IFJ2022

Projektová dokumentácia do predmetov IFJ a IAL

Tým xjokay00, varianta BVS

7.12.2022

Dávid Jókay, xjokay00: 25%
Peter Kováč, xkovac66: 25%
Marek Buch, xbuchm02: 25%
Matúš Gazdík, xgazdi04: 25%

Obsah

Úvod

V projekte IFJ2022 do predmetov IFJ a IAL bolo úlohou implementovať prekladač jazyka IFJ22 do IFJcode22. Program načíta zdrojový kód z STDIN, vykoná kontrolu správnosti programu a program preloží do cieľového jazyka.

1 Práca v tíme

1.1 Rozdelenie

Rozdelenie práce na projekte bolo rovnomerné a každý člen tímu sa podieľal rovnakým dielom, a preto má každý percentuálne hodnotenie 25%. Konkrétnejšie rozdelenie zachytáva tabuľka 1.

Meno	Priradená práca
Marek Buch	Lexikálna analýza, Syntaktická analýza, Tabuľka symbolov, Sémantická analýza, LL tabuľka, Dokumentácia
Matúš Gazdík	Lexikálna analýza, Syntaktická analýza, Syntaktická analýza výrazov, LL tabuľka
Dávid Jókay	Organizácia, Lexikálna analýza, Tabuľka symbolov, Sémantická analýza, Dokumentácia
Peter Kováč	Lexikálna analýza, Syntaktická analýza, Syntaktická analýza výrazov, Testovanie, Refaktorizácia

Tabuľka 1

1.2 Spôsob práce

Hoci sme na projekte začali pracovať relatívne skoro, t.j. začiatkom októbra, zvolili sme nedostatočné tempo a tým sme sa dostali do časového sklzu. Zo začiatku sme sa stretávali na týždennej báze a pracovali sme na projekte spoločne. Neskôr sme zintenzívnili náš prístup a pracovali na projekte denne, najčastejšie vo dvojiciach. Prácu sme si delili postupne, podľa aktuálnej situácie.

1.3 Komunikácia

Komunikácia medzi členmi tímu prebiehala zväčša osobne, prípadne pomocou aplikácie Discord. Komunikácia prebiehala v poriadku, každý člen tímu aktívne komunikoval a nemali sme problémy pri plánovaní spoločných stretnutí.

1.4 Verzovací systém

Pre správu súborov a zdrojového kódu sme používali systém GIT. Ako vzdialený repozitár sme používali GitHub. Využívali sme aj funkciu LiveShare v editore Visual Studio Code. Funkcia LiveShare nám umožnila paralelne pracovať na rovnakej časti projektu v prípade, že osobné stretnutie nebolo možné.

2. Lexikálna analýza

Pri implementácii prekladaču sme začali tvorbou lexikálnej analýzy. Využili sme pri tom znalosti konečných automatov.

2.1 Automat

Automat, použitý pri tvorbe lexikálnej analýzy je implementovaný vo funkcii **`dka()`**, v súbore **`tokenizer.c`**, v ktorom sa celá lexikálna analýza odohráva. Implementovaný automat je zobrazený v tabuľke 2.

Funkcia **`dka()`**je deterministický konečný automat implementovaný pomocou opakujúceho sa **`switch`** statementu, kde každý **`CASE`** značí jeden stav automatu. Prechody medzi jednotlivými stavmi automatu sú dané prečítaným znakom z **`stdin`**. Ak načítaný znak nesúhlasí zo žiadnym znakom, ktorý aktuálny stav povoľuje, je vrátená chyba. Ak je načítaná sekvencia znakov, ktorú jazyk IFJ2022 povoľuje, je vytvorený a uložený nový token.

2.2 Token

Abstraktný dátový typ **`token_t`** obsahuje atribúty **`token_type`** a **`value`**. **`token_type`** môže nadobudnúť hodnoty {`TOK_ID, TOK_ID_FUNCTION, TOK_KEYWORD, TOK_SEPARATOR, TOK_OPERATOR, TOK_LIT, TOK_EPILOG, TOK_SPECIAL`} a **`value`** je skutočný načítaný reťazec. Token je uložený do abstraktného dátového typu **`token_storage_t`**. **`token_storage_t`** má ako atribúty dynamicky alokované pole **`tokens`** (kde sú tokeny uložené) , **`num_tokens`** a **`array_len`**.

Komentáre a prázdne znaky sú tokenizerom ignorované a nie sú z nich vygenerované tokeny.

Súčasťou lexikálnej analýzy sú aj neskôr využívané funckie **`get_token()`** a **`get_token_keep()`**, kde **`get_token()`** vracia aktuálny token a posúva ukazateľ na ďalší a **`get_token_keep()`** vracia aktuálny token.

3. Syntaktická analýza

Syntaktická analýza je najkomplikovanejšou časťou projektu a je umiestnená v súbore **`parser.c`** a **`expression.c`** . Syntaktická analýza je riadená LL-gramatikou, bližšie špecifikovanou v tabuľke 3. Použili sme metódu rekurzívneho zostupu, pričom výrazy sú spracovávané separátne v **`expression.c`**

Každé pravidlo LL gramatiky má svoju vlastnú funkciu, ktoré su postupne volané v **`parser.c`**. **`parser.c`** dostáva tokeny pomocou funkcií **`get_token()`** a **`get_token_keep()`** , kde ich postupne volajú jednotlivé funkcie pravidiel.

4. Sémantická analýza

Sémantická analýza prebieha v **`parser.c`** za pomoci **`symtable.c`** a **`expression.c`**, kde je uložená tabuľka symbolov.

4.1 Tabuľka symbolov

Tabuľku symbolov sme implementovali ako binárny vyhľadávací strom, kde každý uzol značí unikátnu premennú alebo funkciu. Uzly sú v strome ukladané na základe ich abecednej hodnoty.
V prípade, že sa ich abecedná hodnota rovná, je nový uzol uložený ako pravý potomok posledného uzlu s rovnakým názvom.

V prípade ukladania funkcie, má uzol nastavené atributy **`name`**, **`return_type`**, **`arguments`** a **`num_arguments`** (a samozrejme **`left`** a **`right`**).

Pri ukladaní id premennej jej sú nastavené **`name`**, **`scope`** a **`datatype`** (**`left`** a **`right`**).

4.2 Spracovanie výrazov

Spracovanie výrazov prebieha pomocou precedenčnej analýzy v súbore **`expression.c`** S využitím abstraktnej dátovej štruktúry typu viazaný zoznam sme implementovali zásobík, ktorý sme využili pri spracovávaní výrazov.

Pri rozhodovaní o asociativite sme používali vopred definovanú precedenčnú tabuľku, zobrazenú v tabuľke 3.

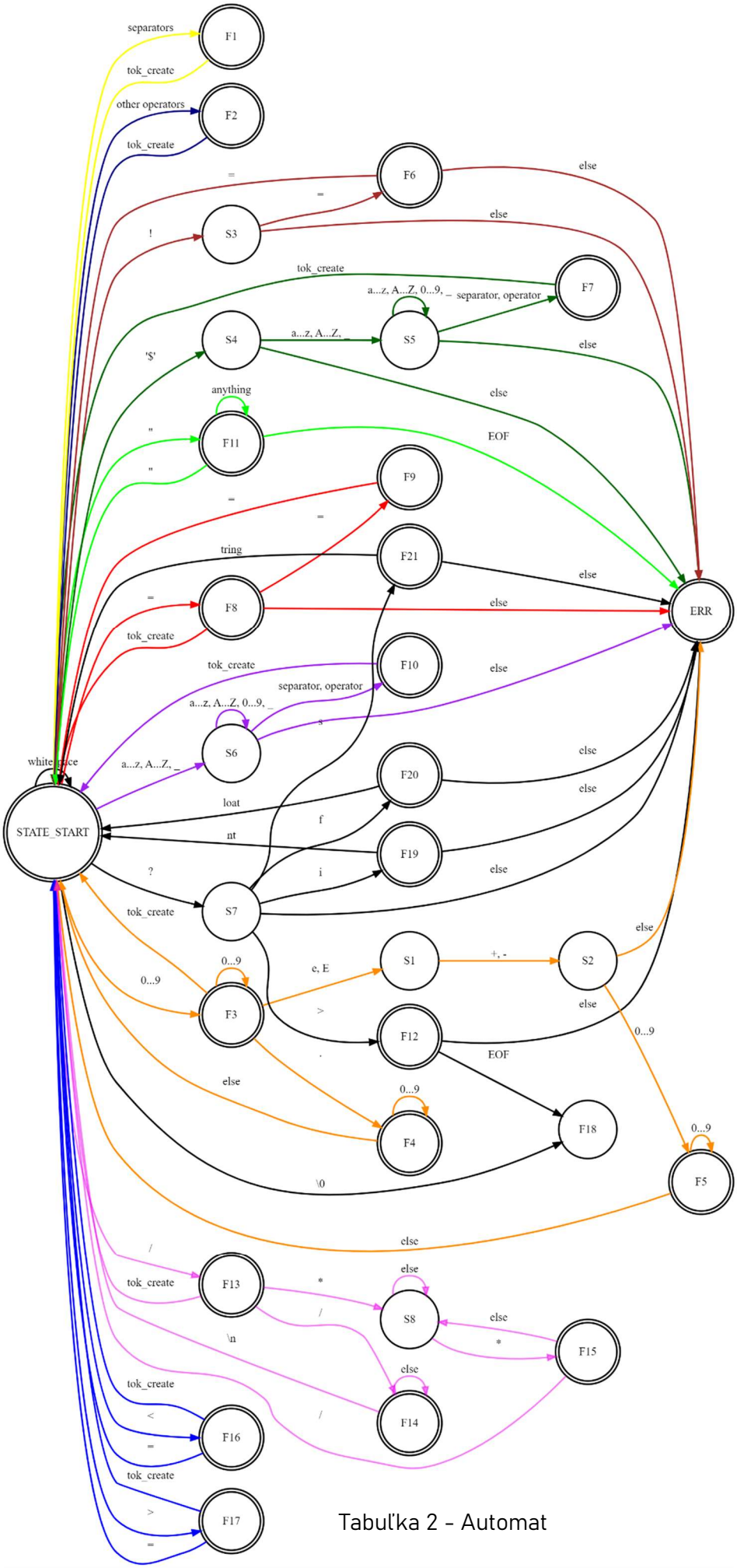
5. Makefile

Súčasťou repozitára je aj súbor Makefile, ktorý príkazom **`make`** vytvorí spustiteľný súbor **`ifj22.`** Príkaz **`make clean`** odstráni všetky objekotvé súbory a spustiteľný binárny súbor.

Záver

V úvode riešenia projektu sme pochybili pri plánovaní. Dostali sme sa teda do situácie kedy sme museli riešiť projekt pomerne rýchlo. Vďaka prednáškam z predmetov IFJ a IAL sme už poznali rôzne koncepty a algoritmy, ktoré sme pri implementácii prekladaču využili.

Bol to pre nás prvý projekt podobných rozmerov a zložitosti, pri ktorom sme sa naučili pracovať v tíme, riešiť zložité problémy pod tlakom a získali sme lepšie zručnosti pri programovaní.



F1	STATE_SEP
F2	STATE_OP
F3	STATE_LIT_NUM
F4	STATE_LIT_NUM_FLOAT
F5	STATE_LIT_NUM_FLOAT_E_3
F6	STATE_NOT_2
F7	STATE_ID
F8	STATE_EQUAL_1
F9	STATE_EQUAL_2
F10	STATE_KEYWORD
F11	STATE_LIT_STR
F12	STATE_EPLIOG_2
F13	STATE_COMMENT_START
F14	STATE_COMMENT_SINGLE
F15	STATE_COMMENT_MULTI2
F16	STATE_SMALLER_EQUAL
F17	STATE_GREATER_EQUAL
F18	STATE_FINISHED
F19	STATE_QUESTION_INT
F20	STATE_QUESTION_FLOAT
F21	STATE_QUESTION_STRING
S1	STATE_LIT_NUM_FLOAT_E_1
S2	STATE_LIT_NUM_FLOAT_E_2
S3	STATE_NOT_1
S4	STATE_ID_START
S5	STATE_ID_MAIN
S6	STATE_KEYWORD_MAIN
S7	STATE_EPILOG
S8	STATE_COMMENT_MULTI

Tabuľka 2 - Automat

1	<program>	prolog	<stlist>						
2	<stlist>	EOF							
3	<stlist>	epilog							
4	<stlist>	<st>	<stlist>						
5	<stlist>	<fdef>	<stlist>						
6	<st>	"if"	"("	<expr>	")"	"{"	<fstlist>	<else>	
7	<st>	"while"	"("	<expr>	")"	"{"	<fstlist>		
8	<st>	"return"	<return_cond>						
9	<st>	<idfun>	"("	<funccallarg>					
10	<st>	<id>	"="	<expr>	";"				
11	<fstlist>	"}"							
12	<fstlist>	<st>	<fstlist>						
13	<else>	"eps"							
14	<else>	"else"	"{"	<fstlist>					
15	<return_cond>	";"							
16	<return_cond>	<expr>	";"						
17	<funccallarg>	")"	";"						
18	<funccallarg>	<expr>	<next>						
19	<next>	")"	";"						
20	<next>	","	<expr>	<next>					
21	<fdef>	"function"	<idfun>	"("	<fun_args>	":"	<type>	"{"	<function_body>
22	<fun_args>	<argf>	<more_argf>						
23	<fun_args>	")"							
24	<argf>	<type>	<id>						
25	<more_argf>	")"							
26	<more_argf>	","	<argf>	<more_argf>					
27	<function_body>	<fstlist>	"}"						

Tabuľka 3 – LL gramatika

	*/	"+-."	<>	"==="	(	)	i	$
*/	>	>	>	>	<	>	<	>
"+-."	<	>	>	>	<	>	<	>
<>	<	<	>	>	<	x	<	>
"==="	<	<	<	>	<	x	<	>
(	<	<	<	<	<	eq	<	x
)	>	>	>	>	x	>	x	>
i	>	>	>	>	x	>	x	>
$	<	<	<	<	<	x	<	end

Tabuľka 4 – precedenčná tabuľka

	prolog	EOF	epilog	"if"	"("	<expr>	")"	"{"	"while"	"return"	<idfun>	<id>	"="	";"	"}"	<eps>	"else"	","	"function"	<idfun>	":"	<type>
<program>	1																					
<stlist>		2	3	4					4	4	4	4							5			
<st>				6					7	8	9	10										
<fstlist>				12					12	12	12	12			11							
<else>																13	14					
<return_cond>						16								15								
<funccallarg>						18	17															
<next>							19											20				
<fdef>																			21			
<fun_arg>							23															22
<argf>																						24
<more_argf>							25															
<function_body>				27					27	27	27	27			27							

Tabuľka 5 – LL tabuľka