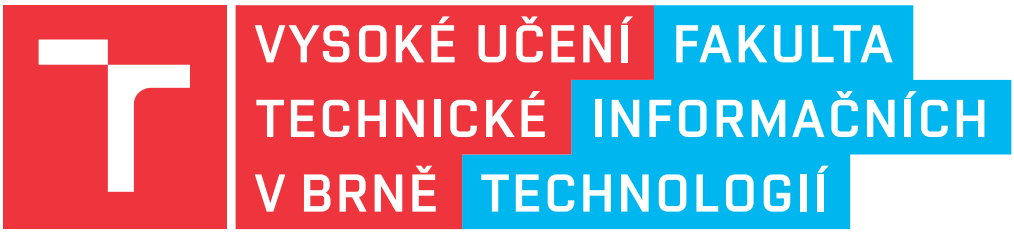


FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Formální jazyky a překladače

Dokumentace projektu do IFJ a IAL

Tým 066, varianta II

Roman Ondráček	xondra58	28%	vedoucí
Pavel Raur	xraurp00	28%	
František Jeřábek	xjerab25	28%	
Radim Lipka	xlipka02	16%	

Seznam implementovaných rozšíření

BOOLOP, BASE, FUNEXP, IFTHEN TABUNARY

Obsah

1 Úvod

Tento dokument byl vytvořen jako dokumentace společného projektu do předmětů Formální jazyky a překladače a Algoritmy. Jsou zde popsány postupy implementace jednotlivých komponent překladače a problémy spojené s jejich implementací.

2 Rozbor částí překladače

Zadáním bylo vytvořit překladač jazyka **IFJ19**, který je podmnožinou jazyka Python 3, do cílového jazyka **IFJcode19**. Jelikož se jedná o variantu projektu II, bylo nutné implementovat tabulku symbolů implementovat jako tabulku s rozptýlenými položkami.
Implementovaný překladač se dělí na několik dílčích modulů:

- Lexikální analyzátor
- Syntaktický analyzátor
- Sémantický analyzátor
- Generátor cílového kódu

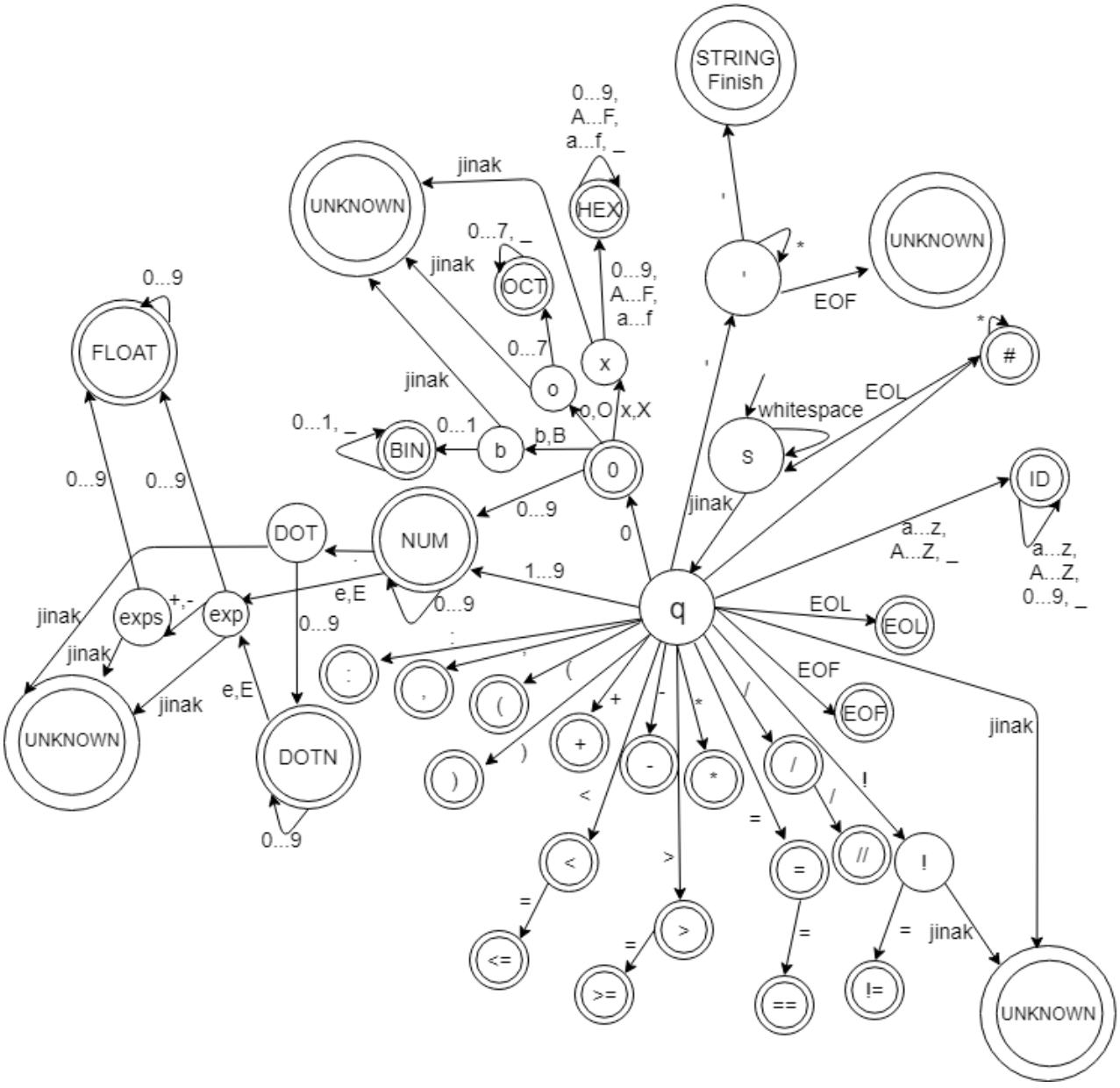
Při samotném překladu se tedy dostane ke slovu jako první lexikální analyzátor, poté přijde na řadu syntaktický analyzátor se sémantickou analýzou a následně generátor cílového kódu.

2.1 Lexikální analýza

Úkol lexikálního analyzátoru je načítat jednotlivé lexikální jednotky (lexémy) a převádět je na jednotlivé tokeny, které reprezentují daný lexém v dalších částech překladače.

Část lexikální analýzy překledače je naimplementována v modulu **scanner.c** pomocí *konečného stavového automatu*, který byl pro tuto část překladače navrhnut. Nerozpoznatelný vstupní token je označen T_UNKNOWN a token reprezentující interní chybu překladače, například chybu alokace paměti T_ERROR.

Důležitou součástí lexikální analýzy překladače jazyka IFJ19 je také zásobníkový automat, který je použit pro práci s odsazením jednotlivých řádků vstupního souboru.



Obrázek 1: Automat lexikální analýzy

2.2 Syntaktická analýza

Implementace syntaktické analýzy se nachází v modulu **parser.c**. Jedná se o implementaci metodou shora dolů, konkrétněji metodou *rekurzivního sestupu*, která je založena na LL–gramatice a LL–tabulce.

Vstupem syntaktické analýzy jsou jednotlivé tokeny z lexikální analýzy.

Jedním z úkolů syntaktické analýzy je naplnění tabulky symbolů.

2.2.1 LL–Gramatika a LL–Tabulka

1. <code> → <body> <eols> EOF
2. <body> → <definitions> <statements>
3. <definitions> → <eols> <definition> <definitions>
4. <definition> → ϵ
5. <definition> → DEF IDENTIFIER (<function_params>) : <eols> INDENT <statements> DEDENT
6. <definition> → <function_call>
7. <function_call> → IDENTIFIER (<function_params>)
8. <function_params> → ϵ
9. <function_params> → <function_param> <function_nparam>
10. <function_nparam> → , <function_param> <function_nparam>
11. <function_nparam> → ϵ
12. <function_param> → IDENTIFIER
13. <statements> → ϵ
14. <statements> → <statement> EOL <eols> <statements>
15. <statement> → <returnRule>
16. <statement> → <condition>
17. <statement> → <asignment>
18. <statement> → <whileRule>
19. <statement> → PASS
20. <returnRule> → RETURN <return_expression>
21. <return_expression> → ϵ
22. <return_expression> → (<return_expression>)

23. <return_expression> → <expression>

24. <condition> → IF <condition_expression>: EOL <eols> INDENT <statement> DEDENT <else_condition>

25. <else_condition> → ϵ

26. <else_condition> → ELSE : EOL <eols> INDENT <statement> DEDENT

27. <condition_expression> → (<condition_expression>)

28. <condition_expression> → <expression>

29. <asignment> → IDENTIFIER = <expression>

30. <whileRule> → WHILE <condition_expression> : EOL <eols> INDENT <statement> DEDENT

31. <eols> → ϵ

32. <eols> → EOL <eols>

	DEF	ID	(	)	INDENT	DEDENT	RETURN	IF	ELSE	:	WHILE	PASS	EOF	EOL	=	,
<code>	1	1												1		
<body>	2	2												2		
<defs>	3	3												3		
<def>	5	6														
<f_call>		7														
<f_params>		9														
<f_nparam>		10														10
<f_param>		12														
<stats>																
<stat>		17					15	16			18	19				
<retRule>							20									
<ret_exp>			22													
<condition>								24								
<else_cond>									26							
<cond_exp>			27													
<assignment>		29														
<whileRule>											30					
<eols>	31	31	31	31	31	31	31	31	31	31	31	31	31	32	31	31

Tabulka 1: LL–Tabulka

2.2.2 Precedenční syntaktická analýza

Precedenční syntaktická analýza je využita pro zpracování výrazů a je řízena přecedenční tabulkou. Stejně jako syntaktická analýza je precedenční analýza implementována v modulu **parser.c**.

Jelikož bylo implementováno i rozšíření TABUNARY, bylo potřeba se vypořádat s rozdílem mezi unárními operátory plus a mínus a jejich binární podobou, protože do této části syntaktické analýzy přicházejí jako od sebe nerozpoznatelné tokeny.

V precedenční analýze je hojně využit zásobník. Každý příchozí token je porovnáván s aktuálním obsahem vrcholu zásobníku a podle predenční tabulky je prováděna náležitá operace. Ty jsou typicky dvě, buď je zpracovávaný token pouze uložen na zásobník nebo je uvolněn aktuální vrchol zásobníku. Ten je redukován podle jednoho z předem definovaných pravidel. Výraz se takto postupně zpracováná a postupně redukuje, dokud není dosažen konec zpracovávaného výrazu, který je symbolizován znakem **$**. Stejný znak je na začátku precedenční analýzy vložen na vrchol zásobníku, takže posledním porovnáním u každého výrazu, který je bez chyby, je porovnání **$** s **$**. Po dokončení všech redukcí a po nalezení konce výrazu tedy vznikná výsledný strom.

$E \rightarrow id$	$E \rightarrow E + E$	$E \rightarrow E < E$
$E \rightarrow (E)$	$E \rightarrow E - E$	$E \rightarrow E > E$
$E \rightarrow id()$	$E \rightarrow E * E$	$E \rightarrow E <= E$
$E \rightarrow id(E)$	$E \rightarrow E \,/\, E$	$E \rightarrow E >= E$
$E \rightarrow id(E, ...)$	$E \rightarrow E \,//\, E$	$E \rightarrow E\ and\ E$
$E \rightarrow E == E$	$E \rightarrow E\ != E$	$E \rightarrow E\ or\ E$
$E \rightarrow not\ E$		

Tabulka 2: Redukční pravidla

	$*$	$/$	$//$	$+$	$-$	$<$	$>$	$<=$	$>=$	$==$	and	or	not	$!=$	$($	$)$	id	$\$$
$*$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$/$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$//$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$+$	$<$	$<$	$<$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$-$	$<$	$<$	$<$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$<$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
$>$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
$<=$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
$>=$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
$==$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
and	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$>$	$>$	$<$	$<$	$<$	$>$	$<$	$>$
or	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$>$	$<$	$<$	$<$	$>$	$<$	$>$
not	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$<$	$>$	$<$	$>$	$<$	$>$
$!=$	$<$	$<$	$<$	$<$	$<$	$-$	$-$	$-$	$-$	$-$	$>$	$>$	$<$	$-$	$<$	$>$	$<$	$>$
$($	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$=$	$<$	$-$
$)$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$-$	$>$	$-$	$>$
id	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$=$	$>$	$-$	$>$
$\$$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$-$	$<$	$-$

Tabulka 3: Tabulka precedenční syntaktické analýzy

2.3 Sémantická analýza

Sémantické kontroly jsou přidruženy k rekurzivnímu sestupu syntaktické analýzy. Naimplementovány jsou v modulu **semantic_analysis.c**. Největší část sémantických kontrol spočívá v kontrole datových typů výrazů, které vstupují do různých operací. V případě nejednotnosti těchto datových typů poté probíhá implicitní přetypování. Jelikož byla implementována podpora rozšíření BOOLOP, kromě implicitního přetypování z typu *int* na *float*, je nutné podporovat i přetypování z typu *bool* na *int* a *float*.

2.4 Generátor cílového kódu

Generátor cílového kódu byl implementován v modulu **inter_code_generator.c**. Úkolem tohoto modulu je vytvářet cílový kód, tedy v našem případě kód **IFJcode19**.

2.5 Tabulka symbolů a další datové struktury

Tabulka symbolů je implementována jako tabulka s rozptýlenými položkami v modulu **symtable.c**. Tabulka symbolů uchovává informace o všech jednotkách vyskytujících se ve vstupním programu.

Pro práci s různě dlouhými vstupními řetezci byl v modulu **dynamic_string.c** implementován datový typ dynamický string. V lexikální analýze je také použit již dříve zmíněný zásobník v modulu **stack.c** pro práci s různými úrovněmi zanoření lexémů.

Precedenční analýza poté pracuje se zásobníkem z modulu **tree_element_stack.c**, na kterém probíhá porovnávání vrcholu tohoto zásobníku s nově přicházejícími tokeny.

3 Implementovaná rozšíření

3.1 BOOLOP

Pro implementaci rozšíření BOOLOP byla přidána další pravidla do precedenční analýzy, konkrétně pravidla pro **not, or, and** a **==** a zároveň další sémantická pravidla pro kontrolu datových typů vstupujících do těchto operací.

3.2 BASE

Implementace rozšíření BASE spočívá v přidání a vypořádání se s dalším způsobem zadávání číselných konstant. Do lexikální analýzy byly přidány stavy pro přijímání lexémů konstant zapsaných v binární, oktalové a hexadecimální soustavě, jejich převod na tokeny a předání dalším částem překladače.

3.3 FUNEXP

Rozšíření FUNEXP požadovalo podporu volání funkce jako součást výrazu a také výrazy jako parametry funkce při jejím volání. Do precedenční analýzy pro to byla přidány pravidla, která toto umožňují.

3.4 IFTHEN

V rozšíření IFTHEN se měl podporovat ternární operátor, **elif**, a konstrukce **if** bez části **else**. Právě poslední zmíněné rozšíření je naším překladačem podporováno.

3.5 TABUNARY

V rozšíření TABUNARY se mělo rozlišovat mezi unárním a binárním mínusem a plusem v precedenční analýze a podporou odsazování jak pomocí mezer, tak pomocí tabulátorů i prolínání těchto dvou způsobů odsazení. Část tohoto rozšíření, ve které se mělo rozlišovat právě mezi binárními a unárními operátory nebyla implementována. Pro část rozšíření, která se týká odsazování pomocí tabulátorů bylo třeba rozšířit zásobík u lexikální analýzy, aby bral v úvahu i tabulátory a ne jenom mezery.

4 Týmová spolupráce

Od začátku řešení projektu jsme se pravidelně scházeli nejdříve přibližně jedenkrát týdně a s blížícím se termínem odevzdání poté i vícekrát za týden. Pro rychlou komunikaci mezi sebou v reálném čase jsme si zvolili službu **Slack.com**, kde jsme ihned řešili všechny problémy, které nastaly.

Jako verzovací systém jsme použili **Git** hostovaný na stránce **GitHub**.

4.1 Rozdělení práce

- Roman Ondráček - implementace tabulky symbolů, implementace generátoru cílového kodu, pomocné abstraktní datové typy, testování
- Pavel Raur - implementace lexikální analýzy, implementace generátoru cílového kodu
- František Jeřábek - implementace syntaktické analýzy, implementace sémantické analýzy
- Radim Lipka - dokumentace, testování

5 Závěr

Cílem projektu bylo pochopit a prakticky si vyzkoušet metody a techniky vysvětlované v předmětech IFJ a IAL a ze všeho nejvíc pochopit, z čeho se skládá překladač a jak fungují jeho jednotlivé části.

Práce na projektu, ať už studium teorie, návrhy nebo samotná implementace nám přinesla řadu zkušeností, nových znalostí a rozšířila nám obzory na poli překladačů, jazyků a algoritmů.

6 Použitá literatura, nástroje, programy

- Alexandr Meduna a Roman Lukáš, Formální jazyky a překladače, prezentace k přednáškám
- Jan M. Honzík, Ivana Burgetová, Bohuslav Křena, Algoritmy, prezentace k přednáškám
- Zbyněk Křivka a Radim Kocman, Demonstrační cvičení IFJ, Implementace překladače IFJ19
- cppreference.com, https://en.cppreference.com/
- Manuálové stránky jazyka C
- GitHub, https://github.com/
- Slack, https://slack.com/