Dokumentace k projektu

Interpret jazyka IFJ15

Tým 044, Varianta b/2/II

Rozšíření SIMPLE

25. listopadu 2015

Dušan Valecký (vedoucí)	xvalec00	xx%
Jakub Vitásek	xvitas02	xx%
Juraj Vida	xvidaj00	xx%
Jaroslav Vystavěl	xvysta02	xx%
Marek Vyroubal	xvyrou05	00%

Obsah

1 Úvod

Tato dokumentace popisuje implementaci interpretu imperativního jazyka IFJ15, který je zjednodušenou podmnožinou jazyka C++11. Pro přehlednost je dokumentace rozdělena do kapitol a podčástí, jejichž obsah odpovídá kapitole.

2 Struktura projektu

2.1 Lexikální analyzátor

...

2.2 Syntaktický analyzátor

Pro syntaktickou kontrolu byla zvolena metoda rekurzivního sestupu. Aby byla implementace co nejvíce bezproblémová, bylo podstatné převést prvotní návrh gramatiky na LL gramatiku, jejíž pravidla byla využita k zachycení syntaktických chyb. K jejich identifikaci byla vytvořena enumerace, do níž každá chybová hláška vrací index. Podle něj se poté formátuje výpis chyby a samozřejmě i návratová hodnota celého interpretu.

Sémantická kontrola v rámci rekurzivního sestupe je nicméně implementačně zajímavější, než samotná syntaktická analýza. Pro ukládání informací o proměnných, funkcích a formálních parametrech funkcí byla využita tabulka s rozptýlenými položkami. Proměnné, na které se během rekurzivního průchodu narazí, se ukládají primárně do globální tabulky symbolů `commTable`. Tato tabulka obsahuje několik důležitých položek, které jsou podstatné pro sémantickou kontrolu. Je nimi zejména `varType`, do níž se ukládá typ načtený v pravidle `type()`, a `scope`, jež obsahuje momentální rámec definice/deklarace proměnné.

Rámce jsou další velice podstatnou součástí sémantické analýzy. Pro každý blok, který definuje pravidlo COMM_SEQ, je inkrementována celočíselná globální proměnná `currScope`. Na konci tohoto pravidla je opět dekrementována, obsahuje tudíž vždy aktuální rámec. Při kontrole redefinice/redeklarace proměnné pak lze tohoto rámce využít. Pro zjednodušení vyhledávání položek v tabulce symbolů podle rámce byla vytvořena funkce `htReadScope` s parametrem navíc, specifikujícím hledaný rámec.

Poslední důležitou částí sémantické kontroly je kontrola, zda při volání definované funkce byl vložen správný počet parametrů a správný datový typ pro každý z nich. Kontroly počtu parametrů bylo docíleno pomocí využití synonym v tabulce symbolů, kdy pro každý parametr funkce byla vytvořena v tabulce `paraTable` položka s klíčem (neboli pomyslnou maskou), s tím, že každá položka obsahuje typ parametru a jeho pořadí. Poté už při volání funkce stačí jen každý vložený parametr porovnat záznam v tabulce `paraTable`, jenž obsahuje pořadí souhlasící s momentálním pořadím vložených parametrů. Opět byla pro zjednodušení vyhle-

dávání podle pořadí implementována nová vyhledávací funkce `htReadOrder`, jež navíc přijímá vyhledávané pořadí položky.

2.3 Precedenční analýza výrazů

...

2.4 Interpret

...

3 Řešení vybraných algoritmů a datových struktur

3.1 Boyer-Mooreův algoritmus

...

3.2 Heap Sort

...

3.3 Tabulka s rozptýlenými položkami

...

4 Práce v týmu

4.1 Příprava a plán vývoje

Zezačátku vývoje byla každému členovi týmu určena jeho role a přidělen úkol. Při rozhodování, komu jakou část přidělit, bylo vzato v potaz základní rozložení složitosti a také schopnosti daného jedince. Časové rámce pro každou část byly stanoveny pouze velice abstraktně a spoléhalo se spíše na svědomitý přístup každého člena týmu.

4.2 Verzovací systém

Pro jednoduchou a příjemnou kontrolu verzí a možnost rollbacku v případě nalezení chyby byla využita technologie Git s frontendem na stránce Bitbucket, jelikož oproti Githubu nabízí nezpoplatněný privátní repozitář. Během vývoje byla využita možnost vytvoření větve (branch), aby mohl probíhat souvislý vývoj s různým kódem, a poté spojení těchto dvou částí do jedné. Pro ověřování důvodu výskytu chyb byl hojně využíván i příkaz checkout, který

dočasně načte soubory z některého ze starších commitů. To nám dalo možnost se takřka "vrátit zpět v čase" a zkompilovat projekt se starším kódem, nad kterým jsme pak spustili testovací sadu a zjistili, co dříve fungovalo. Další a poslední výhodou je i prokazatelnost odvedené práce, díky níž můžeme jednoznačně dokázat, kdo odvedl jakou práci, případně i její množství a významnost.

4.3 Rozdělení práce v týmu

Implementaci projektu započal kolega Vida, a to naprogramováním lexikálního analyzátoru. Dále se podílel na vytvoření prvotní kostry projektu a gramatiky, návrhu instrukční sady, generování instrukcí a také rozsáhlou pomocí při debuggingu.

Kolega Valecký přispěl tvorbou gramatiky, LL tabulky a pomocí se začátkem rekurzivního sestupu, jeho hlavní doménou však byla precedenční syntaktická analýza (dále pouze PSA). Dále byl jmenován vedoucím týmu, takže většina organizačních záležitostí zůstala na něm.

Jakub Vitásek dostal na starost rekurzivní sestup (dále pouze RS) a potažmo veškerou syntaktickou analýzu v rámci RS, celou sémantickou analýzu s využitím tabulky symbolů, programování vestavěných funkcí jazyka IFJ15 a dalé testovací skripty, založení a zaučení do verzovacího systému, pomoc s řízením týmu a obecný debugging.

4.3 Rozdělení bodů v týmu

Zanedbání časného započetí práce se stalo osudovým kolegovi Vyroubalovi, který za celou dobu implementace poskytl jen zanedbatelné množství využitelného kódu a na vývoji či konzultacích se takřka vůbec nepodílel. Tím se celý tým dostal do časového skluzu. Proto jsme byli nuceni změnit rozložení bodů.

5 Závěr

LL gramatika

1) <PROGRAM> -> <FUNC_N>
2) <FUNC_N> -> <FUNC> <FUNC_N>
3) <FUNC_N> -> E
4) <VAR_DEF> -> <TYPE> id <INIT>;
5) <VAR_DEF> -> auto id <INIT>;
6) <INIT> -> = <EXPR>
7) <INIT> -> E
8) <TYPE> -> int
9) <TYPE> -> double
10) <TYPE> -> string
11) <FUNC> -> <TYPE> id <PAR_DEF_LIST> <DEC_OR_DEF>
12) <DEC_OR_DEF> -> <COMM_SEQ>
13) <DEC_OR_DEF> -> ;
14) <PAR_DEF_LIST> -> (<PARAMS>)
15) <PARAMS> -> <TYPE> id <PARAMS_N>
16) <PARAMS> -> E
17) <PARAMS_N> -> , <TYPE> id <PARAMS_N>
18) <PARAMS_N> -> E
19) <COMM_SEQ> -> { <STMT_LIST> }
20) <STMT_LIST> -> <STMT> <STMT_LIST>
21) <STMT_LIST> -> E
22) <STMT> -> if(<EXPR>) <COMM_SEQ> <IF_N>
23) <STMT> -> for(<VAR_DEF> <EXPR>; <ASSIGN>) <COMM_SEQ>
24) <STMT> -> <COMM_SEQ>
25) <STMT> -> <VAR_DEF>
26) <STMT> -> cin >> id <CIN_ID_N>;
27) <STMT> -> cout << <COUT_TERM>;
28) <STMT> -> <RETURN>
29) <STMT> -> id <CALL_ASSIGN>
30) <CALL_ASSIGN> -> = <EXPR>;
31) <CALL_ASSIGN> -> (<terms>);
32) <TERMS> -> id <TERMS_N>
33) <TERMS> -> E
34) <TERMS_N> -> , id <TERMS_N>
35) <TERMS_N> -> E
36) <ASSIGN> -> id = <EXPR>
37) <CIN_ID_N> -> >> id <CIN_ID_N>
38) <CIN_ID_N> -> E
39) <COUT_TERM> -> id <COUT_TERM_N>
40) <COUT_TERM_N> -> << <COUT_TERM>
41) <COUT_TERM_N> -> E
42) <RETURN> -> return <EXPR>;
43) <IF_N> -> else <COMM_SEQ>
44) <IF_N> -> E

Precedenční tabulka

	+	-	*	/	==	!=	<=	>=	>	<	string	double	int	id	f	(	)	,	$
+	>	>	<	<	>	>	>	>	>	>		<	<	<	<	<	>	>	>
-	>	>	<	<	>	>	>	>	>	>		<	<	<	<	<	>	>	>
*	>	>	>	>	>	>	>	>	>	>		<	<	<	<	<	>	>	>
/	>	>	>	>	>	>	>	>	>	>		<	<	<	<	<	>	>	>
==	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
!=	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
<=	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
>=	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
>	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
<	<	<	<	<	>	>	>	>	>	>	<	<	<	<	<	<	>	>	>
string					>	>	>	>	>	>							>	>	>
double	>	>	>	>	>	>	>	>	>	>							>	>	>
int	>	>	>	>	>	>	>	>	>	>							>	>	>
id	>	>	>	>	>	>	>	>	>	>							>	>	>
f																=			
(	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	=	
)	>	>	>	>	>	>	>	>	>	>							>	>	>
,	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	=	
$	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<