

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Jméno a příjmení: Martin Sloup
Osobní číslo: A08N0111P
Obor: ININ/SWI
E-mail: msloup@students.zcu.cz
Datum odevzdání: 6. ledna 2009

Semestrální práce z předmětu

**Formální jazyky a překladače**

Doplnění cyklu DO – WHILE do jazyka PL/0

# Zadání

Do stávajícího překládače PL/0 doplňte cyklus „**DO** příkaz **WHILE** podmínka;“.

Mezi klíčovými slovy **DO** a **WHILE** musí být právě jeden příkaz, má-li jich tam být více, musí být uzavřeny v bloku **begin - end**.

# Analýza

Stejně jako již implementovaný cyklus s podmínkou na začátku, tedy **WHILE-DO** cyklus, i **DO-WHILE** cyklus s podmínkou na konci bude součástí bloku **statment** jazyka PL/0. Do bloku statment u syntaktického analyzátoru jazyka PL/0 doplníme následující větev:

DO

statment

WHILE

condition

Cyklus DO-WHILE není prakticky nutné popisovat. Obecně, pokud je pravdivá podmínka (blok condition) na konci cyklu, dojde ke skoku na začátek cyklu.

# Řešení

Řešení implementace DO-WHILE cyklu s podmínkou na konci je obdobné jako u WHILE-DO cyklu, teda cyklu s podmínkou na začátku. V DO-WHILE cyklu je pouze potřeba znát adresu začátku smyčky. Je to také adresa, kde začíná kód uvedený v bloku statment uvnitř cyklu. A také je nutné znát adresu příkazu, který následuje za cyklem. Nejlépe to vysvětlí následující pseudokód s návěštími:

ZACATEK: *kód bloku statment* - kód uvnitř cyklu *kód bloku condition* - podmínka cyklu JPC 0 KONEC - není-li podmínka pravdivá, pak se skočí na konec
 JMP 0 ZACATEK - jinak se skočí na začátek cyklu

KONEC: …

# Implementace

Jako cílový jazyk implementace mého řešení cyklu DO-WHILE jsem si při zadávání semestrální práce vybral programovací jazyk PHP. U implementace DO-WHILE cyklu jsem se inspiroval již implementovaným cyklem WHILE-DO. Znovupoužil jsem terminální symboly DO a WHILE tohoto cyklu. Při dopisování kódu překládače jsem použil kostru z kapitoly Analýza a napsal zdrojový kód, který generuje cílový kód představený v kapitole Řešení.

V mém řešení se používají dva skoky, u kterých je nutné dopředu znát adresu cílového skoku. U prvního skoku na začátek cyklu to není problém. Stačí si na začátku cyklu zapamatovat adresu, kde začíná statment blok. U druhého skoku, kdy se skáče na instrukci nacházející se za koncem cyklu, je to trochu problém. I když i to není problém, je to totiž adresa instrukce toho skoku plus dvě instrukce. Každopádně obecně správným řešením je zapamatovat si pozici instrukce skoku a při konci bloku cyklu stačí přistoupit k zpět k instrukci skoku a zapsat aktuální adresu instrukce (adresa instrukce následovaná za cyklem).

Zdrojový kód cyklu DO-WHILE překládače jazyka PL/0 byl napsán do metody statment za kód cyklu WHILE-DO. Níže se nachází výsledný zdrojový kód:

} elseif ($sym == Kdosym) { // začíná cyklus DO-WHILE
 $cx\_do\_begin = $cx; // uložíme si pozici pro skok na začátek cyklu
 getsym(); // přečteme následující symbol
 $pom = nuluj(); // vynulovaní pomocné proměnné
 sjednot($pom, $fsys); // sjednocení obsahu proměnné $pom s $fsys
 $pom[Ksemicolon] = 1; // nastavíme, že se má statment ukončit po nalezeni středníku
 statement($pom,$tx,$lev); // zavoláme blok statment

 if ($sym == Ksemicolon) getsym(); // skončilo-li to na středník, přečteme další znak
 else error(10); // jinak vyhodíme chybu o nesprávném symbolu příkazu

 if ($sym == Kwhilesym) getsym(); // je-li symbol WHILE, pokračujeme dál
 else error(33); // jinak vyhodíme chybu o nenalezeném symbolu WHILE

 // nyní zpracujeme blok condition
 $pom = nuluj(); // vynulovaní pomocné proměnné
 sjednot($pom, $fsys); // sjednocení obsahu proměnné $pom s $fsys
 $pom[Ksemicolon] = 1; // nastavíme, že se má condition ukončit po nalezeni středníku
 condition($pom,$tx,$lev); // zavoláme blok condition
 $cx\_jpc = $cx; // uložíme si pozici instrukce JPC pro pozdější použití
 gen(Kjpc,0,0); // pokud výsledek 0, přeskočíme následující instrukci
 gen(Kjmp,0,$cx\_do\_begin); // pokud výsledek 1, skočíme na začátek cyklu
 $code[$cx\_jpc]->a = $cx; // nastavíme do JPC adresu následující instrukce
} // Konec přidání cyklu DO-WHILE

Kromě tohoto kódu bylo nutné doplnit množinu počátečních symbolů příkazů o symbol DO, tedy:

$statbegsys[Kdosym] = 1;

A také přidat chybovou hlášku, pokud nebyl nalezen očekávaný symbol WHILE:

function error($n) {
 ...
 case 33 : echo 'ocekavano "while"';
 break;
 ...
}

# Příklady ukázkového kódu v jazyce PL/0

## Příklad 1

 0 JMP 0 1
 1 INT 0 6
 2 LIT 0 5
 3 STO 0 4
 4 LIT 0 0
 5 STO 0 3
 6 LIT 0 1
 7 STO 0 5
 8 LOD 0 3
 9 LOD 0 5
 10 OPR 0 2
 11 STO 0 3
 12 LOD 0 5
 13 LIT 0 1
 14 OPR 0 2
 15 STO 0 5
 16 LOD 0 5
 17 LOD 0 4
 18 OPR 0 10
 19 JMC 0 21
 20 JMP 0 8
 21 RET 0 0

var vysledek, vstup, iterator;

begin
 vstup:= 5;

 vysledek:= 0;
 iterator:= 1;
 do
 begin
 vysledek:= vysledek + iterator;
 iterator:= iterator + 1;
 end;
 while iterator < vstup;
end.

## Příklad 2

 0 JMP 0 1
 1 INT 0 4
 2 LIT 0 0
 3 STO 0 3
 4 LOD 0 3
 5 LIT 0 1
 6 OPR 0 2
 7 STO 0 3
 8 LIT 0 4
 9 LIT 0 4
 10 OPR 0 12
 11 JMC 0 13
 12 JMP 0 4
 13 RET 0 0

var a;

begin
 a:= 0;
 do
 a:= a + 1;
 while 4 > 4;
end.

## Příklad 3

 0 JMP 0 1
 1 INT 0 6
 2 LIT 0 0
 3 STO 0 3
 4 LOD 0 3
 5 LIT 0 1
 6 OPR 0 2
 7 STO 0 3
 8 LOD 0 3
 9 LIT 0 5
 10 OPR 0 10
 11 JMC 0 13
 12 JMP 0 4
 13 LOD 0 5
 14 LOD 0 3
 15 OPR 0 2
 16 STO 0 5
 17 LOD 0 4
 18 LIT 0 1
 19 OPR 0 2
 20 STO 0 4
 21 LOD 0 4
 22 LIT 0 5
 23 OPR 0 10
 24 JMC 0 26
 25 JMP 0 2
 26 RET 0 0

var x, y, vysledek;

begin
 do
 begin
 x:= 0;

 do
 x:= x + 1;
 while x < 5;

 vysledek:= vysledek + x;

 y:= y + 1;
 end;
 while y < 5;
end.

# Závěr

Dle zadání jsem do překládače jazyka PL/0 implementoval podporu pro cyklus DO-WHILE. Při návrhu a implementaci DO-WHILE cyklu jsem se inspiroval jeho bratříčkem, cyklem WHILE-DO. Při implementaci řešení mi chvíli trvalo, než jsem prozkoumal, jak je celý překládač poskládán dohromady. K čistotě kódu překladače PL/0 napsaném v jazyce PHP nemám moc výhrad. Pouze by bylo vhodné se držet jednotného formátování. Do semestrální práce jsem přidal pár příkladů ukazující použití DO-WHILE cyklu včetně ukázek zanoření.