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

Každý program musí obsahovat vždy základní strukturu. Blok **main** je výchozí místo spuštění programu, **start** označuje začátek a **end** konec bloku. Základní syntaxe vypadá následovně:

```
start

    main
    start

    end
end
```

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

3.1 Gramatika jazyka

```
program     : START constant* globals procedure* main END;

constant    : CONST ID ASSIGN INT_VALUE SEMI;
globals     : variable*;
procedure   : PROCEDURE ID LPAREN arguments RPAREN body return_val;
return_val  : RETURN ID SEMI;

int_var     : INT ID (ASSIGN INT_VALUE)? SEMI;
boolean_var : BOOLEAN ID (ASSIGN BOOLEAN_VALUE)? SEMI;

arguments   : argument (COMMA argument)*;
argument    : INT ID | BOOLEAN ID;
body        : START locales statement* END;
locales     : variable*;
variable    : int_var | boolean_var;
statement   : call
            | assigment
```

```
            | assigment_p
            | re_until
            | do_while
            | while_do
            | if_else;

call        : CALL ID LPAREN var (COMMA var)* RPAREN (LPAREN return_id RPAREN)? SEMI;
return_id   : ID;
re_until    : REPEAT START (call | assigment | assigment_p)* END UNTIL LPAREN expression RPAREN;
do_while    : DO START (call | assigment | assigment_p)* END WHILE LPAREN expression RPAREN;
while_do    : WHILE LPAREN expression RPAREN start_do (call | assigment | assigment_p)* END;
start_do    : START;
if_else     : IF LPAREN expression RPAREN
                START (call | assigment | assigment_p | re_until | while_do | do_while)*
                END else_part START (call | assigment | assigment_p | re_until | while_do | do_while)* END;

else_part   : ELSE;
assigment   : ID (COMMA ID)* ASSIGN (var | expression) SEMI;
assigment_p : LBRACK ID (COMMA ID)* RBRACK ASSIGN
                LBRACK (var | expression)(COMMA (var | expression))* RBRACK SEMI;

var         : NEG? LPAREN expression RPAREN | value | ids;
value       : INT_VALUE | BOOLEAN_VALUE;
ids         : ID;
expression  : simpleExp ((EQUAL | NOT_EQUAL | LT | LE | GE | GT) simpleExp)*;
simpleExp   : term ((ADD | SUB | OR) term)*;
term        : var ((AND | MUL | DIV) var)*;

main        : MAIN body;
```

Kapitola 4

Vzorové programy a generované instrukce

4.1 Program: Podmínky

Zdrojový kód:

```
start
    const KONSTANTA := 5;
    int x := 2;
    int y;
    boolean xx := true;
    boolean yy;

    main
    start
        yy := true;
        [x, y] := [10, 15];
        if(xx && KONSTANTA>3)
        start
            y:= 5;
        end
        else
        start
            y:= 3;
        end
```

```
    end
end
```

Instrukce:

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

```
33 RET 0,0
```

4.2 Program: Cykly

Zdrojový kód:

```
start
    const KONSTANTA := 5;
    int x := 2;

    main
    start
        int b;
        int a := 3;

        while (a = 3)
        start
            a := a + 1;
        end

        do
        start
            x := x * ((2 + a) * 3);
            a, b := a + 1;
        end while (!(a < KONSTANTA))
    end
end
```

Instrukce:

```
0 JMP 0,1
1 INT 0,4
2 LIT 0,2
3 STO 0,3
4 JMP 0,5
5 INT 0,2
6 LIT 0,0
```

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

4.3 Program: Procedury

Zdrojový kód:

```
start
    procedure secti(int a, int b)
    start
        int c;
        c := a + b;
    end
    return c;

    main
    start
        int y;

        call secti(3, 4)(y);
    end
end
```

Instrukce:

```
0 JMP 0,1
1 INT 0,3
2 JMP 0,18
3 INT 0,5
4 LIT 0,3
5 STO 0,3
6 LIT 0,4
7 STO 0,4
8 INT 0,1
9 LIT 0,0
10 STO 0,5
11 LOD 0,3
12 LOD 0,4
13 OPR 0,2
14 STO 0,5
15 LOD 0,5
```

```
16 STO 1,4
17 RET 0,0
18 INT 0,1
19 LIT 0,0
20 STO 0,3
21 CAL 1,3
22 RET 0,0
```

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`