SWE 514 Computer Systems Project Documentation

Fall 2017

Prepared by:Olgu Koçoğlu

SWE 514 Computer Systems
Homework 1 (due Nov. 30th)
(The project can be done in groups of 3 students)

In this project, you will implement a translator for a language called Simple that will compile Simple language code to A86 code. The grammar for the Simple language will be as follows:

stm → **id :=** *expr*
| **print** *expr*
| **if** *expr* **then** *stm*
| **while** *expr* **do** *stm*
| **begin** *opt_stmts* **end**

opt_stms → *stmt_list*
| **ε**

stmt_list → *stm* ; *stmt_list*
| *stm*

expr → *term moreterms*

moreterms → **+** *term moreterms*
| **-** *term moreterms*
| **ε**

term → *factor morefactors*

morefactors → ***** *factor morefactors*
| **/** *factor morefactors*
| **mod** *factor morefactors*
| **ε**

factor → **(** *expr* **)**
| **id**
| **num**

Your translator should be able to parse codes given in the Simple language following the grammar rules given above. Note that **id** is an identifier (variable) and **num** is a number. You can make the following assumptions:

- All numbers used in the source code can be assumed to be non-negative integers.
- / operation is integer division (i.e. fractional part is truncated).
- You can assume that the tokens are separated by whitespace (blanks or newline). For example **2*(3+4)** will be given as **2 * (3 + 4)** .
- There will be no unary minus (i.e. **–** is used as binary operation).

Documentation (written document describing how you implemented your project)	15%
Comments in your code	5%
Simple Compiler implementation and tests	80%
Bonus: Remove the assumption that tokens are separated by whitespace.	10%

Introduction:

This program is designed to get data file for simple language as an input and produce an x86 assembly data file(.asm) with the same name for the input code.

It uses a class called “assembly” and many functions in it. The program asks you to type the input file’s name which must be in the same directory with the program and produces the “asm” file with the same name in the directory.

This program is written in the python 3.5 language and the spyder IDE is used for coding.

The program automatically parses the entire input and creates an array which has all the expressions and statements. There is no need to seperate tokens with whitespace. The program seperates all the expressions, variables and integers without the need of whitespace and checks for correctness except statements, which need to have “;” or a new line between them.

Functions In The Class:

showcommands(self): It returns array of the assembly code for the input.

showvariables(self): It returns the variables of the input.

definevariables(self,myarray): It adds commands for defining variables at the beggining of the code.

parsing(self): It parses the whole input, seperates the integers, expressions, variables and statements and puts them in an array.

expr(self,myarray): It looks for the expressions in “parsedarray”.

term(self,myarray,token): It calls factor function for the previous token of the expression token and morefactor function for the expression element.

moreterms(self,myarray,index): It looks if the expression is '+' or '-', and adds necessary assembly code to the commands array.

factor(self,myarray,index): It looks for the token if it is an integer, variable or parantese. If it is a parantese it calls expr function for inside of the paranteses. Finally it adds necessary assembly code to the commands array.

morefactors(self,myarray,index): It looks if the expression is '*', '/' or 'mod'. If it is, it calls factor function for the previous token and finally adds necessary assembly code to the commands array.

opt_stms(self,myarray): It looks for the statements in the “parsedarray”. If there is no statement, it raises an error.

stmt_list(self,myarray,index): It looks for the statements in the parsedarray and calls the stm function if there is one.

stm(self,myarray,index): It looks the token and distinguish which statement it is. Then it adds necessary assembly code to the commands array and make the statement element blank to mark it has already processed.

Conclusion:

The program successfully creates the assembly code from an input file written in the simple language.

It successfully makes while, if, variable defining, begin-end, print and expression operations which includes addition, subtraction, multiplication, division and modular arithmetic.