CSCI204 Assignment 3

Due: by May 28, 14:30. Resubmission is open: till June 4, 14:30
Marks: 16 marks

Objective: get practical experience in using:

- string and binary data
- advanced I/O and manipulator;
- templates and STL

General Requirements

- Follow the common principles of OO programming when you design your classes
- Put your name, student number at the beginning of each source file

```
/*-------------------------------------------------------
Student's Name:
Student's email address:
Laboratory group (group code and time):
Purpose of this assignment:
-------------------------------------------------------*/
```

- Add comments to the source code to make your solution easier to follow

---

Assignment requirements:

There are three tasks in this assignment.

Task 1: Template (8 marks)

`1. ListNode`

Define and implement a **template class `ListNode`** in a namespace **`MYLIB`** in a file **`OrderedList.h`**, which has **a template data member** and **two pointers** that point to the *previous* **`ListNode`** object and *next* **`ListNode`** object. Define and implement necessary constructor(s) and other member functions in the file **`OrderedList.h`**.

2. OrderedList

Define and implement a **template class `OrderedList`** as a **container** in a file **`OrderedList.h`** that can be used to store template data in a Doubly Linked List (DLL) with nodes ordered by the data values from the smallest one to the biggest one. Define two data members **`head`** and **`tail`** as pointers of **`ListNode` type**. They point to the head and tail of a DLL.

3. iterator

Define a **nested class `iterator`** in the template class **`OrderedList` that can be used to traversal the DLL**. It contains a data member of a **`ListNode`** pointer points to a node in the DLL. Define **constructors**, overloading operators, such as ++ (pre-fix and post-fix increment operator) that change an **iterator** points to the next node, ***** (dereference operator) returns the data that the iterator points to, and **!=** (not equal) to compare two **iterators** in the class **iterator**.

Define a **constructor**, **destructor** for the template class **`OrderedList`**. Define a member function **`begin()`** for the template class **`OrderedList`** that returns an iterator object points to the beginning of the DLL. Define a member function **`end()`** for the template class **`OrderedList`** that returns an iterator object points to the end of the DLL. Define a member function **`insert(const T &)`** for the template class **`OrderedList`** that take a template data as a parameter, insert a new node with the data value into the correct location in the DLL.

Implement member functions for the template class **`OrderedList`** and nested class **`iterator`** in the file **`OrderedList.h`**.

4. Student

Define a class **`Student`** in a file **`Student.h`**, which contains student number, name and email. Define overloading insertion operator (<<) to print out data members of a **`Student`** object. Define overloading extraction operator (>>) to get input data to a **`Student`** object. Define overloading comparison operator <= (or <) that compare two **`Student`**s' objects by their emails.

Define necessary member functions, such as constructors, etc., for the class **`Student`**.
Implement member functions, friend input operator, output operator for the class **`Student`** in a file **`Student.cpp`**.

Download a file **`task1Main.cpp`** to test the DLL by different data (**integer**s, **double**s and **`Student`** objects). You will get data from the keyboard, add them into the ordered DLL, then print out results.

Testing:

You can compile the task 1 by

```
CC -o task1 task1Main.cpp Student.cpp
```

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

```
./task1
How many integers? 5
Input an integer: 18
Input an integer: 2
Input an integer: 13
Input an integer: 9
Input an integer: 7
Output integers:
2 7 9 13 18
How many doubles? 8
Input a double: 17.5
Input a double: 12.5
Input a double: 11.3
Input a double: 19.8
Input a double: 18.4
Input a double: 10.2
Input a double: 21.4
Input a double: 22.2
Output doubles:
10.2 11.3 12.5 17.5 18.4 19.8 22.2 31.4
How many student records? 4
Input number: 1234567
Input name: Cart Dong
Input email: cd28@uow.edu.au
Input number: 1234568
Input name: Bob Smith
Input email: bs36@uow.edu.au
Input number: 1234570
Input name: Mark Twain
Input email: mt12@uow.edu.au
Input number: 1234571
Input name: Alice Montage
Input email: am12@uow.edu.au
Output students:
1234571, Alice Montage, am12@uow.edu.au
1234568, Bob Smith, bs36@uow.edu.au
1234567, Cart Dong, cd28@uow.edu.au
1234570, Mark Twain, mt12@uow.edu.au
```

You can download the testing file **input1.txt** from Moodle and save it into your working directory, test your program by using following method:

```
./task1 < input1.txt
```

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

Task 2: file I/O and manipulations (5 marks)

Download the source code **Date.h** and **Date.cpp** from **Moodle** for this task.

1. Account

Define a class **Account** in a file **Account.h** that contains data members **account number**, **name**, **sex**, **date of birth (Date type)**, **address** and **account balance**. Define **constructor(s)**, overloading operators, include assignment operator (=), less than and equals to operator (<=), insertion operator (<<), and extraction operator (>>) for the class **Account**. Define other necessary member functions.

Implement member functions and overloading operators for the class **Account** in a file **Account.cpp**.

Define you own manipulator **Currency** that takes two integers as **width** and **precision** for the output of currency in the file **Account.h**. Implement the manipulator **Currency** in the file **Account.cpp**. The manipulator Currency will be used in the insertion operator for Account balance. The manipulator Currency will set output currency symbol as "$", the width of output currency, the precision of currency and filled by zeros ('0's) if the width of currency is not long enough.

Hint: Define fixed size char arrays instead of strings for some data members defined in the class Account (such as name, address).

Hint: Use iomanip and Currency that defined to generate formatted outputs for the account records.

2. AccountManagement

Define a class **AccountManagement** in a file **AccountManagement.h** that contains a data member of a container **OrderedList**, which will be used to store account records. Define member functions:

- **`loadData(const char *)`** will load Account record from a given text file and store the records in the container of **`OrderedList`**.
- **`displayData()`** will use iterator of **`OrderedList`** object to traversal the DLL and display formatted output data of accounts.
- **`saveData(const char *)`** will save the accounts from DLL to a given binary file.

Implement the member functions in a file **`AccountManagement.cpp`**.

Download a file **`task2Main.cpp`** to test your task 2.

Testing:

Use CC to compile the source files by

```
CC –o task2 task2main.cpp Account.cpp Date.cpp AccountManagement.cpp
```

and run the program by

```
./task2 accounts.txt accounts.dat
```

The output records on the screen can be found in a text file **`output2.txt`**.

The input text file **`accounts.txt`** can be downloaded from Moodle. The sorted Account records will be saved in a binary file **`accounts.dat`**.

Note: Your solutions of task 2 should work on different testing data / files.

Task3: STL map and iterator (3 marks)

Use the source code files **`Date.h`**, **`Date.cpp`**, **`Account.h`**, and **`Account.cpp`** that used in task 2.

1. AccountMap

Define a class **`AccountMap`** in a file **`AccountMap.h`**. It contains a data member, which is a **multimap** container that can be used to store Account records. The key of the container is a char pointer of account's name. You will define a **comparison function `CompareCharArrays`** to compare two **`char`** arrays in the file **`AccountMap.h`** for the **multimap** object. For example:

```
multimap<const char *, Account, CompareCharArrays> accounts
```

Define a **destructor** for the class **`AccountMap`** to release dynamic memory allocated for the container to avoid memory leaks.

Define a member function **`loadData(const char *)`** for the class **`AccountMap`** that load account records from a given binary file (created in task 2), insert the records into the **multimap** container.

Define a member function **`displayData()`** for the class **`AccountMap`** that use **iterator** of the container to display all records.

Implement the member functions in a file **`AccountMap.cpp`**.

Download a file **`task3Main.cpp`** from Moodle to test your task 3.

Testing:

Use CC to compile the source files by

```
CC –o task3 task3Main.cpp Account.cpp Date.cpp AccountMap.cpp
```

and run the program by

```
./task3 accounts.dat
```

The binary file **`accounts.dat`** is generated by your task 2. The outputs of this task look like the results in a text file **`output3.txt`**.

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

Submission:

All assignments must be submitted electronically via the submit system. For this assignment you must submit all the files via the command (**in one line**):

```
$ submit -u your_user_name -c CSCI204 -a 3 OrderedList.h Student.h
Student.cpp Account.h Account.cpp AccountManagement.h
AccountManagement.cpp AccountMap.h AccountMap.cpp
```

and enter 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**.

After submit your assignment successfully, please check your email of confirmation. You should keep the email for the reference.

NOTES:

1. SUBMIT AS EARLY AS POSSIBLE. YOU CAN RESUBMIT LATER IF NECESSARY. Only the latest submission will be marked.

3. SUBMISSION VIA EMAIL IS NOT ACCEPTABLE. YOU HAVE TO USE SUBMIT COMMAND TO SUBMIT YOUR WORK.

4. ASSIGNMENT WITHOUT PROPERLY FILLED ASSIGNMENT HEADERS IN SUBMITTED FILES WILL NOT BE MARKED

4. The submitted file names must be the same as in the submission example. Files with other names will not be tested by the submit system and therefore will not be marked.

5. Enquiries about the marks can only be made within a maximum of 1 week after the assignment results are published. After one week the marks cannot be changed.

6. 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. Plagiarism will result in a **FAIL** grade being recorded for that assessment task.