Creative Software Programming Practice (week-15)

There are 19 problems. Solve the following and create directories p1, p2, ..., p19 in the week-15 directory to solve each problem. You don't have to solve all problem. Each task is a review of what you have learned before the final exam, and you ask questions you cannot solve.

1. Write a program for a sorted number array.
    A. Implement class SortedArray in the code skeleton.
    B. Take an arbitrary number of integers from the user and add them to a SortedArray instance.
    C. Process user commands as described in the Input.
    D. This program should take user input repeatedly
    E. Note that
        i. DO NOT write your own code for sorting and getting min / max. Use STL functions instead.
    F. Input:
        i. First, integer numbers to build a sorted array.
        ii. 'ascend' – Print out the numbers in ascending order.
        iii. 'descend' – Print out the numbers in descending order.
        iv. 'max' – Print out the maximum number among the numbers.
        v. 'min' – Print out the minimum number among the numbers.
        vi. 'quit' – Quit the program.
    G. Output: The result of each command.
    H. Files to submit:
        i. main.cpp – main() must be in this file.
        ii. sorted.h – Just copy the following code skeleton.
        iii. sorted.cpp – Implements SortedArray member functions.

iv. A CMakeLists.txt to generate the executable

```
$ ./sorted_array
9 3 6 2 7
ascend
2 3 6 7 9
descend
9 7 6 3 2
max
9
min
2
10 3
ascend
2 3 3 6 7 9 10
quit
$
```

Code skeleton:

```
class SortedArray
{
public:
	SortedArray();
	~SortedArray();

	void AddNumber(int num);

	 std::vector<int> GetSortedAscending();
	 std::vector<int> GetSortedDescending();
	 int GetMax();
	 int GetMin();

private:
	 std::vector<int> numbers_;
};
```

2. Write a program for an integer set.

   A. Implement class IntegerSet in the code skeleton.

   B. This program should take user input repeatedly

   C. Note that

      i. **DO NOT use std::set<int>**. Use std::vector<int> as shown in the code skeleton.

   D. Input:

      i. 'add' - Add the input number to the set and print out the set. (Do not add if the number already exists in the set.)

      ii. 'del' – Remove the input number from the set and print out the set. (Do nothing if the number is not in the set.)

iii. 'get' – Print out the element at the input position of the set. (If the input position exceeds the size of the set, print out -1.)

iv. 'quit' – Quit the program.

E. Output: The result of each command.

F. Files to submit:

i. main.cpp – main() must be in this file.

ii. intset.h – Just copy the following code skeleton.

iii. intset.cpp – Implements IntegerSet member functions.

iv. A CMakeLists.txt to generate the executable

```
$ ./integer_set
add 9
9
add 6
6 9
add 7
6 7 9
add 9
6 7 9
del 6
7 9
del 6
7 9
get 0
7
get 3
-1
quit
$
```

Code skeleton:

```
class IntegerSet {
       public:
              IntegerSet();
              ~IntegerSet();

              void AddNumber(int num);
              void DeleteNumber(int num);

              int GetItem(int pos);
              std::vector<int> GetAll();

       private:
              std::vector<int> numbers_;
}
```

3. Write a program for an answering machine

A. Implement class MessageBook in the code skeleton.

B. This program should take user input repeatedly

C. Input:

   i. 'add' [phone number] [message string] – Save [message string] for [phone number]

      1. If you save a new message to the number that already has a message, the previous message is overwritten.

      2. [message string] should be able to contain spaces. You may need to use std::getline().

   ii. 'delete' [phone number] – Delete the saved message for [phone number]

   iii. 'print' [phone number] – Print out the saved message for [phone number]. If there is no message for [phone number], just print out an empty string.

   iv. 'list' – Print out all phone numbers and its message.

   v. 'quit' – Quit the program

D. Output: The result of each command.

E. Files to submit:

   i. main.cpp – main() must be in this file.

   ii. message.h – Just copy the following code skeleton.

   iii. message.cpp – Implements MessageBook member functions.

   iv. A CMakeLists.txt to generate the executable

```
$ ./message_book
add 1112222 hello
add 2231144 nice to meet you
add 1234321 too
print 2231144
nice to meet you

list
1112222: hello
1234321: too
2231144: nice to meet you
delete 1112222
list
1234321: too
2231144: nice to meet you
quit
$
```

Code skeleton:

```
class MessageBook {
 public:
  MessageBook();
  ~MessageBook();
  void AddMessage(int number, const std::string& message);
  void DeleteMessage(int number);
  std::vector<int> GetNumbers();
  const std::string& GetMessage(int number);

 private:
  std::map<int, std::string> messages_;
};
```

4. Write a program for integer set operations.
    A. Implement functions in the code skeleton.
    B. This program should take user input repeatedly
    C. Input:
        i. { num1 num2 ... numk1 } OP { num1 num2 ... numk2 }
        ii. OP:
            1. + : Union
            2. * : Intersection
            3. - : Difference
        iii. 0 – Quit the program.
    D. Output: The resultant set of operations.
    E. Files to submit:
        i. main.cpp – main() must be in this file.
        ii. setfunc.h – Function declarations (Just copy the following code skeleton).
        iii. setfunc.cpp – Function definitions.
        iv. A CMakeLists.txt to generate the executable

```
$ ./simple_int_set
{ 1 2 3 } + { 3 4 5 }
{ 1 2 3 4 5 }
{ -1 5 3 2 } - { 1 2 3 }
{ -1 5 }
```

```
{ -1 5 3 2 } * { 1 2 3 }
{ 2 3 }
0
$
```

Code skeleton:

```
std::set<int> parseSet(const std::string& str);
void printSet(const std::set<int>&);
std::set<int> getIntersection(const std::set<int>& set0, const std::set<int>& set1);
std::set<int> getUnion(const std::set<int>& set0, const std::set<int>& set1);
std::set<int> getDifference(const std::set<int>& set0, const std::set<int>& set1);
```

5. Write a program that works as follows:
   A. Class C inherits from class B, class B inherits from class A.
   B. Each class has a public member function test().
      i. A::test() returns a string "A::test()".
      ii. B::test() returns a string "B::test()".
      iii. C::test() returns a string "C::test()".
   C. Create objects of class A, B, and C by new operator and put them into std::vector<A*> arr.
   D. Call the test() function of each element of arr to show the execution result as shown below. Each element of arr must be deallocated after use.
   E. Do not use the type casting operator throughout the code.
   F. Input: None
   G. Output: The result for calling test() functions
   H. Files to submit:
      i. A C++ source file
      ii. A CMakeLists.txt to generate the executable

```
$ ./classes
A::test()
B::test()
C::test()
$
```

6. Write a program that works as follows:

A. Class C inherits from class B, class B inherits from class A.

B. Each class has a public member function getTypeInfo().

   i. A::getTypeInfo() returns a string of "This is an instance of class A".

   ii. B::getTypeInfo() returns a string of "This is an instance of class B".

   iii. C::getTypeInfo() returns a string of "This is an instance of class C".

C. Define the following two functions in the global scope. Both functions print out strings obtained by calling getTypeInfo() of the object passed as argument.

   i. void printObjectTypeInfo1(A* object)

   ii. void printObjectTypeInfo2(A& object)

D. Create objects of class A, B, and C by new operator and put them into std::vector<A*> arr.

E. Call printObjectTypeInfo1() and printObjectTypeInfo2() by passing each element of arr as an argument. Each element of arr must be deallocated after use.

F. Do not use the type casting operator throughout the code.

G. Input: None

H. Output: The result for printObjectTypeInfo1() and printObjectTypeInfo2()

I. Files to submit:

   i. A C++ source file

   ii. A CMakeLists.txt to generate the executable

```
$ ./print_info
This is an instance of class A
This is an instance of class A
This is an instance of class B
This is an instance of class B
This is an instance of class C
This is an instance of class C
$
```

7. Write a program that works as follows:

   A. Class C inherits from class B, class B inherits from class A.

B. Add member variables.

   i. Add memberA, a private member variable of type int* to class A.

   ii. Add memberB, a private member variable of type double* to class B.

   iii. Add memberC, a private member variable of type std :: string* to class C.

C. Constructors

   i. The constructor of class A takes an [int] argument, allocates memberA with new, stores the int argument value in the allocated space, and prints the string "new memberA".

   ii. The constructor of class B takes a [double] argument, allocates memberB using new, stores the double argument value in the allocated space, and prints out the string "new memberB". And it calls the constructor of class A from the initialization list, passing an integer 1 as the argument, to initialize memberA.

   iii. The constructor of class C takes a [const std::string&] type argument, allocates memberC with new, stores the string in the allocated space, and prints the string "new memberC". And it calls the constructor of class B from the initialization list, passing a double value 1.1 as the argument, to initialize memberB.

D. Destructors

   i. The destructor of class A uses delete to free memberA and prints "delete memberA".

   ii. The destructor for class B uses delete to free memberB and prints "delete memberB".

   iii. The destructor of class C uses delete to free memberC and prints "delete memberC".

E. A, B, and C all have member functions void print().

   i. A::print() prints out the data stored in the space pointed to by memberA.

   ii. B::print() calls A::print() first, and then prints out the data stored in the space pointed to by memberB.

   iii. C::print() calls B::print() first, and then prints out the data stored in the space pointed to by memberC.

F. Take an integer, real number, and string from the user, and then create objects of class A, B, and C by new operator with user inputs and put them into std::vector<A*> arr.

G. Call the print() function of each element of arr. Each element of arr must be deallocated after use.

H. Do not use the type casting operator throughout the code.

I. Input: None

J. Output: The result for print(), and creating and destructing objects.

K. Files to submit:

   i. A C++ source file

   ii. A CMakeLists.txt to generate the executable

```
$ ./print_member
20 3.14 test
new memberA
new memberA
new memberB
new memberA
new memberB
new memberC
*memberA 20
*memberA 1
*memberB 3.14
*memberA 1
*memberB 1.1
*memberC test
delete memberA
delete memberB
delete memberA
delete memberC
delete memberB
delete memberA
$
```

8. Write a program that works as follows:

   A. Define class Shape that has a constructor taking width and height as parameters (double type).

   B. Define class Triangle and class Rectangle which inherit from class Shape. Each subclass also has a constructor taking width and height as parameters (double type).

   C. All classes have a member function, double getArea().

      i. Shape::getArea() is a pure virtual function.

   D. Take inputs from the user and create objects of proper type according to the input by new operator, and then put those objects into std::vector<Shape*> arr.

E. When the user enters 0, call the getArea() of each element of arr to print out calculated area. Each element of arr must be deallocated after use.

F. Do not use the type casting operator throughout the code.

G. This program should take user input repeatedly

H. Input:

   i. 'r' [width] [height] – Create a rectangle.

   ii. 't' [width] [height] – Create a triangle.

   iii. 0 – Print the results and quit the program.

I. Output: Areas of the shapes.

J. Files to submit:

   i. main.cpp – main() must be in this file.

   ii. shape.h – Class definitions

   iii. shape.cpp – Class member function definitions (implementations)

   iv. A CMakeLists.txt to generate the executable

```
$ ./shapes
r 10 5
t 2 4
t 10 5
r 4 3
0
50
4
25
12
$
```

9. The following program prints out 2. Modify the program using one of the C ++ type casting operators only once so that it prints out 2.5 (actual division result).

   A. Do not use C's casting operator or C++'s reinterpret_cast.

B.
```
#include <iostream>
using namespace std;
int main()
{
    int a = 10;
    int b = 4;
    cout << a / b << endl;
    return 0;
}
```

C. Input: None

D. Output: The division result.

E. Files to submit:

i. A C++ source file.

ii. A CMakeLists.txt to generate the executable

```
$ ./division
2.5
$
```

10. Write a program that works as follows using the following code:

A.
```
class B
{
public:
    virtual ~B() {}
};
class C : public B
{
public:
    void test_C() { std::cout << "C::test_C()" << std::endl; }
};
class D : public B
{
public:
    void test_D() { std::cout << "D::test_D()" << std::endl; }
};
```

B. Take arbitrary number of "B", "C", and "D" strings, and then creates one object of class B, C, or D for each string, and put those object into std::vector<B*> arr.

C. When the user enters 0, the for statement traverses each element of arr,

i. Call C::test_C() if the element is a C type object

ii. Call D::test_D() if the element is a D type object (using dynamic_cast)

iii. Do nothing for the B type object

D. Note that this problem is intended to practice how to use dynamic_cast, so remember that using dynamic_cast in this way is not desirable.

E. Each element of arr must be deallocated after use.

F. This program should take user input repeatedly

G. Input: B or C or D or 0

H. Output: The result described in the problem.

I. Files to submit:

i. A C++ source file.

ii. A CMakeLists.txt to generate the executable

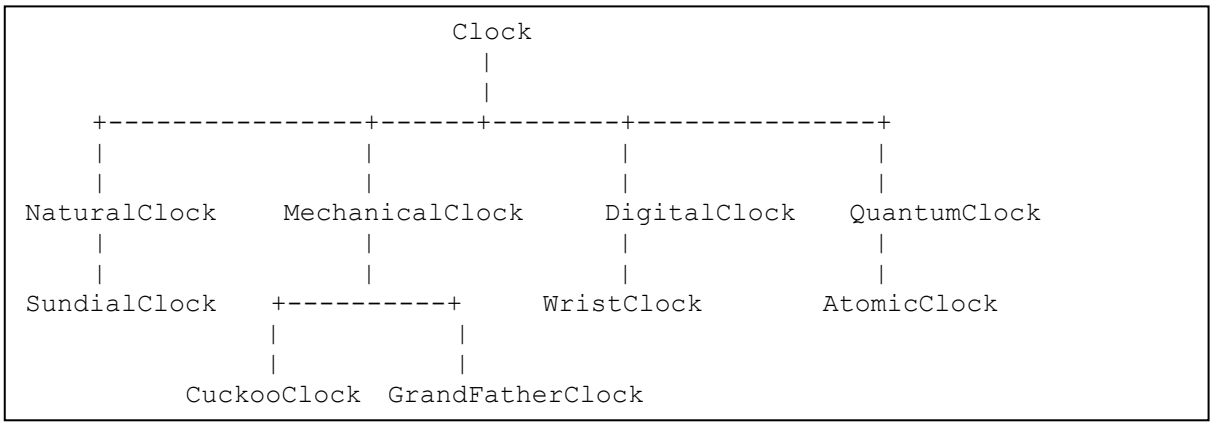
```
$ ./dynamic_cast
B
C
D
B
C
0
C::test_C()
D::test_D()
C::test_C()
$
```

11. Write a program that implements several types of clocks described below.

A. Simulate the clock's behavior during the input seconds, and print out the current reference time after the simulation and the error of each clock accumulated during the simulation.

B. The following is the clock type and class hierarchy diagram.

C.



D. The clock's behavior is implemented in clock_time.h and clock_time.cpp in the attached zip file. Use these two files without any modification to create the following clock classes.

E. Clock, NaturalClock, MechanicalClock, DigitalClock, QuantumClock are abstract base classes.

F. SundialClock, CuckooClock, GrandFatherClock, WristClock, AtomicClock are concrete derived classes.

G. Class Clock has the following protected member variables:

i.
```
ClockTime _clockTime;
double _driftPerSecond;
double _totalDrift;
```

H. The constructor of Clock must take arguments of initial hours, minutes, seconds, and errors per second as follows:

i.
```
Clock(int hour, int minute, int second, double driftPerSecond)
```

I. The constructors of NaturalClock, MechanicalClock, DigitalClock, and QuantumClock must take the same arguments as Clock's constructor, and only pass the arguments to the constructor of the parent class.

J. The constructors of SundialClock, CuckooClock, GrandFatherClock, WristClock, and AtomicClock take only three parameters: hours, minutes, and seconds, and only pass the arguments and the error per second for each clock type to the constructor of their parent class.

K. The error per second for each clock type:

SundialClock	0.0
CuckooClock	0.0
GrandFatherClock	0.000694444
WristClock	0.000347222
AtomicClock	0.000034722

L. Class Clock has three member functions:

i.
```
void reset();
void tick();
virtual void displayTime() = 0;
```

ii. reset() and tick() are not virtual functions; they exist only in the Clock class.

iii. displayTime() is a pure virtual function, and all concrete derived classes must implement this function.

iv. Clock::reset() simply calls the reset() of _clockTime, a protected member of the Clock class.

v. Clock::tick() calls the increment() of _clockTime, a protected member of the Clock class, and needs to implement some additional functionalities.

vi. displayTime() function of each concrete class basically uses the display() of _clockTime, which is a protected member of class Clock, but you need to do some additional formatting of the displayed text (see the example below).

M. The simulation proceeds in the following order.

i. Create and insert SundialClock, CuckooClock, GrandFatherClock, WristClock, and AtomicClock objects into std::vector<Clock *>. The initial time is 0 hours 0 minutes 0 seconds.

ii. First, reset all clock objects.

iii. Before starting the simulation, print out "Reported clock times after resetting:" and display the current time of each clock using displayTime().

iv. Print out "Running the clocks ...", and then call tick() for the number of seconds input by the user to run the simulation (1 tick per second).

v. After the simulation, print out "Reported clock times after running:" and display the current time of each clock using displayTime().

vi. Delete clock objects.

N. Input: An integer (representing the simulation time in seconds)

O. Output: The simulation results

P. Files to submit:

i. clock_time.h, clock_time.cpp – Do not modify them.

ii. main.cpp - main() must be in this file.

iii. clocks.h – Class definitions

iv. clocks.cpp – Class member function definitions (implementations)

v. A CMakeLists.txt to generate the executable

```
$ ./clock_time
604800
Reported clock times after resetting:
SundialClock 00:00:00, total drift: 0
CuckooClock 00:00:00, total drift: 0
GrandFatherClock 00:00:00, total drift: 0
WristClock 00:00:00, total drift: 0
AtomicClock 00:00:00, total drift: 0

Running the clocks...

Reported clock times after running:
SundialClock 24:00:00, total drift: 0
CuckooClock 24:00:00, total drift: 420
GrandFatherClock 24:00:00, total drift: 210
WristClock 24:00:00, total drift: 20.9999
AtomicClock 24:00:00, total drift: 0
$
```

12. Write a program that works as follows:

A. Implement the operators of the following MyString class.

B.

```
#ifndef __STRING_H__
#define __STRING_H__

// my_string.h - DO NOT modify this class definition
class MyString
{
public:
    // Implement operators
    MyString& operator=(const MyString& b);
    MyString operator+(const MyString& b);
    MyString operator*(const int b);
    friend    std::ostream&    operator<<(std::ostream&    out,    MyString&
my_string);
    friend    std::istream&    operator>>(std::istream&    in,    MyString&
my_string);

private:
    std::string str;
};

#endif //__STRING_H__
```

C. DO NOT modify the given my_string.h. Do not add any other member functions or member variables, do not change the member access modifiers (public, private).

D. This program should take user input repeatedly

E. Input:

i. 'new' – Create two MyString instances (named a, b) that is initialized to the following user inputs using the overloaded operator>>.

ii. [object] + [object] – Print out the concatenated string of two MyString instances [object]. [object] can be 'a' or 'b' using the overloaded operator+.

iii. [object] * [integer] – Print out the string [object] [integer] times. [object] can be 'a' or 'b' using the overloaded operator*.

iv. 'quit' – Quit the program

F. Output: The output of the operations

G. All output should be printed using the overloaded operator<<.

H. Files to submit:

i. main.cpp - main() must be in this file.

ii. my_string.h – DO NOT modify it.

iii. my_string.cpp – Class MyString's member function definitions (implementations)

iv. A CMakeLists.txt to generate the executable

```
$ ./string
new
enter a
Hanyang
enter b
University
a * 3
HanyangHanyangHanyang
a + b
HanyangUniversity
quit
$
```

13. Write a program that works the same as the prob 12 program, using the following class MyString2 instead of class MyString.

A. The goal is using a copy constructor instead of the assignment operator.

B.

```
#ifndef __STRING_H__
#define __STRING_H__

// my_string2.h - DO NOT modify this class definition
class MyString2
{
public:
    // Add constructors you need, including copy constructor

    // Incorrect implementation of assignment operator.
    // Do not use the assignment operator.
    // Do not correct this because the goal is to prevent using the
assignment operator.
    MyString2& operator=(const MyString2& b) { return *this; };

    // Just use the same implementations for these operators
    MyString2 operator+(const MyString2& b);
    MyString2 operator*(const int b);
    friend std::ostream& operator<<(std::ostream& out, MyString2&
my_string);
    friend std::istream& operator>>(std::istream& in, MyString2&
my_string);

private:
    std::string str;
};

#endif //__STRING_H__
```

C. DO NOT modify the given my_string2.h. Do not add any other member functions or member variables, do not change the member access modifiers (public, private). Do not correct the wrong implementation of the assignment operator.

D. The input, output, and example are the same as prob 1.

E. Files to submit:

   i. main.cpp - main() must be in this file.

   ii. my_string2.h – DO NOT modify it.

   iii. my_string2.cpp – Class MyString2's member function definitions (implementations)

   iv. A CMakeLists.txt to generate the executable

14. Write a program that works as follows:

A. Implement the constructor, destructor, and operators of the following MyVector class.

B.

```
#ifndef __MY_VECTOR_H__
#define __MY_VECTOR_H__

// my_vector.h - DO NOT modify this class definition
class MyVector {
public:
	// Implement constructor & destructor
	MyVector();
	MyVector(int length);
	~MyVector();

	// Implement operators
	MyVector& operator=(const MyVector& b);
	MyVector operator+(const MyVector& b);
	MyVector operator-(const MyVector& b);
	MyVector operator+(const int b);
	MyVector operator-(const int b);
	friend std::ostream& operator<< (std::ostream& out, MyVector& b);
	friend std::istream& operator>> (std::istream& in, MyVector& b);

private:
	int length;
	double *a;
};

#endif // __MY_VECTOR_H__
```

C. DO NOT modify the given my_vector.h. Do not add any other member functions or member variables, do not change the member access modifiers (public, private).

D. This program should take user input repeatedly

E. Input:

i. 'new' [length] – Create two MyVector instances (named a, b) of length [length] and fill them with the following user inputs.

ii. [object] [op] [object] – Apply [op] to two MyVector instances [object]. [op] can be '+' or '-', [object] can be 'a' or 'b'.

iii. [object] [op] [integer] – Apply [op] to [object] and [integer]. [op] can be '+' or '-', [object] can be 'a' or 'b'.

iv. 'quit' – Quit the program

F. Output: The output of the operations

G. Files to submit:

i. main.cpp - main() must be in this file.

ii. my_vector.h – DO NOT modify it.

iii. my_vector.cpp – Class MyVector's member function definitions (implementations)

iv. A CMakeLists.txt to generate the executable

```
$ ./MyVector
new 10
enter a
2 3 4 5 6 7 8 9 10 11
enter b
3 4 6 2 7 8 9 3 4 1
a + 3
5 6 7 8 9 10 11 12 13 14
a + b
5 7 10 7 13 15 17 12 14 12
quit
$
```

15. Write a program that works the same as the prob 14 program, using the following class MyVector2 instead of class MyVector.

A. The goal is using a copy constructor instead of the assignment operator.

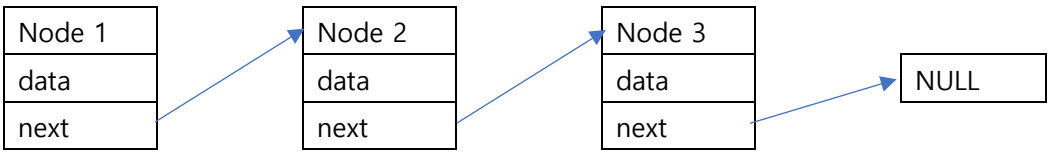
16. Write a program for templated singly linked list.

A. The list stores the following Node class template instances.

B.

```
template <class T>
class Node
{
        public:
                T data;
                Node<T>* next;
}
```

C. Diagram of singly linked list:



D. Complete the following class template List that implements singly linked list.

i. Hint: Set the next of the last node of the liked list to NULL.

E.

```
template <class T>
class List {
private:
    Node<T> *head;
public:
    List() : head(NULL) {};
    ~List();//free resources
    List(T* arr, int n_nodes);//create a list with n_nodes
    void insert_at(int idx, T& data);
    void remove_at(int idx);
    void pop_back();
    void push_back(const T& val);
    void pop_front();
    void push_front(const T& val);
    friend ostream& operator<<(ostream& out, List& rhs);//print out nodes
};
```

F. Use the following main function to test your List class template.

G.

```
int main(){
		int array[5] = {12, 7, 9, 21, 13 };
		List<int> li(array, 5);
		cout<< li << endl; //12,7,9,21,13

		li.pop_back();
		cout<< li << endl; //12,7,9,21

		li.push_back(15);
		cout<< li << endl; //12,7,9,21,15

		li.pop_front();
		cout<< li << endl; //7,9,21,15

		li.push_front(8);
		cout<< li << endl; //8,7,9,21,15

		li.insert_at(4, 19);
		cout<< li << endl; //8,7,9,21,19,15

		li.remove_at(1);
		cout<< li << endl; //8,9,21,19,15

		return 0;
	}
```

H. Input: None

I. Output: Printed results of running main() function

J. Files to submit:

i. main.cpp – Use the given main() code as is.

ii. list.h – Implementation of class template List (Write the function bodies in the header file)

iii. A CMakeLists.txt to generate the executable

```
$ ./list
12,7,9,21,13
12,7,9,21
12,7,9,21,15
7,9,21,15
8,7,9,21,15
8,7,9,21,19,15
8,9,21,19,15
$
```

K.

```
#ifndef __MY_VECTOR_H__
#define __MY_VECTOR_H__

// my_vector2.h - DO NOT modify this class definition
class MyVector2
{
public:
       // Implement constructor & destructor
       MyVector2();
       MyVector2(int length);
   MyVector2(const MyVector2& mv);
       ~MyVector2();

   // Incorrect implementation of assignment operator.
   // Do not use the assignment operator.
   //  Do  not  correct  this  because  the  goal  is  to  prevent  using  the
assignment operator.
   MyVector2& operator=(const MyVector2& b) { return *this; };

   // Just use the same implementations for these operators
       MyVector2 operator+(const MyVector2& b);
       MyVector2 operator-(const MyVector2& b);
       MyVector2 operator+(const int b);
       MyVector2 operator-(const int b);
       friend std::ostream& operator<< (std::ostream& out, MyVector2& b);
       friend std::istream& operator>> (std::istream& in, MyVector2& b);

private:
       int length;
       double *a;
};

#endif // __MY_VECTOR_H__
```

L. DO NOT modify the given my_vector2.h. Do not add any other member functions or member variables, do not change the member access modifiers (public, private). Do not correct the wrong implementation of the assignment operator.

M. The input, output, and example are the same as prob 1.

N. Files to submit:

   i. main.cpp - main() must be in this file.

   ii. my_vector2.h – DO NOT modify it.

   iii. my_vector2.cpp – Class MyVector's member function definitions (implementations)

   iv. A CMakeLists.txt to generate the executable

17. Write a program that works as follows:

A. Complete exception handling in the following code.

i. If a bad allocation occurs (when n≤0) in the constructor, throw an exception and catch it in the main() and prints "caught in the main".

ii. Objects created inside the try block calls destructors when an exception occurs, so make sure you implement the destructor properly, and check which constructors and destructors are called.

B.

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

class A
{
	public:
		A(int n)
		{
			//implement something here
			cout << "ID=" << n << ": constructed\n";

			n_ID = n;
			data = new int[n];
		}
		~A()
		{
			cout << "ID=" << n_ID << ": destroyed\n";
			//implement something here
		}
	private:
		int* data = NULL;
		int n_ID;
};

int main(){
	try{
		A a(3);
		A b(2);
		{
			A c(1);
			A d(0);
			A e(-1);
		}
	}
	//implement something here
	return 0;
}
```

C. Input: None

D. Output: Printed results as follows.

E. Files to submit:

i. A C++ source file

ii. A CMakeLists.txt to generate the executable

```
$ ./exception1
ID=3: constructed
ID=2: constructed
ID=1: constructed
ID=1: destroyed
ID=2: destroyed
ID=3: destroyed
caught in the main
$
```

18. Write a program that works as follows:

A. If you throw an object 'a' as shown in the following code, 'a' is copied and used internally because the destructor of 'a' is called at the end of the try scope (ie, "Objects Thrown as Exceptions Are Always Copied ").

B. When you run the following program, modify the program so that it prints out as shown in the example output.

C. Hint: What do you need for an object to be copied properly?

19. Write a program that works as follows:

A. Complete exception handling in the following program to print out results as shown in the example below.

B. Use "chain handler" (multiple catch blocks)

C. Use the printing code (cout) only in the catch blocks.

D.

```
#include <iostream>

using namespace std;

class A
{
};
class B : public A
{
};
class C : public B
{
};

int main(){
	int n;
	cout << "input num(0~2):";
	cin >> n;

	try{
		if(n == 0)
			throw A();
		else if(n == 1)
			throw B();
		else if (n ==2)
			throw C();
		else
			throw string("invalid input");
	}
	//implement here

	return 0;
}
```

E. Input: 0, 1, or 2

F. Output: Printed results as follows.

G. Files to submit:

i. A C++ source file

ii. A CMakeLists.txt to generate the executable

```
$ ./exception3
input num(0~2):0
throw A has been called
$ ./exception3
input num(0~2):1
throw B has been called
$ ./exception3
input num(0~2):2
throw C has been called
$ ./exception3
```

```
input num(0~2):3
invalid input
$
```