



IFJ Projekt 2017

Dokumentace

IFJ17 (Podmnožina Basic) – tým 37, varianta I.

Vedoucí týmu: Kristyna Jandová - **xjando04**
Členové: týmu Vilém Faigel - **xfaige00**
Nikola Timkova - **xtimko01**

Obsah

IFJ Projekt 2017.....	1
IFJ17 (Podmnožina Basic) – tým 37, varianta I.....	1
1 Úvod.....	2
1.1 Varianta.....	3
2 Teorie řešení.....	3
2.1 Lexikální analýza.....	3
2.2 Syntaktická analýza.....	3
2.3 Precedenční syntaktická analýza.....	3
3. Algoritmy.....	4
3.1 Binární vyhledávací strom.....	4
4 Práce v týmu.....	4
4.1 Nástrahy a útrapy týmového projektu.....	4
4.2 Rozdělení práce.....	4
5 Závěr.....	5
6 Metrika.....	5
7 Literatura reference?.....	5
Příloha 1.....	6
Příloha 2.....	7
Příloha 3.....	8

1 Úvod

Zadáním bylo vytvořit program v jazyce C, který načte zdrojový kód zapsaný ve zdrojovém jazyce IFJ17 a přeloží jej do cílového jazyka IFJcode17 (mezikód). Jazyk IFJ17 je zjednodušenou podmnožinou jazyka FreeBasic, jenž staví na legendárním jazyce BASIC.

Jedná se o staticky typovaný, imperativní jazyk.

1.1 Varianta

Zvolili jsme variantu I, což znamená, že tabulka symbolů je implementována pomocí binárního vyhledávacího stromu.

2 Teorie řešení

Řešení jednotlivých modulů překladače.

2.1 Lexikální analýza

Lexikální analyzátor – neboli Scanner je první fází překládacího procesu. Stará se o rozdělení vstupního kódu na jednotlivé části – takzvané tokeny.

Proces tokenizace probíhá za pomoci metody `_scanner_next(string *word)`, ve které je naprogramován konečný automat lexikální analýzy (Viz obrázek 1.). Tato metoda je volána v metodě `scanner_next_token()`, která se stará o výsledné složení vráceného tokenu.

Token obsahuje **flag** - o jaký token se jedná (reprezentováno číslem definovaným v tokens), **ID** - hodnotu slova, **line a position** – definuje pozici tokenu

Tokenem **není**:

- Bílý znak
 - mezera
 - tabulátor
 - vícenásobné odřádkování
- Komentář
 - jednořádkový
 - víceřádkový

Narazí-li scanner na chybný token vrací `LEXICAL_ERROR` – tedy číslo 1 a ukončí program

2.2 Syntaktická analýza

Syntaktický analyzátor – neboli Parser je stěžejním modulem celého překladače. Náš parser je založen na metodě automatu řízeného stackem. (Viz příloha 2) Postupně volá ze Scanneru tokeny za pomoci metody `scanner_next_token()`. Kontroluje jejich souslednost pomocí vhodně navržené gramatiky. Tuto gramatiku naleznete popsanou v příloze 3.

Prací parseru je i kontrola sémantiky, a to v úrovni jestli nedochází ke střetu datových typů. Další sémantická kontrola se provádí až na úrovni interpretu.

Stane-li se že parser narazí v kódu na výraz, volá precedenční analýzu, která tento výraz vyhodnotí.

Další funkcí parseru je ta, že při kontrole syntaxe ukládá nalezené tokeny do tabulky symbolů.

V případě že parser narazí na chybu vrátí SYNTAX_ERROR – tedy číslo 2 a ukončí program

2.3 Precedenční syntaktická analýza

Součástí parseru je precedenční syntaktický analyzátor, který jsme pro přehlednost oddělili do samostatného souboru. Tento analyzátor se stará o správnou syntaxi výrazů.

Kontrola probíhá za pomoci precedenční syntaktické tabulky (Viz příloha 4)

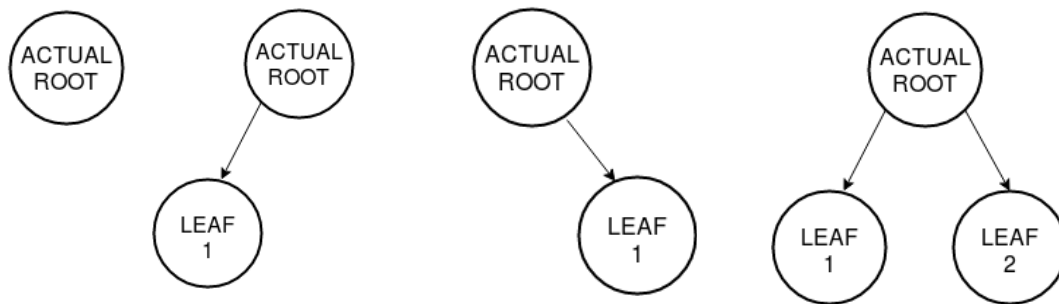
3. Algoritmy

popis použitých algoritmů při zpracování dat

3.1 Binární vyhledávací strom

Binární vyhledávací strom je typ stromu který má z uzlu právě 0 až 2 potomky.

Přípustné binární (pod)stromy:



Binární strom se může dále rozvíjet a to tak, že daný list může být i uzlem pro další listy.

Pro rovnoměrné rozložení binárního stromu je implementována funkce ***void tree_balance(struct tree *t);***

4 Práce v týmu

Popis spolupráce a rozdělení vývoje

4.1 Nástrahy a útrapy týmového projektu

V prvopočátku projektu byl náš tým velmi entuziastický – pravidelná komunikace, navržení milníků tak, aby bylo vše stihnuto s předstihem a zbyl čas na rozšíření, chuť všech členů do práce.

Bohužel s přibývajícím časem se množil neochota, chyběla pravidelná komunikace a čas mezi dotazem a odpovědí na stav práce se prodlužovaly

V poslední fázi vývoje jsme se museli uchýlit k drastické metodě vynucení spolupráce – k výhrůžce přidělení 0% za projekt, proto je procentuální rozdělení nevyvážené.

4.2 Rozdělení práce

4.2.1 Rozdělení vývoje

Kristyna Jandová	Scanner, dokumentace, příprava obhajoby
Vilém Faigel	Parser, instrukce, podpůrné systémy
Nikola Timkova	Precedenční analýza
Václav Doležal	IAL část, generování IFJcode17

4.2.1 Procentuální rozdělení

Kristyna Jandová
Vilém Faigel
Nikola Timkova
Václav Doležal

5 Závěr

Projekt se podařilo dokončit a otestovat na různých druzích vlastních testovacích příkladů, a to i na serveru Merlin.

Projekt pro nás byl velkým zdrojem zkušeností, které zajisté využijeme i v následujících projektech či v praxi.

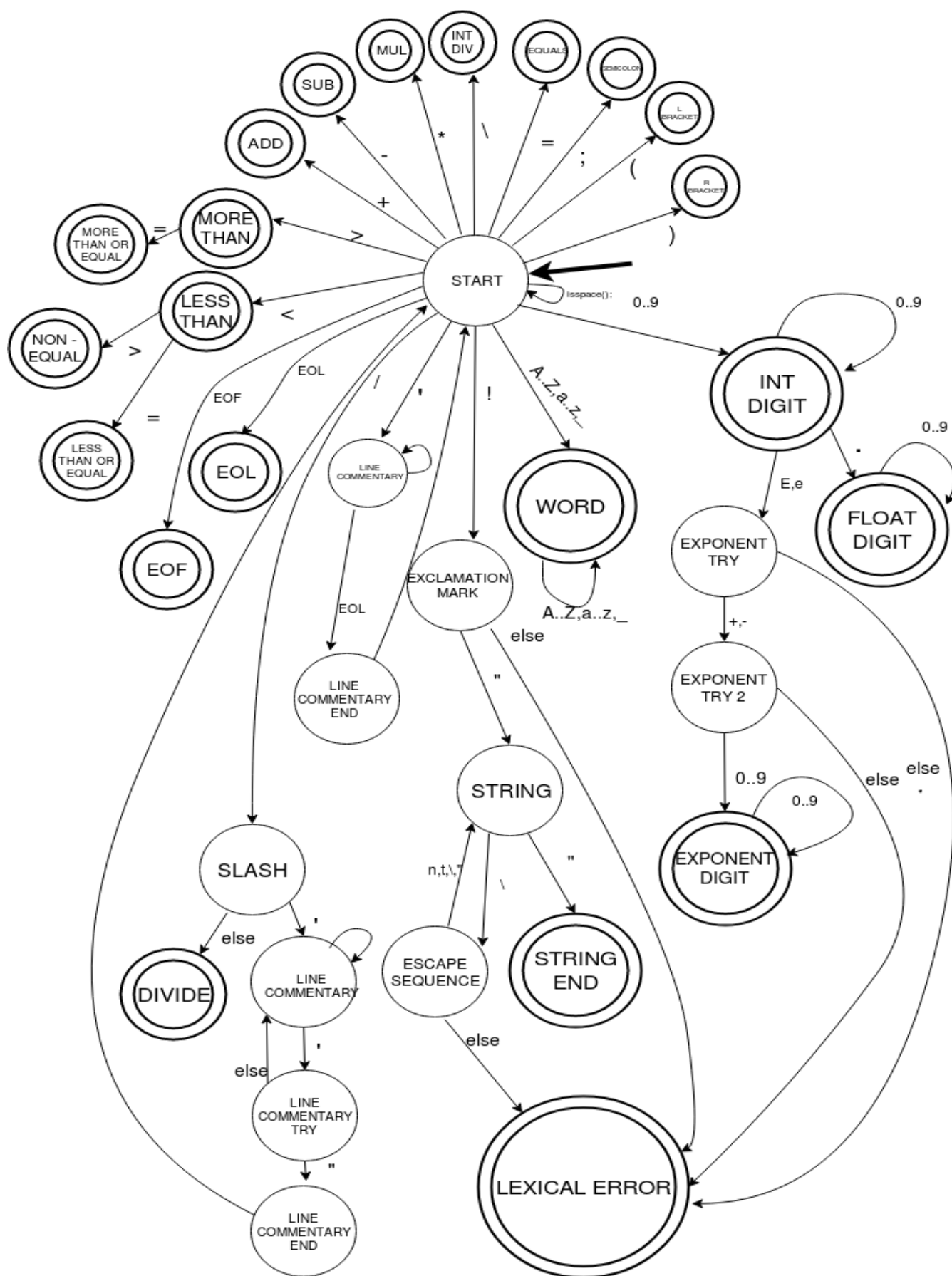
6 Metrika

Počet řádků kódu:

Velikost sprustitelného souboru:

7 Literatura

Příloha 1



Příloha 2

1. <PROGRAM> -> <FUNCTION_LIST>SCOPE
EOL<STATEMENT_LIST> END SCOPE <FUNCTION_LIST>
2. <FUNCTION_LIST> -> ϵ
3. <FUNCTION_LIST> -> <FUNCTION> <FUNCTION_LIST>
4. <FUNCTION> -> DECLARE FUNCTION (id) lb <PARAM> rb AS <TYPE> EOL
5. <FUNCTION> -> FUNCTION (id) lb <PARAM> AS <TYPE> EOL
<STATEMENT_LIST> END FUNCTION
6. <PARAM> -> ϵ
7. <PARAM> -> (id) AS <TYPE> <PARAM>
8. <PARAM> -> , (id) AS <TYPE> <PARAM>
9. <STATEMENT_LIST> -> ϵ
10. <STATEMENT_LIST> -> <STATEMENT> <STATEMENT_LIST>
11. <STATEMENT> -> DIM (id) AS <TYPE> EOL
12. <STATEMENT> -> DIM (id) AS <TYPE> <EXPRESSION> EOL
13. <STATEMENT> -> RETURN <EXPRESSION> EOL
14. <STATEMENT> -> (id) = <EXPRESSION> EOL
15. <STATEMENT> -> INPUT (id) EOL
16. <STATEMENT> -> PRINT <TERM> EOL
17. <STATEMENT> -> (id) = (func_id) lb <PARAM> rb EOL
18. <STATEMENT> -> IF <EXPRESSION> THEN EOL <STATEMENT_LIST>
<ELSEIF> <ELSE> END IF
19. <STATEMENT> -> DO WHILE <EXPRESSION> EOL <STATEMENT_LIST>
LOOP

20. <ELSEIF>	->	ϵ
21. <ELSEIF> <ELSEIF>	->	ELSE IF <EXPRESSION> THEN EOL <STATEMENT_LIST>
22. <ELSE>	->	ϵ
23. <ELSE>	->	ELSE EOL <STATEMENT LIST> END IF
24. <TERM>	->	;
25. <TERM>	->	(id) <TERM>
26. <TERM>	->	;(id) <TERM>
27. <TERM>	->	<STRING> <TERM>
28. <TERM>	->	; <STRING> <TERM>
29. <TYPE>	->	INTEGER
30. <TYPE>	->	DOUBLE
31. <TYPE>	->	STRING

Příloha 4

	+	-	*	/		>	<	>=	<=	=	<>	()		\$
+	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
/	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
>	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
>=	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<=	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
=	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<>	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
i	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
\$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Legenda:

‘>’ - Redukce

‘<’ - Handle

‘ ‘ - Nedefinováno

‘=’ - Vyjímka

‘E’ - Exit

Pravidla:

E -> E

E -> (E)

E-> E x E