### Static Keyword

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following,

1. Static variable in functions
2. Static Class Objects
3. Static member Variable in class
4. Static Methods in class

#### Static variables inside Functions

Static variables when used inside function are initialized only once, and then they hold there value even through function calls.

These static variables are stored on static storage area , not in stack.

void counter()

{

static int count=0;

cout << count++;

}

int main(0

{

for(int i=0;i<5;i++)

{

counter();

}

}

Output : 0 1 2 3 4

Let's se the same program's output **without using static** variable.

void counter()

{

int count=0;

cout << count++;

}

int main(0

{

for(int i=0;i<5;i++)

{

counter();

}

}

Output : 0 0 0 0 0

If we do not use static keyword, the variable count, is reinitialized everytime when counter() function is called, and gets destroyed each time when counter() functions ends. But, if we make it static, once initialized count will have a scope till the end of main() function and it will carry its value through function calls too.

If you don't initialize a static variable, they are by default initialized to zero.

#### Static class Objects

Static keyword works in the same way for class objects too. Objects declared static are allocated storage in static storage area, and have scope till the end of program.

Static objects are also initialized using constructors like other normal objects. Assignment to zero, on using static keyword is only for primitive datatypes, not for user defined datatypes.

class Abc

{

int i;

public:

Abc()

{

i=0;

cout << "constructor";

}

~Abc()

{

cout << "destructor";

}

};

void f()

{

static Abc obj;

}

int main()

{

int x=0;

if(x==0)

{

f();

}

cout << "END";

}

Output : constructor END destructor

You must be thinking, why was destructor not called upon the end of the scope of if condition. This is because object was static, which has scope till the program lifetime, hence destructor for this object was called when main() exits.

#### Static data member in class

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.

Static member variables (data members) are not initialied using constructor, because these are not dependent on object initialization.

Also, it must be initialized explicitly, always outside the class. If not initialized, Linker will give error.

class X

{

static int i;

public:

X(){};

};

int X::i=1;

int main()

{

X obj;

cout << obj.i; // prints value of i

}

Once the definition for static data member is made, user cannot redefine it. Though, arithmetic operations can be performed on it.

#### Static Member Functions

These functions work for the class as whole rather than for a particular object of a class.

It can be called using an object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

*Example*:

class X

{

public:

static void f(){};

};

int main()

{

X::f(); // calling member function directly with class name

}

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

It doesn't have any "this" keyword which is the reason it cannot access ordinary members. We will study about "this" keyword later.

### Const Keyword

Constant is something that doesn't change. In C and C++ we use the keyword **const** to make program elements constant. Const keyword can be used in many context in a C++ program. Const keyword can be used with:

1. Variables
2. Pointers
3. Function arguments and return types
4. Class Data members
5. Class Member functions
6. Objects

#### 1) Constant Variables

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while declared.

int main

{

const int i = 10;

const int j = i+10; // Works fine

i++; // This leads to Compile time error

}

In this program we have made **i** as constant, hence if we try to change its value, compile time error is given. Though we can use it for substitution.

#### 2) Pointers with Const

Pointers can be made **const** too. When we use const with pointers, we can do it in two ways, either we can apply const to what the pointer is pointing to, or we can make the pointer itself a const.

#### Pointer to Const

This means that the pointer is pointing to a const variable.

const int\* u;

Here, u is a pointer that points to a **const int**. We can also write it like,

int const\* v;

still it has the same meaning. In this case also, v is a pointer to an int which is const.

#### Const pointer

To make the pointer const, we have to put the **const** keyword to the right of the \*.

int x = 1;

int\* const w = &x;

Here, w is a pointer, which is const, that points to an int. Now we can't change the pointer but can change the value that it points to.

**NOTE :** We can also have a const pointer pointing to a const variable.

const int\* const x;

#### 3) Const Function Arguments and Return types

We can make the return type or arguments of a function as const. Then we cannot change any of them.

void f(const int i)

{

i++; // Error

}

const int g()

{

return 1;

}

#### Some Important points to remember

1. For built in types, returning a const or non-const, doesn't make any difference.

const int h()

{

return 1;

}

it main()

{

const int j = h();

int k = h();

}

Both j and k will be assigned 1. No error will occur.

1. For user defined data types, returning const, will prevent its modification.
2. Temporary objects created while program execution are always of const type.
3. If a function has a non-const parameter, it cannot be passed a const argument while making a call.

void t(int\*) { }

If we pass a const int\* argument, it will give error.

1. But, a function which has a const type parameter, can be passed a const type argument as well as a non-const argument.

void g(const int\*) {}

This function can have a int\* as well as const int\* type argument.

#### 4) Const class Data members

These are data variables in class which are made const. They are not initialized during declaration. Their initialization occur in the constructor.

class Test

{

const int i;

public:

Test (int x)

{

i=x;

}

};

int main()

{

Test t(10);

Test s(20);

}

In this program, **i** is a const data member, in every object its independent copy is present, hence it is initialized with each object using constructor. Once initialized, it cannot be changed.

#### 5) Const class Object

When an object is declared or created with const, its data members can never be changed, during object's lifetime.

**Syntax :**

const class\_name object;

#### Const class Member function

A const member function never modifies data members in an object.

**Syntax :**

return\_type function\_name() const;

#### Example for const Object and const Member function

class X

{

int i;

public:

X(int x) // Constructor

{ i=x; }

int f() const // Constant function

{ i++; }

int g()

{ i++; }

};

int main()

{

X obj1(10); // Non const Object

const X obj2(20); // Const Object

obj1.f(); // No error

obj2.f(); // No error

cout << obj1.i << obj2.i ;

obj1.g(); // No error

obj2.g(); // Compile time error

}

Output : 10 20

Here, we can see, that const member function never changes data members of class, and it can be used with both const and non-const object. But a const object can't be used with a member function which tries to change its data members.

#### Mutable Keyword

Mutable keyword is used with member variables of class, which we want to change even if the object is of const type. Hence, mutable data members of a const objects can be modified.

class Z

{

int i;

mutable int j;

public:

Z()

{i=0; j=0;}

void f() const

{ i++; // Error

j++; // Works, because j is Mutable

}

};

int main(0

{

const Z obj;

obj.f();

}