



FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY



Semestrální práce

Překladač jazyka Ligma

Milan Janoch a Jakub Pavlíček



Obsah

1	Zadání	2
2	Návrh jazyka	5
2.1	Zvolená rozšíření jazyka Ligma	5
2.2	Omezení jazyka	5
2.3	Konstrukce jazyka	6
2.3.1	Povinné	6
2.3.2	Rozšiřující	7
2.3.3	Vlastní	9
3	Implementace	10
3.1	Gramatika	10
3.2	Struktura projektu	10
3.3	TODO - podrobnosti k implementaci + grafíky	11
4	Testování	12
4.1	Lexikální analýza	12
4.2	Syntaktická analýza	12
4.3	Sémantická analýza	12
4.4	Generování PL/0 instrukcí	13
5	Uživatelská dokumentace	14
5.0.1	Prerekvizity	14
5.0.2	Překlad a spouštění	14
6	Závěr	16
	Seznam výpisů	17

Zadání

1

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. Dále je také potřeba zvolit si pro jakou architekturu bude jazyk překládán (doporučeny jsou instrukce PL/0, ale je možné zvolit jakoukoliv instrukční sadu pro kterou budete mít interpret).

Jazyk musí mít minimálně následující 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

Překladač který bude umět tyto základní věci bude hodnocen deseti body. Další body (alespoň do minimálních 20) je možné získat na základě rozšíření, jsou rozděleny do dvou skupin, jednodušší za jeden bod a složitější za dva až tři body. Další rozšíření je možno doplnit po konzultaci, s ohodnocením podle odhadnuté náročnosti.

Jednoduchá rozšíření (1 bod):

- každý další typ cyklu (for, do .. while, while .. do, repeat .. until, foreach pro pole)

- else větev
- datový typ boolean a logické operace s ním
- datový typ real (s celočíselnými instrukcemi)
- datový typ string (s operátory pro spojování řetězců)
- rozvětvená podmínka (switch, case)
- násobné přiřazení ($a = b = c = d = 3;$)
- podmíněné přiřazení / ternární operátor ($min = (a < b) ? a : b;$)
- paralelní přiřazení ($\{a, b, c, d\} = \{1, 2, 3, 4\};$)
- příkazy pro vstup a výstup (read, write - potřebuje vhodné instrukce které bude možné využít)

Složitější rozšíření (2 body):

- příkaz GOTO (pozor na vzdálené skoky)
- datový typ ratio (s celočíselnými instrukcemi)
- složený datový typ (Record)
- pole a práce s jeho prvky
- operátor pro porovnání řetězců
- parametry předávané hodnotou
- návratová hodnota podprogramu
- objekty bez polymorfismu
- anonymní vnitřní funkce (lambda výrazy)

Rozšíření vyžadující složitější instrukční sadu než má PL/0 (3 body):

- dynamicky přiřazovaná paměť - práce s ukazateli
- parametry předávané odkazem
- objektové konstrukce s polymorfním chováním
- instanceof operátor
- anonymní vnitřní funkce (lambda výrazy) které lze předat jako parametr

- mechanismus zpracování výjimek

Vlastní interpret (řádkový, bez rozhraní, složitý alespoň jako rozšířená PL/0) je za 6 bodů.

Kromě toho že by program měl fungovat se zohledňují i další věci, které mohou pozitivně nebo negativně ovlivnit bodování:

- testování - tvorba rozumné automatické testovací sady +3 body (pro inspiraci hledejte test suit pro LLVM nebo se podívejte na Plum Hall testy, ale jde skutečně jen o inspiraci, stačí výrazně jednodušší řešení).
- Kvalita dokumentace -x bodů až +2 body podle kvality a prohrěšků (vynechaná gramatika, nesrozumitelné věty, příliš chyb a překlepů, bitmapové obrázky pro diagramy s kompresními artefakty, ...).
- Vedení projektu v GITu -x bodů až +2 body podle důslednosti a struktury příspěvků.
- Kvalita zdrojového textu -x bodů až +2 body podle obecně znýmých pravidel ze ZSWI, PPA a podobně (magická čísla, struktura programu a dekompozice problému, božské třídy a metody, ...)

Návrh jazyka

2

2.1 Zvolená rozšíření jazyka Ligma

- cyklus do-while
- cyklus for
- cyklus repeat-until
- datový typ boolean a logické operace s ním
- else větev
- násobné přiřazení ($a = b = c = d = 3;$)
- parametry předávané hodnotou
- návratová hodnota podprogramu

2.2 Omezení jazyka

- Při deklaraci proměnné je třeba vždy nastavit hodnotu
- V hlavičce for cyklu se musí deklarovat proměnná
- Podporované datové typy proměnných: int, boolean
- Podporovaná aritmetika pro boolean - $\&\&$, $\|$, $!$, $()$, $==$, $!=$
- Funkce musí být definovány až pod samotnými příkazy v nejvrchnějším scope
- Identifikátor nesmí obsahovat speciální znaky (kromě `_`) a začínat číslem

2.3 Konstrukce jazyka

2.3.1 Povinné

2.3.1.1 Definice celočíselných proměnných

```
1 int a = 5;
```

2.3.1.2 Definice celočíselných konstant

```
1 const int b = -8;
```

2.3.1.3 Přiřazení

```
1 int a = 10;  
2 int b = a;
```

2.3.1.4 Základní aritmetika a logika

Aritmetika:

```
1 int a = 3;  
2 int b = 50;  
3  
4 int c = a + b;  
5 int d = b - a;  
6 int e = b / a;  
7 int f = a * b;  
8 int g = ---a;  
9 int h = +a;  
10 int i = (a + b - 10) % 5;
```

Operátory pro porovnání:

```
1 boolean a = 1 < 50;  
2 boolean b = 2 <= -5;  
3 boolean c = 5 >= 6;  
4 boolean d = 15 > 20;  
5 boolean e = 5 == 5;  
6 boolean f = 10 != 22;
```

2.3.1.5 Cyklus - while

```
1 int a = 0;
2 while (a < 10) {
3     a = a + 2;
4 }
```

2.3.1.6 Podmínka if (bez else)

```
1 int a = 18;
2 if (a != 0) {
3     a = a + 9;
4 }
```

2.3.1.7 Definice funkce a její volání

```
1 int res = foo();
2
3 func int foo() {
4     return 10;
5 }
```

2.3.2 Rozšiřující

2.3.2.1 Cyklus do-while

```
1 int a = 10;
2 do {
3     a = a + 8;
4 } while (a < 50);
```

2.3.2.2 Cyklus repeat-until

```
1 int a = 5;
2 repeat {
3     a = a + 1;
4 } until (a >= 10);
```

2.3.2.3 Cyklus for

```
1 for (int a = 0 to 10) {
2     ...
3 }
```


2.3.2.4 Else větev

```
1 if (false) {  
2     ...  
3 } else {  
4     ...  
5 }
```

2.3.2.5 Datový typ boolean a logické operace s ním

Aritmetika:

```
1 boolean a = true;  
2 boolean b = false;  
3  
4 boolean c = a && b;  
5 boolean d = a || b;  
6 boolean e = !a;  
7 boolean f = ((a || b) && b);
```

Operátory pro porovnání:

```
1 boolean a = true == false;  
2 boolean b = true != false;
```

2.3.2.6 Násobné přiřazení

```
1 int a = 5;  
2 int b = 6;  
3 int c = 10;  
4 int d = -10;  
5  
6 b = a = d = c = -80;
```

2.3.2.7 Parametry předávané hodnotou

```
1 int res = foo(3,4);  
2  
3 func int foo(int a, int b) {  
4     return a * b;  
5 }
```

2.3.2.8 Návrátová hodnota podprogramu

```
1 int num = 8;
2 int res = foo(num);
3
4 func int foo(int a) {
5     return a - 1;
6 }
```

2.3.3 Vlastní

2.3.3.1 Mocnina

Exponent mocniny musí být nezáporné celé číslo - interval $\langle 0; n \rangle$.

```
1 int a = 8 ^ 3;
```

2.3.3.2 Komentáře

```
1 // Tohle je jednoradkový komentář
2
3 /*
4 Tohle je vnorený komentář
5 */
```

Implementace

3

K vytvoření překladače jazyka Ligma je použit nástroj ANTLR. Zdrojové soubory se nachází v adresáři `/ligma/src/main/java/ligma`, soubor s předpisem gramatiky v `/ligma`. Syntaxe jazyka připomíná kombinaci JavaScriptu a ANSI C.

3.1 Gramatika

Gramatika jazyka definovaná v souboru `Ligma.g4` popisuje lexikální a syntaktická pravidla pro překladač. Obsahuje pravidla pro klíčová slova, datové typy, literály, operátory a struktury, jako jsou cykly, podmínky, funkce a přiřazení hodnot. Lexikální část gramatiky (`lexer rules`) rozpoznává jednotlivé tokeny, zatímco syntaktická část (`parser rules`) určuje hierarchii a strukturu příkazů. Gramatika je navržena tak, aby umožnila parsování komplexních výrazů včetně operátorů s různou prioritou. Tento soubor tvoří základ pro generování parsovacího stromu, který je dále využíván v překladači.

3.2 Struktura projektu

- `enums` - výčtové typy (PL/0 instrukce, operátory, scopy, datové typy)
- `exception` - definované třídy pro chyby během překladu
- `generator` - generátory PL/0 instrukcí
- `ir` - adresář s třídami reprezentující veškeré příkazy/výrazy
- `listener` - listenery pro hlášení chyb při lexikální/syntaktické analýze
- `table` - implementace tabulky symbolů
- `visitor` - visitory pro průchod derivačním stromem
- `App.java` - vstupní bod programu

3.3 TODO - podrobnosti k implementaci + grafíky

Projekt obsahuje sadu automatických testů, které testují lexikální, syntaktickou či sémantickou analýzu. Všechny testy (celkem TODO scénářů) jsou automaticky spouštěné při vytváření výsledného `.jar` souboru pomocí skriptu `run.sh` (případně pro platformu Windows `run.bat`). Testování bylo prováděno na operačních systémech Windows a MacOS (a to zejména pro zajištění multiplatformnosti - např. ukončovací znaky na konci řádek).

4.1 Lexikální analýza

Testy pro lexikální analýzu spouští třída `ExpressionLexicalTest`. Obsahuje celkem TODO negativních testů, které validují správnou funkčnost lexikální analýzy. Testovací soubory se nachází v adresáři `/src/test/resources/lexical`.

4.2 Syntaktická analýza

Testy pro syntaktickou analýzu spouští třída `ExpressionSyntaxTest`. Obsahuje celkem TODO testů (pozitivních + negativních). Testovací soubory se nachází v adresáři `/src/test/resources/syntax`. Důraz u negativních testů byl kladen zejména na provádění neplatných operací (např. chybějící operandy/operátory) či používání neplatných instrukcí.

4.3 Sémantická analýza

Testy pro sémantickou analýzu spouští třída `ExpressionSemanticTest`. Obsahuje celkem TODO testů (pozitivních + negativních). Testovací soubory se nachází v adresáři `/src/test/resources/semantic`.

4.4 Generování PL/0 instrukcí

Ukázkové zdrojové kódy přeložené do instrukční sady PL/0 se nachází v adresáři `/src/main/resources` - složka `/programs` pro zdrojové kódy Ligy, složka `/output` pro vygenerované PL/0 instrukce.

Uživatelská dokumentace

5

5.0.1 Prerekvizity

Pro úspěšné přeložení je vyžadováno:

- Java verze 23 (kvůli podpoře markdown komentářů)
- Maven verze 3.9.9

5.0.2 Překlad a spouštění

Pro překlad přejděte do adresáře `ligma` a spusťte skript `run.sh` (případně `run.bat`).

Výpis 5.1: Překlad a vytvoření spustitelného souboru

```
1 C:\Users\pavlicejk\ligma> run.bat
2
3 [INFO] Scanning for projects...
4 [INFO]
5 [INFO] -----< ligma:ligma >-----
6 [INFO] Building ligma 1.0
7 [INFO]    from pom.xml
8 [INFO] -----[ jar ]-----
9
10 .....
11 .....
12
13 [INFO] -----
14 [INFO] BUILD SUCCESS
15 [INFO] -----
16 [INFO] Total time: 12.525 s
17 [INFO] Finished at: 2024-12-24T21:13:47+01:00
18 [INFO] -----
```

Skripty spustí příkaz `mvn clean install`, který stáhne veškeré potřebné knihovny, přeloží projekt (vytvoří adresář `target` s přeloženými soubory), následně spustí

testy a vytvoří finální soubor `ligma.jar` ve složce `target`. Následně se spustí ukázkový program a vygeneruje soubor s PL/0 instrukcemi.

Program následně můžete spouštět pomocí příkazu

Výpis 5.2: Spuštění programu

```
1 C:\Users\pavlicejk\ligma\target> java -jar ligma.jar <input-file>  
    <output-file>
```

, kde `<input-file>` je cesta k souboru se zdrojovým kódem jazyka Ligma a `<output-file>` je výsledný soubor s PL/0 instrukcemi.

V rámci semestrální práce byl vytvořen překladač jazyka Ligma (*Language Sigma*) do instrukcí PL/0. Během překládání zdrojových kódů do instrukcí PL/0 je průběh překladače podrobně logován do konzole. Pro testování vygenerovaných instrukcí ukázkových programů byl využit on-line interpret <https://home.zcu.cz/~lipka/fjp/pl0/>. Veškeré testovací scénáře jsou uloženy v složce `/src/main/resources`. Nevalidní konstrukce/zápisy jazyka Ligma jsou řádně testovány automatickým testem.

Seznam výpisů

5.1	Překlad a vytvoření spustitelného souboru	14
5.2	Spuštění programu	15