|  |
| --- |
| vYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  Fakulta informAčních technologií |
| Dokumentace k projektu z předmětu IFJ/IAL |
| Implementace jazyka IFJ14  Varianta b/3/I  Tým číslo 45 |
|  |
|  |
|  |

Tomáš Coufal xcoufa09, vedoucí týmu – doplnit rozdělení bodů

Roman Halík xhalik01

Yurij Hladyuk xhlady00

Jakub Jochlík xjochl00 V Brně, 14. prosince 2014

# OBSAH

[1 ÚVOD 4](#_Toc406166916)

[2 ZADÁNÍ 5](#_Toc406166917)

[3 ŘEŠENÍ 6](#_Toc406166918)

[3.1 Implemetace 6](#_Toc406166919)

[3.1.1 Lexikální analýza 6](#_Toc406166920)

[3.1.2 Syntaktická analýza 6](#_Toc406166921)

[3.1.3 Sémantická analýza 6](#_Toc406166922)

[3.1.4 Interpret 6](#_Toc406166923)

[3.1.5 Binární vyhledávací strom 6](#_Toc406166926)

[3.2 Speciální techniky 6](#_Toc406166927)

[3.3 Algoritmy 6](#_Toc406166928)

[3.3.1 Shell sort 6](#_Toc406166929)

[3.3.2 Boyer Moorův algoritmus 6](#_Toc406166930)

[3.4 Testování 7](#_Toc406166931)

[3.5 Práce v týmu 7](#_Toc406166932)

[4 ZÁVĚR 8](#_Toc406166933)

[5 PŘÍLOHY 9](#_Toc406166934)

[5.1 Konečný automat 9](#_Toc406166935)

[5.2 Pravidla LL gramatiky 9](#_Toc406166936)

[5.3 Precedenční tabulka 9](#_Toc406166937)

[LITERATURA 10](#_Toc406166938)



|  |
| --- |
|  |

# 1 ÚVOD

Tato dokumentace pojednává o tvorbě a úskalích týmového projektu, jehož cílem je naprogramovat interpret jazyka IFJ14, jenž je podmnožinou jazyka Pascal. Budeme se postupně zabývat jednotlivými částmi překladače a popíšeme řešené problémy při implementaci. Taktéž zmíníme způsob práce v týmu a organizaci práce.

# 2 ZADÁNÍ

Jazyk IFJ14 je podmnožinou jazyka Pascal, jedná se tedy o staticky typovaný jazyk, který je case insensitive. Nezáleží tedy na velikosti písmen. Pro každý tým bylo přiděleno specifické zadání. Pro náš tým konkrétně varianta b/3/I, která znamená následující:

Implementace tabulky symbolů pomocí **binárního** **vyhledávacího** **stromu**

Implementace řazení pomocí algoritmu **shell** **sort**

Vyhledávání podřetězce v řetězci za využití **Boyer-Mooreova** **algoritmu**

Program načítá zdrojový soubor jako parametr příkazové řádky a vyhodnotí, zdali je kód syntakticky i sémanticky v pořádku, a jestliže ano, provede kód. V případě chyby vrací jako návratovou hodnotu kód chyby definovaný zadáním projektu.

Překladač umí pracovat s datovými typy **integer**, **real**, **string** a **boolean**, zpracovávat výrazy s **aritmetickými** operátory a s operátory **porovnávání**. Dále umožňuje větvení **if-else** a cyklus **while**. Samozřejmostí jsou blokové komentáře. Navíc umožňuje pracovat se čtyřmi vestavěnými funkcemi:

**Lenght** – vrací délku vstupního řetězce

**Sort** – seřadí vstupní řetězec tak, že znak s nižní ordinární hodnou předchází znaku s vyšší ordinární hodnotou, vrací řetězec obsahující seřazené znaky

**Find** – vyhledá ve vstupním řetězci první výskyt zadaného podřetězce, vrací jeho pozici

**Copy** – vrátí podřetězec vstupního řetězce, určený jeho délkou a počátkem

# 3 ŘEŠENÍ

## 3.1 Implemetace

3.1.1 Lexikální analýza

Lexikální analyzátor pro tento projekt IFJ je založen na deterministickém konečném automatu. Mezi jeho hlavní funkce patří rozpoznání a klasifikaci lexémů, které následně reprezentuje pomocí tokenů. Zároveň také odstraňuje komentáře a všechny bílé znaky. Token v našem projektu je formou dvouprvkové struktury. První prvek určuje samotný typ tokenu, druhý pak dodatečné informace, je-li potřeba.

Vstupem je zdrojový program v jazyce IFJ14 a výstupem je token. Ten je pak dále použit jako vstup pro syntaktický analyzátor.

Konečný automat pro námi vytvořený lexikální analyzátor je tvořen 35 stavy, z nichž 21 je stavů konečných. Z důvodu přehlednosti bude následující diagram rozdělen do dvou částí:

* Přerušovaný stav ‘’int’’ značí místo rozdělení

**Hlavní tělo konečného automatu:**

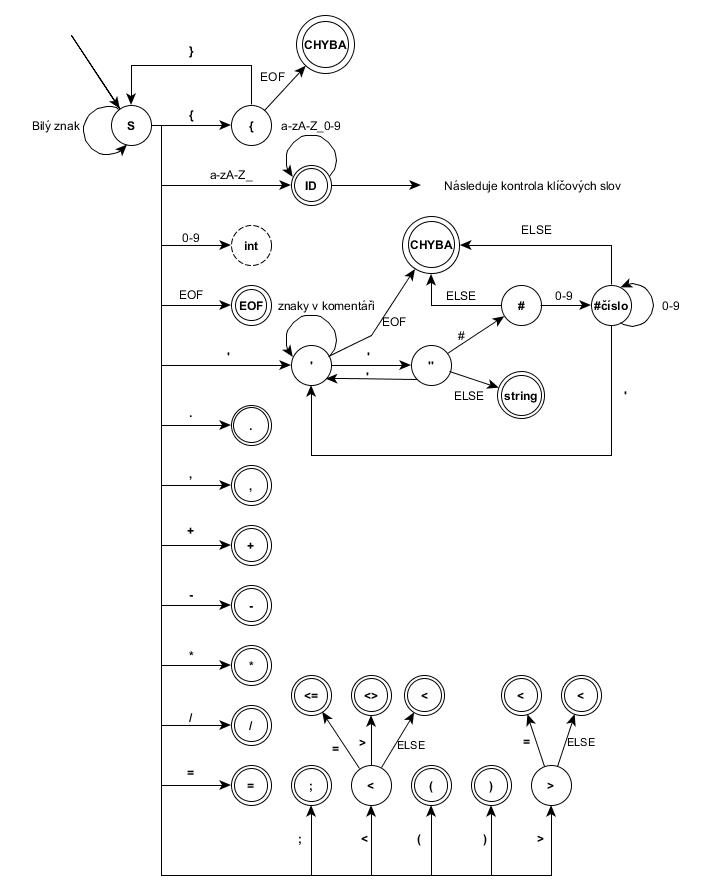
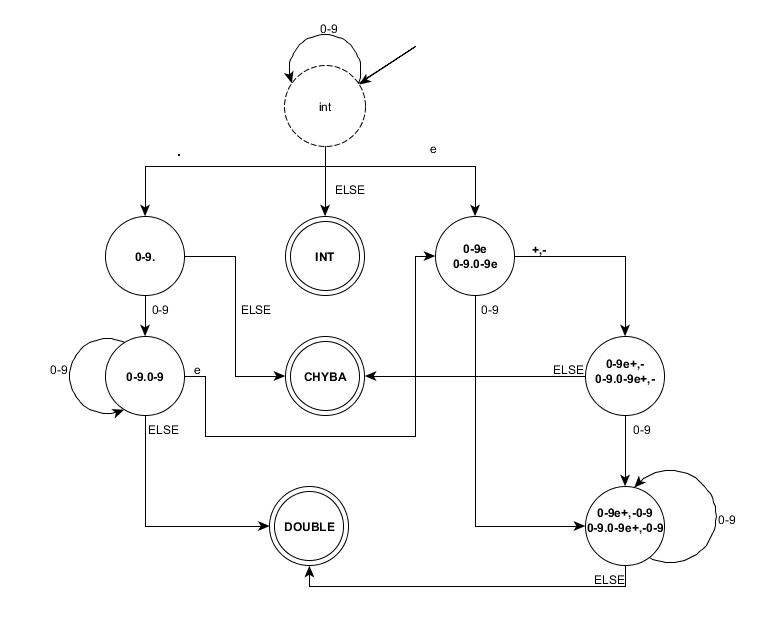


Diagram 1.1

**K-A pro číselný literál:**

Poznámka: Znak “e” zastupuje jak velké “E” tak malé “e”

Diagram 1.2

3.1.2 Syntaktická analýza

Doplní Jura

3.1.3 Sémantická analýza

Doplní Tom

### 3.1.4 Interpret

Úkolem interpretu je vykonávání instrukcí v podobě tří-adresného kódu, který přijímá od parseru (modul obsluhující syntaktickou a sémantickou analýzu). Samotný interpret je vlastně instrukční sada obsahující aritmetické instrukce, instrukce porovnávání, instrukce vestavěných funkcí a speciální instrukce – například instrukce skoku nebo přiřazení.

Interpret nejprve načte jednu instrukci, vykoná ji a načte instrukci následující, dokud není konec programu. Výjimkou je instrukce skoku, která provádí skok v seznamu instrukcí na danou adresu – návěští.

3.1.5 Binární vyhledávací strom

Doplní Tom

## 3.2 Speciální techniky

Neustálé vystavení stresu z nestihnutí deadline nepochybně povede ke zdárnému dokončení projektu v termínu

## 3.3 Algoritmy

### 3.3.1 Shell sort

Shell sort, neboli algoritmus řazení se snižujícím se přírůstkem, je řadící algoritmus založený na principu záměny dvou prvků vzdálených o stejný krok, který je na počátku řazení roven polovině délky řetězce, který řadíme. S každým průchodem se délka kroku půlí.

Implementace algoritmu proběhla na základě znalostí nabytých v předmětu IAL.

### 3.3.2 Boyer-Mooreův algoritmus

Boyer-Mooreův algoritmus je metoda umožňující rychlé vyhledání podřetězce v zadaném řetězci. Základní specifikum tohoto algoritmu spočívá v úvaze, že některé znaky, které se nikdy nemohou rovnat vyhledávanému vzorku, lze přeskočit. Touto metodikou lze velmi urychlit vyhledávání.

Samotný princip spočívá v průchodu řetězcem zleva doprava a porovnáváním posledního znaku vzorku s řetězcem, přičemž při shodě se dále porovnávají další znaky vzorku, směrem doleva. Jestliže je neshoda nalezena na i-tém znaku vzorku, posun v rámci řetězce se provede o délku vzorku - i + 1, tedy přesně za znak, který byl odlišný. Jinak se provádí posun o celou délku vzorku. Jestliže je nalezena shoda, algoritmus končí a vrací pozici nalezeného podřetězce. (Indexováno od 1)

Při implementaci algoritmu byly taktéž využity znalosti získané na přednáškách a v materiálech předmětu IAL.

## 3.4 Testování

Testovací soubory, pokusné odevzdávání

## 3.5 Práce v týmu

Práci jsme si rozdělili přibližně následujícím způsobem:

Tomáš Coufal – sémantická analýza, binární strom

Roman Halík – interpret

Yurij Hladyuk – syntaktická analýza

Jakub Jochlík – lexikální analýza

Doplňkové záležitosti se rozdělily dle potřeby a aktuálního vytížení členů týmu. Na vše přitom dohlížel náš vedoucí, který celý projekt „tmelil“ ve funkční celek.

Rozdělení práce v našem týmu bylo z počátku chaotické, lépe řečeno žádné. Každý se snažil především samostudiem přijít na to, jak to vlastně celé má fungovat a kde začít. Když se čas krátil, bylo nutné už určit, kdo co udělá a začít poctivě pracovat. Začali jsme se méně scházet a plánovat a více programovat. Ke sdílení kódu byl využit repozitář GIT.

# 4 METRIKY KÓDU

- možná bysme mohli napsat..

# 5 ZÁVĚR

Blabla naučili sme se nové věci, bla bla spolupráce v týmu, bla bla testování, bla bla bez rozšíření asi bla bla bla bla

# 6 PŘÍLOHY

- buď jako přílohy, nebo rovnou do textu, to je asi jedno, rozmyslet ;)

## 6.1 Konečný automat

## 6.2 Pravidla LL gramatiky

## 6.3 Precedenční tabulka

# LITERATURA

Bude třeba zmínit použití fronty z materiálů z wisu