integer;
           B2:integer);
        1: (C1:integer;
           C2:integer;
           C3:real)
       end;
```

Pak pro VETA.P = 0 jsou uvažovány položky A, B1, B2
VETA.P = 1 jsou uvažovány položky A, C1, C2, C3

Pevná část variantního záznamu specifikuje položky, které se vyskytují ve všech variantách, pro náš případ položka A. Tato část může chybět. Za pevnou částí následuje rozlišovací položka, jejíž identifikátorem je P a jejíž typ je 0..1. Dále je uveden seznam, jímž se specifikují jednotlivé varianty této části záznamu. Specifikace každé varianty má tvar

O: (B1, B2: *integer*);
1: (C1, C2: *integer*; C3:*real*);

a definuje jednu variantu, jejíž struktura je dána uvedeným seznamem položek a která je aktuální tehdy, když hodnota rozlišovací položky se rovná některé z rozlišovacích konstant uvedených v této specifikaci.

Příklad: Podle pohlaví určeme nejdůležitější atributy - plat (u mužů) a rozměry (u žen) pro zadaný datový soubor DATA.DAT.

```
uses Crt;
type TMIRY=(PRSA,PAS,BOKY);
var OSOBA:record
    PRIJMENI,JMENO,VEK:string[5];
    case POHLAVI:char of
    'M':(PLAT:real);
    'Z':(MIRY:array[TMIRY] of integer);
    end;
F:text;
begin
ClrScr;
assign(F,'DATA.DAT');
reset(F);
while not eof(F) do
with OSOBA do
begin
read(F,JMENO,PRIJMENI,POHLAVI);
```

```
    case POHLAVI of
    'M': begin
        readln(F,PLAT);
        write(JMENO,PRIJMENI);
        if PLAT>10000 then writeln(' je O.K.')
                    else writeln('   neni O.K.');
        end;
    'Z': begin
        readln(F,MIRY[PRSA],MIRY[PAS],MIRY[BOKY]);
        write(JMENO,PRIJMENI);
        if (MIRY[PRSA]>85) and (MIRY[BOKY]>90)
        and (MIRY[PAS]<60)
           then writeln(' je O.K.')
           else writeln(' neni O.K');
        end;
    end;
    end;
    end.
```

```
JM1  PR1  M 1100
JM2  PR2  Z 86 59 91
JM3  PR3  Z 87 65 87
JM4  PR4  M 5999
```

Výsledky:

```
JM1  PR1  je O.K.
JM2  PR2  je O.K.
JM3  PR3  není O.K.
JM4  PR4  není O.K.
```

3.10.3 Typ množina

☞ **var C: set of char;**

Datové objekty, jejichž hodnotami jsou množiny, specifikujeme jako typ množina.
Popis typu množina má tvar

set of bázový typ

kde bázový typ může být pouze označení ordinálního typu pomocí identifikátoru typu nebo popisem typu.

Příklad:

```
var N,M: set of 0..9;
   ZN1: set of char;
   ZN2: set of 'A'.. 'Z';
```

Příklad:

```
type TDEN =(PONDELI,UTERY,STREDA,CTVRTEK,PATEK,SOBOTA,NEDELE);
var  DNY: set of TDEN;
```

Definování hodnot je možno přiřazením, definicí nepřímé konstanty nebo definicí konstanty s uvedeným typem

1. příkaz přiřazení

```
N:= [0,1,2,3,4]; nebo N:=[0..4];
ZN:=['A','B','C','D','X','Y','Z'];nebo ZN:=['A'..'D','X'..'Z'];
DNY:=[PONDELI..NEDELE];
```

2. denificí nepřímé konstanty

```
const N = [0..4];
    ZN = ['A'..'Z'];
    DNY = [PONDELI..NEDELE];
```

3. definicí konstanty s udaným typem

```
const N: set of 0..9 = [0..4];
nebo
type TN = set of 0..9;
const N:TN=[0..4];
const ZN: set of 'A'..'Z'=['A'..'D','X'..'Z'];
nebo
type TZN=set of 'A'..'Z';
const ZN: TZN=['A'..'Z'];
const DNY: set of (PONDELI, UTERY, STREDA, CTVRTEK, PATEK,
        SOBOTA, NEDELE) = [PONDELI..PATEK];
nebo
type TDNY = (PONDELI, UTERY, STREDA, CTVRTEK, PATEK, SOBOTA,
         NEDELE);
const DNY: set of TDNY = [PONDELI .. PATEK];
```

U množin jsou definovány obvyklé množinové a relační oparace (operandy musí být typu množina). Uvažujeme proměnné N a M, které mají přiřazeny hodnoty M: [0,2,3,4,6]; N: = [4,6,8,9].

operace:	výsledek:
+ sjednocení	M + N = [0,2,3,4,6,8,9]
* prunik	M * N = [4,6]
- rozdíl	M - N = [0,2,3]

Výsledná hodnota uvedených operací je typu množina

operace:	výsledek:
= rovnost	M = N *false*
<> nerovnost	M <> N *true*
<= je obsaženo v	M <= N *false*
>= obsahuje	M >= N *false*

Typ výsledku relačních operátorů je *boolean*.

in relace "je prvkem množiny" 5 **in** M	*false*

Prvním operandem je výraz ordinálního typu a druhým je množina s kompatibilním typem báze. Uvedený příklad lze rovněž zapsat pomocí (5=0) **or** (5=2) **or** (5=3) **or** (5=4) **or** (5=6)

Pro následující deklarace

```
type DEN = (PONDELI, UTERY, STREDA, CTVRTEK, PATEK, SOBOTA,
NEDELE)
var DEN_P, DEN_V, TYDEN: set of DEN;
   DEN_N:DEN;
```

lze proměným přiřadit například následující hodnoty

```
DEN_N:=SOBOTA;
DEN_V:=[SOBOTA..NEDELE];
TYDEN:=[PONDELI..NEDELE];
```

a lze použít následující příkaz

if DEN_N **in** DEN_V **then** *write* ('VOLNÝ DEN') **else** ('PRACOVNÍ DEN');

Příklad:

Určeme počet výskytů znaků A,B,C,D,E v souboru DATA.DAT. Datový soubor má tvar našeho programu.

```
{pocet vyskytu pismen A az E}
var TAB:array ['A'..'E'] of integer;
   ZN:char;
   PISMENA:set of 'A'..'E';
   F:text;
begin
    assign(F,'A:DATA.DAT');
    reset(F);
    for ZN:='A' to 'E' do
    TAB[ZN]:=0;
    PISMENA:=['A'..'E'];
    while not eof(F) do
    begin
       read(F,ZN);
      if ZN in PISMENA then
           TAB[ZN]:=TAB[ZN]+1
    end;
writeln('ZNAK: POCET VYSKYTU:');
for ZN:='A' to 'E' do
writeln(ZN:3,TAB[ZN]:7)
end.
```

Program lze upravit rovněž tak, že místo proměnné PÍSMENA typu množina použijeme konstantu typu množina. Program má pak následující tvar:

```
{pocet vyskytu pismen A az E}
var TAB:array ['A'..'E'] of integer;
     ZN:char;
     F:text;
const PISMENA=['A'..'E'];
begin
    assign(F,'A:DATA.DAT');
    reset(F);
    for ZN:='A' to 'E' do
    TAB[ZN]:=0;
    while not eof(F) do
    begin
       read(F,ZN);
      if ZN in PISMENA then
           TAB[ZN]:=TAB[ZN]+1
    end;
writeln('ZNAK: POCET VYSKYTU:');
for ZN:='A' to 'E' do
writeln(ZN:3,TAB[ZN]:7)
end.
```

Jestliže nechceme použít proměnné či konstanty typu množina, program lze zapsat pomocí podmíněného příkazu **if** a pak má program následující tvar:

```
{pocet vyskytu pismen A az E}
var TAB:array ['A'..'E'] of integer;
   ZN:char;
   F:text;
begin
    assign(F,'A:DATA.DAT');
    reset(F);
    for ZN:='A' to 'E' do
    TAB[ZN]:=0;
    while not eof(F) do
    begin
       read(F,ZN);
      if (ZN>='A') and (ZN<='E') then
           TAB[ZN]:=TAB[ZN]+1
    end;
writeln('ZNAK: POCET VYSKYTU:');
for ZN:='A' to 'E' do
writeln(ZN:3,TAB[ZN]:7)
end.
```

Příklad:

Určeme stejné znaky prvního a druhého řádku. Řádky dat zapisujeme na klávesnici, každý řádek ukončíme známým znakem konce řádku (cr-lf)

```
type MNOZINA=set of char;
var P1,P2,PV:MNOZINA;
   ZN:char;
begin
P1:=[ ];{prazdna mnozina}
repeat
  read(ZN);
  P1:=P1+[ZN];{mnozina znaku prvniho radku}
until eoln;
P2:=[];{prazdna mnozina}
repeat
  read(ZN);
  P2:=P2+[ZN];{mnozina znaku druheho radku}
until eoln;
PV:=P1*P2;{prunik znaku prvniho a druheho radku}
for ZN:=#32 to #255 do
if ZN in PV then write(ZN);
end.
```

3.10.4 Typ soubor

Se základy použití souborů jsme se již seznámili v úvodu skript a textové soubory byly využívány v mnoha příkladech. Nyní se seznámíme s dalšími možnostmi souborů.

Soubor je posloupnost složek, které jsou stejného typu. Složky mohou být jakéhokoliv typu, mimo soubor. Na rozdíl od pole a záznamu, počet složek v souboru není pevný. Soubory jsou obvykle ukládány na vnější paměť (harddisk,disketa) a pouze jedna složka souboru je přímo přístupná v daném okamžiku. Další složky jsou přístupné sekvenčním zpracováním.

Deklarace souboru

Soubory jsou specifikovány jako soubory
- s udaným typem
- bez udání typu
- textové soubory

Příklad:

```
type TRECORD = record
          JM: string;
          PR: string;
          PLAT: real;
          end;
   TSOUBOR = file of TRECORD;
var       SOUBOR: TSOUBOR {s udaným typem}
          VSTUP : text;   {textový soubor}
          TEST : file;    {bez udání typu}
```

Každý soubor musí být před vytvářením nebo čtením otevřen pomocí
Reset (f); *Rewrite* (f);

Příkaz Reset (f) otvírá již exitující soubor pro čtení, Rewrite (f) vytváří nový, prázdný soubor. Jestliže již existoval, je zrušen.

f musí být typu **file of**.., či *text* a musí mu být přiřazeno jméno pomocí příkazu
Assign (f,JM);
Jméno JM může být

- " standardní vstup či výstup
-'JMENO SOUBORU' - konstanta typu **string**
- JMENO_SOUBORU - identifikátor proměnné typu **string**
- 'A:\DNAME1\DNAME2\JMENO_SOUBORU' - disková jednotka A:, adresář DNAME1, DNAME2 a jméno souboru JMENO_SOUBORU
- 'LPT1' či 'PRN' - výstup na tiskárnu.

Poznámka:

Standardní textový soubor input a output je otevřen resp. uzavřen automaticky při spuštění resp. při ukončení programu a nemusí se uvádět v příkazech čtení a výstupu. Tedy následující příkazy jsou ekvivalentní

read (*input*, I);	*read* (I);
write (*output*, I);	*write* (I);

Vstup a výstup se provádí pomocí příkazů *read, write,* pro typ *text* a **file of** .., pro typ **file** příkazy *BlockRead* a *BlockWrite*. Po ukončení operací vstupu a výstupu může být soubor uzavřen příkazem *Close* (f).

Standardní procedury a funkce pro všechny typy souborů:

1. *Assign*, *Reset, Rewrite, Close*

Erase (f) - vymaže externí soubor.

Rename (f, NOVE_JMENO) -souboru spojenému s f je přiřazeno nové jméno zadané na NOVE_JMENO (typu **string**).

RmDir('d:\Adr') - zruší adresář, specifikovaný jako parametr typu **string** na diskové jednotce d: a adresáři \Adr.

GetDir (d,Adr) - aktuální adresář jednotky zadané na d (typu *byte*), kde 0 značí aktuální adresář, 1 jednotku A, 2 jednotku B, atd.,je uložen na Adr (typu **string**).

MkDir ('d:\Adr') - vytvoří adresář Adr.

ChDir ('d:\Adr') - změna aktivního adresáře na nový adresář Adr.

2. *Eof* (f) vrací hodnotu *false*, nebyl-li přečten *Eof* a hodnotu *true*, byl-li nalezen konec souboru.

IOResult - výsledkem je číslo *integer*, označující stav operace vstup,výstup. Při {$I-} je výsledek 0 pro úspěšné operace vstup, výstup, jinak je výsledkem číslo chyby.

Příklad:

MkDir ('A:\D1\D2')

Příklad:

```
{$I-} {vypnutí kontroly chyb při vstupu a výstupu}
MkDir (ParamStr (1));
if IOResult <> O then writeln ('Chyba I/O');
{$I+} {zapnutí kontroly chyb při operaci vstup a výstup}
```

Program použijeme například následovně:

C:\TURBO\NONAME01.EXE A:\D1\D2

Pokud použijeme více paramatrů, oddělíme je mezerou.

Textové soubory

Textové soubory jsou uvažovány jako sekvence znaků organizovaných v řádcích, kde každý řádek je ukončen znakem eoln (<=). Příkazy *read* a *write* povolují hodnoty, které nejsou typu *char*, na tyto se automaticky převedou. V souboru jsou tedy data vždy uložena jako typ *char*. Například při zápisu *write* (f,I), kde I je typu *integer* se uloží hodnota převedená na typy *char*, při čtení

read (f,I) se opět typy *char* převedou na typ *integer*.

Při čtení a zápisu textových souborů se používájí následující procedury a funkce (nejsou uvedeny procedury a funkce, které byly již popsány)

Append(f) -otvírá již existující soubor pro připojení dalších řádků. Byl-li již soubor otevřen, uzavře se a znovu otevře.

SetTextBuf(f,buf,Rozměr) - rozměr standardní délky "buf" (vyrovnávací paměť) je 128 bytů. V případě, že žádáme rozšíření "buf" (dochází k redukci pohybu diskové hlavy), uvedeme požadovaný maximální počet bytů v "buf". Nový rozměr zůstává v platnosti, pokud neuvedeme příkaz *Assign*. V proměnné Rozměr specifikujeme skutečný počet bytů v "buf" v bytech. Není-li uveden, pak se uvažuje délka udaná *SizeOf* (buf).

Příklad:

```
{kopirovani souboru DATA_R.DAT na DATA_W.DAT}
uses Crt;
var F_R,F_W:text;
   ZN:char;
   BUF:array[1..2048] of char;
begin
ClrScr;
assign(F_R,'c:\DATA_R.DAT');
```

```
SetTextBuf(F_R,BUF);
reset(F_R);
assign(F_W,'c:\DATA_W.DAT');
rewrite(F_W);
repeat
  Read(F_R,ZN);
  Write(F_W,ZN);

Write(output,ZN);
until Eof(F_R);

close(F_R);
close(F_W);
end.
```

Poznámka:
Pro výstup na tiskárnu lze použít několik možností zápisu

a)

```
var Lst:text;
  begin
  Assign (Lst, 'LPT'); {nebo 'PRN'}
  rewrite (Lst);
  write (Lst, 'tisk na tiskárně');
  Close (Lst)
  end.
```

b)

```
  uses Printer; {unit Printer}
  begin
  write (Lst, 'tisk na tiskárně');
  end.
```

c) Pro výstup na tiskárnu nebo obrazovku lze použít zápis

```
Uses Crt;
  var F:text;
      TISK: char;
begin  writeln ('TISK NA TISKARNU A/N-jinak monitor-') ;
    read (TISK);
    if (TISK='A')  or  (TISK='a') then assign (F,'PRT')  else assignCrt (F);
    rewrite (F);
    writeln ('zde je výstup');
    close (F);
end.
```

Příklad: Sestavme program pro vytvoření externího textového souboru SOUBOR. Vstupní data čtěme z klávesnice. V daném programu vytvořme úsek pro výstup externího textového souboru SOUBOR na obrazovku monitoru.

```
{vytvoreni souboru z klavesnice na disk}
{vystup souboru z disku na monitor}
```

```
type TPOLE=array[1..100] of real;
var X,Y:TPOLE;
   I,N:integer;
   SOUBOR:string[255];
   FFILE:text;
begin
{vytvoreni souboru na disk}
write('zadej jmeno souboru: ');
read(SOUBOR);
readln;
assign(FFILE,SOUBOR);
rewrite(FFILE);
repeat
   write('zadej N>2 a N<=100: ');
   read(N);
until (N>2) and (N<=100);
writeln('zadej souradnice (oddelovac mezera   nebo novy  radek)');
for I:=1 to N do
begin
   read(X[I],Y[I]);
   writeln(FFILE,X[I],Y[I]);
end;
close(FFILE);
{cteni souboru na disku}
write('zadej jmeno souboru ');
readln;readln(SOUBOR);
assign(FFILE,SOUBOR);
reset(FFILE);
while not eof(FFILE) do
begin
   readln(FFILE,X[I],Y[I]);
   writeln(X[I],Y[I]);
end;
close(FFILE);
end.
```

```
Výsledky:
zadej jmeno souboru: DATA
zadej N>2 a N<100: 3
zadej souradnice(oddelovac mezera nebo novy radek)
1 1
2 2
3 3
soubor DATA
 1.0000000000e+01 1.0000000000e+01
 2.0000000000e+01 2.0000000000e+01
 3.0000000000e+01 3.0000000000e+01
```

Soubory bez udání typu

Tento typ souboru slouží k přímému přístupu k souborům na disku bez ohledu na typ a strukturu.Příkazy *Reset* a *Rewrite* obsahují dodatečný parametr, který udává délku věty. Standardní délka věty, je-li parametr vynechán je 128 bytů, vhodnější je však užívat délku 1byte, tj.*Reset* (f,1) a *Rewrite* (f,1). Délka věty nesmí být větší,než 64 Kb, tj. 65535 bytů.

Pro soubory bez udání typu lze použít následující procedury a funkce:

FilePos(f) - udává pozici ukazatele v souboru. Pro počátek = 0, pro konec souboru je rovno *FileSize* (f) a *Eof* (f)=true.

FileSize (f) - udává velikost souboru v bytech.

Seek (f, N) resp. *Seek* (f, *FileSize*(f)) - nastaví ukazatel na pozici záznamu N resp. na konec souboru.

Truncate(f) - zrušení všech záznamů za ukazatelem pozice v souboru, ukazatel pak v poloze konec souboru a *Eof* (f) = *true*.

BlockRead (f,buf,N,NR);

buf - je netypový parametr.

N - počet bytů vyrovnávací paměti, nejlépe podat pomocí *SizeOf*(buf).

Funkce *SizeOf* (argument) vrací počet bytů obsazených argumentem.

NR - počet skutečně přečtených záznamů. Tento parametr je volitelný.

BlockWrite (f, Buf, N, NW);

f, Buf - význam je obdobný jako u *BlockRead*.

N - součet přečtených záznamů.

NW - počet skutečně zapsaných vět. Po ukončení zápisu je NW rovno nule.

Příkaz *BlockWrite* čte N nebo méně vět ze souboru f do paměti, počínaje prvním bytem v Buf. Počet skutečně přečtených vět, který může být menší nebo roven hodnotě v N je uložen na NR. Parametr NR je volitelný, a po přečtení celého souboru je roven nule.

Příklad: Překopírujme soubor DATA_R.DAT na nový soubor DATA_W.DAT. Kopírovaný soubor zobrazujme při kopírování na obrazovce.

```
{kopirovani souboru DATA_R.DAT na DATA_W.DAT}
uses Crt;
var F_R,F_W,F_M:file;
   N_R,N_W:word;
   BUF:array[1..2048] of char;
begin
ClrScr;
assign(F_R,'c:\DATA_R.DAT');
reset(F_R,1);
assign(F_W,'c:\DATA_W.DAT');
rewrite(F_W,1);
assign(F_M,ParamStr(2));
rewrite(F_M,1);

repeat
  BlockRead(F_R,BUF,SizeOf(BUF),N_R);
  BlockWrite(F_W,BUF,N_R,N_W);
  BlockWrite(F_M,BUF,N_R,N_W);
until (N_R<>N_W) or (N_R=0);
close(F_R);
close(F_W);
close(F_M);
end.
```

DATA:
Datový soubor DATA_R.DAT
má tvar:

```
111 222 333
444
555
666
```

Výsledek:
Datový soubor DATA_W.DAT má tvar souboru DATA_R.DAT.

Příklad: Vytvořme soubor DATA_W.DAT z dat, zadaných na klávesnici.Počet čtených vět je zadán na proměnnou N.

```
{kopirovani souboru z klavesnice na DATA_W.DAT}
{a doplneni souboru DATA_W.DAT }
uses Crt;
var F_R,F_W,F_M:file;
   N_R,N_W,N,I:word;
   BUF:array[1..2048] of char;
begin
ClrScr;
assign(F_R,paramstr(1));
reset(F_R,1);
assign(F_W,'c:\DATA_W.DAT');
rewrite(F_W,1);
assign(F_M,ParamStr(2));
rewrite(F_M,1);
write('zadej pocet vet: ');
read(N);
for I:=1 to N do
 begin
 BlockRead(F_R,BUF,SizeOf(BUF),N_R);
 BlockWrite(F_W,BUF,N_R,N_W);
 BlockWrite(F_M,BUF,N_R,N_W);
 end;
writeln('delka souboru',filesize(F_W));
seek(F_W,filesize(F_W));
writeln('pozice ukazatele',filepos(F_W));
write('doplneni souboru:');
 BlockRead(F_R,BUF,SizeOf(BUF),N_R);
 BlockWrite(F_W,BUF,N_R,N_W);
 BlockWrite(F_M,BUF,N_R,N_W);
close(F_R);
close(F_W);
close(F_M);
end.
```

DATA:

```
111 222 333
444
555
666
delka souboru: 28
doplneni souboru: 999
```

Výsledek:
111 222 333
444
555
666
999

Soubory s udaným typem

Tento typ souboru slouží k vytvoření zadaného typu souboru, který se vytváří v binárním tvaru. Na rozdíl od textového souboru a souboru bez udaného typu, nelze tento soubor vytvářet zvoleným editorem, ale pouze pomocí zápisu čtených hodnot např. v následujícím tvaru

Příklad:

```
vytvoreni souboru z klavesnice na DATA.DAT}
{a doplneni souboru DATA.DAT}
uses crt;
type TOSOBA=record
        JMENO:string[3];
        ADRESA:string[3];
        PLAT:integer;
        end;
var F:file of TOSOBA;
    OSOBA:TOSOBA;
    I,N:integer;
begin
clrscr;
assign(F,'c:\DATA.DAT');
rewrite(F);
writeln('zadej pocet zaznamu: ');
readln(N);
writeln('zadej ',n:2,' zaznamu');
for I:=1 to N do
begin
  readln(OSOBA.JMENO,OSOBA.ADRESA,OSOBA.PLAT);
  write(F,OSOBA);
end;
reset(F);
while not eof(F) do
begin
  read(F,OSOBA);
  writeln(OSOBA.JMENO,OSOBA.ADRESA,' ',OSOBA.PLAT);
end;
writeln('delka souboru: ',FileSize(F));
seek(F,FileSize(F) div 2);
writeln('pozice v souboru: ',FilePos(f));
readln(OSOBA.JMENO,OSOBA.ADRESA,OSOBA.PLAT);
write(F,OSOBA);
reset(F);
while not eof(F) do
begin
  read(F,OSOBA);
```

```
  writeln(OSOBA.JMENO,OSOBA.ADRESA,' ',OSOBA.PLAT);
end;
close(F);
end.
```

DATA

```
Zadej pocet zaznamu:
4
Zadej 4 zaznamy:
J1  A1  1
J2  A2  2
J3  A3  3
J4  A4  4
```

```
Výsledky:
J1  A1  1
J2  A2  2
J3  A3  3
J4  A4  4
Data:
delka souboru: 4
pozice v souboru: 2
JN  AN  999
Výsledky:
J1 A1  1
J2  A2  2
JN  AN  999
J4  A4  4
```

Vytvořený soubor DATA.DATje v binárním tvaru:

```
♥J1  ♥A1  ♦♥J2  ♥A2  ♦♦JN  ♥AN  M♥J4  ♥J4  ♥A4  ☼
```

Jak je zřejmé z uvedeného příkladu, lze použít pro tento typ souboru následující příkazy (s významem obdobným, jako u souboru bez uvedeného typu)

FileSize (f) - udává rozměr souboru
Seek (f,N) - nastaví ukazatel na konec N-tého záznamu
Truncate (f) - zrušení všech záznamů za ukazatelem pozice v souboru.
FilePos (f) - udává pozici ukazatele v souboru.

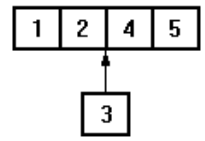
4. Dynamické datové struktury

4.1 Sekvenční a dynamická datová struktura

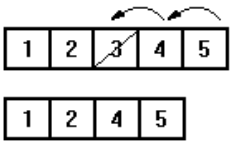
Dosud byly popsány proměnné, jejichž struktura byla určena definicí či deklarací a při provádění programu se již neměnila. Tyto struktury, jejichž obsah se během své existence nemění, jsou statické datové struktury. Při řešení některých úloh je však vhodnější použití dynamické datové struktury, jejíž rozsah se během výpočtu mění. Typickým příkladem je úloha přidání či zrušení prvku v seznamu zobrazeném v následujících obrázcích.

přidání prvku:

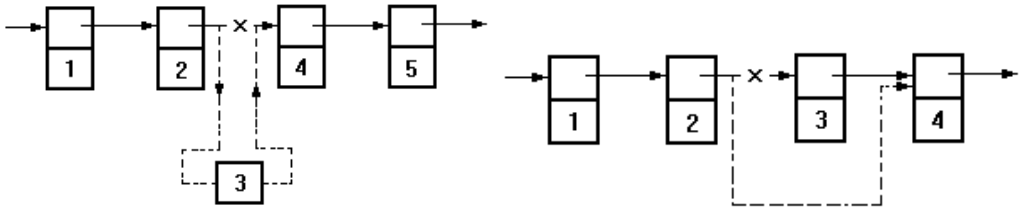
a) sekvenční metoda



zrušení prvku:



b) dynamická datová struktura - použití ukazatelů

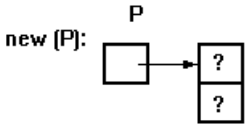


Při sekvenčním způsobu při umístění nového prvku je nejdříve nutné uvolnit příslušné složky doprava a na uvolněné místo přiřadit požadovanou hodnotu. Obdobná situace nastává při zrušení prvku v seznamu.

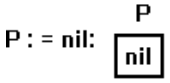
Uvedené operace se efektivněji realizují použitím ukazatelů, kdy každý prvek seznamu je uložen v samostatném objektu, který kromě hodnoty prvku seznamu obsahuje také ukazatel na další prvek seznamu.

Je nutné vytvořit objekt, který obsahuje hodnotu či hodnoty a ukazatel či ukazatele na jiné objekty. Dynamická proměnná není zavedena deklarací proměnných, ale vzniká provedením speciálního příkazu. Dynamickou proměnnou identifikujeme pomocí ukazatele, což můžeme považovat za abstrakci adresy umístění dynamické proměnné v paměti.

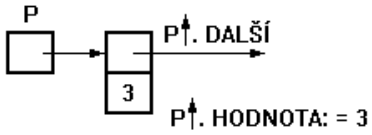
Dynamické proměnné a ukazatele se vytváří pomocí standardní procedury *new*.



Zvláštní hodnotou, která neidentifikuje žádnou dynamickou proměnnou a může být přiřazena libovolné proměnné typu ukazatel je hodnota označená klíčovým slovem *nil*.



Uvažujeme nyní následující schéma



Pro dané schéma může být zavedena následující deklarace a příkazy

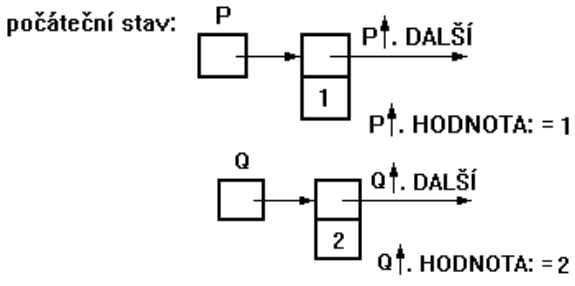
```
type   TUKAZATEL=↑BAZ_TYP;
       BAZ_TYP=record
                  HODNOTA: integer;
                  DALSI: TUKAZATEL;
                  end;
var    P:TUKAZATEL;
begin  new (P);
       P ↑.HODNOTA:=3;
 .....
end.
```

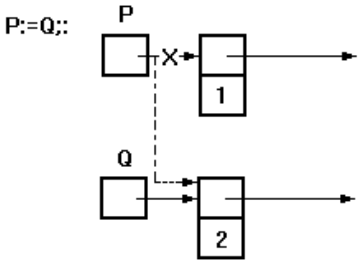
P ↑ je označení dynamické proměnné, kterou identifikuje hodnota proměnné P typu ukazatel. V našem případě tedy můžeme definovat hodnotu v záznamu HODNOTA jako P↑.HODNOTA:=3; a ukazatel P ↑.DALSI:=nil či ukazovat na jiný objekt.

Uvažujeme nyní následující deklaraci

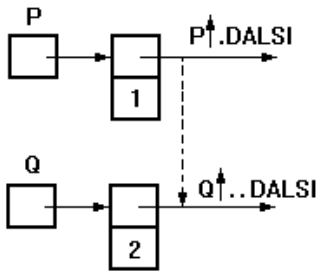
```
type   TZAZNAM=↑ZAZNAM;
       ZAZNAM =   record HODNOTA : 1..10;
                         DALSI : TZAZNAM;
                  end;
var P, Q: TZAZNAM;
```

Pak proměnné typu ukazatel P a Q může být přiřazena hodnota jiné proměnné téhož typu. Předpokládejme, že hodnoty v záznamu P ↑.HODNOTA: = 1; Q ↑.HODNOTA: = 2; a dynamické proměnné a ukazatele byly vytvořeny příkazem *new* (P); *new* (Q); a vycházíme vždy z počátečního stavu.

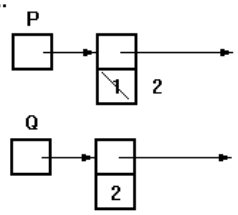




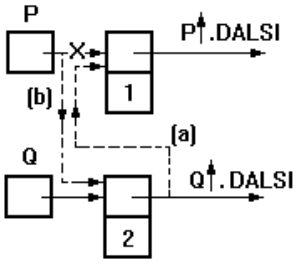
P↑.DALSI: = Q↑.DALSI;:



P↑.HODNOTA: = Q↑.HODNOTA;:



Q↑.DALSI: = P; (a):
P:= Q; (b):



Proměnné, ktreré jsou téhož typu ukazatel mohou být použity jako operandy relačních operátorů = a <>. Operandem v relaci může být také hodnota nil, tedy například možno použít zápis

while (P↑.DALSI <> nil) **and** (P=Q) **do**...

4.2 Seznam

Nejčastější dynamickou datovou strukturou je seznam (list). Uveďme nyní některé typické operace, které se při zpracování seznamů používají. Seznam se skládá z dynamických proměnných typu ZAZNAM, pro jejichž identifikaci používáme ukazatel typu UKAZ.

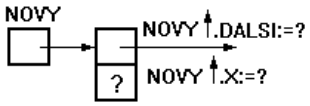
```
type   HODNOTA=1..10;
       UKAZ=↑ZAZNAM
       ZAZNAM=record   X:HODNOTA;
                       DALSI:UKAZ;
                 end;
var    SEZNAM, NOVY: UKAZ;
       I:1..10;
       UKAZATEL, POM, ZACATEK: UKAZ;

begin {vytvoření seznamu}
       SEZNAM:= nil;
       for I:=10 downto 1 do
       begin  new (NOVY);
              NOVY↑.X:=I;
              NOVY↑.DALSI:=SEZNAM;
              SEZNAM:=NOVY;
       end;
```

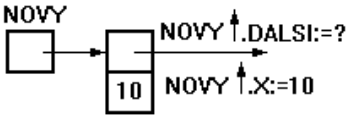
schéma vytvoření seznamu:

SEZNAM:=nil;
I:=10;

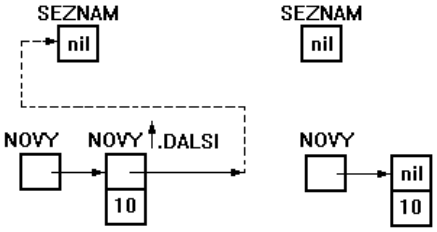
new (NOVY);



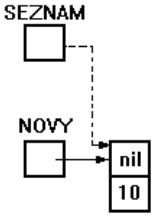
NOVY↑.X:=10;



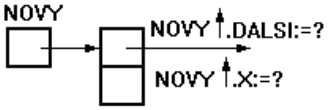
NOVY↑.DALSI:=SEZNAM



SEZNAM:=NOVY



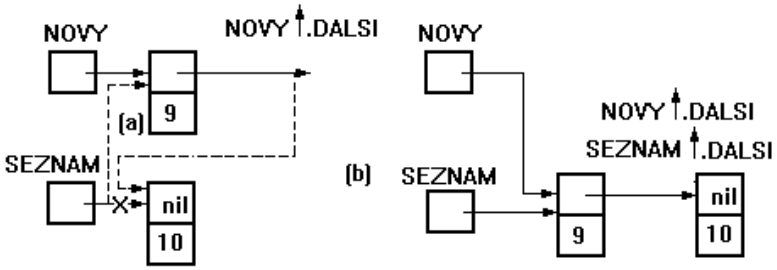
I:=9
new (NOVY)



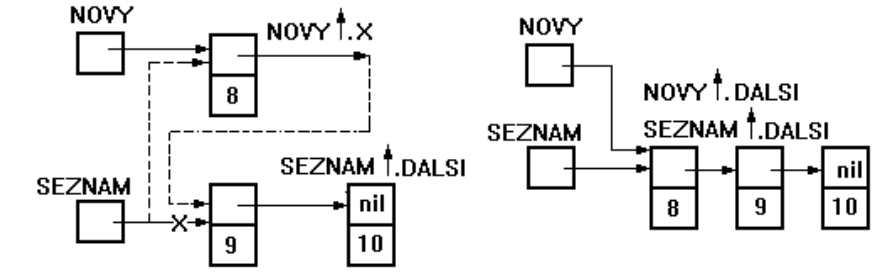
NOVY↑. X:=9

NOVY↑.DALSI:=SEZNAM (a)

SEZNAM:=NOVY (b)

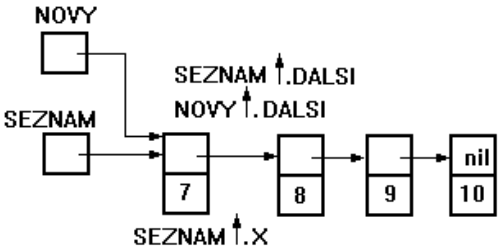


```
I:=8
new (NOVY)
NOVY↑.DALSI:=SEZNAM
SEZNAM:=NOVY
```

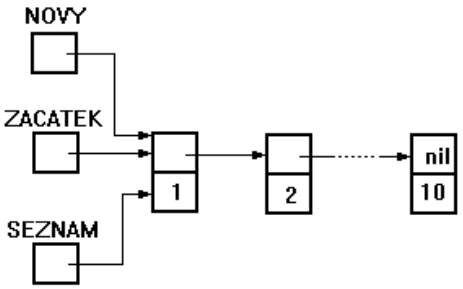



```
I:=7
new (NOVY);
NOVY↑.X=7;

NOVY↑.DALSI:=SEZNAM;
SEZNAM:=NOVY;
.....
```

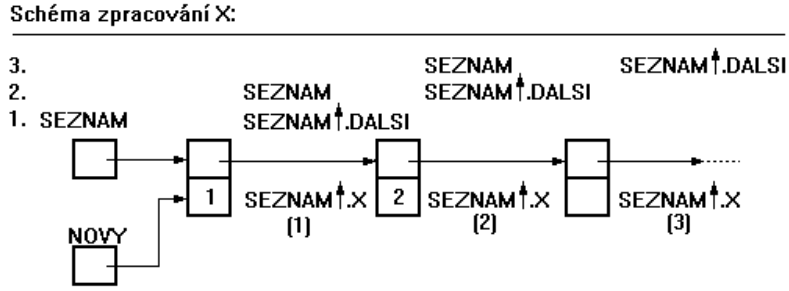



```
ZACATEK:=SEZNAM;
```



{zpracování proměnné X - pro jednoduchost uvažujme pouze tisk obsahu X}

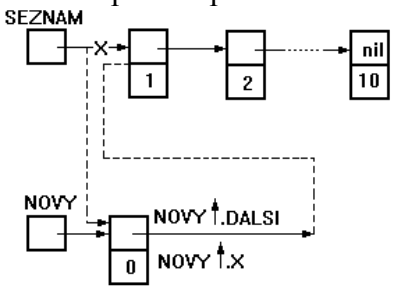
while SEZNAM <> **nil do**
begin *write* (SEZNAM↑.X);
SEZNAM:=SEZNAM↑DALSI
end;
{obnovení ukazatele SEZNAM na začátek lze provést}
{buď}
SEZNAM:=ZACATEK;
{nebo}
writeln;
while SEZNAM <> NOVY **do**
begin *write* (SEZNAM↑.X);
SEZNAM↑.DALSI:=SEZNAM
end;
Schéma zpracování X:



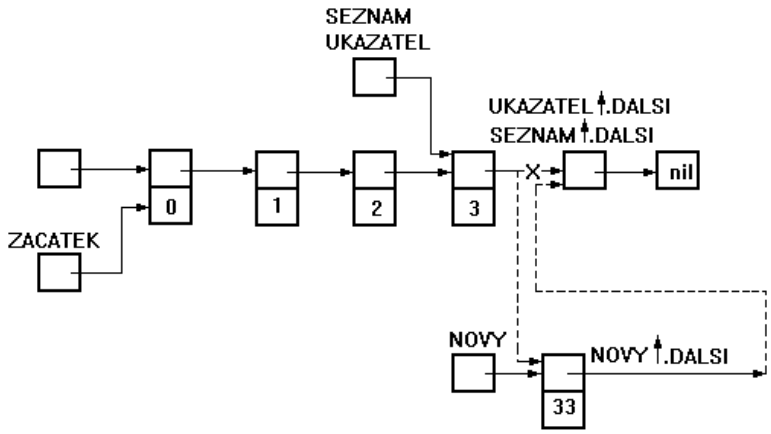
{vložení nové hodnoty do seznamu na začátek}

new (NOVY);
NOVY↑.X:=0;
NOVY↑.DALSI:=SEZNAM;
SEZNAM:=NOVY;

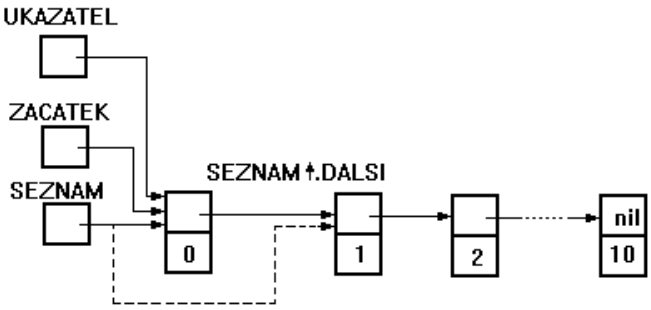
schéma přidání prvku na začátek:

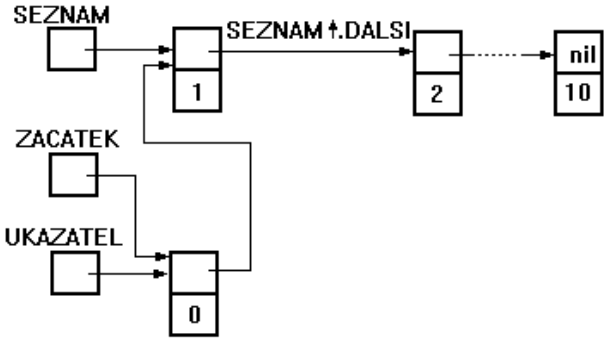


{vložení nové hodnoty na označené místo}
{nejdříve nutno nastavit ukazatel na žádané místo, například}
repeat
UKAZATEL:=SEZNAM;
SEZNAM:=SEZNAM↑.DALSI
until SEZNAM↑.X=3;
{algoritmus vkládání}
new (NOVY);
NOVY↑.X:=33;
NOVY↑.DALSI:=UKAZATEL↑.DALSI;
UKAZATEL↑.DALSI:=NOVY;
schéma vkládání:



{odebrání - zrušení prvního prvku}
UKAZATEL:=ZACATEK; {nastaveni ukazatele UKAZATEL na zacatek}
SEZNAM:=SEZNAM↑.DALSI;
schéma zrušení:

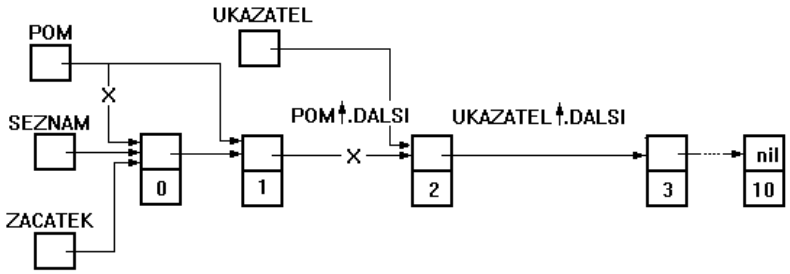




Pozor, UKAZATEL ukazuje stále na první prvek seznamu, tedy zrušit odebrání lze příkazem SEZNAM:=UKAZATEL (rovněž SEZNAM:=ZACATEK).

```
{odebrání označeného prvku ukazatelem UKAZATEL}
{ukazatel POM nastavíme tak, že ukazuje v seznamu prvek, který předchází označenému prvku}
POM:=ZACATEK; {nastavení ukazatele POM na začátek}
while POM↑.DALSI<>UKAZATEL do POM:=POM↑.DALSI;
{odebrání prvku, na který ukazuje ukazatel UKAZATEL}
POM↑.DALSI:=UKAZATEL↑.DALSI;
```

schéma odebrání označeného prvku:



5. Vlastní a standardní jednotky UNIT

5.1 Vlastní jednotky

U rozsáhlých programových projektů ,můžeme narazit na problém velikosti paměti pro kód programu (maximální rozměr je 64 KB), nebo se snahou vytvořit samostatné programy (knihovnu), které mohou být sdíleny různými programy. Turbo Pascal vyšších verzí umožňuje sestavit vlastní jednotky (moduly) a tím program velkého rozměru rozdělit. Jednotky mohou využívat další vlastní nebo standardní jednotky. Každá jednotka se skládá z části **interface** (propojovací) a **implementation** (implementační) a má následující strukturu

Unit jméno_modulu;
Interface
Uses seznam jednotek, které daná jednotka využívá;
veřejná deklarační část (**const, label, type, var**, atd); Všechny deklarace může používat také jiná jednotka, uvedená v seznamu jednotek za **Uses**.
Procedure jméno_procedury (parametry);
Function jméno_funkce (parametry): typ funkce; uvádí se pouze hlavička procedury či funkce a její parametry
Implementation
Vnitřní deklarační část, platná pouze v rámci jednotky
Procedura jméno_procedury (parametry);
lokální deklarační část
begin
    příkazy procedury
end;
function jméno_funkce (parametry): typ funkce
Ve vlastním zápisu procedury či funkce uvedené v interface, možno parametry a typ funkce vynechat.
begin
Inicializační část jednotky pro počáteční inicializaci proměnných, pro otevření souborů, atd. Není-li použito, begin se neuvádí.
end.

Příklad:
Sestavme program pro součet matic C = A + B. Čtení matic a součet sestavme jako samostatnou programovou jednotku.

```
unit P30;
interface
type TA=array[1..10,1..10] of real;
procedure CTI(var X:TA;N:integer);
procedure SUM(var X, Y, Z:TA;N:integer);
implementation
procedure CTI;
var I, J:integer;
begin
        write(´zadej prvky matice oddelene mezerou ci novym radkem:  ´);
        for I:=1 to N do
        for J:=1 to N do
read(X[I,J])
end;
procedure SUM;
var I,J:integer;
```

```
begin
        for I:=1 to N do
        for J:=1 to N do
        Z[I,J]:=X[I,J]+Y[I,J]
end;
end.
```

```
uses P30;
var     A,B,C:TA;
        I,J,N:integer;
        begin
repeat
        write(´zadej rozmer ctvercove matice 10>=N>1:  ´);
        read(N);
        until (N>1) and (N<=10);
        writeln;
        CTI(A,N);CTI(B,N);SUM(A,B,C,N);
        for I:=1 to N do
        begin
                writeln;
                for J:=1 to N do
                write (C[I,J])
        end;
end.
```

5.2 Standardní jednotky

V Turbo Pascalu má uživatel možnost využívat předdefinované konstanty, proměnné, funkce a procedury, které jsou součástí standardních jednotek.

Jednotka SYSTÉM
Tato jednotka obsahuje standardní funkce a procedury uvedené v předchozích kapitolách. Jednotku není nutné uvádět v Uses, její připojení k programu sestavenému uživatelem je provedeno automaticky.

Jednotka DOS
Slouží pro komunikaci s operačním systémem pomocí funkcí a procedur pro volání jádra operačního systému a programu BIOS.
Jednotka OVERLAY
Umožňuje vytváření rozšiřujících modulů, čímž snižuje nároky programu na paměť a umožňuje spouštět programy, které přesahují kapacitu operační paměti.

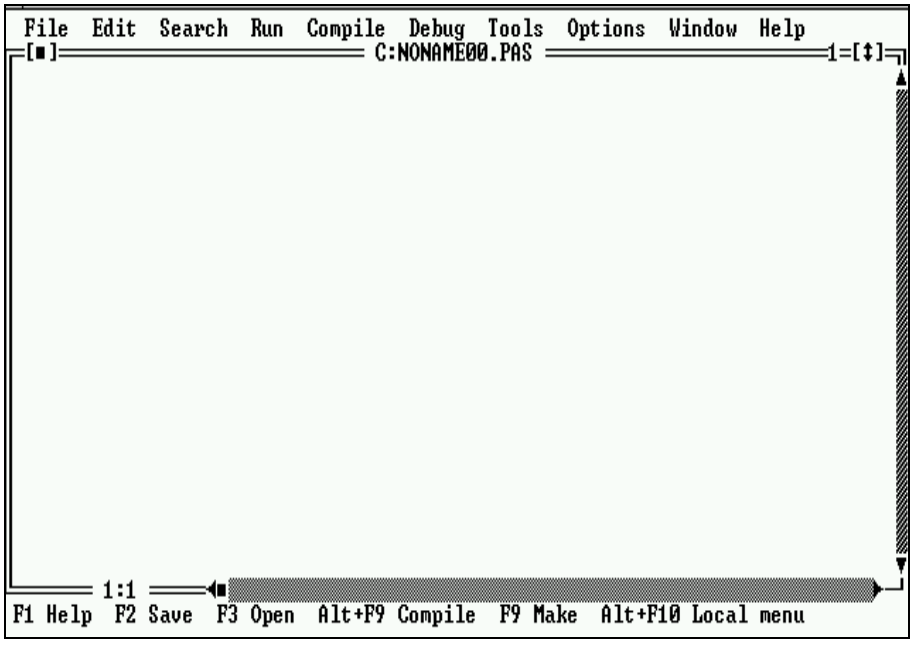
Jednotka PRINTER
Obsahuje propojení výstupů z programu na tiskárnu. Deklaruje textový soubor LST a přiřadí jej k zařízení LPT1-PRN.

Jednotka CRT
Jednotka CRT obsahuje procedury a funkce pro podporu práce v textovém editoru.
Tato jednotka obsahuje programy grafiky od vysokoúrovňových až k bitově orientovaným. Při použití grafické jednotky je nutno volat proceduru *InitGraph* (GraphDriver, GraphMode, cesta). Tato indikuje technické prostředky grafiky, zavede a inicializuje příslušný grafický řídící program, nastaví systém do grafického režimu a předá řízení volajícímu programu.

Příloha 1:

INTEGROVANÉ VÝVOJOVÉ PROSTŘEDÍ TURBO PASCALU - IDE

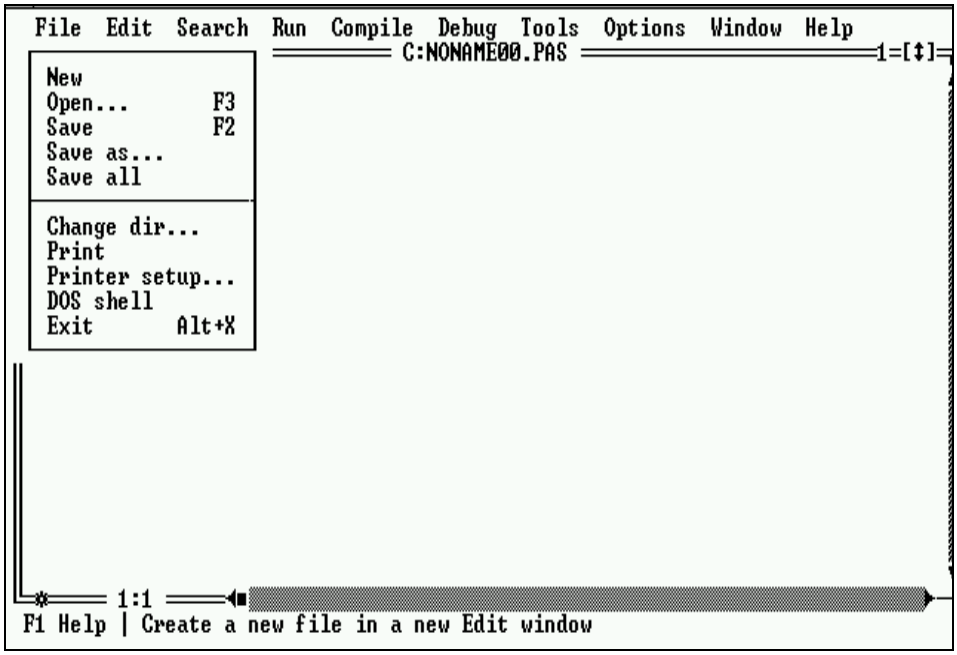
Při spuštění se zobrazí následující obrazovka:



Obrazovka je rozdělena na následující části:
- horní řádka obrazovky je hlavní menu.
- střední část obrazovky je oblast okének
- poslední řádka obrazovky je stavový řádek.

Hlavní menu a další vnořená menu

Hlavní menu se nachází na první řádce a slouží pro primární přístup ke všem příkazům IDE. Je vždy zobrazené, s výjimkou zobrazené uživatelské obrazovky a je kdykoliv přístupné- pomocí F10 nebo pomocí myši. Z hlavního menu se lze dostat do všech vnořených menu. Menu vždy obsahuje několik nabídek, z nichž jen jedna je zvýrazněna. Volbou nabídky provedeme příslušný příkaz.



Výběr příslušné nabídky
a) **pomocí klávesnice**
- aktivaci hlavního menu provedeme stisknutím *F10*
- pomocí směrových kláves vybereme požadovaný příkaz
  nebo
  stiskneme barevně odlišené písmeno požadovaného příkazu – neplatí v hlavním menu.

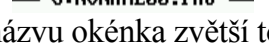
Zobrazení podmenu lze zrušit pomocí *Esc*, kdy se dostaneme po každém stisknutí *Esc* o jednu úroveň menu výše. Hlavní menu zrušit nelze.
b) **pomocí myši**
- stiskneme levé tlačítko myši na kurzoru nastaveném na zvoleném názvu menu, čímž se zobrazí další nabídka (nezobrazuje-li se další nabídka, příkaz se provede)
- stiskneme levé tlačítko na příslušné nabídce.

Poznámka: Nabídky, které nemají v daném okamžiku smysl, jsou barevně odlišeny a nedají se vybrat.

Oblast okének

Okénko je oblast obrazovky, kterou můžeme přemísťovat, měnit její velikost, zvětšit na plnou obrazovku, zviditelnit, překrývat, zavírat a otvírat. Otevřených okének může být tolik, kolik se jich vejde do paměti, ale vždy jen jedno je aktivní. Aktivní okénko je to, ve kterém právě pracujeme, je dvojitě orámováno a je na povrchu obrazovky (není překryto jiným okénkem). Jednotlivé části okénka mají následující význam.

- Název okénka nebo jméno souboru v něm uloženého



Stisknutím tlačítka dvakrát po sobě na názvu okénka zvětší toto okénko na celou obrazovku.
Stisknutím tlačítka a pohybem myší při stisknutém tlačítku na názvu okénka se celé okénko posune.

- Uzavírací značka slouží k rychlému uzavření okénka

Stiskneme tlačítko myši na uzavírací značce. Z klávesnice nutno použít *Window/Close* nebo *Alt + F3*.

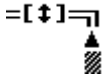
- Posuvný pás s palcem slouží k rychlému přesunu kurzoru v souboru

Kurzor myší přesuneme na šipku příslušného směru pohybu a opakovaně stlačíme tlačítko myši, palec ukazuje pozici zobrazené části souboru nebo nastavíme kurzor na palec a táhneme palec při stlačeném tlačítku myši na požadované místo.

- Roh pro změnu velikosti okna

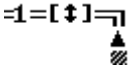
Nastavíme kurzor myši na tuto značku, stlačíme tlačítko myši a táhnutím myši okénko zvětšujeme či zmenšujeme. Z klávesnice se použije *Window/Size/Move* nebo *Ctrl + F5*. Pomocí *Shift* a směrových kláves okénko upravíme. Činnost ukončíme pomocí *Enter*.

- Značka pro zvětšení na plnou obrazovku nebo zmenšení okénka na původní [ ] rozměr



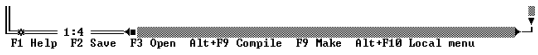
Stlačíme tlačítko myši na značce, čímž se zvětší či změnší okénko. Z klávesnice pomocí *Window / Zoom* nebo *F5*.

- Číslo okénka



Číslo okénka, které je číslováno 1 až 9 a lze je snadno aktivovat. Stlačíme tlačítko myši kdekoliv ve viditelné části okénka. Z klávesnice stiskneme *Alt* a číslo okénka. *Alt + O* vypíše seznam všech otevřených a uzavřených okének a umožní rychlé aktivování požadovaného okénka. U editovacího okna se navíc zobrazuje  kde znak * značí, že soubor v tomto editovacím okénku byl změněn a nebyl ještě uložen a dvojice čísel informuje o aktuální pozici kurzoru v souboru, přičemž je nejdříve uvedeno číslo řádky a pak číslo sloupce.

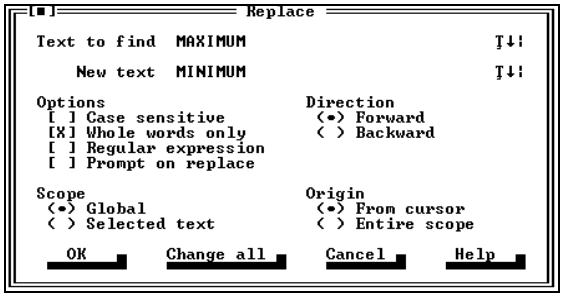
- Stavová řádka



Tato řádka se nachází na spodní řádce obrazovky a má tyto funkce:
- napovídá název a význam "horkých" kláves, které lze momentálně využít v souvislosti s aktuálním stavem IDE. Stlačením zvolené "horké" klávesy nebo stlačením tlačítka myši na této nápovědě znamená volbu této akce,
- informuje o právě probíhající činnosti IDE,
- slouží pro stručný popis významu právě označené nabídky menu.

Dialogová okénka

Je-li nabídka v menu následována třemi tečkami, pak se po výběru otevírá dialogové okénko. Tvar okénka a jeho obsah je pro každou nabídku odlišný.



V dialogovém okénku se mohou objevit tyto položky:
1) **Akční tlačítka**, kde

OK značí provedení všech změn nebo akcí provedených v tomto dialogovém okénku a odchod z něho.

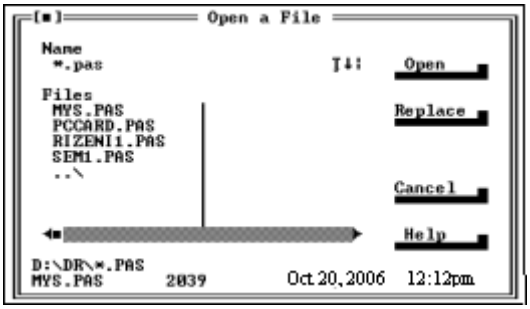
Cancel všechny provedené změny nebo nastavení jsou zrušeny a dialogové okénko je opuštěno, stejný význam má klávesa *Esc*.

Help zobrazí se příslušná nápověda.

2) **Vstupní řádka.** Dovoluje zapsat požadovaný text pomocí klávesnice, obvykle jméno souboru. Vstup textu se ukončí pomocí *Enter*.

3) **Historie** - seznam historie. V seznamu historie lze listovat pomocí myši (stiskneme tlačítko myši s kurzorem nastaveným na značce -) nebo pomocí směrové klávesy. Vybraný text se klávesou Enter zobrazí ve výstupní řádce, kde může být dále upravován.

4) **Seznam položek**. Typické použití je při volbě *File/Open*, kdy do vstupní řádky napíšeme *.PAS a v seznamu položek se objeví výčet všech dostupných souborů s příponou .PAS.



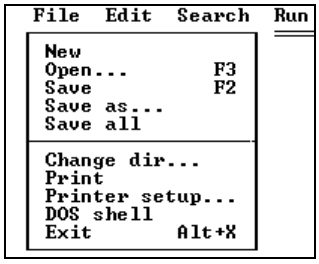
5) **Volby.** Takto je označen blok voleb, které se dají jednotlivě zapnout [X] nebo vypnout [ ]. Volby jsou na sobě zcela nezávislé. Zapnutí nebo vypnutí se provede stlačením tlačítka myši na jménu volby
nebo
pomocí *Alt* a zvýrazněného písmene volby
nebo
pomocí klávesy *Tab* vybrat skupinu voleb a pomocí směrových kláves označit volbu a tu přepnout pomocí klávesy mezera.

6) **Přepínače**. Takto je označen blok na sobě závislých položek přepínače, z nichž jen jedna může být aktivní. Ta je pak označena (•) a všechny ostatní ( ). Označení se provede jako u voleb (viz. bod 5).

7) **Stavová řádka.** Tato řádka se zobrazuje jen v některých dialogových okénkách a poskytuje komplexnější informace o položce vybrané v seznamu položek (viz bod 4). Typickým příkladem jsou informace o vybraném souboru jako čas a datum vytvoření, velikost, atd.
Mezi jednotlivými položkami se lze pohybovat pomocí myši (stlačením tlačítka s nastaveným kurzorem na žádané položce) nebo pomocí klávesy *Tab* (pohyb dopředu), *Shift+Tab* (pohyb dozadu), nebo pomocí *Alt* a příslušného zvýrazněného písmene.

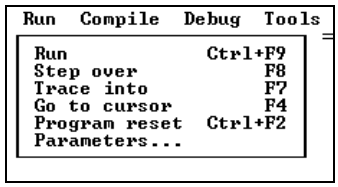
Vytvoření nového souboru

V zobrazeném IDE volíme v hlavním menu příkaz *File/New*. Volbu možného menu provedeme nejpohodlněji pomocí pomocí myši.

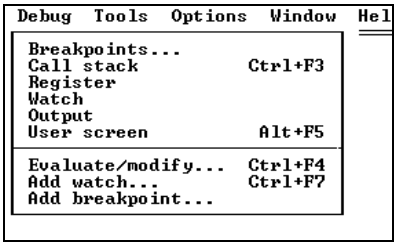


Do editačního okénka, se jménem NONAME00.PAS zapíšeme text našeho programu, např.

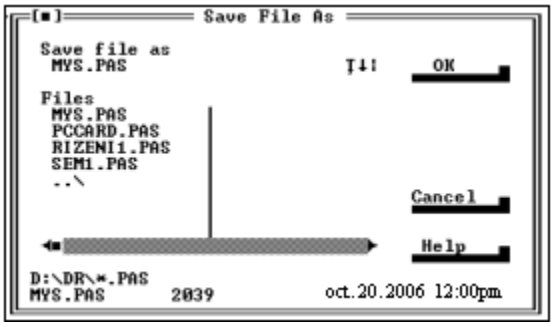
V hlavním menu volíme příkaz *Run* a v zobrazeném navrženém okénku volíme příkaz *Run*.



Program se přeloží a spustí. Výsledek se zobrazí v uživatelském okénku, které je přístupné příkazem *Debug/User Screen* (nebo pomocí kláves *Alt+F5*). Přepnutí zpět do IDE lze stlačením *Esc* či *F10*.

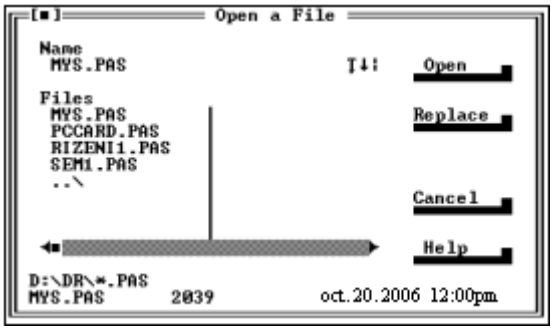


Chceme-li uložit nově vytvořený soubor nebo pod jiným jménem uložit již dříve vztvořený soubor, volíme příkaz *File/Save as* a do zobrazeného okénka do vstupní řádky zapíšeme nové jméno (včetně diskové jednotky a adresáře), nebo potvrdíme původní NONAME jméno. Již dříve uložený soubor lze pod stejným jménem uložit příkazem *File/Save*.



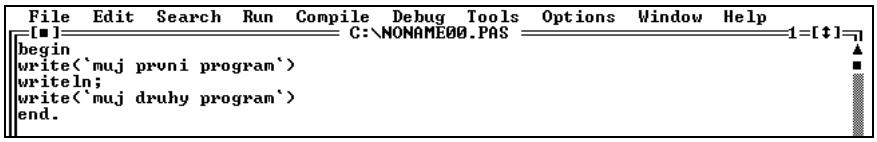
Editace - úprava souboru

Jestliže chceme editovat dříve vytvořený soubor, volíme příkaz *File/Open*



a do zobrazeného okna zapíšeme jméno souboru, který je uložený na disku, případně diskovou jednotku a adresář. Výběr *Open* v okně značí natažení souboru do nového okénka, výběr *Replace* značí nahrazení zdrojového souboru nově zvoleným souborem
V editačním okně doplníme dříve vytvořený program např. následovně:

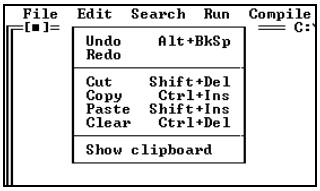
V hlavním menu volíme příkaz *Run* a v zobrazeném vnořeném menu *Run*. V editačním okně se zobrazí na prvním řádku indikace chyby "Error 85:";"expected". Kurzor je v textu umístěn za výskytem chyby.



Opravíme chybu, doplníme středník ve druhém řádku a program opět spustíme. Další postup je obdobný s dříve uvedeným postupem pro vytváření nového souboru.

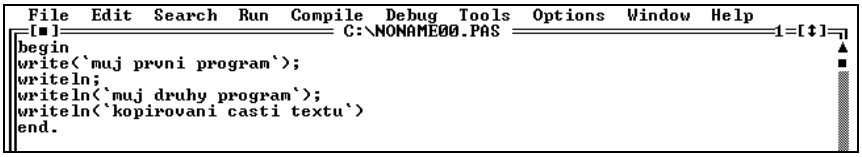
Kopírování textu

Příkaz *Edit/Cut*, *Edit/Copy* a *Edit/Paste* umožňují vybraný text přesunout nebo překopírovat do "Clipboard" (odkládacího prostoru) a z "Clipboard" do zvoleného místa v souboru.
Uvažujme předchozí editovaný soubor, ve kterém chceme část textu překopírovat.

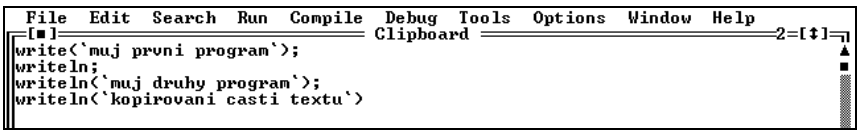


Nejdříve musíme v souboru NONAMEOO.PAS vybrat kopírovanou část, která se zvýrazní po výběru prosvícením. Výběr je možno provést pomocí klávesnice nebo pomocí myši.

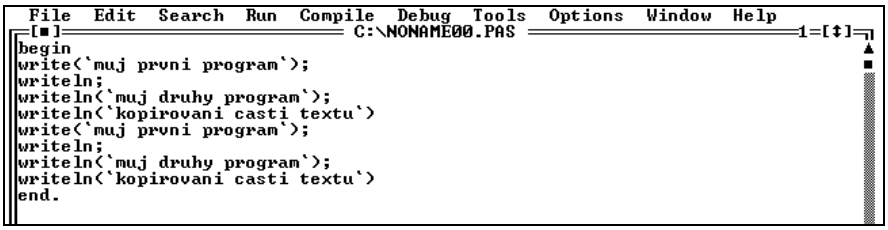
1. pomocí směrových kláves přesuneme kurzor na začátek zvoleného textu a stiskneme *Ctrl+K+B*. Pak kurzor přesuneme na konec textu a stiskneme *Ctrl+K+K*.
2. stlačíme tlačítko myši na začátku vybíraného textu a táhneme kurzor myši na konec vybíraného textu.



Zvolíme příkaz *Edit/Copy*, čímž přesuneme vybraný blok do "Clipboard".



Nastavíme kurzor do místa v souboru v aktivním editovacím okénku, do kterého chceme vložit text z "Clipboard" - schránky a volíme příkaz *Edit/Paste*. Soubor má nyní tvar



Na pátém řádku doplníme středník a aplikujeme již dříve uvedený postup pro spuštění programu.

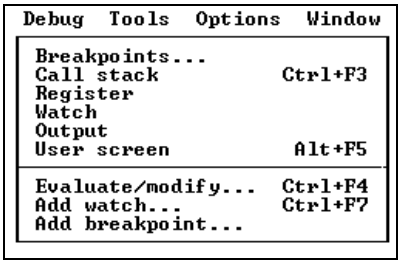
Kopírování příkladů z helpu

Při kopírování příkladů z helpu je vhodné použít příkazu *Copy example*. Všechny příklady v helpu jsou již předem označeny a dají se zkopírovat do "Clipboard" rovnou, bez nutnosti je předem označovat.

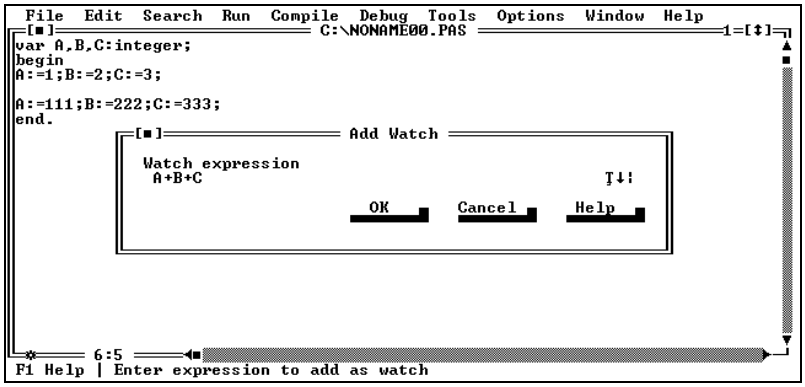
Trasování programu

V případě, že chceme prohlížet činnost programu po krocích, je možno využít příkazu *Run/Trace into*. Pro případ trasování je výhodné zobrazovat obsah proměnných či výsledky výrazů, kterými program krokujeme.

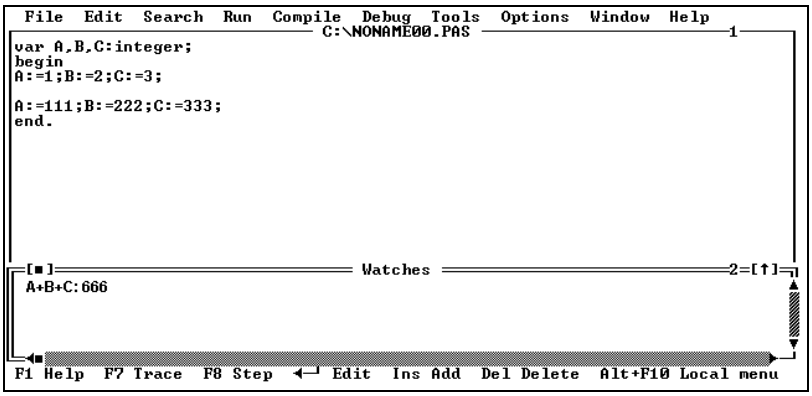
Pomocí *File/Open* vybereme požadovaný program a příkazem *Debug/Watch* zobrazíme sledovací okénko pro umístění zvolených proměnných. Příkazem *Debug/Add Watch* vložíme proměnné či výrazy do sledovacího okénka *Watch*.



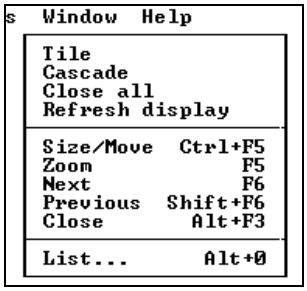
Vložení je možné například zapsáním jména proměnné či výrazu ve vstupní řádce otevřeného okénka *Add Watch*.



Volíme příkaz *Run/Trace into* nebo lépe postupně stlačíme *F7* a v okně *Watch* se zobrazí obsahy proměnných.



V případě, že chceme program trasovat až od řádku, na který nastavíme kurzor, tj. ne od počátku, lze použít příkaz *Run/Goto cursor*. Program se spustí až k řádce, na které se nachází kurzor a od této řádky je možno provádět trasování - pomocí klávesy *F7* (příkaz *Run/Trace into*).
Zobrazení otevřených oken současně lze pomocí příkazu *Window/Tile*.



Zobrazení základních příkazů lze provést pomocí *Alt+F10* nebo stlačením pravého tlačítka myši.

LITERATURA

1. Turbo Pascal - User's Guide, Borland International, 1998
2. Turbo Pascal - Reference Guide, Borland International, 1998
3. Bowles, K.L., Franklin, S.D.,Volper, D.J.: Problem Solving UCSD Pascal. Springer Verlag, New York Berlin Heidelberg Tokio, 1983
4. Wirth, N.:Systematic Programming,London,Prentice Hall, Inc.1976
5. Atkinson, L.: Pascal Programming,New York,John Wiley & Sons, 1990
6. Zaks, R.: Introduction to Pascal, San Francisco,Sybex Inc.,1990
7. Lines, M.V.: Pascal as a Second Lauguage, Prentice Hall,Inc,1984
8. Tenenbaum, A.M.,Augenstein,M.J.: Data Structures Using Pascl.Prentice Hall, Inc., 1980
9. Molnár, L.: Programovanie v jazyke Pascal. Alfa, Bratislava,1987
10. Jinoch, J.,Müller, K.,Vogel, J.: Programování v jazyku Pascal.SNTL Praha 1987
11. Olehla, M.: Počítače a programování. TU Liberec 1992, 1997, 2004, 2007
12. Olehla, M.: Computers and Programming. TU Liberec 1992
13. Turbo Pascal - Object - Oriented Programming Guide, Borland International, 1999
14. Borland Pascal with Objects Version 7.0,Programmer's Reference, Borland International, 1999
15. Borland Pascal with Objects Version 7.0, Language Guide Reference, Borland International, 1999
16. Borland Pascal with Objects Version 7.0, Programming Guide, Borland International, 1999