Section 20 : Constants, Preprocessor Directives and Namespaces

15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 20 : Constants, Preprocessor Directives and Namespaces

Constant :

- Const int x = 10;
  - x++ or x = 5 is not allowed.
  - It's a constant identifier, not a variable.
  - When we said constant, it means we cannot modify its value.
  - Int count x = 10 is same.
  - Int *ptr = &x is not allowed
    - We cannot store the addresses of constant identifiers to the pointer.
  - But we can have a const pointer : const int *ptr = &x;
- So constant identifiers cannot be modified throughout the program.
- We also use #define x 10, what the difference?
  - That is preprocessor directives and it is executed and it is performed before the compilation process starts. It's just a symbolic constant. That's not consume memory. That's not part of language, outside/pre compiler. So if we have any constraints that are globally used in the project, then we can use this.
  - Where const int x = 10; is a constant identifier. It will consume memory. Part of program and compiler. If we have constant inside the function or class, then we can use this type.

Constant Pointers (Pointer to a Constant) :

- Const int *ptr = &x this means :
  - this pointer ptr can point on x and it can access x, read x but it cannot modify x.
  - So the pointer cannot modify the data because it will treat the data as a constant.
- Int const *ptr = &x is same as const int *ptr = &x.
- We can make a pointter to point on something else, some other data, but still we cannot modify the data.

Constant Pointer of type Integer :

- Int *const ptr = &x, this means data is not constant ptr is constant.
  - It means we ptr is pointing to x, we can't change it to point to something else.
  - Now data is not locked, pointer is locked.

Constant pointer to integer constant :

- Const int * const ptr = &x
- So this pointer cannot be modified to point to any other data, and even it cannot modify the data also.
- Both data and pointe is locked.

Const in Functions :

- If we want to restrict a function to modify the data members of a class, we can put a const in function definition.
  - Void Display() const {…}

Const in Call by reference :

- If we want call by reference, in call by reference memory is same for the variables so a function can modify the actual values. But we don't want it, so we can make the call by reference argument as constant.
  - Void fun(const int &x, int &y);
- Parameters can also be made as constants.

Preprocessor Directives/Macros :

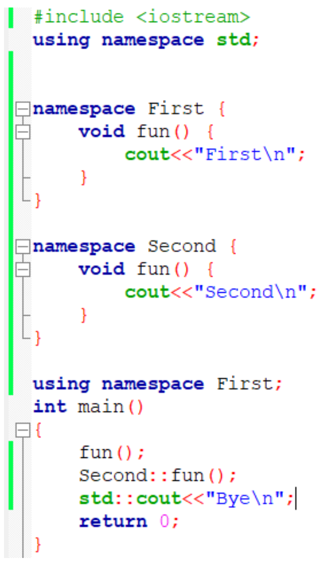
- These are instruction to compilers.
- We use #define for defining constants.
  - #define PI 3.14
  - Wherever we use PI, the value 3.14 will be replaced.
  - The compiler will see everywhere PI as 3.14
  - In machine code it's not cout<<PI, but it's cout<<3.14
  - #define C count
    - Cout<<10; is same as C<<10;
- We can also define functions using #define
  - #define SQR(x) (x*x)
  - Wherever we write SQR(5), it will be replaced as 5*5 before the compilation process starts (Pre compiler preprocessor directives).
- #define MSG(x) #x
  - It means whatever x will be it will be a string (in double quotes)
  - Cout<<MSG(Hello);   // is equal to "Hello"

-
  ```
  #ifndef
          #define PI 3.1425
  #endif
  ```

  (define only if it's not define, if we don't use #ifndef and we already define Pi earlier it will give error, so it's recommend to use #ifndef)

Namespaces :

- If we have 2 function will same name, it will give compiler error.
- To remove this ambiguity we can use namespaces.



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


namespace First {
    void fun() {
        cout<<"First\n";
    }
}

namespace Second {
    void fun() {
        cout<<"Second\n";
    }
}

using namespace First;
int main()
{
    fun();
    Second::fun();
    std::cout<<"Bye\n";
    return 0;
}
```

- We can keep these namespaces in separate file and we can include the header file and use that namespace in our main function or other part of the program.

- Scope operator :: is used to signify the namespace.
- Namespace is used to group class, objects and functions.
- Use of Namespace : To structure a program into logical units.
- The general syntax for accessing the namespace variable : namespace:operator.
- keyword is used to access the variable in the namespace : using.