FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/FJP - Semestrální práce

Překladač vlastního jazyka do instrukcí PL/0

Pavel Kotva - A17N0077P
kotva@students.zcu.cz

Jan Kohlíček - A17N0075P
kohl@students.zcu.cz

7. ledna 2018

Obsah

Kapitola 1

Zadání

Cílem práce bude vytvoření překladače zvoleného jazyka. Je možné inspirovat se jazykem PL/0, vybrat si podmnožinu nějakého existujícího jazyka nebo si navrhnout jazyk zcela vlastní. Jazyk bude překládán do instrukcí PL/0.

Jazyk musí mít základní konstrukce:

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

Rozšiřující konstrukce:

- každý další typ cyklu
- datový typ boolean a logické operace s ním
- else větev
- násobné přiřazení
- paralelní přiřazení
- parametry předávané hodnotou
- návratová hodnota podprogramu

Kapitola 2

Syntaxe

2.1 Proměnné a přiřazení

Jazyk je staticky typový, všechny proměnné musíme nejprve deklarovat s jejich datovým typem.

Datové typy:

int - označení celočíselného datového typu.

boolean - logický datový typ. Boolean nabývá dvou hodnot: **true** (pravda) a **false** (nepravda).

Deklarace proměnné bez přiřazení:

```
int x;
boolean y;
```

Deklarace proměnné s přiřazení:

```
boolean x := true;
boolean y := false;
int z := (3 * 3) + 1;
```

Deklarace proměnných s násobným přiřazením:

```
int a;
int b;
int c;
a, b, c := 5;
```

Deklarace proměnných s paralelním přiřazením:

```
int a;
int b;
[a, b] := [10, 15];
```

2.1.1 Konstanty

Konstanty jsou deklarovány s klíčovým slovem **const** a mohou obsahovat jen celé číslo.

```
const KONSTANTA := 5;
```

2.2 Podmínky (větvení)

Podmínky zapisujeme pomocí klíčového slova **if**, za kterým následuje logický výraz. Pokud je výraz pravdivý, provede se následující příkaz. Pokud ne, následující příkaz se přeskočí a pokračuje se větví **else**. Zapsat větev **else** je povinné.

```
int i := 1;

if(i > 1)
start
    i := 2;
end
else
start
    i := 4;
end
```

2.2.1 Operátory

Relační operátory, které můžeme ve výrazech používat:

Operátor	Zápis
Rovnost	=
Je ostře větší	>
Je ostře menší	<
Je větší nebo rovno	>=
Je menší nebo rovno	<=
Nerovnost	!=
Obecná negace	!
A zároveň	&&
Nebo	||

Tabulka 2.1: Relační operátory

Podmínky je možné skládat a to pomocí dvou základních logických operátorů:

Operátor	Zápis
A zároveň	&&
Nebo	||

Tabulka 2.2: Logické operátory

2.3 Cykly

2.3.1 While

Prvním typem cyklu je **while** cyklus funguje jednoduše opakuje příkazy v bloku dokud platí podmínka. Syntaxe cyklu je následující:

```
int i := 0;

while(i < 4)
start
    i := i + 1;
end
```

2.3.2 Do–while

Posledním typem cyklu je **do–while**. Je téměř stejný jako while, ale kontrolní podmínka je umístěna až na konec cyklu. Máme tedy jistotu, že minimálně jednou cyklus vždy proběhne. Syntaxe cyklu je následující:

```
int i := 0;

do
start
    i := i + 1;
end while(i < 4)
```

2.4 Procedury

Procedura je logický blok kódu, který jednou napíšeme a poté ho můžeme libovolně volat bez toho, abychom ho psali znovu a opakovali se. Proceduru deklarujeme v globálním prostoru, někde nad **main**.

2.4.1 Procedura bez parametrů

Syntaxe procedura bez parametrů je následující:

```
procedure nazevprocedury()
start
    int b := 5;
end

call nazevprocedury();
```

2.4.2 Procedura s parametry

Procedura může mít také libovolný počet vstupních parametrů (někdy se jim říká argumenty), které píšeme do závorky v její definici. Rozšíříme tedy stávající proceduru o parametr **int a** a ten potom přidáme s konkrétní hodnotou do volání procedury:

```
procedure nazevprocedury(int a)
start
```

```
    int b := 5 + a;
end

call nazevprocedury(3);
```

2.4.3 Procedura s návratovou hodnotou

Procedura může dále vracet nějakou hodnotu, správně by se ji mělo říkat funkce, ale z historických důvodů a zachování kompatibility nazýváme procedura. Procedura může vracet právě jednu hodnotu pomocí příkazu **return**. Syntaxe je následující:

```
procedure nazevprocedury(int a)
start
    int b := 5 + a;
end
return b;

int y;
call nazevprocedury(3)(y);
```

Kapitola 3

Implementace

Pro implementaci překladače byl vybrán jazyk **Java** a nástroj pro tvorbu syntaktických analyzátorů, kompilátorů a překladačů gramatiky **ANTLR**. **ANTLR 4.7** byl vybrán pro jeho snadnou a rychlou použitelnost.

Popis adresářové struktury:

+ `docs` - dokumentace
+ `src` - zdrojové kódy
  . `FJPLexer.g4` - obsahuje tokeny
  . `FJPParser.g4` - obrahuje gramatiku

Kapitola 4

Vzorové programy a generované instrukce

4.1 Program: Podmínky

Zdrojový kód:

```
kod
```

Instrukce:

```
instrukce
```

4.2 Program: Cykly

Zdrojový kód:

```
kod
```

Instrukce:

```
instrukce
```

4.3 Program: Procedury

Zdrojový kód:

`kod`

Instrukce:

`instrukce`

Kapitola 5

Závěr

V semestrální práci byl vytvořen překladač do instrukcí **PL/0**. Splnily jsme všechny body, které jsme si určily v zadání. Syntaxe je originálním mixem syntaxe **C** a **Pascalu**. Zdrojové kódy jsou k dispozici na githubu: `https://github.com/Ahantuon/fjp`