`const` Keyword in C++

Constant is something that doesn't change. In C language and C++ we use the keyword `const` to make program elements constant. `const` keyword can be used in many contexts in a C++ program. It 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

A variable declared as constant, will **not allow any modifications after its initialization.** Constant variables are created using the keyword `const`

A const variable must be assigned a value at the time of its declaration.

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

main()
{
    int a1=10;      //normal variable
    const int a2=20;        //const variable

    //works fine as normal variable
    a1++;

    //this would give error as
    //we are trying to change value in const variable
    a2++;

    return 0;
}
```

Once initialized, if we try to change its value, then we will get compilation error.

if we try to change its value, we will get compile time error. Though we can use it for substitution for other variables.

2) Pointers with `const` keyword

Pointers can be declared using **const** keyword too. If we make a pointer **const**, we cannot change the pointer. 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 constant.

Constant Pointer

- If we make a pointer const, Its address cannot be changed after initialization
- This means that the pointer will always point to the same address.
- Here address of `const` pointer cant be changed but the value that is pointed can be changed

```
main()
{
  int a = 4,b=5;
  int* const ptr = &a; // const pointer p pointing to the variable a
  //ptr=&b;//invalid :const pointer address cannot be changed
  *ptr=6;//valid,value at ptrs address is non_const
}
```

Note that we cannot change the pointer p but can change the value of a.

Pointer to Constant Variable

Here, the ***value that is pointed by the pointer cannot be modified***

```
main()
{
        int a = 4,b=5;
        const int *p=&a;//a becomes constable variable
        //*p=5;//inavlid:a is const
        p=&b;//valid: address pointed by p is non_const
}
```

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.

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 functionName(const int x)

{

    x++;     // error

}
```

main.cpp

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

int add(const int x)
{
    int res=0;
    int error=20;
    res = x+error;
    return res;
}
int main()
{
  int ans = add(50);
  cout<<"Addition:"<<ans;
}
```

As seen below, if we try to manipulate the value of a const function argument, an error would be raised by the compiler.

```cpp
int add(const int x)
{   x=x+100;
    return x;
}
```

In the above example, we are trying to change the value of the const function argument “x”, resulting in a compiler error

4) Defining Class Data members as `const`

These are data variables in class which are defined using `const` keyword. They are not initialized during declaration. Their initialization is done in the constructor.

```cpp
#include<iostream>
using namespace std;
class Demo
{
    public:
    const int X = 10;

};
int main()
{
   Demo d;
   cout<<d.X;

   // d.X = 200; // Error If we modified value
}
```

5) Defining Class's Member function as `const`

A `const`member function ***cannot change any data members of the class*** and it also cannot call any non-const function

It is a ***read-only function***

To make any member function const, we add the *const* keyword after the list of the parameters after the function name.

```cpp
#include<iostream>
using namespace std;
class test
{
    public:
    int x;
    void func() const
    {
        x = 0; // this will give compilation error
    }
};
```

The above code will give us compilation error because 'func()' is a const member function of class test and we are trying to assign a value to its data member 'x'.

A const object can only call a const member function, this is because a const object cannot change the value of the data members and a const member function also cannot change the value of the data member of its class. So

We cannot make constructors const Generally, const objects initialize the values of their data members through constructors and if we make the constructor const, then the constructor would not change the values of the data members and the object would remain uninitialized.

6) Defining Class Object as `const`

We cannot modify the data members of a const object, ***only one initialization is allowed for all the copies of that object***

`const` objects are ***reads only objects***

We make an object const by writing the const keyword at the beginning of the object declaration as shown below.

main.cpp

```cpp
#include<iostream>
using namespace std;
class Demo
{
    public:
    int X = 10;

};
int main()
{
   const Demo d;
   cout<<d.X;

}
```

We made the object a of class Demo 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.

Below Code throws an error id modified anything in class

main.cpp

```cpp
#include<iostream>
using namespace std;
class Demo
{
    public:
    int X = 10;

};
int main()
{
   const Demo d;
   d.X=100;
   cout<<d.X;

}
```

input

Compilation failed due to following error(s).

```
main.cpp: In function 'int main()':
main.cpp:12:8: error: assignment of member 'Demo::X' in read-only object
    d.X=100;
        ^~~
```