CS 202 - Computer Science II

Midterm Sample

Release date: Sunday, 10/15/2017, 6:00 am

Test Objectives: The main test objectives of this Midterm Sample will comprise of: iostream, file I/O, C-style strings, arrays, pointers, ADTs (class/struct), encapsulation, inheritance, polymorphism.

Program 1 (65 pts):
In this program you are required to implement the item database management structure and operations for a library. The library carries up to **LIBRARY_N_BOOKS=1000** (can be declared as global const int) books in its shelves.

[25pts] Each **Book** should be a class containing the members (all private):

- **m_id**: a `const int`, a unique identifier for the Book object (no two objects can ever have the same m_id at any time).
- **m_title**: a C-style string (**255** characters max), the book title. A default title **`"norenter"`** corresponds to an *invalid* Book object.
- **m_isbn**: an `int` array (**13** integers long), a special identifier for a book issue (same-title books can have different m_isbn depending on whether they are paperback, hardcover, etc.). A default isbn **[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]** corresponds to an *invalid* Book object.
- **m_available**: a `bool`, whether the book is available for renting out or it is already in someone's possession.
- **m_renter**: a C-style string (**255** characters max), the person who currently has the book in their possession. If the book is available for rent this should default to **`"norenter"`**.
- **s_idgen**: a `static int` class member to serve as an auxiliary variable in order to generate unique object identifiers for m_id.

and the Book ADT should implement:

- **Get/Set** methods where possible for **m_id**, **m_title**, **m_isbn**, **m_available**, **m_renter.**
- A **Get** method for the static member **s_idgen**.
- A **Valid** method, returning a `bool` when the calling Book object is valid (it checks for the two conditions of m_title and m_isbn that indicate an *invalid* Book object).
- An **operator+** overload, taking in a C-style string of the name of the person to rent the book, and returning a `bool` (true on success), and. It is used to mark a book as reserved/rented (the book should be available beforehand for the function to succeed).
- a **Free** method, returning nothing, and taking in no arguments. It is used to mark the book as available.

and you should additionally define:

- an insertion **operator <<** overload, for Book objects, which should output the Book's: **m_title**, **m_id**, **m_isbn**, **m_renter** (**`"norenter"`** if it is available).

You are also required to implement:

- a **custom Default Constructor**, it should properly handle m_id and should initialize all other members such that they correspond to an *invalid* and available Book object.
- a **Parametrized Constructor**, with a default parameters list containing initializing arguments

for the title, isbn, and renter (there should also be default parameters specified in the list passed for m_isbn and m_renter to initialize the Book object as *invalid* and available).

- a **custom Copy Constructor**. It should copy over the other Book object's data but mark the newly instantiated Book object as available.
- A **custom Destructor**. It should not perform any special functionality.
- a **custom Assignment Operator**. It should assign to the calling object all the values of the object on the right-hand-side (all the values it is permitted to perform assignment on).

[25pts] The Library should be a class containing the members (all private):

- **m_name**: a C-string (255 characters max), the name of the Library.
- **m_inventory**: a Book object array (LIBRARY_N_BOOKS long), the Library's book inventory. The array will be fixed-size, so if one or more positions (m_inventory array indexes) do not contain an actual book entry, the respective Book object should be *invalid*.

and the Library ADT should implement:

- a **Get/Set** method for its m_name member variable.
- a **FindOpenSpot** method, which should return a Pointer to a Book object of m_inventory which is empty (contains an *invalid* Book object). It should return `NULL` if all the array positions are taken up already.
- an **operator[]** overload, takes in a C-string (the title of a Book to look-up) and returns a Pointer to the first Book object of m_inventory which matches that title. It should return `NULL` if no Book with a matching title is found.
- an **operator[]** overload, takes in an `int`, the index of a Book in m_inventory and returns that Book object by-Reference.
- a **RentBook** method method, takes in an int (the index of the Book to rent in m_inventory) and a char* (the name of the person renting). The method returns a bool (true on success). If there is no valid Book in that index or the Book is already rented out, it should just return false.
- an **operator+** overload to add a Book to its m_inventory. It should return bool (true on success), i.e. if there is no empty position in m_inventory to place the book, it shouldn't add it and should return false.

and you should additionally define:

- an insertion **operator <<** overload, for Library objects, which should output all the Librarys's *valid* Book objects and their corresponding index in the m_inventory array.

You are also required to implement:

- a **Parametrized Constructor**, allowing to set the m_name of the Library.

[10pts] The program will sequentially perform these actions:

- **Create**: A Library named "DeLaMare Science and Engineering Library".
- **Import**: Read-in from a file named LibraryIndex.txt a list of books to populate the Library's inventory. The list will have the format (can also have duplicate books):

[Book name] isbn_number (rented.to) eg:
<LibraryIndex.txt>
[Animal Farm] 9789380070520 (J.Doe)
[War and Peace] 9788420649313 (none)
[Beyond Good and Evil] 9780521770781 (none)
[The Wealth of Nations] 9780786514854 (A.Smith)
…

- **Ask** the user for the index of the Book they want to rent and their own name. Use RentBook to **reserve** the Book (can succeed or fail, depending on user input).
- **Export**: Write-out to the same file named LibraryIndex.txt the list of books as it stands now (after the interaction with the user).

Pointers must be used for any array manipulation actions, pointers and references must be used in function prototypes and parameter lists, not square brackets. Pointers can only be moved by incrementing or decrementing (i.e. ++ or - -), or by setting the pointer back to the base address using the array name. Follow encapsulation rules and implement data protection mechanisms. You don't have to write code split up into header/source files.

[5pts] You should write and use your own functions for:

- C-style string copying
- C-style string comparison
- conversion of C-style numeric string to int-array (beware of array size incompatibility). You may use atoi in your implementation if you want.
- int-array copying (beware, int arrays have no null-character termination).
- int-array terminal output (again, int-arrays are not null-terminated).
- int-array file output (again, int-arrays are not null-terminated).

Question 1 (7.5 pts):
What will happen in this C++ program?

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

void printArray(int arr[], int size){
  for (int i=0; i<size; ++i){
    cout << arr[i] << " ";
  }
  cout << endl;
}

void fillArrayAscending(int arr[], int size){
  for (int i=0;i<size;++i){
    arr[i] = i;
  }
}


const int ARRAYSIZE = 10;

struct MyStruct{
  int intArray[ARRAYSIZE];
};

void fillStructArrayAscending(MyStruct st_in){
  fillArrayAscending(st_in.intArray, ARRAYSIZE);
}

void printStructArray(MyStruct st_in){
  printArray(st_in.intArray, ARRAYSIZE);
}


int main(){

  MyStruct my_struct;

  printStructArray(my_struct);

  fillStructArrayAscending(my_struct);

  printStructArray(my_struct);

  return 0;
}
```

Question 2 (7.5 pts):
What will happen the A), B) C++ programs?

A)

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

struct MyStruct{

  void PrintIntVar(){
    cout << intVar;
  }

  int intVar;
};




int main(){

  MyStruct ms;
  ms.intVar = 1;
  ms.PrintIntVar();

  return 0;
}
```

B)

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

class MyClass{

  public:
    void SetIntVar(int v){
      m_intVar = v;
    }

    void PrintIntVar(){
      cout << m_intVar;
    }

  private:
    int m_intVar;
};


int main(){

  MyClass mc;
  mc.SetIntVar(1);
  mc.PrintIntVar();

  return 0;
}
```

Question 3 (7.5 pts):
What will happen in this C++ program?

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

class TestClass{

    TestClass(){
      cout << m_intTest;
    }

    TestClass(int intTest){
      m_intTest = intTest;
      cout << m_intTest;
    }

  private:
    int m_intTest;

};


int main(){

  TestClass tc(1000);

  return 0;
}
```

Question 4 (7.5 pts):
What will happen in this C++ program?

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

class StaticClass{

  public:
    static int count;

    StaticClass(){
       m_count = 0;
       count++;
    }

    StaticClass(int count_in){
      m_count = count_in;
      count++;
    }

    void CountUp(){
      m_count++;
    }

    int GetCount(){
      return m_count;
    }

  private:
    int m_count;
};

int StaticClass::count = 0;

int main(){

  StaticClass sc_a;

  sc_a.CountUp();

  StaticClass sc_b(sc_a.count);

  sc_b.CountUp();

  StaticClass sc_c(sc_b);

  sc_c.CountUp();

  cout << sc_a.GetCount() <<" "<<
          sc_b.GetCount() <<" " <<
          sc_c.GetCount() <<" " <<
          StaticClass::count << endl;

  return 0;
}
```

Question 5 (7.5 pts):
What will happen in this C++ program?

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

class BaseClass{

  public:
    void SetIntVar(int i){
      m_intVar = i;
    }

    int GetIntVar(){
      return m_intVar;
    }

  private:
    int m_intVar;
};


class DerivedClass : public BaseClass{

  public:
    void SetDoubleVar(double d){
      m_doubleVar = d * m_intVar;
    }

    double GetDoubleVar(){
      return m_doubleVar;
    }

  private:
    double m_doubleVar;
};

int main(){

  BaseClass b_result;

  BaseClass b1;
  b1.SetIntVar(10);

  DerivedClass d2;
  d2.SetDoubleVar(2.5);

  b_result.SetDoubleVar((double)b1.GetIntVar() + d2.GetDoubleVar());
  cout << b_result.GetDoubleVar();

  return 0;
}
```

Question 6 (7.5 pts):
What will happen in this C++ program?

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

class Parent{

  public:
    virtual void SetValue(int value){
      m_value = value;
    }

    virtual int GetValue(){
      return m_value;
    }

  protected:
    int m_value;
};

class Child : public Parent{

  public:
    virtual void SetValue(int value){
      m_precisionValue = value;
    }

    virtual double GetValue(){
      return m_precisionValue;
    }

  private:
    double m_precisionValue;
};

int main(){

  Child c;

  c.SetValue(1);

  cout << c.GetValue()/2 << endl;

  return 0;
}
```