



FORMÁLNÍ JAZYKY A PŘEKLADAČE
2020/2021

Implementace překladače imperativního jazyka IFJ20

Tým 018, varianta II

Vedoucí projektu: Šesták Pavel(xsesta07)

Spoluřešitelé: Kulíšek Vojtěch(xkulis03), Plevač Lukáš(xpleva07)

Brno, 17. února 2021

Obsah

1	Úvod	2
1.1	Komunikace mezi moduly	2
1.2	Návratové hodnoty aplikace	2
2	Práce v týmu	3
3	Lexikální analyzátor	4
4	Syntaktický analyzátor	6
4.1	Rekurzivní sestup	6
4.1.1	LL gramatika	6
4.2	Precedenční analýza	8
4.2.1	Gramatika pro precedenční analýzu	8
4.2.2	Precedenční tabulka	8
5	Sémantický analyzátor	9
5.1	Tabulka symbolů	9
5.2	Kontrola datových typů	9
6	Generování kódu	10

1 Úvod

1.1 Komunikace mezi moduly

Komunikace mezi moduly probíhá převážně přes soubor globálních proměnných definovaných v rámci *global_variables.h*. Token reprezentuje aktuální zpracovávaný token od lexikální analýzy (Sekce 3 - Lexikální analyzátor). Proměnná *symtable* reprezentuje tabulku symbolů (Sekce 5.1 - Tabulka symbolů). *Precedence_stack* je globální zásobník pro precedenční analýzu zdola nahoru, použitou pro analýzu logických výrazů. *Data_type_queue* je fronta datových typů, která je použita u přiřazování výrazů do proměnných při kontrole datových typů. S frontou *token_queue* pracuje část syntaktické analýzy implementován rekurzivním sestupem a ukládá do ní tokeny do kterých se bude přiřazovat z důvodu kontroly datových typů.

1.2 Návrátové hodnoty aplikace

Návratové hodnoty explicitně vyplývají ze zadání a jejich definice je součástí hlavičkového souboru *error_codes.h*.

2 Práce v týmu

Práce na projektu probíhala v týmu, jehož složení je konstantní od začátku studia. Pro verzování projektu jsme zvolili git hostovaný na portálu GitLab. Kořenový adresář projektu je rozdělen do čtyř složek, pro oddělení jednotlivých částí projektu. Složka */doc* obsahuje popis těchto složek, dokumentaci v TeXu a popis jak je formátován zdrojový kód v rámci projektu. Druhá sekce */include* obsahuje hlavičkové soubory s definicemi a externími funkcemi s komentářem, které slouží pro komunikaci mezi jednotlivými moduly. Sekce */src* obsahuje zdrojové soubory překladače. Poslední sekcí je */tests*, v rámci které jsou implementovány automatické testy na jednotlivé moduly a datové struktury. Při commitu byl využit pipelining pro build a následné provedení automatických testů. Komunikace ohledně projektu probíhala skrz Facebookovou skupinu a schůzky pořádané na platformě Discord. Práce byla rozdělena do separátních bloků, které byly rozděleny průběžně mezi členy týmu. Schůzky probíhali převážně z důvodu upřesnění funkcionality daných modulů a integrace jednotlivých modulů do funkčního celku.

3 Lexikální analyzátor

První blok překladače je lexikální analyzátor známý také jako skener. Jeho úkolem je rozložit zdrojový kód ze standardního vstupu na sekvenci tokenů. Lexikální analyzátor byl implementován pomocí konečného automatu (Obrázek 1). Konečné stavy automatu reprezentují jednotlivé tokeny. V případě, že automat se nachází v koncovém stavu a znak na vstupu není schopný zpracovat, vrací token do syntaktického analyzátoru. V opačném případě kdy automat není schopen znak přijmout a automat se nenachází v koncovém stavu končí překlad na lexikální chybu. Každý stav je reprezentován funkcí a aktuální stav je uložen pomocí ukazatele na funkci.

Stavy konečného automatu byly reprezentovány funkcemi, kdy další stav byl uložen v ukazateli na funkci, který byl volán v hlavní smyčce. Konečný automat je doplněn mnoha boolovskými proměnnými, které mu pomáhají se rozhodovat, zda-li číst další znak ze vstupu nebo zda byl načten již kompletní token.

Token je reprezentován strukturou obsahující informace o typu tokenu, jeho atributu a datovém typu. Jako atribut se rozumí hodnota literálu či název identifikátoru. Typ tokenu a datový typ je realizován pomocí výčetového typu. Atribut je realizován pomocí datového typu union.

Komunikační rozhraní s lexikálním analyzátozem je pomocí funkce *get_token()*, která aktualizuje hodnotu globální proměnné token.

Lexikální analyzátor má počítadlo aktuálního řádku *lex_line_counter*, které je přístupné syntaktické i sémantické analýze při chybovém výstupu pro informování uživatele, na kterém řádku byl překlad ukončen.

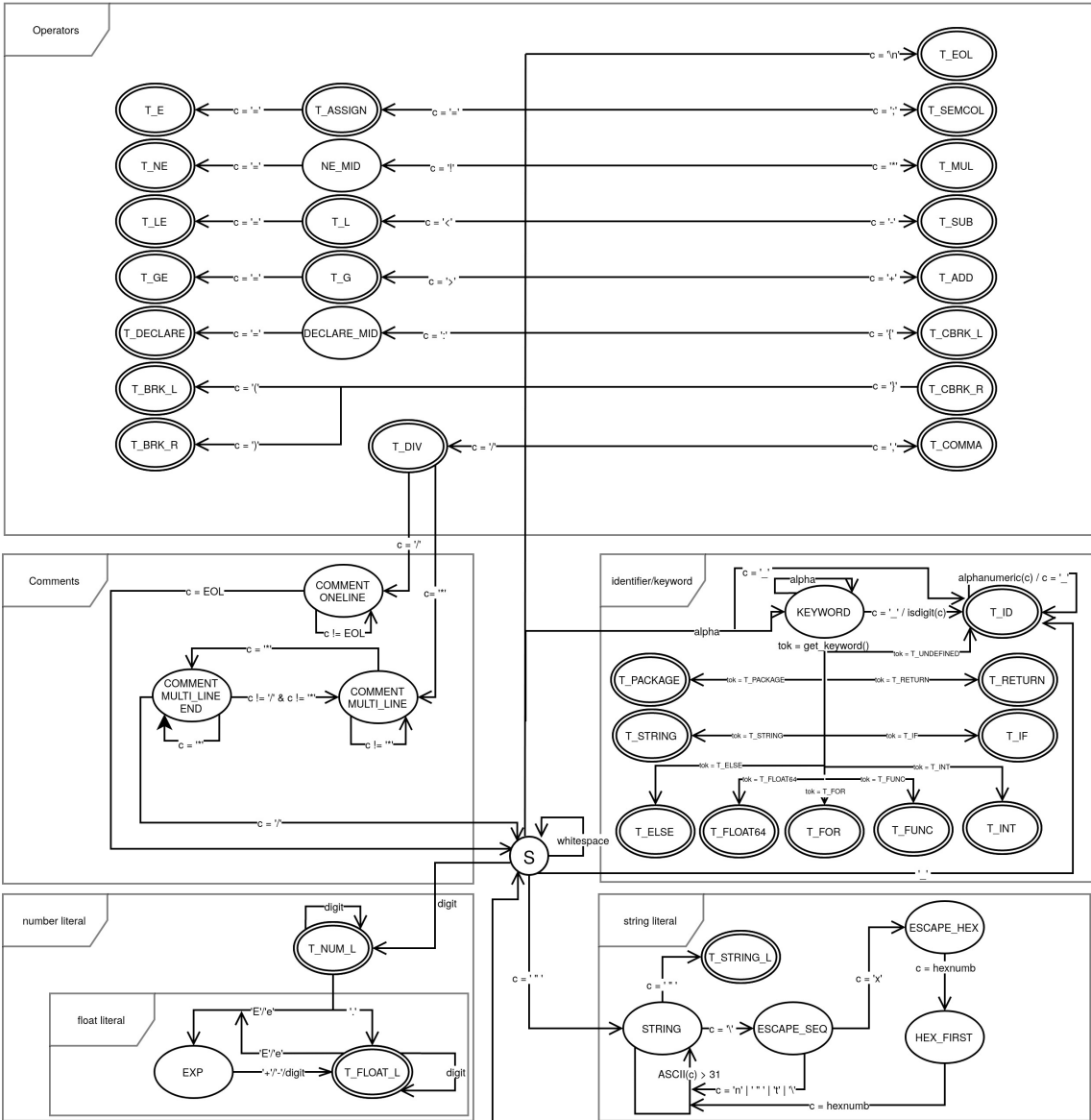
Obrázek 1: Návrh konečného automatu pro lexikální analyzátor

```
digit = ^[0-9]$
```

```
alpha = ^[A-Za-z]$
```

alpha-numeric = alpha+numeric

```
hexnumb = ^[0-9a-fA-F]$
```



4 Syntaktický analyzátor

Syntaktická analýza řídí překlad a je rozdělena do dvou celků - analýza pomocí rekurzivního sestupu a precedenční analýzy. Hlavní část syntaktické analýzy je implementována rekurzivním sestupem, kde každý neterminál reprezentuje samostatnou funkci. Syntaktická analýza volá lexicální analyzátor, který jí vrací další token ze vstupu. V rámci rekurzivního sestupu se snaží pokračovat na základě tokenu podle jednoho z pravidel definovaných v rámci LL gramatiky. V případě, že narazí na výraz tak syntaktický analyzátor volá precedenční analýzu, která výraz vyhodnotí a uloží datový typ na zásobník datových typů. Syntaktická analýza provede kontrolu datových typů. V průběhu simulace derivačního stromu se volají sémantické kontroly a provádí se generování cílového kódu.

4.1 Rekurzivní sestup

Rekurzivní sestup syntaktické analýzy je implementován v rámci souboru *syntax_analyzer.c*.

4.1.1 LL gramatika

$$G_{LL} = (N, \Sigma, P, E)$$

$N = \{ \langle \text{statement} \rangle, \langle \text{can_EOL} \rangle, \langle \text{expression} \rangle, \langle \text{multy_assign} \rangle, \langle \text{expression_list} \rangle, \langle \text{param_list} \rangle, \langle \text{param} \rangle, \langle \text{comparison} \rangle, \langle \text{expression} \rangle, \langle \text{rel_operator} \rangle, \langle \text{for_declaration} \rangle, \langle \text{for_comparison} \rangle, \langle \text{for_assignment} \rangle, \langle \text{func} \rangle, \langle \text{param_define} \rangle, \langle \text{return_list} \rangle, \langle \text{return_data_type} \rangle, \langle \text{data_type} \rangle, \langle \text{mu_param_define} \rangle, \langle \text{prolog} \rangle \}$

$\Sigma = \{ T_EOF, T_ELSE, T_FLOAT64, T_FOR, T_FUNC, T_IF, T_INT, T_PACKAGE, T_RETURN, T_STRING, T_ID, T_NUM_L, T_FLOAT_L, T_STRING_L, T_COMMA, T_CBRK_R, T_CBRK_L, T_SEMICOL, T_SUB, T_ADD, T_NE, T_MUL, T_E, T_GE, T_G, T_EOL, T_LE, T_L, T_BRK_R, T_ASSIGN, T_DECLARE, T_BRK_L, T_DIV \}$

$P = \{$

1: $\langle \text{can_EOL} \rangle \rightarrow \epsilon$

2: $\langle \text{can_EOL} \rangle \rightarrow T_EOL \langle \text{can_EOL} \rangle$

3: $\langle \text{prolog} \rangle \rightarrow \langle \text{can_EOL} \rangle T_PACKAGE \text{ main } T_EOL \langle \text{func} \rangle // \text{define package}$

4: $\langle \text{func} \rangle \rightarrow \langle \text{can_EOL} \rangle T_FUNC T_ID T_BRK_L \langle \text{param_define} \rangle T_BRK_R \langle \text{return_list} \rangle T_CBRK_L \langle \text{statement} \rangle T_CBRK_R \langle \text{can_EOL} \rangle \langle \text{func} \rangle$

5: $\langle \text{func} \rangle \rightarrow T_EOF$

6: $\langle \text{param_define} \rangle \rightarrow \epsilon$

7: $\langle \text{param_define} \rangle \rightarrow T_ID \langle \text{data_type} \rangle$

8: $\langle \text{param_define} \rangle \rightarrow T_ID \langle \text{data_type} \rangle T_COMMA$

$\langle \text{mu_param_define} \rangle$ 9: $\langle \text{mu_param_define} \rangle \rightarrow T_ID \langle \text{data_type} \rangle$

10: $\langle \text{mu_param_define} \rangle \rightarrow T_ID \langle \text{data_type} \rangle T_COMMA \langle \text{mu_param_define} \rangle$

11: $\langle \text{return_list} \rangle \rightarrow T_BRK_L \langle \text{return_data_type} \rangle T_BRK_R$

12: $\langle \text{return_list} \rangle \rightarrow \epsilon$

13: $\langle \text{data_type} \rangle \rightarrow T_INT$

14: $\langle \text{data_type} \rangle \rightarrow T_FLOAT64$

15: $\langle \text{data_type} \rangle \rightarrow T_STRING$

16: $\langle \text{return_data_type} \rangle \rightarrow \langle \text{data_type} \rangle$

17: $\langle \text{return_data_type} \rangle \rightarrow \langle \text{data_type} \rangle T_COMMA \langle \text{return_data_type} \rangle$

18: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle$

19: $\langle \text{statement} \rangle \rightarrow \epsilon$

20: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle T_ID T_DECLARE \langle \text{can_EOL} \rangle \langle \text{expression} \rangle T_EOL \langle \text{statement} \rangle$

21: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle T_ID \langle \text{multy_assign} \rangle T_ASSIGN \langle \text{can_EOL} \rangle \langle \text{expression_list} \rangle T_EOL \langle \text{statement} \rangle$

22: $\langle \text{multy_assign} \rangle \rightarrow \epsilon$

23: $\langle \text{multy_assign} \rangle \rightarrow T_COMMA \langle \text{can_EOL} \rangle T_ID \langle \text{multy_assign} \rangle$

24: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle T_ID T_BRK_L \langle \text{can_EOL} \rangle \langle \text{param_list} \rangle T_BRK_R T_EOL \langle \text{statement} \rangle$

25: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle T_ID T_BRK_L T_BRK_R T_EOL \langle \text{statement} \rangle$

28: $\langle \text{param_list} \rangle \rightarrow \langle \text{param} \rangle$

29: $\langle \text{param_list} \rangle \rightarrow \langle \text{param} \rangle T_COMMA \langle \text{can_EOL} \rangle \langle \text{param_list} \rangle$

30: $\langle \text{param} \rangle \rightarrow \text{T_STRING_L}$
 31: $\langle \text{param} \rangle \rightarrow \text{T_ID}$
 32: $\langle \text{param} \rangle \rightarrow \text{T_NUM_L}$
 33: $\langle \text{param} \rangle \rightarrow \text{T_FLOAT_L}$
 34: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle \text{T_IF} \langle \text{comparison} \rangle \text{T_CBRK_L} \text{T_EOL} \langle \text{statement} \rangle$
 $\text{T_CBRK_R} \text{T_ELSE} \text{T_CBRK_L} \text{T_EOL} \langle \text{statement} \rangle \text{T_CBRK_R} \text{T_EOL} \langle \text{statement} \rangle$
 35: $\langle \text{comparison} \rangle \rightarrow \langle \text{expression} \rangle \langle \text{rel_operator} \rangle \langle \text{can_EOL} \rangle \langle \text{expression} \rangle$
 36: $\langle \text{rel_operator} \rangle \rightarrow \text{T_L}$
 37: $\langle \text{rel_operator} \rangle \rightarrow \text{T_LE}$
 38: $\langle \text{rel_operator} \rangle \rightarrow \text{T_G}$
 39: $\langle \text{rel_operator} \rangle \rightarrow \text{T_GE}$
 40: $\langle \text{rel_operator} \rangle \rightarrow \text{T_E}$
 41: $\langle \text{rel_operator} \rangle \rightarrow \text{T_NE}$
 42: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle \text{T_FOR} \langle \text{for_declaration} \rangle \text{T_SEMICOL} \langle \text{for_comparison} \rangle$
 $\text{T_SEMICOL} \langle \text{for_assignment} \rangle \text{T_CBRK_L} \text{T_EOL} \langle \text{statement} \rangle \text{T_CBRK_R} \text{T_EOL} \langle \text{statement} \rangle$
 43: $\langle \text{for_declaration} \rangle \rightarrow \text{T_ID} \text{T_DECLARE} \langle \text{can_EOL} \rangle \langle \text{expression} \rangle$
 44: $\langle \text{for_declaration} \rangle \rightarrow \epsilon$
 45: $\langle \text{for_comparison} \rangle \rightarrow \langle \text{comparison} \rangle$
 46: $\langle \text{for_comparison} \rangle \rightarrow \epsilon$
 47: $\langle \text{for_assignment} \rangle \rightarrow \text{T_ID} \langle \text{multy_assign} \rangle \text{T_ASSIGN} \langle \text{can_EOL} \rangle \langle \text{expression_list} \rangle$
 48: $\langle \text{for_assignment} \rangle \rightarrow \epsilon$
 49: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle \text{T_RETURN} \langle \text{expression_list} \rangle \text{T_EOL} \langle \text{statement} \rangle$
 50: $\langle \text{statement} \rangle \rightarrow \langle \text{can_EOL} \rangle \text{T_RETURN} \text{T_EOL} \langle \text{statement} \rangle$
 51: $\langle \text{expression_list} \rangle \rightarrow \langle \text{expression} \rangle$
 52: $\langle \text{expression_list} \rangle \rightarrow \langle \text{expression} \rangle \text{T_COMMA} \langle \text{expression_list} \rangle$

}

$E = \{ \langle \text{prolog} \rangle \}$

4.2 Precedenční analýza

Precedenční analýza zdola nahoru je v rámci překladače využita k analýze výrazů. Analýza výrazu je implementována v rámci souboru *expression_analyser.c* a přístupná přes volání *expression_analyser()*. Uchovává indexy do precedenční tabulky, a implementuje práci se zásobníkem pomocí pravidel definovaných v tabulce. Výsledný datový typ výrazu je uložen do fronty datových typů.

4.2.1 Gramatika pro precedenční analýzu

$$G_{\text{exp}} = (N, \Sigma, P, E)$$

$$N = \{ \langle E \rangle \}$$

$$\Sigma = \{ T_NUM_L, T_FLOAT_L, T_STRING_L, T_ID, T_ADD, T_SUB, T_MUL, T_DIV, T_BRK_L, T_BRK_R, T_EOF \}$$

$$P = \{ \begin{array}{l} 1: \langle E \rangle \rightarrow \langle E \rangle T_ADD \langle E \rangle \\ 2: \langle E \rangle \rightarrow \langle E \rangle T_SUB \langle E \rangle \\ 3: \langle E \rangle \rightarrow \langle E \rangle T_MUL \langle E \rangle \\ 4: \langle E \rangle \rightarrow \langle E \rangle T_DIV \langle E \rangle \\ 5: \langle E \rangle \rightarrow T_BRK_L \langle E \rangle T_BRK_R \\ 6: \langle E \rangle \rightarrow T_NUM_L \\ 7: \langle E \rangle \rightarrow T_FLOAT_L \\ 8: \langle E \rangle \rightarrow T_STRING_L \\ 9: \langle E \rangle \rightarrow T_ID \\ \end{array} \}$$

$$E = \{ \langle E \rangle \}$$

4.2.2 Precedenční tabulka

Precedenční tabulka je implementována jako dvou rozměrné pole znaků

	T_NUM_L	T_FLOAT_L	T_STRING_L	T_ID	T_ADD	T_SUB	T_MUL	T_DIV	T_BRK_L	R_BRK_R	T_EOF
T_NUM_L	-	-	-	-					-		
T_FLOAT_L	-	-	-	-					-		
T_STRING_L	-	-	-	-					-		
T_ID	-	-	-	-					-		
T_ADD											
T_SUB											
T_MUL											
T_DIV											
T_BRK_L										=	-
T_BRK_R	-	-	-	-					-		
T_EOF										-	-

5 Sémantický analyzátor

Sémantický analyzátor je volán syntaktickým analyzátozem. Kontroluje korektní volání funkcí a jejich deklarace.

5.1 Tabulka symbolů

Tabulka symbolů podle naší verze zadání měla být implementována jako hashovací tabulka. Položka hashovací tabulky je definována jako datový typ *symtable_item_T*, který v sobě uschovává informace o datovém typu, názvu a zda byla již deklarovaná proměnná. V případě funkcí ještě nese informaci o datových typech parametrů a návratových hodnot. V rámci projektu je použito pole hashovacích tabulek, které je celé implementováno jako dvou rozměrné pole. Rozsah pole je definován pomocí konstant *SYM_MAX_SCOPE* a *SYM_TOP_ITEMS*. Tabulka symbolů dále ošetřuje predikci funkcí, které nebyli doposud deklarovány. Za tímto účelem byly přidány do tabulky symbolů příznaky pro zjištění, zdali byla funkce deklarována nebo volána.

5.2 Kontrola datových typů

Kontrola datových typů je založena na porovnávání datových typů ve dvou frontách. V případě že se jedná o přiřazení tak Syntaktická analýza jakmile narazí na přiřazení, tak proměnné do kterých se bude přiřazovat, uloží do globální fronty *token_queue*. Precedenční analýza výrazů ukládá výsledný datový typ do globální fronty *data_type_queue*. Jakmile precedenční analýza zpracuje výrazy, syntaktická analýza provede porovnání těchto zásobníků, jakmile neseď počet prvků nebo se datové typy neshodují, tak je překlad ukončen sémantickou chybou.

V případě že se kontroluje výraz za klíčovým slovem *return* tak syntaktický analyzátor nekládá nic do *token_queue*, ale volá tabulku symbolů, která vrací návratový typ dané funkce definovaný v prototypu funkce.

6 Generování kódu

Funkce pro generování kódu jsou implementovány v rámci souboru *code_generator.c*. Podpůrné funkce jsou volány ze syntaktické analýzy a precedenční analýzy výrazů. V projektu není implementován optimalizátor. Generovaný kód pracuje se zásobníkem a díky tomu nevzniká velké množství kompilátorových proměných. Většinu konstrukcí jsme schopni generovat přímo, výjimkou je deklarace uvnitř cyklu. Deklarace proměných uvnitř cyklů je přepsána na přiřazení. Všechny proměnné, které mají být v bloku cyklů deklaravány se ukládají do struktury nafukovacího pole a po ukončení bloku for cyklů je vygenerováno návěští, na které se skáče před vstupem do prvního cyklu v rámci bloku cyklů. Za toto návěští jsou generovány deklarace a následně se skáče zpět na začátek cyklu.