Vysoké učení technické v Brně
Fakulta informačních technologií

Dokumentácia k projektu IFJ/IAL

Tím xmicham00, varianta X

Martin Michálik (xmicham00) : 00 [Vedúci]

Matúš Magyar (xmagyam00) : 00

Šimon Škoda (xskodas00) : 00

Jaroslav Vrbiniak (xvrbinj00) : 00

14.10.2025 - 03.12.2025

1. Úvod

Táto dokumentácia popisuje implementáciu prekladača jazyka IFJ25 do medzikódu IFJcode25. Projekt je implementovaný v jazyku C a pozostáva z modulov: lexikálna analýza, syntaktická analýza, sémantické kontroly, generovanie kódu a tabuľka symbolov. Začali sme tím že sme si rozdelili úlohy na začiatok a to tak ze sme si spravili tabuľku symbolov, abstraktný syntaktický strom alebo inak ast, stack a parser.

2. Architektúra projektu

Zdrojové súbory v projekte sú rozdelené podľa funkcie:

- **ast.c/h** – abstraktný syntaktický strom
- **builtins.c/h** – spracovanie vstavaných funkcií
- **errors.c/h** – správa chybových kódov
- **expression.c/h** – precedenčná analýza výrazov
- **generator.c/h** – generovanie IFJcode25
- **main.c** – vstupný bod programu
- **parser.c/h** – syntaktická analýza a riadenie prekladu
- **scanner.c/h** – lexikálna analýza
- **symtable.c/h** – tabuľka symbolov (AVL strom)

3. Lexikálna analýza

Modul scanner.c na ktorom sa podieľal Martin Michálik implementuje lexikálny analyzátor pre jazyk IFJ25. Je založený na deterministickom konečnom automate, ktorý postupne číta znaky zo vstupného súboru a transformuje ich na tokeny využívané syntaktickým analyzátorom. Scanner rozpoznáva všetky kľúčové slová, identifikátory, globálne premenné, číselné literály vo viacerých formátoch (celé, desatinné, exponenciálne aj hexadecimálne), reťazce s podporou escape sekvencií a multi-line stringov, komentáre vrátane vnorených blokových komentárov, operátory a interpunkciu.

Funkcie `next_char()`, `putback()` a `skip()` implementujú nízkoúrovňové čítanie znakov. Spracovanie komentárov zabezpečuje rekurzívna funkcia `ignore_multiline_comment()`. Identifikátory a kľúčové slová sú rozlišované pomocou `check_keyword()`. Pri výskyte neplatnej lexémy modul nastaví lexikálnu chybu a vráti `LEX_ERROR`.

4. Precedenčná tabuľka operátorov

	(	)	i	+	-	*
(	<	=	<	<	<	<
)		>		>	>	>
i		>		>	>	>
+	<	>	<	>	>	<
-	<	>	<	>	>	<
*	<	>	<	>	>	>
/	<	>	<	>	>	>
<	<	>	<	<	<	<
<=	<	>	<	<	<	<

5. LL Gramatika

1: <prolog> → import
2: <program> → class
3: <main_func> → static
4: <func_list> → “ } ”
5: <func_list> → static
6: <func> → static
7:
8:
9: <param_list> → “) ”
10: <param_list> → IDENTIFIER
11: <param_list_tail> → “) ”
12: <param_list> → “ , ”
13: <block> → “ { ”
14: <stmt_list> → “ } ”
15: <stmt_list> → IDENTIFIER, if, return, var, while
16: <stmt> → var
17: <stmt> → IDENTIFIER
18: <stmt> → if
19: <stmt> → while
20: <stmt> → return
21:
22:
23: <var_decl> → var
24: <assign> → IDENTIFIER
25:
26: <if_stmt> → if
27: <while_stmt> → while
28: <return_stmt> → return
29: <func_call> → IDENTIFIER
30:
31: <arg_list> → “) ”
32: <arg_list> → “ (”, FLOAT_LITERAL, IDENTIFIER, INT_LITERAL, Null, STRING_LITERAL
33: <arg_list_tail> → “) ”
34: <arg_list_tail> → “ , ”

```
35: <expr> → “ ( ”, FLOAT_LITERAL, IDENTIFIER, INT_LITERAL, Null, STRING_LITERAL
36: <expr_cont> → “ ) ”, EOL, “ , ”
37: <expr_cont> → OP
38: <term> → IDENTIFIER
39: <term> → INT_LITERAL
40: <term> → FLOAT_LITERAL
41: <term> → STRING_LITERAL
42: <term> → Null
43: <term> → “ ( ”
44: <op> → OP_PLUS
45: <op> → OP_MINUS
46: <op> → OP_MULTIPLY
47: <op> → OP_DIVISION
48: <op> → OP_EQUAL
49: <op> → OP_NOT_EQUAL
50: <op> → OP_LESS_THAN
51: <op> → OP_LESS_THAN_EQL
52: <op> → OP_GREATER_THAN
53: <op> → OP_GREATER_THAN_EQL
54: <op> → OP_IS
55: <eols> → EOF, “ } ”
56: <eols> → EOL
```

6. LL Gramatika verzia 2

```
1: Start -> Start: "[whitespace|EOL]"
2: Start -> F1: "+"
3: Start -> F2: "-"
4: Start -> F3: "*"
5: Start -> F4: "/"
6: F4 -> C1: "/"
7: C1 -> C1: "ε - EOL"
8: C1 -> Start: "EOL"
9: F4 -> C2: "*"
10:      C2 -> C2: "ε - *"
11:      C3 -> C2: "ε - *"
12:      C2 -> C3: "*"
13:      C3 -> Start: "/"
14:      Start -> F5: "="
15:      F5 -> F6: "="
16:      Start -> F7: "<"
17:      F7 -> F8: "="
18:      Start -> F9: ">"
19:      F9 -> F10: "="
20:      Start -> Q1: "!"
21:      Q1 -> F11: "="
22:      Start -> F12: "("
```

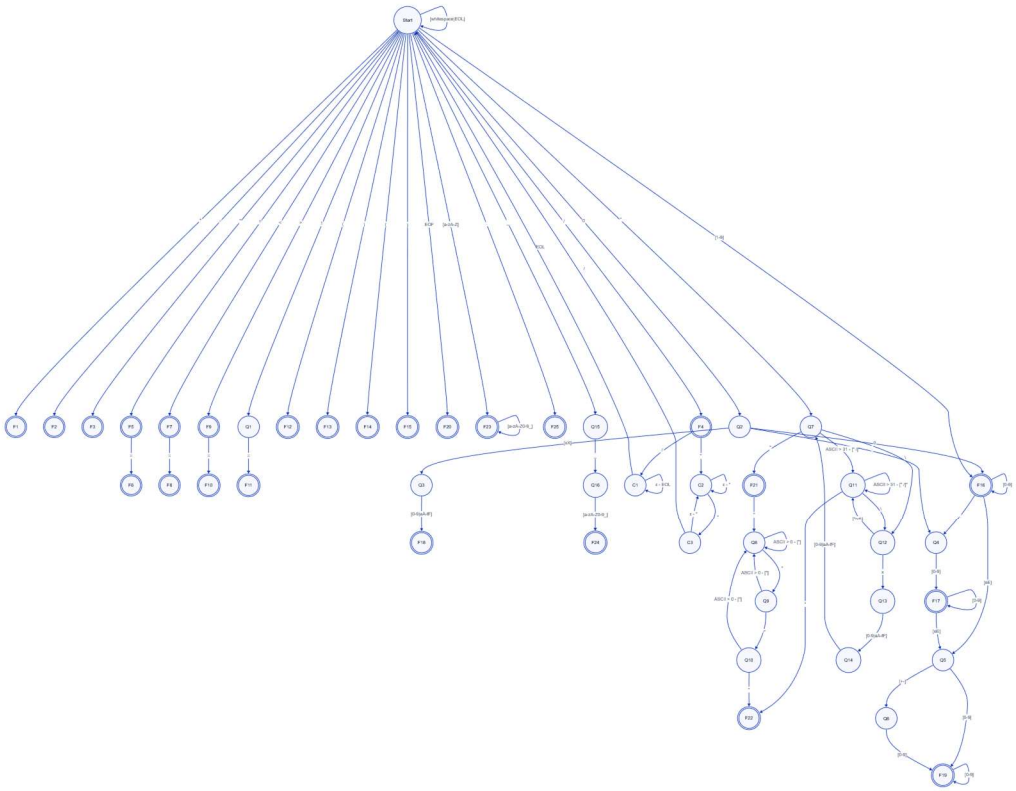
```
23:      Start -> F13: ")"
24:      Start -> F14: "{"
25:      Start -> F15: "}"
26:      Start -> Q2: "0"
27:      Q2 -> F16: "0"
28:      Q2 -> Q3: "[xX]"
29:      Q3 -> F18: "[0-9|aA-fF]"
30:      Start -> F16: "[1-9]"
31:      F16 -> F16: "[0-9]"
32:      F16 -> Q4: "."
33:      Q2 -> Q4: "."
34:      Q4 -> F17: "[0-9]"
35:      F17 -> F17: "[0-9]"
36:      F16 -> Q5: "[eE]"
37:      F17 -> Q5: "[eE]"
38:      Q5 -> Q6: "[+-]"
39:      Q5 -> F19: "[0-9]"
40:      Q6 -> F19: "[0-9]"
41:      F19 -> F19: "[0-9]"
42:      Start -> F20: "EOF"
43:      Start -> Q7: '"'
44:      Q7 -> Q11: 'ASCII > 31 - [" /]"'
45:      Q11 -> Q11: 'ASCII > 31 - [" /]"'
46:      Q11 -> F22: '"'
47:      Q7 -> Q12: '\'
48:      Q12 -> Q11: '["nrt\]'
49:      Q11 -> Q12: '\'
50:      Q7 -> F21: '"'
51:      Q12 -> Q13: 'x'
52:      Q13 -> Q14: '[0-9|aA-fF]'
53:      Q14 -> Q7: '[0-9|aA-fF]'
54:      F21 -> Q8: '"'
55:      Q8 -> Q8: 'ASCII > 0 - ["]'
56:      Q8 -> Q9: '"'
57:      Q9 -> Q8: 'ASCII > 0 - ["]'
58:      Q9 -> Q10: '"'
59:      Q10 -> Q8: 'ASCII > 0 - ["]'
60:      Q10 -> F22: '"'
61:      Start -> Q15: "_"
62:      Q15 -> Q16: "_"
63:      Q16 -> F24: "[a-zA-Z0-9_]"
64:      Start -> F23: "[a-zA-Z]"
65:      F23 -> F23: "[a-zA-Z0-9_]"
```

```
66:       Start -> F25: "."
```

7. LL Tabulka

TBD

8. Diagram konečného automatu



9. Syntaktická analýza

Parser na ktorom sa podielal Martin Michálik je implementovaný v `parser.c` ako kombinácia rekurzívneho zostupu a volania modulu na spracovanie výrazov. Pravidlá jazyka IFJ25 sú implementované ako funkcie:

- `parse_program()`
- `parse_class()`
- `parse_function()`
- `parse_statement()`

Výrazy sú spracované precedenčným analyzátorom v `expression.c`.

10. Chybové hlásenia

Program vracia chybové hlásenia podľa zadania

- `ERR_LEXICAL = 1,` -- Chyba v rámci lexikálnej analýzy
- `ERR_SYNTAX = 2,` -- Chyba v rámci syntaktickej anlýzy
- `ERR_SEM_UNDEF = 3,` -- Nedefinovaná funkcia alebo premenná
- `ERR_SEM_REDEF = 4,` -- Redefinícia funkcie alebo premennej
- `ERR_SEM_PARAMS = 5,` -- Nesprávny počet argumentov alebo typ parametru vstavanej funkcie
- `ERR_SEM_TYPE_COMPAT = 6,` -- Chyba typovej kompatibility vo výrazoch
- `ERR_SEM_OTHER = 10,` -- ostatné chyby
- `ERR_RUNTIME_PARAM_TYPE = 25,` -- Behová chyba – zlý typ parametru vstavanej funkcie
- `ERR_RUNTIME_TYPE_COMPAT = 26,` -- Behová chyba – typová kompatibilita vo výrazoch
- `ERR_INTERNAL = 99,` -- interná chyba prekladača

11. Sémantická analýza

Semantická analíza pozostáva z kontroli refinícií premenných a refinícií funkcií, kontroli existencie symbolov v príslušnom scope, kotroli arity volaných funkcií a zo základnej typovej kontroli literálov. Na semantickej analýze sa podielal každý člen tímu. A súčasťou sémantickej analýzy je aj modul `symtable.c` na ktorom sa podielal Šimon Škoda a Martin Michálik obsahuje implementáciu AVL stromu, ktorý uchováva symboly, vrátane informácií o parametroch funkcií a je jeden z hlavnych koreňov IFJ compileru.

12. Generovanie IFJcode25

Modul `generator.c` na ktorom sa podielal Martin Michálik zodpovedá za generovanie výstupného kódu v jazyku IFJcode25. Obsahuje implementáciu buffrovania výstupu, správu globálnych a lokálnych premenných, tvorbu rámcov funkcií a automatické generovanie unikátnych labelov. Generátor podporuje polymorfné operátory, správu dočasných premenných a typové kontroly podľa špecifikácie jazyka. Implementuje tiež generovanie riadiacich štruktúr (if, else alebo while), volanie funkcií a tvorbu vestavaných funkcií. Výsledkom je funkčný IFJcode25 program so správnou štruktúrou a kontrolou typových chýb.

13. Testovanie

Súčasťou projektu sú testovacie súbory `scanner_test.c` a `symtable_test.c`. Tieto súbory boli použité na testovanie pre pozitívne testy syntakticky správnych programov a negatívne testy lexikálnych, syntaktických a sémantických chýb. Samozrejme toto nebolo jediné testovanie ktoré prebiehalo aj mimo tzchto suborov počas celej doby práce na tomto projekte.

14. Záver

Projekt implementuje funkcionalitu prekladača IFJ25 podľa špecifikácie. Kód je modulárny a testovaný. Tabuľka symbolov implementuje AVL strom pre efektívne vyhľadávanie. Kľúčom tohto projektu bolo rozširiť svoje programovacie schopnosti v jazyku C, priučit sa k práci v tíme, naučiť sa ako funguje compiler a ako si vytvoriť compiler. Projekt je možné rozširiť o ďalšie rôzne variácie funkcionalít.