Programování

Funkce a procedury v Pascalu

There are
10 types
of people
in the world:

Those who
understand binary
and those
who don't.

Martin Urza

Motivace

- Začneme-li psát trochu složitější programy, zjistíme, že je občas třeba vykonávat nějakou činnost víckrát.
  - Pokud tyto činnosti následují bezprostředně za sebou, lze použít cyklus. To bohužel neřeší ostatní případy.
  - Uvažme program, který má přečíst tři posloupnosti čísel ukončených nulou. V první a třetí posloupnosti má hledat největší číslo, ve druhé má hledat nejmenší.
    - Lze napsat kód, který hledá nejvyšší číslo, potom kód, který hledá nejnižší a pak znovu kód pro nalezení nejvyššího (značná část programu by se ale opakovala).
  - Je možné použít nějaký kód víckrát?
    - Ano, k tomu slouží funkce a procedury.
      - Navíc mají funkce a procedury ještě další význam.

Co je procedura?

- Procedura je pojmenovaný kód.
- Úplně mimo hlavní program (tedy mimo begin a end.) napíšeme nějaký kód (libovolně dlouhý) a ten nějak pojmenujeme.
- Kdekoliv dál pak můžeme tento kód spustit tím, že jeho jméno použijeme jako příkaz.
  - Tomuto spuštění se říká volání (časté je pak slovní spojení „volat proceduru“, případně „volat funkci“).
  - Proceduru lze volat víckrát.
- Poté, co je kód procedury vykonán, vrátí se program zpět tam, odkud byla procedura volána a pokračuje v původní činnosti.

Syntaktický zápis procedury

- Deklarace procedury začíná klíčovým slovem procedure; následuje jméno, které volí programátor; za jménem lze nepovinně psát závorky ().
- Proceduru voláme jménem, za které lze také psát závorky ().
- Kód procedury píšeme mezi klíčová slova begin a end.
- Ve FP lze deklarovat procedury před i za deklarací proměnných.
  - Rozdíl je ve viditelnosti (více na dalším slajdu), zatím deklarujte za proměnné.

```
procedure identifier();
begin

end;
```

```
identifier();
```

```
program foobar;

procedure foo();
begin
  writeln('foo');
end;

var
  bar : boolean;

procedure baz();
begin
  writeln('baz');
end;

begin
  foo();
  baz();
end.
```

Viditelnost

- Viditelnost v Pascalu platí vždy zdola nahoru. Z každého místa v kódu lze používat jen entity deklarované výše.
- Proměnnou lze použít jen v proceduře deklarované pod touto proměnnou.
- Proceduru A lze též volat z procedury B, je-li A vidět z B (což je v případě, že B je deklarována níže než A).
- Příklad: Proměnná baz je viditelná z foo, nikoliv z writeBar; writeBar je vidět z foo, naopak nikoliv; z hlavního kódu je vidět vše.

```
program foobar;

procedure writeBar();
begin
  write('Bar');
  write('Bar');
  writeln();
end;

var
  baz : boolean;

procedure foo();
begin
  write('Hordy barbaru!');
  writeln();
  writeBar();
  writeBar();
  writeBar();
  writeBar();
  baz := true;
end;

begin
  writeBar();
  foo();
end.
```

Lokální a globální proměnné

- Proměnné, které jsme v dosavadních programech deklarovali, jsou tzv. globální. To znamená, že jsou vidět z kódu hlavního programu i procedur deklarovaných pod nimi.
  - Hlavní program i všechny procedury je mohou měnit i číst.
- V každé proceduře lze deklarovat i lokální proměnné.
  - Ty se deklarují za klíčovým slovem var před begin dotyčné procedury.
  - Tyto proměnné jsou viditelné pouze v dané proceduře.

```
program findMaxNumber;


procedure findMax();

  var
    input : integer;
    max   : integer;

begin
  readln(input);
  max := input;

  while (0 < input)
    do
    begin
      readln(input);
      if (max < input)
        then max := input;
    end;

  writeln(max);
end;


begin
  findMax();
end.
```

Životnost lokálních proměnných

- Připomenutí: Co jsou to proměnné?
  - Pojmenované kusy paměti!
- Deklarujeme-li globální proměnnou, program si pro ni při svém startu vyžádá paměť (ta proměnná je pak pojmenováním tohoto místa v paměti). Tato paměť (tedy i její jméno) existuje až do ukončení programu.
- Lokální proměnná je vytvořena až ve chvíli, kdy je volána procedura, které tato proměnná patří. Když procedura skončí, paměť je uvolněna, takže lokální proměnná zaniká (a obsah je ztracen).
  - Při opětovném volání procedury je proměnná znovu vytvořena, ale samozřejmě v ní není předchozí obsah.

Kolize jmen

- Rozhodne-li se programátor pro stejné pojmenování globální i lokální proměnné, je to teoreticky možné.
  - Pochopitelně není možné mít dvě proměnné se stejným jménem „vedle sebe“ (program nelze přeložit).
- Je-li v nějakém kódu použit identifikátor proměnné, který mohl vztahovat k různým proměnným (typicky globální a lokální proměnné stejného jména), vždy se vztahuje k proměnné lokální.
- **Napsat takový kód, ze kterého je viditelných více různých proměnných se stejným jménem, je velmi špatný nápad!!**
  - Řešení je jednoduché – přejmenování proměnné.

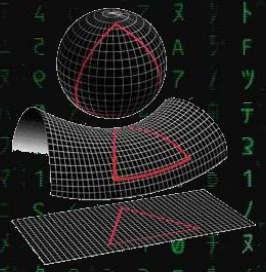
Cvičení

- Napište program, který přečte ze vstupu posloupnost kladných čísel (ukončenou nulou) a vypíše největší z nich. Poté načte další, ze které vypíše nejmenší. Potom ještě jednu, ze které vypíše opět nejvyšší číslo.
- ** Upravte program tak, aby nejprve vypsal nejmenší číslo, pak nejvyšší a nakonec zas nejnižší (opět přečte ze vstupu tři posloupnosti).*

```
program findExtreme;

procedure findMax();
  var
    input : integer;
    max   : integer;
begin
  readln(input);
  max := input;
  while (0 < input)
    do
    begin
      readln(input);
      if (max < input)
        then max := input;
    end;
  writeln(max);
end;

procedure findMin();
  var
    input : integer;
    min   : integer;
begin
  readln(input);
  min := input;
  while (0 < input)
    do
    begin
      readln(input);
      if ((0 < input) AND (min > input))
        then min := input;
    end;
  writeln(min);
end;

begin
  findMax();
  findMin();
  findMax();
end.
```

Parametrizace procedur

- Uvažme program, který provádí několik výpočtů a jejich výsledky vypisuje na výstup. Některé výsledky mohou být „kritické“ (například takové, které nenáleží do námi určeného intervalu). Kritický výsledek chceme označit vykřičníkem.
- Abychom při každém výpisu nemuseli ověřovat, zda je výsledek kritický, chceme použít proceduru pro výpis, která automaticky rozhodne o tom, zda je výsledek kritický a případně vypíše i vykřičník.
- Ověřování „kritičnosti“ bude sice pro všechna čísla stejné, ale procedura musí nějak „vědět“, které číslo má ověřit a poté vypsat (případně s vykřičníkem).



Co jsou parametry procedury?

- Parametry procedury jsou lokální proměnné, jejichž hodnotu lze nastavit ve chvíli, kdy je procedura volána.
  - Tato hodnota je nastavována z kódu, který proceduru volá.
- Krom toho, že jsou tyto proměnné při volání procedury nastaveny na nějakou hodnotu, mají všechny vlastnosti obyčejných lokálních proměnných.
  - Jejich životnost je tedy také omezena a po skončení procedury zanikají.
  - Lze do nich normálně přiřazovat hodnoty.

Předávání parametrů hodnotou

- Parametry procedury se deklarují v závorce za identifikátorem procedury.
- Při volání procedury se do závorky píší **výrazy** odpovídajících typů.
  - Zadané výrazy jsou **zkopírovány** do parametrů (v pořadí, v jakém byly zadány – musí jich být stejný počet).
- Při volání procedury *identifier* (vpravo) je *vb* boolean a *vi* integer, protože parametr *b* je boolean a *i* integer.
- Při volání *foo* (vpravo) je *0 = rmndr* výraz typu boolean a *loop1* integer.

```
procedure identifier(
  b : boolean;
  i : integer);
begin

end;
```

```
identifier(vb, vi);
```

```
program foobar;

var
  loop1 : integer;
  rmndr : integer;

procedure foo(
  bar : boolean;
  baz : integer);
begin
  if (bar)
    then writeln(baz)
    else writeln('x');
end;

begin
  for loop1 := 0 to 69
    do
    begin
      rmndr := loop1 mod 2;
      foo(0 = rmndr, loop1);
    end;
end.
```

Cvičení

- Napište program, který ze vstupu přečte délku strany čtverce a vypočítá jeho obsah. Je-li tento obsah větší než 32, vypíše za něj vykřičník.
- Napíšete-li výpis jako proceduru, usnadní Vám to další zadání.
- ** Modifikujte program, aby spočítal obsah čtverce, pak obsah obdélníku a poté opět obsah čtverce (pro každý výpočet jsou zadány nové hodnoty). Je-li obsah větší než 32, program za něj vypíše vykřičník. Je-li obsah menší než 8, program vypíše vykřičník před něj.*

```
program squareSurface;

var
  length : integer;

procedure print(output : integer);
begin
  write(output);
  if (32 < output)
    then write('!');
  writeln();
end;

begin
  write('Delka strany: ');
  readln(length);
  print(length * length);
end.
```

Reference

- Představte si program, ve kterém často potřebujete například zaměňovat hodnoty dvou proměnných.
- Protože se tato činnost děje v programu často, jsou k ní potřeba tři příkazy a ještě další proměnná, bylo by dobré mít proceduru, která tuto záměnu provede.
  - Jenže když napíšeme proceduru, která vymění obsah svých dvou parametrů, co se stane?
    - Do parametrů se zkopírují hodnoty proměnných, jejichž obsah chceme zaměnit.
    - V proceduře se provede záměna hodnot **parametrů** (tedy **ne původních proměnných**).
    - Procedura skončí a parametry zmizí (byly to lokální proměnné). Z vnějšího pohledu se „nestalo nic“.

Předávání parametrů odkazem

- Předávání parametrů odkazem se od předávání hodnotou syntakticky liší tím, že se před proměnnou napíše klíčové slovo var.
- Do proměnné se nic nekopíruje, nýbrž je tato proměnná jen jiné pojmenování pro proměnnou původní (stejná paměť má dvě různá jména). Ta po skončení procedury nezaniká!
- Procedura *swapInts* (vpravo) pracuje s (přejmenovanými) proměnnými *foo* a *bar*, ne jejich kopiemi.

```
procedure identifier(
  var b : boolean;
  var i : integer);
begin

end;
```

```
identifier(pb, pi);
```

```
program swap;

var
  foo : integer;
  bar : integer;

procedure swapInts(
  var a : integer;
  var b : integer);
  var
    c : integer;
begin
  c := a;
  a := b;
  b := c;
end;

begin
  readln(foo);
  readln(bar);
  swapInts(foo, bar);
  writeln(foo);
  writeln(bar);
end.
```

Proměnné a výrazy

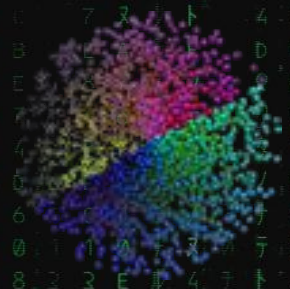
- Proměnná je pojmenovaný kus paměti (s daty).
  - Z proměnné lze data **číst**, ale též do ní i **zapisovat**.
- Výraz je něco, co má nějakou hodnotu, kterou lze spočítat a **číst**, avšak **do výrazu nelze zapisovat**.
  - příklady výrazů: 2, 2 * 4, 8 + a, b, c = d, true, 'x', 3.14
- Proměnná je výraz. Ne každý výraz je ale proměnná.
- Proměnná může být použita všude tam, kde je vyžadován výraz stejného typu.
  - Hodnota výrazu je pak rovna hodnotě proměnné.
- Výraz obecně ale nelze použít tam, kde proměnná.
  - Do proměnné je možné přiřadit, do výrazu ne.

Výrazy nelze předávat jako odkazy

- Jsou-li proceduře předávány argumenty hodnotami, při jejím volání se hodnoty **zkopírují** do parametrů procedury (lokálních proměnných).
  - Tyto hodnoty mohou být i výrazy, například výraz 2 lze zkopírovat do proměnné typu integer.
- Při předávání odkazem se nic nekopíruje. Parametry procedury (teoreticky lokální proměnné) jsou jen jiným pojmenováním pro proměnné, které byly proceduře předány pro volání.
  - Změní-li se tato proměnná v proceduře, změní se i v místě, odkud byla volána.
  - Například výraz 2 se nemůže změnit, je to pořád 2.

Funkce

- Funkce mají všechny vlastnosti a možnosti procedur s jedním rozšířením navíc.
- Funkce mají **návratovou hodnotu**, která umožňuje předat kódu, odkud byla volána, nějaký výsledek.
- Na rozdíl od procedury je funkce vždy nějakého typu (může mít všechny typy, které mohou mít proměnné) a hodnotu tohoto typu může po svém ukončení předat kódu, který funkci volal.
- Návratová hodnota je z pohledu funkce lokální proměnná (podobně jako parametry).
- Jinak jsou funkce totéž, co procedury (také mají své parametry, lze je předávat odkazem a hodnotou, ….).



Syntaktický zápis funkce

- Za závorkou s parametry následuje typ funkce, resp. návratové hodnoty.
- Kdekoliv v kódu funkce lze nastavit návratovou hodnotu přiřazením do proměnné, jejíž jméno se shoduje se jménem funkce (z té proměnné lze i číst, je to lokální proměnná).
- Návratovou hodnotu lze pak v kódu používat jako výraz (např. přiřadit).
- Funkce *pow* (vpravo) počítá výraz $base^{exp}$, takže do proměnné *foo* je přiřazen výraz bar^{baz}.

```
function identifier() : type;
begin
  identifier := expression;
end;
```

```
variable := identifier();
```

```
program foobar;

var
  foo : integer;
  bar : integer;
  baz : integer;

function pow(
  base : integer;
  exp  : integer) : integer;
var
  loop1 : integer;
begin
  pow := 1;
  for loop1 := 1 to exp
    do pow := pow * base;
end;

begin
  readln(bar);
  readln(baz);
  foo := pow(bar, baz);
  writeln(foo);
end.
```

Cvičení

- Napište program, který dostane na vstupu číslo a rozhodne, zda se jedná o prvočíslo.
- Použijete-li k řešení příkladu funkci, usnadní to práci na dalším příkladu.
  - Návratovou hodnotu booleovské funkce lze ověřit podmínkou.

```
program parity;

var input : integer;

function isPrime(num : integer) : boolean;
  var
   loop1 : integer;
begin
  isPrime := true;
  for loop1 := 2 to (num - 1)
    do
    if (0 = (num mod loop1))
      then isPrime := false;
end;

begin
  readln(input);
  if (isPrime(input))
    then writeln('PRIME')
    else writeln('NOT A PRIME');
end.
```

- ** Napište program, který dá uživateli vybrat, zda chce ověřit prvočíselnost, nebo paritu (sudost/lichost). Např. uživatele vyzve, aby zadal 0 pro prvočíselnost a 1 pro paritu. Poté ověří vybranou vlastnost vstupu.*

Zkrácená syntax deklarace proměnných

- Následující deklarace funkcí jsou ekvivalentní.

```
function foo(bar : integer; baz : integer);
begin
end;
```

```
function foo(bar, baz : integer);
begin
end;
```

  - Takto lze deklarovat libovolný počet proměnných stejného typu.
- Klíčové slovo var (pro předávání parametrů odkazem) je nutné použít vždy před skupinou proměnných a pak toto klíčové slovo platí pro všechny z nich.
  - V zápisu *a, b, c, d : integer; var e, f: integer; g, h : integer; i, j : integer; k, l : boolean; var m: boolean; var n : boolean;* jsou proměnné *a*, *b*, *c*, *d*, *g*, *h*, *i*, *j*, *k* a *l* inicializovány hodnotami a proměnné *e*, *f*, *m*, *n* jsou odkazy.

Rekapitulace

- Funkce a procedury představují kód mimo hlavní program, ze kterého je můžeme **volat**.
- Funkcím a procedurám lze předávat **parametry**, což jsou lokální proměnné, které lze nastavit při volání.
  - Parametry lze předávat **odkazem** a **hodnotou**.
    - Při předávání hodnotou se výraz do parametru **kopíruje**, při předávání odkazem je použita rovnou proměnná, která byla funkci či proceduře předána.
    - Změny parametrů, které byly předány odkazem, se projevují i mimo kód funkce/procedury.
- Funkce se liší od procedury **návratovou hodnotou**, která má svůj typ, ve funkci je to lokální proměnná; funkci pak lze ve volajícím kódu použít jako výraz.