INDEX#

MILITARY COLLEGE OF SIGNALS
FINAL EXAM - BESE15
OBJECT ORIENTED PROGRAMMING PARADIGM

Instructor: Aisha Khalid Khan

Time: 150 Minutes
Max Marks: 50

Instructions
1. Understanding of questions is part of the examination.
2. State any assumptions (which you think are necessary) at the beginning of your solution.
3. Assume in all programs necessary header files are included.
4. Do not use a lead pencil for answering any of the questions.
5. Attach the question paper at the back of your answer booklet. Do not forget to write your **index#** on the question paper.
6. You have to do **question#2** and **question#4** on this **question paper**.
7. This question paper has **5** pages.

Question 1: Give short and comprehensive answers to the following questions. **[11 points]**

a) When are the following objects/variables created and when are they destroyed in a program? [2]
   i. `static` objects/variables
   ii. Dynamic objects/variables
   iii. Local objects/variables
   iv. Base class part of a derived class object

b) Why can't the stream insertion operator **`<<`** be overloaded as a class member function and still retain its standard functionality (i.e., `cout<<x`; where *x* is an object type)? [1]

c) Given the declarations [2]
```
Person p("Mehmood"); //Person is the base class
Employee e("Ayaz", 20000); //Employee is derived from Person
```
Which of the following is/are illegal? Explain why?
   i. `e = p;`
   ii. `p = e;`
   iii. `Person *ptr = &e;`
   iv. `Employee *eptr = &p;`
   v. `Person& pref; pref = e;`

d) Comment on why this statement is true or false.
If you overload operator+, you *must* return an object of the same type as your parameters. Otherwise, it won't compile. For example, the following code will *not* compile:
`int operator+(const Foo& lhs, const Foo& rhs);` [1]

e) What is the difference between the compiler directives `#include <xyz>` and `#include "xyz"` ? [1]

f) Consider a `class ABC` which has a member function `void foo()`, how can you write a statement in another `class DEF` so that all member functions of `ABC` become friends of `DEF`? How will you write a statement which makes only `foo()` a friend of `DEF`? [1]

g) Consider the following class [3]
```
class C {
     private:
          int pr;
     public:
          int pu;
          const int cu;
          static int su;
};
```
How can you initialize each of the four members of class C? How can you change any of the members to 100? If there is more than one way, show all different ways. If impossible, state why?

[Bonus Questions: Each question carries 1 mark]

h) Is it possible to prohibit creation of objects of a class without using virtual functions? If so, how can it be achieved?

i) Who is the author of the book Thinking in C++ ?

j) How would you define a new datatype 'Boolean' that represents the two values 'true' and 'false' ?

k) Write down instruction# 4,5 and 6 on your answer booklet and also follow them.

Question 2: What would be the output of the following code segments? Write your answers in the spaces provided. Do not forget to dry run the programs on your answer booklet. **[1.5+5+5+1.5]**

(PART A) **]**

```
class Base {
public:
  virtual void foo() {
   cout << "Base::foo()" << endl;
  }
  virtual void foo(int i) {
   cout << "Base::foo(" << i << ")";
  }
};
```

```
class Der : public Base {
public:
  virtual void foo() {
    cout << "Der::foo()" << endl;
  }
};
void main()
{
  Der d1;
  d1.foo(3);
}
```

Output

(PART B) **]**

```
class A {
 public:
  A(){cout<<"A ctor"<<endl;}
  A(const A& a) {
      cout<<"A copy ctor"   << endl;
  }
 virtual ~A() {
      cout<<"A dtor"<<endl; getch();
 }
 virtual void foo() {
      cout << "A foo()" << endl;
 }
 virtual A& operator=(const A& rhs)
{
      cout << "A op=" << endl;
 }
};
```

Output

```
class B : public A {
 protected:
  A a; // don't forget about me!
 public:
  B() { cout << "B ctor" << endl; }
  virtual ~B() {
      cout<<"B dtor"<<endl; getch();
  }
  virtual void foo() {
      cout << "B foo()" << endl;
  }
};

A foo(A& input) {
      input.foo();
      return input;
}

void main() {
      B myB; cout<<endl;
      A myA; cout<<endl;
      myA = foo(myB); cout<<endl;
      getch();
}
```

PTO

(PART C))

```
class A{
  public:
   int i;
   A() :i(1){};
   int f(){return (i+1);}
   virtual int g(){return (i+2);}
};

class B: public A{
  public:
   int f(){return 3;}
   virtual int g(){return 4;}   };
```

```
class C: public A{
 private:
   int i;
 public:
   C():i(0){}
   virtual int g(){return (5+i+A::i);}
   void setI(int ii)
   {
     i = ii;
   }
};
```

```
void main()
  A *pa;
  A a;
  B b;
  C c;
  c.setI(5);
  b.i=3;
  c.A::i=5;
  pa=&a; cout<< pa -> f()<<endl; cout<<pa -> g()<<endl;
  pa=&b; cout<< (pa -> f() + pa -> g())<<endl;
  pa=&c; cout<< pa -> f()<<endl; cout<<pa -> g()<<endl;
}
```

Output

(PART D))

```
int& func(int& x)
{
   x--;
   return x;
}

void main()
{
  int x = 10;
  func(x) = 5;
  cout<<"\nAnswer is:"<<func(x);
}
```

Output

Question 3: The following code does not produce an error when it is compiled; however, there is still something logically wrong with it. What is that? How can it be corrected? **[3]**

```
int *get_i()
{
   int i = 2;
   return &i;
}
```

```
void main()
{
  int *i;
  i = get_i();
  i += 5;
 cout<<i;
}
```

PTO

Question 4: Please attempt this question in the box given below. Cross out all statements in **main( )** that will not compile. State in one sentence why each error is caused. **[4]**

```
class A{
 public:
  A(){}
  virtual int output() = 0;
 private:
  int i;
};

class B: public A{
 private:
  int j;
};
```

```
class C{
 public:
  int f(int a){return x*a;}
 protected:
  void setX(int a){x=a;}
  int getX(){return x;}
 private:
  int x;
};

class D: public C{
 private:
  int z;
};
```

```
void main(){
      A objA;
      B objB;
      C objC;
      D objD;
      C.setX(2);
      cout<<C.getX();
      D.setX(1);
      D.f(3);
}
```

Your Answer

Question 5: **[7]**

In this question, you are going to implement a class hierarchy for representing the customers of a mobile phone company. There are only two types of customers, postpaid and prepaid. Postpaid customers have to pay a fixed line rent every month which is added to their monthly bill. The call rate per minute is different for the two types of customers. So, the billing function for the two types of customers is also different

Monthly bill for prepaid customers = number of minutes of outgoing calls * call charges for each minute
Monthly bill for postpaid customers = line rent + (number of minutes of outgoing calls * call charges for each minute)

Implement the class hierarchy which will consist of a **`customer`** class, a **`prepaid_customer`** class and a **`postpaid_customer`** class, so that the following code in main (see next page) can be executed properly. Use a dynamic array to hold customer's name and provide the appropriate constructors and destructors (you **do not** have to write copy constructor or assignment operator). Keep all your data private. Make all functions that will be required to enable the following code to run properly. You do not have to rewrite the code in `main().` You only have to write one additional line in `main()` to delete all customers created using **`new`** (as mentioned in the code on the next page).

PTO

```
void main()
{
     customer* Array[3];
     Array[0] = new prepaid_customer (“Omar Bilal”, 400, 2.5); //a customer named
               //Omar Bilal has made outgoing calls of 400 minutes in one month
               //and call charges for each minute are Rs.2.5

     Array[1] = new postpaid_customer(“Shaista Khan”, 510,1.5,150); //a customer
               //named Shaista Khan has made outgoing calls of 510 minutes in
               //one month, call charges for each minute are Rs.1.5 and the
               //monthly line rent is Rs.150
     Array[2] = new postpaid_customer(“Rubina Sheikh”,544,1.25,250);
     for(int i=0;i<3;i++)
     {
          cout << “Customer” <<array[i].getName() << “owes Rs.”
               << array[i].compute_bill()<<endl;
     }
     //WRITE A STATEMENT TO DELETE ALL CUSTOMERS
} //end main
```

Question 6: **[4]**

Write a function that copies two arrays of **`integers`** into a single, larger array of **`integers`** which is returned. Before returning the array, the function also prints out the single larger array after the copy. The prototype for your copy function should look like this:

```
int *copy_arrays(int a[], int a_size, int b[], int b_size); //where a_size is the
                //size of array a[] and b_size is the size of array b[]
```

Question 7: **[8]**

For this question, write the classes in separate header (.h) and the source (.cpp) files. So, you will be creating 4 files **`(Vehicle.h, Vehicle.cpp, Car.h, Car.cpp)`**

1. Implement a class for a **`Vehicle`**. Each vehicle has a **`registration_number`** (use dynamically allocated array of **`char`**) and **`mileage`** (use **`long`**). Both of these members are private. Write a 2-argument constructor with default arguments **`“No Number”`** for **`registration_number`** and **`0`** for **`mileage.`** Implement a copy constructor for **`Vehicle.`**
2. Derive a class **`Car`** from **`Vehicle`**. A car also has **`owner`** (use **`string`**). Write a 3-argument constructor with default arguments **`“No Car Number”`** for **`registration_number`** , **`0`** for **`mileage`** and **`“No name”`** for **`owner`**. Implement a copy constructor for **`Car.`**
3. Suppose you want a polymorphic **`show()`** method for both **`Vehicle`** and **`Car`** applicable for dynamic binding. The methods just display all the information about the **`Vehicle/Car`**. Show all changes needed in the header and source files created above.
4. Implement a global (non-member) function that will allocate a **`Vehicle`** or a **`Car`** based on an integer argument (0 for **`Vehicle`**, anything else for **`Car`**). The function will allocate the object, and then return it to the calling program. After writing the function, write a single statement to demonstrate how this function will be called in main using a **`Vehicle`** pointer.
5. **[Bonus (1 mark)]** How can the function in (4) be called in main using a reference of a **`Vehicle`** object?
6. How can you make the **`Vehicle`** class abstract?

x--------------------x------------------------x-----------------------Good Luck ☺---------------x---------------x-------------------------------x