MAY-2014

Roll No _____

Examination May-2014

Total no of page-1

BTCS-305

Object oriented Programming using C++

Paper ID-A1129

Time 3 hrs

Max Marks 60

Note : Attempt four question from section B and two from section C. Section A is compulsory

Section A 2x10=20

Q1. 1. What is the use of constructors
2. What is the use explicit constructor
3. What is a virtual class.
4. What is a friend functions
5. What is meant by initialization of a variable?
6. What is polymorphism
7. What is advantage of scope resolution operator
8. What is a ambiguity in inheritance
9. Distinguish between static members & variables? How are they useful?
10. What are virtual constructors? Give relevant examples to explain

Section B 4x5=20

Q 2. How are Constructors & destructors invoked in derived classes? What actually happens when a destructor is invoked?

Q 3. What do you understand by inheritance? Give its various types and access mechanisms.

Q 4 Explain how base class member functions can be invoked in a derived class if the derived class also has a member function with the same name.

Q 5 Write a program to accept a line & count the number of words in c++

Q 6. Write a program to accept the record of 5 persons with their names, age & addresses & also display them. Use structures to implement the program.

Section C 2x10=20

Q 7. What is the use of default & copy constructors. Is a constructor and distructor mandatory for a Class. Explain by giving examples in each case.

Q 8. Write a program to overload the +, -, X, % operator to find the addition, subtraction, multiplication and division of Complex numbers

Q 9. Write a program to copy the contents of one file into another using command line arguments. How are binary files different from text files in CPP?

End

BTCS-302-18

OOPs May-2014

Section-A

Q-1 1. What is the use of constructors?

→ A constructor is a special method of a class or structure in object-oriented programming that initialize an object of that type. A constructor is an instance method that usually has the same name as class, and can be used to set the values of the members of an object.

Q-2 What is the use of Explicit constructor?

→ The explicit function specifier controls unwanted implicit type conversion. It can only be used in declarations of constructors within a class declaration.

Q-3 What is virtual class?

→ A virtual class is a nested inner class whose functions and member variables can be overridden and redefined by subclasses of an outer class.

Q-4 What is friend function?

→ A friend function is defined outside the class but it has the right to access all private and protected members of the class

Spiral

Q.5 What is meant by initialization of a variable?

→ Initialization is the process of locating and using the defined values for variable data that is used by a computer program.

Q.6 What is polymorphism?

→ The word polymorphism means having many forms. It means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Q.7 What is advantages of Scope resolution operator?

→ Advantages are:

1) To access the global variable when there is a local variable when there is a local variable with same name.
2) C++ program to show that we can access a global variable.

Q.8 What is ambiguity in inheritance.

→ A situation in which something has more than one possible meaning and may therefore cause confusion. Like in multiple inheritance, a single class is derived from two or more parent classes.

Spiral

Distinguish between static member & variable? How are they useful?

→ The main difference between static and normal variable is that static member is initialized only once in program and retains it's value, unlike the normal value that gets initialized every time.

A variable declared static within the body of a function maintains its value between invocations of function. A variable declared static within a module is accessible by all functions

Q.10 What is virtual constructors? Give relevant example to explain.

→ Declaring something virtual in C++ means that it can be overridden by a sub-class of the current class, however the constructor is called when the object is created a sub class of the class you must be creating the class so there would never be any need to declare constructor.

Spiral

Section - B

How are constructors and destructors invoked in derived class? What actually happens when a destructor is invoked?

→ The constructors are used to initialize member variable of the object, and the destructor is used to destroy the object.

The compiler automatically invokes constructor and destructor.
The derived class constructor passes arguments to the base class constructor.

Eg →
```
# include <iostream>
using namespace std;
class parent
{
  public:
  Parent (int i)
  { int x = i;
    cout << "Inside base class's parameterised constructor" << endl;
  }
};
class Child : public Parent
{
  public:
  child (int j) : Parent (j)
  {
    Cout << "Inside sub class parametrized constructo" << endl;
  }
};
```

Spiral

```
int main() {
    Child obj1(10);
    return (0);
}
```

A destructor is a member function which destruct or deletes an object.

A destructor function is called automatically when the object goes out of scope.

Destructors have same name as the class preceded by a tilde (~)

Q3) What do you understand by inheritance? Give its various types and access mechanism.

➔ Inheritance in C++

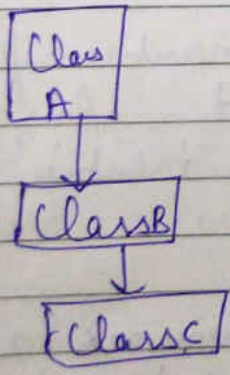
The capability of a class to derive properties and characteristics from another class is called inheritance.
Inheritance is one of the most important feature of Object Oriented Programming.

Sub class → The class that inherits properties from another class is called Sub class or Derived class.

Super class → The class whose properties are inherited by sub class is called Base class or Super class.

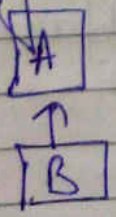
Syntax →

```
Class Subclass_name : access_mode base-class
{
    // body of subclass
};
```

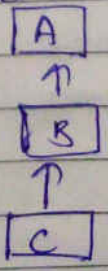


TYPES : →

(i) Single level

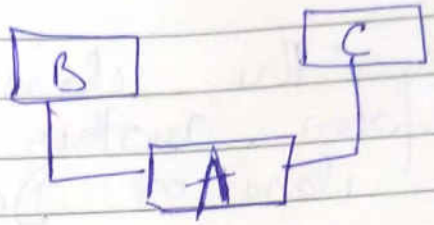


(ii) Multi level

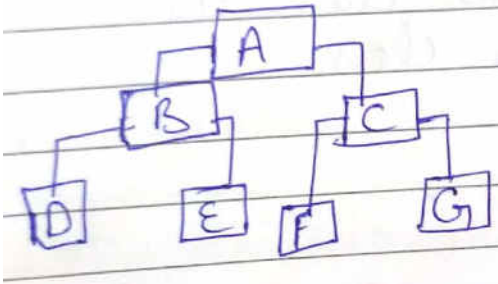


Spiral

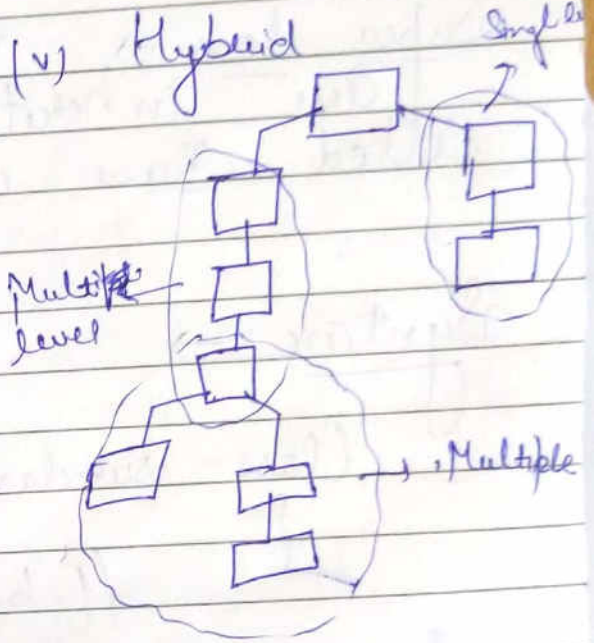
(iii) Multiple →



(iv) Hierarical



(v) Hybrid



Q4) Explain how base class member functions can be invoked in a derived class if the derived class also has a member function with same name.

→ This problem is also known as ambiguity in multiple inheritance

In multiple inheritance, a single class is derived from two or more parent have same named member function

```
#include <iostream>
using namespace std;
class A
{   public:
    void fun();
};
void A::fun()
{ cout << "fun() called";
}
int main()
{   A a;
    a.fun()
    return 0;
}
```

Q5) Write a programe to accept a line and cout the number of words in C++.

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    char str[50];
int count = 0, i;
cout << "Enter a string:";
gets(str);
}
```

Spiral

```
for (i=0; str[i]!='\0'; i++)
{
  if str[i]==' '
  count++;
}
cout << "Number of words in the strings are " << count+1;
return 0;
}
```

Q5) Write a program to accept the record of 5 persons with their names, age and addresses and also display them. Use structures to implement the program.

```
#include <stdio.h>
Struct student
{
    char name[50];
    int roll;
    float marks;
};
int main()
{
cout<<"printf("Enter information:\n");
```

```
cout << " Enter name : ";
fgets(s.name, 50, stdin);

cout << " Enter Roll Number : ";
cin >> s.rollNo;

cout << " enter marks ;
cin >> s.marks;

cout << " Displaying Information ";
cout << " Name : ";
puts(s.name);

cout << " Roll No. : " << s.roll;

Cout << " Marks : " << s.marks;
return 0;
}
```

Section-C

What is use of default & Copy Constructors. Is a constructor and distructor mandatory for a class. Explain by giving examples in each case

A⇒ **Default Constructor:**

A default constructor is used to initialize data members of an object with a legal initial value. It doesn't carry any parameter in its prototype

Copy Constructor: A copy constructor is used to copy the value of one object to another data member by member. It carries an object parameter in its prototype.

Example:

```
class abc
{
  int a;
  int b;
  abc()    // default Constructor
  {
  a=0;
  b=0;
  }
  abc(abc obj) // Copy Constructor
  {
  this.a = obj.a;
  this.b = obj.b;
  }
```

Spiral

Write a program for overloading operator for multiplying

```
void main()
{
abc ob = new ob();
System.out.println(ob.a);
System.out.println(ob.b);
ob.a = 10;
ob.b = 20;
abc ob2 = new abc();
System.out.println(ob2.a);
System.out.println(ob2.b);
ob2 = ob;
System.out.println(ob2.a);
System.out.println(ob2.b);
}}
```

Write a program to overload the +, -, X, % operator to find the addition, Subtraction, multiplication and division of complex numbers.

Sol

```
#include <iostream>
using namespace std;
class complex {
  Private:
    complex (int r=0, int i=0) {real=r;
      imag=r;}

    complex Operator+(complex const &obj)
{
    complex rea;
   res. real = real + Obj. real;
   res. imag = imag + Obj. imag;
   return res;
}
    complex res;
   res. real = real - Obj. real;
   res. imag = imag - Obj. Imag;
    return res;
}
   complex operator x (complex (const &obj)
   {
     Complex res;
     res. real = real x Obj. real;
     res. imag = imag x Obj. imag;
     return res;
   }
   complex operator %. (complex const &obj)
```

Spiral

```
{
complex res;
res.real = real + obj.real;
res.imag = imag + obj.imag;
return res;
}
void print() { cout << real << "+i"
<<imag << endl; }
};
int main()
{
complex C1(10,5), C2(2,4);
complex C3 = C1+C2
C3.print();
complex C4 = C1-C2
C4.Print()
complex C5 = C1*C2
C5.print();
complex C6 = C1/C2
C6 print();
}
```

Write a program to copy the contents of one file into another using command line arguments. How are binary files different from text files in CPP?

Sol

```
# include <iostream>
using std :: cout;
using std :: end;
using std :: ios;

# include <fstream>
using std :: ifstream
using std :: of stream

int main ( int argc , char * argv[])
{
  if (argc ! = 3)
    cout << " usage : copytile infile_name
        outfile_name" << end1;
  else
  {
    ifstream infile ( argv[1], ios :: in);
    if ( ! infile)
    {
      cout << argv [1] << " could not be
      opened" << end1;
      return-1;
    }
    ofstream outfile ( argv[2], ios :: out);
    if ( ! out File)
    {
      cout << argv [2] << " could not be
      opened" << end1;
```

Spiral

```
return -2;
}
    char c = inFile.get();

    while (infile)
    {
      outfile.put(c);
      c = infile.get();
    }
}
return 0;
}
```

Dec-2014

Roll No. | | | | | | | | | | | |

Total No. of Pages: 02

Total No. of Questions: 09

B.TECH(EE, 3D ANIMATION & GRAPHICS, CSE, ECE, ETE, IT) (Sem.-3rd)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code: BTCS-305

Paper ID: [A1129]

Time: 3 Hrs.

Max. Marks: 60

INSTRUCTIONS TO CANDIDATE:

(i)Section –A, is Compulsory.

(ii)Attempt any four questions from Section-B.

(iii)Attempt any two questions from Section-C.

Section –A

Q.1.(a) What is scope resolutions operator? (2)

(b) What is default coustructor? (2)

(c) What is the use of virtual keyword? (2)

(d) What are abstract classes? (2)

(e) What are nested classes? (2)

(f) What are manipulators? (2)

(g) What is the difference between calling method of constructor and destructor? (2)

(h) What is the need of friend functions? (2)

(i) When do we use reference operator with objects? (2)

(j) What is late binding? (2)

Section –B

Q.2. What are the different ways in which we can have abstraction in Objected Oriented Progamming? Explain each with the help of an example. (5)

Q.3. Explain the constructors with default arguments. How constructors can be called explicitly? (5)

Q.4. What do you mean by an array of objects? Explain how members of objects can be accessed in array of objects with the help of C++ program (5)

Q.5. Why principle of substitution cause slicing problem? What are the different ways to avoid it? (5)

Q.6. What are the advantages of using Template functions? How are they different from the macros? (5)

Section –C

Q.7. What is the difference between overloading and overriding of a function? Write a program in C++ to overload= = operator and campare two objects using the operator. (10)

Q.8. What are the different ways to achieve the polymorphism in C++? Explain the pure polymorphism with example. (10)

Q.9. What is the use of Templates in C++? Explain the use of Standard temple library with the help of a C++ Program. Also Explain how exceptions are caught in C++ (10)

---:END:---

OOPS

December 2014.

Object oriented programming

BTCS-302-18

Section - A

Q1 ⇒ (a) Scope resolution operator ⇒

In C++, scope resolution operator is :: . It is used for following purposes ⇒

(1) To define member functions outside the class.

(2) To remove ambiguity problem.

(b) Default Constructor ⇒

Default Constructor is a constructor which does not take any argument. It has no parameters.

(c) Virtual function ⇒

Uses of virtual function :-

(1) It is used to tell the compiler to perform dynamic linkage on the function.

(2) When base class pointer contains the address of the derived class object, always executes the base class function. This issue can be resolved by using "virtual function".

(d) Abstract Class ⇒

A abstract class is a class that cannot be instantiated and usually implemented as a class that has one or more pure virtual function.

(e) Nested Class ⇒

A nested class a class that is declared in another class. A nested class is also a member variable of enclosing class and has the same access rights as the other members.

(g) Calling of Constructor and Destructor.

For Constructor =>

Constructor is called automatically while the object is created.

For Destructor =>

Destructor is called automatically, as block is excited or program terminates.

(h) Manipulators =>

Manipulators are operators used in C++ for formatting output. The data is manipulated by the programmer's choice of display.

Examples => endl, setfill.

(h) Friend function =>

Need of friend function =>

(1) Private data can be accessed from this function.

(2) To break concept of data hiding.

(i) Reference Operator =>

We use reference operator with object, when we have to store the address of a variable.

(j) Late Binding =>

This is run time polymorphism. In this type the compiler adds code that identifies the object type at running than matches the call with the right function definition.

Set Day :- B.

(Q) Data abstraction is a property by virtue of which only essential details are displayed to the user.

Data abstraction may also be defined as the process of identifying only the required characteristics of object ignoring the irrelevant details.

There are two way in which we have abstraction in oops :-

(a) Abstract class :-
* A abstract class is class that is declared with abstract keyword.
* An abstracted class may or may not have abstract method.
* There can be no object of an abstract class
* An abstract class can have parameterized constructors and default constructor.

(b) Abstract method or function :-
* An abstract method is method that is declared without an implementation.
* A method defined abstract must always be redefined in the subclass.
* Making compulsory or Either make subclass itself abstract.

Program for example :-

```
# include <iostream>
```

```
class base
{   int x ;
public:
      virtual void fun() = 0;
       int getX()
      { return x ; }
};
class derived : public base
{
    int y;
public:
    Void fun()
    {  cout << "fun() called";
    }
};
int main(void)
{
    derived d;
    d.fun ;
}
```

3. Default Constructor :- It is constructor that either has no parameters, or if it has parameters, all the parameters are default value.

Examples :-

```
#include <iostream>
class construct
{  public:
      int a, b ;
      Construct()
      {      a = 10;
};    }      b = 20;
```

```
int main()
{   Construct c;
    Cout << "a: " << c.a << endl << "b:" << c.b;
}.
```

Explicit call to Constructor:-

```
# include <iostream>
  Class Test.
  {
    public:
     Test()
    { Cout << " Constructor is executed;
    }
  };
  int main()
  { Test();          // Explicit call to constructor.
    Test t;          // local object.
  }.
```

(4) Array of the object :- An array of objects, all of whose elements are of the same class, can be declared just as an array of any built in type.

* The array of type class contains the object of class as its individual elements.
* An array of class type is known as array of object.

Syntax :- class_name array_name [size];

Program :-

```
# include <iostream>
class Myclass
{ int x;
  public:
   void setx (int i)
  { x = i; }
    int getx
  { return x;
};
void main ()
{
   Myclass obj [4];      // Array of object.
    int i;
   for (i=0; i<4; i++)
     obj [i]. setx (i)   // call function
                         // with array object
   for (i=0; i<4; i++)
     cout << "obj [" << i << "]. getx():"
          << obj [i]. getx () << "\n";
}
```

(5) Slicing :- It means that data added by a subclass are discarded when object of subclass is passed or returned by value or from a function expecting a base class object.

Program :-

```
class base
{ int x, y; };
class derived:
```

```
public base
{ int z,w; };
int main()
{
    derived d;
    base b = d;          // object slicing.
}.
```

Object slicing :⇒ It happens when derived class object is assigned to a base class object, additional attributes of derived class object, are sliced off to form base class object.

How to avoid it :⇒ We can avoid unexpected behaviour with the use of pointers.

Program :⇒

```
Void someFunc ( base * objp)
{   objp → display();
}
int main()
{   base *bp = new base(33);
    derived *dp = new derived(45,54);
    Some func(bp);
    Some func(dp);   // No Object slicing
}
```

(6) Templates :⇒ A template is a simple and yet very powerful tool in C++.

Overloading

Advantages of Templates:-

* Reduce the repetition of code.
* Static - polymorphism (= performance) and other compile time calculations are good.
* Increase safety at no cost.
* Policy based design (flexibility, Easier change etc) are very Easy.
* Concept checks Easily.

Differ ~~m~~ template from macros :-

There are major difference between a template and a macro. A macro is merely a string that the compiler replaces with the value that was defined

Eg.

```
#define STRING_TO_BE_REPLACED.
"value to replace WITH"
```

A template is way to make functions ~~is~~ independent of data types. This can't be accomplished using Macros

Eg.

A sorting function doesn't have to care whether it's sorting integers or letters since the same algorithm might apply anyway.

SECTION-C

Overloading	Overriding
1. More than one method with same name, different prototype in same scope is called overloading	1. More than one method with same name, same prototype on different scope is called as overriding
2. In case of ~~method~~ overloading, parameter must be different	2. In case of overriding, parameter must be same
3. Overloading is the example of Compile time polymorphism	3. Method overriding is the example of Run time polymorphism
4. Method overloading is performed within class	4. Method overriding occurs in two classes
5. In case of method overloading, return type can be same or different	5. In case of method overriding return type must be same.
6. Static binding is being used for overloading methods.	6. Dynamic binding is being used for overriding.

Polymorphism

```
#include<iostream>
using namespace std;
class Circle
{
private:
  float radius;
public:
  Circle(float r=0):radius(r) { }
void ChangeRadius(int radius) { this->radius=radius; }
float getRadius() { return radius; }
float getArea() const { return 3.14 *radius *radius; }
bool operator==(Circle a);
};
inline bool Circle::operator==(Circle a)
{
return this->radius==a.radius; }
int main()
{
Circle A(10), B(20), C(10);
Cout<<"Circle A: Radius="<<A.getRadius()<<"units, Area="
    <<A.getArea()<<"Sq.units\n";
Cout<<"Circle B: Radius="<<B.getRadius()<<"units, Area="
    <<B.getdata()<<"Sq.units\n";
Cout<<"Circle C: Radius="<<C.getRadius()<<"units, Area="
    <<C.getdata()<<"Sq.units\n";
Cout<<"A==B:";
if(A==B)
  Cout<<"Circles are equal.\n";
else
  Cout<<"Circles are not equal.\n";
Cout<<"A==C:";
if(A==C)
  Cout<<"Circles are equal.\n";
```

cout << "Circles are not equal. \n";
}

Charlie
Date
Page No.

Polymorphism → Polymorphism is the ability of a message to be displayed in more than one form

Polymorphism is divided into two types:-

- Compile time Polymorphism
- Run time Polymorphism

1. Compile time Polymorphism → This type of Polymorphism is achieved by function overloading & operator overloading

- Function overloading → When there are multiple function with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and changed in type of arguments.

```
#include <iostream>
using namespace std;
class
{
public:
void function(int x)
{
cout << "Value of x is " << x << endl;
}
void function (double x)
{
cout << "value of x is " << x << endl;
}
```

```
void function (int x, int y)
{
    cout << " value of x and y is " << x << ", " << y << endl;
}
};
int main()
{
  function obj1;
  obj1.func(7);
  obj1.func(9.132);
  obj1.func(85,64);
  return 0;
}
```

Operator Overloading →

```
#include <iostream>
using namespace std;
class Complex
{
  private:
    Complex(int r=0, int i=0)
{
  real = r; imag = i;
}
Complex operator + (Complex const &obj)
{
  Complex res;
  res.real = real + obj.real;
  res.imag = imag + obj.imag;
  return res;
```

```
    ss.real = real + obj.real;
    ss.imag = imag + obj.imag;
    return ss;
}
void print()
{
cout<<real<<" + i " <<imag<<endl;
}
};
int main()
{
   Complex c1(10,5), c2(2,4);
   Complex c3 = c1+c2
    c3.print();
}
```

2. Runtime polymorphism :- This type of polymorphism is achieved by function overriding

- Function Overriding → On the other hand occurs when a derived class has a definition for one of the member function of the base class. This base is said to be overridden

```
#include<iostream>
using namespace std;
    class base
{
public:
   virtual void print()
```

```
{
cout << " print base class" << endl;
}
void show()
{
 cout << " show base class" << endl;
}
};
 int main()
{
   base * bptr;
    derived d;
    bptr = &d;
 bptr -> print();
bptr -> show();
  return 0;
}
```

Pure Polymorphism → Pure polymorphism occurs when a single function can be applied to arguments of a variety of types. In pure polymorphism, there is one function code body and a no. of interpretations. The other extreme occurs when we have a no. of different functions.

Charlie
Date
Page No.

Templates is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. eg. a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

C++ adds two new keywords to support templates: 'template' and 'typename'. The second keyword can always replaced by keyword class.

Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contain only function/class, but compiled code may contain multiple copies of same function/class.

Standard Template Library → The Standard Template Library is a set of C++ template classes to provide common programming data structures and functions such that a lists, stacks, arrays, etc. It is a library of container classes, algorithms and iterators. It is a generalized library and so, its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL

```
#include<iostream>
using #include<queue>
using namespace std;
int main()
{
  queue<int>q;
  int n, item;
  Cout<<"enter integers to EnQueue and 0 to
                    Stop EnQuing"<<endl;
  Cin>>n;
  while (n)
{
  q.push(n);
    Cin>>n
}
Cout<<"The Size of the queue is!"<<q.Size()<<endl;
Cout<<"The first element that entered the queue is!"<<
                                  q.front()<<endl;
Cout<<"The last element that entered the queue is!"<<q.back
                                  ()<<endl;
Cout<<"DeQueuing ....."<<endl;
  while (!q.empty())
{
  item=q.front();
  Cout<<item<<" ";
    q.pop();
}
Cout<<"\n executed successfully \n";
  return 0;
}
```

May-2015

Roll No. | | | | | | | | | | |

Total No. of Questions : 09

Total No. of Pages : 02

B.Tech.(ECE)/(EE)/(EEE)/(EIE)(Sem.–3)

B.Tech.(IT) (2007 to 2010 Batch)/(CSE) (2007 Onwards Batch)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code : CS-252

Paper ID : [A0304]

Time : 3 Hrs.

Max. Marks : 60

INSTRUCTION TO CANDIDATES :

1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students has to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students has to attempt any TWO questions.

SECTION - A

1. **Write briefly:**

(a) Is cout a function or object? Justify your answer.

(b) "*An unsigned int can occupy double space as compared to signed int.*" Explain how?

(c) What is the importance of scope resolution operator in C++?

(d) What is the value of x in expression x= (a=10, a*a)?

(e) Differentiate static and dynamic binding.

(f) Write the main characteristics of destructors.

(g) Can a function be defined inside the main () function?

(h) How memory is allocated dynamically in C++?

(i) What will be the output of the code?

```
int main()
{ int num[]={ 1,2,3,4,5};
Cout<<num[3-2+3];}
```

(j) Define generic function with suitable example.

SECTION - B

2. Write a program in C++ using class Array that prints the elements stored in the array data member in reverse order. (5)

3. Define a constructor. Can we overload constructors? Justify your answer with example. (5)

4. Why do we need virtual functions? When do we make a virtual function "Pure"? (3,2)

5. Describe the various classes available for file operations. (5)

6. Write a class ACCOUNT that represents your bank account and then use it. The class should allow you to deposit money, withdraw money, calculate interest, and send you a nasty message if money is not sufficient in the account. Use constructor to provide initial amount in the account. (5)

SECTION-C

7. Create a class Shape which has two overloaded member functions: Area () & Perimeter(). Depending upon whether the dimensions are input as integers or floating-point numbers, calculate the area and perimeter of three different shapes. The dimensions of the shape would be entered by the user. The output should be in the format as the input. (10)

8. a) What is Inheritance? Explain different types of Inheritance. (4)

   b) Describe all access specifiers used in inheritance. Also depict the effect of inheritanceonthe visibility of members in tabular form. (6)

9. a) Differentiate function overloading with function overriding. (5)

   b) What are the merits and demerits of using friend function? (5)

Section-A

May-2015

(a) Is cout a function or object? Justify your answer.

→ The cout object in C++ is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard c output stream stdout.

It is defined in <iostream> header file.

The cout object is ensured to be initialized during or before the first time an object of type ios_base::Init is constructed.

After the cout object is constructed, it is tied to cin which means that any input operation on cin executes cout.flush().

The "c" in cout refers to "character" and 'out' means "output", hence cout means "character output". The cout object is used along with the insertion operator (<<) in order to display a stream of characters.

(b) "An unsigned int can occupy double space as compared to signed int." Explain how?

→ Let's see what is meant by signed and unsigned variables.

Signed and unsigned variables tell us whether it can take positive values, negative values or both. Signed keyword is used for those variables which can take positive as well as negative values. Unsigned keyword is used for those variables which can take only values which are zero or positive i.e. without - (negative sign).

We can use signed and unsigned keywords with only

'int' and 'char' data types.

For example normally an integer variable of size 4 bytes can take values from -2,147,483,648 to 2,147,483,647, whereas if we declare 'x' as

unsigned int x;

then 'x' can take values from 0 to 4,294,967,295 (since 'x' is now as unsigned variable).

(c) What is the importance of scope resolution operator in C++?

The :: (scope resolution) operator is used to get hidden names due to variables scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. You can use the unary scope operator if a namespace scope or global scope name is hidden by a particular declaration of an equivalent name during a block or class. For example, if you have a global variable of name my_var and a local variable of name my_var, to access global my_var, you'll need to use the scope resolution operator.

For example,

```
#include <iostream>
using namespace std;
int my_var = 0;
int main (void) {
int my_var = 0;
:: my_var = 1; // set global my_var to 1
```

```
    my_var = 2; // set local my_var to 2
    cout << ::my_var << " , " << my_var;
    return 0;
}
```

This will give the output:

1, 2.

The declaration of my_var declared in the main function hides the integer named my_var declared in global namespace scope. The statement ::my_var = 1 accesses the variable named my_var declared in global namespace scope.

You can also use the scope resolution operator to use class names or class member names. If a class member name is hidden, you can use it by prefixing it with its class name and the class scope operator. For example,

```
#include <iostream>
using namespace std;
class X {
public:
static int count;
};
int X::count = 10; // define static data member
int main () {
  int X = 0; // hides class type X
  cout << X::count << endl; // use static member of
                               class X.
}
```

This will give output:

10

(D) What is the value of x in expression x = (a = 10, a * a)?

→
```
#include <iostream>
using namespace std;
int main()
{
  int a;
  cout << (a=10, a*a);
  return 0;
}
```

Output: 100

(E) Differentiate between static & dynamic binding.

→

Static Binding	Dynamic Binding
(i) Events occur at compile time are "Static Binding".	(i) Events occur at run time are "Dynamic Binding".
(ii) All information needed to call a function is known at compile time.	(ii) All information need to call a fn come to know at run time.
(iii) Efficiency.	(iii) Flexibility
(iv) Fast execution.	(iv) Slow execution
(v) Early Binding	(v) Late binding.
(vi) Overloaded function call, overloaded operators.	(vi) Virtual function in C++, overridden methods in Java.

(f) Write the main characteristics of destructors.

→ (i) Destructors have the same name as

that of the class they belong to preceded by a ~ (tilde).

(ii) Similar to constructors, the destructors do not have return type and not even void.

(iii) Constructors and destructors cannot be inherited, though a derived class can call the constructors and destructors of the base class.

(iv) Destructors can be virtual, but constructors cannot.

(v) Only one destructor can be defined in the destructor. The destructor does not have any arguments.

(vi) Destructors neither have default values nor can be overloaded.

(vii) Programmer cannot access addresses of constructors and destructors.

(viii) TURBO C++ compiler can define constructors and destructors if they have not been explicitly defined.

(ix) Constructors and destructors can make implicit calls to operators new and delete if memory allocation / de-allocation is needed for an object.

(x) An object with a constructor or destructor cannot be used as a member of a union.

(G) Can a function be defined inside the main() function ?

→ Not directly, but yes the static functions inside local classes.

C++ doesn't support that directly.

e.g:-
```
int main
{
    struct X {
    static void a ()
    {
      }
};
  X::a (a);
return 0;
}
```

(H) How memory is allocated dynamically in C++ ?

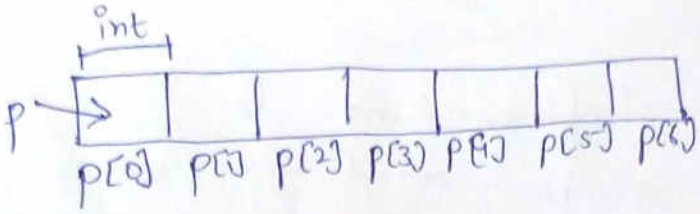
→ The new operator denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Syntax to use new operator :- To allocate memory of any data type, the syntax is:

pointer-variable = new data-type;

Example :
```
int *p = new int [10]
```

dynamically allocates memory for 10 integers continously of type int and returns pointer to the first element of the sequence, which is assigned to p (a pointer). p[0] refers to first element, p[1] refers to second element and so on.



(I) What will be the output of the code?

```
int main()
{ int num [] = {1,2,3,4,5};
cout << num [3-2+3]; }
```

→ Output : 5

(J) Define generic function with suitable example.

→ Generics is the idea to allow type (integer, string, ... etc and user-defined types) to be a parameter to methods, classes and interfaces.

The method of Generic Programming is implemented to increase the efficiency of the code. Generic programming enables the programmer to write a general algorithm which will work with all data types. It eliminates the need to create different algorithms if the data type is an

integer, string or character.

The advantages of generic Programming are:

(i) Code Reusability

(ii) Avoid function overloading

(iii) Once written it can be used for multiple times and cases.

Generics can be implemented in C++ using Templates. Templates is a simple and yet very powerful tool in C++. Example:

```
#include <iostream>
using namespace std;
template <typename T>
T myMax (Tx, Ty)
{ return (x > y) ? x : y;
}
int main ()
{
cout << myMax <int> (3, 7) << endl;
cout << myMax < double > (3.0, 7.0) << endl;
cout << myMax <char> ('g', 'e') << endl;
return 0;
}
```

Output:

```
7
7
g
```

Section-B

Question 2 → Write a program in C++ using class Array that prints the elements stored in the array data member in reverse order?

Answer →

```
// Iterative C++ program to reverse an array
#include <bits/stdc++.h>
using namespace std;

/* Function to reverse arr[] from start to end*/
void reverseArray (int arr[], int start, int end)
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

/* utility function to print an array */
void printArray (int arr[], int size)
{
    for (int i = 0; i < size; i++)
    cout << arr[i] << " ";
    cout << end;
}

/* Driver function to test above functions */
int main()
```

```
{
    int arr[] = {1,2,3,4,5,6};
    int n = sizeof(arr)/sizeof(arr[0]);
      // To print original array
       PrintArrey(arr, n);

      // Function calling
       reverseArray(arr, 0, n-1);
     Cout << " Reversed array is" << end;
      // To print the Reversed array
       PrintArray(arr, n);
     return 0;
}
```

Output →

1 2 3 4 5 6

Reversed array is

6 5 4 3 2 1

Qu → Define a constructor. Can we overloaded constructor.

3) A constructor is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values at time of object creation. It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

```
    cout << " Enter length and breadth respectively ";
    cin >> length >> breadth;
  }
  int Area Calculation () { return length * breadth; }
  void DisplayArea (int temp)
  {
    cout << " Area : " << temp << endl ;
  }
};
int main()
{
  Area A1 , A2 (2,1);
  int temp;
  cout << " Defult Area when no argument is passed " <<en
  temp = A1. Area Calculation ();
  A1. Display Area (temp);

  cout << " Area when (2,1) is passed as argument? <<en
  temp = A2. Area Calculation ();
  A2. Display Area (temp);

  return 0;
  }
```

output ->

```
Default Area when no argument is passed.
Area : 10
Area when (2,1): is passed as argument
Area: 2
```

- numeric data types are set to 0.
- char data types are set to null character ("\0")
- reference variables are set to null.
- Overloaded constructors have the same name but different number of arguments.
- Depending upon the number and type of argument passed, specific constructor is called.
- Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

Justification by example,

```
# include <iostream>
using namespace std;
Class Area
{
    Private :
        int length;
        int breadth;
    Public :
        // constructor with no arguments
        Area () : length (5), breadth (2) { }

        // constructor with two arguments.
        Area (int l, int b) : length (l), breadth (b) {}
    Void GetLength ()
    {
```

Question→ Why do we need virtual functions? When we make a virtual function "Pure"?

Answer→ A virtual function a member function which is declared within a base class and is re-defined by a derived class. When you refer to a class object and using pointer or a reference to the base class, you call a virtual function for that object and execute the derived class's version of the function.

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

Rules for Virtual Function→

1) Virtual functions cannot be static and also cannot be a friend function of another class.
2) Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.
3) The prototype of virtual functions should be same in base as well as derived class.
4) A class may have virtual destructor but it cannot have a virtual constructor.

Pure→

A virtual function will become pure virtual function when you append "=0" at the end of declaration of virtual function. Pure virtual function doesn't have body or Implement. We must implement all pure virtual functions in derived class.

Pure virtual function is also known as abstract function.

A class with at least one pure virtual function or abstract is called abstract class. We can't create an object of abstract class. Member functions of abstract class will be invoked by derived class object.

5Que→ Describe the various classes available for file.?

5Answer→ [In C++ file handling provides a mechanism to store] In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.

ofstream→ stream class to write on file.
ifstream→ stream class to read from files.
fstream→ stream class to both reed and write from /to files.

File Handling→

Files are a means to store data in a storage device. C++ file handling provides a mechanism to store output of program in a file and read from a file on the disk. So far, we have been using <iostream> header file which provide functions Cin and Cout to take console and write output to a console

and write output to a console respectively. Now we introduce one more header file <fstream> which provides data types or classes (ifstream, ofstream, fstream) to read from file and write to a file.

File opening modes →

A file can be opened in different modes to perform read and write operations. function to open a file i.e open() takes two arguments : char* filename and ios::mode. C++ supports the following file open modes →

Mode	Explanations
ios::in	open a file for reading.
ios::out	open a file for writing.
ios::app	Appends data to end the file.
ios::binary	file pointer moves to the end of the file but allows to writes data in any location in the file.
ios::binary	Binary file.
ios::trunc	Deletes the contents of the file before opening.

If a file is opened in ios::out mode, then by default it is opened in ios::trunc mode also i.e the contents of the opened file is overwritten. If we open a file using ifstream class, then by default it is opened in ios::in mode and if we open a file using ofstream class, then by default it is opened in ios::out mode. The fstream class doesn't provide any default mode.

6 Que → write a class ACCOUNT that represent your bank account and then use it. The class should allow money, withdraw money, calculate intrest and send you a msg message if money is not sufficient. in the account).

6 Answer →

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

class Account {

  Private :                          // data member.

     char * name;
     char * address;
     char * city;
     char * pcode;

     float acc_bal;

  Public:

     Account :: Account () {
     name = NULL;
     address = NULL;
     Pcode = NULL;
```

```
    acc_bal = 0.00 ;
}
void addcust () {                    // add account method
    cout << "\n Enter your name:" ;  cin >> name ;
    cout << "\n Enter your address; cin >> address;
    cout << "\nEnter your city ; cin >> city ;
    cout << "\n Enter yays postal code"; cin >> pcode ;
    cout << "\n Enter current account balance:" cin >>
    acc_bal;
}
void deposit (float bal) {
    float dep ;
    cout << "\nEnter amount to be deposited: ";
                                        cin >> dep
    bal = bal + dep ;
    acc_bal = bal;
}
void withdraw (float bal ) {
    float wdraw ;
    bal = bal - wdraw;
    acc_bal = bal ;
}
void show data () {
    cout << "Name: " << name
    cout << "\nAddress:" << address;
    cout << "\n City " << city;
    cout << "\n Account Balance:$" << acc_bal << endl;
}
```

```
int main ( ) {
/*
  cout << "\t\t\n\n" << "Main Menu";
  cout << "\t\n\n" << "a - Add a account,";
  cout << "\t\n" << "s - Select by letter,";
  cout << "\t\n" << "d - Deposit money!";
  cout << "\t\n" << "w - withdraw money";
  cout << "\t\n" << "s Show Account Information";
  cout << "\t\n" << "q - Quit Application \n\n";
  cout << "\t" << "Choice :";
*/
  Account cust[10]; // array of 10 instance of
                    account
  int i;
  for (i = 0; i < 10; i++) {
  cust[i].getdata ();
  cust[i].show data ();
  }
}
```

Section-C

1) Create a class shape which has two overloaded member functions: Area() & Perimeter(). Depending upon whether the dimensions are input as integers or floating-point numbers, calculate the area and perimeter of three different shapes. The dimensions of the shape would be entered by the user. The output should be in the format as the input.

→
```
#include <iostream>
#include <conio.h>
using namespace std;
class measure
{
  public:
        void shape (int r);
        void shape (int l, int b);
        void shape (float t, int d, int e);
        void shape (long a);
        void shape (float c, long int g);
        void shape (double j);
        void shape (float h, double f);
};
void measure :: shape (int r)
{ cout << " area of the circle is " << 3.14 * r * r;
}
void measure :: shape (int l, int b)
{ cout << " area of rectangle is: " << l * b;
}
```

```
void measure :: shape (float t, int d, int e)
    cout << " area of triangle is " << t * d * e;
}
void measure :: shape (long a)
{ cout << " area of square is " << a * a;
}
void measure :: shape (float c, long int g)
{ cout << " Volume of the cone is " << (1/3) * 3.14 * c * c * g
}
void measure :: shape (double j)
{
  cout << " Volume of the sphere is " << (4/3) * 3.14 * j * j * j;
}
void measure :: shape (float h, double f)
{ cout << " Volume of cylinder " << 3.14 * f * f * h;
}
int main ()
{
    int a, d, e, l, b;
    float t, c, h;
    long a;
    int ch;
    double j, f;
    long int g;
    measure obj;
    cout << " Calculation of area and volume ";
    cout << " area of circle ";
    cout << " area of rectangle ";
```

```
cout <<" area of triangle";
cout <<" area of square ";
cout <<" Volume of cone";
cout <<" Volume of sphere";
cout <<" Volume of cylinder";
cout <<" Enter your choice ";
cin >> ch;
switch (ch)
{ case 1:
        cout <<" enter the value of radius ";
        cin >> r;
        obj. shape (r);
        break;
  case 2:
        cout <<" Enter the sides of rectange ";
        cin >> b ;
        obj. shape (b);
        break;
  case 3:
        cout <<" Enter the sides of triangle ";
        cin >> d >> e;
        obj. shape (0.5, d, e);
        break;
  case 4:
        cout <<" Enter the sides of square";
        cin >> a;
        obj. shape (a);
        break;
  case 5:
```

```
cout << " Enter the radius of the cone";
cin >> c;
cout << " Enter the height of the cone";
cin >> g;
obj. shape (c, g);
break;
case 6:
    cout << " Enter the radius";
    cin >> b;
    obj. shape (b);
    break;
case 7:
    cout << " Enter the radius";
    cin >> f;
    cout << " Enter the height";
    cin >> n;
    obj. shape (f, f);
    break;
default :
    cout << " The choice entered is wrong! ";
}
getch();
}
```

(8) (a) What is Inheritance? Explain different types of Inheritance.

(b) Describe all the access specifiers used in Inheritance. Also depict the effect of Inheritance on the visibility of

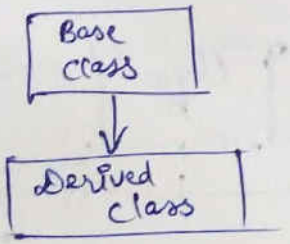
members in tabular form.

→ Inheritance is the process of creating a new class, called the derived class, from the existing class, called the base class. The inheritance has many advantages, the most important of them being the reusability of code.

However, inheritance may be implemented in different combinations in OOPs languages as illustrated in figure and they include:

- → Single Inheritance
- → Multilevel "
- → Hierarchical "
- → Hybrid "
- → Multipath "
- → Multiple "

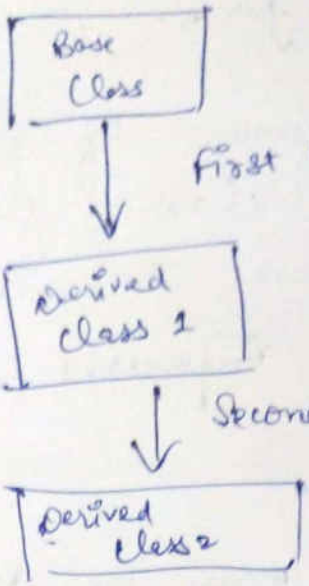
(i) Single Inheritance:-



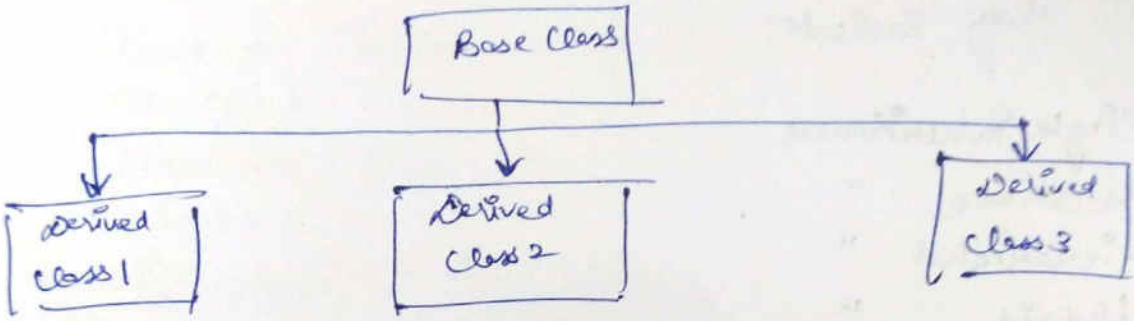
when a derived class to inherit properties and behavior from a single base class, it is called as single inheritance.

(ii) Multi-level Inheritance:-

A derived class is created from another derived class is called multilevel inheritance.

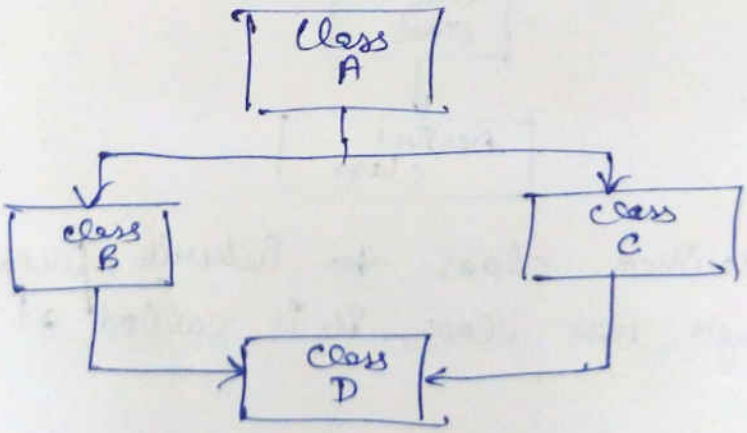


(iii) Hierarchical inheritance:-



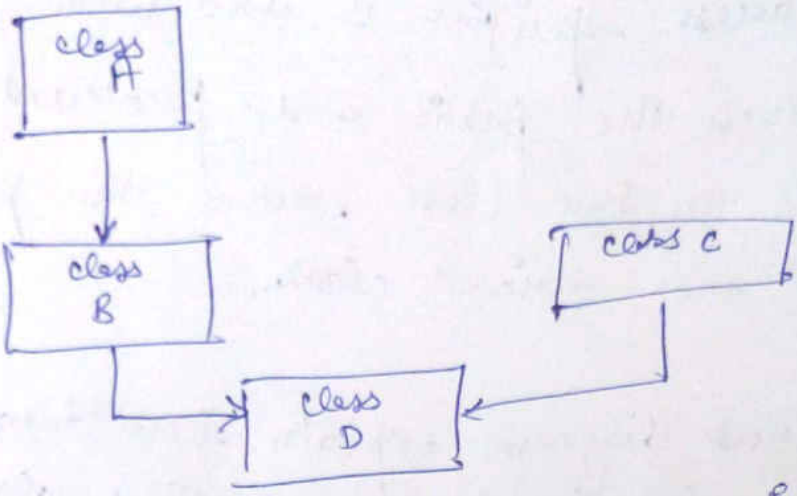
More than one derived class are created from a single base class is called Hierarchical inheritance.

(iv) Hybrid Inheritance:-



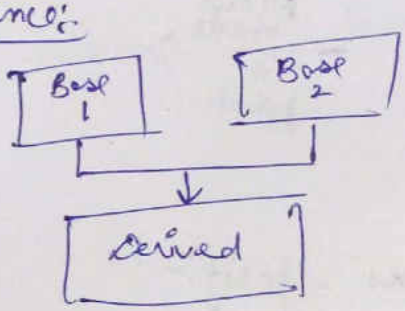
Any combination of above three inheritance (single, Hierarchical and multilevel) is called hybrid Inheritance.

Multipath Inheritance:-



Multipath Inheritance is a method of inheritance in which one derived class can inherit properties of base class in different paths. This inheritance is not supported. .NET languages such as C#.

(vi) Multiple Inheritance:



Multiple inheritances allows programmers to create classes that combine aspects of multiple classes of their corresponding hierarchies.

8(b) C++ Access Specifiers:-

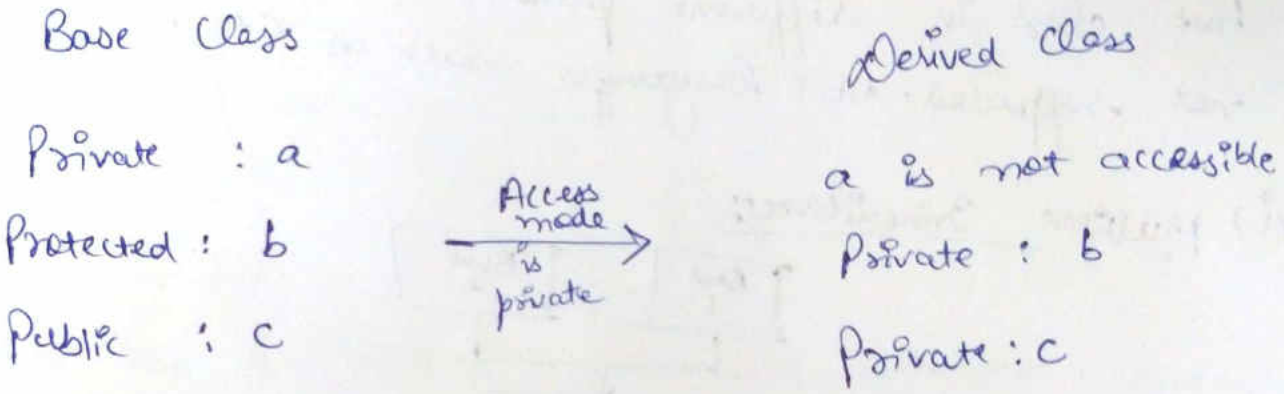
Access Specifiers can be either private or protected or public. They are divided as:

→ private acess specifier
→ protected acess specifier
→ public " "

(i) Private access specifier:-

If private access specifier is used while creating a class, then the public and protected data members of the base class become the private member of the derived class.

Following block diagram explain how data members of base class are inherited when derived class access mode is private.



(ii) Protected Access specifier:-

If protected access specifier is used while deriving class then the public and protected data members of the base class are inaccessible.

Following block diagram explain how data members of base class are inherited when derived class access mode is protected.

Base Class

Private : a

Protected : b

Public : C

Access mode is protected

Derived Class

a is not accessible

Protected : b

Protected : C

(iii) Public access specifier:

If public access specifier is used while deriving class then the public data members of the base class becomes the public member of the derived class and protected members becomes the protected in the derived class but the private members of the base class are inaccessible.

Base Class

Private : a

Protected : b

Public : C

Access mode is public

Derived Class

a is not accessible

Protected : b

Public : C

Table as asked in question:-

Base class member access specifier	Type of Inheritance: Public Inheritance	Protected Inheritance	Private Inheritance
Public	Public is derived class can be accessed directly by member	accessed directly by member and friend f^n's.	Private is derived class be accessed directly by member of friend f^n's.
Protected	Can be accessed by member and friend functions	by member and friend function.	Can be accessed directly by member and friend f^n's.
Public	can be access by member and friend f^n through public or protected member of base class.	Hidden in derived class, can be accessed by member of friend f^n's through public or protected	Hidden in derived class. can be accessed by member of friend f^n's in public & protected.

Que→ (a) Differentiate function overloading with function overriding.
(b) What are the merits and demerits of using friend function?

Answer → • Function overloading Vs Function overriding.

(1) Inheritance - Overiding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.

(2) Function signature - Overloading functions must differ in function signature ie→ either number of parameter or type of parameters should differ.

(3) Scope of functions - overriding function are in different scopes; whereas overloaded functions are in same scope.

(4) Behavior of functions → overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

Basis for comparision,	overloading,	overriding,
• Prototype	Prototype differs as number or type of parameter may differ.	All aspect of prototype must be same.
• Keyword	No keyword applied during overloading.	function which is to be overriden is preceded by keyword 'virtual', in the base class.
• Distinguishing factor	Number or type of parameter differ which determines the version of function is being called	Which class's function being called by the pointer, is determined by address of which class's object is assigned to that pointer.
• Defining pattern.	function are redefined with same name, but different number and type of parameter.	Function is defined preceded by a keyword 'virtual' in main class and refined by derived class with out keyword.
• Time of accomplishment	compile time	→ Runtime.
• Constructor/ virtual function	Constructors can be overloaded	Virtual function can be overriden

4(b)

A friend function is the friend of the given class which can access all the public, private and protected variables of that class in which it is declared. Friend function of the class is declared inside the class but is defined outside the class but it isn't the member function of the class and is defined with the keyword 'friend';

Syntax →

```
class class-name {
private:
private variable of class
Public:
friend return_type function_name (argument);
};
return_type function_name(argument)
{
```

merits of friend function are-

1) It acts as the bridge between two classes by operation on their private data's.
2) It is able to access members without need of inheriting the class.
3) It can be used to increase the versatility of overloading operator.
4) It provides functions that need data which isn't normally used by the class.

Raja

→ Demerites of friend function are →

1) It violates the law of data hiding by allowing access to private members of the class from outside the class

2) Breach of data Integrity.

3) Conceptually messy.

4) Runtime polymorphism in the member cannot be done

5) Size of memory occupied by objects will be maximum.

—O—

Dec-2015

Roll No. | | | | | | | | | | | | | | Total No. of Pages : 02

Total No. of Questions : 09

B.Tech.(CSE/ECE/Electronics & Computer Engg./ETE/IT)
(2011 Onwards)
B.Tech.(Electronics Engg. / 3D Animation & Graphics) (2012 Onwards)
(Sem.–3)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code : BTCS-305
Paper ID : [A1129]

Time : 3 Hrs. Max. Marks : 60

INSTRUCTIONS TO CANDIDATES :
1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION-A

1. Write briefly :

a) What do you understand by array of class objects? Discuss with example.

b) What is an inline function? Give example.

c) How you can open and close a file?

d) Differentiate between early and late binding.

e) Discuss different methods to pass arguments to a function.

f) What are container classes?

g) What do you mean by function overloading? Give example.

h) What is a copy constructor? Explain with example.

i) Differentiate b/w function template & class template.

j) What are dangling pointers? Give example.

SECTION-B

2. What is a friend function? Explain its importance with an example.

3. Discuss pointer arithmetic of C++ with examples.

4. What do you mean by Multiple Inheritance? Explain with the help of an example.

5. What is Operator overloading? Write a program in C++ to overload binary operator*.

6. What are templates? Write their syntax and usage. Design a function template in C++ to sort an array.

SECTION-C

7. Explain the declaration, accessing and usage of static data members and static member functions with the help of suitable examples.

8. What is Polymorphism? How do we attain Run time Polymorphism? Explain with an example.

9. Given two base classes B1 and B2 having private data members m1, n1 and m2, n2 respectively. Using multiple inheritance derive class D1 privately where data members of D1 m3, n3 stores larger of m1, m2 and n1, n2 respectively. Use constructors for objects initialization. Write appropriate functions for each class.

Section - A

Dec-2015

1(a) What do you understand by array of class objects? Discuss with the example.

→ Array could be defined as a data type which can store similar kind of data.

So array of class object is collection of all ~~kind~~ objects of class in a single set ie array.

example:

```
#include <iostream.h>
#include <conio.h>
class Emp
{
 int Id;
 char name[25];
public:
 void getdata()
 { cout << "Enter Employee Id";
   cin >> Id;
   cout << "Enter Employee name";
   cin >> Name;
 }
 void putdata()
 { cout << Id << "\t" << Name;
 }
};
 void main()
 { int i;
   Employee E[2];
   for (i=0; i<2; i++)
    { cout<<"Enter details of "<< i+1 << "Employee";
      E[i].getdata();
    }
 cout<<"Details of " << " Employees";
```

O/P

```
Enter details of 1 Employee
Enter Employee Id 101
Enter Employee Name abc
Enter details of 2 Employee
Enter Employee Id 102
Enter Employee Name def
Details of Employees
101  abc
102  def
```

```
for(i=0;i<3;i++)
E[i].putdata();
}
```

Here in this example E[i] is the objects and i is the numbers of objects, here i=3 ∴ 3 objects E[0], E[1].

Q2) What is an inline function give example.

Inline function is one of the important feature of C++. C++ provides an inline function to reduce the function call. Inline function is a function that is expanded in line when it is called. When inline function is called, whole code of the inline function gets inserted or substituted at the point of inline function call. This is done at compile time.

Inline keyword is used to make a function inline.

Eg:
```
#include <iostream.h>
using namespace std;
inline int cube(int a)
{
return a*a*a;
}
int main()
{
cout << "The cube of 10 is";
cout<<cube(10);
return 0;
}
```

O/P

The cube of 10 is 1000

We can't use loop in inline function, it should be single line simple statement otherwise compiles would skip that function.

How can you open and close a file?

→ We can create seperate files to read and write the outputs and inputs etc.

Opening a file

A file must be opened before you can read or write to it. Either ofstream or fstream objects may be used to open a file for writing. And ifstream object is used to open a file in reading mode.

Main modes are ios::in for reading
ios::out for writing

Syntax: void open (const char *filename, ios::openmode mode);

closing a file

When a C++ program terminates, it automatically terminates all the streams, releases all allocated memory and close all opened files. But it is a good practice that a programmer should always close file after use.

Syntax: Void close();

Eg.
```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{ int data;
  ifstream infile;        // object of ifstream
  infile.open("abc");     // abc = filename
  cout << "Reading data"; // opening file
  cout << data << endl;   // reading from file
  infile.close();         // closing file
  return 0;
}
```

Q-4 Difference between late binding and early binding.

Binding refers to process of converting identifiers into address.

Early binding

As name suggest, compiles directly associates an address to function call with a machine language instructions that tells the mainframe to leap the address of function.

Late binding

In this, the compiles adds code that identifies the kind of object at runtime then matches the call with the right function definition. This can be achieved using virtual function.

Early binding

```
#include <iostream>
using namespace std;
class base
{ public:
  void show() { cout<<"Base"; }
};
class derived : public base
{public
void show () { cout<<"Derived"; }
};
int main()
{
 Base *Bp;
 Bp->show();
 derived dp
 Bp = &dp;
 Bf->show();
 return 0
}
```

O/P
Base
Base

Late binding.

```
#include <iostream>
using namespace std;
class base
{ public:
virtual void show () { cout<<"Base"; }
};
class derived: public base
{ public:
void show() { cout<<"Derived"; }
};
int main()
{ Base *Bp;
  Bp->show();
  derived dp;
   Bp = &dp;
   Bp->show();
   return 0
  }
```

O/P Base
Derived

Discuss different methods to pass arguments to a function.

The main objective of passing argument to function is message passing. The message passing is also known as communication between two functions.

The methods are as follows

- call by value
- call by address
- call by reference.

→ call by value.

In this type, values of actual arguments are passed to formal arguments and operation is done on formal arguments. Any change in the formal argument does not change the actual arguments because formal arguments are photocopy of actual arguments.

```
fname ( int, int )
```

→ call by Address

In this type, instead of passing values, addresses of actual parameters are passed to the function by using pointers. Function operated on address instead of value.

```
fname ( int *, int * )
```

→ call by Reference

In C++ it is possible to pass arguments by reference. In this '&' operator is used. They declare aliases for object variables and allow the programmer to pass argument by reference.

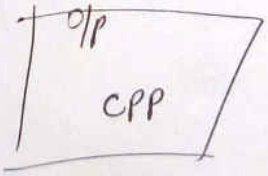
```
fname (int &a)
```

Q-1 what are containes classes?

A containes class is a class that is used to hold objects in memory and external storage. A containes class act as a generic holder. A containes class has a predefined behaviour and a well-known interface. A containes class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a containes class contains a group of mixed objects, the containes is called the heterogenous containes; when the containes is holding a group of objects that are all the same, containes is called homogenous containes.

eg:

```
class first {
public:
   void show() { cout << "CPP"; }
};
class second {
   first f;
   public:
     second()
       {
         f.show();
     }; }
int main()
{
  second ob;
return 0;
}
```



What do you mean by function overloading ? Give example.

Polymorphism is a very important concept of cpp in which same function act differently in different situation.

Function overloading is a part of polymorphism in which same function act differently

Example

```
#include <iostream>
using namespace std;
Void show (int a)
{ cout << "Integer"; }
Void show (double a)
{ cout << "Double"; }
Void show (char const *c)
{ cout << "character"; }
int main ()
{
  show (10);
  show (10.10);
  show ("abc");
  return 0;
}
```

o/p

Integer
Double
Character.

4. What is a copy constructor? Explain with example.

→ In object oriented programming language constructor is a special kind of function that is automatically called whenever object of class is created.

Copy constructor is a type of constructor which is used to creat a copy of an already existing object of a class type. It is usually of form X(X&), where X is class name. The compiler provides a default copy constructor to all the classes.

```
#include <iostream>
using namespace std;
class point
{ private:
    int x,y;
public;
    point (int x1, int x2) { x=x1; y=x2; }
    point (const point &p2) { x=p2.x; y=p2.y; }
    int get x() { return x; }
    int get y() { return y; }
};
int main()
{
    point p1 (10,15);
    point p2=p1;
cout << p1.get x()<< "\t"<< p1.get Y(),
cout << p2.get x()<< "\t"<< p2.get Y());
return 0;
}
```

o/p

10	15
10	15

Diff. bw function template and class template.

A template is a c++ programming feature that permits function and class operations with generic types, which allows functionality with different data types without recoding entire code block for each type.

We could use class template when you want to create a class that is parameterised by a type, and a function template when you want to create a function that can operate on many different types.

eg:

```
template <typename T1, typename T2>
class pair {
    T1 first;
    T2 second;
};
```

} class template

```
template <typename T>
T min (T a, T b)
{ return a<b ? a : b;
}
```

} function template

∴ A class template becomes a class, a function template becomes a function.

1 What are dangling pointers? Give example.

Dangling pointers arise when an object is deleted or de-allocated without modifying the value of the pointer, so that the pointer still points to the memory location of the de allocated memory. In short pointer pointing to non-existing memory location is called dangling pointer.

It could also be defined as a pointer that points to invalid data or data which is not valid anymore.

example.

```
class *obj = new class();
class *obj2 = obj;

delete obj;
obj = nullptr;
```

In this example
obj2 is pointing to something which is not valid anymore.

The pointer can become dangling pointer in 3 ways.

- De-allocation of memory
- calling a function (by another ptr that points to something invalid)
- Variable goes out of scope.

Section - B

2. What is friend function? Explain its importance with an example.

Ans → Friend functions are those functions which can access all the functions and variables of a class though it is not a member function of that class. Actually to share a function among two or more classes friend functions are used. If it is declared so, then it will able to access all variables and functions of those classes

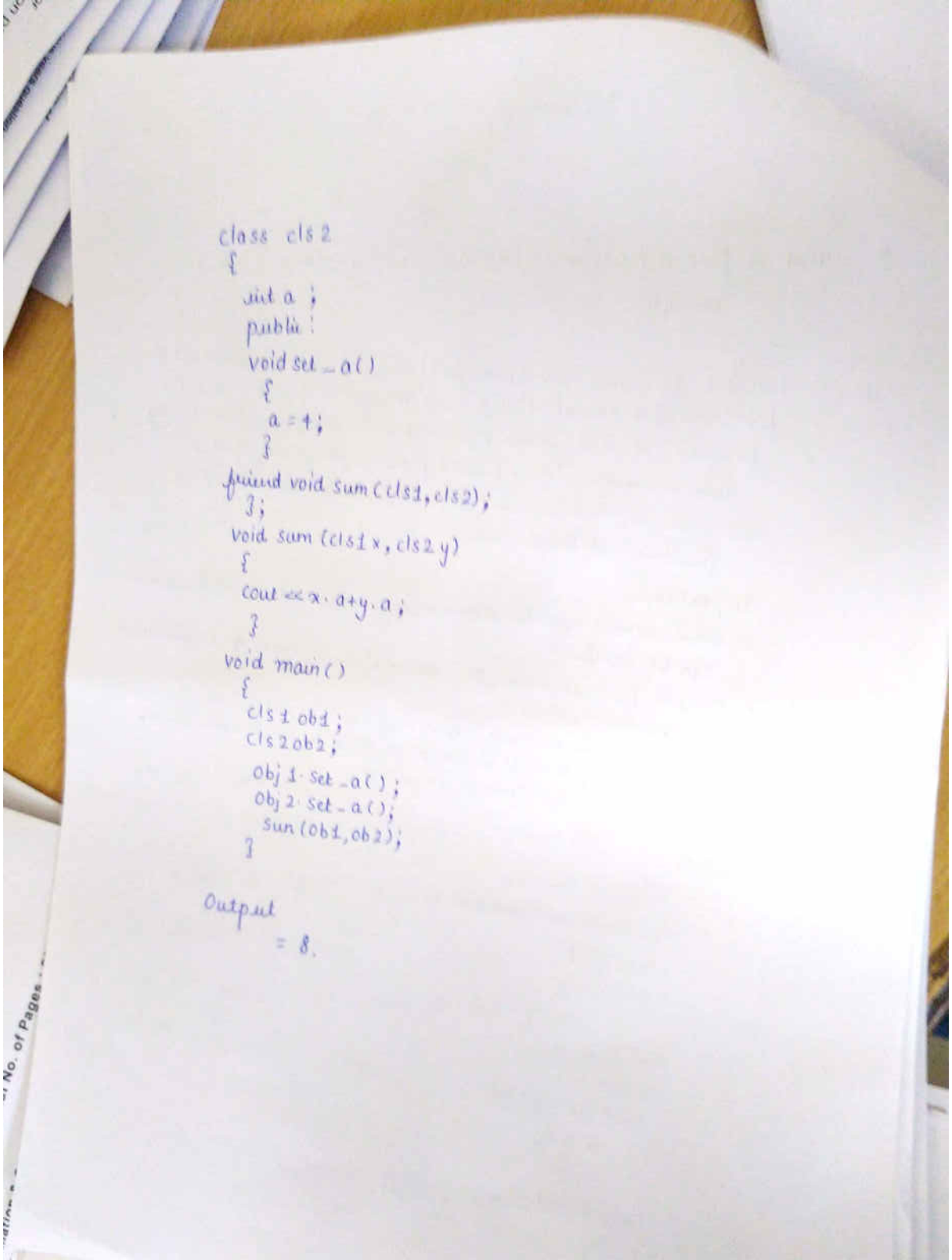
Importance :-

1. When certain operator overloading is required friend functions can be useful.
2. Friend function makes the creation of some types of I/O functions easier.
3. Sometimes two or more classes may contain interrelated members which may need to be operated at a time. In such times, a friend function is required.

Example :-

```
// friend function
#include <iostream.h>
class cls 2 ;
class cls 1

int a ;
public :
 void set_a ( ) {
     a = 4 ;
      }
friend void sum (cls 1, cls 2) ;
  } ;
```



```
class cls2
{
  int a ;
  public:
  void set_a()
   {
    a = 4;
   }
friend void sum(cls1, cls2);
 };
void sum (cls1 x, cls2 y)
 {
 cout << x.a+y.a;
 }
void main()
 {
  cls1 ob1;
  cls2 ob2;
   obj1.set_a();
   obj2.set_a();
    sum(ob1, ob2);
 }
```

Output

= 8.

Ques-3. Discuss pointer arithmetic of C++ with examples.

Ans→ Pointers are variables of integral type, because addresses are integers. Pointers also have information about the type to which they point. Thus it makes sense to allow certain kinds of arithmetic for pointers.

- The unary operators ++ and -- (pre and post-fix versions of each)
- (pointer)+(int) = (pointer)
- (pointer) − (pointer) = (int)

Note that these operations are defined only when they make sense. For eg. if two pointers p_1 and p_2 point two diff arrays, then $p_1 - p_2$ is not defined.

```
int  A[10]
int* A = new int[10]
```

Pointer Arithmetic

Example

```
// T is some type
T*A = new T[20] //A is the address of the first T object
T*B;   // B can store an address for T but has no valid value
++A; //now A is the address of the second object
B = A--; // assigns B the value of A, then decrements A
++B; // increments B; now B points to the index 2 element of A.
Std:: cout << B - A; // output is 2, the difference b/w the indexes.
A = B+3; // now A is the address of the index 5 element
```

The following illustrates the use of pointer arithmetic in loop control variables:

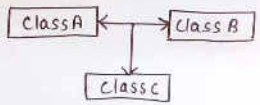
```
char A[10] = {'a','b','c','d','e','f','g','h','i','j'};
char *i, *j, *k;
for (i=A; i!=A+10; ++i)
 {
```

```
for (j=A; j!=A+10; ++j)
 {
  for (K=A; K!=A+10; ++K)
  {
   std::cout *i<<*j<<*k<<'\n';
  }
 }
}
```

Ques-4. what do you mean by multiple inheritance? Explain with the help of an example.

Ans → If a class is derived from two or more base classes then it is called multiple inheritance In C++ multiple inheritance a derived class has more than one base class.
It is inheritance of characters by a child from mother and father

C++ multiple Inheritance Block Diagram



Multiple Inheritance

```
# include <iostream.h>
 using namespace std;
  class A
  {
    public:
    int x;
    void getx()
   {
    cout << "entervalue of x:"; cin >> x;
    }
  };
 Class B
 {
```

```
public:
int y;
void gety()
{
cout <<"enter value of y :"; cin >>y;
   }
};
class c : public A, public B
 {
  public:
   void sum()
    {
    cout <<"Sum = " << x+y;
       }
    };
   int main()
  {
   c obj1;
   obj1.getx();
   obj1.gety();
   obj1.sum();
   return 0;
}
```

Output

enter value of x : 5
enter value of y : 4
Sum : = 9

Ques-5. What is operator overloading ? Write a program in C++ to overload binary operator".

Ans : If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading. In C++ we can overload methods, constructors,

Overloaded properties

Types of overloading in C++

- Function overloading
- operator overloading

Binary operator Overloading

```
#include <iostream.h>
#include <conio.h>
class complex {
        int a,b;
        public:
        void getvalue()
        {
        cout<<"Enter value of Complex nos a,b:";
        cin>>a>>b;
        }
Complex operator+(complex ob)
 {
  complex t;
  t.a = a+ob.a;
  t.b = b+ob.b;
   return(t);
  }
complex operator-(complexob)
   complex t;
   t.a = a-ob.a;
   t.b = b-ob.b;
    return(t);
  }
   void display()
   {
    cout<<a<<"+"<<b<<"i"<<"\n";
   }
  };
  void main()
  {
   clrscr();
```

```
complex obj1, obj2, result, result1;
  obj1.getvalue();
  obj2.getvalue();
  result = obj1 + obj2;
  result = obj1 - obj2;
  cout << "Input values :\n";
   obj1.display();
   obj2.display();
   cout << "Result";
   result.display();
   result1.display();
    getch();
  }
```

Sample output

Enter the value of complex no.s

4 5

Enter the value of complex no.s

2 2

Input value	Result
4 + 5i	6 + 7i
2 + 2i	2 + 3i

Ques-6. what are templates? write their syntax and usage. Design a function template in C++ to sort an array.

Ans :- A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for diff data types.

Syntax

```
template <class Type>
  function defination;
```

Why Type is called a formal parameter of the template.

* Type:
→ Specifies type of parameters
→ Specifies return type
→ Declares variables within the function

Usage:

Function templates are implemented like regular functions, except they are prefixed with the keyword template.
Using function templates is very easy just use them like regular functions. When the compiler sees an instantiation of the function template for ex the call max(10,15) in fn main, the compiler generates a fn max(int, int).

Program:

```
#include <iostream>
using namespace std;
template <class T>
void bubble sort (T a[], int n)
{
for (int i=0; i<n-1; i++)
for (int j=n-1; i<j; j--)
if (a[j] < a[j-1])
swap (a[j], a[j-1]);
}
int main () {
int a [5] = {10, 50, 30, 40, 20};
int n = size of (a) / size of (a[0]);
bubble sort (a, 5);
cout << " sorted array :";
for (int i = 0; i<n; i++)
cout << a[i] << " ";
cout << endl;
return 0;
```

Output → Sorted array → 10, 20, 30, 40, 50

Section-C

Q7 Explain the declaration, accessing & usage of static data members & static member function with the help of suitable example.

Ans Static data members: These are those members which are declared by using the keyword "static" in front of the data member. Static members always have default value "zero" for int & for string it is "null". Always remember that static data members are used in static member functions.

Static member functions: Those which are declared by using "static" keyword in front of their name. and in this we use we use the static data members.

* Declaration of Static data members & Static member functions :-

⇒
```
Class A{
public:
    static int Ob;          // static data-member
A(){
  Ob++;
}
static int getO(){          // static member-function
    return Ob;
}
};
```

```
int A::ob=0;
int main(){
cout<<"Initial value in ob variable "<<A::getO();
A obj1();
A obj(8);
cout<<" final value stored in ob variable"<<A::getO();
return 0;
}
```

Output ⇒ Initial value in ob variable 0
// Constructor called in background
// Constructor called in background
final value stored in ob variable 2

In the above example the static data-member & static data functions are used & the are accessed by the following commands:

First - static data member "ob" is accessed inside the constructor "A()" of "class A". i.e. `ob++;`

Second - The static data function or the static member function is accessed by the main function using the scope resolution operator only rather than making a separate object to call that. i.e.

`A::getO();`

→ This helps in reducing the space complexity of the program. as we don't need to create objects to call static member functions. We can just use them wherever we want to perform operations without making different object & hence without consuming extra memory.

Ex: A program that counts the entry of people at a certain place with their details stored in memory

```
=> class count {
    public:
        static int countP;
    count(){
        cout<<"Enter Persons Name";
        cin >> name;
        countP++;}
    static int getCount(){
        return countP;
    }
int count::countP = 0;
int main()
{
    Cout D;
    Count A;
    count B;
    count C;
    cout<<" Total People in :"<< count::getCount();
    return 0;
}
```

Calling static member function with out object, directly by using scope resolution operator.

Output :

```
Enter Person's Name Piyush
Enter Person's Name Kushagar
Enter Person's Name Rahul
Enter Person's Name Ishu
Total People in : 4
```

Q. What is Polymorphism? How do we attain Run time Polymorphism? Explain with an example.

Ans. **Polymorphism**: It is a concept of object oriented programing languages & it simply means existance of a single interface to entities of different types or the use of a single symbol to represent different types.

⇒ A function / operator can behave differently depending on the context in which they are used.

⇒ We have already seen operator overloading in which a single operator '+' serves different purposes for different data types.

There are 2 types of P

Run-time polymorphism: This type of polymorphism is achieved by function overriding. function overide occurs when one of the derived class has the definition of one of the function of base class, that base class function is said to be overriden. This is achieved by using virtual functions.

- Virtual functions: It is a function type which is declared normally as a normal function but a keyword "virtual" is also written in front of it while declaring the function name. except that the defination & calling is similar to that of a normal function.

Syntax:

```
virtual function_return_type function_name( ){
    // defination
}
```

⇒ This kind of function supports overriding & hence used in run-time polymorphism. These functions have very much & very very much, I mean what of of never ending importance in object oriented programming languages.

Example:

```
class Complex{
    private
        int real, imag;
    public
        Complex( int r=0, int i=0)
        {   real = r;
            imag = i;
        }
    Complex operator + (Complex const &obj){
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print () {
        cout << "real << " +i" << imag ;
        }
int main()
{
    Complex c1(10, 5), c2(2,4);
    Complex c3 = c1+c2; (1st calls operator+)
    c3.print();
}
```

Output:

12 + i9) // Adds the given 2 complex no's

MAY-2016

Roll No. | | | | | | |

Total No. of Questions : 09 Total No. of Pages : 02

B.Tech.(Electronics Engg./3D Animation & Graphics) (2012 Onwards)
B.Tech.(CSE/ECE/Electronics & Computer Engg./ETE/IT) (2011 Onwards)
(Sem.–3)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code : BTCS-305
Paper ID : [A1129]

Time : 3 Hrs. Max. Marks : 60

INSTRUCTIONS TO CANDIDATES :
1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION-A

1. Write briefly :

1. Draw flow chart to find the largest of three numbers.
2. Why are classes in C++ called Abstract Data Types?
3. Explain the difference between abstraction & data hiding.
4. Why are classes in CPP called Abstract Data Types?
5. Explain the use of get & put pointer in file handling.
6. Explain how memory is allocated to classes & objects.
7. Distinguish between static members & variables. How are they useful?
8. What are virtual constructors? Give relevant examples to explain it.
9. What are static functions? Explain how friendship function is used in C++.
10. What are the various input statements of C++?

SECTION-B

2. What is a constructor and destructor? What is the use of default & copy constructors? Is a constructor mandatory for a Class? Explain by giving examples in each case.

3. Explain how base class member functions can be invoked in a derived class if the derived class also has a member function with the same name.

4. What is a virtual function? Explain its usage with example.

5. What are the various File Opening modes? How is (ios::app) mode different from (ios::ate mode)?

6. Explain what is overloaded operator & how does a compiler proceed to execute an overloaded operator.

SECTION-C

7. Write a program to copy the content of a data file to another file. Make use of the exception handling conditions also.

8. Write a class to represent a vector (a series of float values). Include member functions to perform the following tasks:

   a) To create the vector

   b) To modify the value of a given element

   c) To multiply by a scalar value

   d) To display the vector in the form (10, 20, 30,...)

   Write a program to test your class

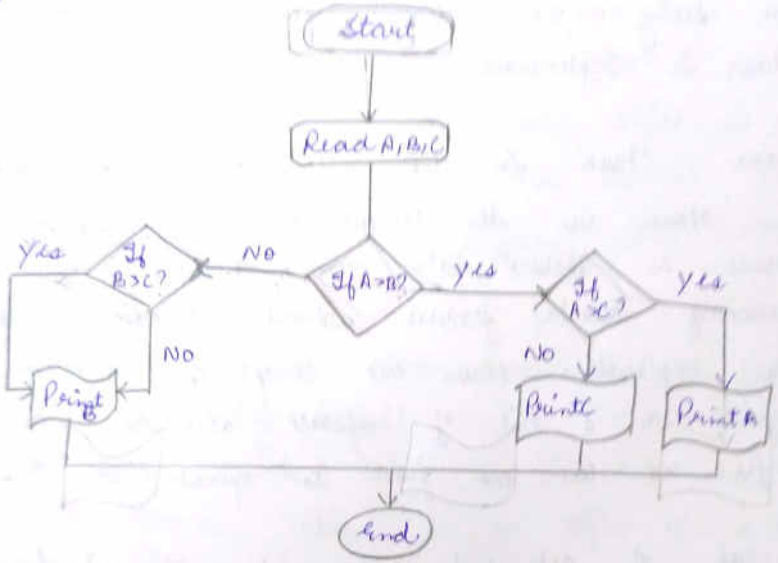
9. Write a program to overload the plus operator to add two complex numbers.

May 2016 Examination

SECTION-A

Q.1. Draw a flowchart to find largest of three no.

Ans:-



Q.2. Why are classes in C++ called abstract data types?

Ans Since the Classes use the concept of data abstraction, they are known as abstract data types. Abstract refers to act of representing essential features without including background details or explanation. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate these attributes.

Q.3. Explain the difference b/w abstraction & data hiding?

Ans. Abstraction: Abstraction allows us to represent complex real world in simplest manner. It is process

of identifying the relevant qualities and behaviors of an object should possess, in other words, represents necessary feature without representing background details.

Data hiding: It is a process of hiding all internal details of an object from outside real world. It restricts client from seeing its internal view where behaviours of abstraction is implemented.

Q4. Why are Class in CPP called abstract data types?

Ans. Since the classes use the concept of data abstraction, they are known as abstract data types. Abstraction refers to act of representing essential features without including background details or explanation. Class use concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes

Q5. Explain use of get and put pointer in file handling.

Ans Each file has associated two pointers known as file pointer. One of them is called input or get pointer and other is called output or put pointer. We can use these pointers to move through the files while reading or writing. The pointer is used for reading the contents of given file location. Each time an input or output operation takes place the appropriate pointer is automatically advanced.

Q.6. Explain how memory is allocated to Classes & objects.

Ans Class is not allocated any memory. This is partially true. The functions in a class are allocated memory which are shared by all the objects of class. Only data in class is allocated memory when an object is created. Every object has its own memory for data.

allocation to object: We have stated that space for objects is allocated when they are created not when the class is specified. This statement is partially true. Actually the member functions are created and placed in memory space only once when they are defined as a part of class specification. Since all objects belonging to that class use same member functions, no separate space is allocated for member functions when objects is created, separate memory locations for the objects are essential because the member variables will hold different data values for different objects.

Q.7. Distinguish between static members & non-static variables. How are they useful?

Ans A static member belongs to a class where as a non-static member belongs to an object of a class. This means that a static member can be called by a class and by object of class. A static member can access only other static members of class, where as a non static member can access both static & non-static member of a class.

There is only one copy of a static variable & its value remains the same even when the class is instantiated whereas in case of non-static variables, every time the class is instantiated, the objects have their own copy of these variables.

Q.8 What are virtual constructor? Give relevant examples to explain it.

Ans A Constructor can't be virtual. There are some reasons that justify this statement. First, to create Object the Constructor of object class of must be of type of Class. But this is not possible with a virtual Implemented Constructor, the virtual table would not have been created to resolve any virtual function calls. Thus a virtual Constructor itself would not have anywhere to look up to. As a result, it is not possible to declare Constructor as virtual.

Q9. What are static functions? Explain how friend function is used in C++.

Ans Like static member variables, we can also have static member functions. A member function that is declared static has following properties:

(a) A static function can have access to only other static members declared in same class.

(b) A static member function can be called using class-name as follows:

Class_Name :: function name

Friend functions: The functions that are declared with 'friend' keyword are known as friend functions. A function can be declared as a friend in any no. of classes. It has full access rights to private members of Class.

Features of Friend function:

1. It can be invoked like normal function, with help of object.
2. It has objects as argument.
3. It is not in scope of class to which it has been declared as friend.

What are various input statements in C++?

When computer gets data from keyboard, the user is said to be acting interactively. Putting data into variables using cin & the operator >>. The syntax of cin together with >> is

cin >> variable;

If two variable then

cin >> variable 1 >> variable 2;

This is called input statement. In C++, >> is called stream extraction operator.

4. int feet;
   int inches.

Then input is cin >> feet >> inches;.

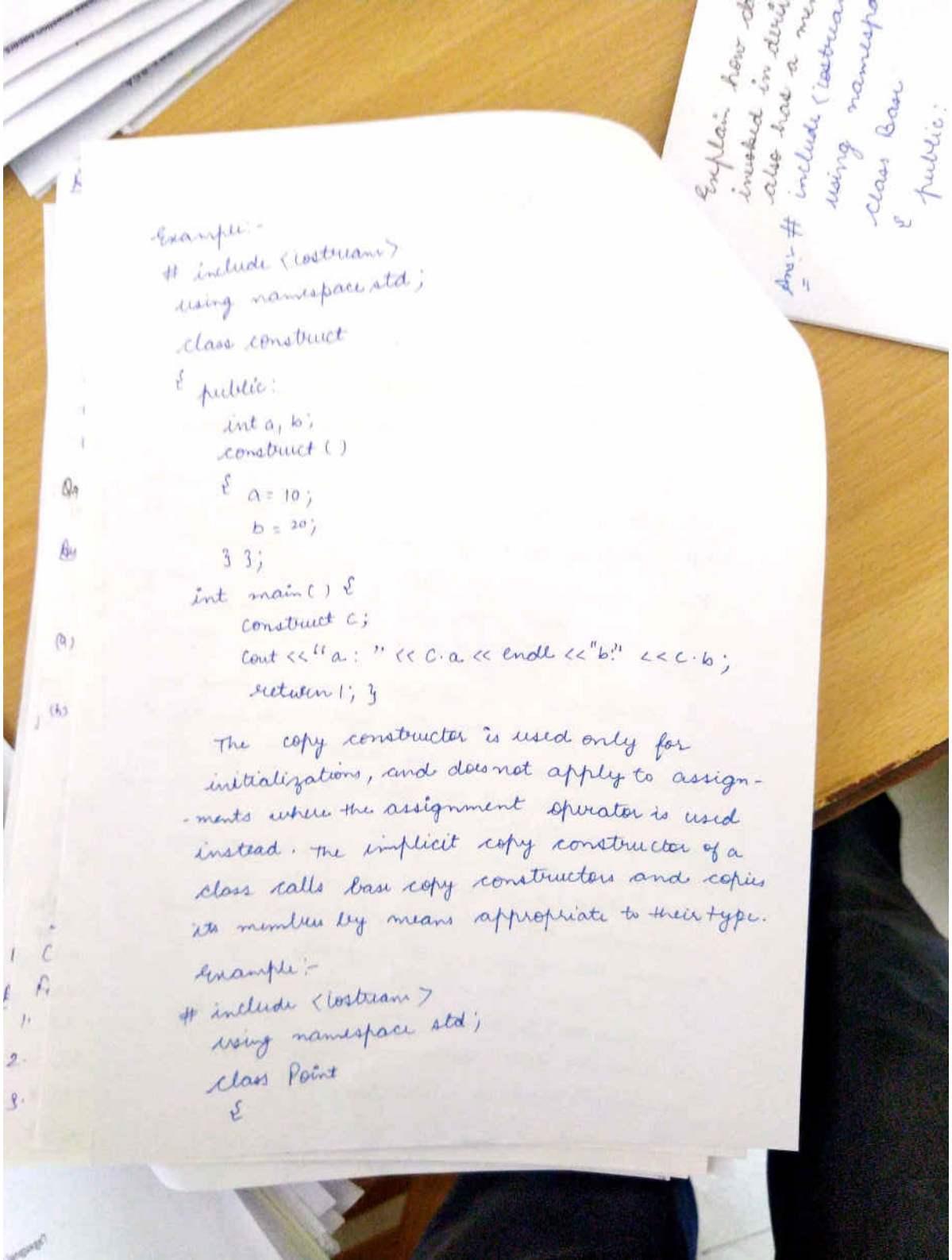
SECTION-B

Q2:- What is constructor and destructor? What is the use of default & copy constructors? Is a constructor mandatory for a class? Explain by giving examples in each case.

Ans:- In class-based object-oriented programming, a constructor is a member of a class, it is generally of void type means a constructor never returns any value. It is used to assign the class level variables. It is a special type of subroutine called to create an object.

A distructor is a special member function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime.

A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values. The compiler will implicity define A::A() when the compiler uses this constructor to create an object of A. The constructor will have no constructor initializer and a null body.



Example:-

```
# include <iostream>
using namespace std;
class construct
{ public:
    int a, b;
    construct ()
    { a = 10;
      b = 20;
    } };
int main() {
    construct c;
    cout << "a: " << c.a << endl << "b:" << c.b;
    system("pause"); }
```

The copy constructor is used only for initializations, and does not apply to assignments where the assignment operator is used instead. the implicit copy constructor of a class calls base copy constructors and copies its members by means appropriate to their type.

Example:-

```
# include <iostream>
using namespace std;
class Point
{
```

Explain how class member functions can be invoked in derived class if the derived class also has a member function with the same name.

Ans:-

```
# include <iostream>
using namespace std;
class Base
{ public:
  virtual void print( ){
    cout << "Printing from base" << endl;
  }
  void print (int num){
  cout << "printing number from base : " <<
          num << endl;
  } };
class Derived : public Base
{ using Base :: print;
public:
  void print( ){
    cout << "Printing from derived " << endl;
  } };
int main( )
{
  Derived x;
  x. print( );
  x.Base :: print( 1 );
  // x. print(1); // gives a compilation error
  return 0;
}
```

```
private:
  int x, y;
public:
  point (int x1, int y1)     { x = x1; y = y1; }
   // copy constructor
  point (const point & p2) { x=p2.x; y = p2.y; }
  int getX( )   { return x; }
  int getY ( )  { return y; }
  };
int main( )
{
   point  p1(10, 15) ; // Normal constructor is
                          called here
  point p2 = p1.; // copy constructor is called here.
  cout << " p1. x = " << p1. getx( ) << ", p1. y = "
                                  << p1. gety( );
cout << "\n p2.x = " << p2. getx( ) << ", p2.y = "
                                  << p2. gety( );
  return 0;
}
```

~~strictly~~ It is never mandatory to have a default constructor. It is mandatory to have a no args constructor only if it is explicitly or implicitly called. If there is no constructor present in a class, one default constructor is added at compile time.

Q4 :- What is virtual function? Explain its usages with example.

Ans :- A virtual function a member function which is declared within base class and is redefined by 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 function.

(i) Virtual functions ensure that the correct fn is called for an object, regardless of the type of reference used for function call.

(ii) They are mainly used to achieve Runtime polymorphism.

(iii) Functions are declared with a virtual keyword in base class.

(iv) The resolving of function call is done at Run-time.

Example :-

```
#include <iostream>
using namespace std;
class base
{ public:
   virtual void print( )
   { cout << "print base class" << endl; }
   void show( )
   { cout << "show base class" << endl; }
```

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

Ques 5:- What are the various file operating modes? How is (ios::app) mode different from (ios::ate mode)?

Ans.

MODE	MEANING	FOPEN RETURNS IF FILE:-	
		EXISTS	NOT EXISTS
→ r	Reading	–	NULL
→ w	writing	overwrite on existing	Create new file
→ a	Append	–	Create new file

r+	Reading + Writing	New data is written at the beginning overwriting existing data	Create new data File
w+	Reading + writing	over write on existing	Create New file
a+	Reading + Appending	New data is append at the end of file	Create new file

ios:: ate " sets the stream's position indicator to end of the stream on opening. "
ios:app " set the stream's position indicator to the end of the stream before each output operation. "
This means the diffference that ios:: ate puts your positions to the end of the file when you open it. ios:: app instead puts it at the end of the file every time you flush ~~when~~ your ~~open~~ it. stream.

Ques 6:- Explain what is overloaded operator & how does a compiler proceed to execute and overloaded operator.

Ans:- operator overloading is syntactic and is used because it allows programming using notation nearer to target domain. and allows user- defined types a similar

level of syntactic support as types builtin to language.

operator of overloading does not change the expressive power of a language, as it can be emulated using function calls.

Example:-

```
class complex
{ private:
    int real;
    int img;
  public:
    complex(int r, int i) { real = r; img = i; }
    complex Operator ++(int);
    complex & operator ++();
};
complex & complex::operator ++()
{ real++; imag++;
  return * this;
}
complex complex::operator ++(int i)
{ complex C1(real, imag);
  real++, imag++;
  return C1; }
int main()
{ complex C1(10, 15);
  C1++; ++C1; return 0; }
```

SECTION-C

Write a program to copy the content of a data file to another file. Make use of exception handling "and" also.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ifstream fin;
  fin.open("data.txt", ios::in);
  Ofstream fout;
  fout.open("new.txt", ios::out);
  char ch;
  char line[80];
  int i=1;
while (fin.get(ch)) {
  fin.get(line, 80, '.');
fout << "line" << i << ":" << line << endl;
i++;
}
  fin.close();
  cout << "done" << endl;
  return 0;
}
```

Q.7. Write a class to represent a vector (a series of values). Include member functions to perform following operations.

(a) To create the vector.

(b) To modify value of given element.

(c) To multiply by a scalar value.

(d) To display the vector in form (10, 20, 30, ...)

Write a program to test your class.

Ans.

```
Class Vector
{
  float vec[10], temp, temp2;
  void create()
  {
     int i;
    cout << "Enter values of i";
    for(i=0; i<10; i++)
    {
       cin >> vec[i];
    }
  }
  void modify()
  {
    cout << "Enter elements to modify";
    cin >> temp;
    cout << "Enter new value";
    cin >> temp2;
    for(i=0; i<10; i++)
    {
      if(vec[i] == temp)
      {
        vec[i] = temp2;
      }
    }
    void display()
    {
      for(i=0; i<10; i++)
        cout << vec[i];
```

```
};
void main()
{
    vector v1;
    v1.create();
    v1.modify();
    v1.display();
    getch();
}
```

Q.a. Write a program to overload the plus operator to add two complex no.

A1.

```
#include<iostream>
using namespace std;
Class A
{
    int a, b;
    public:
    A() { };
    A(int i, int j)
    {
        a = i;
        b = j;
    }
    void show()
    {
        cout << a << "+i" << b;
    }
    A operator +(A);
};
A A::operator +(A obj)
{
    A temp;
    temp.a = a + obj.a;
```

```
    temp.b = b + Obj.b;
    return (temp);
}
int main()
{
    A c1(5,6), c2(7,8), c3;
    cout << "The 1st no is:";
    c1.show();
    cout << "\n The 2nd no is:";
    c2.show();
    c3 = c1 + c2;
    cout << "\n the sum is:";
    c3.show();
}
```

Dec-16,

Roll No. | | | | | | | | | |

Total No. of Questions : 09

Total No. of Pages : 02

B.Tech.(Electronics Engg./3D Animation & Graphics) (2012 Onwards)
B.Tech.(CSE/ECE/Electronics & Computer Engg./ETE/IT) (2011 Onwards)
(Sem.–3)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code : BTCS-305
Paper ID : [A1123]

Time : 3 Hrs.

Max. Marks : 50

INSTRUCTION TO CANDIDATES :

1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION-A

1. Write briefly :

   1. What is meant by object slicing?
   2. What is the use of this pointer?
   3. What is the functionality for error handling file could not open in C++?
   4. What is the use of inline member functions?
   5. How is the order of invocation of destructors different from constructors?
   6. What are enumerated data types?
   7. What is meant by precedence rule for evaluation of expressions?
   8. What are the major features of structured languages?
   9. What is the difference between high level and a low level language?
   10. Consider a pointer declaration int i=10,*p; p=&i;

       Is p++ a valid statement, justify?

1 | (S2)-992

SECTION-B

2. What is the main difference between array of pointers and pointer to an array? Explain with the help of a suitable example.

3. Write a program in C which implements tower of Hanoi using recursion.

4. Write a program to search a key string in a array of strings, if a key string is found then return its position and then replace that key string by any string using pointers.

5. Describe the following manipulators :

   a) Setw()

   b) Setiosflags

   c) Setprecision

   d) Setfill()

   e) Resetiosflags()

6. What is the difference between static and dynamic memory allocation? What are inline functions? Explain.

SECTION-C

7. Explain how base class member functions can be invoked in a derived class if the derived class also has a member function with the same name. How are stream operators overloaded in CPP?

8. a) What do you understand by inheritance? Give its various types and access mechanisms. What are the advantages of scope resolutor & referencing?

   b) Briefly describe the class hierarchy provided by C++ for stream handling.

9. a) When are C++ copy constructors, assignment operators, and destructors, respectively, invoked?

   b) At what time are overloaded methods (as opposed to overridden) resolved?

SECTION-A

Dec-16

Ques 1 A) What is meant by object slicing?

Ans 1 A) Object slicing occurs when you assign derived class objects using base class variables. The result of this is that the derived class part of an object can be "sliced off" and lost.

Consider the following code, where Derived is derived from Base:

```
Derived d;
Base b = d;
```

Ques 1 B) What is use of this pointer?

Ans 1 B) Every object has access to its own address through an important pointer called 'this' pointer. The 'this pointer' is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to the invoking object.

Friend functions donot have a this pointer because friends are not members of a class. Only member f^n's have this pointer.

Q1C) What is the function used for error handling if file couldn't open in C++?

Ans 1C) An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

(1) throw :- a program throws an exception when a problem shows up. 'throw' keyword is used.

(2) catch :- A problem catches an exception with an exception handler at the place in a program where you want to handle the problem. 'catch' keyword is used

(3) try :- try block identifies a block of code for which particular exceptions will be activated; followed by one or more catch block.

Ques 1D) What is the use of inline member functions?

Ans 1D) The inline functions are used to increase execution time of a program. functions can be instructed to compiler to make them inline so that compiler can replace those function's definition wherever those are being called.

Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime.

To make a function inline, start its definition with the keyword "inline"

How is the order of invocation of destructors different from constructors?

Ans It

class X (Base class 2)

↓

class Y (Base class 1)

↓

class Z (Derived class)

Order of constructor call :-

1. X()
2. Y()
3. Z()

The constructors are called in order :- first of all, the base class is called, then the first derived class is called then the last class is called at the end

↓

constructor.

Order of invocation of destructors :-

1. ~Z()
2. ~Y()
3. ~X()

The derived class' destructor is called first followed by the second derived class' then destructor then the base class destructor is called

How is the order of invocation of destructors different from constructors?

class X (Base class 2)

↓

class Y (Base class 1)

↓

class Z (Derived class)

Order of constructor call :-

1. X()
2. Y()
3. Z()

The constructors are called in order :-

first of all, the base class is called, then the first derived class is called then the last class is called at the end

↓

constructor.

order of invocation of destructors :-

1. ~Z()
2. ~Y()
3. ~X()

The derived class' destructor is called first followed by the second derived class' destructor then the base class destructor is called

Q1F What are enumerated data types?

Ans1f An enumerated data type declares an optional type name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration.

Program :-

```
#include<iostream>
using namespace std;
enum Genders{MALE, FEMALE};
int main()
{
    Genders gen = Genders.FEMALE;
    return 0;
}
```

Q1G What is meant by precedence rule for evaluation of expressions?

Ans1G Order of evaluation of any expression including order of evaluation of function arguments is unspecified.
Operator precedence tells us how to group expressions.
|| and && are special in that the first operand is always evaluated first and the second operand

(including all sub-expressions) is only evaluated if it is required to determine value of the expression.

for ||, if first operand evaluates to true the second operand is not evaluated because the result of logical-or will always be true.

Similarly, && will not be evaluated if first operand evaluates to false as the logical-and must be false in this case.

Q.1H Write down major features of structured language?

Ans 1H. Structured language is actually a concept used to improvise the code over common procedural language. Structured language is somewhat similar to OOPS, which are blocks of code from sub-routines that define a program's functionality :-

features are :-

- Division of large problem into small procedures and functions.
- absence of goto statement.
- main statements include if-else, case and case statements.

- inclusion of facilities for declaring entry points and external references.
- Extensive set of operators include arithmetic, relational, logical, bit manipulation, shift and part word operators.

Q11 What is the difference b/w high and low level language?

Ans11.

	HIGH LEVEL LANGUAGE	LOW LEVEL LANGUAGE
(1)	A programmer friendly language that provides a high level of abstraction from the hardware.	A machine friendly language and provides no or less abstraction from the hardware.
(2)	Slower than a low level language.	Faster than a high level language.
(3)	Not memory efficient language.	More memory efficient language.
(4)	Requires a compiler or an interpreter to convert program code into machine language.	Requires an assembler to convert program to machine code while machine language is executed by computer directly.

	easily understandable by the programmer.	Easily understandable by the computer.
(6)	Machine independent language	machine dependent language.
(7)	Can run on multiple languages, so portable.	not portable.

Q1] Consider a pointer declaration int i=10; *p; p=&x; is p--; a valid statement, justify?

Ans1]

```
int i=10;
int p,x;
```

p-- is not a valid statement because here p-- can goto segmentation

SECTION-B

Q2) What is the main difference b/w array of pointers and pointer to an array? explain with the help of an example.

Ans 2)

ARRAY OF POINTERS	POINTER TO AN ARRAY
① Array of pointers is an array of the pointer variables. It is known as pointer array.	Pointer to an array is also known as array pointer. We are using pointer to access components of array.
② SYNTAX :- `int *varname[arraysize];`	② SYNTAX :- `datatype (*varname)[size];`
③ Declaration :- `int *ptr[5];`	③ Declaration `int (*ptr)[5]=NULL;`

Program for ARRAY of POINTERS :-

```
#include<iostream>
using namespace std;
int main()
{
    int arr[] ={1,2,3};
    int i, *ptr[size];
```

```
for (i=0; i<size ; i++)
{
    ptr[i] = &arr[i];
}

for (i=0 ; i<size ; i++)
{
    cout << " vallue of array:- "<< i << "is" << *ptr[i];
}
}
```

Output

value of arr[0] = 1
value of arr[1] = 2
value of arr[2] = 3

Program to for pointer to an array:-

```
#include<iostream>
using namespace std;
int main()
{
    int (*a)[5];
    int b[5] = {1,2,3,4,5};
    int i=0;
    a = &b;
    for (i=0; i<5; i++)
    {
        cout << "\n" << *(*a+i);
    }
```

```
return 0;
}
```

Output

1
2
3
4
5

Q3. Write a program in C which implements tower of Hanoi using recursion?

Ans 3

```
#include<stdio.h>
void towers(int, char, char, char);
int main()
{
    int num;
    printf("Enter no. of disks:");
    scanf("%d", &num);
    printf("Sequence of moves involved in TOH are:\n");
    towers(num, 'A', 'B', 'C');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if(num==1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
```

```
ing namespace std;
// returns true if str1[] is a subsequence of str2[]. m
// is length of str1 and n is length of str2.
bool isSubsequence (char str1[], char str2[], int m, int n)
{
    int j=0; // for index of str1
    for (i=0; i<n && j<m; i++)
    {
        if (str1[j] == str2[i])
        {
            j++;
        }
    }
    return (j==m);
}
int main()
{
    char str1[] = "gksrek";
    char str2[] = "geeksforgeeks";
    int m = strlen(str1);
    int n = strlen(str2);
    isSubsequence(str1, str2, m, n) ? cout << "Yes":
                                      cout << "NO";
    return 0;
}
```

```
    return;
}
towers(num-1, frompeg, topeg, auxpeg);
printf("\n Move disk %d from peg %c to
        peg %c", num, frompeg, topeg);
towers(num-1, auxpeg, topeg, frompeg);
}
```

Output:-

```
Enter no. of disks : 3
Sequence of moves involved in TOH are :

move disk 1 from peg A to peg C
move disk 2 from peg A to peg B
move disk 1 from peg C to peg B
move disk 3 from peg A to peg C
Move disk 1 from peg B to peg A
Move disk 2 from peg B to peg C
move disk 1 from peg A to peg C
```

Q4) WAP to search a key string in a array of strings, if a key of string is found then return its position and then replace that key string by any string using pointers.

Ans4)

```
#include<iostream>
#include<cstring>
```

Ques5) Describe following manipulators:

a) setw()

- Sets the field width to be used on output operations.
- behaves as if member width were called with n arguments on the stream on which it is inserted / extracted as a manipulator.
- This manipulator is declared in header <iomanip>

b) setiosflags

- Sets the format flags specified by parameter mask.
- behaves as if member setf were called with mask as argument on the stream on which it is inserted / extracted as a manipulator.

c) setprecision

- Sets the decimal precision to be used to format floating-point values on output operations.
- Return type is unspecified.

d) setfill()

Sets c as the stream's fill character.

behaves as if member fill were called with c as an argument on the stream on which it is inserted as a manipulator.

e) Resetiosflags()

- Unsets the format flags specified by parameter mask.
- behaves as if member unsetf were called with mask as argument on the stream on which it is inserted/extracted as a manipulator

Q6) What is the difference between static & dynamic memory allocation? what are inline functions? Explain.

Ans6)

STATIC MEMORY ALLOCATION	DYNAMIC MEMORY ALLOCATION
(1) Allocated memory is fixed. Once the memory is allocated, it cannot be changed. Memory can't be increased or decreased.	Depending upon insertions and deletions of data elements, the memory can grow or shrink.

	It is not possible to resize after initial allocation.	The memory can be minimized or maximized accordingly.
(3)	It is easy to implement.	It is complex to implement.
(4)	Allocation execution is faster than dynamic memory allocation.	Allocation execution is slower.
(5)	In this case, cannot reuse unused memory	It allows reusing of memory.

The inline functions are used to increase execution time of a program. functions can be instructed to compiles to make them inline so that compiles can replace these functions definitions wherever those are being called.

Compiler replaces the definition of inline f^n's at compile time instead of referring function definition at run time.

to make a function inline, start its definition with the keyword, "inline."

SECTION-C

Q 7) Explain how base class member func can be invoked on a derived class if the derived class also has a member function with the same name. How are stream operator overloaded in CPP?

Ans 7) C++ is able to input and output built-in data types using the stream extraction operator >> and the stream insertion operator <<. The stream insertion and ~~deletion~~ extraction operators can also be overloaded to perform input and output for user-defined-types like an object.

example:-

```
#include <iostream>
using namespace std;

class dist{
private:
    int feet; // 0 to infinite
    int inches; // 0 to 12
public:
    dist(){
        feet=0;
        inches=0;
```

```
dist(int f, int i) {
    feet = f;
    inches = i;
}
friend ostream &operator << (ostream &output,
    const dist &D) {
    output << "F:" << D.feet << "I:" << D.inches;
    return output;
}
friend istream &operator >> (istream &input,
        dist &D) {
    input >> D.feet >> D.inches;
    return input;
}
};

int main()
{
    dist D1(11,10), D2(5,10), D3;
    cout << "Enter value of object:" << D3 << endl;
    cin >> D3;
    cout << "1st Dist: " << D1 << endl;
    cout << "2nd Dist: " << D2 << endl;
    cout << "3rd Dist: " << D3 << endl;
```

```
    return 0;
}
```

O/P $./a.out

```
Enter the value of object:
70
10
first dist : f : 11 I : 10
Second dist : f : 5 I : 11
Third dist : f : 70 I : 10
```

Q8) What is inheritance? Explain its different types. What are the advantages of scope resolution and referencing?

Ans) Inheritance is one of the key features of OOP. Inheritance provides mechanism that allows a class to inherit properties of another class. When a class extends another class, it inherits all non-private members including fields and methods.

Purpose of inheritance:

1. To promote code reuse.
2. To use polymorphism.

```
    inches = 0;
}
```

Types of inheritance :-

(1) Single Inheritance :- where subclasses inherit the features of one superclass. A class acquires the properties of another class.

(2) Multiple Inheritance : where one class can have more than one superclass and inherit features from all parent classes. Multiple inheritance is not supported in java because :-
1) To remove ambiguity
2) To provide more maintainable and clear design.

(3) Multilevel inheritance :- where a subclass is inherited from another subclass. It is not uncommon that a class is derived from another derived class.

(4) Hierarchical Inheritance :- where one class serves as a superclass (base class) for more than one subclass.

(5) Hybrid Inheritance :- a mix of two or more of the above types of inheritance.

advantages of scope resolution :-

→ helps to identify and specify the context to which an identifier refers, particularly by specifying a namespace. In many languages the scope resolution operator is written as "::".

advantages of referencing :-

(1) Safer :- Since references must be initialised, wild references like wild pointers are unlikely to exist.

(2) It is still possible to have references that don't refer to a valid location.

(3) Easier to use

(4) They can be used like normal variables '&' operator is needed only at time of declaration.

When are c++ copy constructs, assignment & destructors, respectively, invoked.

When is copy constructor called?

In c++ a copy constructor may be called in following cases:

1. When an object of Class is returned by value.
2. When a object of Class is passed by value as an argument.
3. When an object is constructed based on another object of same Class.
4. When compiler generates a temporary object.

Section - B

MAY-2017.

(3) What is inline function? How do we make inline function in C++ classes?

Ans Inline function reduces the function call overhead. Inline function is a function that is expanded in line when it is called when the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call.

Syntax :-

```
inline return-type function-name(parameters)
{
    // function code.
}
```

Inline function in C++ classes:-

```
#include<iostream>
using namespace std;
class operation
{
    int a, b, sum;
  public:
    void get();
    void sum();
};
inline void operation::get()
{   cout<<"Enter 2 values for a and b:";
    cin>>a>>b;
}
inline void operation::sum()
{
    sum = a+b;
    cout<<"Sum is:"<<sum;
}
```

```
int main()
{ cout<<"\nprogram using inline function\n";
  operation obj;
  obj.get();
  obj.sum();
  return 0;
}
```

② Can we pass a class object as argument of function? Explain with the help of a example.

Ans The object of a class can be passed as argument of the member function as well as non member function either by value or by reference.

example:

```
#include<iostream>
class weight
{ int kg, g;
  public:
    void getdata();
    void putdata();
    void sumweight(weight, weight);
};
void weight::getdata()
{ cout<<"\n enter value for kg and g: ";
  cin>>kg>>g;
}
void weight::putdata()
{ cout<<"\n kg = "<<kg<<"kg"<<"and "<<g<<"g";
  cout<<" gms";
}
void weight::sumweight(weight w1, weight w2)
{ g=w1.g+w2.g;
  kg=g/1000;
  g=g%1000;
  kg=kg+w1.kg+w2.kg;
}
int main()
{ weight w1, w2, w3;
  cout<<"\n enter weight in kg and g \n";
  cout<<"\n enter for w1";
  w1.getdata();
  cout<<"\n enter for w2";
  w2.getdata();
  w3.sumweight(w1,w2);
  cout<<"\n weight 1";
  w1.putdata();
  cout<<"\n weight 2";
  w2.putdata();
  cout<<"\n Total weight";
  w3.putdata;
  return 0;
}
```

④ output:-

i=6 j=16

steps:

1. int main()
2. i=6 and j=9
3. then function call test(i,&j) where i is call by value and j is call by reference.

(4) Inside the degination

a = a*a;
and *b = *b * *b;

So as i is by value the value will not change in main

so i = 6.

but as j = is call by reference it will become

$*b = (4) * (4)$
$= 16$

(5) back to int main()

printing values of i and j

it will

i = 6 j = 16

exit

(6)

call by values	call by reference
1) In this method, a copy of the variable is passed	1) In this method, a variable itself is passed.
2) changes made in copy of variable never modify the value of variable outside the function	2) change in the variable also affects the value of the variable outside the function.
3) Actual and formal arguments will be created in different memory location	3) Actual and formal arguments will be created in the same memory location.

5. generic function is different from normal function with suitable function.

- A generic function is a function that is declared with type parameters. When called, actual types are used instead of the type parameters

Normal function do not have any such functionality. Normal function. A square integrable function is said to be normal it. However the normal distribution function is also sometimes called the normal function.

Section-A

Q1 (b) Class is different from object.

Class- A class is a blueprint from which you create the instance, i.e. objects.
A class is used to bind data as well as methods together as a single unit.
Class have logical existence.
A class doesn't take any memory space when a programmer create one.
The class has to be declare only once.

Object- An object is the instance of the class, which helps the programmer to use variable and methods from inside the class.
An object act as variable of the class.
Object have a physical existence.
An object takes memory when a programmer create one.
Object can be declared several times depending on the requirement.

(c) Constructor - Constructor is a special method that is automatically called when an object of a class is created.

Destructor - Destructor is a special member function that is executed automatically when an objects is destroyed that has been created by the constructor. It is used to deallocate memory that has been allocated for the object by the constructor.

Static memory Allocation	Dynamic memory allocation
(a) In this case, variable get allocated permently	a) Variables get allocated only if your program unit gets active
b) Allocation is done before program execution	b) Allocation is done during program execution
c) There is no memory reusability	
d) less efficient	c) more efficient

Inheritance support the concept of 'reusability' common that when we want to create a new class and there is already a class that include some of the code that we want, we can derive our class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Friend function - Similar to friend class.
This function can access the private and protected members of another class. A global function can also be declared as friend.

Friend class - Is a class that can access the private & protected members of a class in which it is declared as friend. This needed when we want to allow a particular class to access the private & protected member of a class.

h) Polymorphism achieved at compile time and runtime.
Runtime- This type of polymorphism is achieved by function overriding. other hand occur when a derived class has a definition of one of them member function of the base class. The base function is said to be overridden.

Compile time - when the source code is compile the function invoked are bonded to the compiler during compile time as to invoke which function depending upon the type number of arguments. Such a phenomenon is reffered to early binding.

i) what are the different modes in which C++ file open file in binary mode. One file for input - ifstream open file for output - ofstream.

ii) Briefly define 'this' pointer.
Keyword 'this' represent the current object and passed as a hidden argument to all member function. To this pointer is a constant pointer that holds the memory address of the current object.

d) what is the output of the code.

```
#include <iostream>
using namespace std; int x;
int main()
{ int x=5;
  x=10;
  { x=::x;
    cout<<::x+x+::x+x;
  }
  return 0;
}
```

5+1 output = 0

e) (m+1) updated
(m+1) condition checked.

Section - C

a) How multilevel inheritance is different from multiple inheritance

Multilevel inheritance - not only you can derive a class from the base class but you can also derive a class from the derived class.

```
#include <iostream>
using namespace std;
class A
{   public:
      void display()
    { cout<<" Base class content"; }
};
class B : public A
{   };
class C : public B
{   };
int main()
{   C obj;
    obj.display()
    return 0;
}
```

Multiple inheritance - A class can derived from more than one parents.

```
#include <iostream>
using namespace std;
class Mammal
{ public: mammal()
    { cout<< "mammal can give direct birth"; }
};
```

```
class widgedanimalea {
    public: widgedanimal()
    { cout<<" widged animal can flop"; }
};
class bat : public mammal, public widgedanimal
{   };
int main()
{   bat b;
    b.getsom();
}
```

b) What ambiguity arisises in multiple inheritance and how it resove.

This problem arises sometimes in multiple inheritance when function with same name appear in more than on base cla

```
class M
{ public: void display()
  { cout<<" class M"; }
};
class N
{ public: void display()
  { cout<<" class N"; }
};
class P: Public M, public N
{ public: void display()
  { cout<<" class P";
    M::display(); }
};
void main()
{ P p;
  p.display;
}
```

we can solve this problem by defining a named instance within the derived class using scope resolution operator with the function

9)

```
#include<iostream>
class complex
{ float x, y;
  public: complex()
    { }
  complex(float real, float imag)
  { x = real; y = imag; }
  friend complex operator+(complex a, complex b)
  { complex temp;
    temp.x = a.x + b.x;
    temp.y = a.y + b.y;
    return (temp);
  }
  void display()
  { cout<< x<< "j"<<y; }
};
void main()
{ complex c1, c2, c3;
  c1 = complex(2.5, 3.5);
  c2 = complex(1.6, 2.7);
  c3 = c1 + c2
  cout<<"c1 = "<< c1.display();
  cout<<"c2 = "<< c2.display();
  cout<<"c3 = "<< c3.display();
}
```

(7)

```
#include<iostream>
class shape
{   int l, b, ar, pre;
    void area()
    {   ar = l*l;
        cout<<"sq. area = "<< ar;
    }
```

```
void ar(l, b)
{   ar = l * b;
    cout<<"\n area of rect.="<<ar;
}
void perimeter( )
{  pre = 4 * l;
    cout<<"\n perimeter of sq.="<<pre;
}
void perimeter(l, b)
{   pre = 2 * l * b;
    cout<<"\n perimeter of rectangle="<<pre;
}
};
int main( )
{   shape obj;
}
```

DEC-2017

Roll No. | | | | | | | | | | |

Total No. of Questions: 09 Total No. of Pages: 02

B. Tech. (Electronics Engineering/3D Animation & Graphics/Electronics & Computer Engg/CSE/ECE/ETE/IT) (Sem. 3)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code: BTCS-305

Paper ID: A1129

Time: 3 Hrs. Max. Marks: 60

INSTRUCTIONS TO CANDIDATES:

1. Section A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. Section B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. Section C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION A

1. Write Briefly

a) Differentiate between Call by value and call by reference.

b) What is Data hiding? How it is achieved in C++.

c) What is a destructor? How it is defined.

d) What do you mean by function overloading? Give example.

e) What is the use of this pointer? Discuss.

f) Define Class. How object and class are related to each other? Give example.

g) Define a class template. What are its advantages?

h) Illustrate with an example, how the endl and setw manipulator works.

i) Discuss the concept of abstract class with example.

j) What is static storage class? What are its characteristics?

SECTION B

2. What are the implications of public, protected and private visibility modes?

3. What is a virtual function? Explain need of virtual function through a suitable example.

4. Explain the use of new and delete operators in C++.

5. Discuss pointers related problems with examples.

6. How is the exception handling performed in C++? Write a program that throws an arithmetic exception as and when a number input is greater than 9999.

SECTION C

7. What do you mean by Inheritance? Explain various types of inheritance with the help of suitable examples.

8. What is a constructor? Write its main characteristics. Explain various types of constructor with examples.

9. Write a program in C++ to input and display complex number. Also perform add and subtract operations on the complex numbers illustrating the concept of operator overloading.

1° Dec-17

by value & call

ver 1: (a)

Call by reference copies address of arguments into functions.

Change made to parameter affects arguments.

Actual & formal arguments created in same memory location.

iv) eg°.

```
void swap(int *x, int*y);
  void main()
{ int a= 100, b=200;
  clrscr();
  swap (&a, &b);
 cout<<"value of a:";
   cin>>a
; cout<<" value of b:";
   getch();
}
 void swap(int *x, int *y)
   { int temp;
     temp= *x;
     *x= *y;
     *y = temp;
     }
```

```
matrix :: ~matrix()
{
  for(i=0; i< d1; i++)
    delete p[i];
    delete p;
}
```

This is required because when the pointer to object go out of scope, a destructor is not called implicitly.

```
#include <iostream.h>
  int count=0;
  class test
{ public: test()
  {
   count++;
   cout<<" object number" << count<< "Created";
  }
  ~ test()
  {cout <<"object number "<< count<<"destroyed";
   count--;
  }};
int main()
{ cout << " Inside main block ";
  cout << " Creating 1st object T1 ";
     test T1; // default Constructor called
   { // Block 1
    cout<<" Inside block-1";
    cout<<" creating 2 more objects T2, T3 ";
     test T2, T3; // constructor called.
     Cout<<" leaving Block-1 ";
      }
  Cout<<" Back Inside main block";
     return 0;
}
```

What is the use of this pointer? Rules.
this pointer in C++ is used to represent the address of an object inside a member function.
ore eg, consider an object obj calling one of its member function say method() as obj.method().
Then, this pointer will hold the address of object obj inside the member function method().

) Define class. How object & class are related to each other? Give example.
class is a way to bind the data & its associated functions together. It allows the data to be hidden, if necessary from external use.
When defining a class we are defining a new abstract class type that can be treated like any other built in-data type.
syntax: class classname
keyword {
access specifier : data member;
member function;
};
Once a class has been created, we can create variable of that type by using classname called objects. An object is an instance of a class.
syntax for creating objects: classname objectname;
eg item i;
objects can also be created by placing their name immediately after closing brace of class definations.

w: This manipulator changes the width of the next input/output field. When used in an expression setw(n), sets the width parameter of the stream out or in to exactly n.

```
eg: #include <iostream>
    #include <iomanip>
    int main()
    {
    std::cout << "no setw:" << 42 << "\n";
    }
```

i) Discuss the concept of abstract class with eg?

An Abstract class is one which is not used to create objects. It is designed only to act as base class (to be inherited by other class). An Abstract class is the one which has atleast one pure virtual function.

```
eg: class Base // Abstract class
    {
      int x;
    public: virtual void fun()=0; // pure virtual function
      int getx()
      {return x;}
    };
    class Derived : public Base
    { int y;
    public: void fun() // function defined in derived class.
      { cout << "Hello";
      }
    };
    void main()
```

olated data member can only be accessed by friend function with in the class.

Visibility of Inherited member:-

When a base class is privately Inherited:-

Public members of the Base class become private members of derived class.

i) Private members of the base class are not Inherited.

ii) Protected members of the base class become protected in derived class.

When a base class is protectly Inherited:-

(i) Public members of Base class become protected in derived class.

ii) Private members of Base class are not Inherited

iii) Protected members of the base class are protected Inherited.

eg- class Base

```
{
private: int x;
protected: int y;
public: int z;
base()
{
x=1;
y=2;
z=3;
}
};
int main()
{
Base obj;
}
```

Ans 3

What is a virtual function? Explain ru
virtual function through a suitable
When we use same function name in both
base & derived class, the function in Ba
class is declared as virtual by using
keyword virtual preceding its normal
declaration. When a function is made
determine which function to use at run-
time based on the type of object point
to by the base pointer, rather than the
type of pointer.
A virtual function is a function which is
defined in base class & is re-defined
(over-ridden) in derived class.
Rules for virtual function :-
(i) They must be declared in public section
of class.
ii) They cannot be static & also cannot be
friend of another class.
iii) They are accessed by object pointer.
iv) They are defined in base class & over
-ridden in derived class.
example :

```
class Base
{ public : virtual void show()
           { cout << "Base Class";
           }
};
```

```
class Derived : public Base
{ public : void show()
           { cout << "Derived Class";
           }
};
int main()
{ Base *b; // Base class pointer
  Derived d;
  b = &d;
  b -> show(); // Late Binding occurs
}
```

output : Derived Class

Now, Base class pointer point to derived class object ; i.e. Derived class function is called.

Ans 4

Explain the use of new and delete operators in C++?

C++ supports dynamic allocation and deallo c
-ation of objects using the new & delete ope
-rators. This operators allocate memory
for objects from a pool called the free store.
The new operator call the special function
operator new, & the delete operator calls
the special function operator delete.
The new operator request for the memory
allocation in heap. If the sufficient
memory is available, it Initialize the

Section-C

7. What do you mean by Inheritance? Explain various types of Inheritance with the help of suitable example?

e mechanism of deriving a new class from an d class is called Inheritance (or derivation) The old class is called base class & the new class is called derived class (or subclass). The derived class Inherits some or all of the traits from the base class. A class can inherit proper ties from more than one base class & more than one class can Inherit properties from a single base class.

Syntax:

class derived-classname : visibilitymode base class name

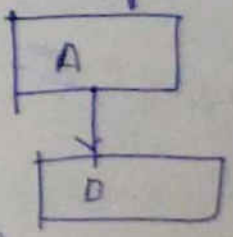
{

// Body of derived class

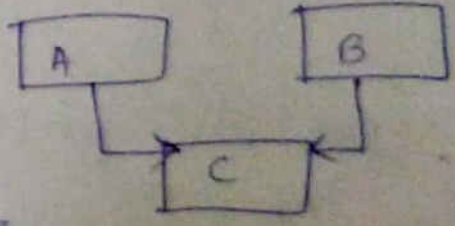
};

Types of Inheritance are as follows:

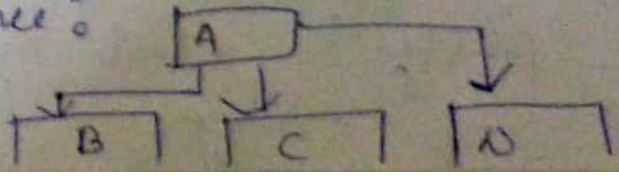
1) single Inheritance:



2) Multiple Inheritance:



3) Hierarchical Inheritance:

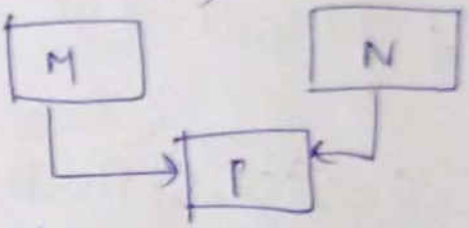


Multiple Inheritance - A class can inherit the attributes of 2 or more classes. This is known as Multiple Inheritance.

Syntax:

```
class Derived : visibility Base1, visibility Base2
{ // Body of derived class
};
```

eg:




```
class M
{ public : int m;
  void getM()
  {
  cout << "Enter m";
  cin >> m;
  }
};
class N
{ public : int n;
  void getN()
  {
  cout << "Enter n";
  cin >> n;
  }
};
class P : public M, public N
{ public : int p;
  void mul()
  { getM();
  getN();
  p = m * n;
  cout << "Mul" << p;
  }
};
void main()
{ P p;
  p.getM();
  p.getN();
  p.mul();
}
```

```
}
return 0;
}
```

xample of delete operator:

```
# include <iostream>
using name space std;
int main()
{
  int size, i;
  int *ptr;
  cout<<"\n\t Enter size of array:";
  cin>>size;
  ptr = new int[size];
  for (i=0; i<5; i++)
  {
    cout<<"\n enter any no:";
    cin>>ptr[i];
  }
  for (i=0; i<5; i++)
    cout<<ptr[i]<<" ";
  delete []ptr;
}
```

Ques 5:
Discuss pointer related problems with examples?

Pointer related problems are as follows:

1) Unintialized pointers might cause segmentation fault.
2) Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
3) Pointers are slower than normal variables.

iv) If pointers are updated with incorrect value, it might lead to memory corruption.

Basically, pointer bugs are difficult to debug. Its programmers responsibility to use pointers effectively and correctly.

Syntax:

```
datatype * var_name;
int * ptr; // ptr can point to an address.
```

```
# include <iostream>
using namespace std;
void geeks()
{ int var=20;
  int * ptr;
  ptr= &var;
cout << "value at ptr:";
cout << "value at var:";
cout << "value at *ptr:";
}
int main()
{
  geeks();
}
```

output:- value at ptr = 0x7ffc b9e9ea4c
value at var = 20.
value at *ptr = 20.

Ques 6:

How is the exception Handling performed in C++? Write a prog that throws an arithmetic exception if when a number Input is greater than 9999.

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arise while a program is running such as an attempt to divide by zero. Exception provide a way to transfer control from one part of a program to another. C++ Exception Handling is built upon four keyword:

try, catch, finally & throw.

- Try - A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch block.
- Catch - A program catches an exception with an exception handler at the place in a program where you want to handle the program. The catch indicate the catching of an exception.
- throw - A program throws an exception when a program show up. This is done using a throw keyword.

program:

```
# include <iostream>
using namespace std;
void main()
```

```
{
  int a;
cout << "Enter a";
  cin >> a;
  try
   {
    if (a > 9999)
    {cout << " Input is greater then 9999";
      throw(a);
     }
  else
  {
    cout << a;
    }
  }
catch (int x)
  {cout << "exception caught: no greater
                then 9999";
    }
    cout << "end";
    }
```

MAY-2018

Total No. of Questions : 09

B.Tech. (Electronics Engg./3D Animation & Graphics) (2012 Onwards) / B.Tech. (CSE/ECE/Electronics & Computer Engg./ETE/IT) (2011 Onwards)

(Sem. – 3)

OBJECT ORIENTED PROGRAMMING USING C++

M Code: 56595

Subject Code: BTCS-305

Paper ID: [A1129]

Time : 3 Hrs. Max. Marks: 60

INSTRUCTIONS TO CANDIDATES:

1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION A

1. a) Differentiate between a local and a static object.

   b) What is the purpose of defining a Destructor function?

   c) Explain briefly what is Exception Handling?

   d) What are the properties of a static data member?

   e) What is the use of Scope resolution operator in C++?

   f) What is an Abstract class?

   g) What is Visibility mode? What are the different inheritance Visibility modes supported by C++?

   h) What is the use of this keyword?

   i) What are C++ streams?

   j) Explain nested class with the help of an example.

SECTION B

2. What is a class? What is the relation between an object and a class? Write a program which shows how to define a class, how to access member functions and how to create and access objects in C++.
3. Explain with examples the different (Variable) storage classes used in C++.
4. Write a program to get character input from the user and store those characters in a file.
5. With the help of a suitable example, show how to access records randomly in a file.
6. Explain the concept of Virtual and Pure Virtual Functions with the help of examples.

SECTION C

7. a) What is inheritance? Explain with example how to inherit a class in C++.

   b) What is Dynamic Memory Allocation? Explain with the help of an example how to create and destroy objects dynamically.

8. Create a class whose object represents a complex number (A complex number contains a real part and an imaginary part). Write a program so that it is possible to add two objects of this class and store the result in third object.
9. What is a Template? Explain with the help of an example how to create a Function Template and a Class Template.

Differentiate between a local and a static object.

:-

Local object	Static object
1. When we define a object inside a function it is known as local object	1. An object become static when static keyword is used in its declaration.
2. When program execute create object on stack means storage is allocated on stack.	2. Static objects are allocated storage in static storage area.
3. Local objects is created each time its declaration is encountered in the execution of program	3. Static object are initialized only once and live until the program terminates.

For eg:

```
class Test
{
   Public:
   Test()
  { cout << "constructor";
  };
void myfunc()
{ static Test obj;
 }
int main()
{ Test T1.
  T1. myfunc();
  return 0;
}
```

For eg! -

```
class Test
{ Public:
  Test()
  { cout << "constructor";
  }
  ~Test()
  { cout << "Destructor";
  };
  void myfunc()
  { static Test obj;
  }
  int main()
  { cout << "starts";
    myfunc();
    cout << "terminates";
    return 0;
  }
```

(b) What is purpose of defining a destructor function?

Soln:→ A destructor is a special membu function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime

Destructor have same name as the class preceded by ~
Destructor don't take any argument and don't return anything.
A destructor function is called automatically when object goes out of scope:
1. the function ends 2. the program ends 3. a delete operator is called

For eg:-

```
class String
{ private
  char *s;
  int size;
  Public:
  String (char *);
  ~String ( )
  };
  String :: String (char *c)
  {
     size = strlen(c);
     s = new char [size+1];
     strcpy (s,c);
  }
```

(c) Explain breifly what is exception handling?

Sol^n:- An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running. C++ exception handling is built upon three keywords: try, catch and throw

- throw:- A program throws an exception when problem shows up. Done using throw keyword.
- catch:- A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
- Try:- Try block identifies a block of code for which particular exceptions will be activated.

For example:-

```
#include <iostream>
Using namespace std;
double division (int a, int b)
{
```

```
    b==0){
    throw "Division by zero condition";
  return (a/b);
}
int main()
{
   int x=50;
   int y=0;
   double z=0;
  try {
  z=division(x,y);
  cout<< z<< endl;
  }
  catch (const char *msg) {
   cout<< msg<< endl;
  }
  return 0;
  }
```

(d) What are the properties of a static data member?

Soln:- Static data member has the following properties:-

1. It is initialized by zero when first object of class is created
2. Only one copy of static data member is created for the entire class and all object share the same copy.
3. Its scope is within class but its lifetime entire program
4. They are used to store the value that are common for all the objects.

Static can be declared as:-

```
Static - datatype variablename;
```

for eg:-

```
class Item
{
    int simple;
    static int count;
    PUBLIC:
    void inc();
    Void print();
};
int Item::count=0;
```

```
void item :: incr()
{
    counter ++;
    simple ++;
}
void item :: print()
{ cout<<"counter is " << counter;
}
void main()
{ item I1;
  I1.incr();
  I1.print();
  item I2;
  I2.incr();
  I2.print();
}
```

(e) What is the use scope resolution operator in C++?

Sol":- Scope resolution operator (::) in C++ programming language is used to define a function outside a class or when we want to use a global variable but also has a local variable with the same name.

for eg:-
```
class cgc
{
  public:
  void output();
};
void cgc :: output()
{
  cout<<"Chandigarh";
}
int main()
{
  cgc c1;
  c1.output()
  return 0;
}
```

What is an Abstract Class?

Abstract class is a class which contains atleast one pure virtual function in it. Abstract classes are used to provide an Interface for its sub class.

Characteristics.

1. Abstract classes cannot be instance, but pointers and references of Abstract class type can be created
2. Abstract class can have normal functions and variables along with a pure virtual function.

For eg:-

```
class Base
{
    Public:
    Virtual void show() ;
};
class derived : public Base
{
    Public:
    void show()
    { cout<<"Virtual";
    }
};
int main()
{
    Base B1;
    Base *b;
    derived d;
      b= &d
      b->show();
}
```

(g) What is visibility mode? Whats are different inheritance visibility modes supported by C++?

Sol^n:- Depending on Acess modifier used while inheritance, the availability of class member of super class class in the sub class changes. It can either be private, protected or public

1. PUBLIC inheritance

This is the most used inheritance mode. In this the public member of base class becomes protected members of sub class and public become public.

2. Private inheritance:-

In private mode, the protected and public members of Base class become private member of derived class.

3. Protected inheritance:-

In protected mode, the public and protected members of Base class become protected members of sub class

TABLE:- Derived class →

Base class	Public	Private	Protected
Private	Not inherited →		
Protected	Protected	Private	Protected
Public	Public	Private	Public

h) What is the use of this pointer?

Sol^n:- The 'this' pointer is passed as a hidden argument to all non static member function calls and is available as a local variable within the body of all non static functions. 'this' pointer is not available in static member functions as static member functions can be called without any object

For eg:

```
class Test
{
  private:
  int x;
  public:
  void Set X (int x)
    { this → x = X;
  }
  void print( )
  { cout << "x = " << x << endl;
  } };
int main( )
```

```
T1;
int x=20;
T1.setX(x);
T1.print();
return 0;
}
```

Q10-) What are C++ streams?

Soln:- In C++ there are number of stream classes for defining various streams related with files and for doing input-output operations. All these classes are defined in the file iostream.h.

1. Ios class is topmost class in the stream classes. It is the base class for istream, ostream and streambuf class.
2. istream and ostream serves the base classes for iostream class. The class istream is used for input and ostream for output.
3. Class ios is indirectly inherited to iostream class using istream and ostream. To avoid the duplicity of data and member function, it is declared as virtual base class.

For eg:-

```
#include <iostream>
using namespace std;
int main()
{
  char x;
  cout<<"Tanvam";
}
```

Q11-j) Explain nested class with the help of an example.

Soln:- A nested class is a class which is declared in another enclosing class. A nested class is a member as such has the same access rights as any other member. The member of an enclosing class have no special acess to members of a nested class

For eg:-

```
# include <iostream>
using name space std;
class close
```

```
{ private :
  int x ;
  class nested
  {
    int y ;
  void nested (Close *e)
  {
    cout << e → x ;
  } } ; } ;
int main()
{
  }
```

Section - B

(2) What is class ? Relation between object and class . Program to define a class . how to access member functions and how to create and access objects in C++.

Soln:- Class :- It is a user-defined data type, which holds its own data members and member functions , which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

An object is an instance of a class. When a class is defined, no memory is allocated but when it is created memory is allocated.

Relation :-

A class define the properties and behaviour for the objects represented by the abstraction. Abstraction is a property of object . It denotes the essential properties and behaviour of an object . It hides data and code . A class thus denotes a category of objects and act as a blueprint for creating such objects. Generally, an object is an instance of a class.

of a class:-

class is defined in C++ using keyword class. The body of class is defined inside the curly brackets and terminate by semicolon at the end.

For eg:-

```
class test
{
  Public :
  int a;
  Void putdata() { a=0; }
};
int main ()
{
  test t1.
  t1. put data();
}
```

Accessing data members and member functions:-

The data members and member function of class can be accessed using the dot ('.') operator with object.

The public data members are also accessed in the same way when however the private data members are not allowed directly by the object. Accessing a data member depends upon the access control of that data member.

For eg:-

```
class AB
{
  Public:
  Static AB ;
  Void print name()
{ cout << "AB=" << AB;
 } };
int main ()
{ AB a;
```

```
    a.printname();
    return 0;
    }
```

One more program to access the objects and datamembers :-

```
class game
{
    Public:
    int Id;
    Game()
    { cout << "constructor" << endl;
      id = 1;
    }
    Game(int x);
    { cout << "PC" << endl;
      id = x;
    };
    int main()
    {
      Game g1;
      cout << " id is" << g1.id << endl;
      Game g2
      cout << "id is" << g2.id << endl;
      return 0;
    }
```

Q3:→ Explain with examples the different (Variable) storage classes Used in C++?

Sol^n→ Storage classes are used to describe the features of a variable. These features basically include the scope, visibility and life time which help us to trace the existence during the runtime of a program.
These specifiers precede the type that they modify.
There are following storage classes, which can be used in C++ program...

- Register
- Static
- Extern
- Mutable

⇒ The auto storage class :-
The auto storage

Register
- Static
- Extern
- Mutable.

⇒ The auto storage class:-
The auto storage class is the default storage class for all local variables.
Foreg:-

```
{
    int mount;
    auto int mount;
}
```

The example above defines two variable with the same storage class, auto can be used within functions i.e. local variables

⇒ The register Storage class:-
The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location)
Foreg:-

```
{
    register int miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it might be stored in register depending on hardware and implementation restrictions.

⇒ The Static storage class:-
The static storage class instructs the compiler to keep a local variable in existence during the life-time of program instead of creating and destroying it

each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C++, when static is used on a class data member, it causes only one copy of that member to be shared by all objects of its class

```
# include <iostream>
void func (void);
static int count = 10;
main()
{
    while (count--)
    {
        func();
    }
    return 0;
}
void func (void)
{ static int i = 5;
  i++;
  std:: cout << " i is" << i;
  std:: cout << " and count is" << count << std::endl;
```

⇒ The extern storage class :-

The extern storage class is used to give refrence of a global variable that is visible to ALL the program files. When you use "extern" the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will be used in other files also, then extern will be used in another file to give refrence of defined variable or function. Just for understanding

is used to declare a global variable.
extern modifier is most commonly used when there are two or more files sharing the same global variable or function are explained below

First file: main.cpp

```
#include<iostream>
int count;
extern void write extern();
main()
{
   count=5;
   write extern();
}
```

Second file: support.cpp

```
#include <iostream>
extern int count
void write extern(void)
{
   std::cout <<" count is"<< count <<std::endl;
}
```

⇒ The Mutable Storage Class

The mutable storage class applies only to class objects, which are discussed later in this tutorial. It allows a member of an object to override const member function. That is, a mutable member can be modified by a const member function

```
#include <iostream>
using namespace std;
class X
{
   public:
      X(int a, int b)
      {
         m=a;
         n=b;
      }
      int m;
      mutable int n;
};
int main()
{
   const X obj(5,2);
```

each time of

```
cout << "m:" << obj.m << "n:" << obj.n << endl;
  obj.n = 8;
  cout << "m:" << obj.m << "n:" << obj.n << endl;
  return 0;
}
```

Q:- WAP to get character input from the user and store those characters in a file?

Soln:-

```
#include <iostream>
#include <fstream>
using namespace std;

ifstream::pos_type size;
char * memblock;
int main()
{
  ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
  if (file.is_open())
  {
    size = file.tellg();
    memblock = new char [size];
    file.seekg (0, ios::beg);
    file.read (memblock, size);
    file.close();

  cout << "the complete file content is in memory";

  delete [] memblock;
}
else cout << "Unable to open file";
  return 0;
}
```

⇒

...sr and store

...n the help of suitable example, show how to access records randomly in a file.

```
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
class student
{
        int rollno;
        char name[20];
        char branch[3];
        float marks;
        char grade;
        public:
            void getdata()
            {
               cout<<"Rollno:";
               cin>>Rollno.;
               cout<<"name:"<<name<<endl;
               cout<<"Branch:"<<Branch<<endl;
               cout<<"marks"<<marks<<endl;
            if(marks>=75)
            {
               grade = 'A';
            }
            else if(marks>=60)
            {
               grade = 'B'
            }
            else if(marks>=50)
            {
               grade = 'C';
            }
            else if(marks>=40)
            {
               grade = 'D';
            }
            else if~~(marks~~
            {
               grade = 'F';
            }
}
```

```
Void put data ()
{
    cout<<" Roll No." << roll no. <<"\t Name :"<< name <<"\n";
    cout << "marks:" << marks << "\t grade:" << grade <<"\n";
}
int getrno()
{ return roll no.;
}
Void modify();
Void Student:: modify()
{ cout<<" Roll NO:"<<rollno<<"\n";
  cout<<" Name :" << name <<"\t Branch:" << branch << "\t marks" <<
                                                    marks<<"\n";
  cout << "Enter new details:\n";
  char nam[20]= " ", br[3]= " ";
  float mks;
  cout<< " New name : (enter '.' to retain old one):";
  cin >> name;
  cout << " New Branch : (Press '.' to return old one):";
  cin>> br;
  cout << " New marks : (Press -1 to retain old one):";
  cin>> mks;
  if (strcmp (nam, ".")!=0)
  {
      strcpy (name, nam);
  }
  if (strcmp (br, ".")!=0)
  {
      strcpy (branch, br);
  }
  if (mks != -1)
  {
     marks = mks
     if (marks >=75)
     {
        grade = 'A';
     }
     else if (marks >=60)
     {
        } grade = 'B';
```

```
(marks >=50)
grade = 'C';
else if (marks >=40)
grade = 'D';
else
grade = 'F';
```

```
    if (marks >= 50)
        grade = 'C';
    }
    else if (marks >= 40)
    {   grade = 'D';
    }
    else
    {   grade = 'F';
    }
}
void main()
{
    clrscr();
    fstream fio("marks.dat", ios::in | ios::out);
    char ans = 'y';
    while (ans == 'y' || ans == "Y")
    {
        stud1.getdata();
        fio.write((char *) & stud1, sizeof(stud1));
        cout << "Record added to file\n";
        cout << "\n want to enter more? (y/n):.";
        cin >> ans;
}
    clrscr();
    int rno;
    long pos;
    char found = 'f';
    cout << "Enter roll no. of student whose record is to modified:";
    cin >> rno;
    fio.seekg(0);
    while (!fio.eof())
    {
        pos = fio.tellg();
        fio.read((char*) & stud1, sizeof(stud1));
        if (stud1.getrno() == rno)
        {
            stud1.modify();
            fio.seekg(pos);
            fio.write((char*) & stud1, sizeof(stud1));
            found = 't';
            break;
```

```
    }
  }
  if (found == 'y')
  {
      cout << "\n Record not found in the file... !!!\n";
      cout << "Press any key to exit ... \n";
      getch();
      exit(2);
  }
  fio.seekg(0);
  cout << "\n Now the file contains: \n";
  while (!fio.eof())
  {
      fio.read ((char*)&stud, sizeof(stud));
      stud.putdata();
  }
  fio.close();
  getch();
}
```

Q6:- Explain the concept of Virtual and Pure Virtual function with the help of examples?

Sol^{n}- Virtual function :- A virtual function is a member function which is declared within base class and re-defined by a derived class.

- They are mainly achieve by Run-time polymorphism.
- Functions are declared with a virtual keyword in base class
- The resolving of function call is done at Runtime

program :-

```
#include <iostream>
using namespace std;
class base
{
  public:
  virtual void print()
  {
    cout << "print base class" << endl;
  }
```

```
    show()
    cout<<" show Base class"<<endl;
    }
};
class derived : public Base
{
  public:
  void print()
  {
    cout<<" print derived class"<<endl;
    }
    void show()
    {
      cout<<" show derived class"<<endl;
    }};
int main()
{
    base *bptr;
    derived d;
    bptr = &d;
    bptr -> print();
    bptr -> show();
}
```

Pure Virtual function:-

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. Eg:- let Shape be a base class. we cannot provide implementation of function draw() in shape. but we know every derived class must have implementation of draw(). Similarly Animal class doesn't have implementation of move(), but all animals must know how to move. We cannot create objects of abstract class

A pure virtual function in C++ is a virtual function for which we don't have implementation, we only declare it. A pure function is declared by assigning 0 in declaration.

For eg:-

```
#include <iostream>
using namespace std;
class Base
{ int x;
  public:
  virtual void fun() = 0
  int getx()
  { return x;
  } };
class derived : public Base
{ int y;
  public:
  void fun()
  { cout << "fun";
  }
};
int main(void)
{ derived d;
  d.fun();
  return 0;
  }
```

(1) A class is abstract, if it has atleast one pure virtual function.

(4) An abstract class can have pointers
we do not ques
derived class, then

//Program:
class base

an have pointers and reference of abstract class type

if we do not override the pure virtual function in derived class, then derived class also become abstract.

(4) An abstract class can have constructors;

```
// Program:-
class base
{ protected:
   int x;
  public:
  virtual void fun()=0
    Base (int i) {x = i;}
  };
  class derived : public Base
  { int y;
    public:
    Void fun () { cout << "x=" << x << ", y =" << y ; }
  };
  int main( )
  { derivedd(4,5);
    d. fun();
    return 0;
    }
```

SECTION-C

Q8:- Create a class whose object represents a complex number. WAP so that it is possible to add two objects of this class and store the result in third object.

Soln:-

```
#include <iostream>
using namespace std;
Class complex
{
  Public:
  Int real;
```

```
int img;
void setvalue()
{
  cin >> real;
  cin >> img;
}
void display()
{
  cout << real << "+" << img << "i" << endl;
}
void sum (complex c1, complex c2)
{
   real = c1.real + c2.real;
   img = c1.img + c2.img;
};
int main()
{
   complex c1, c2, c3;
   cout << " Enter the real and imaginary part of first
                complex no." << endl;
   c1.setvalue();
   cout << " enter the real imaginary part of second
                complex no." << endl;
   c2.setvalue();
   cout << " sum of two complex no." << endl;
    c3.sum(c1, c2);
    c3.display()
    return 0;
}
```

What is a Template ? Explain with help of example how to create a function Template and a class template.

Ans:- A template is a simple and yet very powerful tool in C++. The simple idea to pass data type as a parameter so that we don't need to write the same code for different data types. For eg:-, a software company need sort() for different data types.

Function template:-

```
#include <iostream>
using namespace std;
Template < typename T >
T myMax (T x, Ty)
{
  return (x > y) ? x : y;
}
int main()
{
  cout << my Max>(3,7) << endl;
  cout << my Max < double > (3.0, 7.0) << endl;
  cout << MyMax < char > ('g', 'e') << endl.
  return 0;
}
```

Class template:

```
#include <iostream>
class Array {
private:
T *ptr;
int size;
public:
Array (T arr[], int s);
void print();
};
```

```
template <typename T>
Array <T> :: Array (Tarr[], int s) {
    ptr = new T [s];
    size = s;
    for (int i=0 ; i<size ; i++)
        ptr [i] = arr[i];
}
template <typename T>
void array <T> :: print()
{   for (int i=0; i<size; i++)
        cout << " " << *(ptr+1);
        cout << endl;
}
```

Q:- (a) What is inheritance? WAP to inherit a class in C++?

(b) What is dynamic memory Allocation? Explain with the help of an example how to create and destroy object dynamically.

Soln:- (a) Inheritance:-

The capability of a class to drive properties and characteristics from the another class is called inheritance.

Sub class:- The class that inherits properties from another class is called Sub class or Derived class

Super Class:- The class whose properties get inherited by sub class is called super class or base class

There are five types of inheritance

(a) single inheritance
(b) Multiple inheritance
(c) Multilevel inheritance
(d) Heirarchical inheritance
(e) hybrid inheritance

```
class vehicle
{
    public:
    vehicle()
    {
      cout<<" I am vehicle"<<endl;
    }
};
class car: public vehicle
{
  };
int main()
{
   car c1;
    return 0;
}
```

(b) memory in your C++ program is divided into two parts:-

- The stack:- All variables declared inside the function will take up memory from the stack.
- The heap:- This is unused memory of the program and can be used to allocate the memory dynamically when program runs

You can allocate memory at run-time within the heap for the variable of a given type using special operator which returns the address of the space allocated. This operator is called new operator.

If you are not in need of dynamically allocated memory anymore, you can use delete operator, which de-allocates memory that was previously allocated by new-operator

☆ Dynamic memory Allocation for Arrays

Consider you want to allocate memory for an array of characters, i.e. String of 20 characters. Using same syntax what we have used above we can allocate memory

```
char *pvalue = NULL;
pvalue = new char[20]
```

⇒ To remove the array

```
delete [] pvalue.
```

For eg:-

```
#include <iostream>
using namespace std;
Class Box
{
  public:
   Box()
  {
   cout << "Constructor called" << endl;
  }
 ~ Box()
  {
   cout << " destructed" << endl;
 } };
int main()
{
  Box *myBoxArray= new Box[4];
  delete [] myBoxArray;
   return 0;
}
```

DEC-2018

Roll No. | | | | | | | | | | |

Total No. of Pages : 02

Total No. of Questions : 18

B.Tech.(Electronics Engg.)/(3D Animation & Graphics) (2012 Onwards)
B.Tech.(CSE)/(ECE)/(Electronics & Computer Engg.)/(ETE)/(IT)
(2011 Onwards) (Sem.–3)

OBJECT ORIENTED PROGRAMMING USING C++

Subject Code : BTCS-305
Paper ID : [A1129]

Time : 3 Hrs.

Max. Marks : 60

INSTRUCTION TO CANDIDATES :

1. SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.
2. SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.
3. SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.

SECTION-A

Answer briefly :

1. When you use static data members?
2. What is size of operator?
3. Define *this* pointer.
4. What are *istream* class functions in C++ Programming?
5. What is dynamic memory location?
6. Discuss the rules of defining constructors.
7. Why do we need virtual destructors?
8. How do you call a virtual function in base class?
9. Write the use of function overriding.
10. What is initializers list in C++?

OOPS
UIET - 2018

SECTION-B

11. What is object oriented programming? Explain any five characteristics of object oriented programming languages.

12. Explain public, private and protected access specifiers and show the ambiguity in multiple and multipath inheritance.

13. What do you mean by type conversion? Give an example of basic to object conversion.

14. What is the difference between early binding and late binding in C++?

15. Define Virtual Function. Explain the mechanism of Virtual function.

SECTION-C

16. Define Operator Overloading. Explain how to overload unary operator and binary operator.

17. Write a program in C++ that display entered string into reverse order.

18. What are function templates of C++? Discuss the concept of error handling functions supported in C++.

OOPS
year - 2018
December

tion - A

~~Subject code~~

Static data member

The member whose only one copy is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

Syntax

static data name;

ex

static int count;

Use

Static variables are normally used to maintain values common to the entire class. for example static data member can be used as a counter that record the occurrences of all the objects

2) Sizeof

The operator that determines the size in bytes of a variable or datatype

The size of operator can be used to get the size of classes, structures, unions and any other user defined data type

Syntax

sizeof (datatype)

The datatype is the desired datatype including classes, structures, unions and any other user defined datatype

This pointer

This is a pointer that points to an object for which this function was called.

this is used to represent an object that invokes a member function

for example

The function call A.max() will set the pointer this to the ~~object~~ address of object A.

(4) ~~Istream~~

Istream : The class provides the facilities for formatted and unformatted input.

Istream functions

1. get()

get() function is of two type :-

- get(char *) :- It is used to fetch a character including the blank space, tab and the newline character.
- ~~get()~~ :- It assign the input character to its argument.
- get(Void) :- This function is same as get(char *), the difference it returns the inputcharacter.

2. getline() :- It reads a whole line of text that ends with a newline character

3. read() :- It is used to read characters.

4. ignore() :- It is used to ignore or clear one or more character from input buffer

object for which it
is that invokes it

Dynamic memory location

C++ supports two operators new and delete that perform the task of allocating and freeing the memory dynamically

The object can be created by using new and destroyed by using delete.

Syntax:

pointer variable = new datatype; // for new operator

Ex

p = new int;

here p is a pointer variable

delete pointer variable; // for delete operator

delete p; ~~# for~~

Since new and delete operator allocate the memory at runtime so it is called dynamic memory location

(6) Rules of defining constructor

constructors

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is same as classname.

Rules

1 They should be declared in public section.
2 They are invoked automatically when objects are created.
3 They cannot have return type

4 They cannot be inherited, although a derived class constructor can call the base class

5 Constructors cannot be virtual

6 They cannot refer to their addresses

7 Like other C++ function, they can have default arguments

(7) Virtual destructors

Virtual destructor in base class ensures that the destructor of derived class will be called when using base pointer to the object of that class.

Syntax

// definition

virtual ~ constructor_name()

{

}

example

virtual ~ base() // base is the name of the destructor

{

}

The derived destructor of the base class is declared as virtual - whenever upcasting is done, Destructors of the base class must be made virtual for proper destruction of the object.

First the derived class destructor is called which the base class pointer is pointing then the base class destructor is pointing to.

virtual
I can call a base class ~~same~~ function by taking the base class pointer to point to the object of the base class.

Example

```
class base
{
  public:
  virtual void show()
  {
   cout<<" in show base";
  }
};
class derived: public base
{
  public:
   void show()
 {
 cout<<" in show derived:"
}
};
int main()
{
 base *ptr; // base class pointer.
   *ptr= &base
 base obj; // base class object
   ptr = &obj;
   ptr -> show(); // base version will be called
 return 0;
}
```

Output

show base.

9 Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class. The function in parent class is called overridden function and function in child class is called overriding function.

When we make call to function (undergoing overriding) gets called the child class function (overriding function) gets called.

Example

```
class base
{
public:
 void display()
 {
  cout<<"\n function of parent class";
 }
};
class derived : public base
{
 public:
 void display()
 {
  cout<<"\n function of derived class";
 }
};
int main()
 {
  derived obj;
  obj.display();
 return 0;
}
```

Initialize list is used to initialize the data members of a class. The list of members to be initialized is indicated with constructor as a comma seperator list followed by a colon

Example

```
#include<iostream>
using namespace std;
class point
{
  private:
  int x;
  int y;
  public:
  point(int i, int j): x(i), y(j){ }
  void show()
  {
    cout<<"\n x "<< x;
    cout<<"\n y"<<y;
  }
};
int main()
{
  point p(10,20);
  p.show();
return 0;
}
```

Output

x = 10
y = 20

(11) OOP :-

It is a programming paradigm which treats data as a critical element and doesn't allow it to flow freely around the system. It ties data more closely to the function that operates on it.

Some features of OOP are :-

* Emphasis is on data rather than procedure.
* programs are divided into units called as objects
* Functions that operate on the data of an object are tied together in the data structure.
* Data is hidden and cannot be accessed by external functions.

Various characteristics of Object oriented programming are :-

1 objects

Objects are basic runtime entities in object oriented system. Objects are basically nothing but variables of type class. Objects may represent person, place, bank account, a table of data or any item that the program has to handle. The objects interact by sending messages to one another.

~~2 Object~~

2 Class

The entire set of data and code to manipulate the data of an object can be made a user defined data type with the help of a class. It can be collection of objects similar type.

data abstraction

The wrapping of representing essential features without including the background details or explanations. classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight etc and code (functions) to act on these attributes

4 data encapsulation

The wrapping up of data and functions into a single unit called class is called data encapsulation. The data is not accessible to the outside world and only those functions which are wrapped inside the class can access it

5 Inheritance

Inheritance is the process by which objects of one class acquire the properties of another class.

The concept of inheritance provide the idea of reusability This means we can add additional features to an existing class without ~~existing it~~ modifying it

(12) There are basically three access specifier:
1 private 2 protected 3 public

public, private protected are basically key words:

public: The class members declared as public can be easily accessed by the object of that class by

using dot operator.

```
class xyz
{
  public:   // access specifier
  void display()
  {
   cout<<"\n XYZ";
  }
};
int main()
{
   xyz obj;
   obj.display();
   return 0;
}
```

Output

XYZ

Private

The class members which are declared as private cannot be accessed by the objects of a class directly. The private class members can be accessed by the member functions of the class which are public. class xyz private is the default access specifier.

Example

```
class ABC
{
  private:
   class members;
};
```

ected

class members which are declared protected can be accessed by the member functions of class ~~which~~ and the class immediately derived from it.

Example

```
class xyz
{
  protected:
  class members;
};
```

Ambiguity resolution in inheritance

We may face a problem when a function with same name appear in more than one base class.

```
class M
{
  public:
  void display()
  {
    cout << "\n class M";
  }
};
class N
{
  public:
  void displayl()
  {
    cout << "\n classN";
  }
};
class A: public M, public N
```

```
{
  public:
  void display() // overrides display() of M and N
  {
    M::display();
  }
};

int main()
{
  A obj;
  obj.display();
}
```

output

class M

The duplication of inherited members due to multipath can be avoided by making the common ancestor class (base class) as virtual class base class while declaring the direct or intermediate base classes

```
class A
{
  ...
};
class B1: virtual public A
{
  ...
};
```

: public virtual A // Parent 2

{
}

class C : public B1, public B2

{
}

13 Type Conversion

A type conversion is basically conversion from one type to another. There are two types of type conversion

1 implicit type conversion :- Such type of conversions are done by compiler by its own. They are also known as automatic type conversion

2 Explicit type conversion :- This process is called type casting and it is user defined. Here user can typecast the result to make it of a particular datatype

There types of situations may arise in the data conversion

* Conversion from basic type to class type.
* Conversion from class type to basic type
* Conversion from one class type to another class type

Basic to Object Conversion

```
#include<iostream>
using namespace std;
class time
```

```
{
    int hrs, min;
  public:
    time(int);
    ~time();
    void display();
};

time::time(int t)
{
  cout<<"Basic type to class type conversion";
    hrs = t/60;
    min = t%60;
}

void time::display()
{
  cout<<hrs<<" hours"<<endl;
  cout<<min<<" minutes"<<endl;
}

int main()
{
  int duration;
  cout<<"\n Enter the duration in minutes";
  cin>>duration;
    Time t1=duration;
    t1.display();
    return 0;
}
```

Early binding	Late binding
In this the object is bound to the appropriate function at the compile time	☆ The object is linked to relevant function at the run time
☆ It is also called static binding	☆ It is also called dynamic binding
☆ It does not require to use pointer for implementation	☆ It uses base pointer which calls a relevant function based on the type of object the base pointer is pointing to
☆ It is called compile time polymorphism	☆ It is called run time polymorphism
☆ Examples of early binding are function overloading and operator overloading	☆ Examples of late binding is virtual function

⑮ Virtual function

When a function is declared as virtual, it determines which function to use at run time based on the type of object pointed to by the base pointer rather than the type of pointer

Mechanism

By making a base pointer to point to different function objects, we can execute different versions of the virtual function. We must access virtual function through the use of a pointer declared as a pointer to the base class.

```
#include<iostream>
using namespace std;
class base
{
 public:
  void display()
  {
   cout<<"Display base";
  }
  virtual void show()
  {
   cout<<"\n show base";
  }
};
class derived
{
 public:
  void display()
  {
   cout<<"\n Display derived";
  }
  void show()
{
```

```
n show derived ();

main()

Base B;
derived D;
base *ptr;
ptr = &B;
ptr -> display(); // calls base versions
ptr -> show(); // calls base version
ptr = &D;
ptr -> display(); // calls ~~base~~ version
ptr -> show(); // calls derived version
return 0;
}
```

~~Output~~

Display base
show base

Display base
show derived

Section + C

(16) Operator overloading

C++ has ability to provide the operators with

a special meaning for a data type. The mechanism of giving such special meaning to an operator is known as operator overloading. It provides a flexible option for the creation of new definitions for most of the C++ operators

Overloading of unary minus

```
#include <iostream>
using namespace std;
class space
{
  int x;
  int y;
  int z;
public
  void getdata(int a, int b, int c);
  void display();
  void operator-(); // overload unary minus
};
void space :: getdata (int a, int b, int c)
{
   x = a;
   y = b;
   z = c;
}
void spce :: display()
{
   cout << x << " ";
   cout << y << " ";
```

```
<<"\n";

space :: operator -()

    x = -x;
    y = -y;
    z = -z;
}
int main()
{
 space S;
 S.getdata(10,-20,30);
 cout<<"S:";
  S.display();
  -S;
  cout<<"S:";
   S.display();
  return 0;
}
```

output

```
S: 10 -20 30
S: -10 20 -30
```

```
// overloading binary operators
#include<iostream>
 using namespace std;
  class complex
{
```

```
    float x;
    float y;
    public:
    complex() { } // constructor
    complex ( float real, float imag)
    {
      x = real;
      y = imag;
    }
    complex operator+(complex);
    void display();
};
complex complex :: operator + (complex c)
{
    complex temp;
    temp.x = x + c.x;
    temp.y = y + c.y;
    return (temp);
}
void complex :: display()
{
 cout << x << " + j" << y << endl;
}
int main()
{
 complex c1, c2, c3;
 c1 = complex (1.3, 3.5);
 c2 = complex (1.4, 3.2);
 c3 = c1 + c2;
```

Standard Template Library is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, ~~and functions such as stacks, lists~~. It is a library of container classes, algorithms and iterators. It is a generalized library and so its components are parameterized

STL functions

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

Error handling functions supported in C++

Most of the time the exceptions are thrown in functions that are invoked from within the try block. The point at which the throw is executed is called the throw point. Once an exception is thrown to the catch block, control returns to the throwpoint.

~~type of fun~~
datatype functionname(arglist)
{
    throw(object); // throws exception
}

```
cout<<"c2 = "; c2.display();
cout<<"c3 = "; c3.display();
return 0;
}
```

output

```
c1 = 1.3 + j3.5
c2 = 1.4 + j3.2
c3 = 2.7 + j6.7
```

(17)

```
#include<iostream>
#include<string.h>
int main()
{
 char str[100], temp;
 int i, j=0;
 cout<<"\n Enter any the string: ";
 gets(str); //get function for input string
 i=0;
 j = strlen(str)-1
 while(i<j)
 {
  temp = str[i];
  str[i] = str[j];
  str[j] = temp;
  i++;
  j--;
}
cout<<"Reverse string is: "<< str
return 0;
}
```

```
    // invoke function here }
functionname();

atch(type arg)

    // code to handle exception

#include<iostream>
using namespace std;
void divide(int x, int y, int z)
{
  if((x-y) != 0)
  {
    int R = z/(x-y);
    cout<<"Result = "<<R<<"\n";
  }else
  {
    throw (x-y); // throw pair
  }
}
int main()
{
  try
  {
    cout<<"\n we are inside the try block";
```

```
    divide(10,20,30); // invoke divide()
    divide(10,10,20); // invoke divide()
}
catch(int i)
{
cout << " caught the exception\n";
}
return 0;
}
```

PTU exam May 2019

SECTION-A

Answer briefly :-

1. Destructor - A destructor is a special method called automatically during the destruction of an object.
   Actions executed in the destructor include the following:
   - Recovering the heap space allocated during the lifetime of an object.
   - Closing file or database connections
   - Releasing network resources
   - Releasing resource locks
   - Other housekeeping tasks

2. Dangling pointer - Dangling pointers are pointers that pointing to a memory location that has been deleted. It arise during object destruction, when an object that has an incoming refrence is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. The system may reallocate the previously freed memory, unpredictable behaviour may result as the memory may now contain completely different data.

3. Arrays - An array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. Arrays are commonly used in computer programs to organise data so that a related set of values can be easily sorted or searched.

declared with keyword 'class' that has data and functions (also called member variables and member functions) as its members whose access is governed by three access specifiers private, protected or public. By default access to members of a class is private.

5. Static class - When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member. A static member is shared by all objects of the class. All static data is initialised to zero when the first object is created, if no other initialisation is present.

6. Friend function - Friend functions are those functions which can access all the functions and variables of a class though it is not a member function of that class. Actually to share a function among two or more classes friend functions are used.

7. Call by value - The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In general it means the code within a function cannot alter the arguments used to call the function.

8. Data types - Data types define the type of data a variable can hold, for example an integer variable can hold integer data, a character type variable can hold character data etc. It is of three types Built-in, user-defined and derived.

various programming elements such as variables, functions, arrays, structures, unions and so on. Actually, an identifier is a user defined word.

10. Pure virtual function- A pure virtual function is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 is declaration. It is necessary to redefine this function in the derived class. It is also defined in base class.

SECTION-B

Q 11. Discuss the features of constructors.

Answer
- Constructors are called automatically when objects are created.
- All objects of the class having a constructor are initialised before some use.
- These should be declared in the public section for availability to all the functions.
- Return type (not even void) cannot be specified for constructors.
- These cannot be inherited, but a derived class can call a base class constructor.
- These cannot be static.
- Generated constructors are public.
- A constructor can call member functions of its class.
- These can have default arguments as other C++ functions.
- An object of a class with a constructor cannot be used as a member of an union.

3. It has no concept of derived class.	3. If a class contains at least one pure virtual function, then it is declared abstract.
4. If required, the base class can override a virtual function.	4. In case of pure virtual function derived class has to be difinitely override the pure virtual function.

Q13. Write a program to overload + operator.

Answer.

```
# include <iostream>
using namespace std;
class A
{  int x;
   public: A() { }
           A (int i)
           {  x = i;
           }
           void operator +(A);
           void display;
};
void A :: operator +(A a)
{   int m = x + a.x
```

```
int main()
{
    A a1(5);
    A a2(4);
    a1+a2;
    return 0;
}
```

OUTPUT: The result of addition of two objects is : 9

Q14. Differentiate between static and dynamic memory allocation.

Answer.

Static memory Allocation	Dynamic memory allocation.
1. In this case, variables get allocated permanently	1. In this case, variables get allocated only if your program until gets active.
2. Allocation is done before program execution	2. Allocation is during program execution.
3. It uses the data structure called stack for implementing static allocation.	3. It uses the data structure called heap for implementing dynamic allocation.
4. Less efficient	4. More efficient
5. There is no memory reusability	5. There is memory reusability and can be freed when not required

Q15. Discuss the use of exceptional handling in programming.

Answer. Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords : try, catch and throw.

SECTION - C

Write a note on templates.

A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter

C++ adds two new keywords to support templates: 'template' and 'typename'. The second keyword can always be replaced by keyword 'class'.

Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function or class, but compiled code may contain multiple copies of same function / class.

Example:-

```
# include <iostream>
using namespace std;
template <typename T>
T myMax (T x, T y)
{
    return (x>y)? x:y;
}
int main()
{
  cout << myMax<int>(3,7) << endl;
  cout << myMax<double>(3.0, 7.0) << endl;
  cout << myMax<char>('g','e') << endl;
  return 0;
}
```

Q17. What are the various file opening modes? Explain.

Answer. For every file operation, exists a specific file mode. These file modes allow us to create, read, write, append or modify a file. The file modes are defined in the class ios. Let's see all these different modes in which we could open a file on disk

File Modes	Description
ios::in	Searches for the file and opens it in the read mode only (if the file is found)
ios:out	Searches for the file and opens it in the write mode. If the file is found, its content is overwritten. If the file is not found, a new file is created. Allows you to write to the file.
ios::app	Searches for the file and opens it in the append mode i.e. this mode allows you to append new data to the end of file. If the file is not found a new file is created.
ios::binary	Searches for the file, opens the file (if the file is found) in a binary mode to perform binary input / output file operations.
ios::ate	Searches for the file, opens it and positions the

	pointer at the end of the file. This mode when used with ios::binary, ios::in and ios::out modes, allows you to modify the content of a file.
ios::trunc	Searches for the file and opens it to truncate or deletes all of its content (if the file is found.)
ios::nocreate	Searches for the file and if the file is not found, a new file will not be created.

What are different types of inheritance? Explain.

There are five types of inheritance:-

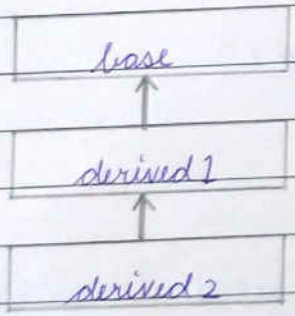
Single inheritance- Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

Syntax:

```
class base
{   //
};
class derived : [access_to_base] base
{   //...
};
```

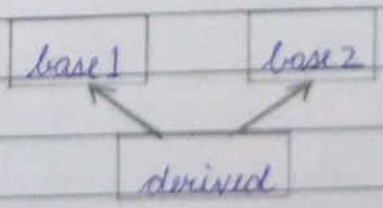
2. Multilevel Inheritance - Multilevel inheritance is a process of deriving a class from another derived class. When one class inherits another class which is further inherited by another class, it is known as multilevel inheritance.

syntax:
```
class base
{ //...
};
class derived1 : [access_to_base] base
{ //...
};
class derived2 : [access_to_base] derived1
{ //...
};
```



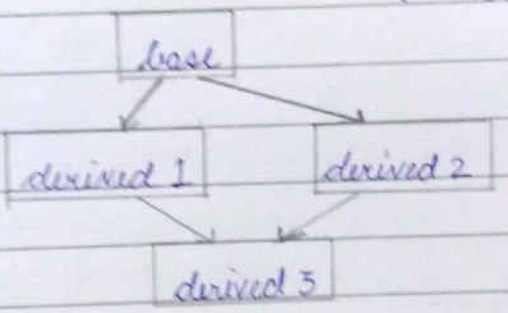
3. Multiple inheritance - Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes

syntax:
```
class base1
{ //...
};
class base2
{ //...
};
class derived : [access] base1 , [access] base2
{ //...
};
```



4. Hybrid inheritance - Hybrid inheritance is a combination of more than one type of inheritance.

syntax:
```
class base
{ //...
};
class derived1 : [access] base
{ //...
};
class derived 2 : [access] base
{ //...
}; class derived 3 : [access] derived1, [access] derived2
                    { //... };
```



5. Hierarichal inheritance - Hierarichal inheritance is defined as the process of deriving more than one class from a base class.

syntax:
```
class base
{ //...
}
class derived 1 : [access] base}
{ //...
};
class derived2 : [access] base
{ //...
};
```