Dokumentácia k projektu IFJ / IAL
Implementácia prekladača imperatívneho jazyka IFJ20

Tým 102, varianta II

9. prosince 2020

Autori:

Jacola Patrik	xjacol00	5%	(vedúci)
Sapák Filp	xsapak05	45%	
Gergel Marek	xgerge01	50%	
Bartuš Dávid	xbartu10	0%	

Obsah

Úvod

Úlohou projektu je návrh a implementácia prekladača jazyku IFJ20, ktorý je zjednodušenou podmnožinou jazyka Go. Implementácia je realizovaná pomocou jazyka C.

Program je rozdelený na 3 časti:

- lexikálna analýza (v súbore "scanner.c")
- syntaktická analýza so sémantickou analýzou (v súbore "parser.c")
- generovanie cieľového kódu

Vybrali sme si variantu II., v ktorej bolo zadanie vytvoriť tabuľku symbolov pomocou tabuľky s rozptýlenými položkami.

1 Lexikálna analýza

Na implementáciu lexikálnej analýzy bol použitý deterministický konečný automat (príloha Konečný automat lexikálneho analyzátoru), ktorého kontrola prechodov a kontrola kľúčových slov je zapísaná v súbore scanner.h. Kompletná implementácia sa nachádza v súbore scanner.c v jedinej definovanej funkcii, ktorá pozostáva z čítania štandardného vstupu , na ktorom by mal byť pridelený zdrojový kód vo formáte jazyku IFJ20. Analyzátor číta znak po znaku a vyhodnocuje najbližší možný token. Tokeny sa ukladajú do zoznamu List, ktorý má v súbore tokenlist.h zadefinované okrem štruktúry aj niekoľko makier na prácu so zoznamom.

2 Syntaktická analýza

Syntaktická analýza je implementovaná v súbore parser.c, ktorá prechádza zoznam tokenov vytvorených v Lexikálnej analýze a kontroluje ich pomocou LL-gramatiky a metódou rekurzívneho zostupu podľa pravidiel v LL - tabuľke. Každé pravidlo má vlastnú funkciu, ktorá pracuje s listom tokenov, prípadne s tabuľkou symbolov. Sémantická analýza prebieha zároveň so syntaktickou analýzou. Pri sémantickej analýze sa používa tabuľka symbolov, kde existuje tabuľka pre definície funkcií a tabuľka pre lokálne premenné. Tabuľka pre definície funkcií je vyplnená pri prvom prechode, pri prechode druhom sa vypĺňa tabuľka pre lokálne premenné. Tabuľka symbolov je implementovaná za pomoci tabuľky s rozptýlenými položkami.

2.1 LL – gramatika

<prog> -> package id EOL <exec>
<exec> -> <func> <func_n>
<func> -> func id (<params>)<func_types><body>
<func_types> -> (<type> <types_n>)
<func_types> -> eps
<types_n> -> , <type> <types_n>
<types_n> -> eps
<type> -> int
<type> -> float64
<type> -> string
<params> -> id <type> <params_n>
<params> -> eps
<params_n> -> , id <type> <params_n>
<params_n> -> eps
<func_n> -> EOL <func> <func_n>
<func_n> -> eps
<body> -> { <statement> <statement_n> }
<statement> -> EOL <definition>
<statement> -> EOL <assignment>
<statement> -> EOL <if>
<statement> -> EOL <for>
<statement> -> EOL <call>
<statement> -> EOL <return>
<statement> -> EOL eps
<statement_n> -> <statement> <statement_n>
<statement_n> -> eps
<definition> -> id := <expression>
<assignment> -> <ids> = <expressions>
<assignment> -> <ids> = <call>
<ids> -> id <ids_n>
<ids_n> -> , id <ids_n>
<ids_n> -> eps
<expressions> -> <expression> <expression_n>
<expression_n> -> , <expression> <expression_n>
<expression_n> -> eps
<if> -> if <expression> <body> else <body>
<for> -> for <definition>; <expression> ; <assignment> <body>
<for> -> for eps ; <expression> ; eps <body>
<call> -> id(<call_param>)
<call> -> id()
<call_param> -> data_float64 <call_param_n>
<call_param> -> data_int <call_param_n>
<call_param> -> data_string <call_param_n>
<call_param> -> id <call_param_n>
<call_param_n> -> , <call_param>
<call_param_n> -> eps
<return> -> return <expressions>

2.2 LL – tabuľka

Čísla v tabuľke znázorňujú poradie pravidla v LL – gramatike.

	package	id	eol	func	(	)	,	int	float64	string	{
<prog>	**1**										
<exec>				**2**							
<func>				**3**							
<func_n>			**15**								
<params>		**11**				**12**					
<func_types>					**4**						**5**
<body>											**17**
<type>								**8**	**9**	**10**	
<types_n>						**7**	**6**				
<params_n>						**14**	**13**				
<statement>			**18;19;20;21;22;23;24**								
<statement_n>			**25**								
<definition>		**27**									
<assignment>		**28;29**									
<if>											
<for>											
<call>		**37;40**									
<return>											
<expression>											
<ids>		**30**									
<expressions>											
<ids_n>							**31**				
<expression_n>			**35**				**34**				**35**
<call_param>		**44**									
<call_param_n>						**46**	**45**				

	}	=	if	else	for	;	data_float64	data_int	data_string	return	$	:=
<prog>												
<exec>												
<func>												
<func_n>											**16**	
<params>												
<func_types>												
<body>												
<type>												
<types_n>												
<params_n>												
<statement>												
<statement_n>												
<definition>												
<assignment>												
<if>			**36**									
<for>					**37;38**							
<call>												
<return>										**47**		
<expression>												
<ids>												
<expressions>												
<ids_n>		**32**										
<expression_n>	**35**											
<call_param>							**41**	**42**	**43**			
<call_param_n>												

3 Dátové typy

3.1 Jednosmerne viazaný zoznam (Linked list)

Jednosmerne viazaný zoznam s prístupom FIFO sme vytvorili ako výstupnú štruktúru z lexikálnej analýzy, z ktorej sa ďalej vykonáva syntaktická analýza. Nachádza sa v súbre tokenlist.h. Zoznam obsahuje odkaz na prvú a poslednú položku. Jednotlivé položky obsahujú typ tokenu uložený v premennej type a celý text tokenu je uložený v premennej value. Na označenie typu tokenu sme použili enum. Ďalej sa v súbore okrem štruktúry nachádzajú aj makrá sa zjednodušenie práce so zoznamom.

3.2 Tabuľka s rozptýlenými položkami (Hash table)

Štruktúru tabuľky s rozptýlenými položkami Htab sme vytvorili na základe tabuľky vytvorenej v predmete IJC, ktorá má oproti IAL verzii navyše štruktúru HTabIterator, ktorá obsahuje odkaz na položku, odkaz na tabuľku a pozíciu v tabuľke. Dáta v položke majú vlastnú štruktúru ktorá sa dá prideľovať ako celok.

3.3 Reťazec (String)

Štruktúru String sme si vytvorili z dôvodu zjednodušenia úpravy reťazca. Obsahuje niekoľko funkcii , ktoré sa volajú medzi sebou. Najužitočnejšia bola strAppendChar , ktorá na koniec reťazca pridala nový znak. Pred každým volaním iných funkcii je nutné zavolať funkciu strClear. Niektoré funkcie zostali nevyužité.

4 Práca v tíme

Vzájomná komunikácia prebiehala cez Discord, kde sme aj ako tím spojili. Na verzovanie nášho projektu sme použili Git a na hosting repozitáru bol použitý GitHub. Nakoľko nie všetci sme skúsení používatelia verzovacích softvérov, tak sme ti to uľahčili používaním softvéru GitKraken.

Po vytvorení tímu sme sa dohodli na pravidelných online stretnutiach cez víkend, ktoré sa časom stali viac nepravidelné. Niektoré sme nahrádzali počas týždňa poprípade sme si dopĺňali ďalšie , ak sme chceli prediskutovať zmeny v práci na projekte.

Prácu sme si nerozdelili úplne ideálne , pretože sa občasne čakalo na štruktúry, ktoré sa predávali ďalej pre správny beh programu. Na začiatku sme však okamžite mali vytvorený Makefile a Error modul aby sme mohli program spustiť a správne ukončiť. Neskôr sa však už situácia viac komplikovala a moduly na projekte sme odkladali na neskôr, čo zapríčinilo ,že sme projekt nestihli dokončiť. Problémy nastali aj v komunikácii a zdieľaní informácii.

5 Rozdelenie práce

Prácu sme sa pokúsili rozdeliť podľa skúsenosti jednotlivca s problematikami projektu. Pribežne sme svoj pokrok na projekte prediskutovávali a zhodnocovali ďalšie postupy.

Jacola Patrik: Konečný automat, Generator

Sapák Filp: Parser, Dokumetace

Gergel Marek: Makefile, Error, Scanner, Tokenlist, String, Symtable, Dokumentace

Bartuš Dávid : Stack, Expressions

6 Záver

Projekt nebol úspešne celý vyriešený, podarilo sa nám dokončiť len lexikálnu a syntaktickú analýzu a dátové štruktúry, na komunikáciu medzi modulmi. Analýza výrazov a generátor výstupného kódu neboli dokončené. Veľké množstvo času nám zabralo pochopiť funkčnosť jednotlivých modulov. Projekt nám ukázal naše nedostatky pri tímovej práci. Taktiež sme mali možnosť vďaka zadaniu vytvoriť si vlastný, nie funkčný, prekladač, z ktorého skúsenosti nám budú v budúcnosti určite užitočné.

7 Zdroje

[1] Alexander Meduna a Roman Lukáš. Prednášky Formální jazyky a překladače [online] [cit. 2020-12-9]. Dostupné z: http://www.fit.vutbr.cz/study/courses/IFJ/public/materials/

8 Prílohy

8.1 Konečný automat lexikálneho analyzátoru