B.C.A Sem 4 Exam paper solution

Page No. 6
Date: / /

401 - Object oriented programming c++

Marks 70.

Q1 (a) Explain the following terms. [06 marks]

1] Class

=> A class is a fundamental block of program that has its own set methods and variables. You can access these methods and variables by creating an object or the instance of the class.

2] Object

=> An object is an instance of a class that encapsulates data and functionality pertaining to that data.

3] Encapulation

=> The wrapping up data and functions together in a single unit is known as encapsulation. It can be achieved by making the data member's scope private and the function's scope public to access these data members. Encapsulations makes the data non-accessible to the outside world.

Page No. 7
Date: / /

Que-1 (b) Answer the following (any two) [12 marks]

1] Explain Enumerated data type in detail with example.

=> An enumeration is user-defined data type that consists of integral constants. To define an enumeration keyword enum is used.

enum {spring, summer, autumn, winter};

Here the name of the enumeration is season.

And, spring, summer and winter are valued of type seasons.

By default spring is 0, summer is 1 and so on. you can change the default value of an enum element during declaration (if necessary)

Eg:

```
enum season
{ spring = 0,
  summer = 4,
  autumn = 8,
  winter = 12,
};
```

* Enumerated type Declaration

When you create an enumerated type only blue-print for the variable is created. Here's how you can create variables of enum type.

```
enum boolean { false, true };
// inside function enum boolean check;
```

Example 1: Enumeration type

```
#include<iostream>
#include<conio.h>
using namespace std;
enum week {sunday, monday, Tuesday,
           wednesday, thursday, Friday,
           Saturday};

void main()
{
 week today;
 today = wednesday;
 cout<<"Day "<<today+1;
 getch();
}
```

Output

Day 4

Date: / /

2] What is reference variable? Explain it with syntax and example.

=> A reference variable is one that refers to the address of another variable. It represents the name of another variable location or value once you initialize the variable references, that variable will be referred to using the variable name or a reference name.

The developers created the reference by using the symbol ampersand (&), to easily identify the variable as reference with the help of this symbol.

Syntax

```
Data_type variable_name = &existing_variable
```

Data_type: It is the basic data type that has been used for declaration.

variable_name: It is a reference variable name used to identify the existing variable. It It is like an identifier.

&existing_variable: It represents the reference or address of the existing variable.

Sample code:

```
String s = "anora";
String s1 = &s;
```

Date : / /

Example program

Consider the program to swap the two numbers using the reference variable as parameter.

```
#include <iostream>
#include <conio.h>

void swap_out(int &d1, int &d2); // Function declaration

void main()
{
int d1=100;
int d2=200;
std:: count<<" Before swap, value 1 is"<<d1<<"\n";
std:: cout<<" Before swap, value 2 is: "<<d2<<"\n";

swap_out(d1, d2); // function calling

std:: cout<<" After swap data value 1: "<<d1
<<"\n";
std:: cout<<" After swap, data value 2:"<<d2
<<"\n";
getch();
}
void swap_out(int& d1, int &d2)

{
 int temp;
```

```
temp = d1; /* save the value at address d1 */
d1 = d2;;
d2 = temp; // put d1 into d2
getch();

}
```

Output

```
Before swap, value 1 is 100
Before swap, value 2 is:200
After swap data, value 1 : 200
After swap, data value 2 : 100
```

Or

3] Explain Memory management operator with example.

=> As we know that new operator is used to create memory space for any data-type or even user defined data type such as an array structures, unions, etc. So the Syntax for creating a one-dimensional array.

pointer-variable = new data-type[size];

=> there are two main operators of memory management are:

1] New operator

2] Delete operator

1] New operator

=> A New operator is used for creating the object which exists and remains in mode which means the allocation of memory will still be active. This remains in active state the existence of a new object is there until the delete() operator is called which will be discussed in the next section.

Syntax

ptr_var = new data_tp

- ptr_var: this represents the name of the pointer variable.
- new: operator for the location of the object for allocation.
- data_tp: represents the type of data used while allocation.

~~code~~
~~#include <iostream>~~
~~#~~

delete operator

⇒ once we no longer need to use a variable that we have declared dynamically we can deallocate the memory occupied by the variable. For this the delete operator is used It returns the memory to the operating system. This is known as memory deallocation.

Syntax

```
delete pointervariable;
```

code

Example:

```
# include <iostream.h>
# include <conio.h>
using namespace std;

void main()
{
// declare an int pointer
int* pointint;
```

```
float * pointFloat;
// dynamically allocate memory
pointInt = new int;
pointFloat = new float;

// assigning value to the memory
* pointInt = 45;
* pointFloat = 45.45F;

cout << *pointInt << endl;
cout << * pointFloat << endl;

// deallocate the memory
delete pointInt;
delete pointFloat;

getch();
}
```

Output

45
45.45

Date: 1 1

Q-2 (a) Answer the following [±0.5 marks]

1] Explain Inline Function.

⇒ the Inline keyword suggests that the compiler substitute the code within the Function definition in place of each call to the Function. In theory, using inline Functions can make your program Faster because they eliminate the overhead associated with Function calls.

⇒ Calling a Function requires pushing the return address on the stack, pushing arguments onto the stack, jumping to the Function body, and then executing a return instruction when the Function finishes.

2] Write a short note on destructor.

⇒ A destructor is a member Function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete or delete[]. A destructor has the same name as the class and is preceded by a tilde (~). For example the destructor For class String is declared.

Q-2[b] Answer the following (any two) [12 marks]

1] Explain function overloading with Example.

=> Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function overloading In function overloading can be considered as an example of a polymorphism feature in c++.

=> If multipale functions having same ~~but~~ name but parameters of the function's should be different is known as function overloding.

=> If we have to perform only one operation and having same name of the functions increased the readability of the program.

=> Suppose you have to perform addition of the given numbers but there can be any number of arguments. If you write the function such as a(int,int) for two parameters and b(int,int,int) for three parameters then it may be difficult for you to understand the behavior of the function because its name differs

Date: / /

The parameters should follow any one or more than one of the following conditions for function overloading

- Parameters should have a different type
  add(int a, int b)
  add(double a, double b)
- Example program for function overloading.

```
#include <iostream>
#include<conio.h>

using namespace std;

void add(int a, int b)
 {
  cout<<"sum = "<<(a+b);
 }
void add(double a, double b)
 {
  cout << endl <<"sum = " << (a+b);
 }
void main()
{
 add(10, 2);
 add(5.3, 6.2);
 getch();
}
```

output

```
Sum = 12
Sum = 11.5
```

2] What is constructor? Explain parameterized constructor with example.

=> Constructors are special function named after the class and without a return type and are used to construct objects. Constructors, like function, can take input parameters.

=> Constructors are used to initialize objects. The parameterized constructors are the constructors having a specific number of arguments to be passed. The purpose of a parameterized constructors is to assign user-wanted speific values to the instance variables of different objects.

Example

=> When we create the object like thise students student ("Joes", 15); then invokes the parameterized constructor with int and string parameters student (string n, int a after object creation

=> The program defines a class named math which has private data members a, b and c, and a public member function add(). The math(int x, int y) function is a parameterized construtor that takes two integer arguments x and y

and initializes the private data members a and b with these values.

⇒ The add() function calculates the sum of the private data members a and b with these values. and member c It then outputs the total sum to the standard output using cout.

⇒ In the main() function an object of the Math class is created using the parameterized constructor and passing the integer values 10 and 25 as arguments. The add() member function of the Math class is then called on the object to calculate and display the sum of a and b.

- Example program

```
#include <iostream.h>
# include <conio.h>

Using namespace std;
// Default constructor parameterized constructor

Class math
{
private:
  int a, b, c
public:
```

⟹

```
Math(int x, int y)
 {
    a=x;
    b=y;
 }
void add()
{
   c = a+b;
   cout <<" total : " <<c;
}

};
void main()
{
  Math o(10,25);
  o.add();
  getch();
}
```

Output

total : 35

OR

3] Explain static data member and static member Function with example.

=> Static data member are class members that are declared using static keywords.
A static member has certain special characteristics which are as follows:

- only one copy of that member is created for the entire class and is shared by all the objects of the class no matter how many objects are created.
- It is initialized before any object of this class is created, even before the main starts.
- It is visible only within the class but its lifetime is the entire program.

Syntax
static data-type
data-member-name;

Example program

```
#include <iostream.h>
#include <conio.h>

Using namespace std;

Class A {
public:
```

=>

```
A()
{
 cout <<" A's constructor called " << endl;
}
class B {
    static A a;

public:
  B()
  {
  cout <<" B's constructor called "<< endl;
  }

};
void main
{
  B b;
getch();
}
```

output

B's constructor called

- Static member function in C++

=> The static keyword is used with a variable to make the memory of the variable static once a static variable is declared its memory can't be changed

Date: / /

Example

```
Class person
 {
  static int index_number;
 };
eap

#include <iostream.h>
#include <conio.h>
using namespace std;
class student {
public:
   static int total;
   student() { total += 1; }
};
int student:: total = 0;
void main()
{
student s1;
 cout <<" Number of students :" << s1. total
      << endl;
  Student s2;
  cout <<" Number of students :" << s2. total
       << endl;
  Student s3;
  cout <<" Number of students :" << s3. total
       << endl;
  getch();
```

Output

```
Number of students: 1
Number of students: 2
Number of students: 3
```

Q-3[A] Answer the following [06 mark]

1] What is operator overloading? Give the Syntax.

⇒ operator overloading is a compile-time polymorphism. For example we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. other example classes where arithmetic operator's may be overloaded are complex Number Fractional numbers Big integers etc.

Syntax

class nameofclass {public returntype operator symbol (arrguments) {}};

2] List the operator which are not overloaded.

⇒ there are total ~~three~~ six types of operator that can not overloaded.

1] conditional or ternary operator (?:)

2] Sizeof

3] Scope Resolution operator

4] class member selector operator (.)

5] Member pointer selector operator (.*)

6] Object type operator

Date : / /

3] Define: unary operator, Binary operator.

=> unary operator :- ~~operators that operate on two operands~~ unary operators are the operators that perform operations on a single operand to produce a new value.

=> Binary operators :- (follow a specific syntax they are placed between the LHS and the RHS) A Binary operator is an operator that operates on two operands to produce a new value (Result) - Most common binary operators are +, −, *, /, etc Binary operators in C are further divided into Arithmetic operators

Q-3[b] Answer the following [Any two] [12 marks]

1] Write a program for implementation of binary plus ('+') operator using member function.

=>
```
#include<iostream.h>
#include<conio.h>

class Number
{
 private:
     int value;
 public:
    Number(int num): value(num) {}

 Number operator +(const
  number & other)
 {
  Number result(value + other.value);

  return result;
 }

 void display()
 {
   std::cout<<"value:" <<value<<std
      ::endl;
 };
```

```
void main()
{
Number num1(5);
Number num2(10);

Number sum = num1 + num2;

  sum.display();

  getch();

}
```

output

15

=> In this program we define a class called "Number" that represents a number. We have a private member variable "value" to store the actual value. The class has a constructor to initialize the value.

=> the operator + function is defined a member function of the class. It takes a constant reference to another Number object and returns a new Number object that represents the sum of the two number's.

=> In the main function we create two Number objects num1 and num2 with values 5 and 10 respectively we than use the + operator to add these two objects together and store the result.

2] Explain with example unary operator can be overloaded using friend function.

=>

Example program using friend function

```
#include <iostream.h>
#include<conio.h>
using namespace std;
class complex
{
  int real, imaginary;
public:
    complex()
    {
    }
    complex(int a, int b)
    {
    real = a;
    imaginary = b;
    }
friend void operator-(complex &c);
 void display()
  {
 cout<<"Real value"<<real<<endl;
 cout<<"Imaginarry value: "<<imaginary<<endl;
  }

};
void operator-(complex &c)
  {
    c.real = -c.real;
    c.imaginary = -c.imaginary;
  }
```

```
void main()
{
 complex c1(10,20);
 cout<<"Real and imaginary value before operation
      :";c1.display();
 cout<<"Real and imaginary value after
     operation:"<<endl;
     c1.display();
    getch();
}
```

Output

Real and imaginary value before operation:
Real value: 10
Imaginary value: 20
Real and imaginary value after operation:
Real value: -10
Imaginary: -20

In general you should define the member function to implement operator overloading friend function has been introduced for a different purpose that we are going to discuss in our.

Syntax

```
friend return_type operator-symbol
(variable 1, variable 2)
{
  //statements;
}
```

Date:

OR

3] Explain class to class type conversion with example.

=> Through class conversion one can assign data that belongs to a particular class type to an object that belongs to another class type.

Example:
Let there be two classes 'A' and 'B' If we want to allocate the details that belong to class 'A' to an object of class 'B' then this can be achieved by B (object of class B) = A (object of class A) where '=' has been overloaded for objects of class type 'B'.

Example program.

```
#include <iostream.h>
#include <conio.h>
class class_type ~~type~~ one
  {
  String a = "GeeksforGeeks";
public:
  string get_string()
  {
    return (a);
  }
```

```
void display()
 {
  cout << a << endl;
 }
};
class class_type_two {
 string b;
public
 void operator=(class_type_one a)
 {
  b= a.get_string();
 }
 void display()
 {
  cout << b << endl;
 }
};
void main()
{
 class_type_one a;
 class_type_two b;
 b = a;
 a.display();
 getch();
}
```

output

Greeks for Greeks

Greeks for Greeks

Q-4 [A] Answer the Following (5 mark) [

1] Explain this pointer.

⇒ The pointer in C++ language is a variable it is also known as locator or indicator that points to an address of a value. The symbol of an address is represented by a pointer. In addition to creating and modifing dynamic data structures they allow programs to emulate call-by-reference.

2] List types of Inheritance.

1] Single Inheritance
2] Multiple Inheritance
3] Multilevel Inheritance
4] Hierachical Inheritance
5] Hybrid Inheritance

Q4 [b] Answer the following (Any two) [12 mark]

1] Explain virtual base class with example.

Syntax for virtual base classes

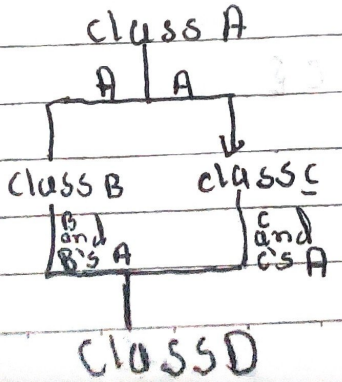
```
class B: virtual public A
{
};
```

Syntax 2:

```
class c: public virtual A
{
};
```

Vitual base classes are used in virtual inheritance in a way of preventing multipal "instances" of a given class appearing in an inheritance hierachy when using multipal inheritances.

Need for virtual Base classes: consider the situation where we have one class A. this class A is inherited by two other classes B and C. both these class are inherited into another in a new class D as shown in figure below.



As we can see from the figure that data members/Function of class A are inherited twice to class D. one through class B and second through class C when any data/Function member of class A is accessed by an object of class D. ambiguity arises as to which data/Function member would be called? one inherited through B or the other inherited through C. this confuses compiler and it displays error.

Example program

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class A{
public:
    void show()
    {
     cout<<"Hello From A\n";
    }
};
class B: public A{
};
class C: public A{
};
class D: public B, public C{
};
void main()
{
  D object;
  object.show();
```

2] Explain virtual function. Write a program for implementation of virtual function.

⇒ A virtual function (also known as virtual methods) is a member function that is declared within a base class and is re-defined (or overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the method.

- virtual functions ensure that the correct function is called for an object regardless of the type of reference (or pointer) used for the function call.
- They are mainly used to achieve Runtime polymorphism.
- Functions are declared with a virtual keyword in base class.
- The resolving of a function call is done at runtime.

★ Rules for virtual functions

1] Virtual functions cannot be static
2] A virtual function can be a friend function of another class.

3] virtual function should be accessed using a pointer or reference of base class type to achieve runtime polymorphism.

4] The prototype of virtual function should be the same in the base as well as the derived class.

5] They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the ~~version~~ virtual function) In that case the base class version of the function is used.

6] A class may have a virtual destructor but it cannot have a virtual constructor.

Example program

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class base{
public:
    virtual void print(){
cout<<"print base class\n";
}
```

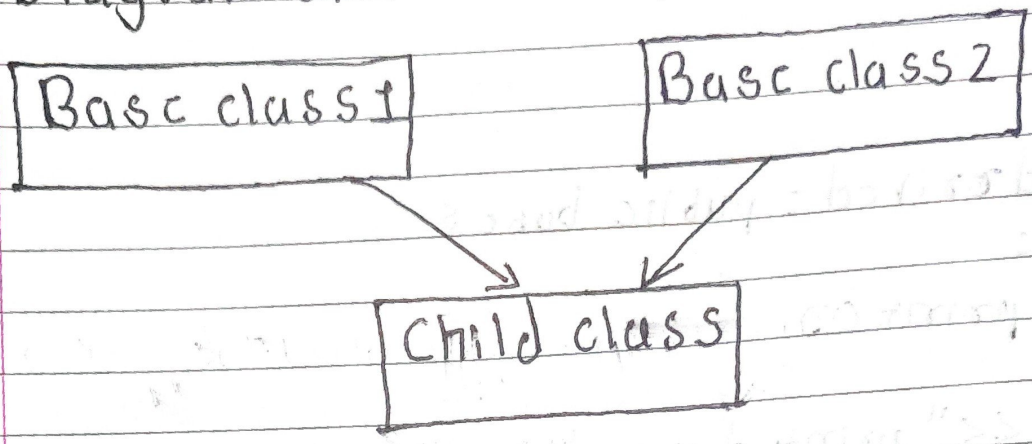
```
void show()
{
cout<<" show base class\n";
}

};
class derived : public base
public:
  void print()
  {
  cout<<" print derived class\n";
  }
  void show()
  {
  cout<<" show derived class\n";
  }
};
 void main()
 {
  base* bptr;
  derived d;
  bptr = &d;
  bptr = &d;
  bptr->print();
  bptr->show();
  getch()
} output
print derived class
show base class
```

OR

3] Explain multiple inheritance with example.

⇒ Diagram of the multiple Inheritance



Multiple Inheritance is the concept of the Inheritance in C++ that allows a child class to inherit properties of behaviour from multiple base classes. Therefore we can say it is the process that enables a derived class to acquire member function, properties characteristics from more than one base class.

In the above diagram there are two-parent classes: Base class 1 and Base class 2 whereas there is only child class. The child class acquires all features from both Base class 1 and base class 2 therefore we termed the type of inheritance as multiple Inheritance.

Syntax

```
class A
{
// code of class A
};
class B
{
// code of class B
};
class C: public A, public B (access modifier class_name)
{
// code of the derived class
};
```

Example program.

```
#include <iostream.h>
#include <conio.h>
using namespace std;

class Base_class
{
 public:

void display()
{
 cout<<" It is the first function of the base class
      << endl;
}
};
Class Base_class2
```

```
{
public:
void display2()
{
 cout<<" It is the second Function of the base
        class" <<endl;
 }
};

class child_class public Base-class public
Base-class 2
{
 public:
void display3()
  {
  cout<<" It is the Function of the derived class
      " <<endl;
  }
};
Void main()
child-class ch;
ch.display();
ch.display2();
ch.display3();
getch();
}
```

output

```
It is the first Function of the Base class
It is the second Function of the Base class
It is the Function of the derived class
```