Dokumentace projektu z IFJ
2018/2019

IFJ18

Tým 119, varianta I

členové týmu:

vedoucí - Daniel Štěpánek, xstepa61 (40%)
David Tříska, xtrisk05 (40%)
Martin Suchánek, xsucha15 (10%)
Ondřej Bravenec, xbrave03 (10%)

Úvod

Naším úkolem bylo vytvoření překladače programovacího jazyka IFJ18, založeného na ruby 2.0.0, do cílového jazyka IFJcode18. Při našem řešení jsem naráželi na spousty problémů především s organizací týmu jako celku, komunikací mezi členy týmu, ale i při celkovém návrhu a rozdělení práce na projektu i následné implementaci. Tuto řadu překážek jsme snažili překonat, ale výsledek ani zdaleka neodpovídal naší představě výsledku, do které jsme věnovali tolik odhodlaní, času a energie.

Rozdělení práce

Daniel Štěpánek - scanner, parser, generování kódu, chybová hlášení
David Tříska - parser, tabulka symbolů, pomocné ADT ,dokumentace
Maťko Suchánek - ADT, dokumentace
Ondřej Bravenec - ADT, dokumentace

Při rozdělování práce jsme se snažili rovnoměrně rozložit povinnosti mezi všechny členy týmu, to ale vždy naráželo na rozdílné úrovně schopností jednotlivých členů týmu. V důsledku toho často nastával stav, kdy přidělený úkol členu týmu musel nakonec vypracovat někdo jiný, tudíž jsme se dostali do velkého časového deficitu.

Řešení projektu

Komunikace v týmu, verzovací systém

V týmu jsme komunikovali přes sociální sítě, především pomocí sociální sítě Facebook, kde také probíhalo pravidelné přeposílání aktuálních verzí projektu nesoucí vždy název sdilena_verze_x.y.z. Změna jednotlivých proměnných(x, y, z) probíhala podle rozsáhlosti provedených změn. Proměnná “y” se zvyšovala, pokud byl do archívu přidán nový soubor a proměnná “z” se zvyšovala, pokud se pouze již obsažené soubory modifikovaly. Verze pro odevzdání nesla číslo 1.11.1. Komunikace neprobíhala jen elektronicky, nýbrž i na pravidelných týmových schůzích.

Lexikální analyzátor (scanner)

Funkce lexikálního analyzátoru je rozdělení vstupního souboru, který je načítán po jednotlivých znacích, na posloupnost Tokenů. Vždy, když je token

rozpoznán, dojde k vytvoření nové instance dat. struktury Token, která je odeslaná parseru, který se stará o její zpracování i následné uvolnění paměti jí využívané. Vzorem implementace lexikálního analyzátoru je konečný automat uvedený v příloze.

Syntaktická a sémantická analýza (parser)

Parser od scanneru požaduje Token, dokud nedostane Token obsahující EOF nebo lexikální error. Parser kontroluje syntaktická pravidla definové LL-gramatikou. Parser nadále kontroluje sémantická pravidla (dodržení dat. typů) a pouze vestavěných funkcí. Vyhodnocení výrazu je definované precedenční tabulkou. Parser také zajišťuje veškerou komunikaci s tabulkou symbolů.

Generování kódu

Generování kódu je řešeno dvěma způsoby. Buďto kód generuje specifická funkce, například generate_strlen(), nebo je kód generován postupným přidáváním částí kódu a následně posílán na standardní výstup.Tato část interpretu není dokončena do konce z důvodu časové tísně.

Popis abstraktních datových typů

Tabulka symbolů - varianta I.

Tabulka je implementována binárním vyhledávacím stromem. Vyhledávání v tabulce probíhá pomocí string identifikátoru položky, ovšem každý uzel má ještě unikátní celočíselný identifikátor pro usnadnění práce s tabulkou symbolů. Tento identyfikator je získán funkcí djb2, která je implementací hashovacího algoritmu djb2. Vytváří se jedna instance - globální tabulka symbolů, a x instancí - lokální tabulky symbolů jednotlivých funkcí.

Token

Token slouží k uchování a reprezentaci stringu načteného scannerem , dále uchovává informaci o jaký typ řetězce se jedná (int, float, string, identifikátor, EOF, EOL). K uložení hodnoty stringu tokenu je pouzit ADT rstring.

rstring

Slouží k ukládání stringů statických předem neznámých velikostí. Je součástí dalších ADT.

dynamic_string

Slouží k ukládání stringů dynamických velikostí.

tokenStack

Použit v parseru, jako zásobník pro konečný zásobníkový automat.

genStack

Zásobník který uchovává pomocné data, pro generování výstupu.

Závěr

Tento projekt byl pro nás dobrá i špatná zkušenost. Poprvé jsme si vyzkoušeli týmovou práci a práci na rozsáhlejším projektu jako takovém. Během implementace jsme si rozšířili vědomosti i procvičili věci již známé.

Při implementaci jsme vycházeli především z přednášek a materiálů předmětů IFJ a IAL. Dále jsme využívali internetové zdroje.

Již výše zmíněné problémy jsou pro nás, i když špatnou, ale přesto cennou zkušeností a v budoucnu se jim budeme snažit jistě předejít.

LL gramatika

1.	<prog>	->	begin <st_list>
2.			
3.	<st_list>	->	<stat> EOL <st_list>
4.	<st_list>	->	end
5.	<stat>	->	input <item>
6.	<stat>	->	print <item> <item_list>
7.	<stat>	->	id = <assign>
8.	<stat>	->	length <item>
9.	<stat>	->	substr <param>
10.	<stat>	->	ord<param>
11.	<stat>	->	chr<param>
12.	<stat>	->	if <expr> then EOL <st_list> else EOL <st_list> end EOL
13.	<stat>	->	while <expr> do EOL <st_list> end EOL
14.	<stat>	->	return <expr>
15.	<stat>	->	def id <param>
16.	<item_list>	->	,<item> <item_list>
17.	<item_list>	->	)
18.	<item>	->	(<item>)
19	.<item>	->	id
20.	<item>	->	int
21.	<item>	->	double
22.	<item>	->	string
23.	<assign>	->	<expr>
24.	<assign>	->	<item>
25.	<expr>	->	(<expr>)
26.	<expr>	->	<item>
27.	<expr>	->	
28.	<param>	->	(<param>)
29.	<param>	->	<item_list>
30.	<param>	->	<param_list>
30.	<type>	->	Integer
31.	<type>	->	Float
32.	<type>	->	String

Precedenční tabulka

E - equal, R - reduce, S - shift, F - failure

+-	*/	\	r	(	)	i	$	
R	S	S	R	S	R	S	R	**+-**
R	R	R	R	S	R	S	R	***/**
R	S	R	R	S	R	S	R	****
S	S	S	F	S	R	S	R	**== <> < <= > >=**
S	S	S	S	S	E	S	F	**(**
R	R	R	R	F	R	F	R	**)**
R	R	R	R	F	R	F	R	**id, int, double, str**
S	S	S	S	S	F	S	F	**$**

Konečný automat pro lexikální analýzu