Dokumentace společného projektu do předmětů IFJ a IAL

Implementace překladače imperativního jazyka IFJ17

Tým 062, variant II

Členové týmu:

Samek Jakub (vedoucí týmu)	xsamek07	25%
Karpíšek Miroslav	xkarpi05	25%
Trubka Jakub	xtrubk00	25%
Roman Zajíc	xzajic16	25%

Rozšíření:

SCOPE, UNARY, FUNEXP, IFTHEN, BASE

Obsah

1. Úvod

V této dokumentaci bude popsán postup tvorby projektu do předmětů *Formální jazyky a překladače* a *Algoritmy*. Projekt je zaměřen na tvorbu překladače pro imperativní jazyk IFJ17, který vychází z jazyku *FreeBasic*.

Překladač lze rozdělit na tři části:

a. Lexikální analýza
b. Syntaktická a sémantická analýza
c. Generátor instrukcí

Náš tým se rozhodl pro implementaci verze II, to znamená implementování tabulky symbolů pomocí tabulky s rozptýlenými položkami (dále hashovací tabulka).

V příloze na konci dokumentu je diagram použitého konečného automatu, precedenční tabulka a LL-gramatika

2. Implementace

Kromě implementování základní požadované funkcionality jsme se rozhodli pro přidání některých rozšíření *(SCOPE, UNARY, FUNEXP, IFTHEN, BASE)*

a. Lexikální analýza

Lexikální analyzátor zpracovává vstupní zdrojový kód a vytváří z něj tokeny pro další části překladače. Jednotlivé tokeny mohou být celočíselné nebo desetinné lexémy, řetězcové lexémy, identifikátory (v této fázi i klíčová slova) a komentáře. Komentáře se ovšem dále neposílají, je pouze kontrolován validní formát zápisu a poté jsou zahozeny. Samotná lexikální analýza vychází z konečného automatu (viz. příloha a.).

Konečný automat je implementovaný konstrukcí *SWITCH-IF*, tedy jednotlivé stavy jsou vybírány ve switchi a v nich je podle podmínky v *IF* rozhodnuto jaký bude následující stav automatu.

b. Syntaktická a sémantická analýza

Syntaktická analýza má za úkol ověřit, jestli posloupnost tokenů (reprezentace vstupního kódu v této části překladače) vyhovuje některému z derivačních stromů. Syntaktická analýze vychází z LL-gramatiky (viz příloha c.).

V sémantické analýze se kontrolují operace na datovými typy, zdali proměnné a funkce byly pouze definovány nebo i deklarovány. Tyto operace jsou závislé na datech v tabulkách symbolů.

Jednotlivá gramatická pravidla jsou implementovány jako funkce v souborech parser.c /.h. Pokud pro daný neterminál existuje více pravidel (neterminál <assigment>), daná funkce vnitřně aplikuje jednotlivá pravidla v konstrukci *IF-ELSEIF*.

c. Generátor instrukcí

V této části dochází k samotnému generování cílového kódu(IFJcode17). Zde jsou implementovány vestavěné funkce jazyka IFJ17, dochází k typové kontrole při dělení a podobně. Instrukce jsou uloženy ve dvou lineárně vázaných seznamech (pro případný průchod optimalizátorem)

3. Algoritmy

a. Hashovací tabulka

Je datová struktura, sloužící vyhledávání dat pomocí klíče, který odkazuje na první položku v tabulce. Pod jedním klíčem může být zřetězeno více položek.

Dalším problémem je překrývání prvků pod jedním indexem v tabulce, v případe, kdy tabulka není zcela naplněna, to lze řešit upravením hashovací funkce.

Konstrukce hashovací tabulky a funkcí nad ní operujících byla částečně převzata/inspirována domácím úkolem z předmětu *Jazyk C.* Jednotlivé tabulku se zřetězují, kvůli simulaci lokální viditelnosti. Další implementace je využita v našem garbage collectoru.

b. Lineární seznam

Je datová struktura, která podobně jako pole uchovává data, na rozdíl od pole je ale dynamická. Jednotlivé prvku v sobě mají ukazatel na následující prvek (případně i na prvek předchozí).

V našem projektu se lineární dvousměrně vázaný seznam používá ve dvou různých implementacích, jedna v syntaktické a sémantické analýze (pro jednoduché vyhledávání prvního neterminálu) a druhá v generátoru instrukcí, kde byl ve dvou případech degradován na zásobník (pro podmíněné výrazy a cykly) a v dalších dvou případech na skladování funkcí a těla programu (pro případnou optimalizaci).

4. Práce v týmu

a. Komunikace a spolupráce v týmu

Komunikace probíhající z menší části na osobních setkáních, ať už po/před výukou nebo i mimo ni, a z větší části konverzací přes sociální sítě, ve spojení s verzováním projektu v soukromém repositáři na GitHubu, jelikož většina týmu s tímto systémem již měla zkušenosti a bylo to pro nás nejsnazší řešení, vedla k velké přehlednosti v projektu.

b. Průběh práce

I přes to, že se na projektu začalo dělat poměrně brzo a pro implementování částí projektu se muselo přiklonit k částečnému samostudiu, tak práce na projektu trvala prakticky až do odevzdání (ať už u obou dobrovolných termínu, tak i u finálního), ve snaze eliminovat co největší spektrum chyb.

c. Dělba práce

Samek Jakub: testování

Karpíšek Miroslav: lexikální analýza, syntaktická a sémantická analýza

Trubka Jakub: generování tří adresného kódu, syntaktická a sémantická analýza

Roman Zajíc: dokumentace, lexikální analýza

5. Závěr

Tento projekt svým rozsahem, množstvím látky, která byla potřebná pro zdárné vytvoření funkčního překladače, a nutností týmové spolupráce byl určitě pro všechny velkou výzvou, ale každému z nás tento projekt přinesl mnoho zkušeností, ať už z hlediska programátorského, tak z hlediska práce v týmu.

6. Reference

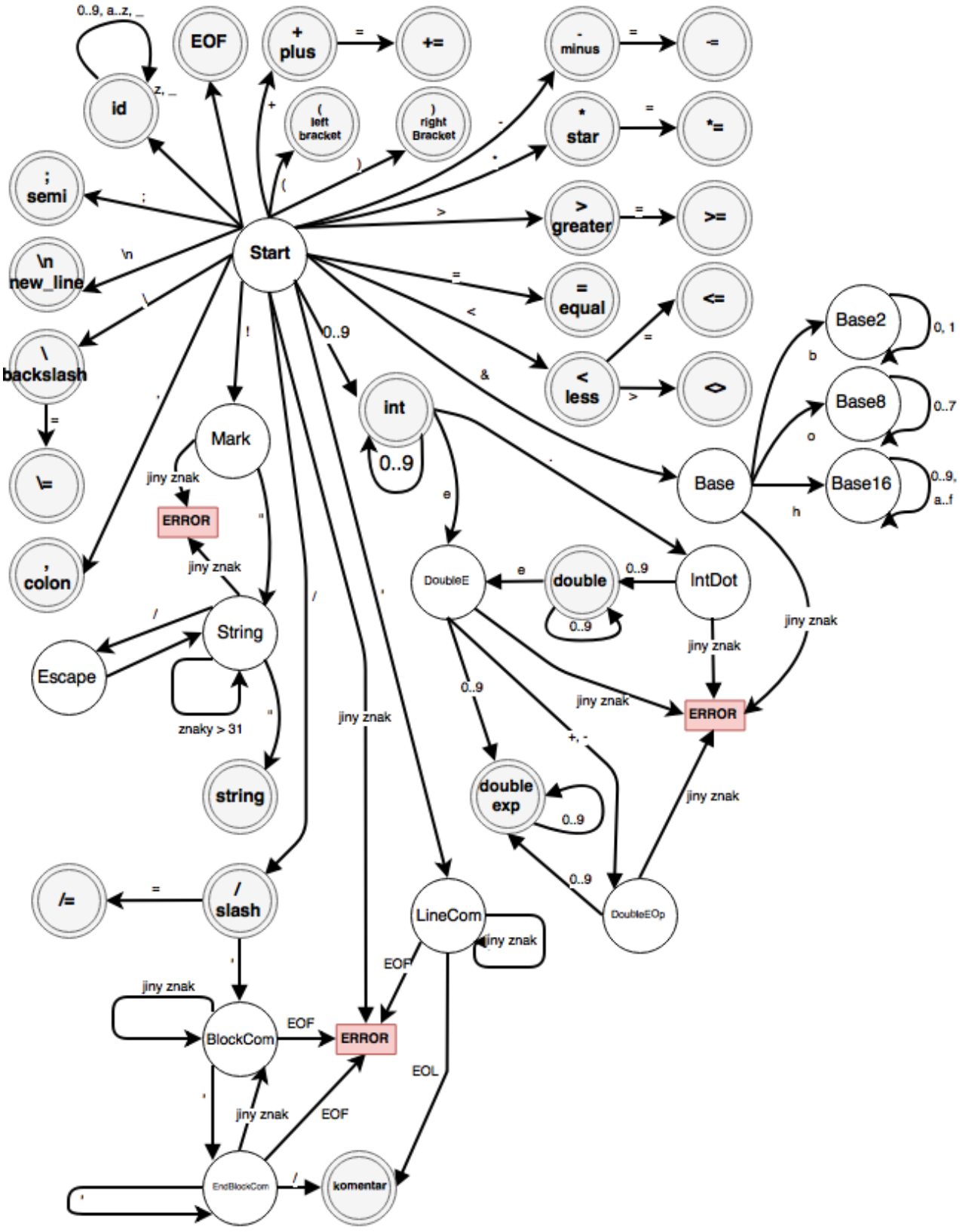
I. MEDUNA, A., LUKÁŠ, R.: *Formální jazyky a překladače IFJ – Studijní opora*. Brno: Vysoké učení technické, Verze: 1.2006+revize 2009-2015

II. KŘIVKA, Z., LUKÁŠ, R.: *Jak na projekt* https://www.fit.vutbr.cz/study/courses/IFJ/private/projekt/jak_na_projekt_prezentace.pdf

III. HONZÍK, J.: *Algoritmy IAL – Studijní opora.* Brno: Vysoké učení technické, verze:17-B

Přílohy

a. Konečný automat



b. Precedenční tabulka

7.	+, -	*, /, \	CMP	$	(	)	TYPES	,	f	i
+, -	>	<	>	>	<	>	<	>	<	<
***, /, **	>	>	>	>	<	>	<	>	<	<
CMP	<	<	0	>	<	>	<	>	<	<
$	<	<	<	0	<	0	<	0	<	<
(	<	<	<	0	<	=	<	=	<	<
)	>	>	>	>	0	>	0	>	0	0
TYPES	>	>	>	>	0	>	0	>	0	0
,	<	<	<	0	<	=	<	=	<	<
f	0	0	0	0	=	0	0	0	0	0
i	>	>	>	>	0	>	0	>	0	0

CMP: =, <>, <, <=, >, >=
TYPES: String, double, integer

c. LL-gramatika

1. <program> -> <declareList> <body>
2. <body> -> SCOPE EOL <statList> END SCOPE

3. <declareList> -> <declare> <space> <declareList>
4. <declareList> -> ε
5. <declare> -> DECLARE FUNCTION id (<paramsList>) AS <type>
6. <declare> -> FUNCTION id (<paramsList>) AS <type> EOL <statList> END FUNCTION

7. <paramsList> -> <param> <paramNext>
8. <paramsList> -> ε
9. <paramNext> -> , <param> <paramNext>
10. <paramNext> -> ε
11. <param> -> id AS <type>

12. <statList> -> <stat> EOL <statList>
13. <statList> -> ε

14. <stat> -> <declareVar>
15. <stat> -> id <assigment> <expr>
16. <stat> -> INPUT id
17. <stat> -> PRINT <expr> ; <exprList>
18. <stat> -> IF <expr> THEN EOL <statList> <ifstat>
19. <stat> -> DO WHILE <expr> EOL <statList> LOOP
20. <stat> -> RETURN <expr>
21. <stat> -> <body>
22. <stat> -> ε

23. <ifstat> -> ELSE EOL <statList> END IF
24. <ifstat> -> ELSEIF THEN EOL <statList> <ifstat>
25. <ifstat> -> END IF

26. <exprList> -> <expr> ; <exprList>
27. <exprList> -> ε
28. <declareVar> -> DIM id AS <type> <definition>
29. <definition> -> = <expr>
30. <definition> -> ε
31. <type> -> INTEGER
32. <type> -> DOUBLE
33. <type> -> STRING
34. $ -> <space> <program> <space> EOF
35. <space> -> EOL <space>
36. <space> -> ε

37. <assigment> -> +=
38. <assigment> -> -=
39. <assigment> -> *=
40. <assigment> -> /=
41. < assigment> -> \=

	<program>	<body>	<declareList>	<declare>	<paramsList>	<paramNext>	<param>	<statList>	<stat>	<ifstat>	<exprList>	<declareVar>	<definition>	<type>	<space>	$	<assigment>
EOF															37		
EOL								12	22		27		40		35	35	
SCOPE	1	2	4					12	21						36	35	
END								13		25							
DECLARE	1		3	5											36	35	
FUNCTION	1		3	6											36	35	
id					7		11	12	15		26						
AS																	
DIM								12	14			29					
=													29				
INPUT								12	16								
PRINT								12	17								
LENGTH											26						
SUBSTR											26						
ASC											26						
CHR											26						
;																	
IF								12	18								
THEN																	
ELSE								13		23							
ELSEIF								13		24							
DO								12	19								
LOOP								13									
RETURN								12	20								
int											26						
double											26						
string											26						
INTEGER														31			
DOUBLE														32			
STRING														33			
(											26						
)					8	10											
,						9											37
+=																	38
*=																	39
/=																	40
\=																	41