CS2040: POPL: Programming Assignment #0
Initial Steps in Imperative Programming using C/C++
Due Wednesday January 21st, 2015 at 08:00 AM

This assignment#0 is aimed at giving you an small introduction to the kind of assignments you can expect in this POPL course. This is an individual assignment.

1. **Reading component:** Start reading Stroustrup's *A Tour of C++* `http://www.stroustrup.com/Tour.html` till chapter#4.

2. **Intset:** Define, implement, and test a set of integers, class **Intset**. It should provide the following operations: union, intersection and difference. The data-structure you use to implement is left to your choice (exceptions include the STL set datastructures such as *std::set* and any other library that provides a direct implementation of sets). The set should not be limited to any specific range of values and must be able to handle large number of elements within itself. The provided interface file has the structure of the class that must be implemented (this part is compulsory). You are however free to add more functions if you need to.

   You must then use your implementation of IntSet and create a menu-driven program with the following options:

   (a) **Insert elements**
   (b) **Delete elements**
   (c) **Check elements**
   (d) **Union**
   (e) **Intersection**
   (f) **Difference**

   The first 3 operations take a number N as input and followed by N elements. For the Check operation only **True** or **False** must be printed. The other 3 operations (Union, Intersection, Difference) take 2 file names as input, the first file name contains the elements of the second set and the second file name will be the output file name in which the set after performing the operation will be printed in ascending order. The set in the memory must also be kept intact for further operations.

   The input and output files will contain one element on each line of the file. There will not be any incorrect entries in these files.

   The following must be the behaviour in special cases:

   - When an element is inserted which is already present, the program should output **X ELEMENT ALREADY PRESENT**, where **X** is the value of the element.
   - When an element is deleted which is not already present, the program should out **X ELEMENT NOT PRESENT**, where **X** is the value of the element.

The number of elements that could be added to your set could be very large and your program must not crash under such circumstances.

3. **Symbol Table:** Design a symbol table class and a symbol table entry class for some language. Have a look at a compiler for that language to see what the symbol table really looks like.

Background Information:

A Symbol Table is a collection of key-value pairs. The key is a string that uniquely identifies a particular binding and the value is data associated with that key. One should be able to INSERT a key-value pair and also LOOKUP the value associated with a key from the table. The data associated with that key could be a value of any type.

Consider a simple example:

```
int temp = 45;
```

Here the key is "temp" and the value associated with it is 45(type is int).

Scopes:

The scope of a name binding is the part of a computer program where the binding is valid: where the name can be used to refer to the entity(value). In other parts of the program the name may refer to a different entity (it may have a different binding), or to nothing at all (it may be unbound).[1]

In this assignment we assume Dynamic Scoping. However C/C++ folllows Lexical scoping. (You can read more about scopes from the above mentioned link).

Consider the following example:

```
 1. int x = 10, y  = 20;
 2. {
 3.    // The outer block contains declaration of x and y, so
 4.    // following statement is valid and prints 10 and 20
 5.    printf("x = %d, y = %d\n", x, y);
 6.    {
 7.        // y is declared again, so outer block y is not accessible
 8.        // in this block
 9.        int y = 40;
10.        x++;  // Changes the outer block variable x to 11
11.        y++;  // Changes this block's variable y to 41
12.        printf("x = %d, y = %d\n", x, y);
13.    }
14.    // This statement accesses only outer block's variables
15.    printf("x = %d, y = %d\n", x, y);
16. }
```

In the above example y is defined twice while x is defined once. Line 5 prints $x = 10, y = 20$. A new **scope** is entered in line 6. Next, when we define y in line 9, it is created inside the newly entered scope. The current symbol table can be visualized as follows:

[1] http://en.wikipedia.org/wiki/Scope_(computer_science)

```
x=10, level=1
y=20, level=1
y=40, level=2
current level=2
```

In line 10, the symbol x refers to the variable in scope level 1, and its value is modified. However, in line 11, the symbol y can refer to one of the two variables named y. The way by which programming languages deal with this is by starting from the current scope level and moving towards outer scopes. The first variable matching the symbol in innermost scope(highest level number) is considered. Thus, for the line 11, the variable y from level 2 would be updated as the other $y$would be hidden by it. The new state of the symbol table can be visualized as follows:

```
x=11, level=1
y=20, level=1
y=41, level=2
current level=2
```

When a closing brace is encountered in line 13 the scope level 2 is exited and removed. Now, the symbol table would look something like what follows:

```
x=11, level=1
y=20, level=1
current level=1
```

Task:

Your task in this assignment is to implement a basic Symbol Table class with various member functions. An interface has been provided to you in SymbolTable.h. Your job is to implment those functions in SymbolTable.cpp.

- **SymbolTable(): (Constructor)** Initialize a new empty Symbol Table
- **~SymbolTable(): (Destructor)** Free memory occupied by SymbolTable.
- **void enterscope(void):** Enter a new scope.
- **void exitscope(void):** Exit current scope and go up one scope. Stop at global scope(topmost scope).
- **void insert(char *key, void *value):** Insert (key,value) pair in symbol table at current scope. It should overwrite previous binding (if any) of current scope.
- **void* probe(char *key):** Lookup for value associated with key only in the current scope. If binding exists, return pointer to value else return NULL.
- **void* lookup(char *key):** Lookup for value associated with key in the current scope. If binding exists, return pointer to value else go up one scope and keep going up until you find binding. If found return pointer to value else return NULL.

Testing:

To generalise the symbol table, the data type of "value" is a pointer of type void*. However for testing purposes the data to be stored is of type int. To compile your code run make and to execute the binary run ./test It contains a menu interface with which your assignment will be tested.

Language and interface You have to use C/C++ for this assignment. You have to use the interface provided.

Submission You should submit the code for each of each of these problems. Submission details would be posted to the class group.

Evaluation You should also write a short report on your code. Your code report should show a *good* understanding of the problem as well as a good summary of what you implemented. The usual rules of plagiarism apply to your code and your report. In particular **do not** directly copy the material from websites. Very good codes/reports will fetch bonus points.

Note: Please note that your submission will be evaluated using a script, so make sure that your program's output matches exactly with what is mentioned in the problem statement.