Dokumentace k projektu IFJ/IAL

Implementace interpretu imperativního jazyka IFJ13

9. prosince 2013

Tým číslo 97, varianta a/1/I

Řešitelé:

František Koláček(vedoucí)	xkolac12	25%
Lukáš Hermann	xherma25	25%
Daniel Stodůlka	xstodu06	25%
Tran Manh Hoang	xtranh00	25%

Obsah

1. Úvod

Tento dokument popisuje navrh a implementaci překladače imperativniho jazyka IFJ13.
Námi vybrané zadání (**a/1/I**) obsahuje tyto **dané detaily implementace**

- **Knuth-Morris-Prattův** pro vyhledávání ve stringu
- Řadící algoritmus **Quicksort**
- **Binární strom** jako tabulka symbolů

Samotný překladač se skladá ze tři části, implementovali jsme také vlastní správu paměti v podobě garbage collectoru. Jádrem překladače je **syntaktický analyzátor**, který má na starosti překlad zdrojového kódu. Pomocí **lexikálního analyzátoru** načítá zdrojový kód a podle sémantiky jazyka IFJ13 jej překladá na posloupnost instrukcí. V případě úspěšného překladu se spustí **intepretace,** která posloupnost instrukci vykoná.

2. Popis řešení

2.1 Lexikální analyzátor - scanner

Definuje struct_Token a funkci getToken(), který na požádání syntaktického analyzátoru načte ze vstupního souboru jeden lexém, který je pak vrácen jako token, jež obsahuje jeho typ. Typem může být například číslo, klíčové slovo, řetězec apod.
Vlastní scanner je řešen jako konečný automat. Načítá znaky z výstupního souboru pomocí funkce getc(). V některých případech načte scanner znak patřící dalšímu tokenu. V těchto případech se volá funkce ungetc(), která vrací znak ze vstupního souboru. V případě neznámého lexému se vrátí příslušný chybový kód.

2.2 Syntaktický analyzátor – parser

Syntaktický analyzátor tvoří jádro celého překladače. Zajišťuje přeložení zdrojového kódu na posloupnost instrukci. Při implementaci syntaktického analyzátoru jsme využili LL gramatiku (viz. Příloha 2), postupovali jsme metodou rekurzivního sestupu. Postupně jsme ukládali veškerá data, se kterými později bude pracovat i interpret, binární strom. Pro zpracovávání výrazů využíváme syntaktickou analýza zdola nahoru.
Oba dva druhy analýz kooperují, hlavní řízení je podle rekurzivního sestupu, avšak když se narazí na výraz, zavolá se analýza zdola nahoru. Podstatou syntaktické analýzy zdola nahoru je precedenční tabulka, na jejímž základě jsou řízeny veškeré redukce a vyhodnocování výrazu.

2.3 Interpret

Pro nás projekt jsme využili generátor tří-adresného kódu, který používá námi vytvořenou instrukční sadu. Generátor rozlišuje plnohodnotnou instrukci a instrukci skoku. Tyto instrukci se pro následné využití interpretu ukládají do listu instrukcí. Interpret zpracovává instrukce lineárně s výjimkou instrukcí skoků nebo návratů z funkcí. Abychom předešli duplicitní kontrole dat, tak při aritmetických operacích interpret také plní funkcí sémantického analyzátoru.

2.4 Správa paměti

Jednotka pro správu paměti je nádstavbou nad základními funkcemi (malloc, realloc apod.) a také práci se soubory. Pro její implementaci jsme využili jednoduchý jednosměrný seznam. Nejpodstatnější částí je funkce gcDispose(), která zajišťuje globální uvolnění veškeré použité paměti.

3. Použité algoritmy

3.1 Knuth-Moris-Prattův algoritmus

Tento vyhledávací algoritmus využívá ke své práci konečný automat. Je optimalizací triviálního způsobu vyhledávání a jeho výhodou je, že se v prohledávaném řetězci nevrací.
Implementuje také prefixovou funkci, která ještě před samotným vyhledáváním zajišťuje uložení informací o řetězci a jeho chování při posunu hledaného řetězci – díky tomuto nedochází ke vracení se k již kontrolovaným znakům.
Automat načítá postupně znaky řetězce, než se dostane do koncového stavu. Pokud načtený znak je koncem řetězce a automat není v koncovém stavu, jedná se o chybu. Počátečním stavem je -1, konečný stav je délka řetězce.

3. 2 Quicksort

Quicksort je jeden z nejrychlejších běžných algoritmů řazení založených na porovnávání prvků. Jedná se o nestabilní, nepřirozený algoritmus, jehož asymptotická časová složitost je linearitmická, výhodou je také jeho jednoduchost. Základní myšlenkou quicksortu je rozdělení řazené posloupnosti čísel na dvě přibližně stejně velké části (metoda rozděl a panuj). Kde v jedné části jsou čísla menší a ve druhé větší, pak je zvolen pivot, který je středem otáčení. Pokud budou obě části samostatně seřazeny, je seřazeno i celé pole. Obě části se pak rekurzivně řadí stejným postupem. Základní Quicksort je nejpomalejší při třídění již setříděných nebo z větší části setříděných polí.

4. Práce v týmu

Práce v týmu probíhala již od počátečního zveřejnění zadání, kdy náš vedoucí rozdal úkoly a stanovil termíny jejich plnění. Díky společnému místu bydliště našeho týmu komunikace nebyla problémem. Ke sdílení a práci se zdrojovými soubory jsme použili GIT.

5. Závěr

V zadaném termínu odevzdání náš interpret zpracovává ukázkové příklady, další ladění probíhá v režii vlastních testovacích skriptů a také programu Valgrind.

6. Příloha

1. Precedenční tabulka

V pořadí : id ( ) + - * / . == != < <= > >=

[TTYPE_VARIABLE] ={ 0 , 0 ,'>','>','>','>','>','>','>','>','>','>','>','>'}

[TTYPE_L_BRACKET]={'<','<','=','<','<','<','=','<','<','<','<','<','<','<'}

[TTYPE_R_BRACKET]={0 , 0 ,'>','>','>','>','>','>','>','>','>','>','>','>'}

[TTYPE_ADDITION]={'<','<','>','>','>','<','>','<','>','>','>','>','>','>'}

[TTYPE_SUBTRACTION]={'<','<','>','>','>','<','>','<','>','>','>','>','>','>'}

[TTYPE_MULTIPLICATION]={'<','<','>','>','>','>','>','<','>','>','>','>','>','>'}

[TTYPE_DIVISION]={'<','<','>','>','>','>','>','<','>','>','>','>','>','>'}

[TTYPE_DOT]={'<','<','>','>','>','<','>','<','>','>','>','>','>','>'}

[TTYPE_EQUAL]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_NOT_EQUAL]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_LESSER]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_LESSER_EQUAL]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_GREATER]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_GREATER_EQUAL]={'<','<','>','<','<','<','>','<','>','>','>','>','>','>'}

[TTYPE_SEMICOLON]={ '<','<', 0 ,'<','<','<', 0 ,'<','<','<','<','<','<','<'}

2. LL gramatika

<program> - <body_program>
<body_program> - <def_function><body_program>
<body_program> - <command>;<body_program>
<body_program> - eps
<def_function> - function idFunction (<params>) { <stat_list> }
<stat_list> - eps
<stat_list> - <command> ; <stat_list>
<param> - id <params>
<param> - eps
<params> - , id <params>
<params> - eps
<command> - id = <assign> ;
<command> - if (expression) { <stat_list> } else { <stat_list> }
<command> - while (expression) { <stat_list> }
<command> - return expression ;
<assign> - expression ;
<assign> - idFunction(<params>) ;
<assign> - boolval(term) ;
<assign> - doubleval(term) ;
<assign> - intval(term) ;
<assign> - strval(term) ;
<assign> - get_string() ;
<assign> - put_string(term) ;
<assign> - strlen(term) ;
<assign> - get_substring(term, num, num) ;
<assign> - find_string(string, string) ;
<assign> - sort_string(string) ;
<num> - eps
<num> - num
<term> - id
<term> - <value>
<value> - num
<value> - string
<value> - bool
<value> - null

3. Konečný automat scanneru