

**COS1512**

October/November 2011

**INTRODUCTION TO PROGRAMMING 2**

Duration 2 Hours

75 Marks

**EXAMINERS**FIRST  
SECONDMS MA SCHOEMAN  
MS P LE ROUX

MS HW DU PLESSIS

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 / Instruksies:**

- 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

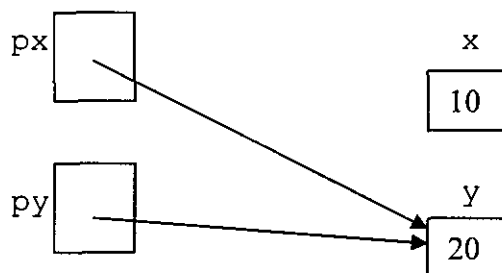
**GOOD LUCK!**

**QUESTION 1****[5]**

- 1.1 Consider the following declaration: (1)  
`char s[6];`  
 What is the maximum string length that can be stored in `s`? Explain your answer
- 1.2 What would be the expected output from the following? (2)  
`char sms[5] = "Gr8";`  
`int index = 0;`  
`while(sms[index] != '\0')`  
`{`  
 `sms[index] = 'x';`  
 `index++;`  
`}`  
`cout<<sms;`
- 1.3 Write C++ statements to accomplish the following (2)  
 • Declare a vector `secretlist` to store objects of type `int` - do not specify the size of `secretlist`  
 • Store the following values into `secretlist` 56 and 28  
 • Reset the first element of `secretList` to 12.

**QUESTION 2****[5]**

- 2.1 Consider the following declarations  
`int * px;`  
`int * py;`  
`int x = 10;`  
`int y = 20;`
- 2.1.1 Provide C++ statements so that pointer variables `px` and `py` point to variables `x` and `y` respectively (2)
- 2.1.2 Write a C++ code segment to recreate the pointer structure given below (1)



- 2.2 Give an example of a dynamic variable (1)
- 2.3 Discuss one advantage to using dynamic arrays (1)

**QUESTION 3****[35]**

Define a class `Booking` that represents one booking for a performance at a venue. This class has four member variables:

- `show`, a string that holds the name of the show or performance
- `nrTickets`, an integer value that indicates the number of tickets booked
- `seats`, a string that holds seats booked (either “standing” or the seat numbers, e.g. “A6–A10”)
- `customer`, a string containing the name of the person who made the booking

In addition, the class should have the following member functions

- A **default constructor** that initializes `show`, `seats` and `customer` respectively to an empty string. `nrTickets` should be initialized to 0
- An **overloaded constructor** that accepts a new booking and sets `show`, `seats`, `customer` and `nrTickets` to specified values
- A **destructor** that does not perform any action
- An **overloaded equality operator** `==` to compare two bookings. The `==` operator is implemented as a **friend** function with the following prototype  

```
bool operator==(const Booking & booking1, const Booking & booking2)
```

 This function returns `true` if `booking1` and `booking2` has been made for the same show and customer, and `false` if not
- An overloaded **operator+** for class `Booking` so that the following are feasible in the main program: `a = b + c`, where `a`, `b`, and `c` are all `Booking` objects. The overloaded **operator+** should add the values of the `nrTickets` member variables. It should also concatenate the value of the `seats` member variable of the second operand to the value of the `seats` member variable of the first operand. The `show` and `customer` member variables should be the same as for the first operand. The overloaded **operator+** should return a `Booking` object. The prototype is as follows  

```
Booking &operator+ (const Booking & b1, const Booking & b2);
```

 The overloaded **operator+** will allow a customer to buy more tickets and book the seats for the tickets under the same name
- A member function `calcFee()` to determine the amount charged for the tickets. Use the following prototype  

```
float calcFee();
```

 The fee for “standing” tickets is R100.00 per ticket. All other tickets cost R200.00 per ticket
- An overloaded **extraction operator** `>>` (implemented as a **friend** function) so that it can be used to input values of type `Booking`
- An overloaded **insertion operator** `<<` (implemented as a **friend** function) that displays all the member variables of a `Booking` object

You should attempt the solutions as follows

- 3.1 Create the header file `Booking.h` that contains the `Booking` class specification (8)
- 3.2 Create the implementation of the class `Booking` including all the friend functions (17)
- 3.3 Complete the application program (`main()`) below by citing the number and writing down the missing statement. This program obtains the detail for a booking from the user and creates an object to represent the booking. It then reads in all existing bookings from a file `Bookings.dat` and checks for a previous booking for the customer for the same show. If a previous booking has been made, the `+` operator is used to update the new booking to include the previous booking. The booking is then displayed as well as the amount due. Be sure to use appropriate member functions in the required statements (10)

```
#include <iostream>
#include <iomanip>

_____1_____ //1.Include files needed

using namespace std;

int main()
{
    _____2_____ //2.Declare input file

    _____3_____ //3.Open the file Bookings.dat and check
                       // that the file exists

//Add a booking
string bShow, bSeats, bCustomer;
int bTickets;
cout << "Enter show name: ";
cin >> bShow;
cout << "Enter seats requested: ";
cin >> bSeats;
cout << "Enter number of tickets requested: ";
cin >> bTickets;
cout << "Enter customer name: ";
cin >> bCustomer,

    _____4_____ //4.Instantiate object newBooking for the
                       // new booking

//check for existing booking
Booking aBooking;
bool found = false;

while (!found && _____5_____) //5. Read in a Booking object
                               //from the input file into object aBooking
{
    if (_____6_____ //6.Compare new booking with aBooking
        found = true;
}

//if previous booking has been made, update the new booking
if (found)
    _____7_____ //7.Update new booking

cout.setf(10s::showpoint);
cout.setf(10s::fixed);
cout << setprecision(2);
cout << "Booking confirmed:" << endl;

    _____8_____ //8.Display the new booking
//9. Calculate and display the amount due for the new booking
cout << endl << "Amount due: R" << _____9_____ << endl;

    _____10_____ //10. Close input file

return 0;
}
```

**QUESTION 4****[15]**

Consider the following class

```

class Loan
{
public:
    Loan();
    Loan(double amt, double rate, int term);
    double get_loanAmt()const;
    double get_intRate() const;
    double get_termOfLoan()const;
    void setAmt(double amt);
    void setRate(double rate);
    void setTerm(int term) ;
    void output(ostream & out) const;
    double calculate_interest() const;
protected:
    double loanAmt;
    double intRate,
    int termOfLoan;
};

```

4 1 Define a new class CarLoan that inherits the functionality of the class Loan. Class CarLoan has additional member variables

- string carModel (the model of the car)
- int modelYear (the year of the car)

Moreover class CarLoan has member functions, setModel() and setYear() to reset member variables carModel and modelYear respectively to values specified by parameters. CarLoan should override functions calculate\_interest() and output() of class Loan. Provide only the interface of class CarLoan in terms of a header file. The header file should contain compiler directives to prevent multiple definitions. Assume that the interface of class Loan is contained in file called Loan.h (7)

4 2 Implement the overloaded constructor for the class CarLoan by invoking the base class constructor (4)

4 3 Consider the following

```

double CarLoan::calculate_interest()const
{
    return (2*loanAmt * intRate * termOfLoan);
}

```

Suppose the access of member variables of class Loan were changed to private. Explain why the redefinition in CarLoan of calculate\_interest would result in compilation errors. Show how you would fix these errors (3)

4 4 Discuss one advantage of using inheritance in question 4 1 (1)

**QUESTION 5****[10]**

Consider the following class Database that maintains a database of customer names

```
class Database
{
public:
    Database();
    void insert(string n);
    void swap(int pos1, int pos2);
    int count();
private:
    vector<string> myData;
};
```

The class Database has the following operations

insert – adds a string to the database

swap – swaps the strings at positions pos1 and pos2, i.e. the string at position pos1 of myData is moved to position pos2, and vice-versa

count – returns the number of strings in the database

- 5.1 Write a template version of the Database class. In other words, redesign the Database interface so that it may be used to create a database of any type of data. For example, Database that contains objects of type Student, Car or Person. Provide only the interface. (5)
- 5.2 Provide the implementation of function swap from the template class Database. (4)
- 5.3 Provide a declaration for a Database object intended to contain objects of type of Student. (1)

**QUESTION 6****[5]**

Function iterativeTriangle() below accepts as a parameter a nonnegative integer and generates a triangular pattern of stars. For example, if the nonnegative number is 4, then the following pattern is generated

```
****
***
**
*
```

```
void iterativeTriangle(int x)
{
    if (x == 0)
        return;
    else
    {
        for (int i = 0; i < x; i++) {
            for(int j = x; j > i; j--) {
                cout<<"*";
            }
            cout<<endl;
        }
    }
}
```

- 6.1 If we rewrite function iterativeTriangle() as a recursive function recursiveTriangle(), what would be the base case? (1)

- 6 2    **Why do we have to include a base case in a recursive function?** (1)
- 6 3    **What is the purpose of the general case in a recursive function?** (1)
- 6 4    **Write down the function header only for function `recursiveTriangle()`** (2)