OOP Question Bank

1. Define Object Oriented Programming. (OOP)

   Object-oriented programming (OOP) is a style of programming characterized by the identification of classes of objects closely linked with the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods. It is a technique based on a mathematical discipline, called "abstract data types," for storing data with the procedures needed to process that data. OOP offers the potential to evolve programming to a higher level of abstraction.

2. List the main features of OOP.

   Inheritance
   Encapsulation
   Abstraction
   Polymorphism
   Method Overriding
   Method Overloading
   Objects
   Classes

3. Define the class

   In object-oriented programming, a class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).

4. Define polymorphism.

   Polymorphism is one of the core concepts of object-oriented programming (OOP) and describes situations in which something occurs in several different forms. In computer science, it describes the concept that you can access objects of different types through the same interface. Each type can provide its own independent implementation of this interface.

5. List the application of OOP.

   Computer graphic applications

CAD/CAM software
Object-oriented database
User interface design such as windows
Real-time systems
Simulation and modeling
Artificial intelligence and expert systems

6. Define Object.
   In OOP, Objects are the things you think about first in designing a program and they are also the units of codes that are eventually derived from the process.
7. List some of the special properties of constructor.
   -A constructor name must be the same as that of its class name -Constructor cannot be inherited.
   -Constructor cannot be static
   -Constructor cannot be virtual
   -Constructor are called automatically when the objects are created
   -Constructor can have default arguments as other c++ functions

8. Define member function.
   Member functions are operators and functions that are declared as members of a class. Member functions do not include operators and functions declared with the friend specifier. These are called friends of a class.

9. Define destructor.
   A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a cell to delete. A destructor has the same name as the class, preceded by tilde (~).

10. Define pointer.
    A pointer is a variable that stores the memory address of an object. Pointers are used extensively in both C and C++ for three main

purposes: to allocate new objects on the heap, to pass functions to other functions, to iterate over elements in arrays or other data structures.

11. Explain the constructor with its type.

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.

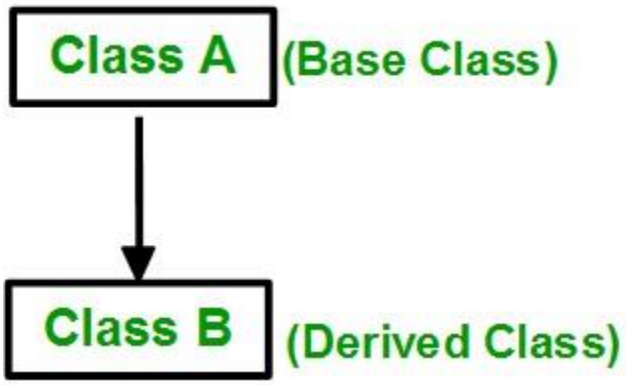
```
<class-name> (list-of-parameters);
```

Types of Constructors

1. Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument con
2. Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you wouldto any other function.structure.
3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class. A detailed article on Copy Constructor.

12. Explain the single inherence with example.



Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.

Syntax:

class subclass_name :
access_mode base_class
{

```
   // body of subclass
    };
#include<iostream> using
namespace std;
class Vehicle {
public:
        Vehicle
        () {
        cout << "This is a Vehicle\n";
        }
};
class Car : public Vehicle {
};
int
main() {
        Car obj;
        return 0;
}
```

13. Explain the concept of operator overloading.

In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

Example:

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

```
        res.real = real + obj.real; res.imag
        = imag + obj.imag;
        return res;
    }
    void print() { cout << real << " + i" << imag << '\n'; }
};
int main()
{
    Com c1(10, 5), c2(2, 4);
    Com c3 = c1 + c2;
    c3.print();
}
```

14. Explain the accessibility mode.

Data hiding is an important concept of Object-Oriented Programming, implemented with these Access modifiers' help. It is also known as the Access Specifier. Access Specifiers in a class decide the accessibility of the class members, like variables or methods in other classes.

1. Public Access Specifier:

This keyword is used to declare the functions and variables public, and any part of the entire program can access it. The members and member methods declared public can be accessed by other classes and functions. The public members of a class can be accessed from anywhere in the program using the (.) with the object of that class.

2. Private Access Specifiers:

The private keyword is used to create private variables or private functions. The private members can only be accessed from within the class. Only the member functions or the friend functions are allowed to access the private data of a class or the methods of a class.

3. Protected Access Specifiers:

The protected keyword is used to create protected variables or protected functions. The protected members can be accessed

within and from the derived/child class.

Example:

```
#include <iostream> using
namespace std;
class MyClass {
  public:
   int x;
};
int main() {
  MyClass myObj;

  myObj.x = 15; cout <<

  myObj.x; return 0;
```

15. Explain the concept of base class and derived class.

In the world of object-oriented programming languages, both base and derived classes play an important role. The base class is the existing class whereas the derived class is one that acquires the properties of a base class. There are many differences between them, let's explore some major differences between a base class and a derived class.

What is a Base Class?

In an object-oriented programming language, a base class is an existing class from which the other classes are determined and properties are inherited. It is also known as a superclass or parent class. In general, the class which acquires the base class can hold all its members and some further data as well. Syntax: Class base_classname{ … }.

What is a Derived Class?

A derived class is a class that is constructed from a base class or an existing class. It has a tendency to acquire all the methods and properties of a base class. It is also known as a subclass or child class.

Syntax: Class derived_classname : access_mode base_class_name
{ … }.

Example:

```
#include<iostream> using
namespace std;
class Vehicle {
public:
        Vehicle()
        {
        cout << "This is a Vehicle\n";
        }
};
class Car : public Vehicle {
};
int
main() {
        Car obj;
        return 0;
}
```

16. Explain the pointer in detail.

ANS:

Pointers are symbolic representations of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. Iterating over elements in arrays or other data structures is one of the main use of pointers.

The address of the variable you're working with is assigned to the pointer variable that points to the same data type (such as an int or string).

Syntax:

```
datatype *var_name;
int *ptr;   // ptr can point to an address which holds int data // C++
```

program to illustrate Pointers

```
#include <bits/stdc++.h>
using namespace std; void
geeks()
```

```
{ int var =
    20; int*
    ptr; ptr =
    &var;
    cout <<
    "Value at
    ptr = " <<
    ptr <<
    "\n"; cout
    << "Value
    at var = "
    << var <<
    "\n"; cout
    << "Value
    at *ptr =
    " << *ptr
    << "\n";
} int
main()
{
geeks
();
return
0; }
```

Output

Value at ptr = 0x7 e454c08cc
Value at var = 20
Value at *ptr = 20

17. Write a program to demonstrate friend function in C++.

    ANS:

    C++ Friend function

    If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function. For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

Program

```
#include <iostream> using namespace std; class base { private: int
private_variable;
protected:
    int protected_variable;
public:
    base()
    { private_variable = 10;
      protected_variable = 99;
    } friend void friendFunction(base& obj);
};
void friendFunction(base& obj)
{ cout << "Private Variable: " << obj.private_variable
      << endl; cout << "Protected Variable: " <<
    obj.protected_variable;
} int
main()
{ base object1;
    friendFunction(object1);
    return 0;
}
```

OUTPUT:

```
Private Variable: 10
Protected Variable: 99
```

18. Write a program to demonstrate operator overloading.

ANS:

```
#include <iostream> using
namespace std; class
OperatorOverload {
private:
    int x;
public:
    OperatorOverload() : x(10) {}
    void operator ++() { x = x + 2;
    } void Print() { cout << "The
    Count is: " << x;
        }
};
int main() {
    OperatorOverload ov;
    ++ov;
    ov.Prin
t(); return
0; }
```

19. Differentiate public and private inheritance with the help of example.

ANS:

Public	**Private**
All the class members declared under public will be available to everyone.	The class members declared as private can be accessed only by the functions inside the class.

The data members and member functions declared public can be accessed by other classes too.	Only the member functions or the friend functions are allowed to access the private data members of a class.
The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.	They are not allowed to be accessed directly by any object or function outside the class.

Example:

```
#include <iostream> using

namespace std;

class Circle {
private: double

    radius;

public:
    void compute_area(double r)
    { radius =

        r;

        double area = 3.14 * radius * radius; cout

        << "Radius is: " << radius << endl; cout <<

        "Area is: " << area;

    }

};
```

```
int
main() {
    Circle obj;
    obj.compute_area(1.5); return
    0;
}
```

20. Explain the accessing array using pointer.

A Pointer is a variable that stores the memory location or address of an object or variable. In other words, pointers reference a memory location, and obtaining the value stored at that memory location is known as dereferencing the pointer.

An Array is the collection of homogeneous elements stored in contiguous memory blocks. So, elements in an array can be accessed using a pointer.

Access elements using Pointer

Pointer has the capability to store the address, So, we can store the address of the first element of the array and then traverse the pointer till we reach the end element.

Methods to store the address of the first elements of the array are mentioned below:

```
int *ptr = arr;
int *ptr =
&arr[0];
```

Example:

```
#include <iostream> using
namespace std;
int main()
{
    int arr[5] = { 6, 2, 5, 7, 4 };

    int* ptr = &arr[0];
```

```
    for (int i = 0; i < 5; i++) {
       cout << "Value of" << i << " arr[" << i << "] is "
                << *(ptr + i) << endl; cout <<
       "Address of " << *(ptr + i) << " is "
                << ptr + i << endl
                << endl;
    }
    retur
    n 0;
}
```

Output:

```
Value of0 arr[0] is 6
Address of 6 is  0x7 c9de51fb0
Value of1 arr[1] is 2
Address of 2 is  0x7 c9de51fb4
Value of2 arr[2] is 5
Address of 5 is  0x7 c9de51fb8
Value of3 arr[3] is 7
Address of 7 is  0x7 c9de51fbc
Value of4 arr[4] is 4
Address of 4 is  0x7 c9de51fc0
```

21. Explain the function overloading with the help of example.

ANS:

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading “Function” names should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.

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

- Parameters should have a different type
- Parameters should have a different number

Syntax:

add(int a, int b) add(double

a, double b) **Example:**

```
#include <iostream> using
namespace std;
void add(int a, int b)
{
cout << "sum = " << (a + b);
}
void add(double a, double b)
{ cout << endl << "sum = " << (a + b);
} int
main()
{ add(10, 2);
      add(5.3,
      6.2);
      return 0;
}
```

Output

sum = 12

sum

=11.5

22. Explain the concept of this pointer.

ANS:

In C++ programming, this is a keyword that refers to the current instance of the class. There can be 3 main uses of this keyword in C++.

- It can be used to pass the current object as a parameter to another method.
- It can be used to refer to the current class instance variable.
- It can be used to declare indexers.

Example:

```
#include<iostream> using

namespace std; class Test

{

privat

e: int

x;

public

:

void setX (int x)
{ this->x = x;

}
void print() { cout << "x = " << x << endl; }
};
int

main() {

Test obj;

int x =

20;

obj.setX(

x);

obj.print
```

```
(); return

0;

}
```

Outpu

t: x =

20

23. Explain new and delete keyword for memory management.

ANS: Dynamic memory allocation in C/C++ refers to performing memory allocation manually by a programmer. Dynamically allocated memory is allocated on Heap, and non-static and local variables get memory allocated on Stack. **new operator:**

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

Syntax to use new operator:

```
pointer-variable = new data-type;
```

delete operator:

Since it is the programmer's responsibility to deallocate dynamically allocated memory, programmers are provided a delete operator in C++ language.

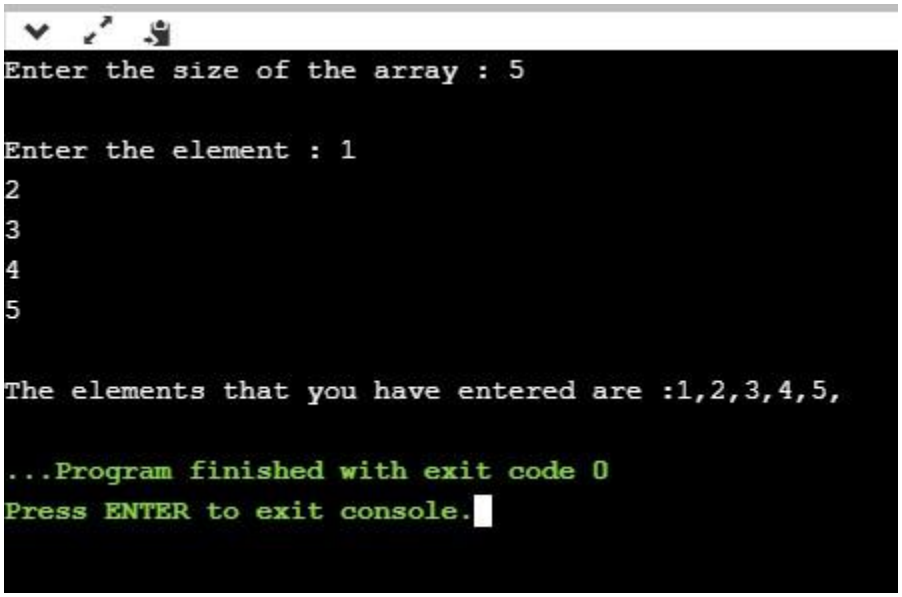
Syntax:

```
// Release memory pointed by pointer-variable delete

pointer-variable;
```

Example:

```
#include <iostream>
using namespace std int
main()
{ int size; int *arr = new int[size];
cout<<"Enter the size of the array : ";
std::cin >> size; cout<<"\nEnter the
element : "; for(int i=0;i<size;i++)
{
cin>>arr[i
];
} cout<<"\nThe elements that you have entered are :";
for(int i=0;i<size;i++)
{
cout<<arr[i]<<","
;
} delete
arr;
return
0;
}
```



24. Write a C++ program demonstrating use of the pure virtual function with the use of base and derived classes.

```
#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; B b;

a = &b; a-

>display();

}
```

25. Explain the Exception Handling with an example.

Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors.

In C++, an exception is an event or object which is thrown at runtime. All exceptions are derived from the std::exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints an exception message and terminates the program.

C++ Exception Handling Keywords

In C++, we use 3 keywords to perform exception handling:

- try
- catch, and
- throw

Example:

```
#include <iostream> using

namespace std; int main()

{ int x = -1; cout << "Before

try \n"; try { cout <<

"Inside try \n"; if (x < 0)

      { throw

         x;

cout << "After throw (Never executed) \n";
```

```
    } } catch (int x ) { cout <<
"Exception Caught \n";
}
cout << "After catch (Will be executed) \n"; return 0;
}
```

Output:

Before try
Inside try
Exception Caught
After catch (Will be executed)

26. Explain the File Handling in OOP.

ANS:

File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).

How to achieve the File Handling

For achieving file handling we need to follow the following steps:-

STEP 1-Naming a file
STEP 2-Opening a file
STEP 3-Writing data into the file STEP 4-
Reading data from the file STEP 5-
Closing a file.

Below are three stream classes of the fstream library for file handling in C++ that are generally used for file handling in C++.

ofstream

The ofstream is derived from the ostream class. It provides the output stream to operate on file. The output stream objects can be used to write the sequences of characters to a file. This class is declared in the fstream header file.

ifstream

The ifstream is derived from the istream class. It provides the input stream to operate on file. We can use that input stream to read from the file. This class is declared in the fstream header file.

fstream

The fstream is derived from the iostream class, and the iostream is further derived from the istream and ostream classes. It provides the input as well as output streams to operate on file.

Example:

```
#include <iostream>

#include <fstream>

using namespace std; int

main () {

  ofstream filestream("testout.txt"); if

  (filestream.is_open())

  { filestream << "Welcome to javaTpoint.\n";

    filestream << "C++ Tutorial.\n";

    filestream.close();

  } else cout <<"File opening is fail.";

  return 0;

}
```

27. Write down the program to demonstrate static keyword in c++.

```
#include <iostream>

class MyClass {
public:
   static int staticVar; // static variable declaration
   int nonStaticVar; // non-static variable declaration
```

```
    MyClass() {
        staticVar++; // increment static variable in constructor
        nonStaticVar++; // increment non-static variable in constructor
    }

    static void printStaticVar() {
        std::cout << "Static variable value: " << staticVar << std::endl;
    }

    void printNonStaticVar() {
        std::cout << "Non-static variable value: " << nonStaticVar << std::endl;
    }
};

int MyClass::staticVar = 0; // static variable definition

int main() {
    MyClass obj1;
    MyClass obj2;
    MyClass::printStaticVar(); // call static function using class name
    obj1.printNonStaticVar(); // call non-static function using object
    obj2.printNonStaticVar(); // call non-static function using object
    return 0;
}
```