FAKULTA INFORMACNÍCH TECHNOLOGIÍ

VYSOKÉ UCENÍ TECHNICKÉ V BRNĚ

Dokumentace k projektu

Implementace překladače imperativního jazyka IFJ22

Tým **xlukas15** **– varianta TRP**

Rozšíření: FUNEXP

**Ondřej Lukášek (xlukas15) – Vedoucí 25 %**

Ondřej Koumar (xkouma02) 25 %

Jonáš Morkus (xmorku03) 25 %

Milan Menc (xmencm00) 25 % 7. prosince 2022

# Úvod

Cílem našeho projektu byl navrhnout a naimplementovat překladač v jazyce C pro jazyk IFJ22, který poté překladač přeloží do cílového jazyka IFJcode22. Jazyk IFJ22 je podmnožinou jazyka PHP. Zvolili jsme si variantu zadání, kde musíme pracovat s tabulkou symbolů pomocí tabulky s rozptýlenými položkami. K samotné implementaci jsme si přidali i rozšíření v podobě FUNEXP.

# Lexikální analyzátor

Námi implementovaná lexikální analýza (soubory *lex.c* a *lex.h*) má za úkol projít celý vstupní program. Lexikální analyzátor přečte celý program a rozdělí je na tokeny, se kterými následně pracují další části našeho překladače.

Program rozděluje tokeny podle námi navrhnutého konečného automatu (obrázek níže) s výjimkou toho, že nejprve zkontroluje, že vstupní program obsahuje prolog, protože pokud se prolog nenačte, tak předá chybovou hlášku a ani nepřejde ke konečnému automatu. Konečný automat jsme museli během naší práce párkrát předělat, protože při testování lexikálního analyzátoru jsme narazili na chyby, které jsme v FSM neměli zahrnuté. Lexikální analyzátor obsahuje jednu základní funkci, která čte token (ReadToken). Po jednom znaku načítá každý znak podle FSM a podle FSM se udělá token nebo neudělá a vyvolá se chyba. Klíčová slova nenačítáme po znacích, ale načítáme rovnou celý řetězec. Taktéž načítáme jako token i epilog a podle něj zjišťujeme, jestli za ním něco následuje. Pokud by ovšem nastalo, že by se ihned po epilogu načetl EOF, tak lexikálnímu analyzátoru řekneme, že posledním tokenem je nadále epilog.

# Syntaktický analyzátor

Návrh implementace syntaktického analyzátoru spočíval v tvorbě LL (1) gramatiky. Při pokusu o první převedení do LL tabulky jsme narazili na problém a to ten, že naše gramatika nebyla LL (1), ale LL (7). Od prvního správného vytvoření LL (1) gramatiky proběhlo asi čtrnáct změn, dokud jsme se nedostali do finální podoby, jelikož jsme nacházeli pořád další chyby anebo možnosti optimalizace. V gramatice zpracováváme i korektnost výrazu.

Samotná implementace syntaktického analyzátoru (soubory *parser.c* a *parser.h*) začala tím, že jsme si nadefinovali dvourozměrné pole, ve kterém byla celá naše gramatika, ale poté jsme přešli na rekurzivní sestup. V rekurzivním sestupu se volají funkce mezi neterminály a následně se jde postupně podle gramatiky.

# Sémantický analyzátor

V sémantickém analyzátoru využíváme precedenční tabulky, podle které se poté využívá dvou zásobníků, kde na jednom přijde expression od syntaktické analýzy a druhý zásobník se využívá pro vyhodnocení výrazu. Na druhý zásobník se postupně pushují části expressionu a pomocí precedenční tabulky se vyhodnocuje co se má pushnout a co nemá, aby byla zachována precedence operátorů.

Sémantická kontrola probíhá ve dvou průchodech. V prvním průchodu načteme nadeklarované/nadefinované funkce a abychom měli splněnou podmínku, že můžeme volat funkci před její deklarací/definicí. Zároveň sémantický analyzátor úzce spolupracuje s tabulkou symbolů, kde se kontrolují různé typy a jaké by měly často být např. při výstupu funkce že je typ takový jaký má být a jestli se nacházejí kompatibilní operátory mezi čísly. Nakonec se generuje kód expressionu, kde jsme využívali zásobníkových instrukcí.

# Tabulka symbolů

TODO – Jsem opice a zapomněl se vás na to zeptat, ale protože už je po 22:00, tak bych dostal beating, kdybych dělal bordel, tak si vás někoho zítra odchytím a vyzpovídám.

# Generování kódu

TODO

# Testování

TODO – Tady bych mluvil akorát o sobě a svém testování, takže na to se zítra taktéž zeptám.

# Práce v týmu

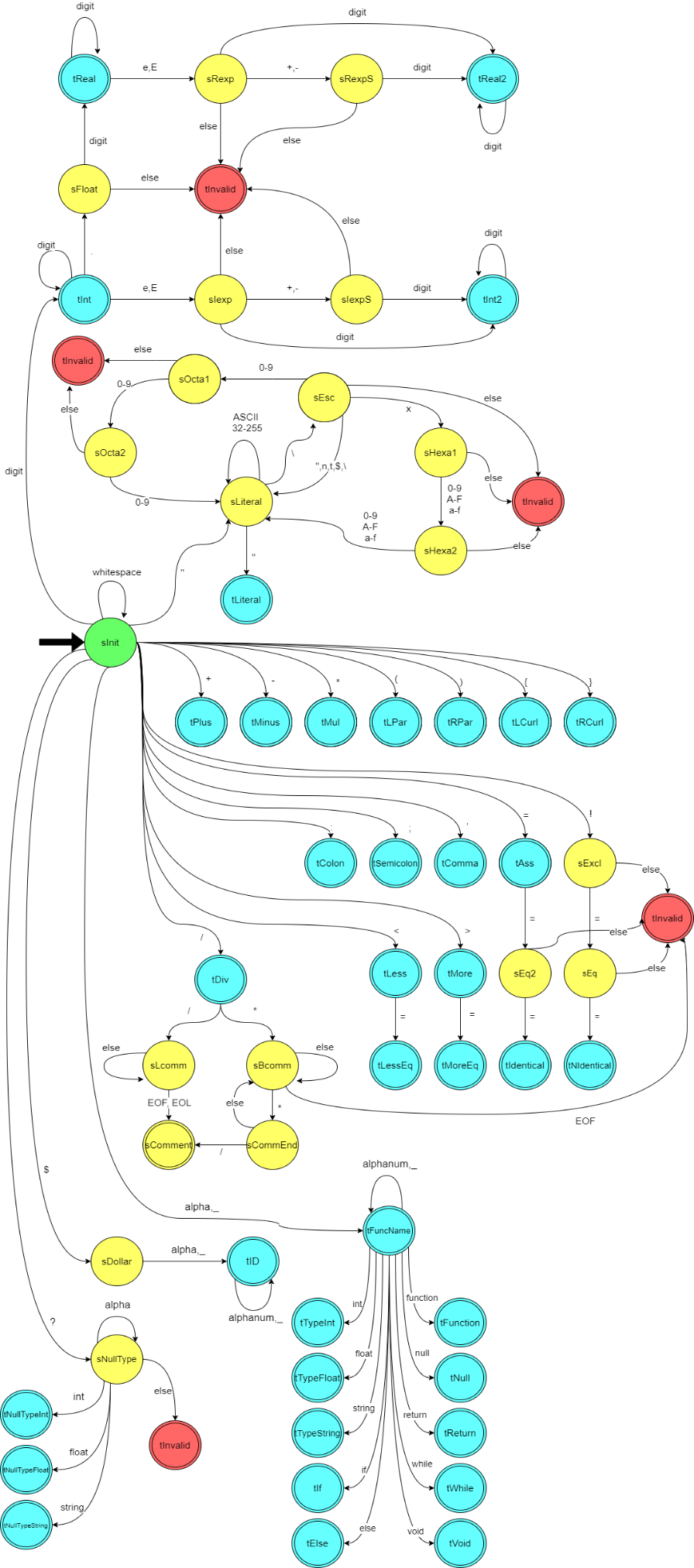
Na projektu jsme začali pracovat na začátku října a nejdříve jsme se drželi každotýdenního setkání a případná krátká setkání na Discordu. Od listopadu jsme se již nescházeli pravidelně, ale namísto toho jsme se častěji scházeli na Discordu a občas jsme se sešli i prezenčně.

Jako komunikační kanál jsme zvolili již zmíněný Discord. Taktéž jsme hned založili GitHub repositář, který jsme využívali ke sdílení našich kódů a materiálů potřebných k práci.

|  |  |
| --- | --- |
| **Člen** | **Práce, kterou udělal nebo na které se podílel** |
| Ondřej Lukášek | Vedení, dělení práce, lexikální analýza, testování, konzultace, syntaktická analýza, sémantická analýza |
| Ondřej Koumar | Lexikální analýza, testování, konzultace, syntaktická analýza, generování kódu, makefile |
| Jonáš Morkus | Lexikální analýza, testování, konzultace, syntaktická analýza, generování kódu |
| Milan Menc | Syntaktická analýza, testování, konzultace, dokumentace |

# Speciální použité techniky a algoritmy

TODO – alokace, support files, dbgmsg



Konečný automat lexikálního analyzátoru (FSM)