Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Semestrální práci

Překladač jazyka do PL/0

Martin Kantořík kantorik@students.zcu.cz
Michal Tušl tuslm@students.zcu.cz

6. ledna 2018

Obsah

1 Zadání

Cílem práce je vytvoření vlastního jazyka a překladače pro tento jazyk. Překládat jsme se rozhodli do instrukční sady PL/0. Při vytváření jazyka jsme se snažili napodobit syntaxi jazyků java a C. Od vytvořeného jazyka jsme požadovali, aby uměl následující základní elementy:

- definice celočíselných proměnných
- definice celočíselných konstant
- přiřazení
- základní aritmetiku a logiku (+, -, *, /, AND, OR, negace a závorky, operátory pro porovnání čísel)
- cyklus (while)
- jednoduchou podmínku (if bez else)
- definice podprogramu (procedura, funkce, metoda) a jeho volání

Dále jsme se rozhodli jazyk rožšířit o další složitější konstrukce. Mezi složitější konstrukce, které jazyk umí patří:

- další typy cyklů (for, do...while, until, repeat...until)
- else větev
- datový typ boolean a logické operace s ním
- rozvětvená podmínka (switch, case)
- podmíněné přiřazení / ternární operátor (min = (a < b) ? a : b;)
- pole a práce s jeho prvky
- parametry předávané hodnotou
- návratová hodnota podprogramu
- komentáře

Za plně funkční překladač pro jazyk, které umí tyto konstrukce by mělo být uděleno 24 bodů. Další bonusový bod by mohl být za realizaci komentářů, které nebyly uvedeny v seznamu možných konstrukcí.

2 Popis řešení

3 Syntaxe

V této kapitole bude krátce okomentována a ukázána syntaxe jazyka.

3.1 Definice proměnných a přiřazení

Deklarace proměnných:

```
int cislo;
 boolean logika;
```

Deklarace konstant:

```
const int CISLO = 5;
const boolean LOGIKA = true;
```

Konstanty musí mít vždy přiřazenou hodnotu již při deklaraci.
Deklarace s přiřazením:

```
int cislo = 5;
boolean logika = true;
```

Přiřazení:

```
cislo = 5 + 3;
cislo = cislo + 1;
```

3.2 Podmínky

3.2.1 Podmínka if

Část programu ve větvi `if` se provede, pokud je splněná podmínka. V podmínce se může využívat všech logických operátorů, viz příklady. Zároveň je možné doplnit na konci větve `if` větev `else`, která se provede v případě, že není splněná podmínka.
Ukázka podmínky:

```
if(!(2 < 3 && 1 > 0) || 1 != 0)
{...}
else{...}
```

Podmínka musí být v závorkách následována ihned po příkazu `if`. Za podmínkou ve složených závorkách se pak nachází část kódu, který se má vykonat v případě splněné podmínky.

3.2.2 Switch

V podmínce `switch` se musí nacházet pouze celé číslo nebo proměnná `int`. Podle dané hodnoty se provede určitý `case` uvnitř `switch`. Zároveň lze na konci `switch` udělat větev `default`, která se provede v případě, že žádný `case` neodpovídal hodnotě v podmínce. Narozdíl od jazyků C a java, se vždy provede pouze jeden `case`.
Ukázka podmínky:

```
switch(2){
 case 1:
 case 2: int a = 2;
 default: int b = 0;}
```

3.2.3 Ternární operátor

Jazyk umožňuje i zkrácený zápis podmínky `if`, případně podmíněného přiřazení.
Ukázka ternární podmínky

```
(cislo < 2) ? cislo = 2 : cislo = 3;
```

Ukázka podmíněného přiřazení

```
cislo = (cislo < 2) ? 2 : 3;
```

3.3 Cykly

Cykly slouží k určitému opakování stejného kódu.

3.3.1 While

Cyklus, který se provádí dokud je splněná podmínka. Platí zde stejná pravidla jako v podmínce `if`.
Ukázka cyklu:

```
while(cislo < 3){
 cislo = cislo + 1;
}
```

3.3.2 Do...while

Podmínka se ověřuje až na konci cyklu, tedy program se vykoná vždy alespoň jednou.
Ukázka cyklu:

```
do{
```

```
 cislo = cislo + 1;
}while(cislo < 3);
```

3.3.3 Until

Podobný cyklus jako `while`, akorát se provádí pokud podmínka je nesplněná. Jakmile se podmínka splní, cyklus končí.
Ukázka cyklu:

```
until(cislo > 3){
 cislo = cislo + 1;
}
```

3.3.4 Repeat...until

Podobný cyklus jako `until`, akorát podmínka se ověřuje až na konci cyklu. Program se tedy vykoná alespoň jedenkrát.

3.3.5 For

Cyklus s určitým počtem opakování. Podmínka se skládá ze tří částí. V první části musí být deklarace proměnné s přiřazením počáteční hodnoty. V druhé části musí být podmínka, při její splnění se bude cyklus provádět. V poslední části je pak operace, která se provede na konci cyklu.
Ukázka cyklu:

```
for(int i = 0; i < 3; i = i + 1){
 ...
}
```

3.4 Pole

3.5 Funkce

Program lze členit do podprogramů pomocí funkcí. Funkce musí být definovány na začátku programu, při definici je důležité klíčové slovo `function`. Funkcím lze předávat parametry a zároveň funkce můžeš vracet jednu hodnotu, viz příklad.
Ukázka funkce:

```
int function soucet(int a, int b){
 return a + b;
}
```

Volání funkce:

```
int c = soucet(1, 2);
```

3.6 Komentáře

Komentáře slouží k označení části kódu, která se nebude překládat do instrukcí. Realizovány byly blokové komentáře, které jsou označeny sekvencí `/*` na začátku bloku a `*/` na konci bloku.
Ukázka komentářů:

```
/* tohle je komentar */
```

4 Testovací příklady

Testování funkčnosti řešení bylo realizováno pomocí testovacích souborů, pro které jsme měli správné posloupnosti instrukcí. Při změnách v překladači se pak pouštěl překlad těchto testovacích souborů a provnávali se výstupní instrukce se správnými.
Testovací soubory lze najít ve složce *tests/testFiles*, programy napsané v našem jazyce mají příponu *.sll*. Přeložené programy do instrukční sady PL/0 mají příponu *.pl*.
Některé kratší ukázky a výstupní instrukce přiložím zde.

4.1 Test přiřazení

Program:

```
int a = 5;
int mn, ob = 5 + a, i = 3, or;
boolean c = true;
a = 3;
const int TEST = 4;
int b = TEST;
int d = b;
int e = b;
if (a < 5) {
 b = 3;
} else {
 b = 8;
}
c = a == b;
```

Instrukce:

```
0 JMP 0 1
1 INT 0 13
2 LIT 0 5
3 STO 0 3
4 LIT 0 0
5 STO 0 4
6 LIT 0 5
7 LOD 0 3
```

```
8 OPR 0 2
9 STO 0 5
10 STO 0 6
11 LIT 0 0
12 STO 0 7
13 LIT 0 1
14 STO 0 8
15 LIT 0 3
16 STO 0 3
17 LIT 0 4
18 STO 0 9
19 LOD 0 9
20 STO 0 10
21 LOD 0 10
22 STO 0 11
23 LOD 0 10
24 STO 0 12
25 LOD 0 3
26 LIT 0 5
27 OPR 0 10
28 JMC 0 32
29 LIT 0 3
30 STO 0 10
31 JMP 0 34
32 LIT 0 8
33 STO 0 10
34 LOD 0 3
35 LOD 0 10
36 OPR 0 8
37 STO 0 8
38 RET 0 0
```

4.2 Test cyklu a podmínek

Zde je otestovaný pouze cyklus `for` a podmínka `if`. Všechny cykly jsou testovány v souboru *tests/testFiles/cykly/testCycles.sll*.
Program:

```
int i;
for(int k = 0; k < 3; k = k + 1){
```

```
 if(i < 3){
 i = i - 1;
 }
 else{
 i = i * k;
 }
}
```

Instrukce:

```
0 JMP 0 1
1 INT 0 5
2 LIT 0 0
3 STO 0 3
4 LIT 0 0
5 STO 0 4
6 LOD 0 4
7 LIT 0 3
8 OPR 0 10
9 JMC 0 28
10 LOD 0 3
11 LIT 0 3
12 OPR 0 10
13 JMC 0 19
14 LOD 0 3
15 LIT 0 1
16 OPR 0 3
17 STO 0 3
18 JMP 0 23
19 LOD 0 3
20 LOD 0 4
21 OPR 0 4
22 STO 0 3
23 LOD 0 4
24 LIT 0 1
25 OPR 0 2
26 STO 0 4
27 JMP 0 6
28 RET 0 0
```

4.3 Testování polí

4.4 Testování funkcí

Ukázka funkce pro součet dvou čísel.
Program:

```
int function soucet (int a, int b) {
 return a + b;
}
int c = soucet(1, 2);
```

Instrukce:

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

5 Závěr

Semestrální práci se podařilo úspěšně dokončit. Nicméně během tvorby jazyka a překladače jsme narazili na řadu problémů a obtíží. Nejhorší byl začátek, kdy bylo potřeba sestavit gramatiku jazyka a zprovoznit nástroj ANTLR pro parsování programu. Dále se pak naučit jak funguje PL/0, a co znamenají jednotlivé instrukce.

Práce byla poměrně rozsáhlá a bylo potřeba mnoho úsilí, aby byly splněny minimální požadavky na rozsah. Na druhou stranu byla práce originální a poskytla nám pohled do fungování překladačů, jak se z nám známeho programovacího jazyka stane posloupnost strojových instrukcí použitelných pro procesor.

Vzhledem k tomu, že práce byla dělána ve dvojicích, se bylo vždy možno poradit, když jsme si nevěděli rady. Zároveň si myslíme, že i komunikace byla lepší, než kdyby jsme tuto práci dělali v početnějším týmu. Ačkoliv si nedovedeme představit, kolik času by bylo potřeba nad touto prací strávit, abychom dosáhli maximálního počtu bodů.

Celý projekt byl veden na githubu na adrese FJP_super_language[1]. Za semestrální práci, tak jak bylo řečeno v kapitole 1, by jsme očekávali 24 bodů. Případně bonusový bod za implementaci komentářů.

[1] Adresa v případě nefunkčnosti odkazu: https://github.com/tuslm/FJP_super_language