Some Questions Related to C++

Introduction to Object Oriented concepts and Design & Principles of object-oriented Programming

1) Define Object-oriented programming and Explain feature of Object oriented Programming. How it is different than procedure oriented programming.

Object Oriented programming :

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc…

Feature of Object oriented Programming :

1. Objects
2. Classes
3. Abstraction
4. Encapsulation
5. Inheritance
6. Overloading
7. Exception Handling

Objects

Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

Class

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers.

Encapsulation

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called the **Base** class & the class which inherits is called the **Derived** class. They are also called parent and child class.

So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

Polymorphism

It is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition. You will learn how to do this in details soon in coming lessons.

Exception Handling

Exception handling is a feature of OOP, to handle unresolved exceptions or errors produced at runtime.

Difference between OOP & POP :

1. POP stands for procedure oriented language and it follows top down approach to solve a problem.

Whereas OOP stands for object oriented language and it follows bottom up approach.

2. In POP, the importance is given to methods and functions to solve an problem. But in OOP, importance is given to Data on which functions are applied.

3. POP follows step by step execution of methods and number of functions . In OOP, number of functions can work at the same time.

4. POP doesn't allow code reusability. Whereas OOP allows it.

5. POP is less secure as data is having least importance whereas OOP gives more importance to data.

6. In POP, its not easy to add new data. But in OOP new objects of data can be made easily.

7. In POP, data and methods are not together whereas in OOP these Two are bound together and its known as data binding.

8. In POP, operator overloading is not allowed and on latter its allowed.

2) Explain all major characteristics of OOP with real world example.

Object

This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as object.

Class

When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Abstraction

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provides different methods to the outside world without giving internal detail about those methods and data.

Encapsulation

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

Inheritance

One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

Some Questions Related to C++

Polymorphism

The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

Overloading

The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

Applications of C / C++ in the Real World :

- Operating Systems
- Development of New Languages
- Computation Platforms
- Embedded Systems
- Graphics and Games
- Games
- Graphic User Interface (GUI) based applications
- Web Browsers
- Advance Computations and Graphics
- Database Software
- Enterprise Software
- Medical and Engineering Applications
- Compilers

Introduction of C++

3) Write down importance of using namespace. How we create and use user defined namespace? Explain nesting user define namespace concepts with small example.

Namespaces become very **important** when it comes to development of a large project. So std is one such **namespace** inside which a lot of symbols are declared. For example: cin, cout, string, vector etc. A small example, which shows how **namespaces** could be helpful in development.

Creation and Use of User Defined Namespace:

Some Questions Related to C++

Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*. This allows organizing the elements of programs into different logical scopes referred to by names.

- Namespace is a feature added in C++ and not present in C.
- A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it.
- Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

```
namespace namespace_name

{

   int x, y; // code declarations where

             // x and y are declared in

             // namespace_name's scope

}
```

- Namespace declarations appear only at global scope.
- Namespace declarations can be nested within another namespace.
- Namespace declarations don’t have access specifiers. (Public or private)
- No need to give semicolon after the closing brace of definition of namespace.
- We can split the definition of namespace over several units.

Example :

```
// Creating namespaces
#include <iostream>
using namespace std;
namespace ns1
{
    int value()    { return 5; }
}
namespace ns2
{
    const double x = 100;
    double value() {  return 2*x; }
}

int main()
{
    // Access value function within ns1
    cout << ns1::value() << '\n';

    // Access value function within ns2
    cout << ns2::value() << '\n';

    // Access variable x directly
    cout << ns2::x << '\n';

    return 0;
}
```

Output:

```
5
200
100
```

Nesting of Namespace :

```
#include <iostream>

int x = 20;
namespace outer
 {
  int x = 10;
  namespace inner
 {
    int z = x; // this x refers to outer::x
 }
 }
int main()
{
  std::cout<<outer::inner::z; //prints 10
  getchar();
  return 0;
}
```

Output : 10

4) What is extractions and insertion operator? write short note on structure of c++ program.

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

Here, it is important to make operator overloading function a friend of the class because it would be called without creating an object.

Following example explains how extraction operator >> and insertion operator <<.

Ex.

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

output: 5****5

Some Questions Related to C++

```
Int main()
{
Int x;
Cin>>x;
Cout<<"****"<<x;}
```

Structure of c++ :

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collecion of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables `name, citNo, salary` to store these information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: `name1, citNo1, salary1, name2, citNo2, salary2`

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

How to declare a structure in C++ programming?

The `struct` keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
char name[50];
int age;
float salary;
};
```

Here a structure `person` is defined which has three members: `name`, `age` and `salary`.

Tokens & Expressions & Control Structure

5) Explain C++ data types and operators in C++ in detail. If we use boolean data type in code then which line must be needed if we want to write "true / flase" instead of "1 / 0"in output screen?

Some Questions Related to C++

C++ Data Types

All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store.
Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires different amount of memory.
Data types in C++ is mainly divided into two types:

1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
   - Integer
   - Character
   - Boolean
   - Floating Point
   - Double Floating Point
   - Valueless or Void
   - Wide Character

2. **Abstract or user defined data type**: These data types are defined by user itself. Like, defining a class in C++ or a structure.

This article discusses **primitive data types** available in C++.

- **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true* or *false*. Keyword used for boolean data type is **bool**.
- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- **Wide Character**: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by **wchar_t**. It is generally 2 or 4 bytes long.

Datatype Modifiers: As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold. Data type modifiers available in C++ are:

- **Signed**
- **Unsigned**
- **Short**
- **Long**

Some Questions Related to C++

Below table summarizes the modified size and range of built-in datatypes when combined with the type modifiers:

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

If we use boolean data type in code then which line must be needed if we want to write "true / flase" instead of "1 / 0"in output screen?

we have to write **cout<<std::boolalpha;**

6) What is scope resolution operator? what are the applications for it?

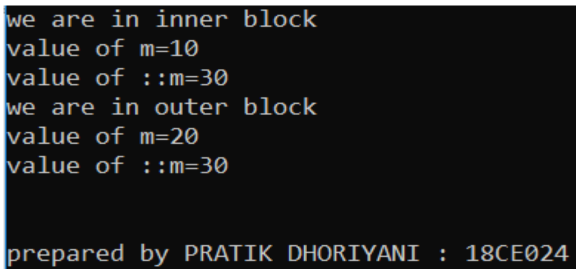
The :: (scope resolution) operator is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. You can use the unary scope operator if a namespace scope or global scope name is hidden by a particular-declaration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

For example,

```
#include<iostream>
using namespace std;
int m=30;
int main()
{
int m=20;
{
int m=10;
cout<<"we are in inner block"<<endl;
cout<<"value of m="<<m<<"\n";
cout<<"value of ::m="<<::m<<"\n";
}
cout<<"we are in outer block"<<endl;
cout<<"value of m="<<m<<"\n";
cout<<"value of ::m="<<::m<<"\n";
return 0;

}
```

Output:



```
we are in inner block
value of m=10
value of ::m=30
we are in outer block
value of m=20
value of ::m=30


prepared by PRATIK DHORIYANI : 18CE024
```

Application:

- To access global variable
- To define function outside class
- To access class's static variable

- In case of multiple inheritance

7) Explain how to allocate and de-allocate memory dynamically in c++.

New operator :

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

- **Syntax :**
  pointer-variable = **new** data-type;

  Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class. Example:

  // Pointer initialized with NULL

  // Then request memory for the variable

  int *p = NULL;

  p = new int;

Delete operator :

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

Syntax:

delete pointer-variable;

Here, pointer-variable is the pointer that points to the data object created by *new*. Examples:
delete p;

delete q;

Functions in c++

8) When do we need to use default arguments in a function? Discuss with small program.

- A default argument is a value provided in function declaration that is automatically assigned by the compiler if caller of the function doesn't provide a value for the argument with default value.
- Example:

```
#include<iostream>
using namespace std;
  int mul(int x, int y, int z=5)
     {
        return (x*y*z);
      }
int main()
{
cout << sum(5,6) << endl;
cout << sum(5,6,10) << endl;
cout << sum(5,5,5) << endl;
return 0;
}
```

Output:

```
150
300
125
```

9) Explain different use of const keyword in C++.

- To give constant value to variable
- To make fuction constant

➢ Declaring a member function with the **const** keyword specifies that the function is a "read-only" function that does not modify the object for which it is called. A constant member function cannot modify any non-static data members or call any member functions that aren't constant.To declare a constant member function, place the **const** keyword after the closing parenthesis of the argument list. The **const** keyword is required in both the declaration and the definition.

Const Pointer:

- If we make a pointer **const**, we cannot change the pointer. This means that the pointer will always point to the same address but we can change the value of that address.
- We declare a const pointer as follows.
- int a = 4;
- int* const p = &a;   // const pointer p pointing to the variable a

Pointer to Const Variable:

- On the contrary to the above case, we can make the variable to which a pointer is pointing as **const**. Let's see how to declare a pointer to const variable.
- const int* p;   // const pointer p pointing to a const int variable
- Here, **p** is a pointer which is pointing to a const int variable. This means that we cannot change the value of that int variable.

Constant Data Members of Class:

- Data members are just variables declared inside a class. **const** data members are not assigned values during its declaration. Const data members are assigned values in the constructor.

-
```
#include <iostream>
using namespace std;
class A
{
const int x;
public:
A(int y)
  {
    x = y;
  }

};
int main()
{
        A a(5);
        return 0;
}
```

Constant Objects of Classes:

- We can also make an object of class const. We make an object const by writing the **const** keyword at the beginning of the object declaration as shown below.
- const A a(4);
- We made the object **a** of class A const by writing the const keyword before the class name at the time of its declaration.

A const class object is initialized through the constructor.

Classes And Object

10) What is friend functions? what is the difference between friend function of a class and a member function of a class?

- **Friend Function** Like friend , a friend function can be given special grant to access private and protected members. A friend function can be:
  a) A function of another class
  b) A global function
- **Difference:**

A **member function** is called using the object of the class. **Member functions** will have the access on its own private variables and the the some public variables of other classes but a **friend function** will have the accessibility on private variable of other classes too.

11) When do we declare a member of a class static? Why? Discuss static member function with example.

As the variables declared as static are initialized only once as they are allocated space in separate static storage so, the static variables **in a class are shared by the objects.** There can not be multiple copies of same static variables for different objects. Also because of this reason static variables can not be initialized inside the class.
we can not declare a static member inside the class. Static member has initial value 0, when it declare in the class.
But we can declare the static data member outside the class by using score resolution With class name.
e.g.

```
#include<iostream>
Using namespace std;
Class a
{
 Static int r;
} A ;
Int a::r=80;
```

Static Member Function:

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the **scope resolution operator ::**

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the **this** pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Example:

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

class a
{
   public:

    static void printMsg()
    {
        cout<<”hi”;
    }
};

int main()
{
    a::printMsg();
}
```

12) What is the difference between C structure, C++ structure and class. How class can be specify?

Keyword for declaration :
For declare class there is the class keyword.
For declare structure of c and C++ keyword is Struct.

Default Access specifier:
Default Access specifier for the class is private, for the structure of C++ is public,
And C structure has not default access specifier.

Declaration of function:
We can declare the function inside the class and Structure but we can not declare function in c structure.

initialization of member variable:
we can initialize variable inside the class but in structure of c and C++ there are initialization inside the structure is not possible.

Difference between C++ CLASS & STRUCTURE :

A) Class
1. A class in C++ can be defined as a collection of related variables and functions encapsulated in a single structure.
2. A class in C++ is just an extension of a structure used in the C language. It is a user defined data type. It actually binds the data and its related functions in one unit.
3. Keyword for the declaration: Class
4. Default access specifier: Private
5. Purpose: Data abstraction and further inheritance
6. Type: Reference
7. Usage: Generally used for large amounts of data.
8. Its object is created on the heap memory.
9. The member variable of class can be initialized directly.
10. It can have all the types of constructor and destructor.

B) Structure
1. A structure in C++ can be referred to as an user defined data type possessing its own operations.
2. A structure and a class in C language differs a lot as a structure has limited

functionality and features as compared to a class.
3. Keyword for the declaration: Struct
4. Default access specifier: Public
5. Purpose: Generally, grouping of data
6. Type: Value
7. Usage: Generally used for smaller amounts of data.
8. Its object is created on the stack memory.
9. The member variable of structure cannot be initialized directly.
10. It can have only parameterized constructor.

13) Describe the concept of private member function of a class.

The class members declared as private is access only the member which is declared in the private section of the class.
Private member aren't allow to be accessed directly by any object or function outside the class.
Only the member function or friend function are allowed to access the private data member of a class.

Constructor and Destructors

14) what is the purpose of copy constructor? Name two situations in which a copy constructors executes.

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to −

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

If a copy constructor is not defined in a class, the compiler itself defines one. If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor. The most common form of copy constructor is shown here −

```
classname (const classname &obj)
{
    // body of constructor
}
```

Situation when copy constructor is called:

1. when an existing object is assigned an object of it own class
```
MyClass A,B;
A = new MyClass();
B=A;
```

2. if a functions receives as argument, passed by value,of an object of a class

```
void food(MyClass a);
food(a);
```

15) What is virtual destructor? Discuss with small example. Can I overload any Destructor?

Deleting a derived class object using a pointer to a base class that has a non-virtual destructor results in undefined behavior. To correct this situation, the base class should be defined with a virtual destructor. For example, following program results in undefined behavior.

```
// CPP program without virtual destructor
// causing undefined behavior
#include<iostream>

using namespace std;

class base {
  public:
    base()
    { cout<<"Constructing base \n"; }
    ~base()
    { cout<<"Destructing base \n"; }
};

class derived: public base {
  public:
    derived()
    { cout<<"Constructing derived \n"; }
    ~derived()
    { cout<<"Destructing derived \n"; }
};

int main(void)
{
  derived *d = new derived();
  base *b = d;
  delete b;
  getchar();
  return 0;
}
```

Although the output of following program may be different on different compilers, when compiled using Dev-CPP, it prints following:

```
Constructing base

Constructing derived

Destructing base
```

Making base class destructor virtual guarantees that the object of derived class is destructed properly, i.e., both base class and derived class destructors are called. For example,

```
// A program with virtual destructor
#include<iostream>

using namespace std;

class base {
  public:
    base()
    { cout<<"Constructing base \n"; }
    virtual ~base()
    { cout<<"Destructing base \n"; }
};

class derived: public base {
  public:
    derived()
    { cout<<"Constructing derived \n"; }
    ~derived()
    { cout<<"Destructing derived \n"; }
};

int main(void)
{
  derived *d = new derived();
  base *b = d;
  delete b;
  getchar();
  return 0;
}
```

Output:

```
Constructing base

Constructing derived

Destructing derived

Destructing base
```

As a guideline, any time you have a virtual function in a class, you should immediately add a virtual destructor (even if it does nothing). This way, you ensure against any surprises later.

16) What is dynamic initialization of objects? Why is it required? Illustrate with an example in C++.

Dynamic initialization of object refers to initializing the objects at run time i.e. the initial value of an object is to be provided during run time. Dynamic initialization can be achieved using constructors and passing parameters values to the constructors. This type of initialization is required to initialize the class variables during run time.

Need of dynamic initialization :

Dynamic initialization of objects is needed as

- It utilizes memory efficiently.
- Various initialization formats can be provided using overloaded constructors.
- It has the flexibility of using different formats of data at run time considering the situation.

Example:

```
#include <iostream>
using namespace std;
class simple_interest
{
   float principle , time, rate ,interest;

   public:
      simple_interest (float a, float b, float c)
 {
         principle = a;
         time =b;
         rate = c;
      }
      void display ( ) {
         interest =(principle* rate* time)/100;
         cout<<"interest ="<<interest ;
      }
};
int main()
{
float p,r,t;
cout<<"principle amount, time and rate"<<endl;
cout<<"2000          7.5       2"<<endl;
simple_interest s1(2000,7.5,2);//dynamic initialization
s1.display();
return 0;
}
```