Dokumentácia k projektu IFJ/IAL

Implementácia interprétu jazyka IFJ19

Tým 010, varianta II:

Dominik Boboš (xbobos00) – vedúci týmu

Peter Hudeček (xhudec34)

Timotej Kováčik (xkovac49)

Dávid Oravec (xorave05) 9. Prosince 2019

Obsah

1 Úvod

2 Rozdelenie

3 Časti

3.1 Lexikálna analýza

3.2 Syntaktická analýza

3.3 Sémantická analýza

3.4 Generátor kódu

4 Pomocné súbory

4.1 Zásobník pre syntaktickú analýzu

5 Práca v tíme

5.1 Problémy pri práci v tíme

6 Záver

7 Prílohy

# Úvod

Cielom projektu bolo vytvoriť program v jazyku C, ktorý načíta zdrojový kód zapísaný v jazyku IFJ19 a preloží ho do cielového jazyka IFJcode19. Pokial preklad prebehne bez chýb, program vracia návratovú hodnotu 0. Ak dojde k chybe, program vracia kód chyby.

# Rozdelenie

## Dominik Boboš (xbobos00)

Tvorba gramatických pravidiel a LL tabulky

Tvorba syntaktického analyzátoru

Tvorba generátoru kódu (spoločne)

## Peter Hudeček (xhudec34)

Tvorba precedenčnej tabulky

Tvorba sémantického analyzátoru (spoločne)

Tvorba generátoru kódu

## 2.3 Timotej Kováčik (xkovac49)

Tvorba gramatických pravidiel a LL tabulky

Tvorba sémantického analyzátoru (spoločne)

Tvorba generátoru kódu (spoločne)

## Dávid Oravec (xorave05)

Tvorba konečného automatu k lexikálnej analýze

Tvorba lexikálneho analyzátoru

Tvorba sémantického analyzátoru (spoločne)

Tvorba generátoru kódu (spoločne)

## Spoločná práca

Testovanie kódu

## 2.6 Zdôvodnenie odchýľky od rovnomerného počtu bodov

Odchýlka od rovnomenrného počtu bodov bola spôsobená nerovnomerným pôvodnným rozdelením práce a spoločnou prácou na istých častiach interpretu.

# Časti

## 3.1 Lexikálna analýza

Úlohou lexikálneho analyzýtoru (scanner.c) je načítať zo zdrojového súboru znaky, definuje dátovu štruktúru **Token**. Tokeny uchovávaju informácie o načítaných znakoch. Hlavnou funkiou je funkcia *getNextToken(),* ktorá po volaní vracia token z načítaného znaku. V prípade konca vstupného súboru funkcia vráti *TYPE\_EOF*. V prílohe 1 sa nachádza model konečného automatu podla ktorého bol lexikálny analyzátor implementovaný.

## 3.2 Syntaktická analýza

Úlohou syntaktického analyzátoru (parser.c) je skontrolovať či postupnosť tokenov prijatých z lexikálneho analyzýtoru predstavuje syntakticky správny program, na základe rekurzivného zostupu. Syntaktický analyzátor pomocou funkcie *getNextToken()* žiada od lexikálneho analyzátoru tokeny, ktorých správnosť je následne analyzovaná na základe LL-tabulky (príloha 2). V prípade že syntax vstupného súboru je správna, parser.c vracia návratovú hodnotu 0, v opačnom prípade vracia kód chyby. Spracovanie výrazou pomocou súbore expr.c.

## 3.3 Sémantická analýza

Sémantická analýza (expr.c) prebieha na základe kontroly operácií dátových typov a ich správnosti zápsiu. Analyzátok kontroluje možné kombinácie zápisu dátových typov a ich návratových hodnôt vo výrazoch a rieši ich prípadné pretypovanie.

## Generátor kódu

Úlohou generátoru je generovať medzikód IFJcode19 po dokončení kontroly správnosti vstupného kódu. Výrazy sú postupne ukladané na zásobníku a následne generované do výsledného kódu. Generátor kódu je implementovaný v súbore generator.c.

# 4. Pomocné súbory

## 4.1 Zásobník pre syntaktickú analýzu

Zásobník obsahuje základné funkcie pre prácu so zásobníkom. Pridali sme funkciu *sym\_insert\_stop\_NT()* ktorá vkladá znak STOP na prvé miesto za TERMINAL. Pridaná bola tak isto funkcia *symbol\_top\_term()* ktorej úholou je vrátiť najvrchnejší terminál zo zásobníku.

## 4.2 Hashovacia tabulka

# 4. Práca v tíme

V tíme bola práca na jednotlivých častiach projektu rozdelená dopredu počas stretnutí pred zadaním projektu. Počas práce na projekte sme v prípade problému zahájili stretnutie na ktorom sme problém detailne prediskutovali a poradili sa o možných riešeniach. Na vzdialenú komunikáciu sme používali aplikáciu Discord. Na zdielanie kódu sme používali repozitár GIT.

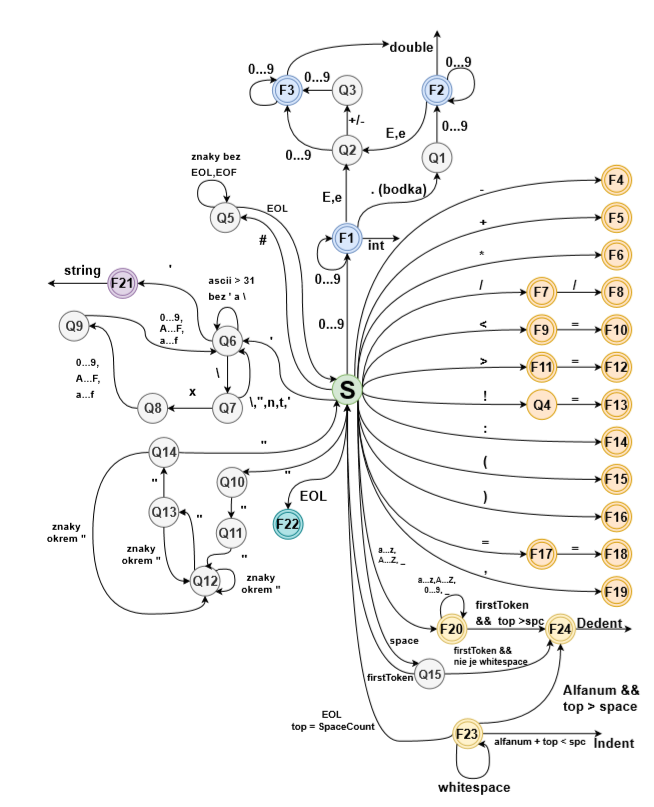
## 4.1 Problémy pri práci v tíme

V tíme sme sa stretli aj s malým množstvom problémov. Miestami zlyhávala komunikácia medzi jednotlivými členmi týmu, ktorú sme sa ale po osobnom stretnutí a diskusí zlepšili.

# 6 Záver

# Prílohy

## Model konečného automatu



## LL-gramatika

1. <prog> -> KEYWORD\_DEF TYPE\_IDENTIFIER(<params>)TYPE\_COLON TYPE\_EOL TYPE\_INDENT <statement> TYPE\_DEDENT <prog>

2. <prog> -> TYPE\_EOL <prog>

3. <prog> -> <statement> TYPE\_EOL <prog>

4. <prog> -> TYPE\_EOF <end>

5. <statement> -> KEYWORD\_IF <expression> TYPE\_COLON TYPE\_EOL TYPE\_INDENT <statement> TYPE\_DEDENT KEYWORD\_ELSE TYPE\_COLON TYPE\_EOL TYPE\_INDENT <statement> TYPE\_DEDENT <statement\_next>

6. <statement> -> KEYWORD\_WHILE <expression> TYPE\_COLON TYPE\_EOL TYPE\_INDENT <statement> TYPE\_EOL TYPE\_DEDENT <statement\_next>

7. <statement> -> KEYWORD\_RETURN <expression>

8. <statement> -> TYPE\_IDENTIFIER TYPE\_ASSIGN\_VALUE <expression> <statement\_next>

9. <statement> -> TYPE\_IDENTIFIER(<params>) <statement\_next>

10. <statement\_next> -> TYPE\_EOL <statement>

11. <statement> -> KEYWORD\_PRINT(<expression>) <statement\_next>

12. <statement> -> KEYWORD\_PASS <statement\_next>

13. <statement> -> KEYWORD\_INPUTS() <statement\_next>

14. <statement> -> KEYWORD\_INPUTI() <statement\_next>

15. <statement> -> KEYWORD\_INPUTF() <statement\_next>

16. <statement> -> KEYWORD\_LEN(TYPE\_STRING) <statement\_next>

17. <statement> -> KEYWORD\_SUBSTR(TYPE\_STRING TYPE\_COMMA TYPE\_INT TYPE\_COMMA TYPE\_INT) <statement\_next>

18. <statement> -> KEYWORD\_CHR(TYPE\_STRING TYPE\_COMMA TYPE\_INT) <statement\_next>

19. <statement> -> KEYWORD\_ORD(TYPE\_INT) <statement\_next>

20. <params> -> TYPE\_IDENTIFIER <param\_next>

21. <params> -> ε

22. <param\_next> -> TYPE\_COMMA TYPE\_IDENTIFIER <param\_next>

23. <param\_next> -> ε

24. <end> -> ε

25. <statement\_next> -> ε