Obsah

1 Znakové a řetězcové datové typy

Znakové a řetězcové datové typy

- Zatím umíme zpracovávat údaje s typy:
  - Celá čísla (v Pascalu *integer*, *short*, ...)
  - Čísla v plovoucí řádové čárce (v Pascalu *real*, *double*, ...)
  - Pravdivostní hodnota (v Pascalu *boolean*)
- Mimo nich je potřeba zpracovávat také textové údaje:
  - Typ „znak“ (v Pascalu *char*)
  - Typ „řetězec“ (v Pascalu *string*)

Znakové typy

- Většinou jde o typ velikosti 1B
  - Někdy se objevují i vícebajtové typy pro uložení řetězců v Unicode
- Některé jazyky (např. C) nerozlišují mezi typem číslo a znak

Funkce pro práci s jednotlivými znaky:

- *Chr* – převod čísla (typ byte) na znak (typ char)
- *Ord* – převod znaku (typ char) na číslo

1.1 Způsob implementace řetězců

Řetězcové znakové typy

- Řetězec je posloupnost jednotlivých znaků
- Problém s tím, jak je řetězec dlouhý
  - V řetězci je přítomný ukončovací znak řetězce. Např.: | z_1 | z_2 | z_3 | ... | 0 |
    z_i - i-tý znak řetězce 0 - ukončovací znak s ordinální hodnotou 0 Používá se např. v jazyce C. Odtud cstring nebo lépe „zero terminated string“.
  - V řetězci je přímo zapsána jeho délka (Pascal). Např.: | d | z_1 | z_2 | z_3 | ... |
    d - délka řetězce (počet znaků) z_i - i-tý znak řetězce Používá se v jazyce Pascal.

1.2 Některé funkce pro práci s řetězci

Některé funkce pro práci s řetězci v Pascalu

- `function Length(S:String):Integer` – funkce vrací počet znaků v řetězci (konstanta d v implementaci řetězce)
- `function Pos(Substr,S:String):Byte` – funkce najde pozici jednoho řetězce v druhém
- `function Copy(S:String;Index,Count:Integer):String` – funkce získá podřetězec z jiného řetězce
- `procedure Delete(var S:String;Index,Count:Integer)` – procedura vymaže z řetězce jeho část
- `procedure Insert(Source:String;var S:String;Index:Integer)` – procedura vloží na určitou pozici řetězce jiný řetězec
- `function Concat(S1[,S2...]:String):String` – funkce spojí řetězce za sebe, stejný význam má i operátor +
- `procedure Val(S:String;var V;var Code:Integer)` – procedura převede znakový řetězec na znakovou hodnotu
- `procedure Str(X; var S:String)` – procedura převede číselnou hodnotu na znakový řetězec (podobně jako Write)

2 Strukturované programování

Strukturované programování

- *Edsger Dijkstra* (1930-2002, hol. prof.) v roce 1968 publikoval článek „Příkaz GOTO je považován za škodlivý“ („GOTO Statement Considered Harmful“), čímž položil základy „strukturovaného programování“.
- Struktura každého programu může být vyjádřena pomocí následujících obratů:
  - Posloupnost instrukcí (známe)
  - Cykly (známe)
  - Větvení (známe)
  - Moduly (mj. procedury a funkce)

2.1 Proč strukturované programování?

Proč strukturované programování?

- Zpřehlednění zápisu v programovacím jazyce
- Možnost znovupoužití částí programu
  - V rámci jednoho programu
  - V jiném programu
- Možnost využití vyšších programátorských technik (např. rekurze)

2.2 Funkce a procedury

Funkce a procedury

- Funkce a procedury jsou samostatné programové bloky
- Funkce má výstupní (návratovou) hodnotu a (většinou) vstupní parametry

```
function spocitejTretiMocninu(x:real):real;
begin
  spocitejTretiMocninu:=x*x*x;
end;
```

- Procedura je funkce bez návratové hodnoty

```
procedure napisAhoj;
begin
  WriteLn('Ahoj');
end;
```

- Použití:

```
napisAhoj;
WriteLn(spocitejTretiMocninu(12.3));
```

2.3 Parametry procedur a funkcí

Parametry procedur a funkcí

- Záhlaví procedury (nebo funkce) určuje seznam *formálních parametrů*:

  ```
  procedure nazev(param; param; ... param);
  ```

- Každé `param` je skupina formálních parametrů, která může mít tvar:
  - `jmeno, ..., jmeno:typ` – parametry předávané hodnotou
  - *`var`* `jmeno, ..., jmeno:typ` – parametry předávané odkazem
- Formální parametr funguje jako lokální proměnná (viz dále) pro použití v těle funkce (procedury).
- Hodnota formálního parametru je nahrazena hodnotou *skutečného parametru*, který je uveden při volání funkce (procedury).

Parametry předávané hodnotou

Hodnota skutečného parametr předávaného *hodnotou* je *zkopírována* do nové paměťové pozice, se kterou se pracuje jako s formálním parametrem.

- Při volání procedury (funkce) může být jako skutečný parametr použit výraz, který má požadovaný datový typ (mj. proměnná, výsledek funkce, konstanta, ...).
- Hodnota formálního parametru je při ukončení funkce (procedury) zapomenuta.

Parametry předávané odkazem

Proměnná formálního parametru předávaného *odkazem* je v paměti na stejném místě jako skutečný parametr.

- Při volání procedury (funkce) *musí* být jako skutečný parametr použita proměnná.
- Hodnota formálního parametru je při ukončení funkce (procedury) zachována a „zkopírována“ do proměnné skutečného parametru.
- Může sloužit jako *výstup* z procedury (funkce).
- Použití také pro předávání větších datových struktur, se kterými funkce (procedura) pracuje.

2.4 Lokální a globální proměnné

Lokální a globální proměnné

- Program může mít (*globální*) proměnné.
  - Jejich platnost je omezena na hlavní blok programu a procedury (funkce) pod deklarací proměnné.
  - Deklarace `var` je mimo blok programu i funkcí a procedur.
- Funkce (procedura) může mít své vlastní (*lokální*) proměnné.
  - Jejich platnost je omezena pouze na funkci (proceduru), ve které jsou deklarovány a příp. na funkce a procedury, které tato funkce (procedura) obsahuje.
  - Deklarace `var` je v bloku funkce (procedury) mimo její tělo:

```
procedure test;
var prom:integer;
begin
  ...
end;
```

2.5 Překrývání proměnných

Překrývání proměnných

- Jména globálních a lokálních proměnných mohou být stejná.
- Nastane-li tento případ, postupuje se podle pravidla „Bližší košile než kabát“.
  - Tj. používá se lokální proměnná a ke globální není přístup.
- *Stejné chování jako lokální proměnné mají i formální parametry funkcí (procedur).*

Překrývání proměnných – příklad

```
procedure procedura1;
begin
   zde nelze použít proměnnou a
end;
var a:integer;
procedure procedura2;
begin
   zde se používá globální proměnná a
end;
procedure procedura3;
```

```
var a:real;
begin
   zde se používá lokální proměnná a
end;
begin
   zde se používá globální proměnná a
end.
```