
Dokumentace ke společnému projektu do IAL a IFJ

# Implementace překladače jazyka IFJ21

Tým 093, varianta 1


| | | |
|---|---|---|
| **Jan Zdeněk** | **(xzdene01)** | 25% |
| Pavel Heřmann | (xherma34) | 25% |
| Alexander Sila | (xsila00) | 25% |
| Maxim Plička | (xplick04) | 25% |

# Obsah

# 1 Úvod

Cílem projektu bylo vytvořit funkční překladač implementovaný v jazyce C. Tento překladač ze vstupního jazyku IFJ21 vytvoří překlad do cílového jazyka IFJcode21.

# 2 Implementace

Způsob implementace a vzájemná kooperace jednotlivých částí je podrobně popsána v následujících kapitolách.

## 2.1 Lexikální analýza

První částí překladače je lexikální analýza, která využívá hlavní funkce `GetToken()`. Tato funkce postupně načítá charaktery ze standartního vstupu a pomocí deterministického konečného automatu vrací jednotlivé tokeny nebo popřípadě chybu lexikální analýzy. Tokeny jsou postupně ukládány do listu tokenů ve funkci `GetTokenList()`. Tento list je následně posílán do `parser.c`, kde probíhá syntaktická a sémantická analýza.

Lexikální analýza dále využívá námi implementovanou knihovnu funkcí pro obousměrně vázaný list v souborech `DLList.c` a `DLList.h`.

Zbytek lexikální analýzy probíha v soubourech `scanner.c` a `scanner.h`.

## 2.2 Syntaktická analýza

Syntaktickou analýzu jsme implementovali v souborech `parser.c` a `parser.h`, pomocí námi vytvořené LL gramatiky. Využili jsme přitom metody rekuzivního sestupu. Pro každé pravidlo jsme implementovali funkci, kde probíhá kontrola daného pravidla. V případě chyby vrací syntaktickou chybu (2).

Pro zpracování výrazů jsme použili precedenční syntaktickou analýzu, která je implementovaná v souborech `expresion.c` a `expresion.h`.

## 2.3 Precedenční syntaktická analýza

Precedenční syntaktická analýza dostane ukazatel na aktivní prvek listu tokenů (potencionální začatek výrazu). Pomocí námi definované precedenční tabulky si získává pravidla pro zpracování výrazu, přičemž využívá pomocnou datovou strukturu (zásobník tokenů), jež je implementována společně s pomocnými funkcemi v `TStack.c` a `TStack.h`. Výsledkem precedenční synstaktické analýzy je zpracovaný výraz, v případě nesprávnosti výrazu vrací patřičnou chybu.

## 2.4 Sémantická analýza

Sémantická analýza probíhá souběžne se syntaktickou analýzou a to za využití tabulky symbolů, která je implementována za pomocí obousměrně vázaného listu struktur, které se nachází v soubourech `symtable.c` a `symtable.h`. Tato struktura reprezentuje všechny již deklarované či definované funkce a proměnné. V průběhu sémantické analýzy jsou tyto struktury definovány a dále probíhá porovnání používaných funkcí a proměnných s tímto listem stuktur.

# 3 Využité datové struktury

## 3.1 Obousměrně vázaný list charakterů

Je používán v lexikální analýze pro dynamickou alokaci hodnoty tokenů. Je implementován v `DLList.c` a `DLList.h`.

## 3.2 Obousměrně vázaný list tokenů

Je používán v syntaktické analýze pro uložení načtených tokenů. Je implementován v `scanner.c` a `scanner.h`.

## 3.3 Obousměrně vázaný list stuktur držící informace o funkcích a proměnných

Je používán v sémantické analýze pro uložení informací o deklarovaných či definovaných funkcích a proměnných. Je implementován v `symtable.c` a `symtable.h`. Rozhodli jsme se pro využití právě obousměrně vázaného listu z důvodu jeho jednoduché implementace a jednoduchosti navigace mezi jednotlivými uzly. Dále je praktické jej využít při mazání rámců a orientaci v nich.

## 3.4 Zásobník tokenů

Je používán v precedenční syntaktické analýze, kvůli nutnosti jeho využití při zpracování výrazů. Je implementován v `TStack.c` a `TStack.h`.

# 4 Práce v týmu

## 4.1 Komunikace

Jako primární komunikační kanál jsme si vybrali aplikaci `Discord` z důvodu jeho jednoduchosti. V komunaci se nenaskytly žádné problémy.

## 4.2 Verzovací nástroje

Jako jediný verzovací nástroj jsme zvolili sdílený repozitář v `Github` z důvodu předchozích kladných zkušeností.

## 4.3 Rozdělení prací

Práci jsme rozdělili rovnoměrně mezi všechny členy týmu. V případě naskytnutí problému při vývoji jsme si navzájem pomáhali.

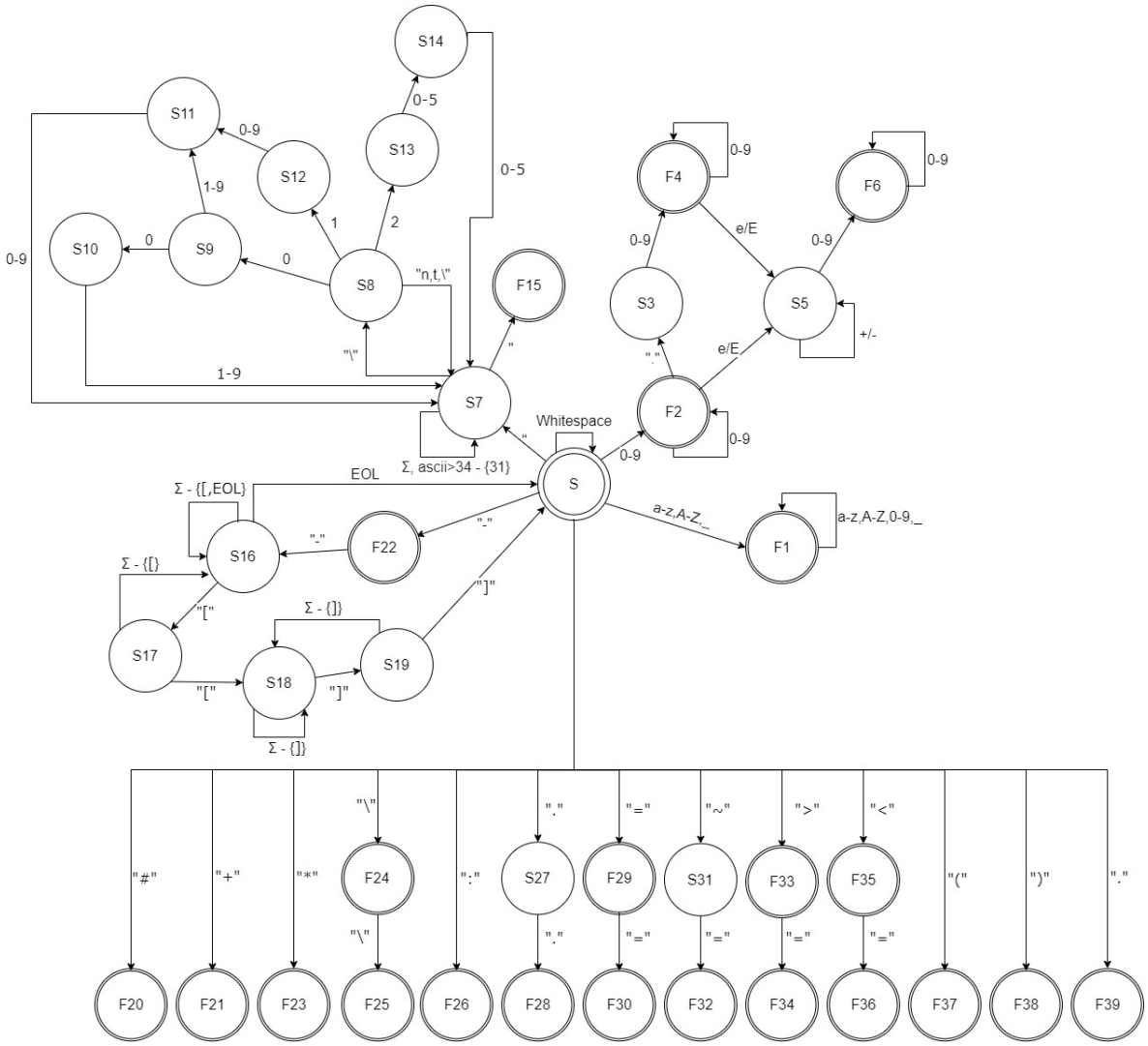
| Jméno | Přidělená práce |
|---|---|
| **Jan Zdeněk** | Vedení týmu, Lexikální analýza, Syntaktická a Sémantická analýza, Hlavní tělo programu |
| Pavel Heřmann | Lexikální analýza, Zpracovávání výrazů |
| Alexander Sila | Syntaktická a Sémantická analýza, Hlavní tělo programu, Testy |
| Maxim Plička | Lexikální analýza, Zpracování výrazů, Dokumentace |

# 5 Závěr

Napříč vývojem se všichni členové shodli, že se s projektem mělo začít dříve vzhledem k jeho komplikovanosti a časové náročnosti. Na druhou stranu všichni nabyli nových zkušeností a dovedností týkajících se vývoje a práce v týmu.

# 6 Zdrojové obrázky

## 6.1 Diagram konečného automatu




S = S_START
F1 = S_ID
F2 = S_INTEGER
S3 = S_NUMBER_DOT
F4 = S_NUMBER_DOT_NUMBER
S5 = S_NUMBER_E
F6 = S_NUMBER_E_NUMBER
S7 = S_STRING_START
S8 = S_STRING_BACKSLASH
S9 = S_STRING_BACKSLASH_ZERO
S10 = S_STRING_BACKSLASH_ZERO_ZERO
S11 = S_STRING_BACKSLASH_ZEROorONE_ONEtoNINE
S12 = S_STRING_BACKSLASH_ONE
S13 = S_STRING_BACKSLASH_TWO
S14 = S_STRING_BACKSLASH_TWO_ONEtoFIVE
F15 = S_STRING_END
S16 = S_COMMENT_START_SECOND
S17 = S_COMMENT_START_BLOCK_FIRST
S18 = S_COMMENT_BLOCK
S19 = S_COMMENT_BLOCK_END
F20 = S_STRLEN
F21 = S_ADD
F22 = S_SUB
F23 = S_MUL
F24 = S_DIV_NUMBER
F25 = S_DIV_INTEGER
F26 = S_COLON
S27 = S_CONCATENATION_FIRST
F28 = S_CONCATENATION_SECOND
F29 = S_SETVALUE
F30 = S_EQ
S31 = S_NEQ_FIRST
F32 = S_NEQ_SECOND
F33 = S_GT
F34 = S_GET
F35 = S_LT
F36 = S_LET
F37 = S_BRACKET_LEFT
F38 = S_BRACKET_RIGHT
F39 = S_COMMA

## 6.2 LL gramatika

1. <program> -> require"ifj21"<main_body>EOF
2. <main_body> -> /*eps*/
3. <main_body> -> global<dec_function><main_body>
4. <main_body> -> function<def_function><main_body>
5. <main_body> -> ID<call_function><main_body>
6. <dec_function> -> ID:function(<data_types>)<return_types>
7. <def_function> -> ID(<params>)<return_types><fce_body>end
8. <call_function> -> (<ids_datatypes>)
9. <fce_body> -> /*eps*/
10. <fce_body> -> local<def_var><fce_body>
11. <fce_body> -> <assign><fce_body>
12. <fce_body> -> if<cond><fce_body>
13. <fce_body> -> while<cycle><fce_body>
14. <fce_body> -> ID<call_function><fce_body>
15. <fce_body> -> return<return><fce_body>
16. <def_var> -> ID:<data_type>
17. <def_var> -> ID:<data_types>=EXPS
18. <def_var> -> ID:<data_types>=<call_function>
19. <assign> -> <ids>=<exps>
20. <assign> -> <ids>=<call_function>
21. <cond> -> EXPthen<fce_body>else<fce_body>end
22. <cycle> -> EXPdo<fce_body>end
23. <return> -> /*eps*/
24. <return> -> <exps_strings>
25. <return> -> <call_function>
26. <data_types> -> /*eps*/
27. <data_types> -> <data_type>,<data_types>
28. <data_types> -> <data_type>
29. <data_type> -> integer
30. <data_type> -> number

31. <data_type> -> string
32. <data_type> -> nil
33. <params> -> /*eps*/
34. <params> -> <param>,<params>
35. <params> -> <param>
36. <param> -> ID:<data_type>
37. <return_types> -> /*eps*/
38. <return_types> -> : <return_type>
39. <return_type> -> <data_type>
40. <return_type> -> <data_type>,<return_type>
41. <ids> -> ID
42. <ids> -> ID,<ids>
43. <exps> -> EXP
44. <exps> -> EXP,<exps>
45. <ids_datatypes> -> ID
46. <ids_datatypes> -> STRING
47. <ids_datatypes> -> NUMBER
48. <ids_datatypes> -> INTEGER
49. <ids_datatypes> -> NIL
50. <ids_datatypes> -> STRING,<ids_datatypes>
51. <ids_datatypes> -> NUMBER,<ids_datatypes>
52. <ids_datatypes> -> INTEGER,<ids_datatypes>
53. <ids_datatypes> -> NIL,<ids_datatypes>

LEGENDA:
ID → konkrétní identifikátor
EXP → konkrétní expression
STRING, INTEGER, NUMBER, NIL → konkrétně napsaný výraz daného datového typu

## 6.3 Precedenční tabulka

| X | +,- | *,/,// | Rel.Op | .. | # | ( | ) | i | $ |
|---|---|---|---|---|---|---|---|---|---|
| +,- | > | < | > | > | < | < | > | < | > |
| *,/,// | > | > | > | > | < | < | > | < | > |
| Rel.Op | < | < | ERR | < | < | < | > | < | > |
| .. | < | < | > | < | ERR | < | > | < | > |
| # | > | > | > | > | ERR | < | > | < | > |
| ( | < | < | < | < | < | < | = | < | ERR |
| ) | > | > | > | > | > | ERR | > | ERR | > |
| i | > | > | > | > | ERR | ERR | > | ERR | > |
| $ | < | < | < | < | < | < | ERR | < | ERR |