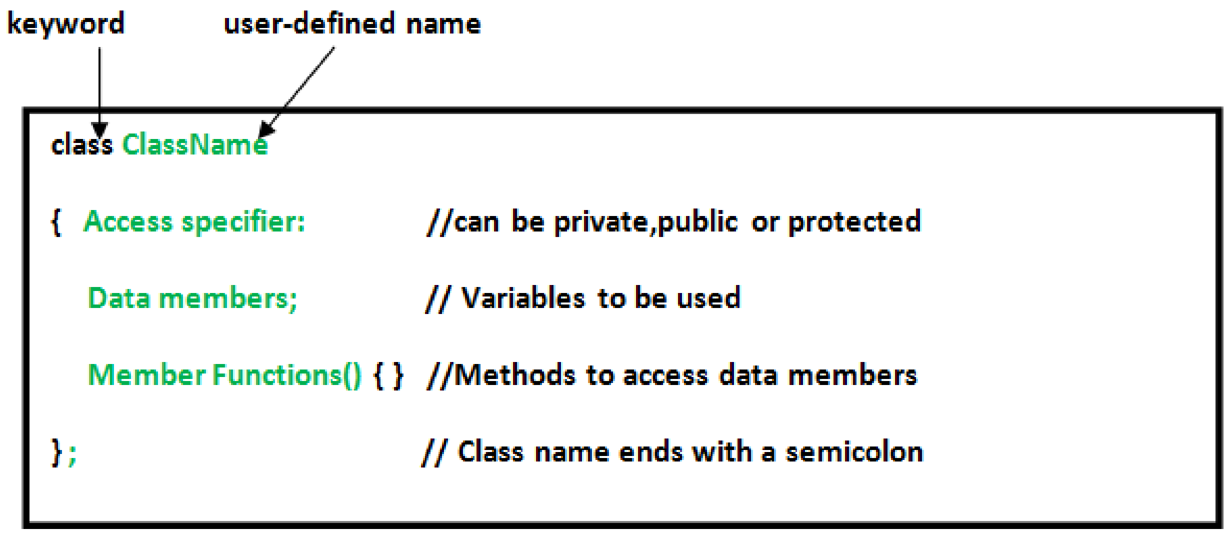
Classes and Objects
Overview of C++

Class: Class is a blueprint of data and functions or methods. Class does not take any space.

Object: Objects are basic *run-time entities in an object oriented system, objects are instances of a class these are defined user defined data types.

Run-time entities :- Any real world object is called as entity.
An entity which is having State, Behavior and Identity is called as run time entity.
Every entity has objects to act on behalf of them.
That's why objects are frequently called as run time entities.



```
class ClassName

{  Access specifier:          //can be private,public or protected

   Data members;              // Variables to be used

   Member Functions() { }     //Methods to access data members

};                            // Class name ends with a semicolon
```

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

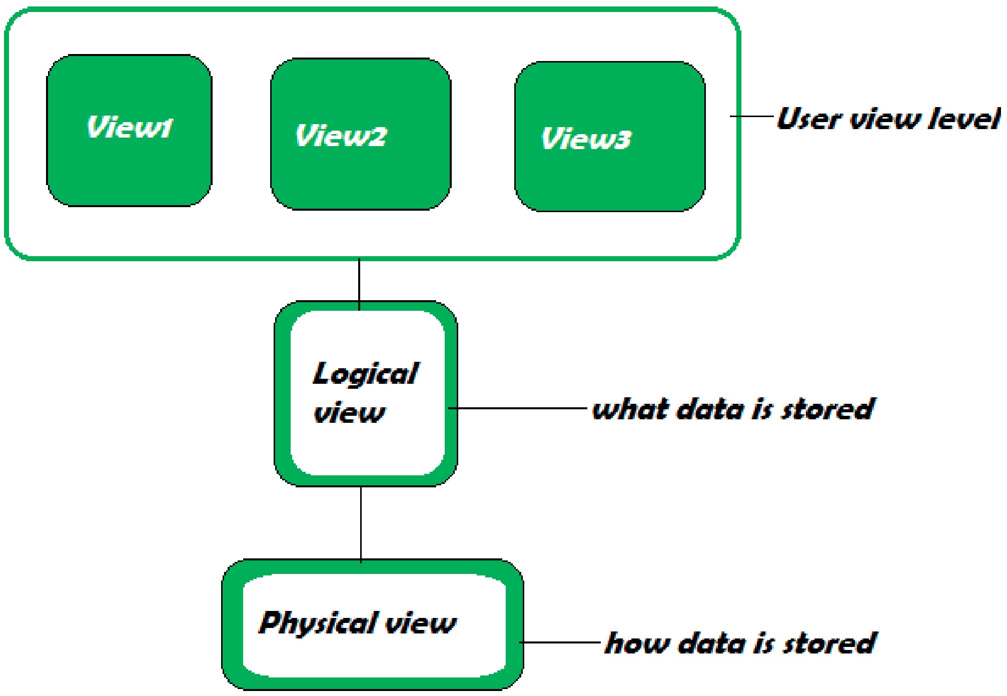
```
ClassName ObjectName;
```

In structured programing such as C. After 25k or 100k lines of codes. It is hard to grasp the totality of the program.
The essence of C++ allows programmer to comprehend and manage more complex programs.

There are four pillars of C++ or more specifically OOP
1. Data Abstraction
2. Encapsulation

3. Polymorphism
4. Inheritance

1. Data Abstraction :-
Abstaction involves the process of representing only the important features and data to the users without showing the complex details or explanation about the features.



3.**Physical Level:**

It is level where data is actually stored. It is can be described with phrase "Blocks of storage."

2.**Logical Level:**

It describes data stored in the database and the relationships between the data. The programmers generally work at this level as they are aware of the functions needed to maintain the relationships between the data.

1. **View Level:**

Application programs hide details of data types and information for security purposes. This level is generally implemented with the help of GUI, and details that are meant for the user are shown.

2. Encapsulation :-

Classes and Objects

Encapsulation is the act of binding data and functions that work with data together and keep them both safe for outside inheritance and misuse.
Eg :- Class in C++

3. Polymorphism

Polymorphism means many faces or impressionist(बहरूपिया). As its name suggest Polymorphism is the ability to process the data in more than one forms.
It consists of methods like function overloading and operator overloading.

There are two types of Polymorphism :-

1. Run-Time Polymorphism
2. Compile-Time Polymorphism

Overloading is Compile-time polymorphism, whereas Virtual function and inheritance are part of Run-time polymorphism

4. Inheritance :-

Inheritance is the ability of one class to inherit the properties from another class. Inheritance allows the reuse of code which reduces redundancy and increases efficiency.

In C++ we do not use .h header file which is new-styled headers defined by Standard C++

using namespace std;
std ---> std is standard C++ library
namespace ---> namespace creates a declarative region or place where various program elements can be placed
using ---> using statement tells compiler that you want to use std namespace.

cout ---> Is an identifier, << is operator for right shift
cin ---> Is an identifier, >> is operator for left shift

We donot have to use address of variable in C++. We will discuss it later.

We can also use EXIT_SUCCESS or EXIT_FALIURE instead of return 0

In C++ there is no .h in header file because iostream is not a header file.
iostream is just a identifier for the header file.

In case of new style header file(where we do not include .h) we have to use namespace std.

Classes and Objects

The purpose of namespace std is to localize the names of identifiers to avoid name collisions.

Classes in C++

In C++ by default members of a class are private. If a member is private only other class member can access it.
When an object of a class is created, an "instance" of the class is said to be created.
There are two ways of creating an object:-
1. Using class name as data type to define objects.
2. Creating objects before terminating the class(same as in structure).

```
Class students{
                //Body of class
              }object;
```

A Class is a logical abstraction, while an object has physical existence. i.e. An object takes up memory but not class.

Why to use static with data_types?

```
#include <stdio.h>

void foo()
{
    int a = 10;
    static int sa = 10;

    a += 5;
    sa += 5;

    printf("a = %d, sa = %d\n", a, sa);
}


int main()
{
    int i;

    for (i = 0; i < 10; ++i)
        foo();
}
```

This prints:

```
a = 15, sa = 15
a = 15, sa = 20
a = 15, sa = 25
a = 15, sa = 30
a = 15, sa = 35
a = 15, sa = 40
a = 15, sa = 45
a = 15, sa = 50
a = 15, sa = 55
a = 15, sa = 60
```

2) Static variables are allocated memory in data segment, not stack segment. See memory layout of C programs for details.

3) Static variables (like global variables) are initialized as 0 if not initialized explicitly. For example in the below program, value of x is printed as 0, while value of y is something garbage.

Scope Resolution Operator(SRO) :-
SRO has 4 uses in C++ that are:-
1. Used for defining a function outside the class.

```
Class students{
                    int func();
                    };
int students::func(){
                    cout<<"This function is defined using SRO.";
}
```

2. When global and local variable are has same name in same scope, We can use SRO to use global Variable.

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

int x;  // Global x

int main()
{
  int x = 10; // Local x
  cout << "Value of global x is " << ::x;
  cout << "\nValue of local x is " << x;
  return 0;
}
```

Output:

```
Value of global x is 0

Value of local x is 10
```

3. SRO is used to access static variable in a class eg:-

```cpp
// C++ program to show that :: can be used to access static
// members when there is a local variable with same name
#include<iostream>
using namespace std;

class Test
{
    static int x;
public:
    static int y;

    // Local parameter 'a' hides class member
    // 'a', but we can access it using ::
    void func(int x)
    {
        // We can access class's static variable
        // even if there is a local variable
        cout << "Value of static x is " << Test::x;

        cout << "\nValue of local x is " << x;
    }
};

// In C++, static members must be explicitly defined
// like this
int Test::x = 1;
int Test::y = 2;

int main()
{
    Test obj;
    int x = 3 ;
    obj.func(x);

    cout << "\nTest::y = " << Test::y;

    return 0;
}
```

Output:

```
Value of static x is 1

Value of local x is 3

Test::y = 2;
```

4. We can use SRO in case of multiple inheritance. When two ancestors have same member variable. We can use SRO to distinguish between them. Eg:-

```
// Use of scope resolution operator in multiple inheritance.
#include<iostream>
using namespace std;

class A
{
protected:
    int x;
public:
    A() { x = 10; }
};

class B
{
protected:
    int x;
public:
    B() { x = 20; }
};

class C: public A, public B
```

```
{
public:
  void fun()
  {
     cout << "A's x is " << A::x;
     cout << "\nB's x is " << B::x;
  }
};

int main()
{
   C c;
   c.fun();
   return 0;
}
```

Output:

```
A's x is 10

B's x is 20
```

Function Overloading :-

One way C++ achieves **Polymorphism** is by function overloading.

By using Funciton Oveloading C++ we can avoid writing different identifier for same function with different arguments.

For eg :-

In C consider a function **abs()** that is used to calculate the absolute value of an integer.

To calculate absolute value of float, long we have to make **fabs()** and **labs().** Which is unnecesary in C++ we use only **abs()** for every type of number, Thanks to function overlaoding.

Operator Overloading :-

In C++, almost all the operators can be oveloaded i.e. They can perform extra funcionality inclusive to their normal function. Eg:-

Classes and Objects

>> and << are basically bitwise operator for left and right shift, but they can be overloaded and can be used for console I/O operations.

We will see operator overloading later.

Inheritance :-

Inheritance is the ability of one class to inherit the properties from another class.
Inheritance allows the reuse of code which reduces redundancy and increases efficiency.
The process involves first defining **base class** which has the basic decription.
The class that inherits the functionality of the base class is known as **derived class**.
The derived class has extra funcitonality inclusive to the base class.
Base class ---> parent class, super class
Derived class ---> child class, sub class

NOTE :- In case of Inheritance, the derived class has only access to public members of the base class not the private members.

Constructor & Destructor :-

Constructor :-

- Constructor is a special function with no(including void) return type.
- Constructor has same identifier(name) as the class.
- Automatically invoked when a object is created.
- Constructor is used to intialize class members when a object is created.
- Constructor is always public.
- Constructor can be declared outside the class using **SRO**.
- Constructor are of three types:-

1. Default Constructor 2. Parameterized Cinstructor 3. Copy Constructor

Destructor:-

- Destructor is a special function with no return type and parameter.
- Has same name as a class name, defined using tilt(~).
- Automatically invoked when life of object ends i.e. Object goes out of scope.
- Destructor is always public.
- Functioning of Destructor is Complement that of Construnctor i.e. Destructs the memory allocated by constructor.
- Destructor can be declared outside the class using **SRO**.

Classes and Objects

When do we need to write a user-defined destructor?

If we do not write our own destructor in class, compiler creates a default destructor for us. The default destructor works fine unless we have dynamically allocated memory or pointer in class. When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed. This must be done to avoid memory leak.

Class Members can fall under three categories on the basis of their access.

1. Public
2. Private
3. Protected

Private access specifier/modifier :-
These class members can only be accessed by class members and friend class/function.
Eg:-
See programs

NOTE:- You cannot use register, auto, extern with class members.
Passing a string to a function.
E.g. :-

```
void fun(char *string){
        cout<<”We need character pointer to pass a string in a function”<<string
}

int main(){
        fun(“Hello World”);
        return 0;
}
```

OUTPUT:-

```
Hello World
```

Structure and Class in C++
Structure defines a special type of class in C++. Structure and classes are same in C++, the only difference is that, In structureby default structure members are public whereas in class they are private.
(Litreally there is no difference. What you can do in class, you can do in structure.)

Union and Class in C++
Union defines a special type of class in C++. This means encapsulation is preserved in Union in C++.

Classes and Objects

Restriction on Union in C++

- Cannot inherit any other classess of any type.
- Union cannot be base calss.
- Cannot have virtual function.
- Cannot have static variable members.
- Cannot have any member that overload the “=” operator.
- No object can be member of union, if that member has an explicit constructor or destructor function.

Anonymous Union :-

They are called anonymous because they do not have a type name.
Members of these classes cannot be called with the dot operator. Instead their memners can be called just like a variable.

Restriction on the Anonymous Union is same as the Union in addition to:-

- They cannot have function members
- They cannot have private or protected members. Only public is allowed.
- Global Anonymous Union will be specified as static.

Eg:-

```
int main(){
        union{     // Does not have a type name name(identifier)
                int a;
                int b;
};

        a = 10;
        b = 20;
        return 0;
}
```

NOTE :- Following is the list of operators, which can not be overloaded −
“ :: ” , “ .”, “ .* ”, “ ?: ”

Friend Functions :-

- Friend function can access the private or the protected members of the class.
- To make friend function we need to use friend keyword.
- We need to make prototype of the friend function inside the class.
- Friend function is declared without using **SRO**.

- Friend function is called just as a normal function without dot operator.
- We have to pass object(of the class where friend function is declared) as a parameter of friend function.
- Friend function cannot be inherited.

For Eg :- See program.

Why do we need friend function?

There may be times where we need a common function that is friend of two or more than two classes.

Forward declaration of a class. See book, Why we need friend function?

In case of friend classes, Class members have acces to all members of the friend class.

Inline Functions :-

We can create inline functions by using inline keyword while defining(not declaring) the function.

Inline functions are function that not actually callled, rather, their code is expanded inline at the point of each invocation.

Why to use inline function?

Inline functions are usefull becuase they are more time efficient. As the work of “overhead” is not performed when inline is used. But inline function takes more storage than the normal function, due to the fact that when function is invoked, it is replaced by its defination.

Will inline will work everytime?

Inline keyword is same as register keyword. It depends on compiler, Inline is just a request not an order. Not all functions can be inlined. Eg:-

Recursive functions, Large functions.

Classes and Objects

Defining Inline Functions with Classes

All the fucntions defined inside a class are by default inline function, We do not need to use inline keyword where function is defined inside a class itself.

Parameterized Constructor :-

It is possible to pass arguments to a constructor.

There are two ways of passing this argument to a constructor. i.e.

Syntax:-

<class name> <object name>(arguments);

class_A object(11494);

OR

<class name> <object name> = <class name>(arguments);

class_A object = class_A(11494);

Constructor with one Parameter : A special case

We can pass arguments to construcor with one parameter like:-

Synatx :-

<class name> <object name> = <argument>;

class_A object = 23;

Static Class Members

1. Static Data Members

When a data member is declared with static keyword, it means you are telling the program that only one copy of the that data member will exists. That means all the object will use the same variable.

NOTE :- When you are declaring a static member inside the class, As we know that class is just a logical abstraction, no space is allocated to the static member, So we need to define it outside the class.

Eg :-

Classes and Objects

See Program

static data members are better alternative for the global variable because global variable do not ensure **Encapsulation**.

2. Static Member Functions

Static Functions are not much used but they can be used to initialize static data member without creating objects.

When Constructor & Destructor execute?

An object constructor is called when the object comes into existence and object destructor is called when object is destroyed.

There are two types of objects :-

1. Local object
2. Global object

- Local object's constructor executes when object declaration is encounterd.
- Local object's destructor executes in reverse order of constructor(Follows LIFO).

- Global object's constructor executes before main() begins execution.
- Global object's destructor executes after main() has terminated.

Note :- We can make local classess inside a function. These classes can only be accessed inside the function and all member function must be declared inside the class body. The local classes may not have access to the local function and variables(except static and extern local variable).

Passing objects to Function

When an object is passed to a function(as an argument), a copy of that object is created and is passed to the function.

When the copy of the object is created, is constructor called then?

No constructor is not called when the object is passed as argument. Why?

Because, if the constructor was called then the data member stored in the copy_object would be changed(or intialized again) which will be a problem.

But destructor is called when the function ends. Why?

Because copy_object would be performing operations that need to call destructor.

For eg:- Allocated memory by the copy_object must be freed when its destroyed.

Returning Objects

To return objects from the function that function must have data type as same as returing object.

Eg :-

```
class A{
//
}

A function(){
return A object;
}
```

When an object is returned a temporary object is created that holds the return values.

After returning this temp object is destroyed.

Object Assignment

Values of one object can be copied to another object.

For Eg:-

```
class A{
//
}
```

```
A obj1, obj2;
obj2 = obj1;   //Assigning data from object1 to object2.
```