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

Semestrální práce z KIV/FJP

Překladač navrženého jazyka Newton

Plzeň 2017 Lukáš Černý, Štěpán Baratta

Obsah

1 Členové týmu

Jméno	Studijní číslo	Email
Lukáš Černý	A17N0065P	luccerny@students.zcu.cz
Štěpán Baratta	A17N0061P	BarattaStepan@gmail.com

Odkaz na Github: xxx

2 Zadání

Cílem této semestrální práce je vytvoření překladače vlastního nebo již existujícího jazyka. Dále je nutné zvolit, pro jakou architekturu bude jazyk překládán.

Mezi základní podmínky, které jazyk musí splňovat, jsou následující konstrukce:

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

3 Popis řešení

Výstup programu bude posloupnost instrukcí vstupního programu. Instrukce jsou generované pro architekturu jazyka PL/0.

Seznam všech instrukcí je následující:

- LIT – 0,A ulož konstantu A do zásobníku
- OPR – 0,A proveď instrukci A
  - 1 unární minus
  - 2 +
  - 3 -
  - 4 *
  - 5 div - celočíselné dělení (znak /)
  - 6 mod - dělení modulo (znak
  - 7 odd - test, zda je číslo liché
  - 8 test rovnosti (znak =)
  - 9 test nerovnosti (znaky <>)
  - 10 <
  - 11 >=
  - 12 >

  - 13 <=

- LOD L,A – ulož hodnotu proměnné z adr. L,A na vrchol zásobníku
- STO L,A – zapiš do proměnné z adr. L,A hodnotu z vrcholu zásobníku
- CAL L,A – volej proceduru A z úrovně L
- INT 0,A – zvyš obsah top-registru zásobníku o hodnotu A
- JMP 0,A – proveď skok na adresu A
- JMC 0,A – proveď skok na adresu A, je-li hodnota na vrcholu zásobníku 0
- RET 0,0 – návrat z procedury (return)

3.1 Vytvoření AST (abstract syntax tree)

Program je implementován v programovacím jazyce Java. Pro vytvoření lexikálního analyzátoru a parseru je použita knihovna ANTLR. Obecný postup při vytváření překladače pomocí ANTLR obsahuje vytvoření gramatiky, která bude popisovat zvolený programovací jazyk. Obsahuje tedy lexikální a parsovací pravidla. Z nich ANTLR vytvoří lexikální analyzátor, který čte vstupní text a rozdělí ho na jednotlivé části (tokeny). Dále jsou tyto tokeny předány parseru, který vytváří abstraktní syntaktický strom a interpretuje kód.

3.2 Procházení stromu

Abstraktní syntaktický strom, který generuje ANTLR je procházen pomocí návrhového vzoru visitor, kdy jsou postupně navštěvovány jednotlivé uzly stromu. Pro každý uzel (pravidlo) je vytvořena metoda, která vykoná potřebný kód pro dané pravidlo a vygeneruje instrukce.

4 Implementace

4.1 Implementovaná rozšířená funkčnost

4.1.1 Jednoduchá rozšíření

- cykly
  - for
  - while
  - do while
  - repeat until
- datový typ boolean
- rozvětvená podmínka `switch`
- násobné přiřazení
- ternární operátor
- paralelní přiřazení

4.1.2 Složitější rozšíření

- parametry předávané hodnotou

- návratová hodnota podprogramu

4.2 Konstrukce programu

4.2.1 for

gramatika

```
BeginFor Identifier Assign factor Colon factor (Colon Int)?
Do
        statement*
EndFor;
```

použití

```
for a = 1 : 10 : 2 do
        b = b + 1;
endfor
```

4.2.2 while

gramatika

```
BeginWhile expression Do statement* EndWhile;
```

použití

```
while a < 10 do
        b = b + 1;
endwhile
```

4.2.3 do while

gramatika

```
Do statement* BeginWhile expression;
```

použití

```
do
        b = b + 1;
while a > 1;
```

4.2.4 repeat until

gramatika

```
Repeat statement* Until expression;
```

použití

```
repeat
        b = b + 1;
until a < 1;
```

4.2.5 switch

gramatika

```
BeginSwitch simpleExpression Of
        caseStatement+
DefaultSwitch Colon statement EndSwitch;
```

použití

```
switch param of
    0: result = false;
    1: result = true;
    default: result = false;
endswitch
```

4.2.6 ternární operátor

gramatika

```
expression Ques expression Colon expression;
```

použití

```
bool a;
. . .
a = b < 3 ? true : false;
```

4.2.7 paralelní přiřazení

gramatika

```
CurlyBracketLeft Identifier (',' Identifier)* CurlyBracketRight
Assign CurlyBracketLeft simpleFactor (',' simpleFactor)*
CurlyBracketRight Semi;
```

použití

```
{a, b, c} = {1, 2, 5};
```

4.3 Vzorové programy

```
constant:
variable:
    int a;
    int i;
    int j;

    fnc int pocitej(int d)
        for i = 1 : 5 do
            for j = 1 : 5 do
                d = d + 1;
            endfor
        endfor

        return d;
    end

main()
    a = pocitej(0);
end
```

Výstup

1	INT	0	7
2	LIT	0	0
3	STO	0	4
4	LIT	0	0
5	STO	0	5
6	LIT	0	0
7	STO	0	6
8	JMP	0	39
9	INT	0	3
10	LIT	0	1
11	STO	1	5
12	LOD	1	5
13	LIT	0	5
14	OPR	0	13
15	JMC	0	36

16	LIT	0	1
17	STO	1	6
18	LOD	1	6
19	LIT	0	5
20	OPR	0	13
21	JMC	0	31
22	LOD	0	−1
23	LIT	0	1
24	OPR	0	2
25	STO	0	−1
26	LOD	1	6
27	LIT	0	1
28	OPR	0	2
29	STO	1	6
30	JMP	0	18
31	LOD	1	5
32	LIT	0	1
33	OPR	0	2
34	STO	1	5
35	JMP	0	12
36	LOD	0	−1
37	STO	1	3
38	RET	0	0
39	LIT	0	0
40	CAL	0	9
41	INT	0	−1
42	LOD	0	3
43	STO	0	4
44	RET	0	0

```
constant:
variable:
    bool a;
    bool result;

    fnc bool pocitej(int d)
        switch d of
            0: result = false;
            5: result = true;
            default: result = false;
        endswitch

        return result;
    end

main()
    a = pocitej(5);
end
```

Výstup

1	INT	0	6
2	LIT	0	0
3	STO	0	4
4	LIT	0	0
5	STO	0	5
6	JMP	0	29
7	INT	0	3
8	LOD	0	−1
9	STO	1	3
10	LOD	1	3
11	LIT	0	0
12	OPR	0	8
13	JMC	0	17
14	LIT	0	0
15	STO	1	5
16	JMP	0	26
17	LOD	1	3
18	LIT	0	5
19	OPR	0	8

20	JMC	0	24
21	LIT	0	1
22	STO	1	5
23	JMP	0	26
24	LIT	0	0
25	STO	1	5
26	LOD	1	5
27	STO	1	3
28	RET	0	0
29	LIT	0	5
30	CAL	0	7
31	INT	0	−1
32	LOD	0	3
33	STO	0	4
34	RET	0	0

5 Závěr