CSCI204/MCS9204 Assignment 3

(Total 15 marks, Due by 11:59 pm sharp on Friday, 30 October, 2015)

Aims

This assignment aims to establish a basic familiarity with C++ classes. The assignment introduce increasingly object-based, C++ style of solution to a problem.

General Requirements

- You should observe the common principles of OO programming when you design your classes.
- You should make proper documentation and implementation comments in your codes where they are necessary.
- Logical structures and statements are properly used for specific purposes.

Objectives

On completion of these tasks you should be able to:

- Code and run C++ programs using the development environment.
- Make effective use of the on-line documentation system that supports the development environment.
- Code programs using C++ in a hybrid style (procedural code using instances of simple classes) and in a more object-based style.
- Manipulate string data.
- Generic programming.
- Standard template classes.

Tasks:

Task 1: Template class, iterator (6.0 marks)

Linked list is a data structure consist a group of nodes that linked. Linked list is a very useful container to store data.

In this task, you will define and implement a template class of a singly linked list.

Define a *namespace* **MyProject** in a file **MyList.h**.

Define a *template* class **Node** that consist two data members: A template *data* and a Node pointer *next*. You may define the necessary member functions for the template class.

Define a *template* class **MyList** in the namespace MyProject in the file MyList.h. The template class consist two data members: A Node pointer *head* and a Node pointer *tail*. The pointer *head* points to the first node of a linked list. The pointer *tail* points to the last node of the linked list. Initially the linked list is empty. Define the following member functions for the class MyList.

- Default constructor.
- Copy constructor.
- Destructor.
- push_front function takes a template data as a parameter and adds it at the front of a linked list (in front of the first node if it is not empty).
- push_back function takes a template data as a parameter and adds it at the tail of a linked list (after the last node if it is not empty).
- isEmpty function returns true if a linked list is empty.
- front function returns the data in the first node of a linked list.
- back function returns the data in the last node of a linked list.
- pop_front function removes the first node in a linked list. It does not return any data.
- pop_back function removes the last node in a linked list. It does not return any data.
- Define nested class **iterator** in the class MyList which consist data members: A Node pointer *current* and a Node pointer *last*. Define the following member functions and overloading operators for the iterator class.
    - Default constructor.
    - Dereference operator * returns the current node's data.
    - Post increment operator ++ moves the current pointer points to the next node in the list.
    - Overloading comparison operator "not equals" compares the iterator with another iterator. Returns true if both iterators' *current* pointers are the same.
- begin function returns an iterator with pointer *current* points to the first node of the list, and pointer *last* points to the tail of the list.
- end function returns an iterator with pointer current set to NULL and pointer last points to the tail of the list.
- print function takes a std::ostream reference parameter and prints all data stored in the linked list to the ostream by **using the iterator define above**.

Implement member functions for the template classes **Node**, **MyList** and iterator in the name space **MyProject** in the file **MyList.h**

Write a driver program in a file **task1Main.cpp** to testing the member functions of the template classes defined above with different data. See the Testing of the task for more details.

Testing:

In this testing program, we will test the template class MyList with different data, such as integers, characters, strings. You can compile the program like (MyList.h should be included in the file task1Main.cpp)

```
g++ -o task1 task1Main.cpp
```

and run the program (no data input for this task)

```
./task1
Push integers 0, ..., 10 to the back of a linked list
Display integers from the front of the linked list
0 1 2 3 4 5 6 7 8 9
Push characters A, ..., Z to the back of a linked list
Display characters from the front of the linked list
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Copy a linked list with characters into another linked list.
Display characters from the front of the copy linked list
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Push strings Monday, ..., Sunday to the back of a linked list
Display strings from the front of the linked list
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

Task2: Stack and expressions (5.0 marks)

Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code.
In this assignment, you are to investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Human generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands-this is called *infix notation*. Computers "prefer" *postfix notation* in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation and then evaluate the postfix version of the expression. Each of these

algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm the stack is used for a different purpose.

In this task, you are to write a C++ program of the infix-to-postfix conversion algorithm.

Define and implement a **Stack** template class in a file **stack.h** which use MyList template class (which has been done in Task 1) as a stack.
Your Stack template class must have the following public functions:

- void push(T): to insert a data to the Stack.
- T pop(): to retrive a data from the Stack.
- bool isEmpty(): this function will return 1 if the Stack is empty or 0 otherwise.
- void print() : this function will print the contents of the Stack WITHOUT destroying the contents of the Stack.
- T stackTop() : this function will return the TOP element of the Stack WITHOUT destroying the contents of the original Stack.

Implement the member functions in the file **stack.h**.

Define a class called **InfixToPostfix** in a file **postfix.h**, which is able to convert an infix notation to a postfix notation.
The class has two private variables:

```
string infix;
string postfix;
```

These variables are used to hold the infix and postfix notations, respectively.

Define the following public member functions for the class **InfixToPostfix**:

- void setInfix(const std::string &) : This is a function to set initial value to the data member *infix*.
- void convert (): This is a function that convert the infix notation hold in the infix variable to a postfix notation variable. You will use the template class Stack defined above in this function.
- std::string getPostfix(): This is a function that returns a postfix notation from the data member *postfix*.

Implement the member functions for the class Infix2Postfix in a file **postfix.cpp**.

The algorithm to convert an infix notation to a postfix notation is as follows:

1. Push the left parenthesis '(' to the stack.
2. Append a right parenthesis ')' to the end of infix expression.
3. While the stack is not empty, read infix from left to right and do the following:
    - If the current input in infix is an operand, copy it to the next element of postfix.
    - If the current input in infix is a left parenthesis, push it onto the stack.
    - If the current input in infix is an operator, then:

        - Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in postfix.
        - Push the current operator in infix onto the stack.
    - If the current input in infix is a right parenthesis, then:
        - Pop operators from the top of the stack and insert them in postfix until a left parenthesis is at the top of the stack.
        - Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

```
+    addition
-    subtraction
*    multiplication
/    division
%    modulus
```

The precedence of operators from the lowest to highest is defined as following:

```
+  -
*  /  %
```

Write a driver program include main function in a file **task2Main.cpp** to declare an instance of InfixToPostfix. Then get an infix notation from the keyboard and convert the infix notation to the postfix notation. Finally print out the postfix notation.

For example, when the input is

```
1 + 3
```

The output of postfix notation should be

```
1 3 +
```

You can download the files **exp1.txt** and **exp2.txt** to test your program. See the testing for more details.

Testing:

You can compile the program by

```
g++ -o task2 task2Main.cpp postfix.cpp
```

Run the program like

```
./task2 < exp1.txt
```

```
AB  CDE  *  RST  UV  XX  /  -  3  *  +  X5  -
```

Test the program by exp2.txt like

./task2 < exp2.txt

12.53 21.2 33.2 15.4 2 / - * + 8.5 2.6 12 2 / + * -

Use bcheck to make sure there is no memory leak.

Task 3: STL container (4.0 marks)

Define a class **Website** in a file **website.h** that consists of *name* and *website*. Define the necessary member functions and insertion operator (<<) for the class Website. Implement necessary member functions and insertion operator for the class in a file **website.cpp**.

Define a class **WebManagement** in a file **webmanagement.h** that consists of a **map** container for **Website** objects. Define the following member functions for the class WebManagement:

- loadWebsites: Loads websites from the given text file into the map container.
- run: Displays a menu, gets user's input and call the corresponding member functions. See the Testing for more details.
- addWebsite: Adds a new website into the map container.
- findWebsite: Finds the website by the given name and display the name and website.
- updateWebsite: Updates the website for a given name.
- removeWebsite: Removes the website for a given name.
- displayAll: Displays all names and their websites that stored in the map container. Use iterator for the map container to display all items.
- saveWebsites: Save the names and their websites into the given text file. Use the same format of data that loaded.
- Other necessary member functions or operators.

Implement the member functions and operators in a file **webmanagement.cpp**.

Write a driver program (include main function) in a file **task3Main.cpp** to get the text file name from the command line arguments, create a WebManagement object, display menu, get choices and manage the website records. See the Testing for more details.

Testing:

You can compile the program by
g++ -o task3 task3Main.cpp website.cpp webmanagement.cpp

Run the program like (Red data indicate input from the keyboard)

./task3 websites.txt
9 records loaded.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 5

ANU:http://www.anu.edu.au
Facebook:http://www.facebook.com
Google:http://www.google.com
IBM:http://www.ibm.com
MICROSOFT:https://www.microsoft.com
UOW:http://www.uow.edu.au
UNSW:http://www.unsw.com.au
UOS:http://sydney.edu.au
WIKIPEDIA:https://en.wikipedia.org/wiki/Main_Page

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 1

Name: UOW
The website for UOW already exists.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 1
Name: EBAY-AU
Website: http://www.ebay.com.au
The website has been added.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 2
Name: Google-AU
The website does not exist.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 2
Name: Google
Google:http://www.google.com

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 3
Name: Google
Website: http://www.google.com.au
The website has been updated.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 4
Name: UOS
The website has been removed.

1. Add a new website

2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 5

ANU:http://www.anu.edu.au
EBAY-AU:http://www.ebay.com.au
Facebook:http://www.facebook.com
Google:http://www.google.com.au
IBM:http://www.ibm.com
MICROSOFT:https://www.microsoft.com
UOW:http://www.uow.edu.au
UNSW:http://www.unsw.com.au
WIKIPEDIA:https://en.wikipedia.org/wiki/Main_Page

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 6
Filename: savedWebsite.txt
9 records have been saved.

1. Add a new website
2. Find a website
3. Update a website
4. Remove a website
5. Display all websites
6. Save all websites
0. Quit
Choice: 0
Bye.

Submission

This assignment is due by 11.59 pm (sharp) on Friday 30 October 2015.

Assignments are submitted electronically via the ***submit*** system.

For this assignment you must submit all the files via the command:

$ submit -u your_user_name -c CSCI204 -a 3 task1Main.cpp MyList.h task2Main.cpp stack.h postfix.h postfix.cpp website.h website.cpp webmanagement.h webmanagement.cpp task3Main.cpp

and input your password.

Make sure that you use the correct file names. The Unix system is case sensitive. You must submit all files in one ***submit*** command line.

Your program code must be in a good programming style, such as good names for variables, methods, classes, and keep indentation.

Submission via e-mail is NOT acceptable.

After submit your assignment successfully, please check your email of confirmation. **You would loss 50% of the marks if your program codes could not be compiled correctly.**

Late submissions do not have to be requested. Late submissions will be allowed for a few days after close of scheduled submission (up to 3 days). Late submissions attract a mark penalty; this penalty may be waived if an appropriate request for special consideration (for medical or similar problem) is made via the university SOLS system *before* the close of the late submission time. No work can be submitted after the late submission time.

A policy regarding late submissions is included in the course outline.

The assignment is an **individual assignment** and it is expected that all its tasks will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during tutorial classes or office hours. Plagiarism will result in a **FAIL** grade being recorded for that assessment task.

End of specification