C Programming Language II — Final Exam

Cheng-Wen Ko.Ph.D., Associate Professor
E-Mail: cwko@cse.nsysu.edu.tw
Department of Computer Science and Engineering
National Sun Yat-sen University

***** Please read the following remarks carefully before you start this exam. *****

- .cpp file: [Question number].cpp (for example: the first question is 1.cpp)
- Execution file: [Question number] (example: 1)
- Place all files (include your "makefile") in the folder of your student ID (example: B1130400XX)
- Compress your folder to [Student ID].ZIP
- **Please make sure that you have included complete files in the ZIP file.**
- **Please check if your ZIP file can be successfully decompressed.**
- **If multiple uploads are found, only the latest upload will be reviewed for this exam.**
- **In additional to the conditions listed in the <Grading>, any compilation errors will result in deduction of your grade.**
- **Make sure that your "makefile" can be successfully compiled, or you'll get zero point.**
- **You are only allowed to use Ubuntu and text editors in Ubuntu during the exam. Otherwise, 50% of the deduction of your grade will be applied to any usage of other OS and its applications.**

1. (75%)

Complete the class definition and the member functions for the matrix operations.
Design a program step-by-step to accomplish the requirements as follows:

a. Define a class named "TwoD" that contains a default constructor that initializes both matrix size to 3, a parameterized constructor with two member variable of type int, and a destructor. You should use dynamic two-dimensional array declaration and determine the matrix size based on the input row and column numbers.
b. Define accessor and mutator functions that allow you to find out or change the value of row, column and the 2-D matrix.
c. Define a copy constructor that initializes an object using another object of the same class.
d. Overload the operator = so that it can be used to assign values of type TwoD.
e. Overload the operator + so that it can be used to perform matrix addition of two matrix with same size.
f. Following the previous question e, rewrite the operator + so that it can detect the mismatched matrix dimensions for incompatible operations. (Hint: You can use "exit(1)" to avoid the program fails when the matrix sizes are not same.)
g. Overload the operator * so that it can be used to perform matrix multiplication of two matrix with same size.
h. Following the previous question g, rewrite the operator * so that it can detect the mismatched matrix dimensions for incompatible operations. (Hint: You can use "exit(1)" to avoid the program fails.)
i. Following the previous question h, rewrite the operator * so that it can be used to perform matrix multiplication of two matrix with different size.
j. Write a function with a file output stream to output the row, column and elements of a matrix to a file.(You will generate a file named "out.txt".) Then, design a constructor including a file input stream that can be used to read the size and values of the matrix from the input stream.

The "TwoD.h" program file is provided and the required functions are in "TwoD.cpp". Please complete the operations required in each function.
Each question will have a corresponding test file, and in order to receive points for that question, your program must successfully execute and produce the correct output when tested against the provided test file. You can refer to the <Sample> below for checking your answers.

< Grading >

a. 9% (default constructor: 2%, parameterized constructor: 3%, destructor: 4%)
b. 16% (accessor – row: 2%, column: 2%, matrix: 3%, mutator – 3%, and two test case: 6% (3% for each case))
c. 6%
d. 5%
e. 5%
f. 5%
g. 5%
h. 5%
i. 7%
j. 12% (file reading and writing each account for 6%)

< Sample >

a.

```
Part a ok.
```

b.

```
1. getRow() OK
2. getCol() OK
3. setRC() OK
4. setMatrix() & getMatrix() OK
5. Index out of range
6. Index out of range
```

c.

```
matrix A:
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

matrix B(copy by A):
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

change matrix A value:
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0

matrix B (value should not be changed):
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24
```

d.

```
matrix A:
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

matrix B(B = A):
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

change matrix A value:
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0

matrix B (value should not be changed):
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24
```

e.

```
matrix A:
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

matrix B:
  1  1  1  1  1
  1  1  1  1  1
  1  1  1  1  1
  1  1  1  1  1
  1  1  1  1  1

matrix C(C=A+B):
  1  2  3  4  5
  6  7  8  9 10
 11 12 13 14 15
 16 17 18 19 20
 21 22 23 24 25
```

f.

```
matrix A:
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24

matrix B:
  1  1  1
  1  1  1
  1  1  1
Matrices not same size
 lhs matrix row, col sizes 5 5
 rhs matrix row, col sizes 3 3
```

g.

```
matrix A:
  0  1  2  3  4
  0  1  2  3  4
  0  1  2  3  4
  0  1  2  3  4
  0  1  2  3  4

matrix B:
  0  0  0  0  0
  1  1  1  1  1
  2  2  2  2  2
  3  3  3  3  3
  4  4  4  4  4

matrix C(C=A*B):
 30 30 30 30 30
 30 30 30 30 30
 30 30 30 30 30
 30 30 30 30 30
 30 30 30 30 30
```

h.

```
matrix A:
  0  0  0  0  0
  1  1  1  1  1
  2  2  2  2  2

matrix B:
  0  1  2  3
  0  1  2  3
  0  1  2  3
  0  1  2  3
Matrices not same size
 lhs matrix row, col sizes 3 5
 rhs matrix row, col sizes 4 4
```

i.

```
matrix A:
  0  0  0  0  0
  1  1  1  1  1
  2  2  2  2  2

matrix B:
  0  1  2  3
  0  1  2  3
  0  1  2  3
  0  1  2  3
  0  1  2  3

matrix C(C=A*B):
  0  0  0  0
  0  5 10 15
  0 10 20 30
```

j.

```
matrix B from in.txt:
  0  1  2  3  4
  5  6  7  8  9
 10 11 12 13 14
 15 16 17 18 19
 20 21 22 23 24
```

2. (10%)

Define a class named Payment that contains a member variable of type float that stores the amount of the payment and appropriate accessor and mutator functions. Also create a member function named paymentDetails that outputs an English sentence describing the amount of the payment.

Next, define a class named CashPayment that is **derived** from Payment. This class should redefine the paymentDetails function to indicate that the payment is in cash. Include appropriate constructor(s).

Then, define a class named CreditCardPayment that is also **derived** from Payment. This class should contain member variables for the name on the card, expiration date, and credit card number. Include appropriate constructor(s). Finally, redefine the paymentDetails function to include all credit card information in the printout.

Create a main function that creates two CashPayment and two CreditCardPayment objects with different values and calls to paymentDetails for each.

<Hint>

```
#include<string>
// Base class to store a payment amount and get a description
class Payment {
public:
        Payment();
        Payment(float amount);
        void setPayment(float amount);
        float getPayment();
        void paymentDetails();
private:
        float amount;
};
// ======================
//      main function
// ======================
   // Create several test classes and invoke the paymentDetails method
   CashPayment cash1(50.5), cash2(20.45);
   CreditCardPayment credit1(10.5, "Fred", "2023/1/5", "123456789");
   CreditCardPayment credit2(100, "Barney", "2022/11/15", "987654321");
```

< Grading >

Member functions	5%
Derive	5%

<Sample>

```
Cash 1 details:
The cash payment amount is 50.5

Cash 2 details:
The cash payment amount is 20.45

Credit 1 details:
The credit card payment amount is 10.5
The name on the card is: Fred
The expiration date is: 2023/1/5
The credit card number is: 123456789

Credit 2 details:
The credit card payment amount is 100
The name on the card is: Barney
The expiration date is: 2022/11/15
The credit card number is: 987654321
```

3. (15%)

Design a program that uses a linked list to create a stack with the following functionalities: push() and pop(). You should define a class named Stack and other member functions that need. The implementation should not use the vector.

Create a menu system that allows the user to select the option to invoke. After every option, you need to show the content of the current stack. If the user enters the option that not served, the system should allow entering again.

<Hint>

```
class Node {
public:     // Finish this section by yourself //
private:
            Node *next;
            int value;
};
class Stack {
public:    // Finish this section by yourself //
private:
            Node *headNode;
            Node *tailNode;
};
```

< Grading >

Link list	3%
Push and pop	8% (4% for each)
Display stack	4%

<Sample>

```
= Stack operations =
Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 5

Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 1
Enter a number:
5
In stack: 5

Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 1
Enter a number:
35
In stack: 5 35

Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 2
In stack: 5

Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 2
Stack is empty

Select an option to enter:
1. Push an element into stack.
2. Pop out an element from stack.
3. Exit.
Enter an option: 3
ubuntu@ubuntu-virtual-machine:~/Desk
```

<End>