CSCI204 Assignment 2

(Total 15 marks, Due by 11:59 pm sharp on Sunday, 11 May, 2014)

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.
- Manipulate string data.

Tasks:

Task 1: (6.5 marks)

A binary code is a type of code that every entry is either 0 or 1. It is very useful in computer science, cryptography, mathematic and engineering. In the computer system, negative binary code represented by its complement value. For example:
A decimal number $(9)_{10}$ *is* $(01001)_2$ *of binary number. A decimal number* $(-9)_{10}$ *is* $(-01001)_2$*. In the computer system, the negative value usually represented by its complement value,* $(10111)_2$*, the most significant (highest) bit is 1, indicates the sign of the value is negative. When the leftmost bit is 0, the sign of the value is positive.*
To get the complement value of a negative binary number, we can invert every bit, add 1 on the inverted value of the binary code. For example:
$(-9)_{10} = (-01001)_2$*, to invert binary code, we have* $(10110)_2$*, to add with 1, we have*

```
   10110
+      1
________
   10111
```

The complement code of $(-01001)_2$ *is* $(10111)_2$*.*

In this task, you will define a class **BinaryCode** in a file **BinaryCode.h** and implement the C++ program code in a file **BinaryCode.cpp**.
In the class **BinaryCode**, declare a dynamic array (such as a char pointer or an short integer pointer) to store a binary code, an integer length to store the length of the binary code (includes the sign bit-the most significant bit). Define the following functions in the class BinaryCode.

- In the class BinaryCode, you will define default constructor, copy constructor and other initialization constructors to set initial value for a BinaryCode object;
- In the class BinaryCode, define following overloading operators:
  - Extraction operator (>>) to get input binary code from keyboard and save it in a BinaryCode object;
  - Insertion operator (<<) to print out the a BinaryCode stream;
  - Assignment operator(=) to make a deep copy of BinaryCode object.
  - Invert operator(!) to invert a BinaryCode object.
  - Complement operator(~) to make a complement BinaryCode object. A complement of a binary code is its invert plus one.
  - Addition operator (+) to compute two BinaryCode objects' addition. You should extend two binary streams to the maximum length of both streams plus one so that the significant bits addition won't be missed. For example: *Two binary streams* $(011011)_2 + (01111)_2$ *will be extended* $(0011011)_2 + (0001111)_2$

    ```
      0011011
    + 0001111
    _________
      0101010
    ```

  - Addition assignment operator (+=) to compute two BinaryCode objects' addition.
  - Subtraction operation (-) to compute two BinaryCode objects' subtraction. The subtraction of two BinaryCode objects is defined as:

binaryCode1 – binaryCode2 = binaryCode1 + ~(binaryCode2), where ~ is complement operator.

- Subtraction assignment operator (-=) to compute two BinaryCode objects' subtraction.
- Multiplication operator (*) to compute two BinaryCode objects' multiplication. To compute the multiplication of two BinaryCode objects, each BinaryCode object should be extended as a signed integer. That is to append number of bits to the left side of a binary stream. Each appended bit's value should be the value of the most significant bit's value of the original binary stream. The new length of binary stream is addition of the two binary streams' length (excludes sign bits).
  *For example, to compute $(01001)_2$ * $(10111)_2$, we will extend two binary streams to 10 bits (5+5 bits), as $(0000001001)_2$ * $(1111110111)_2$. Then compute*
  ```
       1111110111
     * 0000001001
  _______________
       1111110111
    + 1111110111
  _______________
    10001110101111.
  ```
  Keep the last (rightmost) 10 bits as the results, which is $(1110101111)_2$, the complement binary code of $(-81)_{10}$.
- Multiplication assignment operator (*=) to compute two BinaryCodes' multiplication.
- Shift left operator (<<) to shift a BinaryCodes to left a number of bits. For example:
  $(01011)_2$ << 3 will be $(01011000)_2$. It is like $(01011)_2 * 2^3$.
- Shift right operator (>>) to use arithmetic shift method shift a BinaryCode to right a number of bits. For example:
  $(01011)_2$ >> 2 will be $(010)_2$. It is like $(01011)_2 / 2^2$.
  $(1010)_2$ >> 2 will be $(110)_2$.
- Other necessary member functions.

Implement **main()** in a file **task1Main.cpp** to test the functions and operators that defined above (See the Testing examples of the task for more details).

Testing:

Use CC to compile the source files on banshee by

```
$ CC –o task1 task1Main.cpp BinaryCode.cpp
```

You can test the task by run the program

```
$ task1
```

and input data that required. Your program will print out results like following (Red data means input from keyboard):

Testing example 1:

```
Input a decimal number for bc1: 1027
bc1 = 010000000011
!bc1 = 101111111100
~bc1 = 101111111101
bc1 << 4 = 0100000000110000
bc1 >> 2 = 0100000000
Input a binary string for bc2: 100110111
bc2 = 0100110111
!bc2 = 1011001000
~bc2 = 1011001001
bc2 << 3 = 0100110111000
bc2 >> 4 = 010011
bc1 + bc2 = 010100111010
bc1 - bc2 = 01011001100
bc1 * bc2 = 01001101111110100101
bc3 = bc1 = 010000000011
bc3 += bc2 = 010100111010
bc3 = bc1 = 010000000011
bc3 -= bc2 = 01011001100
bc3 = bc1 = 010000000011
bc3 *= bc2 = 01001101111110100101
```

Testing example 2:

```
Input a decimal number for bc1: 852
bc1 = 01101010100
!bc1 = 10010101011
~bc1 = 10010101100
bc1 << 4 = 011010101000000
bc1 >> 2 = 011010101
Input a binary string for bc2: -10100110101001
bc2 = 101011001010111
!bc2 = 010100110101000
~bc2 = 010100110101001
bc2 << 3 = 101011001010111000
bc2 >> 4 = 10101100101
bc1 + bc2 = 101100110101011
bc1 - bc2 = 010110011111101
bc1 * bc2 = 10111010101011001100011 00
bc3 = bc1 = 01101010100
bc3 += bc2 = 101100110101011
bc3 = bc1 = 01101010100
bc3 -= bc2 = 010110011111101
bc3 = bc1 = 01101010100
bc3 *= bc2 = 1011101010101100110001100
```

Testing example 3:

```
Input a decimal number for bc1: -2068
bc1 = 1011111101100
!bc1 = 0100000010011
~bc1 = 0100000010100
bc1 << 4 = 10111111011000000
bc1 >> 2 = 10111111011
Input a binary string for bc2: -1101110010111001
bc2 = 10010001101000111
!bc2 = 01101110010111000
~bc2 = 01101110010111001
bc2 << 3 = 10010001101000111000
bc2 >> 4 = 1001000110100
bc1 + bc2 = 10001101100110011
bc1 - bc2 = 01101010010100101
bc1 * bc2 = 0110111101110000011001110100
bc3 = bc1 = 1011111101100
bc3 += bc2 = 10001101100110011
bc3 = bc1 = 1011111101100
bc3 -= bc2 = 01101010010100101
bc3 = bc1 = 1011111101100
bc3 *= bc2 = 0110111101110000011001110100
```

Note: Your program should work on different testing data.

Task2: (3 marks)

In this task, you will define a class **Date** in a name space **MyLib** in a file **Date.h**. Define data members include day, month and year in the class Date. Define member functions in the class Date. Implement member functions in a file **Date.cpp.** The member functions include

- A function setDate(int, int, int) takes three parameters of day, month and year and set date for the Date object.
- A function setDate(const std::string &) takes a string of date with format "DD/MM/YYYY" and convert the string date to the Date object. "DD/MM/YYYY" represents 2 digits value of day, 2 digits value of month and 4 digits value of year with "/" between them.
- A function validateDate() validates the date of the Date object. If day is invalid, throw an error message "Invalid day. Day should between 1 and xx." (where xx is the maximum day for that month). If month is invalid, throw an error message "Invalid month". If year is invalid, such as less than zero, throw an error message "Invalid year".

- A function toString() converts day, month and year to a string with format "DD/MM/YYYY".
- Other necessary function to return date information.

Implement C++ main() function in a file **task2Main.cpp** that test the member functions specified above. It will get date information and set the date to Date objects, validate the date, catch the exception if the date is invalid. See the Testing of this task for details.

Testing:

Use CC to compile the source files on banshee by

```
$ CC –o task2 task2Main.cpp Date.cpp
```

You can test the task by run the program

```
$ task2
```

and input data that required. Your program will print out results like following (Red data means input from keyboard):

```
Input date (DD/MM/YYYY): 29/02/2014
Invalid day. Day should be between 1 and 28.
```

Testing example 2:

```
Input date (DD/MM/YYYY): 30/04/2001
30/04/2001
Input day: 32
Input month: 5
Input year: 2010
Invalid day. Day should be between 1 and 31.
```

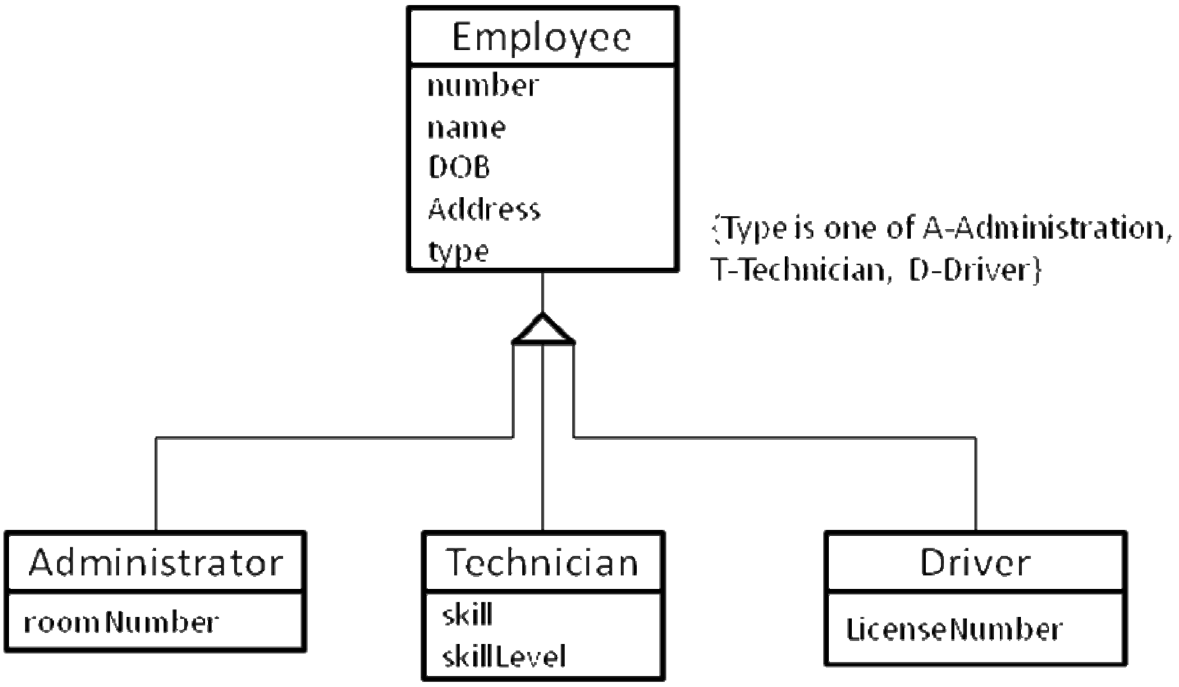
Testing example 3:

```
Input date (DD/MM/YYYY): 03/11/2011
03/11/2011
Input day:2 9
Input month: 2
Input year: 2000
29/02/2000
```

Task3: (5.5 marks)

In this task, you will define and implement inheritance classes.

Define the diagrams of the classes below.



Define a *base class* **Employee** in a file **Employee.h** that described in the diagrams above. Define an extraction operator (>>) to get input values for an Employee from the keyboard or a file; insertion operator (<<) to print out the Employee information. The data member **DOB is the Date type which has been defined in Task 2**. Define necessary constructor(s) and other member functions for the class. Implement insertion operator, extraction operator, constructor and other member functions in a file **Employee.cpp**.

Define a *derived class* **Administrator** in a file **Administrator.h** that described in the diagrams above. Define an extraction operator (>>) to get input values for an Administrator from the keyboard or a file; insertion operator (<<) to print out the Administrator's information. ***You should call the extraction operator and insertion operator defined in the base class for those base class's data members' input / output.*** In the extraction operator, you will catch the exception if the input DOB is invalid. When the exception has been caught, write the administrator information with the error message into a log file **log.txt**.
Note: Do not terminate the program.
Define necessary constructor(s) and other member functions for the class. Implement insertion operator, extraction operator, constructor and other member functions in the file **Administrator.cpp**.

Define a *derived class* **Technician** in a file **Technician.h** that described in the diagrams above. Define Extraction operator (>>) to get input values for a Technician from the keyboard or a file; insertion operator (<<) to print out the Technician's information. ***You***

should call the extraction operator and insertion operator defined in the base class for those data members' input / output of the base class.
In the extraction operator, you will catch the exception if the input DOB is invalid. When the exception has been caught, write the Technician information with the error message into a log file **log.txt**.
Note: Do not terminate the program.
Define necessary constructor(s) and other member functions for the class. Implement insertion operator, extraction operator, constructor and other member functions in the file **Technician.cpp**.

Define a *derived class* **Driver** in a file **Driver.h** that described in the diagrams above. Define Extraction operator (>>) to get input values for a Driver from the keyboard or a file; insertion operator (<<) to print out the Driver's information. ***You should call the extraction operator and insertion operator defined in the base class for those data members' input / output of the base class.***
In the extraction operator, you will catch the exception if the input DOB is invalid. When the exception has been caught, write the Technician information with the error message into a log file **log.txt**.
Note: Do not terminate the program.
Define necessary constructor(s) and other member functions for the class. Implement insertion operator, extraction operator, constructor and other member functions in the file **Technician.cpp**.

Define a class **EmployeeManagemet** in a file **EmployeeManagement.h**. Define two data members: **a dynamic array of Employee pointers.** Each element is a base class Employee pointer that points to a derived class object. Define an integer **length** that stores the total number of Employee objects.
Define constructor(s), destructor and following member functions:

- **loadEmployees(const char *)**: Load all employee records from a given text file and store them in the dynamic array by using extraction operators (>>) for the derived classes Administrator, Technician, Driver and base class Employee.
  Note: The data for employee contain errors, such as wrong date (catch the error) and wrong type of employee. Add the error messages in the file log.txt. Do not stop until the rest of data have been loaded into the dynamic memory.
- **manageEmployees()**:displays a menu and gets input choices, then call the corresponding member functions to perform the actions.
- **displayEmployees()**: Use insertion operators (defined for the derived classes Administrator, Technician, Driver and base class Employee) to display required Employees' information that stored in the array.
- **updateEmployee()**: Update an Employee's information.
- **saveEmployees()**: Save employee data to a given text file.
- Other necessary member functions.

Implement all the member functions defined for the class **EmployeeManagement** in a file **EmployeeManagement.cpp**.

The data format of an employee in a text file are looked like

```
A,1234567,John Wood,12/02/1965,21 Victoria street Depto NSW 2530,311
T,1122334,Taylor Smith,08/04/1978,32 Smith street Wollongong NSW 2500,C++,7
D,1234123,Bob Bright,28/09/1983,121 Princess highway Wollongong NSW 2500,1320671
```

Where ‘A’ represents Administrator type, ‘T’ represents Technician type, and ‘D’ represents Driver type. You can download a text file **employees.txt** for your testing.

Implement C++ main() functions in a file **task3Main.cpp** to get text file name from the command line, declare an instance of EmployeeManagement; then call the instance function **loadEmployees()** to load employee records, call the instance function **manageEmployees()** to perform the activities. Finally save the data to a text file. See the Testing of this task for more details.

Note: Your program should work on different testing data.

Testing:

Use CC to compile the source files by

```
$ CC –o task3 task3Main.cpp EmployeeManagement.cpp Employee.cpp
Administrator.cpp Technician.cpp Driver.cpp Date.cpp
```

and run the program by

```
$ ./task3 employees.txt
```

When the program starts, it loads data from employees.txt (11 records for this testing file) and displays records (9 correct records) that loaded, then displays menu and get input data (red data denote input data from the keyboard. <Enter> means the Enter key).

```
A,1234567,John,12/02/1965,21 Victoria street Depto NSW 2530,311
T,1122334,Taylor,08/04/1978,32 Smith street Wollongong NSW 2500,C++,7
D,1234123,Bob,28/09/1983,121 Princess highway Wollongong NSW 2500,1320671
A,1234568,Marry,20/04/1985,34 Beach Road Fairy Meadow NSW 2500,311
D,1122346,Michael,10/10/1970,24 Mall street Wollongong NSW 2500,1230781
A,1234590,Rose,20/11/1990,232 Princess highway Unanderra NSW 2526,312
D,1234128,Adam,10/12/1987,12 Princess highway Wollongong NSW 2500,1320671
T,1122348,Alvin,15/07/1988,48 Spring road Wollongong NSW 2500,Java,8
T,1122350,Alice,17/10/1975,33 Autumn street Wollongong NSW 2500,Database,7

1. Display employees
2. Update an employee
0. Exit
Your choice: 1
1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 1
Input an employee number: 1234567
A,1234567,John,12/02/1965,21 Victoria street Depto NSW 2530,311
```

```
1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 1
Input an employee number: 1122348
T,1122348,Alvin,15/07/1988,48 Spring road Wollongong NSW 2500,Java,8

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 1
Input an employee number: 1122350
T,1122350,Alice,17/10/1975,33 Autumn street Wollongong NSW 2500,Database,7

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 2
Input an employee type: D
D,1234123,Bob,28/09/1983,121 Princess highway Wollongong NSW 2500,1320671
D,1122346,Michael,10/10/1970,24 Mall street Wollongong NSW 2500,1230781
D,1234128,Adam,10/12/1987,12 Princess highway Wollongong NSW 2500,1320671

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 2
Input an employee type: T
T,1122334,Taylor,08/04/1978,32 Smith street Wollongong NSW 2500,C++,7
T,1122348,Alvin,15/07/1988,48 Spring road Wollongong NSW 2500,Java,8
T,1122350,Alice,17/10/1975,33 Autumn street Wollongong NSW 2500,Database,7

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 2
Input an employee type: A
A,1234567,John,12/02/1965,21 Victoria street Depto NSW 2530,311
A,1234568,Marry,20/04/1985,34 Beach Road Fairy Meadow NSW 2500,311
A,1234590,Rose,20/11/1990,232 Princess highway Unanderra NSW 2526,312

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 0
1. Display employees
```

```
2. Update an employee
0. Exit
Your choice: 2
Employee number: 1234567
A,1234567,John,12/02/1965,21 Victoria street Depto NSW 2530,311
Input new address (Directly Enter for no change): 107 Gipps road Wollongong NSW
2500
New office (0 for no change): 312

1. Display employees
2. Update an employee
0. Exit
Your choice: 1
1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 1
Input an employee number: 1234567
A,1234567,John,12/02/1965,107 Gipps road Wollongong NSW 2500,312

1. Find an employee by a number
2. Find all employees by a type
0. Exit
Your choice: 0
1. Display employees
2. Update an employee
0. Exit
Your choice: 2
Employee number: 1122350
T,1122350,Alice,17/10/1975,33 Autumn street Wollongong NSW 2500,Database,7
Input new address (Directly Enter for no change): <Enter>
New skill level(0 for no change): 9

1. Display employees
2. Update an employee
0. Exit
Your choice: 2
Employee number: 1234128
D,1234128,Adam,10/12/1987,12 Princess highway Wollongong NSW 2500,1320671
Input new address (Directly Enter for no change): 321 Winter road Sydney NSW 2001

1. Display employees
2. Update an employee
0. Exit
Your choice: 0
Input a file name to save the data: newemployees.txt
```

```
Bye
```

The log file **log.txt** contains the following messages:

```
In the text file line 3 has error: Invalid day. Day should between 1 and 28.
In the text file line 9 has an error: 'S' is a wrong employee type
```

The text file **newemployees.txt** contains following data:

```
A,1234567,John,12/02/1965,107 Gipps road Wollongong NSW 2500,312
T,1122334,Taylor,08/04/1978,32 Smith street Wollongong NSW 2500,C++,7
D,1234123,Bob,28/09/1983,121 Princess highway Wollongong NSW 2500,1320671
A,1234568,Marry,20/04/1985,34 Beach Road Fairy Meadow NSW 2500,311
D,1122346,Michael,10/10/1970,24 Mall street Wollongong NSW 2500,1230781
A,1234590,Rose,20/11/1990,232 Princess highway Unanderra NSW 2526,312
D,1234128,Adam,10/12/1987,321 Winter road Sydney NSW 2001,1320671
T,1122348,Alvin,15/07/1988,48 Spring road Wollongong NSW 2500,Java,8
T,1122350,Alice,17/10/1975,33 Autumn street Wollongong NSW 2500,Database,9
```

Submission

This assignment is due by 11.59 pm (sharp) on Sunday 11 May, 2014.

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 2 task1Main.cpp BinaryCode.h
BinaryCode.cpp task2Main.cpp Date.h Date.cpp task3Main.cpp Employee.h
Employee.cpp Administrator.h Administrator.cpp Technician.h Technician.cpp Driver.h
Driver.cpp EmployeeManagement.h EmployeeManagement.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 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% 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 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