CSC3002: Introduction to Computer Science

Assignment 3

Assignment description:

This assignment will be worth 10% of the final grade.

You should write your code for each question in a .cpp and .h file (please name it using the question name, e.g. q1.cpp). Please pack your whole project files into a single .zip file, name it using your student ID (e.g. if your student ID is 123456, then the file should be named as 123456.zip), and then submit the .zip file via Moodle.

You should create an empty project in QT and start your programing.

Please note that, the teaching assistant may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we may check whether your program is too similar to your fellow students' code using Moodle.

Please refer to the Moodle system for the assignment deadline. For each day of late submission, you will obtain late penalty in the assignment marks. If you submit more than three days later than the deadline, you will receive zero in this assignment.

Detailed description on assignment requirement is stated in the last few pages.

Question 1:

Most modern editors provide a facility that allows the user to copy a section of the buffer text into an internal storage area and then paste it back in at some other position. For each of the three representations of the buffer given in this chapter, implement the method

void EditorBuffer::copy(int count);

which stores a copy of the next count characters somewhere in the internal structure for the buffer, and the method

void EditorBuffer::paste();

which inserts those saved characters back into the buffer at the current cursor position. Calling paste does not affect the saved text, which means that you can insert multiple copies of the same text by calling paste more than once.

Test your implementation by adding the commands C and P to the editor application for the copy and paste operations, respectively. The C command should take a numeric argument to specify the number of characters using the technique described in exercise 5 of Chapter 13.

Question 2

In the queue abstraction presented in this chapter, new items are always added at the end of the queue and wait their turn in line. For some programming applications, it is useful to extend the simple queue abstraction into a priority queue, in which the order of the items is determined by a numeric priority value. When an item is enqueued in a priority queue, it is inserted in the list ahead of any lower priority items. If two items in a queue have the same priority, they are processed in the standard first-in/first-out order.

Using the linked-list implementation of queues as a model, design and implement a pqueue_list.h interface that exports a class called PriorityQueue, which exports the same methods as the traditional Queue class with the exception of the enqueue method, which now takes an additional argument, as follows:

```
void enqueue(ValueType value, double priority);
```

The parameter value is the same as for the traditional versions of enqueue; the priority argument is a numeric value representing the priority. As in conventional English usage, smaller integers correspond to higher priorities, so that priority 1 comes before priority 2, and so forth.

Q3

Use the algorithm from section 18.7 of the textbook to implement the PriorityQueue class so that it uses a heap as its underlying representation. To eliminate some of the complexity, feel free to use a vector instead of a dynamic array.

Q4

Write a function

bool pathExists(Node *n1, Node *n2);

that returns true if there is a path in the graph between the nodes n1 and n2. Implement this function by using depth-first search to traverse the graph from n1; if you encounter n2 along the way, then a path exists. Reimplement your function so that it uses breadth-first search instead.

Q5

Complete the definition of the Employee class hierarchy by adding the necessary instance variables to the private sections and the implementations for the various methods. Design a simple program to test your code.

Question Requirement

Q1:

You should write two files: buffer.h, buffer.cpp and P1.cpp.

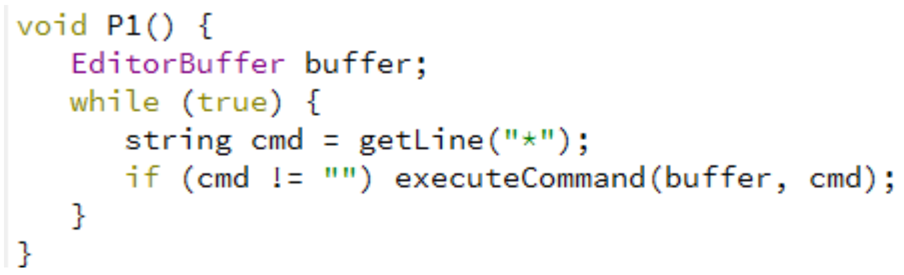
Of them, buffer.h and buffer.cpp should includes declaration and implementation of class EditorBuffer. Please refer to Figure 13-2 of textbook. Note that you should use an array-based implementation of the buffer.

P1.cpp includes the test code and some helper functions. Please refer to Figure 13-3 of textbook. Besides that, you should design two commands C and P, and their corresponding methods to be added to EditorBuffer class:

void EditorBuffer::copy(int count);

void EditorBuffer::paste();

The test code should be in void P1() :

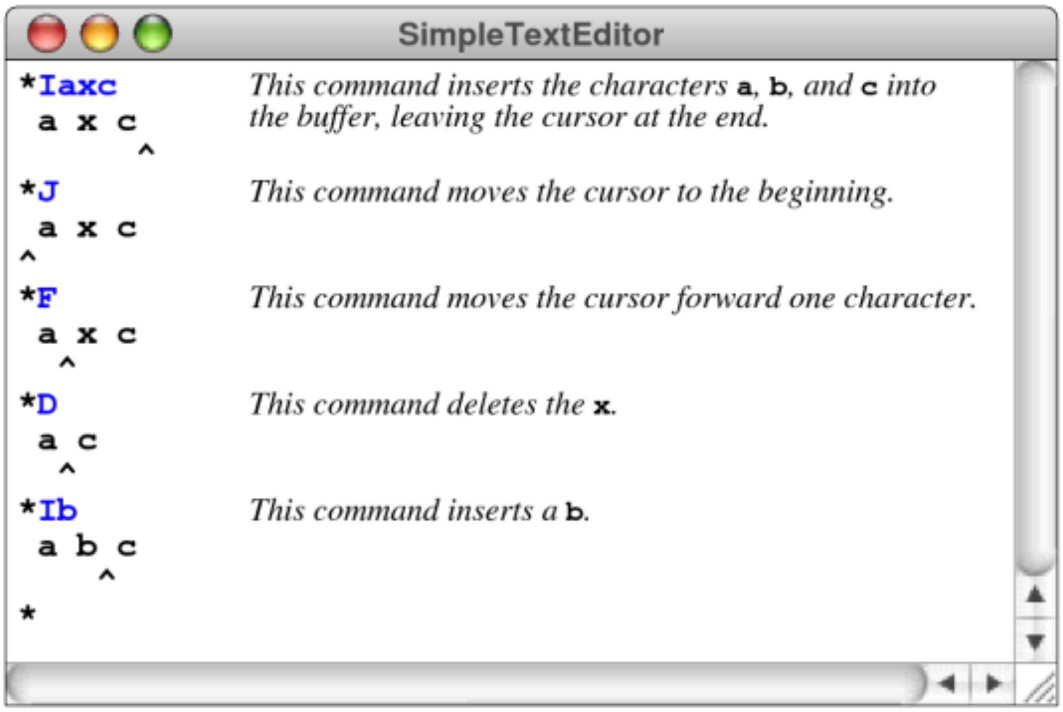


```
void P1() {
   EditorBuffer buffer;
   while (true) {
      string cmd = getLine("*");
      if (cmd != "") executeCommand(buffer, cmd);
   }
}
```

Any methods needed for simulating the process of

```
void executeCommand(EditorBuffer & buffer, string line);
void displayBuffer(EditorBuffer & buffer);
void printHelpText();
```

manipulating editors should be included such as
So that we can test your commands C and P like others as follows:



```
*Iaxc              This command inserts the characters a, b, and c into
 a x c             the buffer, leaving the cursor at the end.
      ^
*J                 This command moves the cursor to the beginning.
 a x c
^
*F                 This command moves the cursor forward one character.
 a x c
  ^
*D                 This command deletes the x.
 a c
  ^
*Ib                This command inserts a b.
 a b c
    ^
*
```

Q2:

You should write two files: pqueue_list.h and P2.cpp.

Of them, pqueue_list.h includes the declaration of class PriorityQueue and the corresponding implementations. You should use linked-list based implementation and includes all the methods in Figure 14-9 of the textbook. To consider priority of each element, you should follow the private section:

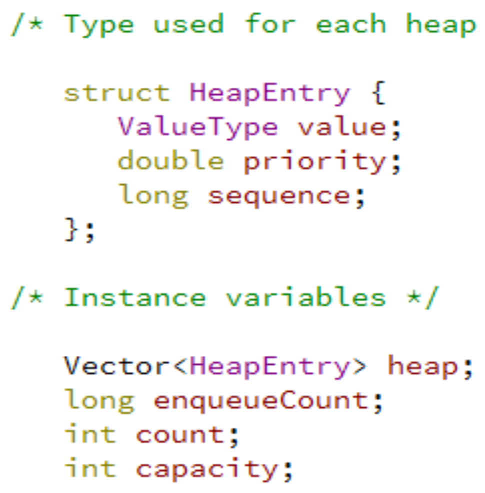
```
private:

/* Type for linked list cell */

   struct Cell {
      ValueType data;                 /* The data value              */
      double priority;                /* The priority of this value  */
      Cell *link;                     /* Link to the next cell       */
   };

/* Instance variables */

   Cell *head;                        /* Pointer to the cell at the head */
   int count;                         /* Number of elements in the queue */

/* Private method prototypes */

   void deepCopy(const PriorityQueue<ValueType> & src);
```

P2.cpp includes the test method void P2() which contains the test code. You should at least declare an instance of PriorityQueue with base type String and enqueue element (value and priority) <"A", 5>, <"B", 2>, <"C", 9>, <"D", 2> and <"E", 7>. Then dequeue these elements one by one and print them out (value and priority) to check the correctness of the order.

Q3:

You should write two files: pqueue_heap.h and P3.cpp.

Of them, pqueue_heap.h includes the declaration of class PriorityQueue and the corresponding implementations. You should use heap based implementation and includes all the methods in Figure 14-9 of the textbook. The instance variables in private section of the class are as follows.



```
/* Type used for each heap entry */

   struct HeapEntry {
      ValueType value;
      double priority;
      long sequence;
   };

/* Instance variables */

   Vector<HeapEntry> heap;
   long enqueueCount;
   int count;
   int capacity;
```

P3.cpp includes the test method void P3() which contains the test code. You should at least declare an instance of PriorityQueue with base type String and enqueue element (value and priority) <“A”, 5>, <“B”, 2>, <“C”, 9>, <“D”, 2> and <“E”, 7>. Then dequeue these elements one by one and print them out (value and priority) to check the correctness of the order.

Q4:

You should write two files: graphtypes.h and P4.cpp.

Of them, graphtypes.h contains a low-level graph abstraction using a set of arcs to represent connectivity between nodes, which refers to Figure 18-3 of the textbook.

P4.cpp should include some helper functions and test code in void P4(). These helper functions include:

- **addNode, addArc and initiateGraph**

```
Node *addNode(SimpleGraph & g, string name);
Arc *addArc(SimpleGraph & g, Node *n1, Node *n2, double cost);
```

```
void initiateGraph(SimpleGraph &g);
```

- **writeGraph**

```
/*
 * Function: writeGraph
 * Usage: writeGraph(g, outfile);
 * ------------------------------
 * Writes out a simple description of the graph g to the output stream.
 */

void writeGraph(const SimpleGraph & g, ostream & out) {
```

- **pathExistsBFS**

```
/*
 * Function: pathExistsBFS
 * Usage: pathExistsBFS(Node *n1, Node *n2)
 * -------------------------------------------
 * Find all paths from n1 to n2 using Breadth-First search
 */
bool pathExistsBFS(Node *n1, Node *n2) {
```

- **pathExistsDFS**

```
/*
 * Function: pathExistsDFS
 * Usage: pathExistsDFS(Node *n1, Node *n2)
 * -------------------------------------------
 * Find all paths from n1 to n2 using Depth-First search
 */
bool pathExistsDFS(Node *n1, Node *n2) {
```

- printAllPathsFound

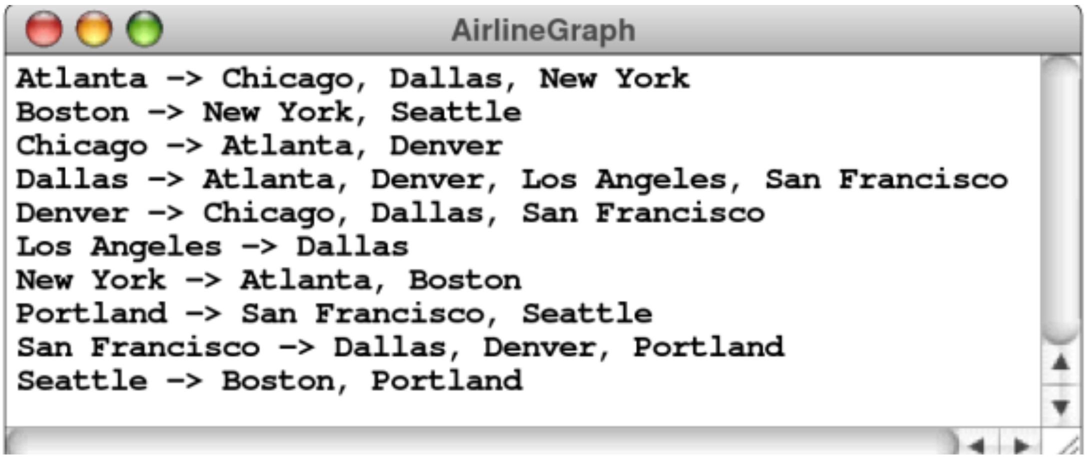
Examples: There are two paths from XX to XX:

1. XX -> XX -> XX…
2. XX -> XX -> XX…

Examples: There is no paths from XX to XX !

Note that for the helper functions, you can change the method signature such as return type and argument type to be suitable for your program.

The test method void P4(), should at first construct a Simple Graph and print it out by iteratively accessing all edges as follows:



```
Atlanta -> Chicago, Dallas, New York
Boston -> New York, Seattle
Chicago -> Atlanta, Denver
Dallas -> Atlanta, Denver, Los Angeles, San Francisco
Denver -> Chicago, Dallas, San Francisco
Los Angeles -> Dallas
New York -> Atlanta, Boston
Portland -> San Francisco, Seattle
San Francisco -> Dallas, Denver, Portland
Seattle -> Boston, Portland
```

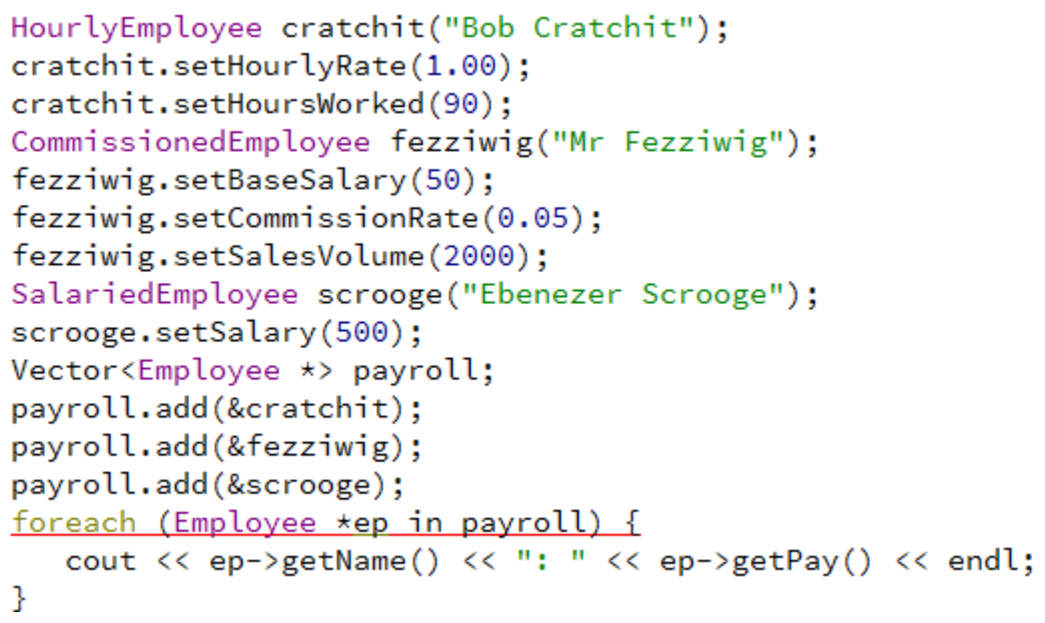
Then, according to user input from console by specifying the start position (airport name, For example, “Altanta”) and the end position (airport name, for example, “Denver”), you test code should use DFS and BFS to find all paths and print them out respectively.

Q5:

You should write three files: Employee.h, Empolyee.cpp and P5.cpp.

Of them, Employee.h and Employee.cpp should include the declaration of class Employee and its subclasses including HourlyEmployee, CommissionedEmployee, and SalariedEmployee, and their corresponding implementations.

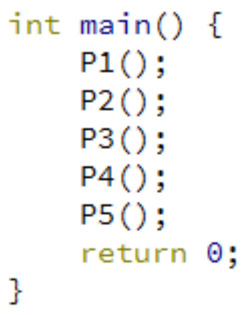
Test code in P5.cpp should be void P5() contains：
and test your program.



```
HourlyEmployee cratchit("Bob Cratchit");
cratchit.setHourlyRate(1.00);
cratchit.setHoursWorked(90);
CommissionedEmployee fezziwig("Mr Fezziwig");
fezziwig.setBaseSalary(50);
fezziwig.setCommissionRate(0.05);
fezziwig.setSalesVolume(2000);
SalariedEmployee scrooge("Ebenezer Scrooge");
scrooge.setSalary(500);
Vector<Employee *> payroll;
payroll.add(&cratchit);
payroll.add(&fezziwig);
payroll.add(&scrooge);
foreach (Employee *ep in payroll) {
   cout << ep->getName() << ": " << ep->getPay() << endl;
}
```

Assignment Submission

Please put all required files in a single QT project which is stated in the first line of each Question Requirement Write a main.cpp file to invoke different questions as follows:



```
int main() {
    P1();
    P2();
    P3();
    P4();
    P5();
    return 0;
}
```

After you test all your questions, zip the whole project in one file named XXX.zip (XXX is your student ID and your name is not required) and then submit it to the Moodle system.

Marking scheme

For each question:

- **0.4 Marks will be given to students who have submitted the program on time.**
- **0.4 Marks will be given to students who wrote the program that meet all the requirements of the questions**
- **0.4 Marks will be given to students who programs that can be compiled without errors and warnings**
- **0.4 Marks will be given to students whose programs produce the correct output if their programs can be compiled (Each question worth 0.1 mark).**
- **0.4 Marks will be given to students who demonstrate good programming habit and style.**

Q&A

1. How to check whether the assignment is submitted on time?

We directly check it through Moodle system

2. How to check whether the assignment meets the requirement?

   a. A zip file containing the whole project is submitted
   b. Zip filename is valid (studentID.zip)
   c. All the required files stated in Each Question Requirement should be included with correct name.
   d. The Main.cpp with correct content should be included.
   e. The content of each file should satisfy the requirement. For example, the declaration of Editorbuffer class should be in buffer.h and its implementation should be in buffer.cpp.
   f. No other additional files are included

3. How to check whether the assignment can be compiled?

   Note that we strongly suggest you to use the QT IDE since your code tested on other IDE may not work on

QT. If you want to use other IDE, please test your code again on QT to avoid such errors. We also strongly suggest you to use the Standford Library if necessary instead of STL or other similar libraries.

If the QT IDE generates any errors when building your project, 0.4 mark will be missed for each question. If any warnings are produced but your code can run, 0.4 mark will be missed for all 5 questions.

4. How to check the assignment gives correct output?

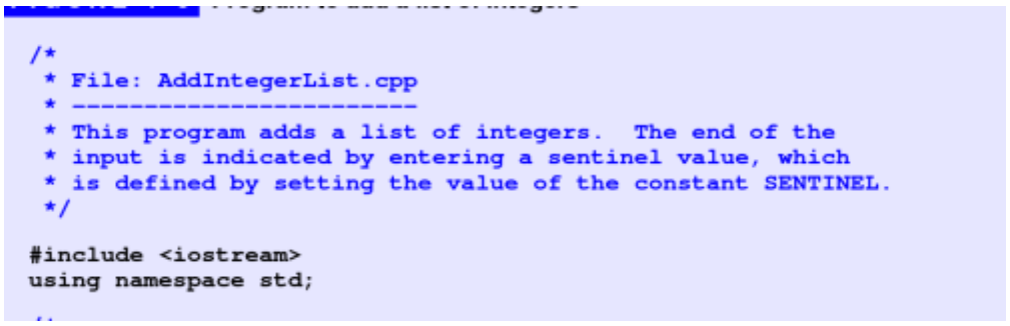
We will run your code if it can be compiled (warnings are tolerated). You test code should be correct and output is also correct. User-friendly test code and output is preferred.

5. How to check the programming style?

- Comments.

  You should include three kinds of comments in your code:

  a. File comments in the beginning of each code file, for example,



```
/*
 * File: AddIntegerList.cpp
 * ------------------------
 * This program adds a list of integers.  The end of the
 * input is indicated by entering a sentinel value, which
 * is defined by setting the value of the constant SENTINEL.
 */

#include <iostream>
using namespace std;
```

b. Function comments just before each function implementation. For example，

```
/*
 * Function: raiseToPower
 * Usage: int p = raiseToPower(n, k);
 * ----------------------------------
 * Returns the integer n raised to the kth power.
 */

int raiseToPower(int n, int k) {
   int result = 1;
   for (int i = 0; i < k; i++) {
      result *= n;
   }
   return result;
}
```

c. Statement comments on the right side of some statements if necessary. You do not have to write comments for each statement.

- Code layout.

  The following points should be pay attention to:

  a. The indentation used

  b. {} used for function, for structure, while structure, switch structure.

  c. meaningful variable name.

  d. dynamic allocated memory should be destroyed after used

  e. class declaration and implementation

We strongly suggest you to adhere to the coding style used in our textbook, which will be followed by us to check your code.