PASCAL 7.0
Basic facts and elementary user manual

Autor: Mgr. Miroslav Candra
Určeno pro vnitřní potřebu Střední školy spojů a informatiky, Tábor

Obecné vlastnosti:

- Je jednoduchý.
- Snadno se obsluhuje.
- Je tzv. procedurální.
- Není case sensitive (nerozlišuje malá a velká písmenka).
- Každý příkaz musí být ukončen středníkem.

Je jednoduchý:

Pascal svojí syntaxí skutečně patří mezi jednodušší programovací jazyky. Většina příkazů je použita z běžné angličtiny (write, read, record), nebo je zkratkou běžně používaných výrazů (ClrScr – **Cl**ea**r Scr**een)

Snadno se obsluhuje:

Nastavené programu není složité. Vlastní vývojoví prostředí je poplatné době vzniku – je dosovské. Nicméně program se dobře obsluhuje pomocí klávesnice i pomocí myši.

Je procedurální:

Kód se zpracovává přesně v takovém pořadí v jakém byl napsán. Na rozdíl od objektově orientovaného programování, kdy počítač čeká na nějakou událost (jako je třeba klik myší) podle které vybere nějakou proceduru která se spustí, tady probíhají příkazy postupně za sebou, bez ohledu na nějaké události.

Tělo programu:

```
Program nazev_programu;
Uses   CRT;
Type
       TData: array [1..10] of integer;
Var
       Pole: TData;
       a: integer;
Begin

End.
```

Tělo programu je pevně dáno (pořadí sekcí nelze měnit), obsahuje však některé sekce (bloky), které nejsou povinné.

- Klíčové slovo **Program -** není povinné. Za toto slovo se uvádí stručná charakteristika (název) programu. Název nesmí začínat číslicí a nesmí obsahovat mezery a zakázané znaky.
- Klausule **USES** – není povinné. Za klíčové slovo USES se píše seznam knihoven, které se připojují k programu.
- Sekce **TYPE** – není povinné. V této sekci se definují uživatelské datové typy. Deklarují se zde kupříkladu pole, záznam.
- Sekce **VAR** – není povinné. V této sekci se deklarují proměnné, které se používají v programu.
- Závorka **Begin** – **End.** – povinná. Tato sekce je jako jediná naprosto povinná. Obsahuje totiž vlastní program.

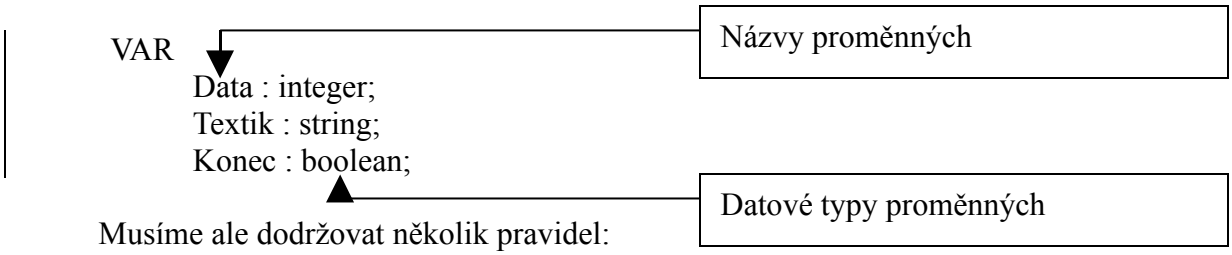
Proměnné a datové typy:

Jedním ze základních prvků programu jsou proměnné. V programu jsou velice důležité, protože právě pomoci proměnných si program pamatuje různé hodnoty (čísla, jména, …).

Proměnou si můžeme představit jako ukazatel do paměti. V sekci Var řekneme, že chceme vytvořit proměnnou data typu byte (var data : byte;). Při překladu programu se v paměti RAM vyhradí místo o velikosti 1 byte. Použijeme-li kdekoliv v programu proměnnou, program se podívá nebo zapíše do vyhrazeného prostoru v RAM.

Deklarace proměnných:

Probíhá v sekci VAR.



Musíme ale dodržovat několik pravidel:

1. Název proměnné musí začínat písmenkem. Je přitom jedno, jestli proměnnou napíšeme malými nebo velkými písmenky.

   Správné deklarace:

   Data : integer;
   Promenna2 : byte;
   A : boolean;
   Dlouhy_nazev : real;
   Udaje, b, c: integer; ← výčet několika proměnných oddělených čárkou.

   Špatné deklarace:

   2Promenna : byte; ← Jako první znak je číslo. TO JE ŠPATNĚ.
   Dlouhy nazev: real; ← Název proměnné je složen ze dvou slov. Název však nesmí obsahovat mezeru, která je zakázaným znakem.

2. Můžeme používat pouze standardní datové typy. Ty si můžeme i sami nadefinovat.

Datové typy:

Datový typ proměnné je velice důležitý, protože určuje jak velké místo v paměti se vyhradí pro proměnnou a zároveň určuje CO mohu do proměnné vložit.

V současné době velkých operačních pamětí se nám to zda nepodstatné, ale při obzvláště velkých aplikacích, nebo v době vzniku pascalu, kdy bylo paměti málo, to smysl má.

Pascal obsahuje různé skupiny datových typů:

- Číselné datové typy
  - Celočíselné
  - Reálné
- Logický datový typ
- Datový typ pro vkládání textu nebo znaků.

Celočíselné datové typy:
- Byte
- Word
- Integer
- LongInt
- ShortInt

Reálné datové typy:
- Real

Logický datový typ:
- Boolean – nabývá dvou hodnot: logická pravda (TRUE), logická nepravda (FALSE)

Datový typ pro vkládání textu nebo znaků:
- String – datový typ určený pro text. Maximální délka řetězce je 255 znaků.
- Char – datový typ určený pro znaky. Znakem se rozumí písmeno, číslo, speciální symboly jako *. Do proměnné lze vložit vždy POUZE jeden znak.

Ordinální datový typ:

Některé výše uvedené datové typy mají zajímavou vlastnost. Určíme-li nějakou hodnotu, dokážeme přesně určit jejího následovníka a předchůdce. Takovýmto datovým typům se říká ORDINÁLNÍ.

Příkladem je datový typ byte, který je celočíselný. Vybereme-li číslo 5, automaticky víme, že předchůdcem je 4 a následovníkem 6. Kdybychom ale zvolili reálný datový typ a vybrali hodnotu 4,8 máme problém. Je následovníkem 5 nebo 4,9 ? A nebude to náhodou 4,85?

Ordinálními datovými typy jsou:
- Celočíselné datové typy
- Datové typy Char a Boolean.

Základní příkazy a rezervovaná slova:

PASCAL obsahuje sadu rezervovaných slov. Jsou to slova, která mají přesný význam, a nesmí být proto nějakým způsobem přejmenovávána nebo upravována.

Zde je jejich kompletní seznam:

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

Příkaz přiřazení

Jeden ze základních příkazů vůbec. Používáme ho tehdy, chceme-li do proměnné vložit nějakou hodnotu. Samotný příkaz je tvořen dvojicí znaků dvojtečka a rovná se : =
Na levé straně je VŽDY proměnná, na pravé straně pak hodnota, kterou do proměnné vkládáme. Na pravé straně může být také výraz, jehož výsledek potom chceme vložit do proměnné.

Př.: a : = 10; ← do proměnné a se vloží hodnota 10
X : = B;
C : = A + B; ← do proměnné C se přiřadí výsledek součtu obsahu proměnných A a B

Příkaz pro výpis

Chceme-li vypsat nějakou informaci na standardní výstup (tím je pro Pascal monitor), musíme použít funkci Write.

Syntaxe: Write (´Vypisovaný text´);

Jak vyplívá ze syntaxe, to co chceme vypsat zapisujeme jako hodnotu funkce do kulatých závorek. Vypisujeme-li text, musí být uzavřen v apostrofech (v pascalu ALT + 39). Chceme-li ale vypsat obsah proměnné, žádné apostrofy nepoužíváme. Chceme-li vypsat najednou více hodnot, oddělujeme je středníkem.

Př.:

Write (´Hodnota proměnné je ´); ← vypisujeme text, je uzavřen v apostrofech

Write (a); ← vypisujeme obsah proměnné, apostrofy se nepoužívají

Write (´Hodnota proměnné je ´, a) ← vypisujeme najednou dvě různé věci – text a obsah proměnné. Obě věci jsou od sebe odděleny čárkou.

Chceme-li, aby pascal po výpisu textu odřádkoval, musíme místo příkazu Write použít WriteLn. Při používání tohoto příkazu platí stejná pravidla jako pro Write.

Příkaz pro čtení

Chceme-li načíst nějaká data ze standardního vstupu (tím je pro pascal klávesnice), musíme použít funkci Read.

Syntaxe Read (a);

Pascal načte obsah bufferu a uloží jej do proměnné a. POZOR, nepoužíváme žádné apostrofy.

Chceme-li po načítání odřádkovat, použijeme příkaz ReadLn.

Př.: Read (a);

Větvení:

Při programování se často dostaneme do situace, kdy se musí program rozhodnout, jak pokračovat dál. V pascalu jsou možné dva způsoby, jak se rozhodnout :

- Porovnáním hodnot (větvení IF)
- Analýzou obsahu proměnné (CASE)

Větvení IF

Zjednodušená syntaxe:

IF logická_podmínka THEN příkaz;

Př.: IF a>b THEN Write(´a je větší´);

Jestliže podmínka PLATÍ (tzn. a je větší než b) provede se příkaz za THEN. Jestliže podmínka neplatí, příkaz Write se ignoruje a program pokračuje dál.

Úplná syntaxe:

IF logická_podmínka THEN příkaz
ELSE příkaz2;

Př.: IF a>b THEN Write(´a je větší´)
ELSE Write(´b je větší nebo rovno´);

Jestliže podmínka PLATÍ (tzn. a je větší než b) provede se příkaz za THEN. Jestliže podmínka neplatí, provede se příkaz za ELSE.

POZOR! Za příkazem ve větví then NESMÍ být středník. Jinak by byl program zmatený. Středník by mu říkal, že příkaz IF skočil a najednou by se mu objevilo klíčové slovo ELSE. Program proto vyhodí chybu.

Chceme-li provádět za větví THEN nebo ELSE více příkazů, musíme je uzavřít do závorky Begin End;

Př.:

```
IF a>b THEN Begin
                    ClrScr;
                    Write(´a je větší´);
               End
          ELSE Write(´b je větší nebo rovno´);
```

CASE

CASE přistupuje k rozhodování úplně jinak. Zatímco If porovnává a rozhoduje se na základě toho jestli podmínka platí nebo ne, Case se podívá na obsah proměnné a potom se rozhodne. Příkaz Case je ukončen End se středníkm.

To klade na programátora větší nároky, protože musí vědět, co by mohla proměnná obsahovat.

Př.:

```
Var
      A: byte;

Begin
      ReadLn (a);
       Case a Of
       1: Write (´Zadal jste jedničku. ´);
       2 .. 5 : Write (´Zadal jste hodnotu z intervalu 2 až 5. ´);
      else Write (´Zadal jste hodnotu větší než 5. ´);
      end;
End.
```

Komentář k příkladu:

Program načte od uživatele hodnotu do proměnné a.
Case se podívá co proměnná a obsahuje. Mohou nastat tři případy:

1. Proměnná a obsahuje číslo 1. V takovém případě se vypíše text „Zadal jste jedničku“. Ostatní části Case se ignorují.
2. Proměnná obsahuje číslo v intervalu od 2 do 5. V takovém případě se vypíše prostřední varianta – text „Zadal jste hodnotu z intervalu 2 až 5.“
3. Proměnná neobsahuje ani 1, ani číslo z intervalu od 2 do 5, ale úplně jiné číslo. V takovém případě se provede větev else.

Cykly

Cykly se využívají tehdy, chceme-li opakovaně provádět nějaký kód – tělo cyklu. Pascal zná tyto cykly:

- For cyklus
- cyklus Repeat – until
- cyklus While

For cyklus

For cyklus je jedním z nejdůležitějších . Oproti ostatním cyklům má několik zásadních výhod:

- má pevně daný počet běhů
- obsahuje řídící proměnnou, kterou lze využit třeba jako počítadlo, obzvláště důležite je její využití u polí (viz následující kapitola – Strukturované datové typy)

Syntaxe:

```
For i := 1 to 10 do
Begin

. . .

End;
```

i … je řídící proměnná. **Musí** být ordinálního datového typu (rozumněj celočíselného). Její hodnota se vždy v každém průběhu cyklu zvyšuje (inkrementuje) o 1.

Takovýto cyklus proběhne celkem 10x. V rúběhu prvního běhu cyklu, bude hodnota i = 1. V průběhu druhého cyklu, bude hodnota i = 2, atd. Cyklus končí, je-li hodnota proměnné i = 10, tzn. Hodnotěm která je uvedena za **to.**

Příklad využití řídící proměnné v cyklu:

```
Var
        i : integer;

Begin
        …
        for i := 1 to 10 do
        Begin
                WriteLn('Průběh číslo' , i);
        End;
```

Existuje ještě jedna varianta FOR cyklu, kde se řídící proměnná nezvětšuje, ale zmenšuje o 1.

```
For i := 10 downto 1 do
Begin
  …
End;
```

Cyklus Repeat – Until

Tento cyklus je poněkud zvláštní. Je to totiž cyklus s podmínkou na konci. Díky tomu tělo cyklu proběhne **vždy** alespoň jednou. Cyklus pribíhá tak dlouho, **dokud nezačne platit podmínka**.

Nejčastěji se tento cyklus využívá, chceme-li uživateli zobrazit nějakou nabídku (hlavní menu), která se má zobrazit opakovaně, vždy po provedení nějaké akce. Uživateli se tak zobrazí volby, které zpracováváme pomocí Case a vždy vyhodnucujeme jestli uživatel chce ukončit svoji činnost (začne platit podmínka) nebo ne (podmínka neplatí) a my mu znova zobrazíme jeho menu.

Syntaxe:

```
Repeat
.
.
.
Until podmínka;
```

Cyklus While

Cyklus while je přesným opakem cyklu Repeat – Until. Podmínka cyklu je na začátku a cyklus běží tak dlouho, dokud platí podmínka.

Syntaxe

```
While podmínka do
Begin
        .
        .
        .
End;
```

Strukturované datové typy:

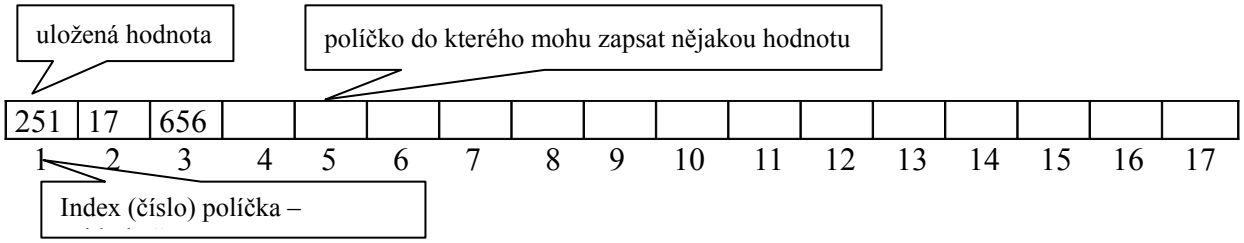
1. datový typ POLE
2. datový typ ZÁZNAM

Typ deklarujeme v sekci označené klíčovým slovem **TYPE**
Využíváme je tehdy, jestliže chceme do jedné proměnné uložit více hodnot.

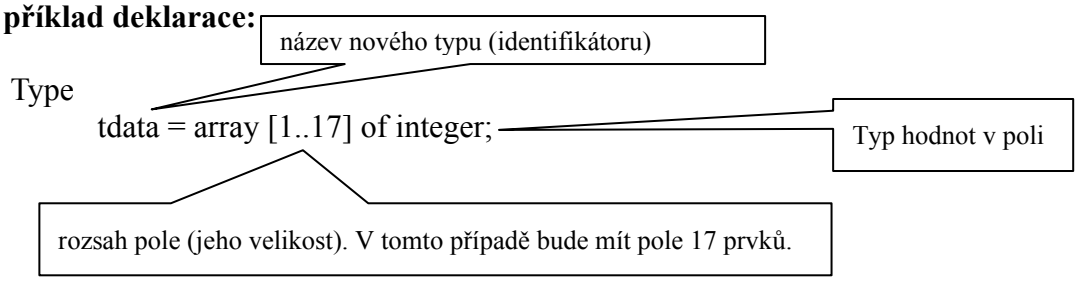
POLE

- klíčové slovo ARRAY
- všechny prvky pole MUSÍ být jednoho typu
- může být jednorozměrné (vektor), dvourozměrné (matice) nebo obecně N rozměrné
- v době překladu musí být známá jeho velikost
- jednotlivá políčka pole se indexují od 1 (v nultém políčku je uložena informace o délce pole)

Standardní představa jednorozměrného pole:



příklad deklarace:



abychom **mohli** pole **používat**, musíme vytvořit proměnnou typu **tdata**

Konkrétní příklad deklarace pole a proměnné:

```
Program ukazka_pole;
uses crt;
type
        tdata = array [1..17] of integer;
var
        pole : tdata;
begin
…
end.
```

Přístup k jednotlivým položkám pole:

zápis do pole na n-tou pozici (od uživatele z klávesnice): **read (pole[n]) ;**
výpis hodnoty z n-té pozice na standardní výstup (monitor): **write(pole[n]);**
(je samozřejmě možné stejným způsobem použít readln a writeln)
přiřazení hodnoty do n-tého políčka: **pole[n] := 25;**

Potřebujeme-li naplnit celé jednorozměrné pole, s výhodou použijeme **FOR cyklus**

Př:. – naplnění 10-ti prvkového pole pomocí for cyklu

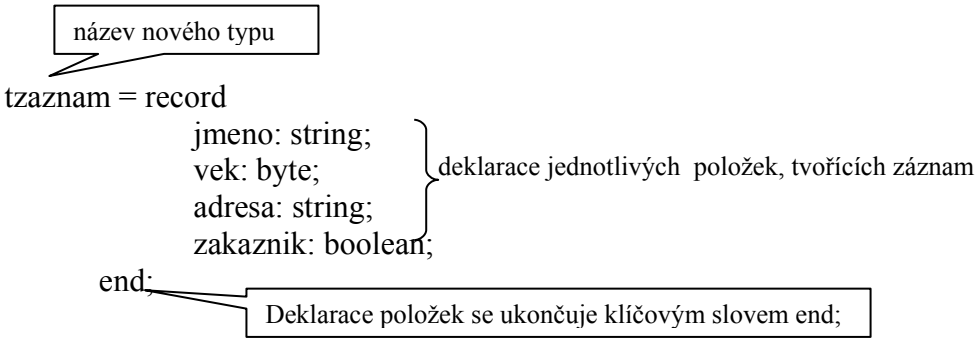
```
FOR i := 1 to 10 do
Begin
        Write(´Zadej hodnotu : ´);
        ReadLn(pole[i]);
End;
```

Výhodou FOR cyklu je skutečnost, že řídící proměnná při každém průběhu zvyšuje svoji hodnotu o 1. Můžeme ji proto s výhodou využít jako ukazatel na chlívek.

ZÁZNAM:

- klíčové slovo RECORD
- pro představu si ho zjednodušujeme a říkáme, že vypadá jako jednorozměrné pole
- položky mohou být **různých datový typů**

Deklarace záznamu:



abychom **mohli** záznam **používat**, musíme vytvořit proměnnou typu **tzaznam**

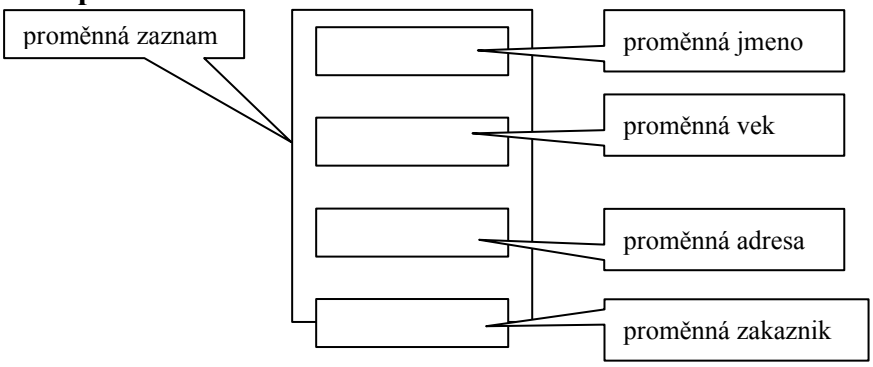
Konkrétní příklad:

```
program ukazka_zaznam;
uses crt;
type
        tzaznam = record
                jmeno: string;
                vek: byte;
                adresa: string;
                zakaznik: boolean;
        end;
var
```

```
        zaznam: tzaznam;
begin
…
end;
```

Proměnná zaznam tedy v sobě obsahuje další 4 proměnné (jmeno, vek, adresa, zakaznik).

Naše představa:



Je možný dvojí přístup:

a) pomocí **tečkové notace**
b) pomocí **with**

tečková notace	with

načtení z klávesnice: **read(zaznam.jmeno);**
výpis na obrazovku: **write(zaznam.jmeno);**
pro ostatní proměnné je stejný postup
 	
Nesmíme zapomenout, že **zaznam** v sobě obsahuje další 4 proměnné. Pomocí tečky vlastně upřesňujeme se kterou z těch 4 proměnných chci pracovat.
	with zaznam do
	begin
	  read(jmeno);
	  write(jmeno);
	end;
	
	použití tečky odpadá

Podprogramy – procedury a funkce

Podprogramy:

Velice často se stává, že v programu musíme jeden výpočet (nebo obecně nějaký kód) opakovat několikrát a to na různých místech programu. Psát pokaždé stejný kód není efektivní a zhoršuje to také přehlednost programu.

Mnohem přijatelnějším řešením je napsat kód jednou a pak ho pouze opakovaně volat. A přesně k tomuto účelu slouží podprogramy. Turbo Pascal 7.0 zná dva druhy podprogramů – procedury a funkce.

Deklarace podprogramů:

Podprogramy se deklarují **POD** sekcí Var, ale **NAD** počátečním BEGIN.

Příklad:

```
Program ukazka;
Uses CRT;
Type
        TPole  = array [1..10] of integer;
Var
        Pole : Tpole;
        A : Integer;

Procedure pozdrav;
Begin
        WriteLn (‘ Zdravím … ☺ ‘)
End;

Begin

        // hlavní program

End.
```

deklarace podprogramu

Procedury – deklarace a volání :

Klíčové slovo při deklaraci procedur je slovo **procedure**. Ihned po tomto klíčovém slově následuje **jméno procedury** (identifikátor) a seznam parametrů.

Př.:

```
Procedure pozdrav (jmeno: string);
Begin
        WriteLn (‘ Ahoj ‘,jmeno);
End;
```

Klíčové slovo **procedure**

Název procedury

Tomuto řádku říkáme **HLAVIČKA PROCEDURY**

Parametr procedury

Procedura se volá v hlavním programu **názvem** procedury. Má-li procedura nějaké parametry, musíme jim při volání procedury předat nějakou hodnotu.

Př.:

```
…
Var
nazev: string;

Procedure pozdrav (jmeno: string);
Begin
        WriteLn (‘ Ahoj ‘,jmeno);
End;

Begin
        …
        Write(‘Zadej jméno :‘);
        ReadLn (nazev);
        Pozdrav(nazev);
        …
End.
```

Volání procedury

Hlavička procedury obsahuje kromě klíčového slova procedure, názvu procedury také jeden parametr – **proměnnou jmeno**.
Proto, když voláme tuto proceduru, **musíme** proměnné jmeno předat nějakou hodnotu. V našem případě se proměnné jméno předá obsah proměnné nazev.

Proměnné deklarované v proceduře nazýváme **lokální proměnné**. Existují pouze a jenom tehdy, když program zavolá proceduru. Po skončení procedury tato lokální proměnná zanikne. To znamená, že takovouto proměnnou mohu využívat POUZE v procedurách.

Naproti tomu proměnné deklarované tak jak jsme zvyklí – v sekci Var, nazýváme **globální proměnné**. Platí úplně všude a pořád, takže je mohu využívat třeba i v proceduře.

Existují dva způsoby jak předat parametry – hodnotou a odkazem. Rozdíl v deklaraci je nepatrný, následky obrovské

Př: A (předávání hodnotou)	**Př: B (předávání odkazem)**

Procedure soucet (x,y: integer);
Begin
  x:=x+y;
  WriteLn(‘ Soucet je ‘, x);
End;
	Procedure soucet (**var** x,y: integer);
	Begin
	  x:=x+y;
	  WriteLn(‘ Soucet je ‘, x);
	End;

Pozorným porovnáním obou příkladů zjistíme, že se liší pouze ve slovíčku VAR v příkladu B. A to je skutečně ten jediný a základní rozdíl. Ovšem důsledky jsou mnohem zajímavější, jak dokazuje následující příklad :

Př: A (předávání hodnotou)

```
Procedure soucet (x,y: integer);
Begin
  x:=x+y;
  WriteLn(‘ Soucet procedury je ‘, x);
End;

Begin

. . .
soucet (a,b);
c :=  a + b ;
WriteLn(‘ Soucet programu je ‘, c);

…
End.
```

Př: B (předávání odkazem)

```
Procedure soucet (var x,y: integer);
Begin
  x:=x+y;
  WriteLn(‘ Soucet procedury je ‘, x);
End;

Begin

. . .
soucet (a,b);
c := a + b;
WriteLn(‘ Soucet programu je ‘, c);

…
End.
```

V případě A, kdy předáváme hodnotou se na obrazovce po spuštění objeví tyto řádky (vstupní hodnoty zadané uživatelem jsou 10 a 5)

Soucet procedury je 15
Soucet programu je 15

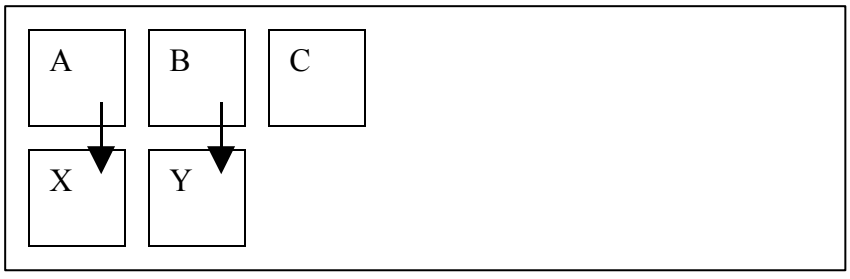
Ovšem v případě B se objeví:

Soucet procedury je 15
Soucet programu je 20

Ve všech případech je výsledek správný, i když v tom posledním případě se nám to může jevit jinak (10+5 je přeci 15 a ne 20 !). Rozdíl je následující:

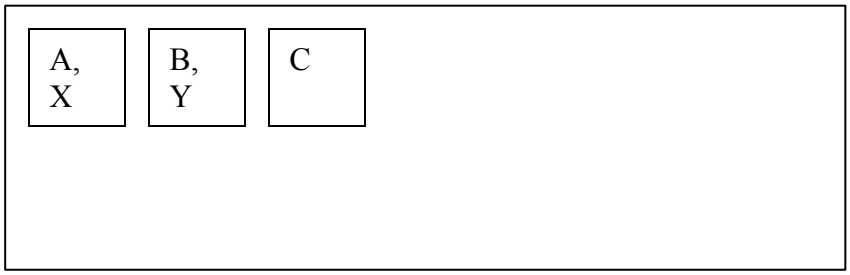
Volání hodnotou:

Když spustíme program, vytvoří se v paměti (v RAM) tři „chlívky“. Jeden pro proměnnou A, druhý pro proměnnou B a třetí pro proměnnou C. Do chlívků pro proměnné A a B se potom uloží uživatelem zadané hodnoty 10 a 5. Jakmile se spustí procedura, vytvoří se další dva nové chlívky pro proměnné X a Y. Do chlívku pro X se překopíruje obsah chlívku pro proměnnou A a stejně tak do chlívku pro Y se překopíruje obsah chlívku B. Když nyní změníme hodnotu v chlívku X, tak se hodnota v chlívku A nemění, protože mezi nimi není žádná vazba.



Volání odkazem :

Zpočátku probíhá vše úplně stejně jako v předchozím případě. Rozdíl nastane v okamžiku, kdy program zavolá proceduru. Tentokrát se nevytvoří chlívky pro proměnné X a Y, ale řekne se: proměnná X je na stejné adrese jako proměnná A a proměnná Y je na stejné adrese jako proměnná B. To znamená, že proměnné A a X odkazují do stejného chlívku a proměnné B a Y taky do stejného chlívku. Jenomže, když změníme obsah proměnné X, změníme i obsah proměnné A. A jelikož proměnná X změní v ukázkové proceduře svoji hodnotu z 10 na 15, hodnota proměnné A se také změní z 10 na 15. Z tohoto důvodu se objeví „nesprávný“ výsledek 20, protože 15 + 5 je skutečně 20.



Funkce:

Deklarace funkce se nepatrně liší od deklarace procedury:

```
Function soucet (x,y: integer) : integer;
Begin
  Function := x + y;
End;
```

Jednak je zde jiné klíčové slovo – slovo **function**. Funkce na rozdíl od procedury může vracet hodnotu, což znamená, že s ní mohu de facto pracovat jako s proměnnou. Dokonce si mohu dovolit i následující zápis:

WriteLn(‘ Výsledek sčítání je : ‘, soucet(a,b))
(Pro méně všímavé, volání funkce je součástí výrazu – WriteLnu)

Volání a předávání parametrů je stejné jako u procedur.

Unity:

Unity jsou externí soubory (jednotky, knihovny), které se po připojení k programu stávají součástí programu. Typickými příklady jsou například unity Crt a Graph. Unity v Pascalu mají standardně příponu .TPU (Turbo Pascal Unit).

Struktura unity:

```
Unit název_unity;

Interface
   // hlavicky procedur

Implementation
   // vlastní procedury

End.
```

Za klíčovým slovem unit následuje název unity. POZOR – název unity a název souboru MUSÍ být shodné. Jinými slovy, je-li soubor unity pojmenován alfa.tpu, **musí** být za klíčovým slovem unit uvedeno také **alfa**.

V sekci Interface jsou uvedeny kompletní hlavičky procedur, které mají být přístupné z venku.

V sekci Implementation jsou uvedeny kompletní hlavičky a těla všech procedur. POZOR. Hlavičky procedur v sekci Interface a Implementation musí být shodné.

Konkrétní příklad:

```
Unit pokus;
Interface
        Procedure pozdrav;
Implemetation
        Procedure cara;
        Begin
                WriteLn(' - - - - - - - - - - - - - - - ');
        End;

        Procedure pozdrav;
        Begin
                WriteLn('Ahoj :    ');
                Cara;
        End;
End.
```

Jak je vidět, v sekci Interface je uvedena pouze hlavička procedury Pozdrav. To znamená, že v programu, ke kterému připojím tuto unitu budu moci tuto proceduru zavolat.

Př.:

```
Program ukazka;
Uses Crt, pokus;

Begin
        Pozdrav;
        Readkey;
End.
```

Kdybych pod volání procedury pozdrav napsal ještě volání procedury cara, program by hlásil chybu – neznámý identifikátor, i když tato procedura v unitě existuje. Není ale uvedena v sekci Interface, takže je z venku nepřístupná. Zavolat ji může pouze sama procedura Pozdrav.

Jak vytvořit Unitu v Pascalu

Nejdříve si rozmyslíme, co ta naše unita bude umět a napíšeme kód, tak jak jsme zvyklí. Dáme si pouze pozor, aby unita mněla svoji strukturu tak jak je uvedena výše. Nepíšeme tudíž žádné program, var, uses a podobně.

Napsanou unitu musíme uložit (soubor … uložit jako …). Nezapomeňte, že soubor se musíme jmenovat stejně, jako unita (musíte použít stejná slova jako za klíčovým slovem unit). Současně musíte respektovat omezení pascalu respektive dosu. Je-li název unity delší než 8 znaků, musíte soubor pojmenovat prvními osmi znaky (mezery a jiné zákeřnosti nepoužívat).

Máme-li unitu uloženou, změníme v menu compile položku destination z memory na disk. Nyní můžeme unitu zkompilovat. Jestliže ji nyní otevřeme po druhé, zjistíme že to již není textový soubor, ale binární.

Soubory:

Někdy se stane, že potřebujeme uložit trvale mnoho dat. Ideální jsou pro tyto účely soubory. Turbo Pascal 7 zná v obecné rovině dva typy souborů – textové a netextové (binární).

Textové soubory:

Chcete-li si představit jak vypadá textový soubor, otevřete si poznámkový blok a napište do něj pár vět. V textovémzo souboru jsou data uspořádána do řádků, kdy každý řádek končí dvojicí znaků CR (#13) a LF (#10).

Textové soubory můžeme otevřít pro **čtení**, **zápis** nebo **přípis**. Platí, že si musíme vybrat pouze jednu variantu, nelze soubor otevřít třeba pro čtení a zápis současně.

Otevřeme-li soubor pro čtení, můžeme z něj pouze číst. Otevřeme-li soubor pro zápis, původní obsah se vymaže a nová data jej nahradí. Otevíráme-li soubor pro zápis, a tento soubor neexistuje, tak bude založen jako nový (pro čtení či přípis musíme otevírat existující

soubory). Otevření pro přípis znamená, že ukazatel se přesune na konec souboru a čeká na nová data.

Postup jak otevřít soubor:

Ještě před tím, než budeme soubor otevírat, musíme v sekci Var nadeklarovat proměnnou typu text, kterou propojíme s fyzickým souborem přímo na disku (Pascal neumí přistupovat přímo k souboru, musí to dělat prostřednictvím proměnné).

Př.: deklarace proměnné typu text

```
Var
        Soubor: text;
```

K vlastnímu propojení fyzického souboru a proměnné typu text slouží procedura **assign**.

Př.: propojení souboru s proměnnou:

```
Var
        Soubor: text;

Begin
        …
        assign (soubor, ‘ data.txt ‘);
        …
```

Otevření souboru **pro čtení**:

```
Reset (soubor);
```

Otevření **pro přípis**:

```
Append (soubor);
```

Otevření **pro zápis**:

```
Rewrite (soubor);
```

Ukončujeme-li práci s programem, nebo chceme-li uzavřít soubor a znova ho otevřít pro jinou činnost, musíme soubor zavřít pomocí procedury **close**:

```
Close (soubor);
```

Konkrétní příklad:

```
Program MO_textove_soubory;

Uses Crt;

Var

  data : text;
  soubor: string;
  obsah: string;

Begin

  ClrScr;

  Write('Zadej nazev souboru, ktery chces otevrit : ');
  ReadLn(soubor);

  Assign(data, soubor); (* Propojeni textoveho souboru s promennou typu text *)

  Reset (data);

  While not eof(data) do
  Begin
   ReadLn(data, obsah);
   WriteLn(obsah);
  End;

  ReadKey;

  Close (data);


End.
```

ReadLn pracuje ze standardním vstupem – klávesnicí. Já ale potřebuji načítat ze souboru, proto jej musím přesměrovat z klávesnice na soubor.

To se dělá tak, že do závorky uvedu jako první proměnnou typu text a až potom proměnnou do které chci načítat data

Fráze

```
While not eof(data) do
  Begin
   ReadLn(data, obsah);
   WriteLn(obsah);
  End;
```

funguje tak, že dokud není konec souboru (**E**nd **O**f **F**ile – EOF) načítá se obsah.

Ještě existuje: EOLn – **E**nd **O**f **Lin**e – konec řádku

Netextové soubory (binární):

Binární soubory se mohou dělit na dvě velké skupiny :

1. Typové (prvky souboru jsou definovaného typu)
2. Netypové (prvky souboru jsou kupříkladu typu záznam – tzn. mají ještě vnitřní strukturu.

Příklad deklarace typového souboru:

```
Var
    data: file of integer;  (*Všechny záznamy budou typu integer*)
```

Příklad deklarace netypového souboru:

```
Type

    TZaznam = record
                jmeno: string;
                vek: byte;
              end;

Var
    data : file of TZaznam;
```

Chceme vytvořit soubor, obsahující jediný údaj — číslo 65. Vytvoříme-li jej jako soubor typu *file of Byte*, bude obsahovat jediný byte s hodnotou 65. Pokud bychom ovšem později četli tento soubor jako textový, byl by jeho jediný byte interpretován jako znak #65 neboli 'A'. Bude-li soubor naopak vytvořen jako textový, bude obsahovat dva byty — znak '6' a znak '5'. Při čtení tohoto souboru jako *file of Byte* by pak první z jeho bytů byl interpretován jako číslo 54 ('6' = #54) a druhý jako číslo 53 ('5' = #53).

Práce s typovým souborem

Otevření typového souboru

Pro otevření existujícího souboru je třeba použít proceduru *Reset*, pro založení nového souboru proceduru *Rewrite*. Obě pracují obdobně jako při otvírání souboru textového.

procedure Reset (typový soubor);
procedure Rewrite (typový soubor);

Obě procedury otevírají fyzický soubor a ukazatel aktuální pozice v souboru nastavují na jeho začátek. Fyzický soubor, otevíraný procedurou *Reset*, musí v okamžiku jejího volání již existovat, kdežto *Rewrite* případně existující fyzický soubor smaže a založí nový. Procedura *Append* nemůže být pro otevření typového souboru použita — otevření existujícího typového souboru pro přípis lze dosáhnout procedurou *Reset* a následným nastavením ukazatele aktuální pozice na konec souboru (procedura *Seek*).

Při použití procedury *Rewrite* je soubor vždy otevřen současně pro čtení i pro zápis (jeho komponenty lze číst i zapisovat), kdežto při použití procedury *Reset* závisí režim přístupu k souboru na hodnotě standardní proměnné *FileMode*.

Definuje režim přístupu k binárnímu souboru, otvíranému procedurou *Reset*. Všeobecně platné hodnoty proměnné a jim příslušející režimy přístupu k souboru jsou:

kód	***režim***
0	pouze pro čtení
1	pouze pro zápis
2	pro čtení i zápis

Přiřazení nové hodnoty proměnné *FileMode* způsobí, že všechna další volání procedury *Reset* vezmou tuto hodnotu v úvahu. Implicitní hodnotou (při startu programu) je 2.

Zpracování typového souboru

Pro čtení aktuální komponenty a zápis do aktuální komponenty jsou stejně jako u textových souborů k dispozici procedury *Write* a *Read*.

Pro práci s netypovým souborem se používají funkce a procedury:

FileSize (data); - ***funkce***
Vrací velikost souboru v komponentách (aktuální celkový počet komponent v souboru).

FilePos (data); - ***funkce***
Vrací číslo komponenty na které je ukazatel (index) počáteční komponenta má index 0).

Seek (data, pozice);
Nastavuje ukazatel aktuální pozice v souboru na hodnotu parametru *Pozice*.

Truncate (data);
Odstraní (vymaže) ze souboru aktuální komponentu a všechny následující.

EOF (data); - ***funkce***
K dispozici je rovněž funkce *EOF*, testující stav konec souboru. Vrací *True* v případě, kdy se ukazatel aktuální pozice v souboru nachází za jeho poslední komponentou (má hodnotu velikosti souboru v komponentách).

Práce s netypovými soubory

Tyto soubory nemají známou strukturu ani velikost komponent souboru. Práce je proto s tímto souborem paradoxně jednodušší, zejména při přenosu velkých objeml dat..

Pro otevření lze použít stejně jako u typových souborů procedury Reset a Rewrite (popsáno výše)

Pro zpracování procedur je možno použít proceudury a funkce uvedené u typových souborů.