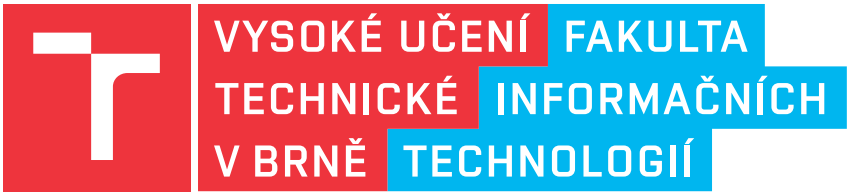


Dokumentace k projektu do IFJ / IAL
Implementace překladače imperativního jazyka IFJ19

11. prosince 2019

Tým 120, varianta II

Autoři:

Březina Jindřich - 25%	xbrezi21@stud.fit.vutbr.cz
Gumančík Pavol - 25%	xguman01@stud.fit.vutbr.cz
Kotáb Dominik - 25%	xkotab01@stud.fit.vutbr.cz
Moravčík Tomáš - vedoucí - 25%	xmorav41@stud.fit.vutbr.cz

Obsah

1 Úvod

Tato dokumentace popisuje návrh a implementaci překladače imperativního jazyka IFJ19, který je zjednodušenou podmnožinou jazyka Python 3. Naše varianta zadání II, měla za úkol implementovat tabulku symbolů, která se nachází v souboru `symtable.c` a `symtable.h` pomocí tabulky s rozptýlenými položkami.

2 Implementace

2.1 Části překladače

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

2.2 Lexikální analyzátor

Lexikální analyzátor, ve zdrojových souborech `scanner.c`, je první část překladače, která pracuje se zdrojovým textem. Čte a rozděluje jednotlivé lexémy na tokeny, se kterými dál pracuje syntaktický analyzátor. Lexikální analyzátor jsme implementovali dle našeho konečného automatu, zpracovává znak po znaku ze vstupu a na základě stavu, ve kterém skončil určí typ a případně hodnotu tokenu. Pokud při zpracovávání dostane lexikální analyzátor znak, se kterým konečný automat nepočítá, mohou nastat dvě věci:

- ukončí se tvorba aktuálního tokenu podle platného konečného stavu automatu a načtený znak se odloží zpátky na vstup
- tvorba tokenu neskončila v platném konečném stavu automatu, lexikální analyzátor toto rozezná jako lexikální chybu a pošle příkaz k ukončení překladu (error code 1)

2.3 Syntaktický analyzátor

Syntaktický analyzátor, ve zdrojových souborech `parser.c`, je hlavní částí překladače a volá všechny ostatní části překladače imperativního jazyka. Syntaktická analýza je řešena metodou rekurzivního sestupu za pomoci LL tabulky. Pro zpracování výrazů jsme využili syntaktickou analýzu shora dolu. Výrazy jsou zpravované pomocí precedenční syntaktické analýzy.

2.4 Sémantický analyzátor

Sémantický analyzátor je pod částí syntaktického analyzátoru. Sémantický analyzátor vyhodnocuje, zda funkce či proměnná nebyla již dříve na stejné úrovni definovaná. Funkce mohou být definované jen v globální tabulce, kdežto proměnné se mohou vyskytovat jak v globální, tak i v lokální tabulce.

2.5 Generátor kódu

Instrukce pro generátor kódu, ve zdrojových souborech `generator.c`, jsou vytvářeny průběžně za běhu samotného programu a jsou ukládány do listu, ze kterého potom generátor čerpá. Generátor je volán pomocí jednotlivých tříadresných instrukcí, který generuje cílový jazyk IFJcode19.

3 Algoritmy

3.1 Tabulka symbolů

Tabulka symbolů, ve zdrojových souborech `symtable.c`, slouží k uchování názvů definovaných funkcí a proměnných. Pro nalezení definované položky v řídkém seznamu slouží tzv. hash, který nám řekne, kde se v tabulce tato položka podle názvu nachází. U proměnné můžeme najít její přiřazenou hodnotu a u funkce její jednotlivé argumenty nebo definované proměnné uvnitř této funkce.

3.2 Zásobníky

V naší implementaci používáme dva různé zásobníky, ve zdrojových souborech `stack.c` a `tokenStack.c`. `stack.c` v sobě uchovává hodnoty indentů. Zásobník naplňuje lexikální analyzátor a správnost indentů následně kontroluje syntaktický analyzátor. `tokenStack.c` slouží syntaktické analýze výrazů. Analyzátor výrazů ukládá tokeny na zásobník tokenů a porovnává je se vstupními tokeny.

3.3 Precedenční syntaktický algoritmus

Precedenční syntaktický algoritmus je sice nejméně efektivní, ale využili jsme ho kvůli jednoduchosti implementace, práce s ním a taky kvůli dřívějšímu obeznámení na přednáškách IFJ.

4 Verzovací systém

Pro jednoduchou kontrolu verzí a možnost návratu při případné chybě jsme používali technologii Git s repositářem umístěným na stránce Github. Díky tomuto projektu jsme se konečně všichni naučili správně využívat textové editory propojenými se vzdáleným repositářem, jelikož doposud jsme nebyly donuceni se s tímto seznámit.

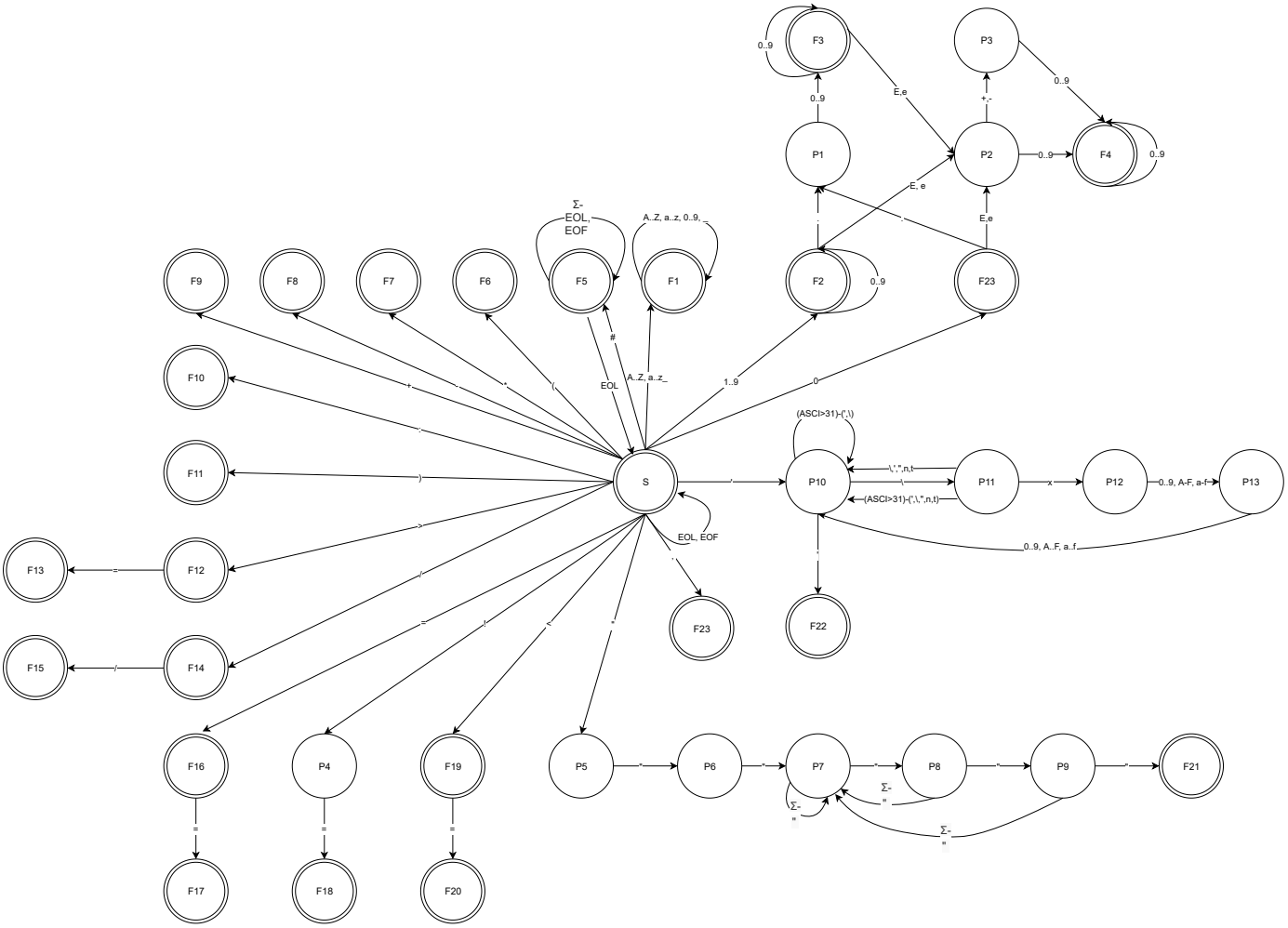
5 Komunikace

S komunikací jsme neměli žádné problémy, samotné programování jsme absolvovali v pokoji našeho velitele, kde jsme vzorně programovali a pokud jsme měli nějaké problémy, mohli jsme se jednoduše svěřit jednomu ze svých kolegů. Vysvětlení našeho problému většinou stačilo nám samotným k tomu, aby jsme problém identifikovali a nakonec vyřešili. Když jsme neměli po boku jeden druhého, spojoval nás komunikační kanál discord, kde jsme využívali jak písemný, tak voice chat.

6 Závěr

Fungování překladače imperativního jazyka by mělo vyhovovat požadovanému zadání bez rozšíření. Dříve jsme zvažovali, že základní program rozšíříme o nějaké to ze snadnějších rozšíření, bohužel nakonec jsme byly rádi, že jsme zvládli samotný základní program.

7 Diagram konečného automatu

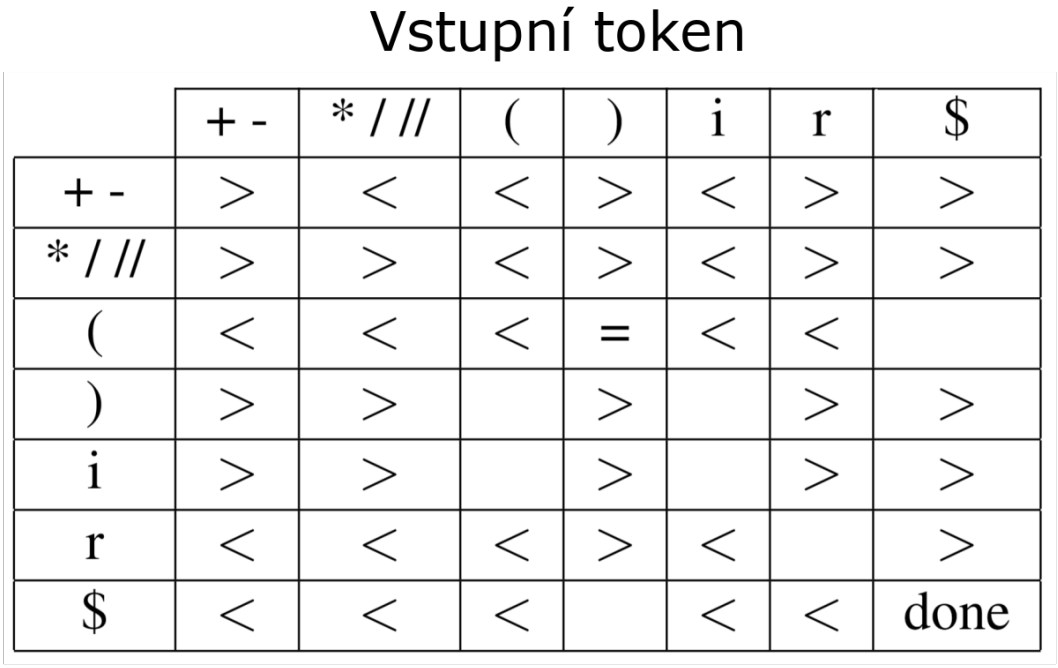


Obrázek 1: Diagram konečného automatu

7.1 Legenda diagramu konečného automatu

Stav	Popis
P1	integer point
P2	exponent
P3	exponent with sign
P4	not equal
P5	documentation string
P6	documentation string
P7	documentation string
P8	documentation string
P9	documentation string
P10	string literal
P11	string literal with (maybe) escape sequence
P12	string literal with escape sequence with x
P13	string literal with escape sequence with x and 1 hexadecimal number
F1	identificator
F2	integer
F3	double
F4	number with exponent
F5	line comment
F6	left parentheses
F7	multiply
F8	minus
F9	plus
F10	colon
F11	right parentheses
F12	greater
F13	greater or equal
F14	divide float
F15	divide int
F16	equals
F17	compare
F18	completed not equal
F19	less
F20	less or equal
F21	completed documentation string
F22	completed string literal
F23	coma

8 Precedenční tabulka



Obrázek 2: Precedenc table

8.1 Legenda Precedenční tabulky

- r = Relační operátory
- i = Datové typy (int, float, string)

9 LL gramatika

S → DEFFUNC S
S → STATEMENT S
S → eoftoken
DEFFUNC → def str leftbracket DEFPARAMS colon eol indent ENDOFDEFFUNC
ENDOFDEFFUNC → STATEMENT END
ENDOFDEFFUNC → return expression THIRDEND
END → dedent
END → eoftoken
SECONDEND → eol
SECONDEND →eoftoken
THIRDEND → eol dedent
THIRDEND → eoftoken
DEFPARAMS → rightbracket
DEFPARAMS → str DEFPARAMSN
DEFPARAMSN → rightbracket
DEFPARAMSN → comma str DEFPARAMSN
STATEMENT → while expression colon eol indent STATEMENT STATEMENTS END
STATEMENT → if expression colon eol indent STATEMENT STATEMENTS dedent else colon eol indent
STATEMENT → STATEMENTS END
STATEMENT → pass SECONDEND
STATEMENT → str STRINGTHINGIES
STATEMENT → expression
STATEMENT → eol
STRINGTHINGIES → leftbracket CALLPARAMS
STRINGTHINGIES → assign expression SECONDEND
CALLPARAMS → rightbracket
CALLPARAMS → str CALLPARAMSN
CALLPARAMS → float CALLPARAMSN
CALLPARAMS → int CALLPARAMSN
CALLPARAMS → doccom CALLPARAMSN
CALLPARAMS → literal CALLPARAMSN
CALLPARAMSN → rightbracket
CALLPARAMSN → comma AFTERCOMMA CALLPARAMSN
AFTERCOMMA → str
AFTERCOMMA → float
AFTERCOMMA → int
AFTERCOMMA → literal
AFTERCOMMA → doccom
STATEMENTS → eps
STATEMENTS → STATEMENT STATEMENTS
S → eps

- LL gramatika je využívaná při řízení syntaktické analýzy

10 Tabulka LL gramatiky

	eoftoken	def	str	leftbracket	colon	eol	indent	return	expression	dedent	rightbracket	comma	while	if	else	pass	asign	float	int	doccom	literal	$
S	3	1	2			2			2				2	2		2						40
DEFFUNC		4																				
STATEMENT			20			39			21				17	18		19						
DEFPARAMS			14								13											
ENDOFDEFFUNC			5			5		6	5				5	5		5						
END	8									7												
THIRDEND	12					11																
SECONDEND	10					9																
DEFPARAMSN											15	16										
STATEMENTS	37		38			38			38	37			38	38		38						
STRINGTHINGIES				22													23					
CALLPARAMS			25								24							26	27	28	29	
CALLPARAMSN											30	31										
AFTERCOMMA			32															33	34	36	35	

Obrázek 3: Vygenerovaná tabulka LL gramatiky