VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentace

Implementace překladače imperativního jazyka IFJ25

Tým `xliskah00`, varianta *TRP-izp*

Hana Liškařová	**`xliskah00`**	**25%**
Matěj Kurta	`xkurtam00`	25%
Martin Bíško	`xbiskom00`	25%
Šimon Dufek	`xdufeks00`	25%

Implementovaná rozšíření: CYCLES, BOOLTHEN, OPERATORS, FUNEXP

26. listopadu 2025

Obsah

1 Úvod

2 Práce v Týmu

2.1 Rozdělení práce mezi členy týmu

2.1.1 Hana Liškařová (`xliskah00`)

- Návrh konečného automatu lexikální analýzy a implementace celé lexikální analýzy.
-
-

2.1.2 Matěj Kurta (`xkurtam00`)

-
-

2.1.3 Martin Bíško (`xbiskom00`)

-
-

2.1.4 Šimon Dufek (`xdufeks00`)

-
-

2.1.5 Společná práce

-

2.2 Rozdělení bodů

3 Struktura projektu

```
projekt/
|-- Makefile
|-- dokumentace.pdf
|-- ast.c
|-- ast.h
|-- builtins.c
|-- builtins.h
|-- codegen.c
|-- codegen.h
|-- error.c
|-- error.h
|-- expressions.c
|-- expressions.h
|-- main.c
|-- parser.c
|-- parser.h
|-- rozdeleni
|-- scanner.c
|-- scanner.h
|-- scope_stack.c
|-- scope_stack.h
|-- semantic.c
|-- semantic.h
|-- stack.c
|-- stack.h
|-- string.c
|-- string.h
|-- symtable.c
|-- symtable.h
|-- token.c
`-- token.h
```

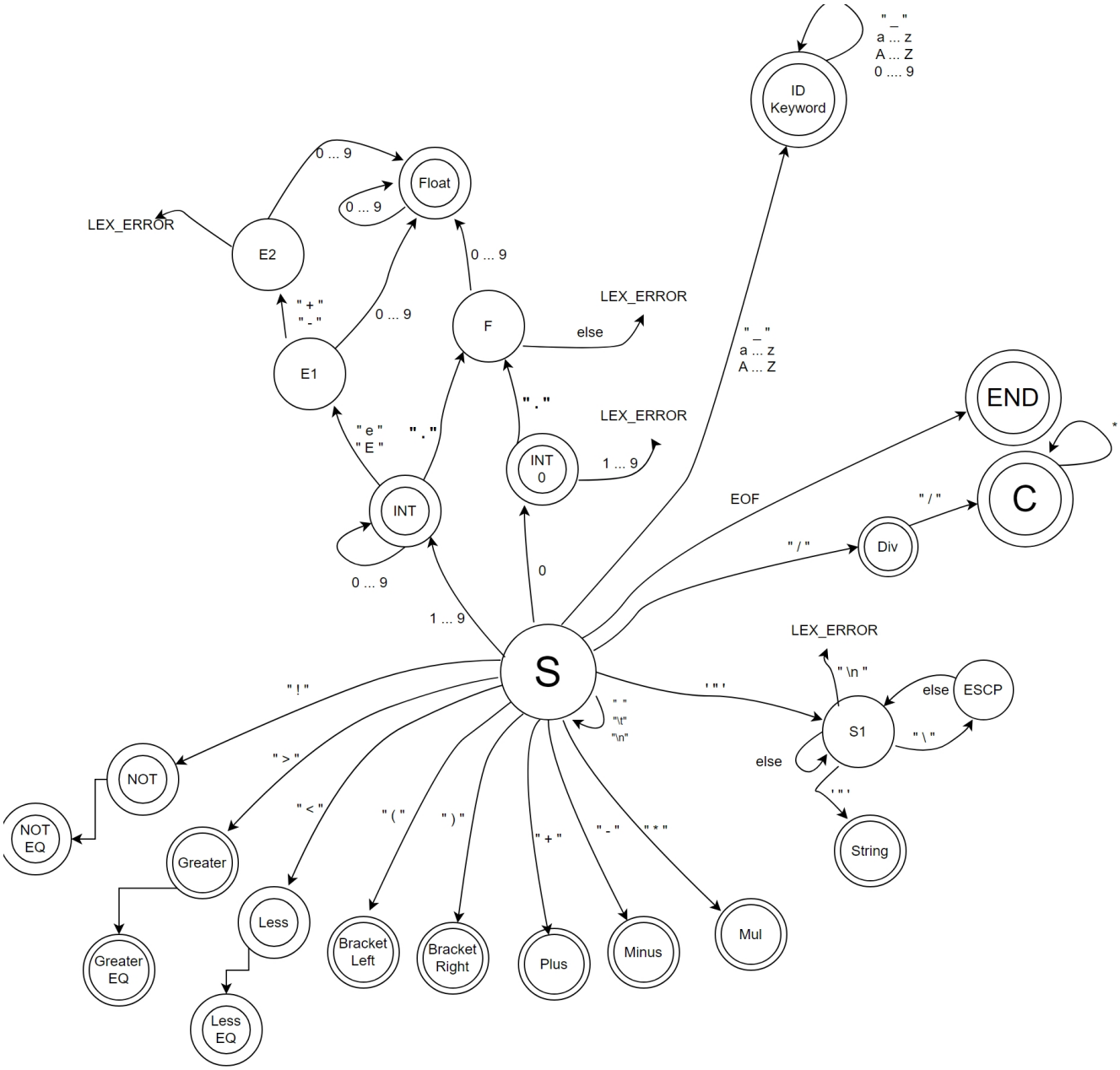
- **Makefile** – skript pro překlad projektu. Kompiluje všechny soubory do binárky `compiler` při `make`, obsahuje také pomocné cíle pro čištění (`make clean`) a testování.
- **dokumentace.pdf** – tento soubor. Projektová dokumentace ve formátu PDF.
- **main.c** – vstupní bod programu. Zajišťuje inicializaci, volání lexikální analýzy, syntaktické analýzy, sémantické analýzy a generátoru kódu, a na závěr uvolnění všech datových struktur.
- **scanner.c** a **scanner.h** – implementace lexikální analýzy jako konečného automatu. Čte znaky ze standardního vstupu a vytváří seznam tokenů, který slouží jako vstup pro syntaktickou analýzu.
- **token.c** a **token.h** – definice struktury tokenu a implementace obousměrného seznamu tokenů. Zajišťuje vytváření, průchod, mazání a základní pomocné operace nad sekvencí tokenů a tokenem samotným.
- **parser.c** a **parser.h** – syntaktická analýza. Zpracovává seznam tokenů podle gramatických pravidel a vytváří abstraktní syntaktický strom (AST).

- **expressions.c** a **expressions.h** – analýza výrazů založená na precedenční tabulce. Využívá zásobník a tokeny k parsování aritmetických a logických výrazů a převádí je do AST uzlů.
- **ast.c** a **ast.h** – definice datových struktur pro abstraktní syntaktický strom (AST) a funkce pro jeho vytváření, práci s jednotlivými uzly a těly, případně výpisy.
- **string.c** a **string.h** – implementace vlastního dynamického řetězce. Zajišťují alokaci, realokaci, konkatenaci, kopírování, mazání a výpis textu, včetně podpory pro generátor kódu.
- **error.c** a **error.h** – definice chybových kódů projektu a společná funkce `error` pro hlášení chyb na `stderr` a návrat odpovídajícího kódu.
- **symtable.c** a **symtable.h** – implementace tabulky symbolů založené na implicitně zřetězené hašovací tabulce. Uchovává informace o funkcích, proměnných a jejich vlastnostech (typ, rozsah, parametry apod.) pro potřeby sémantické analýzy.
- **stack.c** a **stack.h** – obecný zásobník (LIFO) nad ukazateli. Používá se například v analyzátoru výrazů a při práci s rámci sémantické analýzy.
- **scope_stack.c** a **scope_stack.h** – specializovaný zásobník rozsahů (scope stack) pro sémantickou analýzu. Ukládá ukazatele na lokální tabulky symbolů jednotlivých bloků a funkcí, čímž umožňuje kontrolu redeklarací a stínění identifikátorů.
- **builtins.c** a **builtins.h** – registrace vestavěných funkcí jmenného prostoru `Ifj.*` do globální tabulky symbolů (např. `Ifj.read_str`, `Ifj.write`). Ukládají se zejména názvy a arita funkcí.
- **semantic.c** a **semantic.h** – implementace první průchodové sémantické analýzy. Prochází AST, plní globální a lokální tabulky symbolů, kontroluje existenci `main()`, redeklarace, počet argumentů a základní typové kolize u literálových výrazů. - OPRAVIT AŽ BUDE HOTOVÉ
- **codegen.c** a **codegen.h** – generátor výsledného kódu. Rekurzivně prochází AST a vytváří cílový kód ve formátu `IFJcode25`, který je ukládán do dynamického řetězce a na konci vypsán na standardní výstup.
- **rozdeleni** – textový soubor s procentuálním podílem jednotlivých členů týmu.

4 Návrh a implementace

4.1 Lexikální analyzátor

4.1.1 Konečný stavový automat lexikální analýzy



4.2 Syntaktický analyzátor

4.2.1 Gramatická pravidla

1.

2.

3.

4.
5.
6.
7.
8.
9.
10.

4.2.2 LL1 tabulka

Neterminál	Program	Prolog	Function	Command	Declaration	Assignment	Condition	Loop	IfjFunction	FunctionCall	Return

4.2.3 Precedenční tabulka výrazů
