

IFJ2022

Projektová dokumentácia do predmetov IFJ a IAL

Tým xjokay00, varianta BVS

**Dávid Jókay, xjokay00: 25%**

Peter Kováč, xkovac66: 25%

Marek Buch, xbuchm02: 25%

Matúš Gazdík, xgazdi04: 25%

# Obsah

# Úvod

V projekte IFJ2022 do predmetov IFJ a IAL bolo úlohou implementovať prekladač jazyka IFJ22 do IFJcode22. Program načíta zdrojový kód z STDIN, vykoná kontrolu správnosti programu a program preloží do cieľového jazyka.

# 1 Práca v tíme

* 1. **Rozdelenie**

Rozdelenie práce na projekte bolo rovnomerné a každý člen tímu sa podieľal rovnakým dielom, a preto má každý percentuálne hodnotenie 25%. Konkrétnejšie rozdelenie zachytáva tabuľka 1.

|  |  |
| --- | --- |
| Meno | Priradená práca |
| Marek Buch | Lexikálna analýza, Syntaktická analýza, Tabuľka symbolov, |
| Sémantická analýza, LL tabuľka, Dokumentácia, Generovanie kódu |
| Matúš Gazdík | Lexikálna analýza, Syntaktická analýza, Syntaktická analýza výrazov, |
| LL tabuľka, Generovanie kódu |
| Dávid Jókay | Organizácia, Lexikálna analýza, Tabuľka symbolov, Sémantická analýza, |
| Dokumentácia, Generovanie kódu |
| Peter Kováč | Lexikálna analýza, Syntaktická analýza, Syntaktická analýza výrazov, |
| Testovanie, Refaktorizácia, Generovanie kódu |

Tabuľka 1

* 1. **Spôsob práce**

Hoci sme na projekte začali pracovať relatívne skoro, t.j. začiatkom októbra, zvolili sme nedostatočné tempo a tým sme sa dostali do časového sklzu. Zo začiatku sme sa stretávali na týždennej báze a pracovali sme na projekte spoločne. Neskôr sme zintenzívnili náš prístup a pracovali na projekte denne, najčastejšie vo dvojiciach. Prácu sme si delili postupne, podľa aktuálnej situácie.

* 1. **Komunikácia**

Komunikácia medzi členmi tímu prebiehala zväčša osobne, prípadne pomocou aplikácie Discord. Komunikácia prebiehala v poriadku, každý člen tímu aktívne komunikoval a nemali sme problémy pri plánovaní spoločných stretnutí.

* 1. **Verzovací systém**

Pre správu súborov a zdrojového kódu sme používali systém GIT. Ako vzdialený repozitár sme používali GitHub. Využívali sme aj funkciu LiveShare v editore Visual Studio Code. Funkcia LiveShare nám umožnila paralelne pracovať na rovnakej časti projektu v prípade, že osobné stretnutie nebolo možné.

# 2. Lexikálna analýza

Pri implementácii prekladaču sme začali tvorbou lexikálnej analýzy. Využili sme pri tom znalosti konečných automatov.

# 2.1 Automat

Automat, použitý pri tvorbe lexikálnej analýzy je implementovaný vo funkcii dka(), v súbore tokenizer.c, v ktorom sa celá lexikálna analýza odohráva. Implementovaný automat je zobrazený v tabuľke 2.

Funkcia dka()je deterministický konečný automat implementovaný pomocou opakujúceho sa switch statementu, kde každý CASE značí jeden stav automatu. Prechody medzi jednotlivými stavmi automatu sú dané prečítaným znakom z stdin. Ak načítaný znak nesúhlasí zo žiadnym znakom, ktorý aktuálny stav povoľuje, je vrátená chyba. Ak je načítaná sekvencia znakov, ktorú jazyk IFJ2022 povoľuje, je vytvorený a uložený nový token.

# 2.2 Token

Abstraktný dátový typ token\_t obsahuje atributy token\_type a value. token\_type môže nadobudnúť hodnoty {TOK\_ID, TOK\_ID\_FUNCTION, TOK\_KEYWORD, TOK\_SEPARATOR, TOK\_OPERATOR, TOK\_LIT, TOK\_EPILOG, TOK\_SPECIAL} a value je skutočný načítaný reťazec. Token je uložený do abstraktného dátového typu token\_storage\_t. token\_storage\_t má ako atributy dynamicky alokované pole tokens (kde sú tokeny uložené) , num\_tokens a array\_len.

Komentáre a prázdne znaky sú tokenizerom ignorované a nie sú z nich vygenerované tokeny.

Súčasťou lexikálnej analýzy sú aj neskôr využívané funckie get\_token() a get\_token\_keep(), kde get\_token() vracia aktuálny token a posúva ukazateľ na ďalší a get\_token\_keep() vracia aktuálny token.

# 3. Syntaktická analýza

Syntaktická analýza je najkomplikovanejšou časťou projektu a je umiestnená v súbore parser.c a expression.c . Syntaktická analýza je riadená LL-gramatikou, bližšie špecifikovanou v tabuľke 3. Použili sme metódu rekurzívneho zostupu, pričom výrazy sú spracovávané separátne v expression.c

Každé pravidlo LL gramatiky má svoju vlastnú funkciu, ktoré su postupne volané v parser.c. parser.c dostáva tokeny pomocou funkcií get\_token() a get\_token\_keep() , kde ich postupne volajú jednotlivé funkcie pravidiel.

# 4. Sémantická analýza

# Sémantická analýza prebieha v parser.c za pomoci symtable.c a expression.c, kde je uložená tabuľka symbolov.

# 4.1 Tabuľka symbolov

# Tabuľku symbolov sme implementovali ako binárny vyhľadávací strom, kde každý uzol značí unikátnu premennú alebo funkciu. Uzly sú v strome ukladané na základe ich abecednej hodnoty.

V prípade, že sa ich abecedná hodnota rovná, je nový uzol uložený ako pravý potomok posledného uzlu s rovnakým názvom.

V prípade ukladania funkcie, má uzol nastavené atributy name, return\_type, arguments a num\_arguments (a samozrejme left a right).

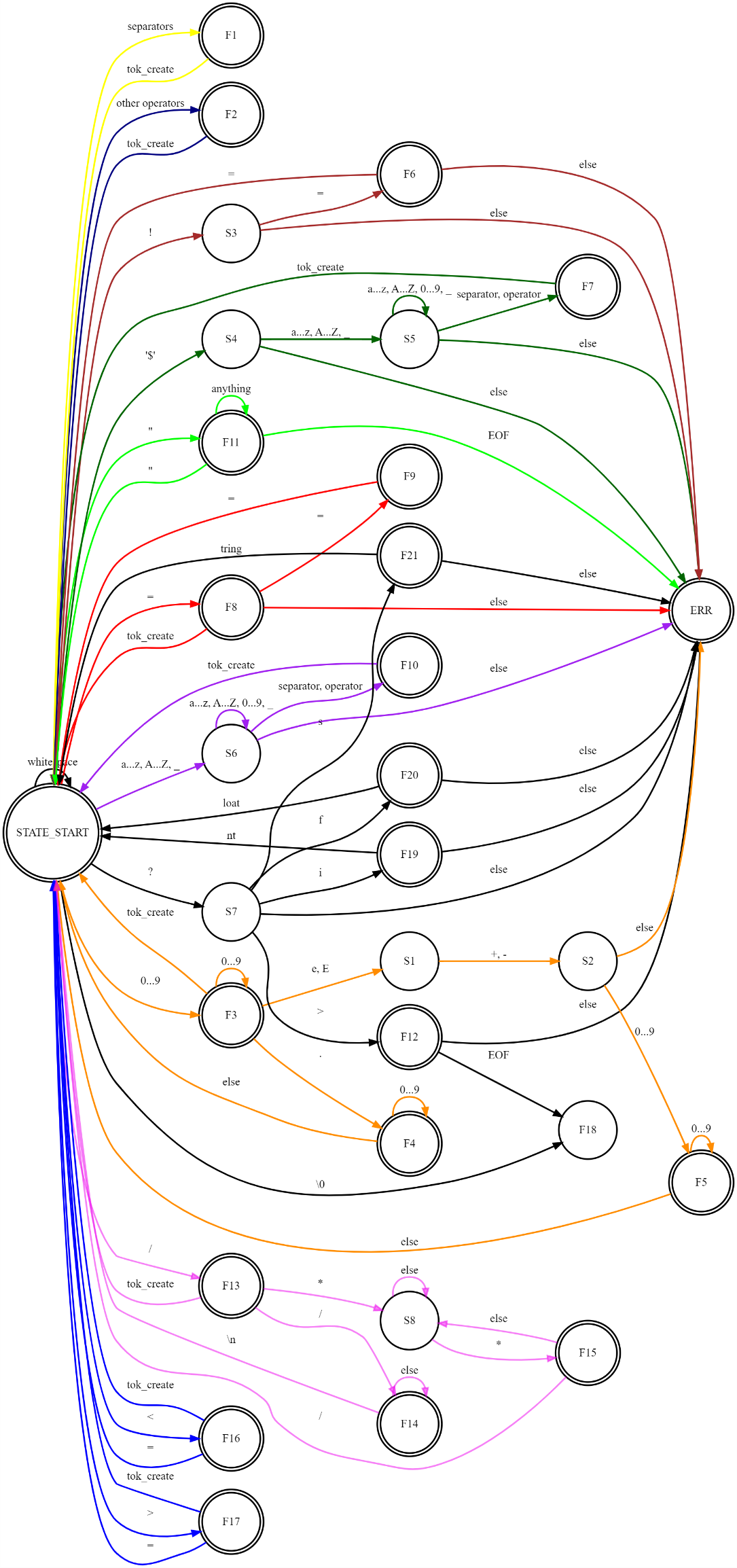
Pri ukladaní id premennej jej sú nastavené name, scope a datatype (left a right).

# 4.2 Spracovanie výrazov

Spracovanie výrazov prebieha pomocou precedenčnej analýzy v súbore expression.c

# 5. Makefile

# 7. Záver

****

|  |  |
| --- | --- |
| F1 | STATE\_SEP |
| F2 | STATE\_OP |
| F3 | STATE\_LIT\_NUM |
| F4 | STATE\_LIT\_NUM\_FLOAT |
| F5 | STATE\_LIT\_NUM\_FLOAT\_E\_3 |
| F6 | STATE\_NOT\_2 |
| F7 | STATE\_ID |
| F8 | STATE\_EQUAL\_1 |
| F9 | STATE\_EQUAL\_2 |
| F10 | STATE\_KEYWORD |
| F11 | STATE\_LIT\_STR |
| F12 | STATE\_EPLIOG\_2 |
| F13 | STATE\_COMMENT\_START |
| F14 | STATE\_COMMENT\_SINGLE |
| F15 | STATE\_COMMENT\_MULTI2 |
| F16 | STATE\_SMALLER\_EQUAL |
| F17 | STATE\_GREATER\_EQUAL |
| F18 | STATE\_FINISHED |
| F19 | STATE\_QUESTION\_INT |
| F20 | STATE\_QUESTION\_FLOAT |
| F21 | STATE\_QUESTION\_STRING |
| S1 | STATE\_LIT\_NUM\_FLOAT\_E\_1 |
| S2 | STATE\_LIT\_NUM\_FLOAT\_E\_2 |
| S3 | STATE\_NOT\_1 |
| S4 | STATE\_ID\_START |
| S5 | STATE\_ID\_MAIN |
| S6 | STATE\_KEYWORD\_MAIN |
| S7 | STATE\_EPILOG |
| S8 | STATE\_COMMENT\_MULTI |

Tabuľka 2 - Automat



Obrázok, na ktorom je stôl

Automaticky generovaný popisTabuľka 3 – LL gramatika

Tabuľka 4 – precedenčná tabuľka