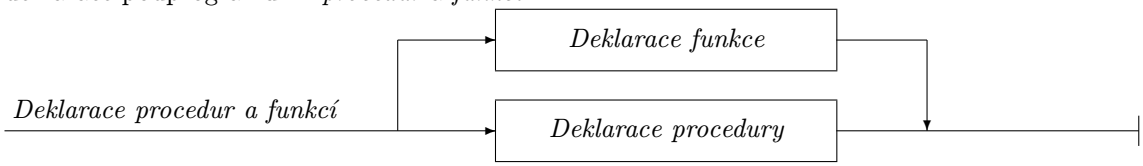
Kapitola 2

Podprogramy; procedury a funkce

Jak již víme, program je v Pascalu blok, který označujeme jménem uvedeným v hlavičce programu. Hlavička programu i blok jsou syntakticky definované kategorie Pascalu, přičemž v bloku rozeznáváme část definicí a deklarací a část příkazovou. V části definicí a deklarací je možné uvádět deklarace podprogramů — *procedur* a *funkcí*.



Podprogram[1], ostatně jak již vyplývá z názvu, je část programu, reprezentující určitý logický celek . Jak procedura, tak i funkce je v Pascalu tvořena blokem, jemuž předchází hlavička procedury popř. hlavička funkce. Již na této úrovni jsou čtenáři známy tzv. standardní podprogramy, ať již jsou to standardní procedury jako *Read, WriteLn, Reset* aj., nebo standardní funkce jako *sin, sqrt, eof* aj.

Blok programu i blok podprogramu (blok) má v našich vybraných pasážích stejnou syntaktickou definici[2]. Liší se pouze ukončením. Blok programu končí **.** (tečkou), blok podprogramu je ukončen **;** (středníkem). V bloku podprogramu jsou mj. popsány jeho objekty: definovány a deklarovány jeho konstanty, typy, proměnné a podprogramy — tzv. vnořené procedury a funkce, jejichž blok je opět syntakticky známá kategorie *blok*.

Objekty, které jsou popsány uvnitř podprogramu jsou objekty lokální, z okolí podprogramu nedostupné. Uvnitř podprogramu je však možno se odkazovat na objekty, které jsou popsány v bloku nadřazeném. Objekty, které jsou popsány např. v bloku programu jsou dosažitelné ze všech podprogramů a jsou vzhledem k nim objekty globálními.
Lokální objekty vznikají v okamžiku volání podprogramu a s ukončením jeho činnosti zanikají - jejich obsahy nejsou definovány.

Potřeba deklarovat podprogram v rámci programu může být motivována mnoha důvody:

a) Použití podprogramů vede k výslednému dobře strukturovanému programu, jenž je přehledný, čitelný a tudíž i srozumitelný a modifikovatelný.

b) Použití podprogramů umožňuje, aby na jednotlivých podprogramech – relativně samostatných úkolech – pracovali různí tvůrci.

c) Použití podprogramu často také znamená zkrácení programu a to v případech, kdy na různých místech v programu je zapotřebí vykonat stejnou sekvenci příkazů. Tuto sekvenci zapíšeme pouze jednou a to ve formě podprogramu, který pak na potřebných místech v programu voláme. Již při řešení jednoduchého úkolu "Násobení dvou matic" je možná spolupráce tří řešitelů:

[1] V textu na místech, kde není nutné rozlišovat mezi pojmy procedura a funkce, používáme pojem podprogram
[2] Ve skutečnosti blok podprogramu může ještě obsahovat *způsob volání* (inline popř. interrupt (pouze u procedur)) a jiné *direktivy* (external popř. forward)

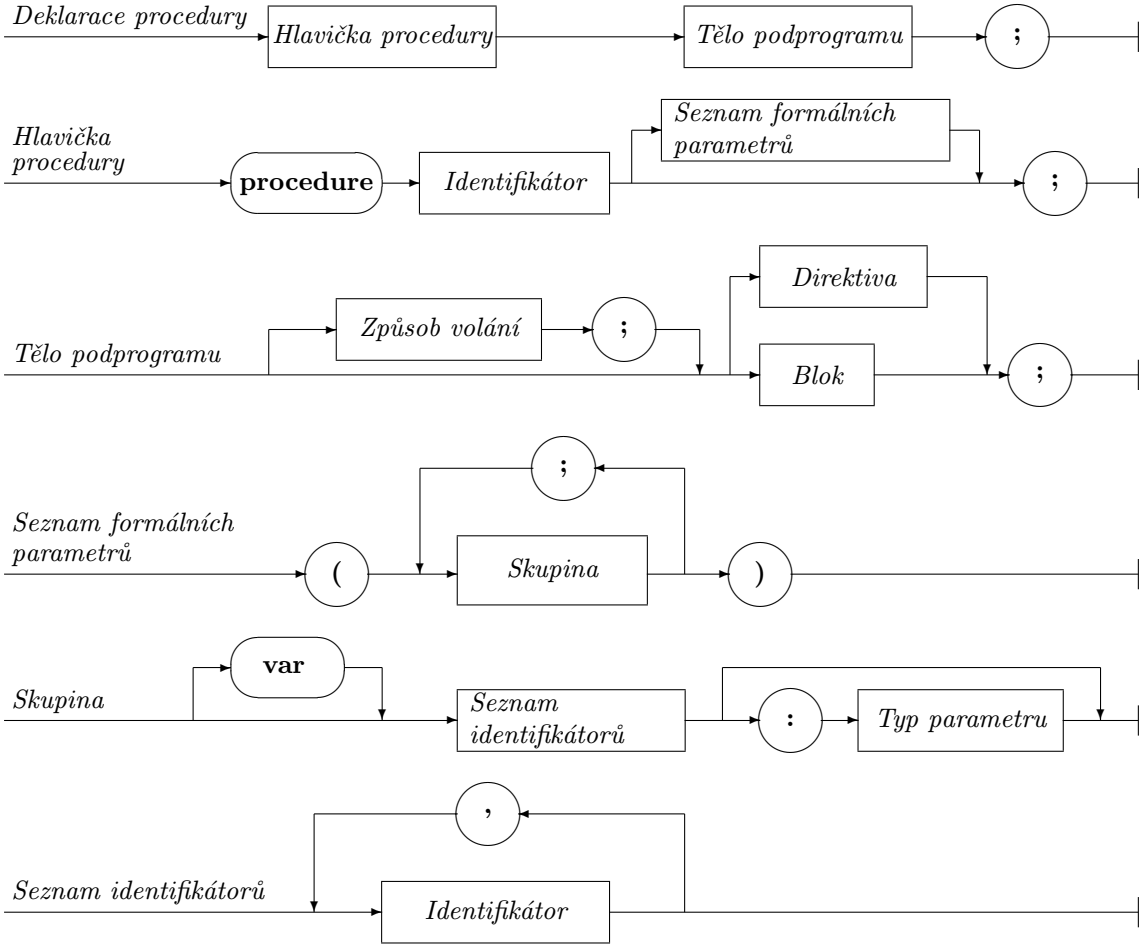
- Prvý napíše podprogram pro načtení hodnot jedné matice (bude volán – použit – dvakrát),
- druhý napíše podprogram pro tisk jedné matice (bude volán třikrát) a
- třetí napíše podprogram pro násobení dvou matic.

d) V neposlední řadě pak umístění podprogramů do knihoven (jednotek) nám umožní se k vytvořenému podprogramu vracet v mnoha aplikacích.Tímto způsoben jsou tvořeny knihovny standardních podprogramů, které podstatným způsobem pak rozšiřují možnosti programovacího jazyka.

Jak jsme již uvedli, v Pascalu rozdělujeme podprogramy na procedury a funkce.
Hlavním úkolem procedury je vykonání určité sekvence příkazů — napsat hlavičku v tabulce, nakreslit obrázek, načíst vstupní hodnoty do strukturované proměnné apod. Volání procedury je příkaz, který vede k provedení úkolu.
Úkolem funkce je poskytnout funkční hodnotu. Volání funkce se tedy vyskytuje ve výrazu (v aritmetickém, v logickém či v jiném výrazu).

Deklarace procedury nebo funkce sestává z uvedení hlavičky procedury nebo funkce, již následuje blok. Blok sestává z části definicí a deklarací a z části příkazové.

2.1 Procedury



Jak je zřejmé z uvedených syntaktických definic, procedura může obsahovat ve své hlavičce seznam formálních parametrů. V okamžiku upotřebení procedury - při tzv. volání procedury — v příkazové části bloku, v rámci jehož části definicí a deklarací byla procedura deklarována, jsou jednotlivé formální parametry nahrazeny parametry skutečnými. Seznam skutečných parametrů je součástí příkazu procedury. Takto je zajištěna komunikace procedury s jejím okolím. Také možná

komunikace prostřednictvím nelokálních (globálních) proměnných představuje spíše speciální, nežli obecný způsob komunikace.
V případě, že procedura při své činnosti hodnoty ze svého okolí nepřijímá ani do něj hodnoty neodevzdává je možno vytvořit proceduru bez parametrů.

♡ **Příklad 2.1.:** Napište podprogram pro tisk úvodní hlavičky nového listu, který má být ve tvaru:

```
          Vysoké učení technické v Brně
                 fakulta stavební
ústav informatiky a automatizace inž. úloh
-----------------------------------------
```

Vlastní řešení je jednoduché, největší „dřina“ je v počítání úvodních mezer každého řádku tak, aby text seděl pokud možno uprostřed.

```
Procedure HLAVICKA;
  var I : byte;
Begin for I:=1 to 15 do Write(' ');
  WriteLn('Vysoké učení technické v Brně');
  for I:=1 to 22 do Write(' ');
  WriteLn('fakulta stavební');
  for I:=1 to 11 do Write(' ');
  WriteLn('ústav informatiky a automatizace inž. úloh');
  for I:=1 to 11 do Write(' ');
  Writeln('-----------------------------------------');
end;
```

V proceduře HLAVICKA je deklarována lokální proměnná I, jejíž použitelnost je omezena pouze na blok, ve kterém byla deklarována.

V uvedeném řešení jsou části, které se stále opakují. Jedná se o tisk úvodních mezer na každém z řádků. V rámci bloku procedury HLAVICKA deklarujme další proceduru TISKMEZ. Této proceduře je však třeba dodat informaci o počtu tisknutých mezer, jenž je pro každý řádek jiný (proměnný).

```
Procedure HLAVICKA;
  var I : byte;
  Procedure TISKMEZ(N: byte);
    var K : byte;
  Begin for K:=1 to N do Write(' ');
  end;  {konec procedury TISKMEZ}
Begin TISKMEZ(15);
  WriteLn('Vysoké učení technické v Brně');
  TISKMEZ(22);
  WriteLn('fakulta stavební');
  TISKMEZ(11);
  WriteLn('ústav informatiky a automatizace inž. úloh');
  TISKMEZ(11);
  Writeln('-----------------------------------------');
end;  {konec procedury HLAVICKA}
```

Pozorný čtenář by již mohl místo procedury pro tisk mezer TISKMEZ deklarovat proceduru pro tisk znaku TISKZNAK (její hlavička by byla ve tvaru **Procedure** TISKZNAK(ZNAK:char; N: byte);)
a využít ji nejen místo procedury TISKMEZ, ale i k tisku posledního řádku hlavičky (místo posledního příkazu procedury HLAVICKA).

♠ **Konec příkladu 2.1.**

Formální a skutečné parametry představují substituční mechanismus, který umožňuje použít logiku podprogramu s různými vstupními hodnotami a dojít tak k různým výsledkům. Zajišťují spojení podprogramu s jeho okolím. Podprogram je realizován na základě formálních parametrů, jež v okamžiku jeho volání nahradí (substituují) parametry skutečné.

Při volání procedury(v příkazu procedury)se tedy za formální parametry dosadí skutečné parametry, které jsou reprezentovány buď hodnotami (výrazy), nebo proměnnými. Pořadí a typ skutečných parametrů musí odpovídat pořadí a typu formálních parametrů, typy musí být identické (kompatibilita nepostačuje).

Způsob nahrazování – substituce – formálních parametrů parametry skutečnými je v Pascalu dvojí:

- substituce hodnotou (volání hodnotou)
  Dosazení hodnoty – formální parametr nahrazovaný hodnotou představuje v bloku podprogramu lokální proměnnou, které je na začátku provádění podprogramu přiřazena hodnota skutečného parametru. Hodnota skutečného parametru zůstává i po provedení podprogramu nezměněna. Je tedy zřejmé, že tímto způsobem substitujeme pouze vstupní parametry procedury.

- substituce proměnnou (volání odkazem)
  Dosazení proměnné – formální parametr je nahrazován konkrétní proměnnou, která tvoří skutečný parametr.Tohoto způsobu substituce se používá k substituci výstupních parametrů procedury. Klíčové slovo var je součástí specifikace formálních parametrů substituovaných proměnnou.

♡ **Příklad 2.2.:** Nechť existuje následující část programu:

```
.
.
   var I : integer;
       B : array[1..2] of integer;
   Procedure COSI(X:integer);
      Begin I:=1;         X:=X+2;
            B[I]:=10;
            I:=2;         X:=X+2
      end;
Begin
 .
 .
 B[1]:=1;    B[2]:=1;
 I:=1;       COSI(B[I]);
 .
 .
end.
```

Při výše uvedené specifikaci formálního parametru X dochází v okamžiku volání procedury COSI k jeho nahrazení hodnotou skutečného parametru B[I]. Formální parametr X má charakter lokální proměnné, jejíž platnost je pouze v bloku procedury COSI.

Činnost uvedené části programu dokumentuje následující sledovací tabulka:

```
B[1]  1                        10                  10
B[2]    1                                           1
I         1                 1      2                2
X     nedef.       1          3      5            nedef.
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
     program     substi-    procedura          program po
                 tuce         COSI             proceduře
```

```
    Je-li hlavička procedury COSI zapsána ve tvaru
              Procedure COSI(var X:integer);
```

a ostatní zůstává beze změny, tj. specifikace formálního parametru obsahuje klíčové slovo var, dochází k jeho nahrazení proměnnou, kterou reprezentuje zápis skutečného parametru. Je třeba si uvědomit, že zápis B reprezentuje konkrétní strukturovanou proměnnou B, zápis B[I] jakousi její složku, která v našem případě při I=1 reprezentuje konkrétní první složku indexované proměnné B[1].

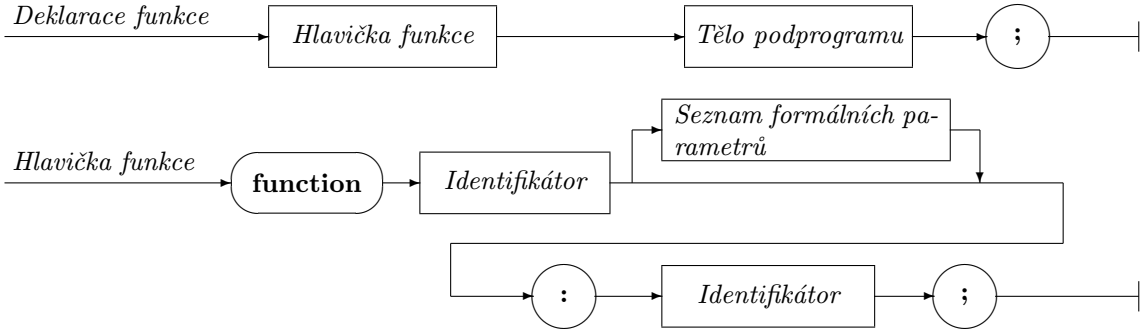
Změnu v činnosti uvedené části programu opět dokumentujeme sledovací tabulkou:

```
B[1]  1        B[1] nahradí X   3   10      12     12
B[2]    1                                            1
I         1                        1        2        2

     program    substituce      procedura COSI    program po
                                                   proceduře
```

♠ **Konec příkladu 2.2.**

2.2 Funkce



Jak jsme si již řekli, je vyvolání procedury uskutečněno tzv. příkazem procedury. Vyvolání funkce se předepisuje výrazem, který nazýváme zápisem funkce. Zápis funkce se může vyskytnout pouze ve výrazu, tj. např. na pravé straně přiřazovacího příkazu.
Jako funkce se zapisují algoritmy, jejichž provedením se vypočte hodnota určitého typu. Typ funkce, tj. typ hodnoty, kterou funkce dodává, nesmí být strukturovaný, tj. může být jednoduchý, popř. typu ukazatel.
Přípustnost skutečných parametrů i způsoby jejich substituce parametry skutečnými se řídí stejnými pravidly jako tomu bylo u procedur. Samotný identifikátor deklarované funkce se musí vyskytnout na levé straně přiřazovacího příkazu ve svém bloku, v jeho příkazové části. Tímto způsobem se definuje výsledná funkční hodnota.

♡ **Příklad 2.3.:** Napište algoritmus v podobě podprogramu pro výpočet N! (faktoriálu z čísla N)
$N! = N * (N-1) * (N-2) * .. * 2 * 1 * 0!$
kde $0! = 1$

Řešení v podobě procedury pozorný čtenář je schopen zřejmě již sestavit:

```
Procedure FAKT(N : byte;  var F : longint);
   var I: byte;
   Begin F := 1;
         for I:=1 to N do F :=  F * I;
   end;
```

Jelikož však výsledkem požadovaného algoritmu je jedna hodnota, navíc hodnota jednoduchého typu, je typickým řešením podprogram v podobě funkce:

```
Function F(N : byte) : longint;
   var I : byte;
```

```
    POM : longint;
Begin POM := 1;
      for I:=1 to N do POM := POM * I;
      F := POM;
end;
```

Všimněmě si zavedení jakési na první pohled zbytečné pomocné proměnné POM v bloku funkce. Co nás k tomu vede? Odpověď na následující tři otázky:

1.

Otázka: Kolik skutečných parametrů musí být uvedeno v zápisu funkce (při volání funkce)?

Odpověď: Tolik, kolik v hlavičce funkce bylo specifikováno formálních parametrů

2.

Otázka: Kde se může vyskytnout zápis funkce?

Odpověď: Pouze ve výrazu

3.

Otázka: Může se vyskytnout identifikátor funkce bez skutečných parametrů?

Odpověď: Alespoň jednou musí, a to ve své deklarované příkazové části a ne jinde, než v levé části přiřazovacího příkazu.
Tedy v příkazové části funkce F by mohlo být místo příkazu $POM := 1$ bez následků $F := 1$. Ovšem v opakujícím se příkaze $POM := POM * I$ nemůže být $F := F * I$, poněvadž F na pravé straně přiřazovacího příkazu nemůže být pouhý identifikátor, ale musí to být zápis funkce.
Poslední možností je zápis požadovaného algoritmu v podobě rekurzívní funkce:

```
Function RF(N : byte) : longint;
   Begin
        if N =0 then RF :=1  else RF :=N * RF(N-1)
   end;
```

♠ **Konec příkladu 2.3.**

2.3 Rekurzívní podprogramy

V Pascalu je možno používat podprogramy rekurzívně. Rekurzívní volání podprogramu je takové volání, které nastane ještě dříve, než předchozí volání bylo ukončeno. Rozlišujeme rekurzi přímou a vzájemnou.
Přímá rekurze je uvedena na konci Příkladu 2.3., tj případ, kdy podprogram volá sám sebe.
Vzájemná rekurze nastává tehdy, když podprogram A volá podprogram B a rovněž tak podprogram B volá podprogram A. U vzájemné rekurze se dostáváme do rozporu s jedním zásadním pravidlem Pascalu, které říká, že objekt musí být deklarován před svým prvním použitím. Tento problém je u vzájemné rekurze vyřešen tzv. předdeklarací hlavičky jednoho z podproramů následovanou direktivou **FORWARD**.

V současné době existují programovací jazyky, jež podporují rekurze. Velice efektivně lze řešit problémy, které při nerekurzívní algoritmizaci jsou prakticky neřešitelné. Z hlediska efektivity výpočtu v Pascalu je však rekurze nevýhodná a podstatně prodlužuje dobu výpočtu. Avšak algoritmy s rekurzí vypadají vesměs pěkně.

2.4 Typ podprogram

Občas konstruujeme podprogramy, jejichž parametrem je rovněž podprogram. Standardní PASCAL tyto možnosti řeší bez jakýchkoliv potíží. Od verze 5.0 Turbo Pascalu je i v tomto prostředí umožněno použít u podprogramů parametr typu podprogram.

Popis typu podprogram, např.

type TYPFUNKCE = **function** (X,Y:byte) : integer;
TYPPROC = **procedure**;

definuje jako množinu přípustných hodnot konkrétního typu podprogram všechny podprogramy, kterými jsou procedury nebo funkce se stejným počtem a typem parametrů, které jsou volány jako vzdálené (FAR) a jsou deklarovány v bloku programu. Způsob volání si Turbo Pascal volí sám. Pokud se podprogram nachází uvnitř programu, je automaticky volán jako blízký (NEAR). Způsob volání jako vzdálený lze v tomto případě předepsat direktivou {$F+}

Do množiny přípustných hodnot nepatří podprorogramy deklarované v jednotce SYSTEM.

Deklarujeme-li proměnné

var F1, F2 : TYPFUNKCE;
P1, P2 : TYPPROC;

pak těmto proměnným lze přiřazovat konkrétní hodnoty - procedury (do P1 a P2) a funkce (do F1 a F2).

Můžeme psát např. tyto přiřazovací příkazy (za předpokladu připojení jednotky CRT):

P1 := ClrScr; P2 := P1;

a poté je možno smazat obrazovku buď příkazem

ClrScr; nebo příkazem P1; nebo příkazem P2;

Způsob použití funkcionálních parametrů ukazuje následující příklad.

♡ **Příklad 2.4.:** Určete hodnotu integrálu

$$INT = \frac{\int_0^{\pi/2} 1 - \sin x \, dx + \int_{\pi/2}^{\pi} x^2 \cos x \, dx}{\int_{-1}^{+1} \frac{\cos^3 x}{1+\sin x} \, dx}$$

K řešení užijme Simpsonovu metodu.

```
Program INTEGRACE;
  const PI = 3.14159;
  type FUNKCE = function(ARGUMENT:real):real;
  var INTEGRAL : real;
  {$F+}
      function F1(X:real):real;
         begin
             F1:= 1 - sin(X);
         end;
      function F2(X:real):real;
         begin
             F2:= sqr(X) * cos(X);
         end;
      function F3(X:real):real;
         begin
             F3:= cos(X)*sqr(cos(X)) / (1 + sin(X))
         end;
  {$F-}
  Function SIMPSON(A,B:real; F: FUNKCE) : real;
     const PD = 10;
     var K : integer;
         H, S, SS, SL : real;
     begin H:= (B-A) / PD;
           S:= F(A) + F(B);
           SS := 0.0; SL:= 0.0; K:=1;
           while K<PD do begin SL:=SL + F(K*H+A);
                               K:=K+2;
```

```
                    end;
          K:=2;
          while K<PD do begin SS:=SS + F(K*H+A);
                              K:=K+2;
                    end;
          SIMPSON := H/3 * (S + 4*SL + 2*SS);
          Writeln((H/3 * (S + 4*SL + 2*SS)):8:3);
   end;
Begin
  INTEGRAL := (SIMPSON(0.0,PI/2,F1) +SIMPSON(PI/2,PI,F2))
              /SIMPSON(-1.0,1.0,F3);
  Writeln('Hodnota integralu je ',INTEGRAL:8:3);
end.
```

♠ **Konec příkladu 2.4.**

2.5 Vlastní knihovny podprogramů

Programátor – uživatel osobního počítače – si může pro svoji potřebu vytvářet své vlastní knihovny podprogramů, neboli programové jednotky[3]), které pak využívá při tvorbě různých programových komplexů (knihovy pro statistické výpočty, knihovnu podprogramů numerických metod, knihovna ovladačů speciálních přídavných zařízení (driverů), knihovna pro kreslení určitých obrazců).

Programová jednotka (knihovna podprogramů) sestává ze čtyř částí:

Hlavička — je uvedena klíčovým slovem unit, za kterým je identifikátor. Na tento identifkátor se pak v odstavci uses daného programu odkazujeme.

Část spojovací — slouží k definici a deklaraci objektů (pro nás to jsou podprogramy). Tato část je uvedena klíčovým slovem interface a v našich aplikacích bude obsahovat hlavičky podprogramů, jimž často předchází uvedení odstavce uses.

Část výkonná — struktura této části je strukturou bloku. Při deklaraci podprogramu, jehož hlavička musí být uvedena v části spojovací, nemusí již být uveden seznam formálních parametrů. Je-li však hlavička podprogramu úplná, pak musí být identická s příslušnou hlavičkou uvedenou ve spovací části.

Inicializační část — má strukturu příkazové části bloku. Obsahuje příkazy, které jsou vykonány při odkazu na programovou jednotku v odstavci **uses**. Často nepovažujeme za potřebné inicializační část uvádět.

Programová jednotka je ukončena klíčovým slovem end za nímž následuje **.** (tečka). Pro úplnost uvádíme příklad jednotky STATISTIKA a následně pak její použití v programu.

♡ **Příklad 2.5.:** Hodláme budovat jednotku pro statistické výpočty. Jako její dva prvé podprogramy vytvořme podprogram pro výpočet aritmetického průměru z řady celočíselných hodnot a podprogram pro vzestupné seřazení řady celočíselných hodnot.

```
Unit STATISTIKA;
  interface
      type      POLE=array[1..20] of integer;
      function  PRUMER(N:byte;X:POLE) : real;
      procedure SETRID(N:byte;var X:POLE);
  implementation
      function PRUMER(N:byte;X:POLE) : real;
         var SUMA : integer;
```

[3]**Knihovna podprogramů** není synonymem **Programová jednotka**. Programová jednotka může obsahovat nejen deklaraci podprogramů, ale i deklaraci proměnných, definici konstant a typů. Programová jednotka je tedy pojmem širším.

```
            I    : byte;
      begin SUMA:=0;
            for I:=1 to N do SUMA:=SUMA+X[I];
            PRUMER:=SUMA/N;
      end;
      procedure SETRID(N:byte;var X:POLE);
         var POM : integer;
             I,J : byte;
      begin for J:=1 to N-1 do
              for I:=1 to N-J do
                 if X[I]>X[I+1] then begin POM   := X[I];
                                           X[I]  := X[I+1];
                                           X[I+1]:= POM;
                                     end;
      end;
end.
```

Uvedená jednotka je na disku jako soubor STATISTIKA.UNI a byla přeložena příkazem *make* na disk jako STATISTIKA.TPU

Použití jednotky STATISTIKA v programu UKAZKA:

```
Program UKAZKA;
   uses STATISTIKA;
   var VEKTOR    : POLE;
       I,J,POCET : byte;
       AP        : real;
   Procedure TISKNI(N:byte; X:POLE);
      var I : byte;
   Begin for I:=1 to N do Writeln(I:2,'. hodnota vektoru je ',X[I]:3);
   end;
Begin Write('Zadej pocet nacitanych hodnot:');Readln(POCET);
      for I:=1 to POCET do begin Write('Napis ',I:2,'. hodnotu:');
                                 ReadLn(VEKTOR[I]);
                           end;
      AP:=PRUMER(POCET,VEKTOR);
      Writeln('Prumerna hodnota vektoru je :',AP:6:2);
      Writeln('Hodnoty vektoru pred setridenim:');
      Writeln('--------------------------------');
      TISKNI(POCET,VEKTOR);
      SETRID(POCET,VEKTOR);
      Writeln('Hodnoty vektoru po setrideni:');
      Writeln('-----------------------------');
      TISKNI(POCET,VEKTOR);
end.
```

♠ **Konec příkladu 2.5.**

2.6 Standardní podprogramy

Norma jazyka Pascal definuje poměrně omezenou množinu standardních podprogramů. Velkou silou implementace Turbo Pascalu od verze 4.0 jsou jeho knihovny podprogramů, které se podle potřeby k hlavnímu programu (a pouze k němu) připojují. Děje se tak zpravidla ihned za hlavičkou programu v části **uses**.

K systému Turbo Pascalu od verze 5.0 bylo standardně dodáváno osm programových jednotek:

System — obsahuje základní podprogramy. K programu je připojena implicitně, nesmí být uvedena v odstavci **uses**.

Crt — obsahuje podprogramy pro práci s obrazovkou a klávesnicí (pohyb kurzoru, barvy apod.).

Dos — podprogramy pro spolupráci s operačním systémem.

Graph — podprogramy pro práci v grafickém režimu (podporuje různé grafické adaptéry).

Printer — umožňuje provádět tisk na tiskárně s použitím procedur *Write* resp. *WriteLn*.

Overlay — podprogramy umožňující provádět překrývání modulů v paměti počítače při běhu programu.

Graph3, Turbo3 — podprogramy pro dosažení kompatibility s nižšími verzemi Turbo-Pascalu (V 3.0)

S vývojem verzí Turbo Pascalů některé jednotky pozbývají významu (Turbo Pascal 7.0 je již zcela o něčem jiném, než byl Turbo Pascal 3.0, i když, pravda, stále máme cca 12 příkazů a základní repertoár datových typů k dispozici) a naopak řada nových prodprogramů s každou novou verzí Turbo Pascalu rozšiřuje možnosti programových jednotek *System, Crt, Dos a Graph*. Turbo Pascal 7.0 přináší další jednotku *Strings* pro efektivní práci s dynamickými řetězcovými datovými strukturami zpřístupňovanými pomocí ukazatele.

V následujících odstavcích jsou uvedeny vybrané podprogramy z některých standardních jednotek.

2.6.1 Jednotka SYSTEM

2.6.2 Jednotka DOS

2.6.3 Jednotka CRT

2.6.4 Jednotka GRAPH