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 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 co 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

```
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 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.

2. For user defined data types, returning const, will prevent its modification.
3. Temporary objects created while program execution are always of const type.
4. 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.

5. 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 initializatio 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 e 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++; }
```

```
 { 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 c
and non-const object. But a const object can't be used with a member function which tries to change its data membe

---

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();
}
```