CSCI204 Assignment 3

(Total 15 marks, Due by 11:59 pm sharp on Friday, 7 June, 2013)

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.
- Understand template and STL

Tasks:

Task 1: Template classes (8.5 marks)

In this task, you will define and implement template classes for a double linked list.

Define and implement a *template class* **Node** and a *template class* **MyList** in a file **MyList.h**.

The template class **Node** has three data members, which are template data, a Node pointer points to the previous node and a Node pointer pointes to the next node. Define constructors and other necessary member functions include **operator** << to print out the template data.

The template class **MyList** has two data members: A Node pointer points to the head of a double linked list, and a Node pointer points to the tail of the double linked list. Define and implement constructors and destructor for the class **MyList**. Define and implement member functions include

- **push_front()** : Add a new node in the front of the double linked list;
- **push_back()**: Add a new node at the tail of the double linked list;
- **front()**: Return the node in the front and the node still remains in the double linked list);
- **pop_front()**: Remove the node in the front;
- **operator=()**: Make a (deep) copy of double linked list from another double linked list of same data type.
- Define and implement **nested** class **iterator** in the template class **MyList**. There are two data members in the class iterator: MyList object (refer to the MyList object) and a Node pointer. Define and implement member functions include:
    - Constructors;
    - Pre-fix increment operator and postfix increment operator that make the pointer points to the next node;
    - Comparison operator != that compare whether two iterator objects point to the same node in the double linked list;
    - Output operator << to print out the current data where pointer points to;
    - Other necessary member functions.
- **Begin()**: return an iterator object where the pointer points to the front node of the double linked list.
- **End()**: return the iterator object where the pointer points to the end of the double linked list.

Implement main() function and other sub functions in a file **task1Main.cpp** to test the template classes defined above. You must test the member functions listed above.

Testing:

You can compile the task 1 by

```
CC –o task1 task1Main.cpp
```

Then run the program like (input data in red colour):

```
task1
Input number of integers: 10
```

Input 10 integers: 0 1 2 3 4 5 6 7 8 9
Integer list 1 (push_front(), loop by using iterator):
9 8 7 6 5 4 3 2 1 0
Input number of integers: 10
Input 10 integers: 10 11 12 13 14 15 16 17 18 19
Integer list 2 (push_back(), loop by using iterator):
10 11 12 13 14 15 16 17 18 19
Integer list 2 = list 1 (loop by using use front() and pop_front():
9 8 7 6 5 4 3 2 1 0
Input number of doubles: 10
Input 10 doubles: 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Double list (loop by using iterator):
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5

Note: Your program of task 1 should work on different testing data.

Task 2: Template classes and Binary file (3.5 marks)

Use the template classes **Node** and **MyList** defined in the task 1. Define a class **Customer** in a file **Customer.h** and implement the member functions in a file **Customer.cpp**. Define three data members for the class Customer: a customer id number, customer's first name and customer's last name. Define and implement constructors and other necessary member functions.

Implement main() function and other functions in a file **task2Main.cpp** to read data of Customers from keyboard and store those data into a MyList object. Then save the data from the object into a given binary file. At the end, read customers' data from the binary file and display the records.

Testing:

You can compile the task by

CC –o task2 task2Main.cpp Customer.cpp

Then run the program like (input data in red colour):
Task2

Input number of records: 10
Input 10 records:
12345 Peter Cook
54321 David Smith
21345 Alice Brown
32145 Angela Smith
45321 William Brown

54123 Alwin Smith
23145 Bob Cook
43215 Do Little
53124 Caven Brown
32451 Chris Smith

Input a file name to save data: customers.dat

10 customer records saved.

Load customer records from the binary file.

12345 Peter	Cook
54321 David	Smith
21345 Alice	Brown
32145 Angela	Smith
45321 William	Brown
54123 Alwin	Smith
23145 Bob	Cook
43215 Do	Little
53124 Caven	Brown
32451 Chris	Smith

Note: Your program of task 2 should work on different testing data.

Hint: You need use iomanip to generate formatted outputs for records.

Task3: STL (3 marks)

Use the class **Customer** define in the task 2, create a map object to store the customer records. The key should be the customer id.

Implement main() function and other functions in a file **task3Main.cpp** to load data from a given binary file into a **map** container, and print out the accounts' information by using iterator according to the id order.

Testing:

Use CC to compile the source files by
CC –o task3 task3Main.cpp Customer.cpp
and run the program by
task3

The binary file can be the file created in the task 2. The outputs of task 3 look like the following:

Input a binary file name: customers.dat

10 accounts have been loaded.

12345 Peter	Cook
21345 Alice	Brown
23145 Bob	Cook
32145 Angela	Smith
32451 Chris	Smith
43215 Do	Little
45321 William	Brown
53124 Caven	Brown
54123 Alwin	Smith
54321 David	Smith

Note: Your solutions should work on different testing data / files.

Submission

This assignment is due by 11.59 pm (sharp) on Friday June 7, 2013.

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

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

$ submit -u your_user_name -c CSCI204 -a 3 MyList.h task1Main.cpp Customer.h Customer.cpp task2Main.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.

Remember the submit command should be in one 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 (25% off the marks for each day late); 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