What is C++?

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C language. Originally named C with Classes, the language was renamed C++ in 1983.

C++ is one of the most popular programming languages with application domains including systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.

The language began as enhancements to C, first adding classes, then virtual functions, operator overloading, multiple inheritance, templates, and exception handling among other features. After years of development, the C++ programming language standard was ratified in 1998 as ISO/IEC 14882:1998. The standard was amended by the 2003 technical corrigendum, ISO/IEC 14882:2003. The current standard extending C++ with new features was ratified and published by ISO in September 2011 as ISO/IEC 14882:2011 (informally known as C++11)

What are C++ storage classes?

auto
register
static
extern

auto: the default. Variables are automatically created and initialized when they are defined and are destroyed at the end of the block containing their definition. They are not visible outside that block

register: a type of auto variable. a suggestion to the compiler to use a CPU register for performance

static: a variable that is known only in the function that contains its definition but is never destroyed and retains its value between calls to that function. It exists from the time the program begins execution

extern: a static variable whose definition and placement is determined when all object and library modules are combined (linked) to form the executable code file. It can be visible outside the file where it is defined.

What are storage qualifiers in C++ ?

They are..

const
volatile
mutable

Const keyword indicates that memory once initialized, should not be altered by a program.

volatile keyword indicates that the value in the memory location can be altered even though nothing in the program
code modifies the contents. for example if you have a pointer to hardware location that contains the time, where hardware changes the value of this pointer variable and not the program. The intent of this keyword to improve the optimization ability of the compiler.

mutable keyword indicates that particular member of a structure or class can be altered even if a particular structure variable, class, or class member function is constant.

```
struct data

{

char name[80];

mutable double salary;
}

const data MyStruct = { "Satish Shetty", 1000 }; //initlized by complier

strcpy ( MyStruct.name, "Shilpa Shetty"); // compiler error
MyStruct.salaray = 2000 ; // complier is happy allowed
```

What is reference ?

reference is a name that acts as an alias, or alternative name, for a previously defined variable or an object.

prepending variable with "&" symbol makes it as reference.

for example:

```
int a;
int &b = a;
```

What is passing by reference?

Method of passing arguments to a function which takes parameter of type reference.

for example:

```
void swap( int & x, int & y )
{
 int temp = x;
 x = y;
 y = temp;
}

int a=2, b=3;

swap( a, b );
```

Basically, inside the function there won't be any copy of the arguments "x" and "y" instead they refer to original variables a and b. so no extra memory needed to pass arguments and it is more efficient.

When do use "const" reference arguments in function?

a) Using const protects you against programming errors that inadvertently alter data.
b) Using const allows function to process both const and non-const actual arguments, while a function without const in the prototype can only accept non constant arguments.
c) Using a const reference allows the function to generate and use a temporary variable appropriately.

When are temporary variables created by C++ compiler?

Provided that function parameter is a "const reference", compiler generates temporary variable in following 2 ways.

a) The actual argument is the correct type, but it isn't Lvalue

```
double Cube(const double & num)

{

  num = num * num * num;

  return num;


}


double temp = 2.0;

double value = cube(3.0 + temp); // argument is a expression and not a Lvalue;
```

b) The actual argument is of the wrong type, but of a type that can be converted to the correct type

```
long temp = 3L;

double value = cuberoot ( temp); // long to double conversion
```

What is virtual function?

When derived class overrides the base class method by redefining the same function, then if client wants to access redefined the method from derived class through a pointer from base class object, then you must define this function in base class as virtual function.

```
class parent

{

   void Show()

{

cout << "i'm parent" << endl;

}

};


class child: public parent

{

   void Show()

{

cout << "i'm child" << endl;

}


};


parent * parent_object_ptr = new child;


parent_object_ptr->show() // calls parent->show() i
```

now we goto virtual world...

```
class parent
```

```
{
   virtual void Show()
{
cout << "i'm parent" << endl;
}
};

class child: public parent
{
   void Show()
{
cout << "i'm child" << endl;
}


};

parent * parent_object_ptr = new child;

parent_object_ptr->show() // calls child->show()
```

What is pure virtual function? or what is abstract class?

When you define only function prototype in a base class without implementation and do the complete implementation in derived class. This base class is called abstract class and client won't able to instantiate an object using this base class.

You can make a pure virtual function or abstract class this way..

```
class Boo
{
void foo() = 0;
}

Boo MyBoo; // compilation error
```

What is Memory alignment?

The term alignment primarily means the tendency of an address pointer value to be a multiple of some power of two. So a pointer with two byte alignment has a zero in the least significant bit. And a pointer with four byte alignment has a zero in both the two least significant bits. And so on. More alignment means a longer sequence of zero bits in the lowest bits of a pointer.

What problem does the namespace feature solve?

Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The namespace feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions.

namespace [identifier] { namespace-body }

A namespace declaration identifies and assigns a name to a declarative region.
The identifier in a namespace declaration must be unique in the declarative region in which it is used. The identifier is the name of the namespace and is used to reference its members.

What is the use of 'using' declaration?

A using declaration makes it possible to use a name from a namespace without the scope operator.

What is an Iterator class?

A class that is used to traverse through the objects maintained by a container class. There are five categories of iterators: input iterators, output iterators, forward iterators, bidirectional iterators, random access. An iterator is an entity that gives access to the contents of a container object without violating encapsulation constraints. Access to the contents is granted on a one-at-a-time basis in order. The order can be storage order (as in lists and queues) or some arbitrary order (as in array indices) or according to some ordering relation (as in an ordered binary tree). The iterator is a construct, which provides an interface that, when called, yields either the next element in the container, or some value denoting the fact that there are no more elements to examine. Iterators hide the details of access to and update of the elements of a container class. Something like a pointer.

What is a dangling pointer?

A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed.

What do you mean by Stack unwinding?

It is a process during exception handling when the destructor is called for all local objects in the stack between the place where the exception was thrown and where it is caught.

Name the operators that cannot be overloaded?

sizeof, ., .*, .->, ::, ?:

What is a container class? What are the types of container classes?

A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behavior and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

What is inline function?

The __inline keyword tells the compiler to substitute the code within the function definition for every instance of a function call. However, substitution occurs only at the compiler's discretion. For example, the compiler does not inline a function if its address is taken or if it is too large to inline.

What is overloading?

With the C++ language, you can overload functions and operators. Overloading is the practice of supplying more than one definition for a given function name in the same scope.

- Any two functions in a set of overloaded functions must have different argument lists.
- Overloading functions with argument lists of the same types, based on return type alone, is an error.

What is Overriding?

To override a method, a subclass of the class that originally declared the method must declare a method with the same name, return type (or a subclass of that return type), and same parameter list.
The definition of the method overriding is:

- Must have same method name.
- Must have same data type.
- Must have same argument list.

Overriding a method means that replacing a method functionality in child class. To imply overriding functionality we need parent and child classes. In the child class you define the same method signature as one defined in the parent class.

What is "this" pointer?

The this pointer is a pointer accessible only within the member functions of a class, struct, or union type. It points to the object for which the member function is called. Static member functions do not have a this pointer.

When a nonstatic member function is called for an object, the address of the object is passed as a hidden argument to the function. For example, the following function call

```
myDate.setMonth( 3 );
```

can be interpreted this way:

```
setMonth( &myDate, 3 );
```

The object's address is available from within the member function as the this pointer. It is legal, though unnecessary, to use the this pointer when referring to members of the class.

What happens when you make call "delete this;" ?

The code has two built-in pitfalls. First, if it executes in a member function for an extern, static, or automatic object, the program will probably crash as soon as the delete statement executes. There is no portable way for an object to tell that it was instantiated on the heap, so the class cannot assert that its object is properly instantiated. Second, when an object commits suicide this way, the using program might not know about its demise. As far as the instantiating program is concerned, the object remains in scope and continues to exist even though the object did itself in. Subsequent dereferencing of the pointer can and usually does lead to disaster.

You should never do this. Since compiler does not know whether the object was allocated on the stack or on the heap, "delete this" could cause a disaster.

How virtual functions are implemented C++?

Virtual functions are implemented using a table of function pointers, called the vtable. There is one entry in the table per virtual function in the class. This table is created by the constructor of the class. When a derived class is constructed, its base class is constructed first which creates the vtable. If the derived class overrides any of the base classes virtual functions, those entries in the vtable are overwritten by the derived class constructor. This is why you should never call virtual functions from a constructor: because the vtable entries for the object may not have been set up by the derived class constructor yet, so you might end up calling base class implementations of those virtual functions

---

What is name mangling in C++?

The process of encoding the parameter types with the function/method name into a unique name is called name mangling. The inverse process is called demangling.

For example Foo::bar(int, long) const is mangled as `bar__C3Fooil'.
For a constructor, the method name is left out. That is Foo::Foo(int, long) const is mangled as `__C3Fooil'.

What is the difference between a pointer and a reference?

A reference must always refer to some object and, therefore, must always be initialized; pointers do not have such restrictions. A pointer can be reassigned to point to different objects while a reference always refers to an object with which it was initialized.

How are prefix and postfix versions of operator++() differentiated?

The postfix version of operator++() has a dummy parameter of type int. The prefix version does not have dummy parameter.

What is the difference between const char *myPointer and char *const myPointer?

Const char *myPointer is a non constant pointer to constant data; while char *const myPointer is a constant pointer to non constant data.

How can I handle a constructor that fails?

throw an exception. Constructors don't have a return type, so it's not possible to use return codes. The best way to signal constructor failure is therefore to throw an exception.

How can I handle a destructor that fails?

Write a message to a log-file. But do not throw an exception.
The C++ rule is that you must never throw an exception from a destructor that is being called during the "stack unwinding" process of another exception. For example, if someone says throw Foo(), the stack will be unwound so all the stack frames between the throw Foo() and the } catch (Foo e) { will get popped. This is called stack unwinding.
During stack unwinding, all the local objects in all those stack frames are destructed. If one of those destructors throws an exception (say it throws a Bar object), the C++ runtime system is in a no-win situation: should it ignore the Bar and end up in the } catch (Foo e) { where it was originally headed? Should it ignore the Foo and look for a } catch (Bar e) { handler? There is no good answer -- either choice loses information.
So the C++ language guarantees that it will call terminate() at this point, and terminate() kills the process. Bang you're dead.

What is Virtual Destructor?

Using virtual destructors, you can destroy objects without knowing their type - the correct destructor for the object is invoked using the virtual function mechanism. Note that destructors can also be declared as pure virtual functions for abstract classes.

if someone will derive from your class, and if someone will say "new Derived", where "Derived" is derived from your class, and if someone will say delete p, where the actual object's type is "Derived" but the pointer p's type is your class.

Can you think of a situation where your program would crash without reaching the breakpoint which you set at the beginning of main()?

C++ allows for dynamic initialization of global variables before main() is invoked. It is possible that initialization of global will invoke some function. If this function crashes the crash will occur before main() is entered.

Name two cases where you MUST use initialization list as opposed to assignment in constructors.

Both non-static const data members and reference data members cannot be assigned values; instead, you should use initialization list to initialize them.

Can you overload a function based only on whether a parameter is a value or a reference?

No. Passing by value and by reference looks identical to the caller.

What are the differences between a C++ struct and C++ class?

The default member and base class access specifiers are different.

The C++ struct has all the features of the class. The only differences are that a struct defaults to public member access and public base class inheritance, and a class defaults to the private access specifier and private base class inheritance.

What does extern "C" int func(int *, Foo) accomplish?

It will turn off "name mangling" for func so that one can link to code compiled by a C compiler.

How do you access the static member of a class?

```
<ClassName>::<StaticMemberName>
```

What is multiple inheritance(virtual inheritance)? What are its advantages and disadvantages?

Multiple Inheritance is the process whereby a child can be derived from more than one parent class. The advantage of multiple inheritance is that it allows a class to inherit the functionality of more than one base class thus allowing for modeling of complex relationships. The disadvantage of multiple inheritance is that it can lead to a lot of confusion(ambiguity) when two base classes implement a method with the same name.

What are the access privileges in C++? What is the default access level?

The access privileges in C++ are private, public and protected. The default access level assigned to members of a class is private. Private members of a class are accessible only within the class and by friends of the class. Protected members are accessible by the class itself and it's sub-classes. Public members of a class can be accessed by anyone.

What is a nested class? Why can it be useful?

A nested class is a class enclosed within the scope of another class. For example:

Example 1: Nested class

```
//
  class OuterClass
  {
    class NestedClass
    {
      // ...
    };
    // ...
  };
```

Nested classes are useful for organizing code and controlling access and dependencies. Nested classes obey access rules just like other parts of a class do; so, in Example 1, if NestedClass is public then any code can name it as OuterClass::NestedClass. Often nested classes contain private implementation details, and are therefore made private; in Example 1, if NestedClass is private, then only OuterClass's members and friends can use NestedClass.

When you instantiate as outer class, it won't instantiate inside class.

What is a local class? Why can it be useful?

local class is a class defined within the scope of a function -- any function, whether a member function or a free function. For example:

// Example 2: Local class

```
//
  int f()
  {
    class LocalClass
    {
      // ...
    };
    // ...
  };
```

Like nested classes, local classes can be a useful tool for managing code dependencies.

Can a copy constructor accept an object of the same class as parameter, instead of reference of the object?

No. It is specified in the definition of the copy constructor itself. It should generate an error if a programmer specifies a copy constructor with a first argument that is an object and not a reference.

Explain how we implement exception handling in C++.

Exception handling in C++ is implemented by using the try{} and catch(){} statements.

When a try block throws an exception, the program leaves the try block and enters the catch statement of the catch block.

If they type of the object thrown matches the arg type in the catch block, catch block is executed for handling the code.

If they are not caught, abort() function is executed by default.

When no exception is deteted or thrown then the control goes to the statement below the catch block.

Explain the concepts of throwing and catching exceptions

C++ provides a mechanism to handle exceptions which occurs at runtime. C++ uses the keywords – throw, catch and try to handle exception mechanism. An entity called an exception class need to be created.
The application should have a separate section called catch block. The code in which an exception is predicted is authored in the try block.
The following code illustrates to handle the exception.

```
#include <iostream>
class Excep {
public:
      const char* error;
      Excep(const char* arg) : error(arg) { }
};


class Test {
   public:
        int i;
    // A function try block with a member initializer
      Test() try : i(0) {
             throw Excep("Exception thrown in A()");
       }
       catch (Excep& e) {
             cout << e.error << endl;
       }
};


// A function try block
void f() try {
      throw E("Exception thrown in f()");
}
catch (Excep& e) {
     cout << e.error << endl;
}
void g() {
     throw Excep("Exception thrown in g()");
}


int main() {
   f();
    // A try block
    try {
       g();
    }
     catch (Excep& e) {
          cout << e.error << endl;
     }
      try {
           Test x;
      }
      catch(...) { }
}
```

The following is the output of the above example:

```
Exception thrown in f()
Exception thrown in g()
Exception thrown in A().
```

Explain terminate() and unexpected() function

terminate() is a library function which by default aborts the program

It is called whenever the exception handling mechanism cannot find a handler for a thrown exception.

unexpected() is called when a function with an exception specification throws an exception of a type that is not listed in the exception specification for the function
A function declaration without a specification like throw(char*) may throw any type of exception, and one with throw() is not allowed to throw exceptions at all.

By default unexpected() calls terminate().

Describe exceptions in C++

Exceptions: Exceptions are certain disastrous error conditions that occur during the execution of a program. They could be errors that cause the programs to fail or certain conditions that lead to errors. If these run time errors are not handled by the program, OS handles them and program terminates abruptly, which is not good. So C++ provides Exception Handling mechanism.

Exceptions are of two types:

Synchronous Exceptions: Errors that occur due to incorrect input data or incorrect handling of array indices ("out of range index"), memory overflow are known as Synchronous Exceptions

Asynchronous Exceptions: The errors that are caused by events that are beyond control of the program are Asynchronous Exceptions. E.g. Keyboard interrupts

C++ exception handling mechanism takes care of only Synchronous Exceptions.

The benefits of Exception Handling are:

1. Program is not terminated abruptly
2. User will understand what errors are occurring in the program.

The three keywords for Exception Handling are:
Try, Catch and Throw.

The program tries to do something. If it encounters some problem, it throws an exception to another program block which catches the exception.

Consider following program code:

```
void main()

{

     int no1, no2;

     try

     {

         cout << "Enter two nos:";

         cin >> no1 >> no2;

         if (no2 == 0)

             throw "Divide by zero";

         else

             throw no1/no2;

     }

     catch (char *s)

     {

         cout << s;

     }
     catch (int ans)
     {
         cout << ans;
     }
}
```

We know that divide by zero is an exception. If user enters second no as zero, the program throws an exception, which is caught and an error message is printed.

Please note that catch is not a function; it is a program block.

Illustrate Rethrowing exceptions with an example.

Rethrowing an expression from within an exception handler can be done by calling throw, by itself, with no exception. This causes current exception to be passed on to an outer try/catch sequence. An exception can only be rethrown from within a catch block. When an exception is rethrown, it is propagated outward to the next catch block.

Consider following code:

```
#include <iostream>
using namespace std;
void MyHandler()
{
        try
        {
                 throw “hello”;
        }
        catch (const char*)
        {
               cout <<”Caught exception inside MyHandler\n”;
               throw; //rethrow char* out of function
        }
}
int main()
{
            cout<< “Main start”;
            try
            {
                   MyHandler();
            }
            catch(const char*)
            {
                  cout <<”Caught exception inside Main\n”;
            }
                  cout << “Main end”;
            return 0;
}
```

O/p:
```
Main start
Caught exception inside MyHandler
Caught exception inside Main
Main end
```
Thus, exception rethrown by the catch block inside MyHandler() is caught inside main();

What are the characteristics of friend functions?

Friend functions are not a part of the class and are external. This function is a “Friend” of a class. This is to say, it has special privileges to access to the class's private and protected members.

What is a friend function?

To allow a non-member function the access to private members of a class, it needs to be friend of that class. Friend functions can access private and protected data of the class. To make a non-member function friend of a class, its declaration needs to be made inside the class and it has to be preceded by the keyword friend. We can also have a member function of a class to be friend of certain other class. Even if a function is not a member of any class, it can still be friend of multiple classes.

If we write equivalent friend function for a member function, then friend function has one extra parameter because being a non-member of the class, it does not have the caller object. Therefore, it does not have this pointer.

Friend functions are very useful for overloading certain types of operators. They also make creation of some type of I/O functions easier.

Consider following example of class 3D having data members x, y and z. Overloaded binary operator * for scalar multiplication. It should work in both cases:

```
ob1 = ob2 * 3;
ob1 = 3 * ob2;
```

Note that first can be achieved through member or friend function. But for the second case we need to write friend function since there is no caller object.

```
class 3D
{
int x, y, z;
public:
3D (int a=0, int b=0, int c=0)
{
x = a;
y = b;
z = c;
}
3D show()
{
cout<<”The elements are:\n”
cout<<”x:”<<this->x<<”, y:<<this->y <<”, z:”<<this->z;
}
friend 3D operator * (3D, int);
friend 3D operator * (int, 3D);
};

3D operator * (3D ob, int i) //friend function’s definition is written outside class
{
3D tob;
tob.x = ob.x * i;
tob.y = ob.y * i;
tob.z = ob.z * i;
return tob;
}

3D operator * (int i, 3D ob) //friend function’s definition is written outside class
{
3D tob;
tob.x = ob.x * i;
tob.y = ob.y * i;
tob.z = ob.z * i;
return tob;
}

int main()
{
3D pt1(2,-4,5), pt2, pt3;
pt 2 = pt1 * 3;
pt3 = -2 * pt1
cout << “\n Point one’s dimensions are: \n“<<pt1.show();
cout << “\n Point two’s dimensions are: \n“<<pt2.show();
cout << “\n Point three’s dimensions are: \n“<<pt3.show();
return 0;
}
```

The o/p would be:

```
Point one’s dimensions are:
x:2, y:-4, z:5
Point two’s dimensions are:
x:6, y:-12, z:15
Point three’s dimensions are:
x:-4, y:8, z:-10
```

Thus, friend functions are useful when we have to overload operators which have no caller object.

Explain the advantages of using friend classes.

There are situations when private data members need to be accessed and used by 2 classes simultaneously. In these kind of situations we can introduce friend functions which have an access to the private data members of both the classes. Friend functions need to be declared as ‘friend’ in both the classes. They need not be members of either of these classes.

What are friend classes? Explain its uses with an example.

A friend class and all its member functions have access to all the private members defined within other class.

```
#include <iostream>
using namespace std;
class Numbers
{
    int a;
    int b;
    public:
           Numbers(int i, int j)
           {
                     a = i;
                     b = j;
           }
           friend class Average;
};

class Average
{
       public:
             int average(Numbers x);
};

int Average:: average(Numbers x)
{
           return ((x.a + x.b)/2);
}

int main()
{
         Numbers ob(23, 67);
         Average avg;
         cout<<”The average of the numbers is: ”<<avg.average(ob);
         return 0;
}
```

Note the friend class member function average is passed the object of Numbers class as parameter.

Guideline that should be followed while using friend function.

When the application is needed to access a private member of another class, the only way is to utilize the friend functions. The following are the guidelines to handle friend functions.

- Declare the function with the keyword ‘friend’ that is followed by return type and followed by the function name.
- Specify the class whose private members are to be accessed in the friend function within parenthesis of the friend function.
- Write the code of the friend function in the class.

The following code snippet illustrates the use of friend function:

```
friend void display(car); //Friend of the class 'car'


void display(car myCar)

{

      cout<<"\nThe color of my car is : "<<myCar.color;

      cout<<"\nThe speed of my car is : "<<myCar.speed;

}
```

What is a base class?

Inheritance is one of the important features of OOP which allows us to make hierarchical classifications of classes. In this, we can create a general class which defines the most common features. Other more specific classes can inherit this class to define those features that are unique to them. In this case, the class from which other classes are inherited is referred as base class.

For example, a general class vehicle can be inherited by more specific classes car and bike. The class vehicle is base class in this case.

```
class Base

{

     int a;

     public:

         Base()

         {

                   a = 1;

                   cout <<”inside Base class”;

         }

};


class Derived:: public Base //class Derived is inheriting class Base publically

{

int b;

public:

Derived()

{

b = 1;

cout <<”inside Derived class”;

}

};
```

What is private inheritance?

When a class is being derived from another class, we can make use of access specifiers. This is essentially useful to control the access the derived class members have to the base class. When inheritance is private:

1. Private members of base class are not accessible to derived class.
2. Protected members of base class become private members of derived class.
3. Public members of base class become private members of derived class.

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

class base
{
      int i, j;
      public:
       void setij(int a, int b)
      {
           i = a;
           j = b;
      }

       void showij()
{
cout <<"\nI:"<<i<<"\n J:"<<j;
}
};

class derived : private base
{
int k;
public:
void setk()
{
//setij();
k = i + j;
}
void showall()
{
cout <<"\nK:"<<k<<show();
}
};
```

```
int main()

{

derived ob;

//ob.setij(); // not allowed. Setij() is private member of derived

ob.setk(); //ok setk() is public member of derived

//ob.showij(); // not allowed. Showij() is private member of derived

ob.showall(); // ok showall() is public member of derived

return 0;

}
```

What is protected inheritance?

When a class is being derived from another class, we can make use of access specifiers. This is essentially useful to control the access the derived class members have to the base class. When inheritance is protected:

- Private members of base class are not accessible to derived class.
- Protected members of base class remain protected in derived class.
- Public members of base class become protected in derived class.

```
#include <iostream>

using namespace std;


class base

{

      protected:

      int i, j;

      public:

      void setij(int a, int b)

      {

             i = a;

             j = b;

      }

      void showij()

{

cout <<"\nI:"<<i<<"\n J:<<j;

}

};


class derived : protected base

{

int k;

public:

void setk()

{

setij();

k = i + j;
}
void showall()
{
cout <<"\nK:"<<k<<show();
}
};

int main()
{
derived ob;
//ob.setij(); // not allowed. Setij() is protected member of derived
ob.setk(); //ok setk() is public member of derived
//ob.showij(); // not allowed. Showij() is protected member of derived
ob.showall(); // ok showall() is public member of derived
return 0;
}
```

What are ways to avoid memory leaks?

A memory leak is the effect of running out of memory.

A memory leak is what happens when you forget to free a block of memory allocated with the new operator or when you make it impossible to so.

The measures you can take are :
Delete before reallocating a memory

Be sure you have a pointer to each dynamic variable so that you do not lose a location that you need to free.

Avoid these combinations : malloc() - delete and new - free()

Avoid these combinations : new - delete [] and new [] - delete.

When are memory leaks important? What are the easiest ways to avoid memory leak?

Memory Leaks:

Memory leaks take up more memory space and causes slow calculation.
Ensure that the loops or conditional statements are executed completely to prevent memory leak occurrence.

The best and easiest way to avoid memory leaks is to make changes at the time of writing programs. The cause for this is, a properly planned program can avoid memory leaks. In addition to it, make a code snippet can be made simple, small which executes fast.

How do you link a C++ program to C functions?

By using the extern "C" linkage specification around the C function declarations.

Explain the scope resolution operator.

It permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.

What are the differences between a C++ struct and C++ class?

The default member and base-class access specifiers are different.

How many ways are there to initialize an int with a constant?

Two.

There are two formats for initializers in C++ as shown in the example that follows. The first format uses the traditional C notation. The second format uses constructor notation.

```
int foo = 123;

int bar (123);
```

How does throwing and catching exceptions differ from using setjmp and longjmp?

The throw operation calls the destructors for automatic objects instantiated since entry to the try block.

What is your reaction to this line of code?

delete this;

It's not a good practice.

What is a default constructor?

A constructor that has no arguments.

What is a conversion constructor?

A constructor that accepts one argument of a different type.

What is the difference between a copy constructor and an overloaded assignment operator?

A copy constructor constructs a new object by using the content of the argument object. An overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

When should you use multiple inheritance?

There are three acceptable answers: "Never," "Rarely," and "When the problem domain cannot be accurately modeled any other way."

What is a virtual destructor?

The simple answer is that a virtual destructor is one that is declared with the virtual attribute.

Explain the ISA and HASA class relationships. How would you implement each in a class design?

A specialized class "is" a specialization of another class and, therefore, has the ISA relationship with the other class. An Employee ISA Person. This relationship is best implemented with inheritance. Employee is derived from Person. A class may have an instance of another class. For example, an employee "has" a salary, therefore the Employee class has the HASA relationship with the Salary class. This relationship is best implemented by embedding an object of the Salary class in the Employee class.

When is a template a better solution than a base class?

When you are designing a generic class to contain or otherwise manage objects of other types, when the format and behavior of those other types are unimportant to their containment or management, and particularly when those other types are unknown (thus, the genericity) to the designer of the container or manager class.

What is a mutable member?

One that can be modified by the class even when the object of the class or the member function doing the modification is const.

What is an explicit constructor?

A conversion constructor declared with the explicit keyword. The compiler does not use an explicit constructor to implement an implied conversion of types. It’s purpose is reserved explicitly for construction.

What is the Standard Template Library?

A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification.

A programmer who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.

Describe run-time type identification.

The ability to determine at run time the type of an object by using the typeid operator or the dynamic_cast operator.

What problem does the namespace feature solve?

Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The namespace feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions.

This solution assumes that two library vendors don't use the same namespace identifier, of course.

Are there any new intrinsic (built-in) data types?

Yes. The ANSI committee added the bool intrinsic type and its true and false value keywords.

What is the difference between the HEAP and the STACK?

HEAP is used to store dynamically allocated memory (malloc). STACK stores static data (int, const).

Where in memory are HEAP and the STACK located relative to the executing program?

The STACK and HEAP are stored "below" the executing program. The HEAP "grows" toward the program executable while the STACK grows away from it.

Describe the data structures of a double-linked list.

A double-linked list structure contains one pointer to the previous record in the list and a pointer to the next record in the list plus the record data.

How do you insert a record between two nodes in double-linked list?

Previous R; Data R; Next R; To insert a record (B) between two others (A and C): Previous.B = A; Next.B = C; Next.A = B; Previous.C = B;

In which data structure, elements can be added or removed at either end, but not in the middle?

queue

Which one is faster? A binary search of an orderd set of elements in an array or a sequential search of the elements.

binary search

What is a balanced tree?

A binary tree is balanced if the depth of two subtrees of every node never differ by more than one

Which data structure is needed to convert infix notations to post fix notations?

stack

What is data structure or how would you define data structure?

In programming the term data structure refers to a scheme for organizing related piece of information. Data Structure = Organized Data + Allowed Operations.

Which data structures we can implement using link list?

queue and stack

List different types of data structures?

Link list, queue, stack, trees, files, graphs

Logic / algorithm to detect whether a loop exists in a linked list or not?

One possible answer is to add a flag to each element of the list. You could then traverse the list, starting at the head and tagging each element as you encounter it. If you ever encountered an element that was already tagged, you would know that you had already visited it and that there existed a loop in the linked list.

What if you are not allowed to alter the structure of the elements of the linked list?

The following algorithm will find the loop:

1. Start with two pointers ptr1 and ptr2.

2. Set ptr1 and ptr2 to the head of the linked list.
3. Traverse the linked list with ptr1 moving twice as fast as ptr2 (for every two elements that ptr1 advances within the list, advance ptr2 by one element).
4. Stop when ptr1 reaches the end of the list, or when ptr1 = ptr2.
5. If ptr1 and ptr2 are ever equal, then there must be a loop in the linked list. If the linked list has no loops, ptr1 should reach the end of the linked list ahead of ptr2.

What do you mean by?

* Syntax Error

* Logical Error

* Runtime Error

Syntax Error: errors generated by compiler at the time of compilation of program. Logical error: errors in program logic. Runtime error: An error that occurs during the execution of a program, Runtime errors indicate bugs in the program

---

Describe tree data structure?

A tree is a recursive structure that usually maps an ordered set of data from an internal definition to some data space. Tree parts are often named after their contemporaries in family trees; trees contain nodes known as parent, child, and sibling. Trees are made of nodes, which can contain both data to be stored and always link to further levels in the tree. Trees are often formed from a single node known as root; alternatively, trees may be built from a set of original nodes--this is known as a forest of trees

---

A queue can be implemented as a linked list or as an array?

Yes

How will you enqueue an item from a queue where linked list is used to implement queue?

A tree is a recursive structure that usually maps an ordered set of data from an internal definition to some data space. Tree parts are often named after their contemporaries in family trees; trees contain nodes known as parent, child, and sibling. Trees are made of nodes, which can contain both data to be stored and always link to further levels in the tree. Trees are often formed from a single node known as root; alternatively, trees may be built from a set of original nodes--this is known as a forest of trees

---

Differentiate data type and data structure?

Data Type = Permitted Data Values + Operations

Data Structure = Organized Data + Allowed Operations.

List the number of operations that we can perform on array data type?

Creation

Initialization

Storing an element

Retrieving an element

Inserting an element

Deleting an element

Searching for an element

Sorting elements

Printing an array

Is the above operations applies to array of any dimension?

Yes

What are Row Major Order and Column Major Order? Draw memory representation of 4 X 3 matrixes both in row major and column major?

Row Major Order: Store first the first row of the array, then the second row of the array and then the next and so on. Such a storage scheme is called Row Major Order.

Column Major Order: Store the array column by column. It is called Column Major Order.)

List different types of linked lists?

Singly, Doubly, Circular

List different type of sorting?

Bubble sort, Insertion sort, Quick sort, Heap sort, two way merge sort

---

Explain logic of bubble sort?

In this sorting algorithm, multiple swapping take place in one pass. Smaller elements move or 'bubble' up to the top of the list.
In this method adjacent members of the list to be sorted are compared. If the item on top is greater than the item immediately below it, they are swapped. This process is carried on till the list is sorted.

Explain logic of insertion sort?

Find the correct position search the list till an item just greater than the target is found. Shift all the items from this point one, down the list. Insert the target in the vacated slot

Explain FIFO and LIFO techniques?

FIFO: First in first out. The element which comes first in list will be taken out first.

LIFO: Last in first out. The last added element in list will come out first.

Which data structures use LIFO and FIFO architecture?

Stack data structure uses LIFO and Queue uses FIFO

What is linear list?

A linear list is an ordered set consisting of a variable number of elements to which addition and deletions can be made. A linear list displays the relationship of physical adjacency.

What is successor and predecessor in a linear list?

Next element to the head of the List is called its successor. Previous element to the tail (if it is not the head of the List) of the List is called its predecessor.

Can we implement linear list using either array index or pointers?

Yes

If linear list is implemented using array index then what is the nature of the list?

The nature is Static and it’s referred to as static implementation of list

The nature is Static and it’s referred to as static implementation of list

The nature is dynamic and it’s referred to as dynamic implementation of list

Define stack.

A Stack is a linear data structure in which data is inserted and deleted at one end (same end) i.e. data is stored and retrieved in a Last In First Out (LEFO) order.

Give some examples where stacks and queues can be used in computer science.

Stacks are used in compilers in passing an expression by recursion; in memory management in operating system;
Queues find their use in CPU scheduling, in printer spooling, in message queuing in computer network etc.

What do you call the store and retrieve operation for stack?

Store = Push and Retrieve = Pop

Give the name of two open ends of queue?

Front and rare

Which end of queue allows you to insert an element?

Front

Which end of queue allows you to delete an element?

Rare

What strongly connected graph?

Graph is strongly connected if there is a directed path from any vertex to any other vertex.

What is weakly connected graph?

Graph is weakly connected if there isn't any directed path from any vertex to any other vertex.

What is the difference between strongly connected and weakly connected graph?

If there is a directed path from any vertex to any other vertex then its strongly connected graph else it's weakly connected.

What is in degree and out degree of vertex?

In degree of a vertex v is the number of edges for which vertex v is a head and out degree is the number of edges for which vertex is a tail.

Tell me two properties or condition where I can say that this is a special type of graph and it is a tree.

Graph is connected and there are no cycles in the graph.

Can we implement Depth First Search traversal method for graph using stack? Explain how?

Yes. By pushing all unvisited vertices adjacent to the one just visited and popping the stack to find the next vertex to visit.

What is the logic behind BFS?

First visit all the adjacent vertices of the start vertex and then visit all -the unvisited vertices adjacent to these and so on.

What is path in graph?

A path in a graph is sequence of vertices such that there is an edge that we can follow between each consecutive pair of vertices.

How can we decide that path is a shortest minimum path?

Shortest path from vertex v to vertex w is a path for which the sum of the weights of the arcs or edges on the path is minimum.

What is single source shortest path problem?

We have a single source vertex and we seek a shortest path from this source vertex v to every other vertex of the graph. It is called single source shortest path problem.

Below is the weighed graph, give me shortest path from A to H?

1) AFDEH = 1 + 3 + 4 + 6 = 14
2) ABCEH. = 2+2+3+6 = 13. Shortest path is 2 one.

What is a spanning tree?

A tree T is a spanning tree of a connected graph G(V,E) such that
1) Every vertex of G (Graph) belongs to an edge in T and
2) The edges in T form a tree.

Graph falls under which category of data structure?

non – linear

Define tree?

A tree can be defined as a finite set of zero or more vertices such that there is one specially designated vertex called ROOT, and the remaining vertices are partitioned into a collection of sub- trees, each of which is also a tree.

What is leaf node?

A node that has no children is called a leaf node.

What are siblings, in tree data structure?

Nodes with the same parent are called siblings.

List sibling nodes in below tree.

K, L, and M and B, C, D is siblings.

What is path in tree?

A path in a tree is a list of distinct vertices in which successive vertices are connected by edges in the tree.

What are leaves and terminal nodes in tree?

Nodes with no children are called leaves, or terminal nodes.

What is binary tree?

Binary tree is a tree which is either empty or consists of a root node and two disjoint binary trees called the left subtree and right subtree.

What is degree of leaf node?

A leaf node has a degree zero, because it does not have any subtrees.

A leaf node has a degree zero, because it does not have any subtrees.

First Set:

Second Set:

first set of trees are 2-trees.
This is a special case of binary tree. It is called a 2-tree or a strictly binary tree. It is a non-empty binary tree in which either both sub trees are empty or both sub trees are 2-trees.

Give four different traversal methods or tree.

The four different traversals of T are In order, Post order, Preorder and Level-by-level traversal.

What is an expression tree?

Each operator node has exactly two branches
Each operand node has no branches, such trees are called expression trees.

Give me in order traversal of given expression tree?

A + B * C - D * E
What is the logic behind in order traversal? (This is for interviewee)
Strategy: Left-Root-Right.
In this traversal, if tree is not empty, we first traverse (in order) the left sub tree; then visit the root node of tree, and then traverse (in order) the right sub tree.

Give me the post order traversal of given expression tree?

A B C * + D E * -

Logic behind traversal: First traverse left(T) (in post order); then traverse Right(T) (in post order); and finally visit root.

Strategy: Left-Right-Root strategy, i.e.

Traverse the left sub tree In Post order

Traverse the right sub tree in Post order.P Visit the root.

Give me the preorder traversal of given expression tree?

- +A*BC*DE

Logic: Visit root first; then recursively perform preorder traversal of Left(T); followed by pre order. traversal of Right(T)

Strategy: Root-Left-Right

traversal, i.e.

Visit the root

Traverse the left sub tree preorder.

Traverse the right sub tree preorder.

Give depth first search of the given binary tree?

The order in which the nodes would be visited is

A B D E C F H I J K G

Traverse the tree given below in preorder, in order, post order and level by level giving a list of nodes visited.

Preorder: *-+EAB+CD

Inorder: E+A-B*C+D

Postorder: EA+B-CD + *-

Level by level: *- + +BCDEA