SECTION A – TRUE/FALSE QUESTIONS [TOTAL 9 MARKS]

Instruction:
This part consists of **SIX (6)** questions. For each question, identify whether the statement in the question is TRUE or FALSE. Verify your answer with appropriate reasons. Each question carries 1.5 marks (0.5 marks for the answer and 1.0 mark for the reason).

1. The declarations for three abstract data types, **`Date`**, **`S_Dates`** and **`U_Dates`**, are given below. Then, variables **`d1`** and **`d2`** are declared of type **`S_Dates`** and **`U_Dates`** respectively.

```
 1  struct Date {
 2    int day, month, year;
 3  };
 4
 5  struct S_Dates {
 6    Date birthDay;
 7    Date registrationDay;
 8    Date graduationDay;
 9  };
10
11  union U_Dates {
12    Date birthDay;
13    Date registrationDay;
14    Date graduationDay;
15  };
16
17  S_Dates d1;
18  U_Dates d2;
```

The memory allocated for **`d1`** and **`d2`** are the same.

2. Given the declarations of an enumerated data type and two variables as follow:

```
1  enum CarModel { KANCIL=1, KENARI, PREVE, SATRIA,
2                  WAJA, WIRA };
3  CarModel p=WIRA;
4  int calc;
```

The statement below is valid.

```
calc = (p - KENARI) / SATRIA;
```

3. The members of a class can be of variables and functions but not instances of other classes.
4. New member variables and functions cannot be added to a derived class.
5. Suppose a class named **`BankAccount`** has two constructors as given in the following class declaration.

```
1  class  BankAccount{
2    private:
3        float balance, rate;
4
5    public:
6        BankAccount()
7        { balance = rate = 0.0; }
8
9        BankAccount(float aBalance, float aRate)
10       { balance = aBalance;  rate = aRate; }
11
12       // declarations of the remaining methods go here
13 };
```

All the statements below are correct to create objects of `BankAccount` and initialize their data.

```
BankAccount account1;
BankAccount account2();
BankAccount account3(3000.00, 0.067);
BankAccount account4 = BankAccount(3000.00, 0.067);
```

6. Refer to the following class and function declarations:

```
1  class Appointment{
2        private:
3          int day, month, year;
4        public :
5          Appointment();
6          void setAppoiment(int aDay,int aMonth, int aYear);
7          friend void changeAppoiment (Appointment &);
8  };
9
10 void viewAppointment (const Appointment &);
11 void changeAppointment (Appointment &);
```

The private members of class **`Appointment`**; **`day, month`** and **`year`**, can be accessed directly by the function **`viewAppointment`**.

SECTION B – STRUCTURED QUESTIONS [TOTAL 66 MARKS]

Instruction:

This part consists of **FIVE (5)** structured questions. The marks for each part of the question is as indicated.

Question 1 (Total – 20 marks)

a) Declare a structure data type named **`TimeType`**, with the following members: **`h`**, **`m`** and **`s`**, representing the components of hour, minute and second of a time, respectively. (2 marks)

b) Declare another structure data type named **`TourType`**, with the following members: **`city`**, **`distance`** and **`t`**. The member **`t`** represents the duration needed to travel to the city and is of type **`TimeType`**. (2 marks)

c) Write a function named **`displayInfo`** to print out data stored in a variable of type **`TourType`**. The data to be printed out should come from the function's parameter. (3 marks)

d) Write a function named **`inputData`**, which prompts the user to key in structure data member values. The function should return the input read as data of type **`TourType`**. (4 marks)

e) Write a void function named **`inputInfo`** which accepts a pointer parameter of type **`TourType`**. The function should read input from the keyboard and assign them to the corresponding members of the structure data. (3 marks)

f) Declare a variable of type **`TourType`** named **`place`**. (1 mark)

g) Write C++ statements to specify the members of **`place`** as follow**:**

`city` – Langkawi.

`distance` – 550 kilometers.

`t` – 9 hours and 30 minutes.

(2 marks)

h) Write C++ statement(s) to specify the members of **`place`** by calling to the function **`inputData`**.

(1 mark)

i) Write C++ statement(s) to specify the members of **`destination`** by calling to the function **`inputInfo`**.

(1 mark)

j) Write C++ statement(s) to display the information in **`destination`** by calling to the function **`displayInfo`**.

(1 mark)

Question 2 (Total – 10 marks)

a) Declare a class named **`Circle`** with a private member variable named **`radius`**. The default constructor of the class initializes **`radius`** to **`0`**. The class also has an overloaded constructor which sets **`radius`** to a value obtained from the constructor's argument. Furthermore, provide a function named **`getArea`** that returns the area of the circle.
Notes: the area of a circle is expressed as follow; where *r* is the radius of the circle.

$$3.14159 \times r^2$$

(5 marks)

b) Using the class `Circle` from 2(a), declare an array named **`collection`** to hold three circles and initialize the radius for each circle to **`20.5`**, **`17.2`**, and **`22.3`**, respectively.

(2 marks)

c) Using an appropriate loop, calculate and then print the total the area of the circles held in the array **`collection`**.

(3 marks)

Question 3 (Total – 10 marks)

a) i. Explain the differences between a class specification file and a class implementation file? (2 marks)

ii. Assume the following class components exist in a program:
`BasePay` class declaration,
`BasePay` member function definitions,
`Overtime` class declaration, and
`Overtime` member function definitions.

In what files would you store each of these components? (2 marks)

b) The program in Figure B1 is a complete object-oriented C++ program. Break this program and rewrite them into three separate files: the specification, the implementation and the application files respectively. (6 marks)

```
1  #include <iostream>
2  using namespace std;
3
4  class Square{
5    private:
6         double side;
7    public:
8         Square(double newSide)
9         { side = newSide; }
10
11        double getSide()
12        { return side; }
13 };
14
15 int main()
16 {
17   Square square(4.2);
18   cout << "The length of the square is: " << square.getSide();
19   return 0;
20 }
```

Figure B1

Question 4 (Total – 13 marks)

The following program generates tickets for tourists visiting a historic palace. The program allows the management to limit the maximum number of tourist per day. There are two machines that will print the ticket. Figure B2 shows an example run of the program.

```
Enter maximum number of tourist to enter AlHamra: 3

Press button to get your ticket

Your ticket number is 1
From machine 2

Press button to get your ticket

Your ticket number is 2
From machine 2

Press button to get your ticket

Your ticket number is 3
From machine 1

Total tickets generated from both machines:  3
```

Figure B2: An example run of the program

Based on the information given and the instructions stated as comments in the following program, i.e. labeled as **(a)** – **(j)**, complete the program with appropriate C++ code to implement the ticket generator.

```
#include <iostream>
#include <cstdlib>

using namespace std;

class TicketMachine;

double machineUtilization(const TicketMachine &machine);

class Maintenance
{   public:
        Maintenance();
        void serviceMachine(TicketMachine &machine);
};
```

```
class Ticket
{   private:
         int number;
    public:
         void setNumber(const TicketMachine &machine);
         void print() const;
};

class TicketMachine
{
    private:
       static int totalTourist;  // The total number of tourist for the day.
       static int ticketNumber;  // A counting number representing the current ticket number.
       int maxTicket;            // The maximum number of tickets that can be created by the machine.
       int totalServed;          // The actual number of tickets generated by a ticket machine

    public:
        TicketMachine( int theMaxTicket=0)
        { maxTicket = theMaxTicket;
          totalServed = 0;
        }

        bool isTicketSoldOut() const
        {  bool result;
           if (ticketNumber >= totalTourist) result = true;
             else result = false;
            return result;
         }

         void generateNumber()
         { ticketNumber++;
           totalServed++;
         }
```

// **(a)**. Define a static member function named **`setMaxTourist`**.
// This function sets the member variable **`totalTourist`** to a value
// obtained from the function's argument. (1.5 marks)

// **(b).** Define a copy constructor that copies the value of **`maxTicket`**
// from other ticket machine. (1.5 marks)

// **(c).** Define an overloaded operator for the operator **+** such that when two machines,
// *e.g.* **`p`** and **`q`**, are added, the result will be the sum of the **`totalServed`**
// of both machines. For example, if the values of **`p.totalServed`**
// and **`q.totalServed`** are **4** and **5**, respectively, the operation
// **`machine1 + machine2`** will evaluate to **9**. (1.5 marks)

```
      // (d). Write statements that specify all the followings  to be friends
      //          to the class TicketMachine:
      //           - the function machineUtilization,
      //           - all member functions from the class Ticket, and
      //           - only the member function serviceMachine from the class Maintenance.
      //                                                                    (1.5 marks)

};   // End of class TicketMachine

// (e). Initialize the static member variables from the class TicketMachine,
//          i.e, totalTourist and totalServed, to 0.                      (1 mark)


double machineUtilization(const TicketMachine &machine)
{ return machine.totalServed / machine.maxTicket; }

Maintenance::Maintenance()
{}

void Maintenance::serviceMachine(TicketMachine &machine)
{
    machine.totalServed = 0;
}

void Ticket::setNumber(const TicketMachine &machine)
{
    number = machine.ticketNumber;
}

void Ticket::print() const
{
    cout << "Your ticket number is " << number << endl;
}

int main()
{
   int maxTourist;

   TicketMachine machines[2]; // To create two TicketMachine objects to generate tickets
   int machineIndex;
   Ticket ticket;


   cout << "Enter maximum number of tourist to enter AlHamra: ";
   cin>> maxTourist;


   // (f). Set the maximum number of tourist for the day.                  (1 mark)


   // (g). Declare another TicketMachine object, named baseMachine
   //          with the maximum ticket it can generate is set to 10.       (1 mark)
```

```
    // (h). Using the copy constructor from the class TicketMachine, copy the object
    //          baseMachine to each machine held in the array machines.          (1.5 marks)


    do
    {
        cout<< "Press button to get your ticket"<< endl;
        cin.get();
        machineIndex = rand()% 2;       // To simulate a uses buys ticket from any of the machines.

        // (i). Generate a ticket number from the corresponding machine.          (1 mark)


        ticket.setNumber(machines[machineIndex]);
        ticket.print();
        cout << "From machine " << machineIndex + 1 << endl << endl;

    } while ( ! machines[machineIndex].isTicketSoldOut() );


    // (j). Using the overloaded operator + from the class TicketMachine, determine and
    //          print the total number of tickets generated by both machines.
                                                                          (1.5 marks)

    return 0;
}
```

Question 5 (Total – 13 marks)

a. Explain with appropriate examples the difference between the concepts of aggregation and inheritance.

(4 marks)

b. Given class declarations in the following code segment. Explain or draw the relationship between the class **`Person`**, **`TextBook`** and **`Course`**.

(3 marks)

```
1  class Person
2  {
3     private:
4              string name;
5  };
6
7  class TextBook
8  {
9        private:
10               Person author;
11 };
12
13 class Course
14 {
15       private:
16               TextBook book;
17               Person instructor;
18 };
```

c. Based on the following code segment, explain the difference between the use of the keyword `protected` at **line 1** and the use of the keyword `protected` at **line 3**.

(2 marks)

```
1  class Car: protected Vehicle
2  {
3        protected:
4                int horsePower;
5                double weight;
6
7  };
```

d. Write the output of the following program. (4 marks)

```
1  #include<iostream>
2  using namespace std;
3
4  class Mother
5  {
6        protected:
7                    string name;
8        public:
9                 Mother()
10                { name ="Mom"; }
11
12               Mother(string aName)
13                { name = aName; }
14
15               void display ()
16                {  cout << "Mother's name: " << name << endl; }
17 };
18
19 class Daughter : public Mother
20 {
21       private:
22                Mother mommy;
23       public:
24                Daughter(): mommy("Ummi")
25                { name = "Baby"; }
26
27                void display ()
28                {
29                    cout << "Daughter's name: " << name << endl;
30                    Mother::display();
31                    mommy.display();
32                 }
33 };
34
35 int main ()
36 {
37    Mother mom("Ana");
38    Daughter rita;
39
40    mom.display();
41    rita.display();
42
43    return 0;
44 }
```

SECTION C - PROGRAMMING QUESTION **[TOTAL 25 MARKS]**

This section consists of **ONE (1)** question only.

Question

Using Object-oriented Programming approach, develop a C++ program for a shipping company to manage their inventory of ships. The company keeps the following information about each ship:

- The name of the ship
- The year that the ship was built
- The address that the ship was registered at. It includes the country where the ship was registered and the registrar office that keeps the registration record.

The UML class diagram for the inventory system is given as in Figure C1. In class **Address**, the function **set** is a mutator used for specifying the member variables, **registrar** and **country**, with values obtained from the function's arguments, while the functions **getRegistrar** and **getCountry** are accessors.

As for the class **Ship**, the function **read** is a mutator that specifies all the member variables, **name**, **yearMade** and **address**, with values obtained from the keyboard. The function **print** on the other hand, is used to print the member variables.

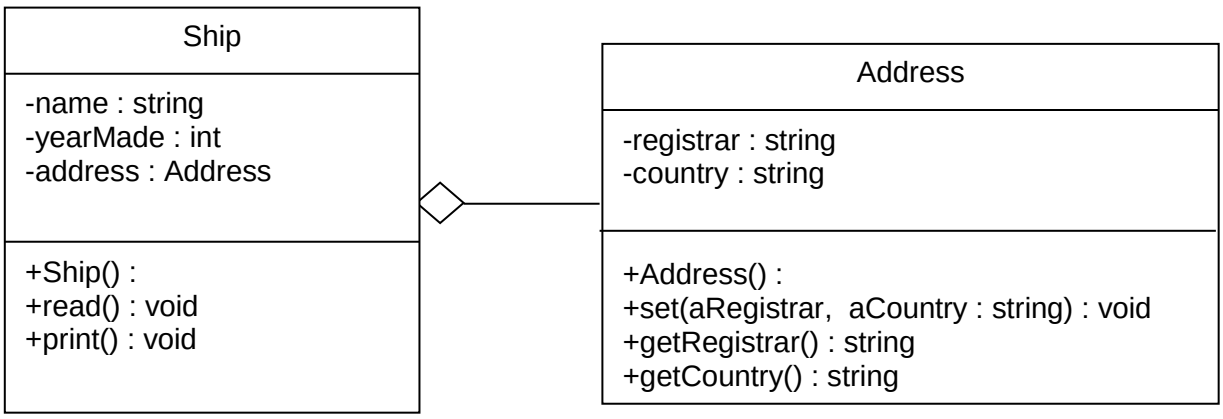

Figure C1: The UML class diagram for the ship inventory system

The implementation of your program must be based on the class diagram and specifications given above. Also, your program should use a dynamically allocated array to store the list of ships. Furthermore, the program should provide a menu-driven interaction for the user to use the program. It should allow the user to add the record for a ship one at a time. It should also provide an operation to display the number of ships and the information of each ship. Figure C2 illustrates an example run of the program. Note that the **bold texts represent** user inputs.

```
======== MENU ========
1. Add a ship
2. Display ships
3. Exit

Choose an operation  => 1


<<< Enter the information of the ship >>>

Ship Name: Super Gemilang
Year Built: 2010
The address the ship was registered:
Registrar Office: Malaysian Shipping Council
Country: Malaysia


Press any key to continue . . .


======== MENU ========
1. Add a ship
2. Display ships
3. Exit

Choose an operation  => 1


<<< Enter the information of the ship >>>

Ship Name: Orion Cruise
Year Built: 2000
The address the ship was registered:
Registrar Office: Shipping Authority Board
Country: Singapore

Press any key to continue . . .
```

```
======== MENU ========
1. Add a ship
2. Display ships
3. Exit

Choose an operation  => 2

<<< Inventory of ships >>>

Total ship: 2

==== Ship List ====

Ship Name: Super Gemilang
Year Built: 2010
Registered at:
  Malaysian Shipping Council, Malaysia

Ship Name: Orion Cruise
Year Built: 2000
Registered at:
  Shipping Authority Board, Singapore


Press any key to continue . . .


======== MENU ========
1. Add a ship
2. Display ships
3. Exit

Choose an operation  => 3
```

Figure C2: Example Run