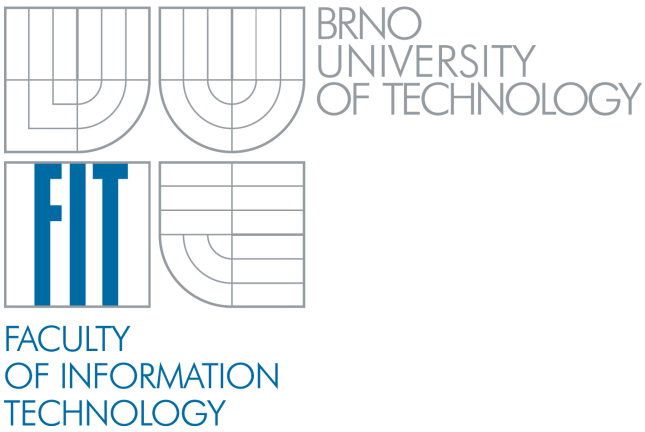
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Dokumentace projektu do předmětu IFJ a IAL

Interpret jazyka IFJ15

Tým 043, varianta b/2/I

Martin Honza (xhonza03)
Patrik Jurnečka (xjurne03)
Hana Slámová (xslamo00)
Frantisek Šumšal (xsumsa01)
Adam Švidroň (xsvidr00)

7. prosince 2015

Obsah

1 Úvod

Dokumentace popisuje návrh a implementaci projektu Interpret jazyka IFJ15 do předmětu IFJ (Formální jazyky a překladače) a IAL (Algoritmy). Vybrali jsme si zadání b/2/I, které nám udávalo, jáký algoritmus máme pro daný problém využít. Pro vyhledávání **Boyer-Mooreův algoritmus**, pro řazení algoritmus Heap sort, který byl využit ve vestavěné funkci **sort**. Poslední ze specifikových pravidel použití, jsme měli využít k implementaci tabulky symbolů **binární vyhledávací strom**.

Cílem projektu bylo vytvořit program, který interpretuje jazyk IFJ15, což je velmi zjednodušenou podmnožinou jazyka C++11.

2 Struktura projektu

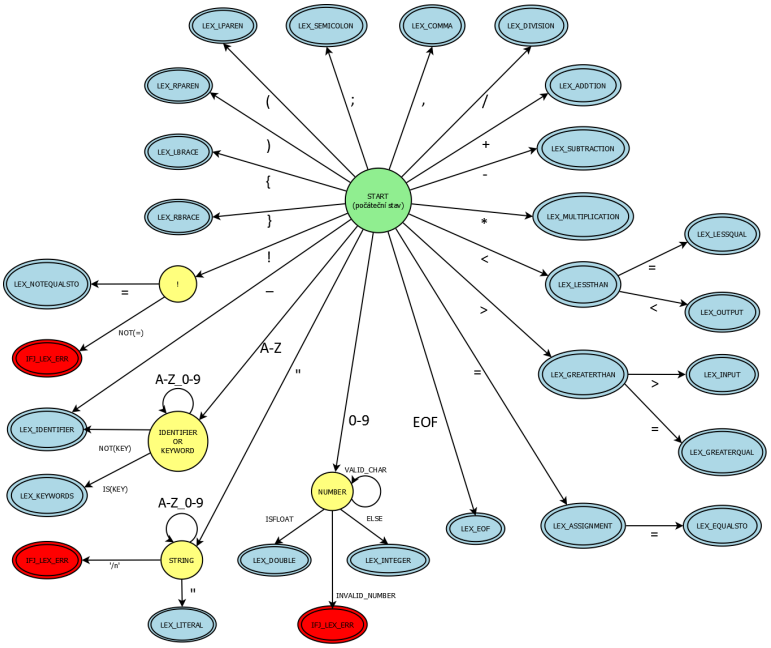
Překladač je rozdělen do tří hlavních celků. Pomocí **lexikálního analyzátoru** načítá zdrojový kód. **Syntaktický analyzátor** zažádá lexikální o token a ověří syntaxi a sémantiku. Po bezchybné kontrole se spustí **interpret**.

3 Lexikální analizátor (scanner)

Analyzátor je implementován v souborech *lex.c* a *lex.h*. Lexikální analyzátor načítá zdrojový kód po znacích a převádí jej na tokeny. Veškerá komunikace s dalšími knihovnami probíhá pomocí funkce *lex_get_token*. Analyzátor je Implementován pomocí konečného automatu (Obrázek 1), přičemž veškeré komentáře ignoruje. Funkce je volána vždy, když syntaktický analyzátor zažádá o token.

Token definuje jednotlivé prvky kódu a určuje jak se k němu má při překladu přistupovat. Analyzátor prochází tabulku klíčových slov (*lex_kw_t keywords[]*), zda-li nejde o rezervované klíčové slovo pro speciální funkci.

3.1 Konečný automat



Obrázek 1: Konečný automat

4 Syntaktická analizátor (parser)

Syntaktický analyzátor postupně volá tokeny z lexikálního analyzátoru. Tokeny jsou zpracovávaný dvěma způsoby. Pomocí LL gramatiky (Tabulka 1) a precedenční syntaktické analýzy (Tabulka 2). Při úspěšném dokončení syntaktické analýzy se generuje instrukční páska pro interpret.

4.1 LL gramatika

LL gramatika postupně ověřuje příchozí tokeny.

1.	$< program >$	$< declrList >$ EOF
2.	$< declrList >$	$< funcDeclr >< declrList >$
3.	$< declrList >$	$< empty >$
4.	$< funcDeclr >$	$< typeSpec >$ ID ($< params >$)
5.	$< typeSpec >$	INT
6.	$< typeSpec >$	DOUBLE
7.	$< typeSpec >$	STRING
8.	$< params >$	$< paramItem >$
9.	$< params >$	$< paramItem >, < params >$
10.	$< paramItem >$	$< typeSpec >$ ID
11.	$< paramItem >$	$< empty >$

Tabulka 1: LL gramatika

4.2 Precedenční syntaktická analýza

Syntaktická analýza zdola nahoru. Dle Zadání jsem analýzu implementovali za pomocí precedenční tabulky, která nám pomáhá ošetřit vstupní výrazy.

	+	−	*	/	<	>	<=	>=	==	!=	(	)	id	$
+	>	>	<	<	>	>	>	>	>	>	<	>	<	>
−	>	>	<	<	>	>	>	>	>	>	<	>	<	>
*	>	>	>	>	>	>	>	>	>	>	<	>	<	>
/	>	>	>	>	>	>	>	>	>	>	<	>	<	>
<	<	<	<	<	<	>	>	>	>	>	<	>	<	>
>	<	<	<	<	<	>	>	>	>	>	<	>	<	>
<=	<	<	<	<	<	>	>	>	>	>	<	>	<	>
>=	<	<	<	<	<	>	>	>	>	>	<	>	<	>
==	<	<	<	<	<	>	>	>	>	>	<	>	<	>
!=	<	<	<	<	>	>	>	>	>	>	<	>	<	>
(	<	<	<	<	<	<	<	<	<	<	<	=	<	
)	>	>	>	>	>	>	>	>	>	>		>		>
id	>	>	>	>	>	>	>	>	>	>		>		>
$	<	<	<	<	<	<	<	<	<	<	<		<	

Tabulka 2: Precenenční tabulka syntaktické analýzy výrazů

5 Sémantický analizátor

6 Interpret

Poslední částí Interpretu je vlastní překlad, který nastane pouze pokud zdrojový kód úspěšně projde syntaktickou a sémantickou analýzou. Interpretační část ma za úkol zpracovat instrukční sadu založenou na 3AC (tří adresné instrukce).

7 Chybové stavy

Pro chybové stavy jsme implementovali vlastní funkci, která je vždy volána při nalezení chyby v kód, jak syntaktické, sémantické nebo při interpretaci. Návratové chybové kódy byly definovány v zadání projektu.

8 Algoritmy z předmětu IAL

8.1 Heapsort

Heapsort neboli „řazení hromadou“ je řadící metoda s lineární složitostí. Algoritmus patří mezi nestabilní algoritmy, protože přesouvá prvky s příliš velkými skoky a také se nechová přirozeně.

8.2 Boyer-Mooreův algoritmus

Dle zadání jsem měli vestavěnou funkci **find**, která by používala pro vyhledání Boyer-Mooreův algoritmus. Boyer-Mooreův algoritmus je jedním velmi efektivním algoritmem pro vyhledávání podřetězců v řetězcích. Algoritmus prochází textem a porovnává vzorek zprava doleva a pro posuv bere ten z výsledků dvou heuristik, který je výhodnější. Jeho efektivita spočívá v časové složitosti. Čím delší je vyhledávaný podřetězec, tím je kratší doba. Funkce je implementována podle skript z předmětu IAL.

- ComputerJumps()

  Stanovení hodnot pole *CharJump*, které určují posuv vzorku. Použití pole CharJump pro postup vzorku činí tento algoritmus mnohem rychlejší než algoritmus Knuth-Morris-Prattův.

  Jednotlivé symboly řetězce mají uloženou svojí ASCII hodnotu.

- ComputerMatchJumps()

  Funkce pro samotný výpočet pole *MatchJump*.

- Boyer_Moor_Alg()

  Stanovení indexu polohy vzorku nalezeného v textu.

8.3 Binární vyhledávací strom (BVS)

Nejpoužívanější implementace pro dynamické vyhledávací tabulky. Algoritmus je implementován rekurzivně, dle opory k předmětu IAL (algoritmy). Vyhledávání v BVS je velmi podobné binárnímu vyhledávání v seřazeném poli.

Vytvořili jsme vlastní knihovnu funkcí. Implementace knihovny je v souborech *bst.c* a *bst.h*.

Rozhraní bst poskytuje funkčnost specifickou pro tabulku symbolů, která je implementována v souborech *stable.c* a *stable.h*.

9 Práce v týmu

Komunikace v týmu probíhala za pomocí IRC Freenode. Pravidelná osobní setkání se uskutečňovalo jednou za čtrnáct dní nebo po domluvě v CVT. Na osobním setkání bylo objasněno, co vše je zapotřebí udělat a také provedeno rovnoměrné rozdělení práce.

Ke sdílení zdrojových kódů jsme využili GitHub s vlastním soukromým kanálem.

9.1 Rozdělení práce

- **Martin Honza:** Precedenční analýza výrazu
- **Patrik Jurnečka:** Dokumentace, prezentace
- **Hana Slámová:** Vestavěné funkce
- **Frantisek Šumšal:** Lexikální analýzátor, syntaktický analyzátor
- **Adam Švidroň:** Heap sort, interpret

10 Závěr

Na projektu se pracovalo téměř dva měsíce, i když jsme začali včas, dokončit projekt jsme nestíhali do pokusného odevzdání. V posledním týdnu před odevzdáním jsme dodělávali interpret a snažili se vše zkompletovat, aby fungoval projekt dle zadání. Dokumentace se tvořila průběžně při dokončení některého logického celku.

Projekt byl velmi náročný, nejen ze strany programové, ale také hlavní roli hrála domluva týmu. Nakonec se to zvládlo dokončit. Z projektu jsme si odnesli velké zkušenosti, jak efektivně organizovat a pracovat v týmu.

Navržená implementace byla nakonec otestována v prostředí operačních systémů Linux a MS Windows 10.

Dokumentace i prezentace jsme psali v jazyce LaTeX.

10.1 Metriky kódu

- Počet souborů: ...
- Počet řádků zdrojového kódu: ...
- Velikost spustitelného souboru: ... (OS Linux 32 bitová architektůra)

Reference

[1] AHO, A. V. *Compilers: principles, techniques,*. 2nd ed. Boston: Pearson/Addison Wesley, c2007. ISBN 03-214-8681-1.