VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

Formální jazyky a překladače – projekt 1

Autori práce:

Marek Dančo - Vedúci	xdanco00 - 40%
Martin Olšiak	xolsia00 - 21%
František Fúdor	xfudor00 - 23%
Jakub Drobena	xdrobe01 - 16%

Obsah

Rozdelenie práce

xdanco00 - Tabuľka symbolov, návrh riešenia, rozdelenie úloh, precedenčná analýza, kontrola kvality kódu, generovanie kódu, dynamické reťazce, sémantická analýza.

xolsia00 - Scanner, syntaktická analýza.

xfudor00 - Dátová štruktúra listu tokenov, prvý prechod tabuľkou symbolov, pomocné funkcie generátora (vyjma funkcie na generovanie precedenčne analyzovaného kódu).

xdrobe01 - Programovanie vstavaných funkcií a testov, písanie dokumentácie.

Rozdelenie práce bolo nerovnomerné kvôli rôznym schopnostiam jednotlivých členov.

Dátové štruktúry prekladača

1.String

Slúži na ukladanie jednotlivých častí kódu ktoré sú následne posielané do CodeAnalyzátoru a používajú sa neskôr aj v tokenoch a aj v Symtable. Je to dátový typ struct ktorý sa skladá z 2 častí a to z dát kde sa nachádzajú samostatné stringy a z dĺžky kde sa nachádza samotná dĺžka reťazca.

2.Token

Slúži na zachovanie už analyzovaných lexémov. Je to Struct v ktorom sa nachádza string lexému a jeho typ, ktorý mu určil analyzátor.

3.List

Táto dátová štruktúra slúži na uskladnenie jednotlivých tokonenov. Neskôr sa používa v parseri, kde sú jeho jednotlivé tokeny previazané jeden za druhým štýlom jednosmerného listu. Obsahuje ukazateľ na prvý token ktorý sa v ňom nachádza, ukazateľ na posledný prvok ktorý sa v ňom nachádza a počet tokenov ktoré sa v ňom nachádzajú. Je to síce jednosmerne previazaný list, ale podporuje aj funkcie na prácu ako so zásobníkom (push a pop), ktoré sa využívajú pri precedenčnej analýze a počítaní ifov.

4.Symtable

Tabuľka symbolov je jednoduchý binárny vyhľadávací strom. Každá položka obsahuje ukazovatele doprava a doľava a dátovú štruktúru symbolu, ktorá sa skladá z identifikátoru, listu parametrov, poľa návratových typov (typu premennej), úrovne, na ktorej bola premenná definovaná (ak je symbol premenná) a bool premennej defined, ktorá určuje, či je premenná platná v danej úrovni. Úrovne platnosti začínajú od 0 a čím vyššia je hodnota, tým menší je daný rozsah.

Časti prekladača

1.Scanner

Scanner slúži na prvotné spracovanie kódu, ktorý je funkciou main načítaný zo stdin. Funkcia codeAnalyzer si vytvorí jednotlivé lexémy, ktoré následne pošle automatu, ktorý zistí, akého typu jednotlivé lexémy sú, a uloží ich s hodnotami do tokenov, ktoré sú následne vložené do token listu, s ktorým budú pracovať ostatné funkcie.

2.Parser

Po naplnení listu codeAnalyzerom sa zavolá funkcia parse rieši syntaktickú analýzu a sémantickú analýzu. Prvý prechod listom tokenov je implementovaný ako samostatná funkcia fillSymtable, ktorá iba zistí správnosť syntaxe deklarácie funkcií a naplní nimi globálnu tabuľku, ktorá obsahuje iba funkcie. Syntaktická časť parsera nebola implementovaná pomocou zadanej LL gramatiky keďže došlo k nečakanej zmene rozdelenia práce. V našom projekte sme použili funkciu syntaktickej analýzy aj ako miesto volania semantickej anaýzy a generovania. Fukcia prejde kód riadok po riadku a rozhodne, aký typ riadku rieši. Ak sa jedná o priradenie, zavolá sa funkcia ktorá rieši syntax príkazu priradenia... Syntax týchto funkcií bol implementovaný podľa zadania projektu v jazyku IFJ20.
Sémantická analýza dostáva kód riadok po riadku a pomocou niekoľkých cyklov určuje, či nenastala nejaká zo sémantických chýb. Ak narazí na nový platný príkaz definície, vloží premennú do lokálnej tabuľky symbolov, ktorá obsahuje iba premenné danej funkcie.

3.Generátor

Generátor už je volaný ako posledný parserom. Generátor pracuje podobne ako sémantická analýza riadok po riadku, s tým, že do generátora sa posielajú potrebné informácie by reference (globálna tabuľka, lokálna tabuľka danej funkcie, zásobník ifov...). Pri prekladaní a stará sa o preloženie kódu do konečného jazyka. Vygenerovaný kód sa ukladá do stringu, ktorý je na konci vypísaný naraz, ak nenastane počas prekladu žiadna chyba.

Práca v tíme

1.Komunikácia

Na komunikáciu medzi členmi tímu sme používali Discord, kde sme mali vytvorené kanály na hlasovú komunikáciu, písomnú komunikáciu a kanál ktorý slúžil na informácie, ktoré časti projektu sú hotové. Pokiaľ nastal nejaký problém riešili sme ho taktiež cez discord.

2.Vývojové prostredie

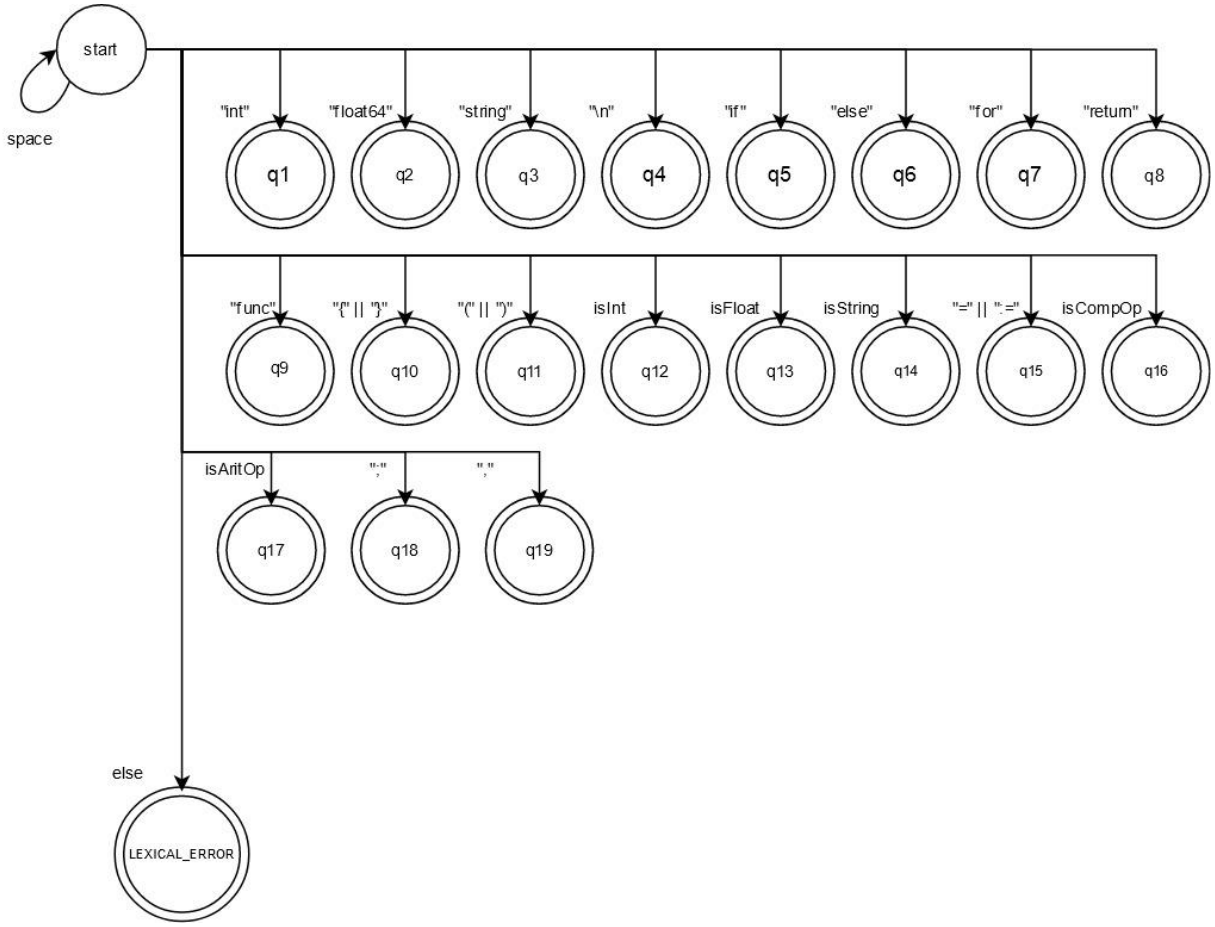
Ako vývojové prostredie sme si zvolili CLion, ktorý je pre študentov zdarma. Jednoducho sa s ňom pracuje na operačnom systéme Windows s previazaním s WSL alebo prípadne s nejakým vzdialeným školským serverom. Keďže všetci používame Windows, toto pre nás bolo vítané.

3.GitHub Repozitár

Pre skupinovú prácu na kóde sme si zvolili GitHub. K nemu sme následne pristupovali cez GitKraken GUI, ktorý nám uľahčil prácu s týmto repozitárom. Pre každú osobitnú časť práce sa vytvorila pred začiatkou práce nová vetva, kde si každý člen spravoval svoju časť práce. Riešenie merge konfliktov mal na starosti xdanco00 ako vedúci tímu a najviac informovaný člen.

Diagram Konečného Automatu

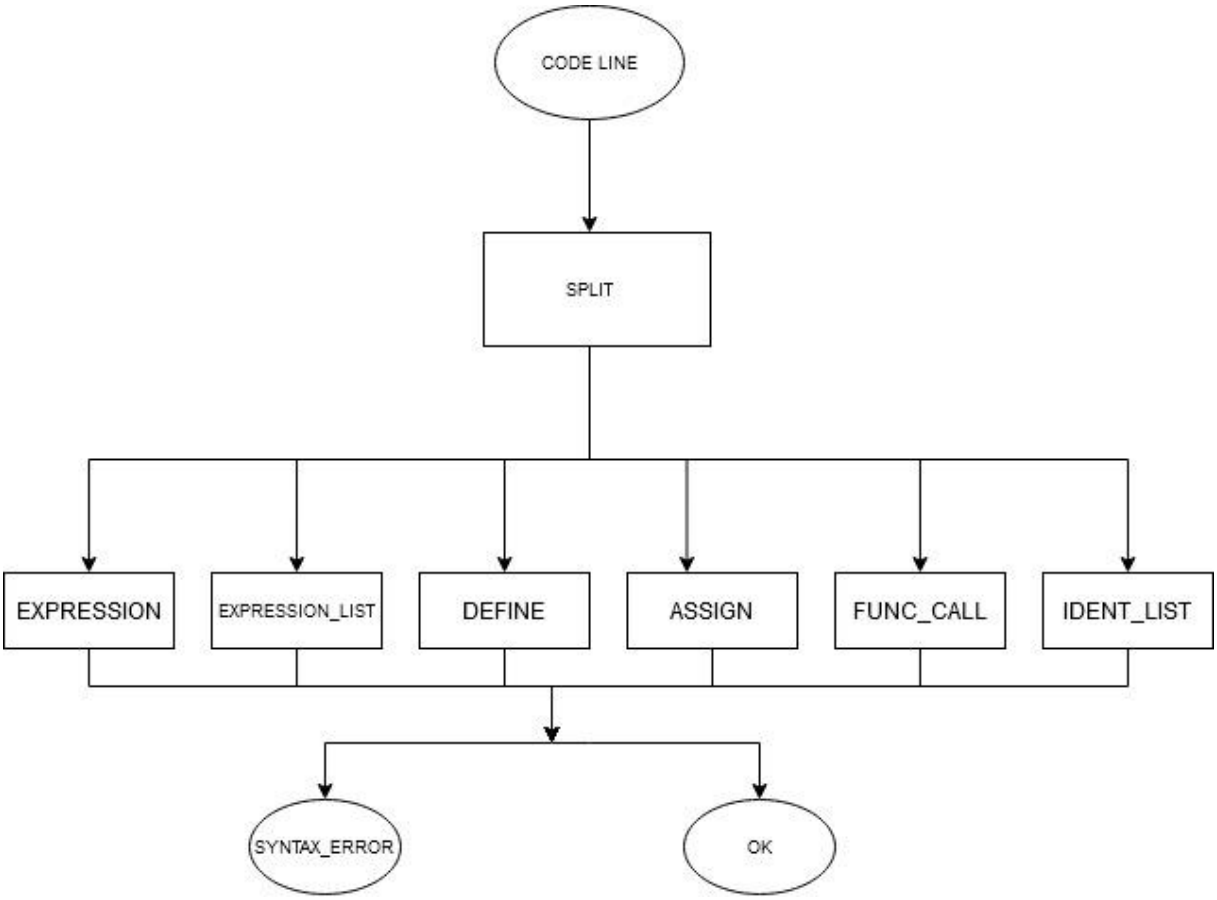
Lexikálny Analyzátor



Legenda:

Q1	INT	Q11	BRACKETS_ROUND
Q2	FLOAT64	Q12	INT_LIT
Q3	STRING	Q13	FLOAT_LIT
Q4	EOL	Q14	STRING_LIT
Q5	IF	Q15	ASIGN_OPERATOR
Q6	ELSE	Q16	COMPARATIVE_OPERATOR
Q7	FOR	Q17	ARITHMETIC_OPERATOR
Q8	RETURN	Q18	SEMICOL
Q9	FUNC	Q19	COMMA
Q10	BRACKETS_CURLY		

Riešenie Syntaktickej Analýzy



Legenda:

SPLIT - rozdelí riadok kódu, analyzuje ho a pošle príslušnej funkcii ktorá skontroluje syntax.

EXPRESSION – skontroluje syntax výrazu

EXPRESSION_LIST – skontroluje syntax zoznamu výrazov

DEFINE – skontroluje syntax definícií

ASIGN – skontroluje syntax príkazu priradenia

FUNC_CALL – skoontroluje syntax volania funkcie

IDENT_LIST – skontroluje syntax zoznamu identifikátorov

Každá z podfunkcii vracia OK pri úspechu a SYNTAX_ERROR ak narazí na syntaktickú chybu.

Keďže sme neriešili syntaktickú analýzu rekurzívnym zostupom namiesto LL tabuliek sme graficky znázornili spôsob jej funcie.