Semestrální práce MI-RUN 2011/2012

Implementace vlastního jazyka

Adam Kučera
Lukáš Kukačka
Jan Řanda
magisterské studium
České vysoké učení technické v Praze
Fakulta informačních technologií
Thákurova 9, 160 00 Praha 6
December 22, 2011

1 Zadání

Your task is to write a program solving one of problems listed below:

- Knapsack problem solver - `http://en.wikipedia.org/wiki/Knapsack_problem`
- SAT solver - `http://en.wikipedia.org/wiki/Boolean_satisfiability_problem`
- Simple SUDOKU solver
- Simple key-value store using B+ trees
- Regular expression matcher/replacer
- Unix utility: grep (no need for regexp support)
- LISP interpreter
- Something like problems above. You may choose your own (but the lab-leader must agree on, indeed)

"Sounds like problems for introductory programming courses, so what's the trick" you may think. Let's make it more challenging:

- Program should take its input from a file and prints output to another file, specified on command line. Input and output files should be human-readable/writable. There are no other requirements on the format except that.
- Program must be written in a programming language implemented by yourself, i.e. you must first design and implement a programming language interpreter or compiler and a runtime for it and then implement a solution of one of the above problems in it. You may design your own language or you may implement a (subset of) existing one - it is up to you.
- You may work in teams, max team size is 2 people.

- You may use any language you want to implement your programming language runtime.

Do not spend much time on parsing. Use some kind of compiler-compiler tool: yacc/bison, javacc, antlr, smaCC, petitparser.

1.1 Odevzdání

To submit your project, send an email to your lab leader with your project attached as a zip archive.

- The subject of submissio email must starts with [MI-RUN Project Submission] (exactly like this, including square brackes) - this helps us to automatically process your emails. Otherwise, your email may get lost in hundreds of other emails we receive.
- The attached filename should be in form of mi-run-¡username¿.zip (mi-run-¡username1¿-¡username2¿.zip respectively) where ¡username¿ is CVUT user name of the author (authors, respectively)
- The .zip archive should contain all the sources plus a script/makefile compile your project WITHOUT running an IDE. A makefile for GNU Make, nmake, build.xml for Apache ANT, Rakefile for Rake or similar is just fine (please, try to use something commonly used - it saves us a lot of time).
- short README.txt describing:
  - how to compile it
  - format of input and output files
- some sample input files

2 Kompilace a spuštění

3 Vlastní algoritmy

Pro otestování funkčnosti jazyka jsme zvolili následující jednoduché algoritmy:

- Hello World
- EvenOddNumber
- FindLargestSmallestNumber
- Bubble sort

Ze zadání jsme pak zvolili implementaci problému batohu (**Backpack - Knapsack problem**). V programu je napevno dán počet položek a maximální nosnost batohu. Dále pak načítáme data ze vstupního souboru, který je ve tvaru *váha cena váha cena váha cena* atd.. Batoh je řešen pomocí hrubé síly a prohledáváním každého možného stavu rekurzí. K tomu je vytvořena metoda *bruteBag()*. V běhu programu porovnáváme aktuální řešení s tím nejlepší a případně je kopírujeme. Výslednou nejlepší cenu spolu s věcmi, které dáme do batohu, tiskneme do nového souboru.

4 Popis jazyka

Jazyk byl napsán v jazyce Java.

Parametry programu:

- program může přijímat textové argumenty (např. pro vstupní a výstupní soubor)
- argumentů je povoleno 10
- z jazyka se k argumentům programu přistupuje přes speciální konstrukci
- argumenty jsou ukládány v tabulce konstant na adresách 0 až 9

4.1 Abstract syntax tree - AST

4.2 Compiler

4.2.1 Syntaktický analyzátor

Syntaktickou analýzu provádíme pomocí Parseru.

4.2.2 Lexikální analyzátor

Je součástí Parseru. Čte ze vstupního souboru znaky reprezentující zdrojový program a z těchto znaků vytváří symboly programu.

4.3 Interpret

Převádí instrukce do bytecode.

4.4 Bytecode

Překládá bytecode do instrukcí.

5 Tabulka konstant

- na zacatku bytecodu
- instrukce CONSTDEF

Tabulka konstant			
Název	Adresa DEC	Adresa HEX	Popis, parametry
print	0	0x00	vytiskne objekt z vrcholu zásobníku na výstup pomocí Object.toString() bez odřádkování (jako java System.out.print())
println	1	0x01	vytiskne objekt z vrcholu zásobníku na výstup pomocí Object.toString() bez odřádkování (jako java System.out.println())
readfileint	10	0x0A	načte ze souboru jeden (první) Integer na zásobník, 1. na zásobníku = název souboru, pokud se něco nepodaří (soubor neexistuje), interpreter skončí
readfileintarr	11	0x0B	načte ze souboru pole Integeru do proměnné, 1. na zásobníku = název souboru, 2. na zásobníku = adresa pole
writetofile	20	0x14	zapíše do souboru objekt ze zásobníku pomocí (Object.toString()), 1. na zásobníku = název souboru, 2. na zásobníku = objekt k zapsání
appendtofile	21	0x15	připojí objekt z vrcholu zásobníku na konec souboru (Object.toString()), 1. na zásobníku = název souboru, 2. na zásobníku = objekt k zapsaní

6 Tabulka instrukcí

Tabulka instrukcí		
Instrukce	HEX	Popis
pushc [cislo]	0x11	vložit číselnou konstantu (hodnotu) na zásobník
pushv [adresa - číslo]	0x12	vložit proměnnou (adresu) na zásobník, podívá se na tabulku proměnných a hodnotu vloží na zásobník
pushsc [adresa]	0x13	vloží na zásobník Object string z tabulky konstant
pop [adresa]	0x01	vyjmutí ze zásobníku a uložení na adresu slotu, vezme hodnotu z vrcholu zásobníku a uloží ji na adresu
arrdef [adresa pole]	0x20	na zásobníku je počet položek, po provedeni je pole na teto adrese
arrpop [adresa pole]	0x2A	vloží hodnotu do pole na adrese, nejvyšší hodnota zas = index v poli, 2. nejvyšší hodnota zas = hodnota
arrpush [adresa pole]	0x2F	z pole na adrese vloží hodnotu na vrchol zásobníku, nejvyšší hodnota zas = index v poli, vloží hodnotu na zásobníku
badd	0x30	sčítání (addition), načte 2 nejvyšší hodnoty ze zásobníku, sečte a výsledek vloží na zásobník
bsub	0x31	odčítání (substraction), načte 2 nejvyšší hodnoty ze zásobníku, odečte a výsledek vloží na zásobník
bmul	0x32	násobení (multiplication), načte 2 nejvyšší hodnoty ze zásobníku, vynásobí a výsledek vloží na zásobník
lab1: (aa:, fff4:, ...)	0xA0	návěští (identifikátor a dvojtečka)
mjmp [návěští]	0x40	skočí na návěští metody, vytvoří kopii environmentu, vloží do stacku enviromentu a nastaví callstack
mret	0x4F	ukončení metody, výsledek je na zásobníku, po zavolání se vrací dle callstacku na pozici callstack + 1
jmp [návěští]	0x50	nepodmíněný skok
jeq [návěští]	0x5A	podmíněný skok, skáče, když 2 nejvyšší hodnoty na zásobníku jsou stejné
jneq [návěští]	0x5B	skok, když nejsou stejné
jlt [návěští]	0x5C	skok, když nejvyšší hodnota na zásobníku je menší nez 2 nejvyšší
jgt [návěští]	0x5D	skok, když nejvyšší hodnota na zásobníku je větší než 2 nejvyšší
jelt [návěští]	0x5E	menší nebo rovno
jegt	0x5F	větší nebo rovno
constdef [adresa] [1...n bytu] 0xCE	0xCD	definuje textovou konstantu, první Integer adresa, druhý délka stringu, potom se čtou byty, ze kterých se sestaví string až do ukončovacího bytu 0xCE (nepatri ke stringu)
call [adresa v method table]	0xCA	volání statické metody (metoda interpretru) na adrese v tabulce metod
stop	0xFF	ukončení programu