|  |  |
| --- | --- |
| Topic – OOPS Fundamentals | |
| B1 | What Is class in Object Oriented Programming Language ?  Classes are user-defined data types that act as the blueprint for individual objects, attributes and methods. Objects are instances of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity. |
| B2 | What is an Object in Object Oriented Programming Language ?  An object is an abstract data type with the addition of polymorphism and inheritance. Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object". An object has state (data) and behavior (code). Objects can correspond to things found in the real world. |
| B3 | What Is Difference Between Class And Interface ? |
| B4 | What Is Method Overloading in Object Oriented Programming Language?  Method or function overloading allows a method with the same name to be declared more than once given that they have different input parameters.  For example, there is a method to calculate the area of shapes. There are different shapes such as circles and rectangles. In these two cases, if someone calls the function calculateArea, the result should be correct for the given shape, regardless of whether the shape is a circle or a rectangle. The declaration of this in a programming language in shown below:  Rectangle: calculateArea(*double length, double width*)  Circle:  calculateArea(*double radius*)  A great advantage of method overloading is that it allows a programmer to use the function appropriately without having to know the inner-workings of that method. |
| B5 | What Is Data hiding in Object Oriented Programming Language ?  Data hiding is an object-oriented programming (OOP) technique specifically used to hide internal object details (i.e., data members). Data hiding guarantees exclusive data access to class members only and protects and maintains object integrity by preventing intended or unintended changes and intrusions. |
| B6 | What are the differences between abstract classes and interfaces?  The Abstract class and Interface both are used to have abstraction. An abstract class contains an abstract keyword on the declaration whereas an Interface is a sketch that is used to implement a class. |
| B7 | What are the Virtual Function in Object Oriented Programming ?  A virtual function is a member function that you expect to be redefined in derived classes. 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.  // CPP program to illustrate  // concept of Virtual Functions  #include<iostream>  using namespace std;  class base {  public:  virtual void print()  {  cout << "print base class\n";  }  void show()  {  cout << "show base class\n";  }  };  class derived : public base {  public:  void print()  {  cout << "print derived class\n";  }  void show()  {  cout << "show derived class\n";  }  };  int main()  {  base \*bptr;  derived d;  bptr = &d;  // Virtual function, binded at runtime  bptr->print();  // Non-virtual function, binded at compile time  bptr->show();    return 0;  } |
| B8 | What is Constructor in Object Oriented Programming ?  Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.  Constructor does not have a return value, hence they do not have a return type.  The prototype of Constructors is as follows:  <class-name> (list-of-parameters);  Constructors can be defined inside or outside the class declaration:-  The syntax for defining the constructor within the class:  <class-name> (list-of-parameters) { // constructor definition }  The syntax for defining the constructor outside the class:  <class-name>: :<class-name> (list-of-parameters){ // constructor definition}  // defining the constructor within the class  #include <iostream>  using namespace std;  class student {  int rno;  char name[10];  double fee;  public:  student()  {  cout << "Enter the RollNo:";  cin >> rno;  cout << "Enter the Name:";  cin >> name;  cout << "Enter the Fee:";  cin >> fee;  }  void display()  {  cout << endl << rno << "\t" << name << "\t" << fee;  }  };  int main()  {  student s; // constructor gets called automatically when  // we create the object of the class  s.display();  return 0;  } |
| B9 | What is Abstract class in Object Oriented Programming?  In programming languages, an abstract class is a generic class (or type of object) used as a basis for creating specific objects that conform to its protocol, or the set of operations it supports.  An abstract class, in the context of Java, is a superclass that cannot be instantiated and is used to state or define general characteristics. An object cannot be formed from a Java abstract class; trying to instantiate an abstract class only produces a compiler error. The abstract class is declared using the keyword abstract.  Subclasses extended from an abstract class have all the abstract class's attributes, in addition to attributes specific to each subclass. The abstract class states the class characteristics and methods for implementation, thus defining a whole interface.  Abstract classes are useful when creating hierarchies of classes that model reality because they make it possible to specify an invariant level of functionality in some methods, but leave the implementation of other methods until a specific implementation of that class (a derived class) is needed. |
| B10 | What is Final Keyword in Object Oriented Programming?  You can use the final keyword to designate virtual functions that cannot be overridden in a derived class. You can also use it to designate classes that cannot be inherited. |
| B11 | What is Pure Virtual funtion in Object Oriented Programming?  C:\Users\Aura\Downloads\ClassDiagram_for_VirtualFunction.pngA pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived class if the derived class is not abstract. Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly. |
| B12 | What are Sealed Modifier in Object Oriented Programming?  Sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, this class cannot be inherited. In C#, the sealed modifier is used to declare a class as sealed. |
|  |  |
| B13 | What is Dynamic or run time Polymorphism in oops ?  Run-Time Polymorphism: Whenever an object is bound with the functionality at run time, this is known as runtime polymorphism. The runtime polymorphism can be achieved by method overriding. Java virtual machine determines the proper method to call at the runtime, not at the compile time. |
| B14 | What is Access Modifier in Object Oriented Programming ?  Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. |
| B15 | What is Friend Function in Object Oriented Programming?  A friend function in C++ is defined as a function that can access private, protected and public members of a class. The friend function is declared using the friend keyword inside the body of the class. Friend Function Syntax: class className { ... .. ... friend returnType functionName(arguments); ... .. ... }  #include<iostream>  using namespace std;  class B; //forward declaration.  class A  {  int x;  public:  void setdata (int i)  {  x=i;  }  friend void max (A, B); //friend function.  } ;  class B  {  int y;  public:  void setdata (int i)  {  y=i;  }  friend void max (A, B);  };  void max (A a, B b)  {  if (a.x >= b.y)  std:: cout<< a.x << std::endl;  else  std::cout<< b.y << std::endl;  }  int main ()  {  A a;  B b;  a. setdata (10);  b. setdata (20);  max (a, b);  return 0;  } |
| B16 | What is Overriding in Object Oriented Programming?  We have two classes: A child class Boy and a parent class Human. The Boy class extends Human class. Both the classes have a common method void eat() . Boy class is giving its own implementation to the eat() method or in other words it is overriding the eat() method. |
| B17 | What is the role of mutable storage class specifier?  The mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const . You cannot use the mutable specifier with names declared as static or const , or reference members. |

1 | P a g e

|  |  |
| --- | --- |
|  |  |
| B18 | Distinguish between shallow copy and deep copy ?  In Shallow copy, a copy of the original object is stored and only the reference address is finally copied. In Deep copy, the copy of the original object and the repetitive copies both are stored. |
| B19 | what is a Refrence variable in Object oriented Programming Language ?  A reference variable is a variable that points to an object of a given class, letting you access the value of an object. An object is a compound data structure that holds values that you can manipulate. A reference variable does not store its own values. |
| B20 | What is a Copy Constructor ?  A copy constructor is a member function that initializes an object using another object of the same class. In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously is known as a copy constructor. |
| B21 | What is this Pointer in Object oriented Programming Language ?  The this pointer is a pointer accessible only within the nonstatic member functions of a class , struct , or union type. It points to the object for which the member function is called. Static member functions don't have a this pointer. |
| I1 | Define Constructor and Destructor?   | S. No. | Constructors | Destructors | | --- | --- | --- | | 1 | Constructor is invoked to initialize the object of a class by the compiler. | Destructor is invoked when the instance is destroyed. | | 2 | Declared as Class\_Name(arguments if any){//constructor body}. | Declared as ~Class\_Name(){}. | | 3 | Constructor can receive parameters. | Destructor cannot accept any parameters. | | 4 | Constructor is used to initialize an object of the class and assign values to data members corresponding to the class. | While destructor is used to deallocate the memory of an object of a class. | | 5 | There can be multiple constructors for the same class. | In a class, there is always a single destructor. | | 6 | Constructor can be overloaded. | Destructor can’t be overloaded. | |
| I2 | How to Load Classes in Object Oriented Programming ?  A class is a blueprint for the object. We can think of a class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object. A class is defined in C++ using keyword class followed by the name of the class.The body of the class is defined inside the curly brackets and terminated by a semicolon at the end. |
| I3 | How to Call Parent Constructor ?  In order to run a parent constructor, a call to parent::\_\_construct() within the child constructor is required. If the child does not define a constructor then it may be inherited from the parent class just like a normal class method (if it was not declared as private). $obj = new OtherSubClass(); |
| I4 | Are Parent Constructor Called Implicitly When Create An Object Of Class?  A constructor cannot be called as a method. It is called when object of the class is created so it does not make sense of creating child class object using parent class constructor notation. |
| I5 | What Happen, If Constructor Is Defined As Private Or Protected?  If a constructor is declared as private, then its objects are only accessible from within the declared class. You cannot access its objects from outside the constructor class. |
| I6 | Define New-style Constructor & Old-style Constructor. Check which One Will Be Called?  Old-style constructors  That syntax is deprecated, and will result in an E\_DEPRECATED error but still call that function as a constructor. If both \_\_construct() and a same-name method are defined, \_\_construct() will be called.  First, the base constructor is called. Then, the base-class members are initialized in the order in which they appear in the class declaration. Finally, the derived constructor is called. |
| I7 | Create Abstract class and method?  We use the abstract keyword to create abstract classes and methods. An abstract method doesn't have any implementation (method body). A class containing abstract methods should also be abstract. We cannot create objects of an abstract class. |
| I8 | Define 3 types of visibility of data member & member function.  There are three types of Visibility modes:   1. Public Visibility mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.  **2. Protected Visibility mode:** If we derive a subclass from a Protected base class. Then both public member and protected members of the base class will become protected in the derived class.  **3. Private Visibility mode:** If we derive a subclass from a Private base class. Then both public member and protected members of the base class will become Private in the derived class. |
| I9 | Create a method which will never inherited ?  You cannot stop a derived class from inheriting all the methods of a base class. The best you can do is to make it a nonmember requiring the base object parameter. Then you would have to downcast the object before calling but you could still call it. |
|  |  |
| I10 | What is the difference between Abstract class and Interface?  The Abstract class and Interface both are used to have abstraction. An abstract class contains an abstract keyword on the declaration whereas an Interface is a sketch that is used to implement a class. |
| I11 | Create parent class for car and child class for car\_model and use car functionality in car\_model class ?.  #stdio<iostream>  class Vehicle  def initialize(vehicle\_name, vehicle\_color)  @vehicle\_name = vehicle\_name  @vehicle\_color = vehicle\_color  end  def description  puts 'This is a vehicle'  end  end  class Car < Vehicle  def description  puts 'This is a car'  end  end  class Bus < Vehicle  def display\_this  puts 'This is a bus'  end  end  # Creating objects  object1 = Car.new('Nissan', 'red')  object2 = Bus.new('Volvo', 'white')  object1.description  object2.description  object2.display\_this |

|  |  |
| --- | --- |
|  |  |
| I12 | Override the parent’s properties and methods in the child class ?  In the same way that the child class can have its own properties and methods, it can override the properties and methods of the parent class. When we override the class's properties and methods, we rewrite a method or property that exists in the parent again in the child, but assign to it a different value or code. |
| I13 | Prevent the child class from overriding the parent’s methods ?  In order to prevent the method in the child class from overriding the parent's methods, we can prefix the method in the parent with the final keyword. |
| I15 | Declare classes and methods as abstract ?  When you have to work with any method declaration in super class, you can do so by the declaration of any of the abstract methods in java as an abstract one. Abstract class in java example can be best defined as a subclass responsibility without any implementation in super class.  Declaring a class as abstract means that it cannot be directly instantiated, which means that an object cannot be created from it. That protects the code from being used incorrectly. Abstract classes require subclasses to further define attributes necessary for individual instantiation. |
| I16 | Can we have non abstract methods inside an abstract class? Explain With Example ?  It can include both abstract and non-abstract methods. It may also include constructors and static methods. It can have final methods, which prevent the subclass from changing the method's body. An abstract class in C++ is one that has at least one pure virtual function by definition.  class Shapes {  public:  virtualint Area() = 0; // Pure virtual function is declared as follows.           // Function to set width.  voidsetval\_width(int w) {  width = w;           }           // Function to set height.  voidsetval\_height(int h) {  height = h;           }  protected:  int width;  int height;  }; |
| I17 | How to create child classes from an abstract class?  An abstract class is a class that contains at least one pure virtual function, and these classes cannot be instantiated.  Abstract Classes came from the idea of abstraction. Before we dive into technicalities, let’s first consider a scenario where you might use this concept. Abstract Class Example Let’s say we are making a calculator which returns the perimeter of the shape we put in. Think of what kind of code you would write for such a calculator. You might start with some initial shapes and hardcode the perimeter by making separate functions inside the Shape class.  This will work, but OOP suggests that we should try to stay close to real-world logic. So, we can make a class Shape as a parent class and then make separate classes like Square and Rectangle as child classes. This will make the code easier to maintain, and if you plan to add anything new in the future, then you can add that in the child class. To implement this feature, we need to use Abstract Classes. Abstract classes in C++ need at least one **pure virtual function** in a class. The classes that inherit the abstract class must define the pure virtual function; otherwise, the subclass will become an abstract class.  class Shape {  public:  // All the functions of both square and rectangle are clubbed together in a single class.  void width(int w) {  shape\_width = w;  }  void height(int h) {  shape\_height = h;  }  int perimeterOfSquare(int s) {  return 4 \* s;  }  int perimeterOfRectange(int l, int b) {  return 2 \* (l + b);  }    protected:  int shape\_width;  int shape\_height;  }; |
| I18 | What are Magic Methods/Functions? List them in OOPS ?  The following method names are considered magical: \_\_construct(), \_\_destruct(), \_\_call(), \_\_callStatic(), \_\_get(), \_\_set(), \_\_isset(), \_\_unset(), \_\_sleep(), \_\_wakeup(), \_\_serialize(), \_\_unserialize(), \_\_toString(), \_\_invoke(), \_\_set\_state(), \_\_clone(), and \_\_debugInfo().  Magic methods are special methods in python that have double underscores (dunder) on both sides of the method name. Magic methods are predominantly used for operator overloading. |
| I19 | How can we used Virtual Function write an examples in oops ?  A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the 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](https://www.geeksforgeeks.org/polymorphism-in-c/) * Functions are declared with a virtual keyword in base class. * The resolving of function call is done at runtime.   Rules for Virtual Functions   1. Virtual functions cannot be static. 2. A virtual function can be a friend function of another class. 3. Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism. 4. The prototype of virtual functions should be the same in the base as well as derived class. 5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used. 6. A class may have [virtual destructor](https://www.geeksforgeeks.org/virtual-destructor/) but it cannot have a virtual constructor.   Compile time (early binding) VS runtime (late binding) behavior of Virtual Functions  Consider the following simple program showing runtime behavior of virtual functions.  // CPP program to illustrate  // concept of Virtual Functions  #include<iostream>  using namespace std;  class base {  public:  virtual void print()  {  cout << "print base class\n";  }  void show()  {  cout << "show base class\n";  }  };  class derived : public base {  public:  void print()  {  cout << "print derived class\n";  }  void show()  {  cout << "show derived class\n";  }  };  int main()  {  base \*bptr;  derived d;  bptr = &d;  // Virtual function, binded at runtime  bptr->print();  // Non-virtual function, binded at compile time  bptr->show();    return 0;  } |
| I20 | How can we Used various type of Constructor write an Examples in oops ?  A constructor is a special type of function with no return type. Name of constructor should be same as the name of the class. We define a method inside the class and constructor is also defined inside a class. A constructor is called automatically when we create an object of a class. We can’t call a constructor explicitly. Let us see the types of constructor.  Remarks on Constructor   1. Constructor is called automatically when we create an object of the class. 2. Name of constructor should be same as the name of the class. 3. Constructor does not return any value. 4. Constructor should have a public access modifier.   Constructor Types   1. Default Constructor 2. Parameterized Constructor 3. Copy Constructor 4. Static Constructor 5. Private Constructor   public class Adminclass  {  string userId = string.Empty;  string password = string.Empty;  //Default constructor, having no any parameter  public Adminclass()  {  userId = "shr";  password = "Pass";  }  } |
| I21 | Which Constructor have no Parameter write an Examples in oops ?  A constructor with no parameters is known as a default constructor. In the example above, Wall() is a default constructor.  #include <iostream>  using namespace std;  class DemoDC {     private:     int num1, num2 ;     public:     DemoDC() {        num1 = 10;        num2 = 20;     }     void display() {        cout<<"num1 = "<< num1 <<endl;        cout<<"num2 = "<< num2 <<endl;     }  };  int main() {     DemoDC obj;     obj.display();     return 0;  } |
| I22 | How to Secured Internal Data using Encapsulation Write a Example in oops  ?  Data Encapsulation is an Object Oriented Programming concept that bind a group of related properties, functions, and other members are treated as a single unit. Class is the best example of Data Encapsulation. It sometimes referred to as data hiding that prevents the user to access the implementation details.  // Program to calculate the area of a rectangle  #include <iostream>  using namespace std;  class Rectangle {  public:  // Variables required for area calculation  int length;  int breadth;  // Constructor to initialize variables  Rectangle(int len, int brth) : length(len), breadth(brth) {}  // Function to calculate area  int getArea() {  return length \* breadth;  }  };  int main() {  // Create object of Rectangle class  Rectangle rect(8, 6);  // Call getArea() function  cout << "Area = " << rect.getArea();  return 0;  } |
| A1 | Write a Programme to create a Class in OOPS ?  // Program to illustrate the working of  // objects and class in C++ Programming  #include <iostream>  using namespace std;  // create a class  class Room {  public:  double length;  double breadth;  double height;  double calculateArea() {  return length \* breadth;  }  double calculateVolume() {  return length \* breadth \* height;  }  };  int main() {  // create object of Room class  Room room1;  // assign values to data members  room1.length = 42.5;  room1.breadth = 30.8;  room1.height = 19.2;  // calculate and display the area and volume of the room  cout << "Area of Room = " << room1.calculateArea() << endl;  cout << "Volume of Room = " << room1.calculateVolume() << endl;  return 0;  } |
| A2 | Write a Programme to Create a Object in OOPS ?  // Program to illustrate the working of  // objects and class in C++ Programming  #include <iostream>  using namespace std;  // create a class  class Room {  public:  double length;  double breadth;  double height;  double calculateArea() {  return length \* breadth;  }  double calculateVolume() {  return length \* breadth \* height;  }  };  int main() {  // create object of Room class  Room room1;  // assign values to data members  room1.length = 42.5;  room1.breadth = 30.8;  room1.height = 19.2;  // calculate and display the area and volume of the room  cout << "Area of Room = " << room1.calculateArea() << endl;  cout << "Volume of Room = " << room1.calculateVolume() << endl;  return 0;  } |
| A3 | Write a Programme to Create an Abstract Class In OOPS ?  // Abstract class  abstract class Animal  {  // Abstract method (does not have a body)  public abstract void animalSound();  // Regular method  public void sleep()  {  Console.WriteLine("Zzz");  }  }  // Derived class (inherit from Animal)  class Pig : Animal  {  public override void animalSound()  {  // The body of animalSound() is provided here  Console.WriteLine("The pig says: wee wee");  }  }  class Program  {  static void Main(string[] args)  {  Pig myPig = new Pig(); // Create a Pig object  myPig.animalSound(); // Call the abstract method  myPig.sleep(); // Call the regular method  }  } |
|  |  |
| A4 | Write a Programme to Create a Encapsulation in OOPS ?  #include <iostream> using namespace std;  class Employee {   private:     // Private attribute     int salary;    public:     // Setter     void setSalary(int s) {       salary = s;     }     // Getter     int getSalary() {       return salary;     } };  int main() {   Employee myObj;   myObj.setSalary(50000);   cout << myObj.getSalary();   return 0; } |
| A5 | How to Implement Access Modifier Write a Proggramme in OOPS ?  Access modifiers are used to implement an important aspect of Object-Oriented Programming known as [Data Hiding](https://practice.geeksforgeeks.org/problems/what-is-data-hiding). Consider a real-life example:  The Research and Analysis Wing (R&AW), having 10 core members, has come into possession of sensitive confidential information regarding national security. Now we can correlate these core members to data members or member functions of a class, which in turn can be correlated to the R&A Wing. These 10 members can directly access the confidential information from their wing (the class), but anyone apart from these 10 members can’t access this information directly, i.e., outside functions other than those prevalent in the class itself can’t access the information (that is not entitled to them) without having either assigned privileges (such as those possessed by a friend class or an inherited class, as will be seen in this article ahead) or access to one of these 10 members who is allowed direct access to the confidential information (similar to how private members of a class can be accessed in the outside world through public member functions of the class that have direct access to private members). This is what data hiding is in practice.  Access Modifiers or Access Specifiers in a [class](https://www.geeksforgeeks.org/c-classes-and-objects/) are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can’t be directly accessed by the outside functions. There are 3 types of access modifiers available in C++:   1. Public 2. Private 3. Protected   // C++ program to demonstrate public  // access modifier  #include<iostream>  using namespace std;  // class definition  class Circle  {  public:  double radius;    double compute\_area()  {  return 3.14\*radius\*radius;  }    };  // main function  int main()  {  Circle obj;    // accessing public datamember outside class  obj.radius = 5.5;    cout << "Radius is: " << obj.radius << "\n";  cout << "Area is: " << obj.compute\_area();  return 0;  }  // C++ program to demonstrate private  // access modifier  #include<iostream>  using namespace std;  class Circle  {  // private data member  private:  double radius;    // public member function  public:  double compute\_area()  { // member function can access private  // data member radius  return 3.14\*radius\*radius;  }    };  // main function  int main()  {  // creating object of the class  Circle obj;    // trying to access private data member  // directly outside the class  obj.radius = 1.5;    cout << "Area is:" << obj.compute\_area();  return 0;  }  // C++ program to demonstrate private  // access modifier  #include<iostream>  using namespace std;  class Circle  {  // private data member  private:  double radius;    // public member function  public:  void compute\_area(double r)  { // member function can access private  // data member radius  radius = r;    double area = 3.14\*radius\*radius;    cout << "Radius is: " << radius << endl;  cout << "Area is: " << area;  }    };  // main function  int main()  {  // creating object of the class  Circle obj;    // trying to access private data member  // directly outside the class  obj.compute\_area(1.5);      return 0;  } |

|  |  |
| --- | --- |
|  |  |
| A6 | Write a Proggramme of Copy Constructor in OOPS ?  // C++ program to demonstrate the working  // of a COPY CONSTRUCTOR  #include <iostream>  using namespace std;  class Point {  private:  int x, y;  public:  Point(int x1, int y1)  {  x = x1;  y = y1;  }  // Copy constructor  Point(const Point& p1)  {  x = p1.x;  y = p1.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  // Let us access values assigned by constructors  cout << "p1.x = " << p1.getX()  << ", p1.y = " << p1.getY();  cout << "\np2.x = " << p2.getX()  << ", p2.y = " << p2.getY();  return 0;  } |
| A7 | Declare and implement an interface and implement more than one interface in the same class.  To declare a class that implements an interface, you include an implements clause in the class declaration. Your class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class.  // C++ program to implement  // Interface  #include <iostream>  #include <string>  using namespace std;  // Interface(Abstract class  // with pure virtual function)  class GFG  {  public:  virtual string returnString() = 0;  };  class child : public GFG  {  public:  string returnString()  {  return "GeeksforGeeks";  }  };  // Driver code  int main()  {  child childObj;  GFG\* ptr;  ptr = &childObj;  cout << ptr->returnString();  return 0;  } |
| A8 | How to implement the polymorphism principle in OOPS ?  In order to implement the polymorphism principle, we can choose between abstract classes and interfaces. In order to ensure that the classes do implement the polymorphism principle, we can choose between one of the two options of either abstract classes or interfaces. |
| A9 | Explain Scope resolution operator with example in OOPS ?.  The scope resolution operator ( :: ) is used for several reasons. For example: If the global variable name is same as local variable name, the scope resolution operator will be used to call the global variable. It is also used to define a function outside the class and used to access the static variables of class. |
| A10 | How to Access child class property to Parent class write an Programme in OOPS ?  The only unusual aspect is that, within child class method definitions, you can't directly access parent class instance variables. For example, if the parent had a height instance variable, child class method definitions wouldn't be able to access this directly.  The reference holding the child class object reference will not be able to access the members (functions or variables) of the child class. This is because the parent reference variable can only access fields that are in the parent class. |
| A11 | Write a Programme of how to define Interface In OOPS ?  #include <iostream>  using namespace std;  class Geo  {  public:  virtual void angle()=0;  };  class Acute: Geo  {  public:  void angle ()  {  cout <<"making it less than 90 degree" <<endl;  }  };  class Right: Geo  {  public:  void angle ()  {  cout <<"Making exactly 90 degree" <<endl;  }  };  int main () {  Acute r;  Right rir;  r.angle();  rir.angle();  return 0;  } |
| A12 | Write a Programme of how to define a Constructor in OOPS ?  // defining the constructor within the class  #include <iostream>  using namespace std;  class student {  int rno;  char name[10];  double fee;  public:  student()  {  cout << "Enter the RollNo:";  cin >> rno;  cout << "Enter the Name:";  cin >> name;  cout << "Enter the Fee:";  cin >> fee;  }  void display()  {  cout << endl << rno << "\t" << name << "\t" << fee;  }  };  int main()  {  student s; // constructor gets called automatically when  // we create the object of the class  s.display();  return 0;  } |
| A13 | How to define a default constructor write a Programme in OOPS ?  A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values. If no user-defined constructor exists for a class A and one is needed, the compiler implicitly declares a default parameterless constructor A::A() . |

4 | P a g e