CSCI124/MCS9124 Assignment 5

(Total 8 marks, Due by 11:59 pm sharp on Sunday, 7 June, 2015)

Aims

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

General Requirements

- Observe the common principles of OO programming when you design your classes.
- Put your name, student number at the beginning of each source file.
- 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.
- Linked list and queue.

Problem Specification

A **priority queue** is an abstract data type which is like a regular queue data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

A priority queue must at least support the following operations:

- *enqueue*: adds an element into the priority queue ordered by the priority values. If the elements have the same priority, the elements following the rule of the queue: first in first out.
- *dequeue*: removes the element from the priority queue that has the *highest priority*, and returns the value.

Some convention functions for the priority queue will be defined,

- *peek*: returns the element value from the priority queue that has the highest priority but still keep the element in the priority queue.
- *removeLowest*: removes the lowest priority element from the priority queue.
- *find*: finds an element in the priority queue and return the element's information. The priority queue should not be changed.
- *incrementPriority*: increases an element's priority. The location of the element should be reset.
- *merge*: merges two priority queues into a new priority queue.
- *print*: prints all the elements in the priority queue.

Implementation

Use a doubly linkedlist to implement the priority queue. Use the following *guideline* to implement the solution to this assignment.

Step 1. Define a structure Task and functions. (1.0 mark)

Define a structure **Task** contains data fields: **userID, taskID, priority** and **commandPath** in a file **task.h**.

- The filed **userID** stored user's identity, such as "user_1", …, "user_n", "guest_1", …, "guest_n", where n is a positive integer.
- The field **taskID** is a no duplicated integer (positive) for the task.
- The field **priority** is a no negative integer of a task's priority. In this assignment a higher priority has a lower value (that means 0 is higher than 1, 1 is higher than 2, etc.).
- The field commandPath stores the path of the command of the task. For example, a user's path can be "/home/user/user_i", i=1, …, n. A guest's command path can be "/home/guest/guest_i", i=1, …, n.

Define the prototype of a comparison function to compare two Task objects by their **priorities** (or **taskIDs)** in the file **task.h**. The comparison function returns a positive integer if the first Task object's priority (or taskID) is higher than the second Task object's priority (or taskID); If the first Task object's priority (or taskID) is lower than the second one, returns a negative value, otherwise returns 0 (zero).
Define an output function to print out the task data (userID, taskID, priority and command path) to an **ostream** object.

Implement the comparison function(s) and output function in a file **task.cpp**.

Hint: To generate the formatted output, you can use the manipulators (#include <iomanip>). Use "setw(length)" to set the output width, use "left" or "right" to make the output aligns to the left or right. For example:

```
char str[8]="Value";
int val = 95;
cout << setw(10) << left << str << setw(5) << right << val << endl;
```

The output looks like

```
Value       95
```

There are 8 white spaces between "Value" and "95".

Step 2. Define a class DNode and a class DLinkedlist and implement their member functions. (4 marks)

Define and implement classes for a doubly linked list in this step.

Define a class **DNode** for the doubly linked list. Each node in a doubly linked list contains a data member (Task type) and two pointers of **DNode** type that can be used to point previous node and next node.

Define a class **DLinkedlist** that consists of two pointers point to the head and back of a doubly linked list. The class **DLinkedlist** can be a friend class of **DNode**. The following member functions must be defined for the class.

- Default constructor;
- Copy constructor;
- Destructor;
- *push_front*: Puts a new element at the front of the doubly linked list.
- *push_back*: Puts a new element at the back of the doubly linked list.
- *pop_front*: Pops the element at the front of the doubly linked list.
- *pop_back*: Pops the element at the back of the doubly linked list.
- *front*: Retrieves the element at the front of the doubly linked list, the doubly linked list won't be changed.
- *back*: Retrieves the element at the back of the doubly linked list, the doubly linked list won't be changed.
- *exists*: Returns true if the element already exists in the doubly linked list. In this assignment the taskID value is unique.
- *insert*: Inserts a new element into the doubly linked list according to its key's value. In this assignment the key value for insert function is the priority of tasks.
- *hasMore*: Returns true is the doubly linked list has more nodes.
- *print*: Prints all elements' information that stored in the doubly linked list to an ostream object.

- *reomve*: Finds the element according to a data field's key value and removes the node from the doubly linked list. The removed element should be returned to the calling function. The key value for finding the element is the taskID.

You may add some other member functions.

Define the classes **DNode**, **DLinkedlist** in a file **dlinkedlist.h** and implement their member functions in a file **dlinkedlist.cpp**. You may use

typedef Task Item;

Then use the type Item to define the data member type for the classes DNode and DLinkedlist.

Step 3. Driver functions for the doubly linked list.

Implement driver functions include **main()** function in a file **main.cpp** to test the doubly linked list member functions.

Step 4. Define a class PQueue and implement its member functions. (2 marks)

Define a class **PQueue** for a priority queue in a file **pqueue.h**. The priority queue will be implemented by using the doubly linked list that defined in the Step 1 and 2.

Define the following member functions for the class PQueue.

- Default constructor;
- Copy constructor;
- *enqueue*: Puts the new element in the priority queue according to the element's key value. In this assignment the key is the priority of tasks. If the task already exists (same taskID), just return without add the new element in the priority queue.
- *dequeue*: Removes the element from the priority queue. The element has the highest priority. The removed element's value will be returned.
- *peek*: Retrieves the element's value from the priority queue. The element has the highest priority in the queue.
- *removeLowest*: Removes the element from the priority queue, where the element has the lowest priority.
- *find*: Finds an element in the priority queue according to the value of a data field. In this assignment use taskID as the key value. Saves the found element in a parameter for the calling function and returns true. Otherwise returns false.
- *incrementPriority*: Finds the element in the priority queue and update its priority by incrementing its priority using the given value. The element should be

reallocated in the priority queue. **Note: In this assignment, incrementing a tasks' priority means reduce its priority's value.**

- *merge*: Merges the current priority queue with another priority queue that passed to the function as a parameter into a new priority queue; Returns the new priority queue. The two priority queues (current one and the parameter one) cannot be changed.
- *print*: Prints the elements in the priority queue to an ostream object.

You may add some other member functions for the class.

Implement the member functions for the class **PQueue** in a file **pqueue.cpp**.

Step 5. Driver functions for the priority queue. (1.0 mark)

Implement driver function(s) in the file **main.cpp** to test the priority queue member functions.
In these driver functions, use random number generator to generate random values for each task's userID, taskID, priority and commandPath (use prefix path with userID). Insert the tasks into two priority queues, such as q1 and q2; merge two queues q1 and q2 into a new queue (for example q3). Test rest of member functions.

See **Testing** for more details.

Hint: You can use **ostringstream** (#include <sstream>) to concatenate strings and integers together. For example,

```
ostringstream outs;
int id = 31;
outs << "user_" << id;          //outs contains "user_31"
char userID[30];
strcpy(userID, (outs.str()).c_str());     //outs converted into string,
                                          //then string converted into a char array,
                                          // then copied the char array to the char array userID
outs.str("");     //Reset outs to empty
outs << "/home/user/" << userID << "/";       //outs contains "/home/user/user_31/"
char commandPath[100];
strcpy(commandPath, (outs.str()).c_str());
```

Testing

You can compile the program by

```
g++ -o ass5 main.cpp task.cpp dlinkedlist.cpp pqueue.cpp
```

Run the program
./ass5

To test the priority queue, you may set the priority values are in the range [0, 9]; the taskID values are in the range [0, 9999]; the userID values are in the range [user_0, user_80], or in the range [guest_0, guest_19].

Only one input from the keyboard for a taskID will be needed for the *find* function. If the task has been found, then change its priority value. Otherwise print out the message, such as "Task *task_id* does not exist."

The assignment 5 testing output example files **output1.txt**, **output2.txt** and **output3.txt** can be downloaded from this site. In the result file output1.txt, the input task id is not exists. In the result file output2.txt, the task id exists and it was near the end of the priority queue. In the result file output3.txt, the task id is the front task's id. After the priority increased by 2, its priority is still zero since the highest priority is zero.

Submission

This assignment is due by 11.59 pm (sharp) on Sunday 7 June, 2015.

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

For this assignment you must submit the 7 files via the command:

$ submit -u your_user_name -c CSCI124 -a 5 task.h task.cpp dlinkedlist.h dlinkedlist.cpp pqueue.h pqueue.cpp main.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 no matter how many time you have submitted the solutions.

Remember the submit command scripts in the file 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%~100% 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 for each day late); this penalty may be waived if an appropriate request for Academic 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