

**COS1512
RCO1512**

May/June 2013

INTRODUCTION TO PROGRAMMING II

Duration 2 Hours

75 Marks

EXAMINERSFIRST
SECONDMISS CNA NOMBEWU
PROF ID SANDERS

MRS MA SCHOEMAN

Closed book examination

This examination question paper remains the property of the University of South Africa and may not be removed from the examination venue

This paper consists of 7 pages.

**This examination paper remains the property of the University of South Africa
and may not be removed from the examination room**

Instructions:

- 1 Answer all questions
- 2 All rough work must be done in your answer book
- 3 The mark for each question is given in brackets next to the question
- 4 Please answer the questions in order. If you want to do a question later, leave enough space

GOOD LUCK!

QUESTION 1**5 MARKS**

1 1 Assume the following declarations

```
char name[21];
char yourName[21];
char studentName[21];
```

For each of the following two statements, explain why the statement is invalid. Also provide the correct statements to achieve the desired effect.

```
if (yourName == name)
    studentName = name;
```

(3)

1 2 Explain the difference between an array and a vector (2)

QUESTION 2**5 MARKS**

2 1 Consider the following code fragment which is assumed to be embedded in a complete and correct C++ program

```
1    int x;
2    int y;
3    int *r = &x,
4    int *s = &y;
5    *r = 35;
6    *s = 98,
7    *r = *s,
8    cout<<x <<"    "<<y<<endl;
9    cout<<*r <<"    "<<*s<<endl;
```

2 1 1 Give the output of the above code fragment. (1)

2 1 2 Explain the function of operator & in line 3. (1)

2.1 3 Explain the function of operator * in line 5 (1)

2 2 Consider the following code which is assumed to be embedded in a complete and correct C++ program

```
int array_size = 10,
int * a,
int i;
a = new int [array_size];
int * p = a;
```

2 2 2 Describe the action of the **new** operator. What does the **new** operator return? (1)

2 2.3 Write code to return the memory allocated to a, back to freestore. (1)

QUESTION 3**35 MARKS**

A bookstore has two types of customers: those who are members of the bookstore, and those who buy books from the bookstore occasionally. For each member of the bookstore, the bookstore keeps track of the number of books bought and the total amount spent. For every eleventh book that a member buys, the bookstore takes the average of the total amount of the last ten books bought, applies this amount as a discount, and then resets the total amount spent to 0

Define a class `BookstoreMember` that represents a record for a member of the bookstore. This class has three member variables.

- **name**, a string that holds the name of the member
- **books_bought**, an `int` representing the number of books the member has bought
- **amount**, a `double` representing the total amount the member has spent

In addition, the class should have the following member functions:

- A **default constructor** that initializes `name` to an empty string, `books_bought` to 0 and `amount` to 0
- An **overloaded constructor** that accepts a new member's name only and initializes both `books_bought` and `amount` to 0
- A **destructor** which does not perform any action.
- **Accessor functions** `get_name()`, `get_books_bought()` and `get_amount()` to return the values stored in each of the object's member variables
- A member function `update_totals()` with one parameter of type `double`, `update_amount`, representing the cost of the book bought. `update_totals()` adds the cost of the book to the total amount the member has spent, and increases the number of books bought by one
- A member function `discount()` to determine whether a member should receive a discount or not. Use the following prototype

```
bool BookstoreMember::discount() const;
```
- A bookstore member receives discount on every eleventh book.
- A member function `reset_counts()` to reset both the member variables `books_bought` and `amount` to 0.
- An **overloaded stream insertion operator** `<<` (implemented as a **friend** function) to output the name of a member, the number of books bought and the total amount the member has spent. Use the following prototype

```
ostream& operator << (ostream& outs, const BookstoreMember& the_member)
```
- An **overloaded stream extraction operator** `>>` (implemented as a **friend** function) that inputs all the member variables of a `BookstoreMember` object.

3.1 Create the header file `BookstoreMember.h` that contains the `BookstoreMember` class

- specification. (9)
- 3 2 Create the implementation of the class `BookstoreMember` including the friend functions (15)
- 3 3 Complete the application program (`main()`) on the next page by citing the number and writing down the missing statement. This program obtains the details for a new purchase by a bookstore member from the user. It then extracts all existing bookstore members from a file `Members.dat` to find the member's record. The member's information is then updated with `update_totals()` and the overloaded operator `<<` used to display updated data for the member (the name of the member, the number of books bought and the total amount the member has spent). The member function `discount()` is used to determine whether a member should receive a discount, and if so, the discount amount is printed and the number of books bought and the amount reset. Be sure to use appropriate member functions in the required statements. (11)

For example, if the input file has the following contents

Input file (Members.dat)

```
Peter 5 2508.50
Sally 9 3500.75
Hester 7 9530.48
Joel 3 1900.35
```

The program should produce the following output

Console output:

```
Enter name of the bookstore member Sally
Enter value of book bought: 350.00
Sally have bought 10 book(s) and spent R 3850.75
Press any key to continue . . .
```

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include "BookstoreMember.h"
using namespace std;

int main()
{
    _____ 1 _____ // 1.Declare input file

    _____ 2 _____ // 2 Open the file Members.dat
    _____ 2 _____ // and check that the file exists

    // Add a new purchase of books
    string memberName;
    int nrBooksBought;
    double amountBought;
```

```

cout << "Enter name of the bookstore member: ";
cin >> memberName;
cout << "Enter value of the book bought: R";
cin >> amountBought,

_____ 3 _____ // 3. Instantiate object aMember to extract a
                      // member's record from file Members.dat

while ( _____ 4 _____ ) // 4. Extract a bookstore member from file
{
    _____ // Members dat
    if ( _____ 5 _____ ) // 5. Compare memberName with aMember to
    // check whether this is the member's record
    {
        _____ 6 _____ // 6. Update the totals for the member
        _____ 7 _____ // 7 Display member data
        if ( _____ 8 _____ ) // 8. Determine whether discount is due
        {
            double discount_amount = _____ 9 _____
            // 9. Calculate discount as the average of the total spent
            // amount on the previous ten books bought
            outs.setf(10s::showpoint);
            outs.setf(10s::fixed),
            outs << "(S)he is entitled to R" << setprecision (2)
                << discount_amount << " discount" << endl;

            _____ 10 _____ // 10. Reset the counts for the
            // member's record
        }
    }
}

_____ 11 _____ // 11 Close file

return 0;
}

```

QUESTION 4**15 MARKS**

Consider the class specification (interface) for the class Student below:

```

class Student
{
    public
        Student (),

```

```

        Student (string stdName, string stdNr);
        string getName() const;
        string getStdNumber() const;
        void setName(string stdName);
        void setStdNumber(string stdNr);
        void displayStdInfo( ) const; //display data members for class

private:
        string name;
        string stdNumber;
}

```

- 4.1 Derive a class `PostGradStd` from class `Student`. This class has additional member variables `promoter` and `thesis` for the promoter and thesis. The class also has additional member functions `getPromoter()`, `getThesis()`, `setPromoter()`, and `setThesis()`. The class should override member function `displayStdInfo()` to display the student's name, student number, promoter and thesis. Provide only the interface of class `PostGradStd` in terms of a header file. The header file should contain compiler directives to prevent multiple definitions. Assume that the interface of class `Student` is contained in an interface file called `Student.h`. (7)
- 4.2 Implement the overloaded constructor for the class `PostGradStd` by invoking the base class constructor. (3)
- 4.3 Indicate whether the following statements are valid or invalid. Give reasons for your answers. (4)
- 4.3.1 The `displayStdInfo` member function is overloaded when `PostGradStd` is derived from class `Student`.
- 4.3.2 If `name` was a protected data member of `Student`, then the following statement embedded within member function `displayStdInfo` in class `PostGradStd` would be legal
- ```
cout << "Student name:" << name;
```

**QUESTION 5****10 MARKS**

Consider the class `Queue` below

```

class Queue
{
 public:
 Queue();
 void add(int item);
 int remove();
 int processNext() const,

```

```

 bool isEmpty() const;
private:
 vector<int> data,
},

```

Intuitively, class Queue is similar to people standing in a queue at a bank, where you have to join the queue at the end. As each person's transactions are processed, you move closer to the front of the queue. Similarly, the class Queue has the following operations.

add() - adds an element to the end of the vector  
 remove() - removes the element at the front of the vector  
 processNext() - returns the element at the front of the vector without removing it  
 isEmpty() - returns true if the vector is empty or false otherwise

- 5.1 Write a template version of the Queue interface so that it can store a queue of any type, e.g. a queue of chars or a queue consisting of a user defined class. Do not provide an implementation. Provide only the interface. (5)
- 5.2 Implement the add member function of the template class Queue. (3)
- 5.3 Provide a declaration for a Queue of strings. (1)
- 5.4 Provide a declaration for a Queue of objects of type Rational. (1)

#### QUESTION 6

5 MARKS

Consider this function

```

void myFunction(int counter)
{
 if(counter == 0)
 return;
 else
 {
 cout << counter << endl;
 myFunction(--counter);
 return;
 }
}

```

- 6.1 For the above function, when will the recursion be considered infinite? (1)
- 6.2 Identify the base case in the above function. (1)
- 6.3 What is the purpose of the general case in a recursive function? (1)
- 6.4 Write down the output for the above function with the following function call  
     myFunction(4); (2)