CS 202 - Computer Science II

Finals Sample

Release date: Wednesday, 12/6/2017

Test Objectives: A comprehensive evaluation of the course material. Everywhere, the use of pointers, bracket-notation, and the built-in C-string library <string.h> functions are allowed.

Program 1 (60 pts):
You are given the working definitions and implementations of two Classes (description-only shown here):

```
class Cover{
  public:
    //Cover() default constructor, sets m_hard to false
    //Cover(bool hard) parametrized constructor, sets m_hard to hard
 //operator<< overload, outputs m_hard to the calling object(cout,etc…)
 //operator>> overload, assigns m_hard a 0/1 value from calling object input(cin,etc…)
    //GetValue() method, returns m_hard by value and does not modify calling object
  private:
    bool m_hard;
};

class Client{
  public:
    //Client() default constructor, leaves m_name uninitialized
    //Client(const char* name) parametrized constructor, copies c-string name to m_name
    //Client(const Client& other) copy constructor, deep-copies data
    //~Client() destructor, deallocates memory as necessary
    //operator= overload, deep-copies data and returns reference to calling object
//operator<< overload, outputs m_name to the calling object(cout,etc…)
//operator>> overload, grabs and copies to m_name input from calling object(cin,etc…)
    //GetName() method, returns m_name by address and does not modify calling object
  private:
    char* m_name;
};
```

(Part1 Grading Scheme: **20pts, 10pts Class Declaration – 10pts Class Implementation**)
(Hint: A smaller subset of such specifications will be required for the actual Finals)
For the first part of this program you are required to implement a Book class, which will has to adhere by these specifications:

Each Book should have the members (all private):
- **m_title**: a <u>C-string</u> (pointer-not array) representing the book title.
- **m_cover**: a Cover class <u>object</u> which represents the book cover.
- **m_client**: a Client class <u>const pointer</u>, the person (renter) who currently has the book in their possession. If the book is not rented out to someone this pointer should be NULL. (Hint: Remember const pointer means the m_client variable can be assigned to pointed to different objects, but it cannot be used to modify the object it points to).
- **m_serial**: a <u>constant</u> size_t number, representing a unique identifier for each Book object.

and the Book class will also have a (private as well):
- **count**: a <u>static</u> size_t, keeping track of the last (greatest) unique Book id (as generated via a constructor).

and for the Book ADT you are required to implement:

- a **Default Constructor**.
- a **Parametrized Constructor**, with a parameters list that allows initialization of all members. The order should be: (m_name, m_cover, m_client, m_serial) and there should be <u>default parameters</u> in the list passed for m_cover, m_client, and m_serial.
- a **Copy Constructor**. It should <u>deep-copy</u> the Book Object's members, <u>except</u> m_client (the new Book will have no renter). Also, a <u>new unique m_serial</u> should be assigned.
- an **Assignment operator=** overload. It should <u>deep-copy</u> the Book Object's members that it can, and <u>shallow-copy</u> m_client (the new Book will have the exact same renter). <u>Returns a reference</u> to the calling object.
- **Get/Set** methods for **m_cover** and **m_client**. Get methods should not allow modification of the calling object.
- **Get** methods for **m_title** and **m_serial**, which should not allow modification of the calling object.
- a **Serialize** method. It should output the Book's: m_title, m_cover, m_serial, and m_client (if a renter exists).
- an **Insertion operator <<** overload. It should use the Serialize method of the passed Book object to output its data to the std::ostream object it takes as its first parameter.

```cpp
#include <iostream>
#include <string.h>
using namespace std;

//fill with return values, parameters, specifiers
class Book {
  public:
  //default ctor
  Book(                                              )         ;

  //parametrized ctor
  Book(


                                                     )         ;

  //copy ctor
  Book(                                    )         ;

  //dtor
  ~Book(                                   )         ;

  //assignment operator
                operator=(                  )         ;

  //get set methods
                GetCover(                   )         ;

                 SetCover(                  )         ;

                GetClient(                  )         ;

                SetClient(                  )         ;

  //serialize method

                Serialize(                  )         ;

  //insertion operator overload
                operator<<(

                                            )         ;

  //data members
  private:




};
```

```cpp
//default ctor
Book::Book(                                                )


{


}

//parametrized ctor
Book::Book(                                                )


{


}


//copy ctor
Book::Book(                                                )


{


}


//dtor
Book::~Book(                                                  )


{


}


//assignment operator
                Book::operator=(                                          )
{


}
```

```cpp
//get set methods
                Book::GetCover(                                    )
{


}
                Book::SetCover(                                    )
{


}
                Book::GetClient(                                   )
{


}

                Book::SetClient(                                   )
{


}

//serialize method
                Book::Serialize(                                   )
{


}


//insertion operator overload
                operator<<(

                                                                   )
{


}
```

(Part2 Grading Sheme: **20pts, 10pts Class Declaration – 10pts Class Implementation**)
(**Hint: A subset of such specifications will be required for the actual Finals**)
For the second part of this program, you are required to implement a ChildrenBook class, which has to adhere by these specifications:

Each ChildrenBook should be a Derived class inheriting from the Book Base class.
  ➢ Rewrite only the members section of the Book base class declaration, so that ChildrenBook has <u>Inheritance</u>-level access to: m_title, m_cover, m_serial.
Each ChildrenBook object will have its own private member:
  • **m_graphic**: a bool representing whether it's a graphic or literature novel.
and for the ChildrenBook derived ADT you are required to implement:
  • a **Default Constructor**.
  • a **Parametrized Constructor**, with a parameters list that allows initialization of all members of the base class, as well as a bool graphic parameter that is used to initialize m_graphic. The order should be: (m_name, m_graphic, m_cover, m_client, m_serial) and there should be <u>default parameters</u> in the list passed for m_cover, m_client, and m_serial.
  • a **Copy Constructor**. It should perform the exact same operations as the Book base class one. Additionally, m_graphic should be copied to the new ChildrenBook.
  • an **Assignment operator=** overload. It should perform the exact same operations as the Book base class one. Additionally, m_graphic should assigned to the calling ChildrenBook.
  • **Get/Set** methods for **m_graphic**. The Get method should not allow modification of the calling object.
  • a **Serialize** method. It should output the ChildrenBook's: m_title, m_graphic, m_cover, m_serial, and m_client (if a renter exists). Notice that it also outputs m_graphic which is the Derived class extra member.
  • an **Insertion operator <<** overload. It should use the Serialize method of the passed ChildrenBook object to output its data to the std::ostream object it takes as its first parameter.

```cpp
#include <iostream>
#include <string.h>
using namespace std;

//fill derived class with return values, parameters, specifiers
class ChildrenBook                                          {
  public:
  //default ctor
  ChildrenBook(                                      )          ;

  //parametrized ctor
  ChildrenBook(                                      )          ;

  //copy ctor
  ChildrenBook(                                      )          ;

  //dtor
  ~ChildrenBook(                                     )          ;

  //assignment operator
                  operator=(                         )          ;

  //get set methods
                  GetGraphic(                        )          ;

                  SetGraphic(                        )          ;

  //serialize method
                  Serialize(                         )          ;

  //insertion operator overload
                  operator<<(

                                                     )          ;


  //data members
  private:




};
```

```cpp
//default ctor
ChildrenBook:: ChildrenBook(                                    )



{



}

//parametrized ctor
ChildrenBook:: ChildrenBook(                                    )



{



}


//copy ctor
ChildrenBook:: ChildrenBook(                                      )



{



}


//dtor
ChildrenBook::~ChildrenBook(                                      )



{



}


//assignment operator
            ChildrenBook::operator=(                             )
{



}
```

```cpp
//get set methods
            ChildrenBook::GetGraphic(                    )
{


}

            ChildrenBook::SetGraphic(                       )
{


}


//serialize method
            ChildrenBook::Serialize(                     )
{


}


//insertion operator overload
                operator<<(

                                                    )
{


}
```

(Part3 Grading Sheme: **20pts, 5pts-per-Question**)
For the second part of this program, you are required to consider the following main():

```cpp
001  #include <iostream>
002  #include <string.h>
003
004  #include "Cover.h"
005  #include "Client.h"
006  #include "Book.h"
007  #include "ChildrenBook.h"
008
009  using namespace std;
010
011  int main()
012  {
013      Client jDoe("John Doe");
014      Book myBook("LOTR ROTC", Cover(true), &jDoe, 999);
015
016      Client jDoeJr("John Doe Jr");
017      ChildrenBook myChildBook("LOTR comic", true, Cover(false), &jDoeJr);
018
019      Book* book_Pt;
020
021      book_Pt = &myBook;
022      cout<<*book_Pt<<endl;
023
024      book_Pt = &myChildBook;
025      cout<<*book_Pt<<endl;
026
027      return 0;
028  }
```

➢ What is the output of lines 022 and 025 (briefly mention for each whether they output just the Base or the Derived class information too)?
➢ Explain what the problem is, when trying to access the Derived class information through a Base class pointer.
➢ How would you propose to fix the aforementioned problem (you are only allowed to modify class declarations and/or implementations, not add any more functions)? Briefly mention the modifications you would make.
➢ What is now the output of highlighted lines 022 and 025 (briefly explain why)?

Program 2 (15 pts):
You are given a class Declaration for a Matrix class, which relies on dynamic memory for storage.

```
001  class DynamicMatrix {
002   public:
003     // 1) instatiates a [0]x[0] NULL matrix
004     DynamicMatrix();
005     // 2) instatiates a [rows]x[cols] matrix with all elements set to
006  [value]
007     DynamicMatrix(int rows,int cols, int value=0);
008     // 3) instantiates via matrix copy
009     DynamicMatrix(const DynamicMatrix& otherDynamicMatrix);
010     // 4) destroys matrix and deallocates dynamic memory
011     ~DynamicMatrix();
012
013     // 5) assignment operator
014     DynamicMatrix& operator=(const DynamicMatrix& other);
015
016     // …
017
018   private:
019     int m_rows;
020     int m_cols;
021     int** m_matrix;
022  };
```

(Hint: Question variants on this can include any Method)
Give the implementation of its Copy Constructor (#3).

Program 3 (15pts, 2.5 pts Node Declaration – 2.5 pts Node Implementation – 5pts Queue Declaration – 5pts Queue Implementation):
You are given a class Declaration for a (Node) Queue class that works with int objects. You have to provide:

➢ a) The templated version of class Node (declaration and implementation). Hint: Also write any possible necessary forward declarations.
➢ b) The templated version of class Queue (declaration, including any necessary forward declarations).
➢ c) The templated (non-templated will receive partial credit) implementation of the Queue methods: push(), pop(), size(), clear(), and the overloaded insertion operator<<.
   (Hint: Question variants on this can include any Method).

```cpp
001  ////////////////////////////////NODE//////////////////////////////
002  class Node{
003  public:
004    Node() : m_next(NULL){ }
005    Node(const int& data, Node* next = NULL) : m_data(data),
006  m_next(next){ }
007    const int& getData() const{ return m_data; }
008    int& getData(){ return m_data; }
009  friend class Queue;  //declaration of friend class
010  private:
011    Node* m_next;
012    int m_data;
013  };
014
015  ////////////////////////////////QUEUE/////////////////////////////
016  class Queue{
017  public:
018    Queue();
019    ~Queue();
020    Queue& operator=(const Queue& other);
021    const int& front() const;
022    int& front();
023    const int& back() const;
024    int& back();
025    void push(const int& value);
026    void pop();
027    size_t size() const;
028    void clear();
029  friend std::ostream& operator<<(std::ostream& os, const Queue& queue);
030  private:
031    Node *m_front;
032    Node *m_back;
033  };
```

Question 1 (5 pts): (Subject: Class Relationships and Operations, you have to follow the program flow by starting at the main function entry point and tracing all the calls (constructors, operators, etc) made subsequently)

What is the expected output of this C++ program?

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Base{
  public:
    Base(){ cout << "B" << ++count << endl; }
      ~Base(){ cout << "~B" << --count << endl; }
    protected:
      static size_t count;
};
size_t Base::count = 0;

class Derived : public Base{
  public:
    Derived(){ cout << "D" << ++d_count << "," << count << endl; }
      ~Derived(){ cout << "~D" << --d_count << "," << count << endl; }
    private:
      static size_t d_count;
};
size_t Derived::d_count = 0;

void fB(){
        Base b;
}
void fD(){
        Derived d;
}

int main()
{
  fB();

  fD();

  return 0;
}
```

Question 2 (5 pts): (Subject: Recursive Functions, you have to learn the examples of recursion given in Lecture 25, and/or potentially identify the case of an Infinite Recursion)

Will these programs work? If yes, what is the expected output?

Left program:

```
001  #include <iostream>
002  #include <string.h>
003
004  using namespace std;
005
006  void rec (int n){
007    if (n < 0){
008      cout << n << endl;
009    }
010    else {
011        rec( n / 10 );
012        cout << ( n % 10 ) << endl;
013    }
014  }
015
016
017  int main()
018  {
019    rec(123);
020
021    return 0;
022  }
023
024
025
026
027
028
029
030
031
032
033
```

Right program:

```
#include <iostream>
#include <string.h>

using namespace std;

int rec (int n) {
  cout << n << " ";
  if (n > 1)
    return rec (n-1) + rec (n-2);
  else
    return n;
}

int main()
{
  int r = rec(4);

  cout << endl << r;

  return 0;
}
```

Question 3 (5 pts): (Subject: Working/Compiling with Templates, you have to apply the rules you have learnt on how to compile templated classes/functions and identify possible compilation errors (e.g. (i) missing forward declaration of Templated class before the forward declaration of a Templated operator overload for this class' objects, (ii) missing empty angled brackets <> in friend templated method declaration unless the method is inlined, etc.) or if compilation goes through, trace the program output from the resulting Templated implementation).

Will this program compile? Briefly mention what it will do if yes.

```
001  #include <iostream>
002  #include <string.h>
003
004  using namespace std;
005
006  template<class T, size_t NROWS, size_t NCOLS> std::ostream&
007      operator<<(std::ostream& os, const Matrix<T,NROWS,NCOLS>& matrix);
008
009
010  template<class T, size_t NROWS=1, size_t NCOLS=1>
011  class Matrix{
012   public:
013     Matrix(){}
014
015    friend std::ostream&
016    operator<< <> (std::ostream& os, const Matrix<T,NROWS,NCOLS>& matrix);
017
018    private:
019     T container[NROWS][NCOLS];
020  };
021
022
023  template<class T, size_t NROWS, size_t NCOLS> std::ostream&
024      operator<<(std::ostream& os, const Matrix<T,NROWS,NCOLS>& matrix){
025
026    for (size_t i=0; i<NROWS; ++i){
027      for (size_t j=0; j<NCOLS; ++j){
028        os  <<  matrix.container[i][j] << " ";
029        }
030        os << std::endl;
031    }
032    os << std::endl;
033
034  }
035
036  int main()
037  {
038    Matrix<float, 10, 5> mat;
039    cout << mat;
040
041    return 0;
042  }
```

Question 4 (5 pts): (Subject: Stack Unwinding / Exception Classes / throw-catch, you have to trace the process of Stack Unwinding)

What is the expected output here?

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class MyException{
 public:
    // instantiates and initializes info string
    MyException(const char* s) : m_info(s){ }
    // sets info string to desired value
    void SetInfo(const char* s){ m_info = s; }
  // handles output of exception object data (info string)
  friend std::ostream& operator<<(std::ostream& os,
                                  const MyException& e){
        os << e.m_info;
        return os;
  }
 private:
    std::string m_info;
};

class A{
  public:
    A(){ cout << "A" << endl; }
    ~A(){ cout << "~A" << endl; }
};

int main(){

  try{
    A anA;

    try{
      A anotherA;

        //error detected
        throw MyException("Something awful happened here...");
    }
    catch(MyException& e){
        cerr << e << endl;
        e.SetInfo( "It's been taken care of!" );
        throw;
    }
  }
  catch(const MyException& e){
    cerr << e << endl;
  }

  return 0;
}
```