# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



# Dokumentace ke společnému projektu pro předměty IFJ a IAL

## **Implementace interpretu imperativního jazyka IFJ16.**

# **Tým 086, varianta a/3/II**

# ZS 2016

### **Řešitelé:**

### Richtarik Lukáš xricht25

### Čechák Jiří xcecha04

### Mlýnek Přemysl xmlyne04

### Mynarčík Petr xmynar05

### Molitoris Miloš xmolit00

**Rozšíření:** UNARY, BASE, FUNEXP, SIMPLE, BOOLOP

7. 12. 2016

**1. Úvod**

Tato dokumentace popisuje implementaci interpretu imperativního jazyka IFJ16, který je zjednodušenou podmnožinou jazyka Java SE 8. Implementován je v jazyce C.

Interpret načte zdrojový soubor v jazyce IFJ16 a následně kontroluje, zdali v něm nejsou lexikální, syntaktické nebo sémantické chyby a během interpretace se kontroluje, zdali nenastanou běhové chyby. V případě nalezení chyby nebo projevení nějaké interní chyby interpretu, interpret vypíše na standardní chybový výstup chybovou hlášku a ukončí se s návratovou hodnotou dané chyby.

Jsou zde popsány implementace modulů, použité algoritmy a také diagram konečného automatu použitého v lexikální analýze, a také LL-gramatika a precedenční tabulka, které byly použity v syntaktické analýze.

**2. Diagram konečného automatu**

**3. LL-gramatika**

**4. Precedenční tabulka**

**5. Implementace modulů a algoritmů**

**5.1 Lexikální analýza**

Lexikální analýza je implementována pomocí konečného automatu. Začíná se vždy v počátečním stavu a v závislosti na načteném znaku ze zdrojového souboru se konečný automat posune do dalšího stavu, kde se již kontroluje na základě načtení následujících znaků, zdali je konkrétní lexém napsán správně.

Pokud je načten znak, který do daného lexému nepatří nebo se jedná o neočekávaný znak, jde o chybu a program je ukončen s návratovou hodnotou 1.

U komentářů se kontroluje, zdali jsou korektně zapsány, u víceřádkových i ukončeny.

Po přečtení a zpracování celého lexému, teď již tokenu, je tento token předán syntaktickému analyzátoru k syntaktické a sémantické analýze.

**5.2 Syntaktická a sémantická analýza**

Syntaktický analyzátor volá funkci get\_token, která se nachází v modulu scanner s implementací lexikální analýzy, a tato funkce předá syntaktickému analyzátoru token, který získá ze zdrojového souboru.

Nejdříve jsou získány globální proměnné a funkce a jejich identifikátory a informace o nich, jako jsou např. parametry u funkce, jsou uloženy do globální tabulky symbolů. Lokální proměnné se ukládají pak do lokálních tabulek symbolů konkrétních funkcí.

Správná syntaxe zdrojového programu se ověřuje rekurzivním postupem za použití LL-gramatiky a precedenční tabulky. Zpracování výrazů je prováděno metodou zdola nahoru podle precedenční tabulky.

Během sémantické analýzy dochází dle potřeby k přetypování proměnných a ke kontrole typů proměnných, návratových hodnot z funkcí, kontrola typů u parametrů funkcí apod. V případě chyby v této části analýzy je návratová hodnota 3, 4 nebo 6.

V případě bezchybné analýzy vzniká tříadresný kód, který se ukládá do instrukčního listu.

**5.4 Interpret**

Pokud nenastane nějaká chyba, tak interpret interpretuje zdrojový program napsaný v jazyce IFJ16. Interpret prochází instrukční list, který obsahuje tříadresné kódy a ty poté postupně interpretuje. V případě získání tříadresného kódu, který představuje skokovou instrukci, pokračuje v interpretaci od odpovídající instrukce.

Interpret také dle potřeby volá vestavěné funkce třídy ifj16.

**5.5 Implementace vyhledávání podřetězce v řetězci**

Pro vyhledávání podřetězce v řetězci byl podle zadání použit Knuth-Morris-Prattův algoritmus. Tento algoritmus pracuje na základě konečného automatu a byl implementován takto:

Pokud je hledaný podřetězec prázdný (má nulovou délku), nachází se tento podřetězec na nulté pozici řetězce a je tedy funkcí vrácena 0. Jinak je na základě řetězce, ve kterém vyhledáváme vytvořeno pole (vektor), který určuje znak, kam se vrátíme v případě neúspěšného porovnávání. Poté konečný automat postupuje po jednotlivých znacích řetězce, při shodě se posune na další znak, jinak se vrátí na znak určený dříve vytvořeným polem.

Porovnává se, dokud není podřetězec nalezen, pak funkce vrátí pozici podřetězce, nebo dokud se algoritmus nedostane na konec řetězce. V takovém případě funkce vrátí -1.

**5.6 Implementace řazení**

K implementaci funkce na řazení řetězce byl použit dle zadání algoritmus Shell sort, který pracuje na principu vkládání.

Nejdříve se získá délka kroku, což je polovina délky řetězce. Následně se v cyklu řadí znaky řetězce, které jsou od sebe v řetězci vzdáleny o velikost kroku a s každou další iterací se délka kroku zmenší o polovinu. Jakmile má krok velikost jedna, jsou řazeny prvky vedle sebe.

**5.7 Implementace tabulky symbolů**

Tabulka symbolů je implementována jako tabulka s rozptýlenými položkami s explicitním zřetězením synonym. K vyhledávání v tabulce symbolů a k ukládání položek do tabulky symbolů slouží klíč, kterým je identifikátor proměnné nebo funkce.

V globální tabulce symbolů jsou uloženy funkce s jejich parametry a globální proměnné. Do lokálních tabulek symbolů se ukládají lokální proměnné.

**6. Práce na projektu**

Nejdříve se rozdělili moduly, na kterých se začalo pracovat. Jakmile byly tyto moduly hotovy a otestovány, začalo se pracovat na dalších modulech. V průběhu práce na jednotlivých modulech se náš tým několikrát sešel k prodiskutování průběhu vývoje a potřebných změn v implementaci již implementovaných modulů.

Pokud nebyla možnost se sejít nebo bylo potřeba něco rychleji sdělit ostatním členům týmu, případně se na něco zeptat, využíval se ke komunikaci společný chat a soukromá skupina na sociální síti Facebook nebo email. K zálohování souborů a sledování jednotlivých verzí vytvářených částí interpretu jsme využívali server GitHub.

**7. Testování**

Každý modul byl testován nejdříve zvlášť během jeho vývoje a také hlavně po jeho dokončení. V případě nalezení chyb při testování modulu, případně více modulů dohromady, byly chyby odstraněny a konkrétní modul byl znovu otestován.

Po dokončení všech modulů a sestavení projektu začalo testování interpretu jako celku za použití námi vytvořených zdrojových kódů v jazyce IFJ16. Dle potřeby byly vytvářeny nové testy a případně upravovány ty staré.

**8. Rozdělení práce**

Lukáš Richtarik (xricht25) – vedoucí týmu: lexikální analyzátor, syntaktický analyzátor

Miloš Molitoris (xmolit00): lexikální analyzátor, syntaktický analyzátor, sémantická analýza, instrukční list

Jiří Čechák (xcecha04): algoritmy pro IAL, lexikální analyzátor, vestavěné funkce, správa paměti, tvorba závěrečných testů, dokumentace

Přemysl Mlýnek (xmlyne04): vestavěné funkce, tvorba závěrečných testů, výpomoc na dalších částech

Petr Mynarčík (xmynar05): interpret, instrukční list a generátor

**9. Implementovaná rozšíření**

UNARY

Podpora unárního mínus a prefixové i postfixové inkrementace a dekrementace.

Jednotlivé operátory jsou získány jako tokeny z lexikálního analyzátoru a pravidla pro práci s nimi jsou uloženy v precedenční tabulce. Prefixové inkrementace a dekrementace jsou provedeny před vyhodnocením celého výrazu. Postfixové inkrementace a dekrementace jsou provedeny postupně po vyhodnocení celého výrazu.

BASE

Podpora celých čísel zadaných ve dvojkové, osmičkové a šestnáctkové soustavě a desetinných čísel zadaných v šestnáctkové soustavě s možností použití znaku ‘\_‘, jako oddělovač číslic pro zpřehlednění zápisu ve všech soustavách, tedy i desítkové.

FUNEXP

Podpora volání statických funkcí uvnitř výrazu a parametry funkcí mohou být zapsány jako výrazy.

SIMPLE

Podpora podmíněných příkazů if i bez části else. V podmíněných příkazech a cyklech lze místo složeného příkazu v složených závorkách použít i jednořádkový příkaz.

BOOLOP

Podpora typu boolean, jeho definice a výpis, booleovských výrazů, hodnot true a false, operátorů ‘!‘ (negace), ‘||‘ (or) a ‘&&‘ (and).

**10. Závěr**

Prací na tomto projektu jsme si ověřili své znalosti v programování v jazyce C a získali i mnoho nových. Dále jsme mohli aplikovat své nově získané teoretické znalosti z předmětu IFJ a teoretické i částečně praktické znalosti (díky projektům v průběhu semestru) z předmětu IAL. Dále jsme všichni obohaceni o velkou zkušenost práce v malém týmu a komunikace s ostatními členy týmu a v neposlední řadě jsme si vyzkoušeli práci se zálohovacím systémem GitHub.