



Dokumentácia k projektu
z IFJ/IAL
**IMPLEMENTÁCIA PREKLADAČA
JAZYKA IFJ20**
Tím 123, varianta I

Členovia tímu:

Martin Novotný Mlinárčik (xnovot)

Šimon Feňko (xfenko01)

Andrej Hýroš (xhyros)

Peter Čellár (xcellar)

8. decembra 2020

Obsah

1	Úvod	2
2	Etapy projektu	2
3	Práca v tíme	2
3.1	Komunikácia v tíme	2
3.2	Správa kódu	2
3.3	Rozdelenie práce	2
4	Implementácia	2
4.1	Lexikálna analýza	2
4.2	Syntaktická analýza - rekurzívny zostup	3
4.3	Precedenčná syntaktická analýza	3
4.4	Sémantická analýza	3
4.5	Generovanie kódu	3
4.6	Testovanie	3
4.7	Tvorba Dokumentácie	3
5	Testovanie	3
6	Zhrnutie	3
7	Prílohy	4

1 Úvod

Cieľom projektu[1] bolo vytvorenie prekladača v jazyku C, ktorý prečíta zdrojový kód zapísaný v zdrojovom jazyku IFJ20, ktorý je zjednodušenou podmnožinou jazyka Go a preloží ho do cieľového jazyka IFJcode19 (medzikód). Vybrali sme si variantu I. a to implementovanie tabuľky symbolov pomocou binárneho vyhľadávacieho stromu.

2 Etapy projektu

1.Návrh štruktúry programu:

- a) konečný automat7
- b) precedenčná tabuľka7
- c) LL tabuľka7
- d) LL gramatika7

2.Lexikálna analýza4.1

3.Syntaktická analýza - rekurzívny zostup4.2

4.Precedenčná syntaktická analýza4.3

5.Sémantická analýza4.4

6.Generovanie kódu4.5

7.Testovanie4.6

8.Tvorba Dokumentácie4.7

3 Práca v tíme

3.1 Komunikácia v tíme

Ako hlavný komunikačný kanál sme zvolili discord (hovory), poprípade messenger. Kvôli zhoršenej epidemiologickej situácii, sme úplne obmedzili osobné stretnutia, čo sa odrazilo na práci, ktorá bola trochu náročnejšia.

3.2 Správa kódu

Ako verzovací systém sme použili Git, hostovaný na službe Github. Tento spôsob nám vyhovoval najviac, pretože každý člen tímu mohol pridávať a upravovať zdrojový kód.

3.3 Rozdelenie práce

Martin Novotný Mlinárčik:

Šimon Feňko:

Andrej Hýroš:

Peter Čellár:

4 Implementácia

4.1 Lexikálna analýza

Lexikálny analyzátor (LA), bola časť projektu, ktorú sme implementovali ako prvú. Je navrhnutý ako konečný automat, podľa ktorého bol lexikálny analyzátor implementovaný. Ten postupne načíta jednotlivé znaky. Zaviedli sme obojsmernú komunikáciu medzi syntaktickou analyzátorom (SA) a

(LA). SA bol potom schopný čítať tokeny a tiež vrátiť naspäť do LA. Tokeny boli čítané pri volaní LA zo štandardného vstupu. Pokiaľ sa analyzátor nedostane do požadovaného stavu, vracia lexikálnu chybu (hodnota 1).

4.2 Syntaktická analýza - rekurzívny zostup

4.3 Precedenčná syntaktická analýza

4.4 Sémantická analýza

4.5 Generovanie kódu

4.6 Testovanie

4.7 Tvorba Dokumentácie

5 Testovanie

Testovali sme pravidelne na kratších, základných testoch, pomocou ktorých sme sa uistovali o správnosti kódu a odstraňovaní chýb.

6 Zhrnutie

blablabla

7 Prílohy



Obr. 1: Konečný automat

	+	-	*	/	()	==	!=	>=	<=	<	>	i	\$
+	2	2	1	1	1	2	2	2	2	2	2	2	1	2
-	2	2	1	1	1	2	2	2	2	2	2	2	1	2
*	2	2	2	2	1	2	2	2	2	2	2	2	1	2
/	2	2	2	2	1	2	2	2	2	2	2	2	1	2
(1	1	1	1	1	3	1	1	1	1	1	1	1	4
)	2	2	2	2	4	2	2	2	2	2	2	2	4	2
==	1	1	1	1	1	2	2	2	2	2	2	2	1	2
!=	1	1	1	1	1	2	2	2	2	2	2	2	1	2
>=	1	1	1	1	1	2	2	2	2	2	2	2	1	2
<=	1	1	1	1	1	2	2	2	2	2	2	2	1	2
<	1	1	1	1	1	2	2	2	2	2	2	2	1	2
>	1	1	1	1	1	2	2	2	2	2	2	2	1	2
i	2	2	2	2	4	2	2	2	2	2	2	2	4	2
\$	1	1	1	1	1	4	1	1	1	1	1	1	1	4

Obr. 2: Precedenčná tabuľka

Obr. 3: LL Tabuľka

```

PROG -> package main eol EOL DEF_FUNC eof
DEF_FUNC -> func id (PARAMS) FUNC_RETLIST_BODY eol EOL DEF_FUNC
DEF_FUNC -> ε
PARAMS -> id TYPE PARAMS_N
PARAMS -> ε / )
PARAMS_N -> , id TYPE PARAMS_N
PARAMS_N -> ε / )
FUNC_RETLIST_BODY -> (FUNC_BODY
FUNC_RETLIST_BODY -> { eol EOL STAT OPTIONAL_RET }
FUNC_BODY -> ) { eol EOL STAT OPTIONAL_RET }
FUNC_BODY -> RETURN_TYPE { eol EOL STAT REQUIRED_RET eol EOL }
REQUIRED_RET -> return exp EXP_N
OPTIONAL_RET -> return eol EOL
OPTIONAL_RET -> eol EOL
EXP_N -> , exp EXP_N
EXP_N -> ε
STAT -> id ID_N/CALL_FUNC
STAT->ifexp{eolEOL STAT}else{eolEOL STATEolEOL}eolEOLSTAT STAT-
>forFOR_DEFexp;id=exp{eolEOL STATEolEOL }eol EOL STAT STAT -> ε
FOR_DEF -> id INIT_DEF exp ;
FOR_DEF -> ε / ;
INIT_DEF -> :=
INIT_DEF -> =
ID_N-> , id ID_N
ID_N -> ε
//CALL_FUNC-> id ( exp EXP_N ) – NAHRADENE DO CALL_FUNC/ASSIGN
TYPE -> int
TYPE -> float64
TYPE -> string
TYPE -> id
EOL -> eol EOL
EOL -> ε

RETURN_TYPE -> TYPE RETURN_TYPE_N RETURN_TYPE -> ε / )
RETURN_TYPE_N -> , TYPE RETURN_TYPE_N RETURN_TYPE_N -> ε / )

CALL_FUNC/ASSIGN -> exp EXP_N eol EOL STAT CALL_FUNC/ASSIGN ->
(FUNC_PARAMS) EOL STAT OPT_ID -> id
OPT_ID -> ε

FUNC_PARAMS -> exp FUNC_PARAMS_N FUNC_PARAMS -> ε
FUNC_PARAMS_N -> , exp FUNC_PARAMS_N FUNC_PARAMS_N -> ε

FOR_ASSIGN -> id = exp ; FOR_ASSIGN -> ε

ID_N/CALL_FUNC -> ID_N INIT_DEF OPT_ID CALL_FUNC/ASSIGN ID_N/
CALL_FUNC -> (FUNC_PARAMS) EOL STAT

```

Citácie

- [1] Zbyněk Křivka, Lukáš Zobal, Dominika Regéciová. *Formální jazyky a překladače*. [ONLINE]. 2020. URL: <https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=/course/IFJ-IT/projects/ifj2020.pdf>.