**Question Bank- OOPS**

**(BCS10T1006)**

Q1. Differentiate between data abstraction and data encapsulation.

| **Parameter** | **Abstraction** | **Encapsulation** |
| --- | --- | --- |
| Use for | Abstraction solves the problem and issues that arise at the design stage. | Encapsulation solves the problem and issue that arise at the implementation stage. |
| Focus | Abstraction allows you to focus on what the object does instead of how it does it | Encapsulation enables you to hide the code and data into a single unit to secure the data from the outside world. |
| Implementation | You can use abstraction using Interface and Abstract Class. | You can implement encapsulation using Access Modifiers (Public, Protected & Private.) |
| Focuses | Focus mainly on what should be done. | Focus primarily on how it should be done. |
| Application | During design level. | During the Implementation level. |

Q2. Discuss the use of public, private and protected access specifiers and their visibility in the class.

public

Data members or Member functions which are declared as public can be accessed anywhere in the program (within the same class, or outside of the class).

protected

Data members or Member functions which are declared as protected can be accessed in the derived class or within the same class.

private

Data members of Member functions which are declared as private can be accessed within the same class only i.e. the private Data members or Member functions can be accessed within the public member functions of the same class.

Q3. Discuss default constructor and parameterized constructor with the help of an example in C++.

C++ constructors are special member functions which are created when the object is created or defined and its task is to initialize the object of its class. It is called **constructor** because it constructs the values of data members of the class.

## **Default constructor**

If no constructor is defined in the class then the compiler automatically creates one for the program. This constructor which is created by the compiler when there is no user defined constructor and which doesn’t take any parameters is called **default constructor**.

## **Parameterized constructor**

To put it up simply, the constructor that can take arguments are called parameterized constructor.

In practical programs, we often need to initialize the various data elements of the different object with different values when they are created. This can be achieved by passing the arguments to the constructor functions when the object is created.

Q4. Write down the use of destructor in C++.

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc

Q5. What is the need of constructor? How it is different from the member function?

**What is constructor?**   
A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) is created. It is special member function of the class because it does not have any return type.  
**How constructors are different from a normal member function?**

A constructor is different from normal functions in following ways:

* + Constructor has same name as the class itself
  + Default Constructors don’t have input argument however, Copy and Parameterized Constructors have input arguments
  + Constructors don’t have return type
  + A constructor is automatically called when an object is created.
    - It must be placed in public section of class.
  + If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

Q6. What is a static data member? How they are used in static functions? Explain with suitable illustrations.

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are:

* 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.
* It is initialized before any object of this class is being created, even before main starts.
* It is visible only within the class, but its lifetime is the entire program

Q7. Define class and objects.

**Class:** A class in C++ is the building block that leads to Object-Oriented programming. 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 C++ class is like a blueprint for an object.  
For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Q8. What do you mean by dynamic binding? How it is useful in OOP?

It is the process of connecting the method call to the specific method at runtime. It is also called as late binding or runtime binding.

The dynamic binding can be acheive by method overriding. The compiler decides which method should be called for a particular call at runtime i.e. the method remains unknown until runtime.

Q9. Explain the use of friend function with the help of suitable example.

## friend Function in C++

A **friend function** can access the **private** and **protected** data of a class. We declare a friend function using the friend keyword inside the body of the class.

class className {

... .. ...

friend returnType functionName(arguments);

... .. ...

}

### ****Use of Friend function in C++****

As discussed, we require friend functions whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

To understand this better, let us consider two classes: Tokyo and Rio. We might require a function, metro(), to access both these classes without any restrictions. Without the friend function, we will require the object of these classes to access all the members. Friend functions in c++ help us avoid the scenario where the function has to be a member of either of these classes for access.

Q10. What is the need of overloading operators and functions?

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **function overloading** and **operator overloading** respectively.

An overloaded declaration is a declaration that is declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded **function** or **operator**, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

Q11. How do we invoke constructor? Can we have more than one constructor in a class? If yes, explain the need for such a situation.

 constructor is not a function, so we don't invoke them as we would a function. Constructors are similar to functions in that they both accept arguments and can be overloaded to provide a choice of construction methods. However, unlike a function, there is no return value (not even void).

We invoke a constructor by instantiating an object of the class. The name of the class serves as the class constructor. Where there is a choice of constructor methods, the specific arguments we pass will determine which constructor is invoked.

Unless the class specifies otherwise, objects can be constructed with or without the new operator. We use the new operator only when we wish to construct an anonymous object on the heap. The new operator will return a pointer to the anonymous object (or a nullptr if object construction fails).

Constructors are the only way to create an object of that class. Classes may have a lot of different instance variables. Some of these fields may be mandatory for Object to function while others may be optional. Constructors are only way to make user create a object with right values. When all are mandatory by design then constructors are created to pass all those arguments; but in other cases, the design of the class may be flexible enough to allow the user of the class to choose how many parameters need to be passed and provide sensible default/initialization values for the rest. We will understand this by some dumb example :-

1) We want to represent an Employee class

2) Employee need to have name, id and salary

3) Employee can have or may not have a company car

Q12. Write down the example to overload unary and binary operators in C++.

Unary Operator Overloading

As the name suggests, Unary operators operate on single operand or data.

Following are the examples of Unary operators:

* Unary minus ( – ) operator
* Logical not ( ! ) operator
* Decrement ( — ) and Increment ( ++ ) operator

**Example:**   
Assume that class Distance takes two member object i.e. feet and inches, create a function by which Distance object should decrement the value of feet and inches by 1 (having single operand of Distance Type).

**Note :** If increment/decrement operators are used before variable, they are called prefix operators i.e ++x. And if increment /decrement operators are used after variable, they are called postfix operators i.e x++.

**Overloading Binary Operator**: In binary operator overloading function, there should be one argument to be passed. It is overloading of an operator operating on two operands.  
Let’s take the same example of class Distance, but this time, add two distance objects.

Q13. State the use of scope resolution operator in C++.

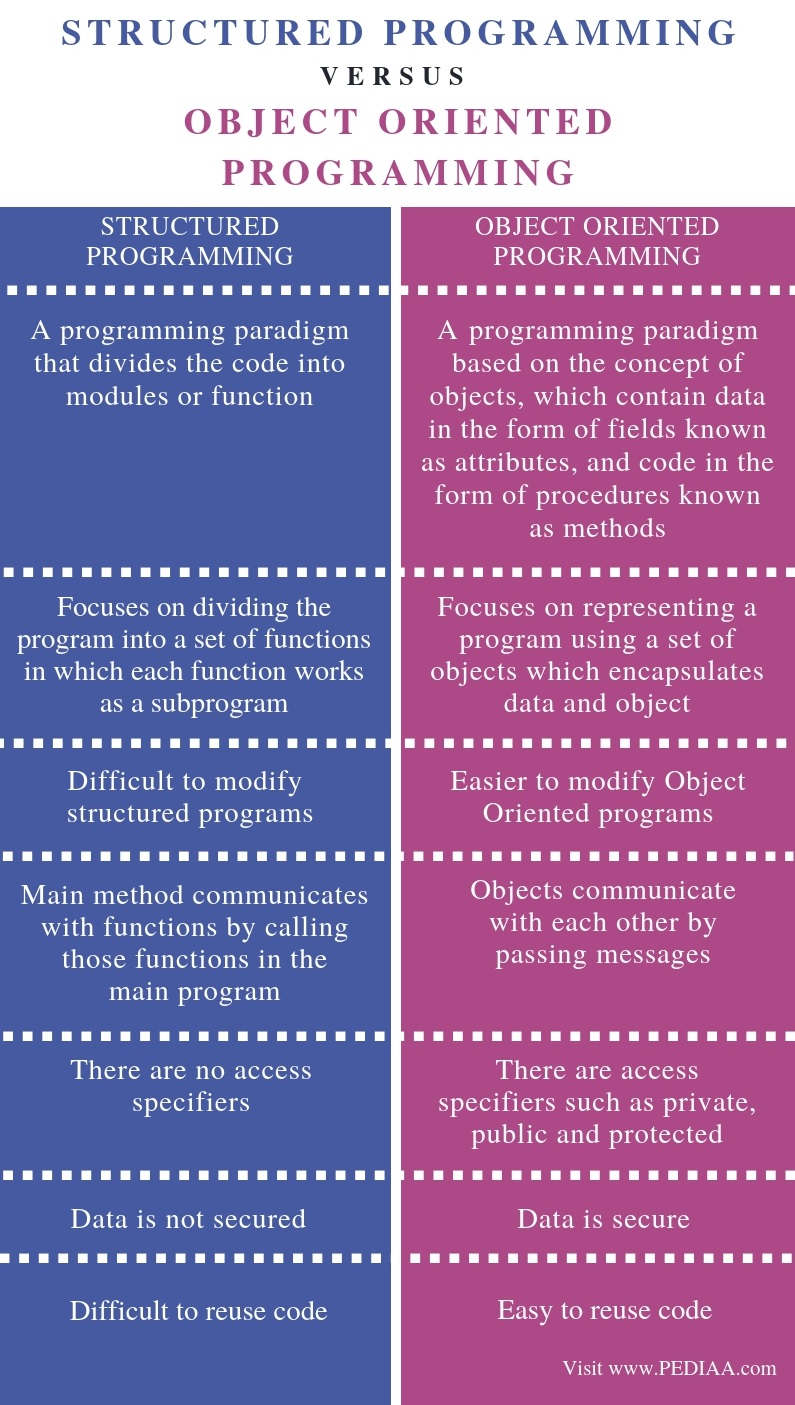
The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

For example, when the global and local variable or function has the same name in a program, and when we call the variable, by default it only accesses the inner or local variable without calling the global variable. In this way, it hides the global variable or function. To overcome this situation, we use the scope resolution operator to fetch a program's hidden variable or function.

Uses of the scope resolution Operator

1. It is used to access the hidden variables or member functions of a program.
2. It defines the member function outside of the class using the scope resolution.
3. It is used to access the static variable and static function of a class.
4. The scope resolution operator is used to override function in the Inheritance.

Q14. Compare and contrast the structured programming and object oriented programming.



Q15. What is a dynamic constructor? Explain with suitable example.

When allocation of memory is done dynamically using dynamic memory allocator [new](https://www.geeksforgeeks.org/new-and-delete-operators-in-cpp-for-dynamic-memory/) in a [constructor](https://www.geeksforgeeks.org/constructors-c/), it is known as **dynamic constructor**. By using this, we can dynamically initialize the objects.

#include <iostream>

using namespace std;

class geeks {

const char\* p;

public:

// default constructor

geeks()

{

// allocating memory at run time

p = new char[6];

p = "geeks";

}

void display()

{

cout << p << endl;

}

};

int main()

{

geeks obj;

obj.display();

}**Explanation**: In this we point data member of type **char** which is allocated memory dynamically by **new** operator and when we create dynamic memory within the constructor of class this is known as dynamic constructor.

Q16. Define a structure that represents Fruit with properties fruit name, fruit type, fruit color. Write a program that accepts data of four fruits and displays the results.

#include <iostream>

using namespace std;

//Defining a struct

struct Fruit

{

  char name[50], type[50], color[50];

  int roll;

} f[4];

int main()

{

  cout << "Enter Fruit and fruit properties: " << endl;

  // accepts data of four fruits

  for(int i = 0; i < 4; ++i)

  {

    f[i].roll = i+1;

    cout << "For roll number" << f[i].roll << "," << endl;

    cout << "Enter fruit name: ";

    cin >> f[i].name;

    cout << "Enter fruit type: ";

    cin >> f[i].type;

    cout << "Enter fruit color: ";

    cin >> f[i].color;

    cout << endl;

  }

  cout << "Displays the results: " << endl;

  //Displays the results

  for(int i = 0; i < 4; ++i)

  {

    cout << "\nRoll number: " << i+1 << endl;

    cout << "Fruit Name: " << f[i].name << endl;

    cout << "Fruit type: " << f[i].type << endl;

    cout << "Fruit color: " << f[i].color << endl;

  }

  return 0;

}

Q17. Compare and Contrast late binding and early binding.

|  |  |
| --- | --- |
| Early Binding vs Late Binding | |
| The process of using the class information to resolve method calling that occurs at compile time is called Early Binding. | The process of using the object to resolve method calling that occurs at run time is called the Late Binding. |
| **Time of Binding** | |
| Early Binding happens at compile time. | Late Binding happens at run time. |
| **Functionality** | |
| Early Binding uses the class information to resolve method calling. | Late Binding uses the object to resolve method calling. |
| **Synonyms** | |
| Early Binding is also known as static binding.. | Late Binding is also known as dynamic binding. |
| **Occurrence** | |
| Overloading methods are bonded using early binding. | Overridden methods are bonded using late binding. |
| **Execution Speed** | |
| Execution speed is faster in early binding. | Execution speed is lower in late binding. |

Q18. What do you mean by implicit and explicit call of constructor? Explain with example.

Q19. Write a C++ program to overload area () function to calculate area of shapes like triangle, square, circle.