COMP322: Assignment 2- Winter 2013

Due at 11:30pm EST, 1 March 2013

1 Background Information

See the file titled `BackgroundInformationAssignment2.pdf` on the course webpage for information necessary to complete the assignment. That pdf contains information about pointers, sparse polynomials, structs, and linked lists which will help with the assignment. In addition, we will go over some of this in class.

2 Assignment goals

On this assignment, you will:

- Learn how to read from a file using a stream
- Practice using *iterators*
- Use a container *class* in the C++library : `vector` (conceptually like an array) and `list` (conceptually like a linked list). This will provide a brief introduction to object oriented programming as we'll have variables that can "do" things.
- Use the *algorithms* that come with the C++library so that you don't need to write everything yourself.

3 Assignment Requirements

In this assignment, you will write a text interpreter in C++relying on the standard library. The language will be expanded/modified in future assignments to become more elaborate. For now, the language will be simple and will consist of executing the instructions a file from top to bottom. Exactly one statement per line will be used.

The language, which we will call `322` has the following rules:

- Any line starting with a ! is ignored by the interpreter (i.e. a comment)
- There are 2 commands that can be given:
  - SET
  - PRINT
- The language is case sensitive meaning that **set** and **pRint** are invalid commands and a variable **aaa** is different than the variable **AAA**.

Each instruction in `322` will have the following rules:

1. **SET** SET is used to create and set the value of a variable. The syntax will always be SET variable = *expression*. *expression* can be one of 2 things: a literal value (e.g. 10.5, -100, etc) or another variable name.

   For example: SET a = 10 SET b = a SET b = 5

   You do not need to worry about more complex expressions such as involving arithmetic. You may assume for simplicity that the spacing is as above. That is, the word SET appears, then a space, then a variable name, another space, an equals, another space, and the expression. However, you may NOT assume that the expression on the right side of the equation is valid. For example, the right side expression may contain a variable that doesn't exist.

   If the left side of the equation has an identifier that doesn't already exist as a variable, your program should create a new variable. If the identifier already exists, your program should overwrite the existing value.

2. **PRINT** PRINT is used to print the value of a variable to the screen. The syntax will always be PRINT *variable*. You may again assume the spacing is as given, but you may not assume that the variable actually exists.

You may also add further instructions to the language `322` if you want, as long as these instructions are backwards compatible with the original language. (You can call your language `322++` if you want.) In this case, you should document your changes in the confession.txt file submitted as a readme below.

You should write a C++program to do the following:

1. Your program should accept a command line argument representing a file name and then open that file.

2. It should load every instruction from the file into the `Memory` struct (see below) according to the rules of the grammar above. Instructions should be executed from top to bottom until the last instruction is executed at which point the program should terminate.

3. If any of the commands are invalid (i.e. they begin with an instruction that doesn't exist or reference a variable that is not found) your program should print an error message and then stop parsing the file at that point.

In order to do this, you should use two structs given to you `Variable` and `Memory` found on the course webpage. The file `Parser.cc` also contains the very basics of a main function to read the command line arguments.

The struct `Variable` contains two fields : name and value. (We will not worry about types for now.) The struct `Memory` contains 3 fields `vector<string> instructions`, `int position`, and `vector<Variable> variables`. The point of the `Memory` struct is when your program first is loaded, you can load all of the instructions into this struct and set the position to be 0. Then, as you execute each instruction, you can modify the variables as appropriate as well as update the position variable.

You may modify the structs given if you like, however, to keep your list of variables, you must use one of the stl containers. (You could use list¡Variable¿ for example if you preferred though.)

3.1 Sample execution

A sample file is provided on the course webpage. For convenience, it is posted here as well:

```
!This program demonstrates what we would do
SET a = 10
SET b = 100
PRINT b
SET b = 10
PRINT b
```

Here is the resulting execution:

```
[dan][~/comp322/assignment2] g++ -o Parser Parser.cc
[dan][~/comp322/assignment2] ./Parser testfile.txt
100
10
```

3.2 Hints

The main hint is try to make use out of the standard library as much as possible. For example, the Memory variable contains a container of all of the variables in your program. For both print and set, you need to search through the list of variables in the container to see if any of them match the variable name specified. This can be accomplished (with a bit of tinkering) by the function in `algorithm` called `find_if`. As we haven't learned about objects yet, you may need to create a global variable to let you use a custom function for this. For reading individual lines from the file, you can use the function `getline` and for breaking the string into many pieces you can use a `istringstream`. If you use the library functions properly it is possible to do this in relatively little code. If you find yourself writing several hundreds of lines of code and getting bugs because of it, you may find it useful to discuss with each other, Dan, or a TA ways to try to reduce your code.

Remember the idea of passing a variable or struct by reference. If you add an & into the function header before the name of the variable, then any changes to the variable in the function will last in the calling function as well. For example, if your header is the following

```
void foo(Memory& memory)
```

you can change `memory` inside the function `foo` and you'll see changes in your calling function.

4 Using/installing g++

g++ is the GNU C++ compiler. A recent version (4.3.2) is available on most of the Ubuntu lab computers. Our solution was developed using version 4.4.1.

We build our solution using the command line:

```
g++ -Wall -o Parser.exe Parser.cc
```

If you want to install g++ on your system, you should be able to find instructions around the web.

For the Mac, the instructions here look reasonably accurate and up-to-date:

`http://www.edparrish.com/common/macgpp.php`

For PC/Windows systems, there are multiple choices, but the Cygwin environment mimics most of the Linux command line on Windows, so it is a good option:

`http://www.cygwin.com`

For Ubuntu (and possibly Debian) users, you can install the free `g++` package with the following command:

```
sudo apt-get install g++
```

If you have trouble, or you're running some other operating system, let us know and we'll try to help you.

5 Submitting your assignment

What To Submit

`Parser.cc`

`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.