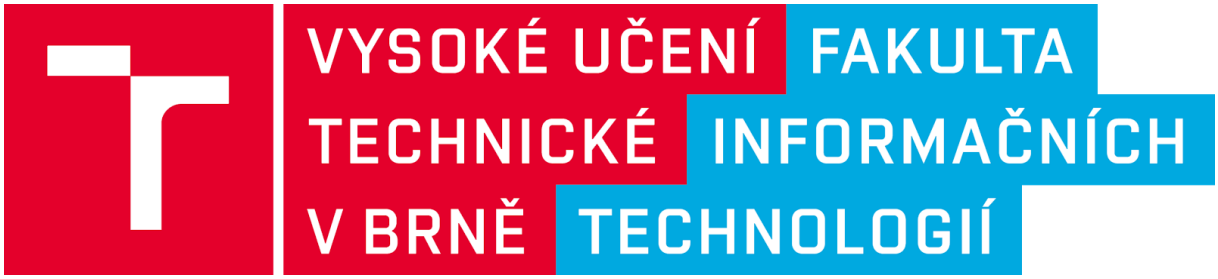
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÝCH TECHNOLÓGIÍ




Dokumentácia k projektu do predmetu IFJ a IAL

# Implementácia interpreta jazyka IFJ16

Tím 026, varianta *b/1/II*

25. novembra 2016

Členovia tímu:
Dávid Bolvanský (xbolva00), % - vedúci tímu
Juraj Dúbrava (xdubra03), %
Tamara Krestianková (xkrest07), %
Martin Marušiak (xmarus07), %
Veronika Svoradová (xsvora01), %

Rozšírenia – SIMPLE, BOOLOP

# OBSAH

# ÚVOD

# 1 IMPLEMENTÁCIA INTERPRETA JAZYKA IFJ16

## 1.1 Lexikálna analýza

Lexikálny analyzátor (scanner) je implementovaný v súboroch scanner.c a scanner.h. Je to jediná časť celého interprétu, ktorá pracuje priamo so zdrojovým súborom. Jeho úlohou je načítať zdrojový kód, odstrániť zbytočné časti ako sú biele znaky a komentáre, a finálne previesť lexémy na tokeny. Token sa skladá z dvoch častí: typ a atribút. Typ tokenu sa určí aplikovaním konečného automatu (viď príloha č. 1) na postupnosť znakov načítaných zo vstupu. Pre podporu nekonečne dlhého reťazca sme vytvorili pomocnú knižnicu, ktorá je implementovaná v súboroch strings.c a strings.h, a ktorá nám umožňuje pracovať s potenciálne nekonečným reťazcom. Ak prijatá postupnosť znakov nie je platná pre žiadne pravidlo, dochádza k lexikálnej chybe pri spracovávaní zdrojového textu. Lexikálny analyzátor poskytuje funkciu get_next_token, ktorá je následne využívaná syntaktickým analyzátorom.

## 1.2 Syntaktická analýza (bez spracovania výrazu)

Syntaktický analyzátor (parser) kontroluje množinu pravidiel udávajúcich prípustné konštrukcie jazyka, ktoré sú dané vytvorenou LL gramatikou pre jazyk IFJ16 (viď príloha č. 2). Je jadrom celého interprétu a riadi všetky ostatné časti. Samotná syntaktická analýza je riešená metódou rekurzívneho zostupu. Kvôli špecifickým požiadavkám jazyka Java, a teda aj IFJ16 ako jeho podmnožinou, dochádza k dvom prechodom v rámci syntaktického analýzátoru. V prvom prechode získavame tokeny z lexikálneho analyzátora a zároveň ich ukladáme do poľa tokenov, ktoré je implementované v súboroch token_buffer.c a token_buffer.h. Ak nenastane žiadna chyba počas prvého prechodu, pokračujeme. V druhom prechode získavame tokeny z práve vytvoreného poľa tokenov. Priebežne sa vrámci prvého a druhého prechodu generujú inštrukcie na inštrukčnú pásku a pri úspešnej syntaktickej a sémantickej analýze dochádza k samotnej interpretácii.

## 1.3 Syntaktická analýza (spracovanie výrazu)

//Juraj

## 1.4 Sémantická analýza

Úlohou sémantickej analýzy je preskúmať logický význam jednotlivých výrazov jazyka a zistiť ich platnosť pre daný programovací jazyk. Aj napriek tomu, že jazyk IFJ16 je podmnožinou jazyka Java, obsahuje isté špecifiká, ktorými sa od nej líši. Kľúčovým prvkom je spolupráca s tabuľkou symbolov. Z tejto tabuľky sa získavajú rôzne informácie a zároveň sa kontroluje resp. rozhoduje, či nedošlo k redeklarácií tried, premenných alebo funkcií. Zároveň sa kontroluje či nedošlo k typovej nekompatibilite pri priradení alebo vo výrazoch a taktiež prípadná nezhoda v počte parametrov funkcie, a rôzne iné sémantické kontroly

## 1.5 Interpret

//Martin

# 2 VSTAVANÉ FUNKCIE

## 2.1 Algoritmy do predmetu IAL

### 2.1.1 Implementácia radenia (Quick sort)

Quick sort alebo "radenie rozdeľovaním" patrí medzi najrýchlejšie metódy radenia polí. Tento algoritmus funguje na veľmi jednoduchom princípe. Pomocou mechanizmu *partition* prehodí prvky do dvoch častí poľa tak, že v ľavej časti sú všetky prvky menšie alebo rovné určitej hodnote (pivot) a v pravej časti sú všetky prvky väčšie ako táto hodnota.

V našom prípade sme za pivot zvolili pseudomedián. Pseudomedián sme vyrátali ako sčítanie ľavej a pravej hranice reťazca. Tento súčet sa nakoniec podelí (div) číslom dva:

pseudomedián = (ľavá hranica + pravá hranica) deleno (2).

Po rozdelení poľa na dve časti rekurzívne voláme funkciu pre opätovné radenie jednotlivých častí poľa. Celý algoritmus je uložený v súbore ial.c.

Priemerná doba výpočtu algoritmu Quick sort je v najlepšom prípade (O*(n*log(n))), no však pri nevhodnom tvare vstupných dát môže byť časová náročnosť tohto algoritmu až

O(n^2). Autorom algoritmu z roku 1962 je Sir Charles Antony Richard Hoare, významná osobnosť v obore teórie a tvorby programov.

### 2.1.2 Implementácia vyhľadávania podreťazca v reťazci (Boyer-Moore)

Boyer-Moore algoritmus je jedným z najefektívnejších algoritmov pre porovnávanie reťazcov. Používa sa na vyhľadávanie podreťazca v inom reťazci. Je vhodný pri hľadaní dlhého podreťazca. Funguje na princípe spracovávania znakov v hľadanom reťazci sprava doľava. Ak dojde k nezhode, nastáva posun. Čím dlhší je vyhľadávaný podreťazec, tým väčší počet znakov v reťazci je možné preskočiť. Pre implementáciu algoritmu sa používajú dve heuristiky. My sme použili obe, čím sa zvýšila efektivita algoritmu. Implementáciu môžete nájsť v súbore ial.c.

## 2.2 Implementácia tabuľky symbolov

Základom tabuľky symbolov je tzv. hashovacia tabuľka, ktorá je určená pre našu variantu zadania. Každá trieda má svoju tabuľku symbolov, v ktorej sú uložené informácie o statických (globálnych)

premenných a funkciách. Rovnako ako triedy tak aj funkcie majú svoju tabuľku symbolov, ktorá obsahuje informácie o parametroch/lokálnych premenných a zároveň o ich počte.

## 2.3 Vstavané funkcie pre načítanie literálu a výpis termov

## 2.4 Vstavané funkcie pre prácu s reťazcom

Medzi vstavané funkcie pre prácu s reťazcom v našom projekte patria nasledovné:

***int length (String s)*** - funkcia vráti dĺžku (počet znakov) reťazca zadaného parametrom "s"

***String substr (String s, int i, int n)*** - funkcia vráti podreťazec zadaného reťazca "s". Hľadaný podreťazec má dĺžku "n" a začína na indexe "i" zadaného reťazca "s".

***int compare (String s1, String s2)*** - funkcia lexikograficky porovnáva dva zadané reťazce "s1" a "s2" a vráti celočíselnú hodnotu: 0, ak sú reťazce "s1" a "s2" rovnaké,

1, ak je "s1" väčší ako "s2",

-1, v ostatných prípadoch

***int find (String s, String search)*** - funkcia nájde prvý výskyt zadaného podreťazca "seach" v reťazci "s" a následne vráti jeho pozíciu. Ak sa jedná o prázdny reťazec, vyskytuje sa vždy v každom reťazci na indexe 0. V prípade, že zadaný podreťazec "search" nie je nájdený, funkcia vráti hodnotu -1. V našom zadaní sme využili metódu Boyer-Moorovho algoritmu, ktorá je podrobne opísaná v časti 3.1.2 a nachádza sa v súboroch ial.c a ial.h.

***String sort (String s)*** - funkcia, ktorá zoradí znaky v danom reťazci tak, aby znak s nižšou ordinálnou hodnotou vždy predchádzal znaku s vyššou ordinálnou hodnotou. V našom projekte sme využili metódu Quick sort algoritmu, ktorý je podrobnejšie opísaný v časti 3.1.1 a nachádza sa v súbore ial.c.

# 3 PRÁCA V TÍME

# 4 ZÁVER

# 5 REFERENCIE