Royal Education Society's

College of Computer Science and Information Technology, Latur.

Department of Computer Application

Academic Year (2022-23)

Choice Based Credit System (CBCS Revised)

Class/Semester: **BCA SY SEM-III**

Name of Paper: **C ++**

Model Question Paper

Q.1 Attempt any FIVE of the following (3 Marks each) 15

a) Explain Dynamic Binding.

Answer:

1. **Dynamic binding:** "Dynamic binding means to select a particular function to run until the runtime. Based on the type of object, the respective function will be called."
2. It is the one of the important concept of Object Oriented Programming.
3. The general meaning of binding is linking something to a thing. we can describe binding as linking function definition with the function call.
4. The concept of dynamic programming is implemented with virtual functions.

Characteristics:

a) Run time function resolving

b) Used to achieve runtime polymorphism

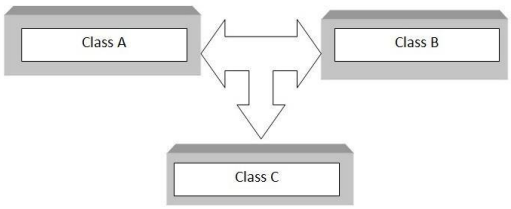
c) All virtual functions are declared in the base class

d) Assurance of calling correct function for an object regardless of the pointer(reference) used for function call.

b) Explain Multiple Inheritance.

Answer:

Multiple Inheritance: "If a class is derived from two or more base classes then it is called multiple inheritance.**"**



1. It allows the new class to inherit the functionality of two or more well developed and tested base classes, thus reusing predefined classes' features.
2. This form of inheritance is useful in those situations in which the derived class represents a combination of features from two or more base classes. For Example: In an educational institute, a member can act like a teacher and a student. He may be a student of higher class as well as teaching lower classes.
3. **The syntax** for declaring a derived class that inherited from more than one base classes is
   class der_class_name : access_ specifier base1, access_ specifier base2, ...
   {

```
//Implementation
};
```

c) Write down benefits of OOP's.

Answer:

Benefits of OOP's:

1. Re-usability

It means reusing some facilities rather than building them again and again. This is done with the use of a class. We can use it 'n' number of times as per our need.

2. Data Redundancy

This is a condition created at the place of data storage (you can say Databases) where the same piece of data is held in two separate places

3. Code Maintenance

It helps users from doing re-work in many ways. It is always easy and time-saving to maintain and modify the existing codes by incorporating new changes into them.

4. Security

With the use of data hiding and abstraction mechanism, we are filtering out limited data to exposure, which means we are maintaining security and providing necessary data to view.

5. Better productivity

This leads to more work done, finishing a better program, having more inbuilt features, and easier reading, writing and maintaining.

6. Polymorphism Flexibility

This means polymorphism is flexible and helps developers in a number of ways.

7. Problems solving

Decomposing a complex problem into smaller chunks or discrete components is a good practice. OOP is specialized in this behavior, as it breaks down your software code into bite-sized – one object at a time.

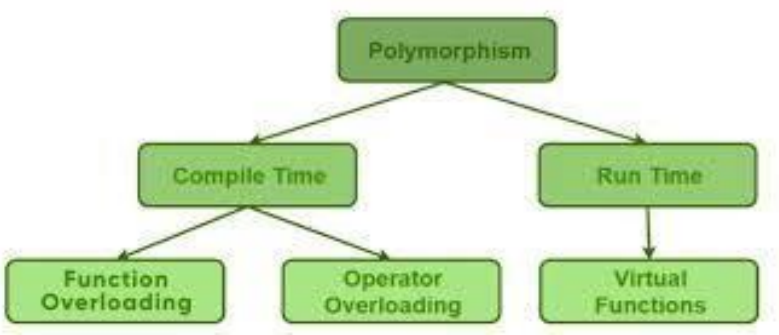
d) What is polymorphism? Explain.

Answer:

Polymorphism: When the same entity (function or object) behaves differently in different scenarios, it is known as Polymorphism.

Types of Polymorphism:

Polymorphism is categorized into two types. The figure below shows the types:



1. Compile Time Polymorphism

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding.

Function overloading and operator overloading is the type of Compile time polymorphism.

I. Function Overloading

Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments

II. Operator Overloading

Operator overloading means defining additional tasks to operators without changing its actual meaning. We do this by using operator function.

The purpose of operator overloading is to provide a special meaning to the user-defined data types. The advantage of Operators overloading is to perform different operations on the same operand.

2. Runtime Polymorphism

In a Runtime polymorphism, functions are called at the time the program execution. Hence, it is known as late binding or dynamic binding. Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.

I. Function overriding

In function overriding, we give the new definition to base class function in the derived class. At that time, we can say the base function has been overridden.

II. Virtual Function

A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void. A virtual function is a member function in the base class. We can redefine it in a derived class. It is part of run time polymorphism. The declaration of the virtual function must be in the base class by using the keyword virtual. A virtual function is not static.

e) What is Function? Explain.

Answer:

1. **Function:** A function is defined as a group of statements or a block of code that carries out specific tasks. Functions are used to minimize the repetition of code, as a function allows you to write the code inside the block.
2. A function can be declared by writing its return type, the name of the function, and the arguments inside brackets

   Return_type function_name (argument list);
3. **Function definition**: A function definition specifies the body of the function. The declaration and definition of a function can be made together, but it should be done before calling it**.**

```
Return_type function_name (argument list)
{
   Statement;

}
```

4. **Calling:**

When a function is called from the main function, then the control of the function is transferred to the function that is called. And then that function performs its task.

5. **Types of Functions in**

There are two types of functions in C++

- Built-in functions
- User-defined functions

Built-in Functions:
These are functions that are already present and their definitions are already provided in the header files.

User-defined Functions:
These are functions that a user creates by themselves, wherein the user gives their own definition.

f) What is destructor? Explain.

Answer:

Destructor:

"Destructor is a special member function that is executed automatically when an object is destroyed that has been created by the constructor".

1. destructors are used to de-allocate the memory that has been allocated for the object by the constructor.
2. Destructor has the same name as their class name preceded by a tilde (~) symbol.
3. It is not possible to define more than one destructor.
4. The destructor is only one way to destroy the object create by constructor. Hence destructor can-not be overloaded.
5. Destructor neither requires any argument nor returns any value.
6. It is automatically called when object goes out of scope.
7. Syntax for defining the destructor within the class

```
~ <class-name>( )
{
   }
```

Syntax for defining the destructor outside the class

```
<class-name>: ~ <class-name>()
{
   }
```

g) Define Static Data Member and Static Member Function and Explain.

Answer:

Static data members: "Static data members are class members that are declared using **static** keywords".

1. Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created**.**
2. The static data members are associated with the class and not with any object, so they are also known as class variables.
3. The static data members must be defined outside the class; otherwise, the linker will generate an error. Also, there is only one definition of a static data member.
4. static data member can be declared in the following way:
   static data_member;
   static data_type identifier;

Static Member Function:" A static member function is a member function of a class that is defined with the static keyword"

1. A static member function cannot define a non-static data member and cannot call a non-static member function.
2. A static member function need not be instantiated and can be called without creating any object of the class by using the scope resolution operator (: :).
3. Declaration syntax:
   static return_type function_name();

Q. 2 Attempt any three of the following (5 Marks each) 15

a) What is Constructor? Explain with program.

Answer:

1. **Constructor: "**Constructor is automatically called when an object (an instance of the class) create. It is a special member function of the class."
2. **explanation:** It has the same name of the class. It must be a public member.
3. Constructor does not Return Values. Default constructors are called when constructors are not defined for the classes.

4. Syntax: class class-name {

```
Access Specifier:
Member - Variables
Member – Functions
public: class-name () {
// Constructor code      }
//... other Variables & Functions}
```

5. There are 3 types of Constructors.

- **Default Constructor**

A constructor with no arguments (or parameters) in the definition is a default constructor. Usually, these constructors use to initialize data members (variables) with real

values.

Note: If no constructor is explicitly declared, the compiler automatically creates a default constructor with no data member (variables) or initialization.

- **Parameterized Constructor**

It contains parameters (or arguments) in the constructor definition and declaration. More than one argument can also pass through a parameterized constructor.

- **Copy Constructor**

A copy constructor is a member function that initializes an object using another object of the same class. It helps to copy data from one object to another.

6. Program to demonstrate the constructor:

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


//using namespace std;
class Example {
   // Variable Declaration
   int a, b;
   public:
      //Constructor
      Example () {
```

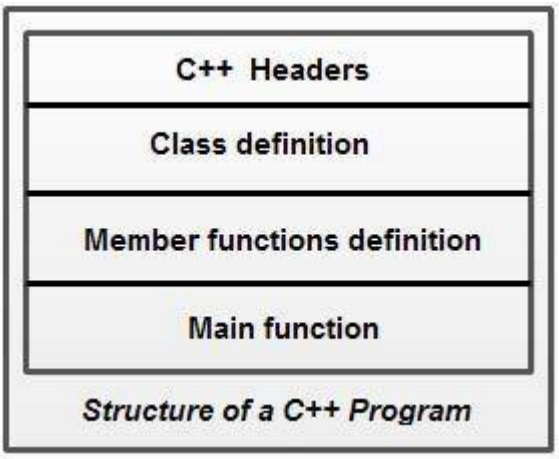
```
        // Assign Values In Constructor
        a = 10;
        b = 20;
        cout << "I am Constructor\n";
    }
    void Display () {
        cout << "Values:" << a << "\t" << b;
    }
};
int main() {
    Example Object;
    // Constructor invoked.
    Object. Display();
     getch()
    return 0;
}
```

```
Output:  I am Constructor
      Values :10      20
```

b) Explain Structure of CPP program with example.

Answer:

1. Program: “Programs are a sequence of instructions or statements.”
2. **explanation:** C++ program structure is divided into various sections, namely, headers, class definition, member functions definitions and main function.
3. Structure of a C++ Program:



4. **Comments:**

   Comments are a vital element of a program that is used to increase the readability of a program and to describe its functioning. Comments are not executable statements and hence, do not increase the size of a file.

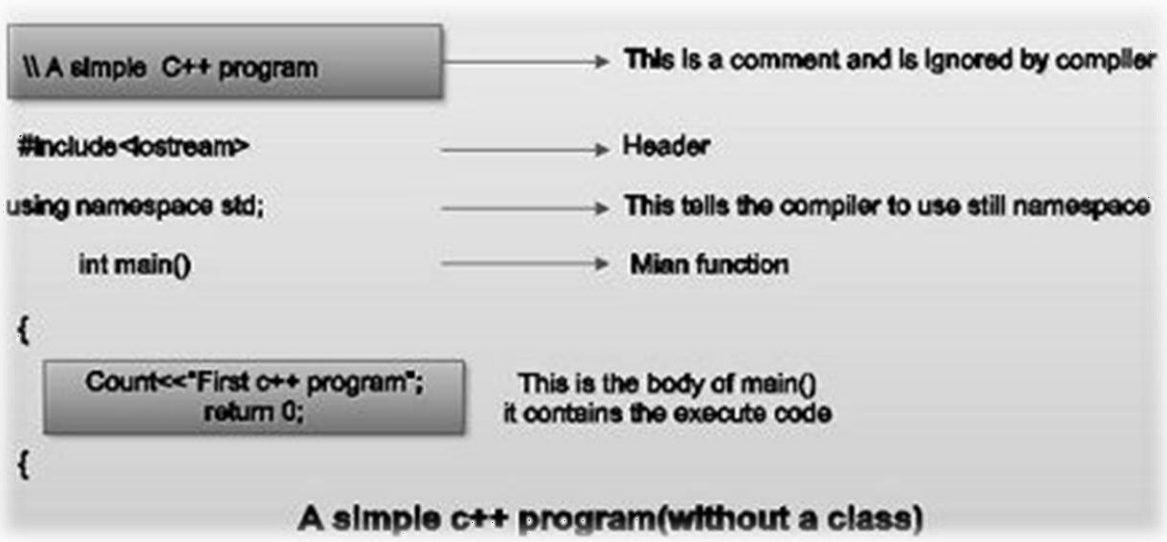
5. **C++ supports two comment styles**: single line comment and multiline comment. Single line comments are used to define line-by-line descriptions. Double slash (//) is used to represent single line comments.

   / / single line comment

   Multiline comments are used to define multiple lines descriptions and are represented as / * * /. For example, consider this statement.

6. **Headers:** In order to use such pre-defined elements in a program, an appropriate header must be included in the program. The standard headers contain the information like prototype, definition and return type of library functions, data type of constants, etc.
7. Standard headers are specified in a program through the preprocessor directive” #include.. When the compiler processes the instruction #inc1ude<iostream>, it includes the contents of iostream in the program.
8. **A namespace** permits grouping of various entities like classes, objects, functions and various C++ tokens, etc., under a single name. This avoids compile-time error that may exist due to identical-name conflicts. the statement using namespace std informs the compiler to include all the entities present in the namespace std.
9. **Main Function:** The main () is a startup function that starts the execution of a c++ program. All C++ statements that need to be executed are written within main (). The compiler executes all the instructions written within the opening and closing curly braces’ {}’ that enclose the body of main ().

10. By default, main () in C++ returns an int value to the operating system. Therefore, main () should end with the return 0 statement. A return value zero indicates success and a non-zero value indicates failure or error.
11. Example:



c) What is Friend Function? Explain with Program.

Answer:

1. "A friend function is defined as a function that can access private, protected and public members of a class."
2. **Explanation:** The friend function is declared using the **friend** keyword inside the body of the class.
3. Friend Function Syntax:

```
class className {
        ... .. ...
    friend returnType functionName(arguments);
        ... .. ...
}
```

4. By using the keyword, the **'friend'** compiler understands that the given function is a friend function.
5. We declare friend function inside the body of a class, whose private and protective data needs to be accessed, starting with the keyword friend to access the data. We use them when we need to operate between two different classes at the same time.
6. The function is not in the 'scope' of the class to which it has been declared a friend.
7. Friend functionality is not restricted to only one class.
8. Friend functions can be a member of a class or a function that is declared outside the scope of class.
9. It cannot be invoked using the object as it is not in the scope of that class. We can invoke it like any normal function of the class.
10. Friend functions have objects as arguments.

11. It cannot access the member names directly and has to use dot membership operator and use an object name with the member name.
12. Program for demonstrating the use of friend function:

13. #include <iostream>

using namespace std;

```
class A
{
  int x=4;
  friend class B; //friend class
};
class B
{
  public:
  void display (A &a)
   {
     cout<<"value of x is:" <<a.x;
   }
};
int main ()
{
  A a;
  B b;
  b. display (a);
   return 0;
}
```

Output:

value of x is:4

d) WAP to demonstrate the use of Hybrid inheritance.

Answer:

1. **Hybrid Inheritance:** "The inheritance in which the derivation of a class involves more than one form of any inheritance is called hybrid inheritance. Basically hybrid inheritance is combination of two or more types of inheritance. It can also be called multi path inheritance.
2. // **Program:**

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

class A
{
        public:
```

```
        int x;
};
class B : public A
{
        public:
        B()     //constructor to initialize x in base class A
        {
          x = 10;
        }
};
class C
 {
        public:
        int y;
        C()  //constructor to initialize y
        {
           y = 4;
      }
};
class D : public B, public C  //D is derived from class B and class C
{
        public:
        void sum ()
        {
           cout << "Sum= " << x + y;
        }
};
int main()
{
       D obj1;         //object of derived class D
        obj1.sum();
        return 0;
}               //end of program
```

Output

Sum= 14

e) WAP to demonstrate the use of Dynamic Cast Operator.

Answer:

1. **Dynamic cast Operator:** The primary purpose for the dynamic_cast operator is to perform type-safe downcasts. A downcast is the conversion of a pointer or reference to a class A to a pointer or reference to a class B, where class A is a base class of B.
2. Program for dynamic cast operator:

```
3.  #include <iostream>
using namespace std;
struct A {
  virtual void f() { cout << "Class A" << endl; }
};

struct B : A {
  virtual void f() { cout << "Class B" << endl; }
};
struct C : A {
  virtual void f() { cout << "Class C" << endl; }
};

void f(A* arg) {
  B* bp = dynamic_cast<B*>(arg);
  C* cp = dynamic_cast<C*>(arg);

  if (bp)
    bp->f();
  else if (cp)
  cp->f();
  else
    arg->f();
};

int main() {
  A aobj;
  C cobj;
  A* ap = &cobj;
  A* ap2 = &aobj;
  f(ap);
  f(ap2);
}
```

Output:
Class C
Class A

Q. 3 Attempt any three of the following (5 Marks each) 15

a) <u>What is Function Overloading? Explain with program.</u>

Answer:

1. "Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters."
2. **Explanation:** When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.
3. The parameters should follow any one or more than one of the following conditions for Function overloading:

   a) Parameters should have a different type

   add (int a, int b)

   add (double a, double b)

   b) They should have a different number

   sum (int a, int b)

   sum (int a, int b, int c)

   c)They should have a different sequence of parameters.

   sum (int a, double b)

   sum (double a, int b)

4. There are two types of function overloading in c++-
   - **Compile time overloading**– In compile time overloading, functions are overloaded using different signatures. Signature of a function includes its return type, number of parameters and types of parameters.
   - **Runtime overloading** -In runtime overloading, functions are overloaded using a different number of parameters.
5. Program:

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

// function with 2 parameters
void display (int var1, double var2) {
   cout << "Integer number: " << var1;
   cout << " and double number: " << var2 << endl;
}

// function with double type single parameter
void display (double var) {
   cout << "Double number: " << var << endl;
}

// function with int type single parameter
void display (int var) {
   cout << "Integer number: " << var << endl;
}
```

```
int main () {

   int a = 5;
   double b = 5.5;

   // call function with int type parameter
   display(a);

   // call function with double type parameter
   display(b);

   // call function with 2 parameters
   display (a, b);

   return 0;
}
```

Output: Integer number: 5
Double number: 5.5
Integer number: 5 and double number: 5.5

b) <u>What is virtual Function? Explain in detail.</u>

Answer:

1. "A virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword."
2. **Explanation:** It is used to tell the compiler to perform dynamic linkage or late binding on the function.
3. There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects.
4. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
5. A 'virtual' is a keyword preceding the normal declaration of a function. When the function is made virtual determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.
6. Virtual functions must be members of some class.
7. Virtual functions cannot be static members.
8. They are accessed through object pointers. They can be a friend of another class.
9. A virtual function must be defined in the base class, even though it is not used.

10. #include <iostream>

{

```
    public:
    virtual void display()
    {
    cout << "Base class is invoked"<<endl;
}
    };
    class B:public A
    {
    public:
    void display()
    {
cout << "Derived Class is invoked"<<endl;
}
    };
    int main()
    {
    A* a;    //pointer of base class
    B b;     //object of derived class
    a = &b;
    a->display();  //Late Binding occurs
    }
```

Output: Derived class is invoked

c) Explain template:

Answer:

1. Templates is defined as a blueprint or formula for creating a generic class or a function.
2. template is also known as generic functions or classes which is a very powerful feature in C++.
3. A keyword “template” in c++ is used for the template’s syntax and angled bracket in a parameter (t), which defines the data type variable.

4. Templates in c++ works in such a way that it gets expanded at compiler time, just like macros and allows a function or class to work on different data types without being rewritten.

5. Types of Templates in C++

   There are two types of templates in C++

   - **Function template**
   - **Class templates**

6. A function template in c++ is a single function template that works with multiple data types simultaneously, but a standard function works only with one set of data types.

7. **Function Template Syntax:**

```
template<class type>ret-type func-name(parameter list)
{
//body of the function
}
```

The class keyword is used to specify a generic type in a template declaration.

8. **class template:**

They are known as generic templates. **Syntax of Class Template**

```
template<class Ttype>
class class_name
{
//class body;
}
```

d) <u>What is scope resolution operator? Explain.</u>

Answer:

1. **"**The Scope Resolution Operator is used to reference the global variable or member function that is out of scope."
2. The operator is represented as the double colon (: :) symbol.
3. It is used to access the global variables or member functions of a program when there is a local variable or member function with the same name.
4. It is used to define the member function outside of a class.
5. It is also used to access the static variable and static function of a class.
6. In the case of multiple Inheritance, it is used to override the function.
7. Program:

```
#include  <iostream>
using namespace std;
class Rectangle
{
   private:
```

```
        int length;
        int breadth;
    public:
        Rectangle (int l, int b){
            length = l;
            breadth = b;
        }
        int area ()
        {
            return length * breadth;
        }
        int perimeter ();
};
int Rectangle::perimeter(){
    return 2*(length + breadth);
}
int main(){
    Rectangle r(8, 3);
    cout << "Area is :"<< r.area() <<endl;
    cout << "Perimeter is :"<< r.perimeter();
}
```

Output: Area is :24
Perimeter is :22

e) <u>What is identifier? Explain reference Variable.</u>

Answer:

1. Identifier is one of the tokens. It is a name given by a programmer which is used to identify the variables, constants, functions, arrays, and also user-defined data.
2. The rules for naming identifiers are as follows −
    - Identifier names are unique.
    - Cannot use a keyword as identifiers.
    - Identifier has to begin with a letter or underscore (_).
    - It should not contain white space.
    - Special characters are not allowed.
    - Identifiers can consist of only letters, digits, or underscore.
    - They are case sensitive.
4. Reference variable is an alternate name of already existing variable. It cannot be changed to refer another variable and should be initialized at the time of declaration and cannot be NULL. The operator '&' is used to declare reference variable.
5. The following is the syntax of reference variable.

    datatype variable_name; // variable declaration

    datatype& refer_var = variable_name; // reference variable

    Here,

    datatype − The datatype of variable like int, char, float etc.

variable_name − This is the name of variable given by user.

refer_var − The name of reference variable.

6. The following is an example of reference variable.

```
#include  <iostream>
using namespace std;
int main() {
  int a = 8;
 int& b = a;
cout << "The variable a : " << a;
 cout << "
The reference variable r : " << b;
return 0;
        }
```

Output:

The variable a : 8

The reference variable r : 8

Q. 4 Attempt any three of the following (5 Marks each) 15

a) WAP for inline Function and Explain.

Answer:

1. An inline function is a function that is expanded inline when it is invoked, thus saving time. The compiler replaces the function call with the corresponding function code, which reduces the overhead of function calls.
2. Program:

```
#include<iostream>
  using namespace std;
// define an inline function
// using the keyword "inline"
inline int setNum()
{
 return 10; // definition of inline function
}
int main()
{
   int num ;
   // call the inline function
   num = setNum();
   // setNum() will be replaced by
   // definition of inline function
   cout << " The inline function returned: " << num ;
   cout << "\n\n";
   return 0 ;
}
```

Output:

The inline function returned 10:

3. Explanation of the Program

4. In the above program, the function setNum() is declared as an inline function using the keyword “inline”. This function simply returns an integer. When a call to this inline function is made, the compiler replaces the calling statement with the definition of the inline function setNum().

5. Instead of storing the memory address of the instruction and going through the process of loading the called function into the memory, the function’s definition is directly replaced at the calling instruction. So, in the instruction, num = setNum(), setNum() is replaced by 10, and 10 is stored in the variable num.

b) WAP for nesting of member function:

Answer:

1. A member function called into another member function is referred to as nesting of member functions.

2. Example Program:

```
#include  <iostream>
using namespace std;
class average
{
int a,b;
public:
void read();
void print();
int avg();
};
void average::read()
{
cout<<"\n enter a and b: ";
cin>>a>>b;
}
void average::print()
{
cout<<"value of a: "<<a;
cout<<"\nvalue of b: "<<b;
cout<<"\naverage is : "<<avg();
}
int average::avg()
{
return (a+b)/2;
}
main()
{
average A;
A.read();
A.print();
}
```

Output:

enter a and b: 10 20

value of a: 10

value of b: 20

average is: 15

c) Distinguish between Constructor and Destructor

Answer:

Sr. No.	Constructor	Destructor
1.	Constructor helps to initialize the object of a class.	Whereas destructor is used to destroy the instances.
2.	It is declared as **className(arguments if any){Constructor's Body }**.	Whereas it is declared as **~ className(no arguments){ }**.
3.	Constructor can either accept arguments or not.	While it can't have any arguments.
4.	A constructor is called when an instance or object of a class is created.	It is called while object of the class is freed or deleted.
5.	Constructor is used to allocate the memory to an instance or object.	While it is used to deallocate the memory of an object of a class.
6.	Constructor can be overloaded.	While it can't be overloaded.
7.	The constructor's name is same as the class name.	Here, its name is also same as the class name preceded by the tiled (~) operator.
8.	In a class, there can be multiple constructors.	While in a class, there is always a single destructor.
9.	There is a concept of copy constructor which is used to initialize an object from another object.	While here, there is no copy destructor concept.
10.	They are often called in successive order.	They are often called in reverse order of constructor.

d) WAP for Binary Operator Overloading.

Answer:

1. Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

```
include<iostream.h>
#include<conio.h>
class complex {
   int a, b;
public:
 void getvalue() {
  cout << "Enter the value of Complex Numbers 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-(complex ob) {
  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;
                result1 = obj1 - obj2;
                cout << "Input Values:\n";
                obj1.display();
                obj2.display();
                cout << "Result:";
                 result.display();
                result1.display();
                   getch();
                }
```

Output:

```
Enter the value of Complex Numbers a, b
4               5
Enter the value of Complex Numbers a, b
2               2
Input Values
4 + 5i
2 + 2i
Result
6 +   7i
2 +   3i
```

e) Explain operation on File.

Answer:

1. File Operations in C++ provides us with four different operations for file handling. They are:

open() – This is used to create a file.
read() – This is used to read the data from the file.
write() – This is used to write new data to file.
close() – This is used to close the file.
We will look into each of these and try to understand them better.

2. Opening files in C++

To read or enter data to a file, we need to open it first. This can be performed with the help of 'ifstream' for reading and 'fstream' or 'ofstream' for writing or appending to the file. All these three objects have open() function pre-built in them.

Syntax:
open(FileName , Mode);
Here:
FileName – It denotes the name of file which has to be opened.

Mode – There different mode to open a file and it explained in this article.

Mode	Description
iso::in	File opened in reading mode
iso::out	File opened in write mode
iso::app	File opened in append mode

3. Writing to File

. We will use fstream or ofstream object to write data into the file and to do so; we will use stream insertion operator (<<) along with the text enclosed within the double-quotes.

Syntax:

FileName<<"Insert the text here";
Program for Writing to File:

4. Reading from file in C++

Getting the data from the file is an essential thing to perform because without getting the data, we cannot perform any task. But don't worry, C++ provides that option too. We can perform the reading of data from a file with the CIN to get data from the user, but then we use CIN to take inputs from the user's standard console. Here we will use fstream or ifstream.

Syntax:
FileName>>Variable;
Content of FileName.txt:

5. Closing a file in C++

Closing a file is a good practice, and it is must to close the file. Whenever the C++ program comes to an end, it clears the allocated memory, and it closes the file. We can perform the task with the help of close() function.

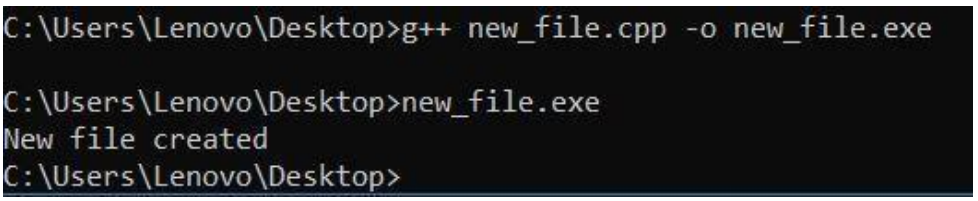
Syntax:

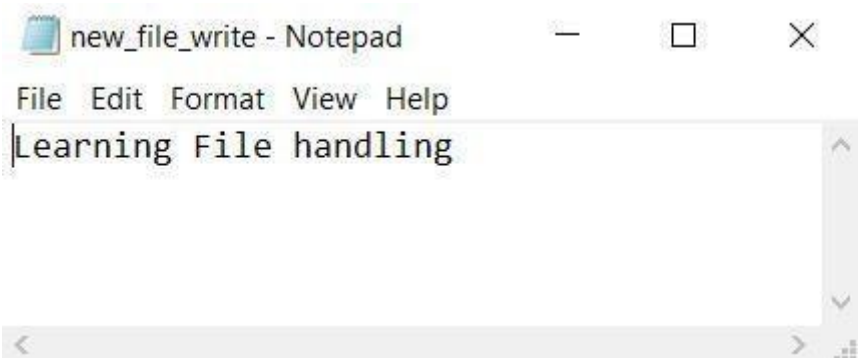
FileName.close();

Program to Close a File:

6. Program:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
fstream new_file;
new_file.open("new_file_write.txt",ios::out);
if(!new_file)
{
cout<<"File creation failed";
}
else
{
cout<<"New file created";
new_file<<"Learning File handling";    //Writing to file
new_file.close();
}
return 0;
}
```





Q .5 Short notes on any three of the following (5 Marks each) 15

a) Default Argument

Answer:

1. **Definition**: A default argument is a value in the function declaration automatically assigned by the compiler if the calling function does not pass any value to that argument.
2. Characteristics for defining the default arguments
   a. The values passed in the default arguments are not constant. These values can be overwritten if the value is passed to the function. If not, the previously declared value retains.
   b. During the calling of function, the values are copied from left to right.
   c. All the values that will be given default value will be on the right.
   d. Example
   e. Void function (int x, int y, int z = 0)
   f. Explanation - The above function is valid. Here z is the value that is predefined as a part of the default argument.
   g. Void function (int x, int z = 0, int y)
   h. Explanation - The above function is invalid. Here z is the value defined in between, and it is not accepted.
3. Program

```
#include<iostream>
using namespace std;
int sum(int x, int y, int z=0, int w=0) // Here values in the default arguments
{ // Both z and w are initialised to zero
return (x + y + z + w); // return sum of all parameter values
}
int main()
{
cout << sum(10, 15) << endl;
cout << sum(10, 15, 25) << endl;
cout << sum(10, 15, 25, 30) << endl;
return 0;
}
```

O/p:

25
50
80

4. **explanation:**

In the above program, we have called the sum function three times.

- Sum(10,15)

When this function is called, it reaches the definition of the sum. There it initializes x to 10 y to 15, and the rest values are zero by default as no value is passed. And all the values after sum give 25 as output.

- Sum(10, 15, 25)

When this function is called, x remains 10, y remains 15, the third parameter z that is passed is initialized to 25 instead of zero. And the last value remains 0. The sum of x, y, z, w, is 50 which is returned as output.

o Sum(10, 15, 25, 30)

In this function call, there are four parameter values passed into the function with x as 10, y as 15, z is 25, and w as 30. All the values are then summed up to give 80 as the output.

b) Standard Template Library

Answer:

1. The Standard Template Library (STL) is a set of template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators.
2. It is a generalized library and so, its components are parameterized. Working knowledge of template classes is a prerequisite for working with STL.
3. STL has 4 components:
   - Algorithms
   - Containers
   - Functions
   - Iterators
4. **Algorithms:** The header algorithm defines a collection of functions specially designed to be used on a range of elements. They act on containers and provide meansfor various operations for the contents of the containers**.**
   - Algorithm
   - Sorting
   - Searching
   - Important STL Algorithms
   - Useful Array algorithms
   - Partition Operations
   - Numeric
   - valarray class
5. **Containers: Containers** or container classes store objects and data. There are in total seven standards "first-class" container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

- Sequence Containers: implement data structures that can be accessed in a sequential manner.
- Container Adaptors: provide a different interface for sequential containers.
- Associative Containers: implement sorted data structures that can be quickly searched (O(log n) complexity).
- Unordered Associative Containers: implement unordered data structures that can be quickly searched

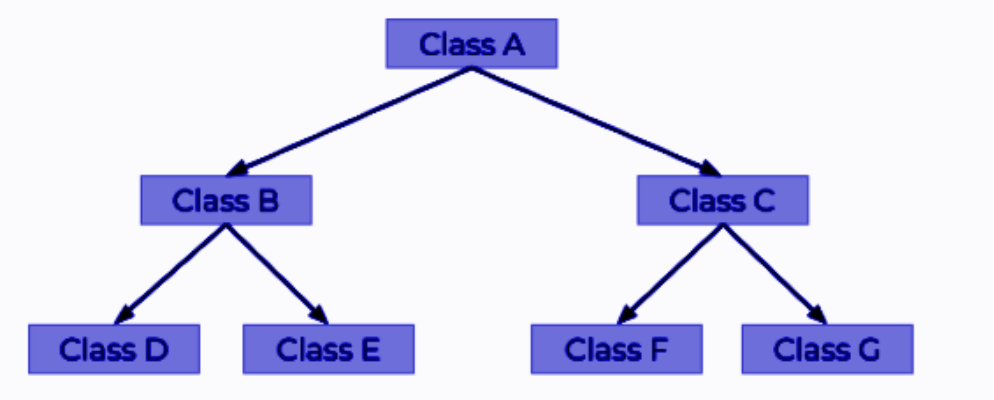
6. **Functions:** The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters tobe passed.
7. **Iterators :** As the name suggests, iterators are used for working upon a sequence of values.

c) Hierarchical Inheritance

Answer:

1) **Definition:** As the name defines, it is the hierarchy of classes. There is a single base class and multiple derived classes. Furthermore, the derived classes are also inherited by some other classes. Thus a tree-like structure is formed of hierarchy.

3.



Here class A is the base class. Class B and Class C are the derived classes of A.

Class D and Class E are derived classes of B. Class F and Class G are derived classes of C. Thus forming the structure of hierarchical inheritance.

4. Syntax

```
Class Parent
{
  statement(s);
};
Class Derived1: public Parent
{
  statement(s);
};
Class Derived2: public Parent
{
   statement(s);
};
class newderived1: public Derived1
   {
      statement(s);
   };
   class newderived2: public Derived2
   {
      statement(s);
};
```

5. **Explanation:** Class Parent is the base class and Derived1 and Derived2 are the class that inherit Parent class. Further, newderived1 is the class that inherits Dervied1, and newderived2 is the class that inherits the Derived2 class. There can be any number of base classes that are inherited by n number of derived classes.

d) Class and Object:

Answer:

Class:

1. A class is the building block that leads to Object-Oriented programming.
2. 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.
3. A class is like a blueprint for an object.
4. A Class is a user defined data-type which has data members and member functions.
5. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

6. Class Class_name

Objects:

1. When a class is defined, only the specification for the object is defined; no memory or storage is allocated.
2. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName;

3. Accessing data members and member functions:

- The data members and member functions of class can be accessed using the dot('.') operator with the object.
- For example if the name of object is obj and you want to access the member function with the name printName() then you will have to write obj.printName() .

4. // C++ program to demonstrate accessing of data members

```
#include <iostream.h>
using namespace std;
class Base {
   // Access specifier
public:
   // Data Members
   string name;
```

```
    // Member Functions()
    void printname() { cout << "Name is:" << name; }
};
int main()
{
    // Declare an object of class Base
    Base obj1;
    // accessing data member
    obj1.name = "ABC";
    // accessing member function
    obj1.printname();
    return 0;
}
```

Output
Name is: ABC

e) Pure Virtual Function.

Answer:

e) Pure Virtual Function

Answer:

1. A pure virtual function is a virtual function that has no definition within the class and initialized to zero is called pure virtual funcation
2. A pure virtual function is a "do nothing" function. Here "do nothing" means that it just provides the template, and derived class implements the function.
3. It can be considered as an empty function means that the pure virtual function does not have any definition relative to the base class.
4. Programmers need to redefine the pure virtual function in the derived class as it has no definition in the base class.
5. A class having pure virtual function cannot be used to create direct objects of its own. It means that the class is containing any pure virtual function then we cannot create the object of that class. This type of class is known as an abstract class.
6. Syntax

There are two ways of creating a virtual function:

```
virtual void display() = 0;
```

or

```
virtual void display() {}
```

7. // C++ program to calculate the area of a square and a circle

```
#include <iostream>
using namespace std;
// Abstract class
```

```
class Shape {
  protected:
   float dimension;
  public:
   void getDimension() {
       cin >> dimension;
   }
   // pure virtual Function
   virtual float calculateArea() = 0;
};
class Square : public Shape {
  public:
   float calculateArea() {
       return dimension * dimension;
   }
};
// Derived class
class Circle : public Shape {
  public:
   float calculateArea() {
       return 3.14 * dimension * dimension;
   }
};
int main() {
   Square square;
   Circle circle;
   cout << "Enter the length of the square: ";
   square.getDimension();
   cout << "Area of square: " << square.calculateArea() << endl;
   cout << "\nEnter radius of the circle: ";
   circle.getDimension();
   cout << "Area of circle: " << circle.calculateArea() << endl;
   return 0;
}
```

OUTPUT:

```
Enter the length of the square: 4
Area of square: 16
Enter radius of the circle: 5
Area of circle: 78.5
```

In this program, virtual float calculateArea() = 0; inside the Shape class is a pure virtual function.