Inline Function and Macros in C++

Md. Alamgir Hossain

Senior Lecturer, Dept. of CSE

Prime University

Mail: *alamgir.cse14.just@gmail.com*

Macros

- A macro is a piece of code in a program that is ***replaced by the value*** of the macro.
- We can define the macro by using the ***#define directive***. Whenever a compiler encounters the macro name, ***it replaces it with the definition of the macro.***
- There is no need to ***terminate the macro definition using a semi-colon (;).***

Macros

```
1  #include <iostream>
2  using namespace std;
3  #define dept "CSE"//Macro
4  #define res 3.1416//Macro
5  int main()
6  {
7      // Print the value of macro defined
8      cout <<"The value of Dept. is: "<<dept<<endl;
9      cout<<"Result is: "<<res<<endl;
10     return 0;
11 }
12
13 ======Output=====
14 The value of Dept. is: CSE
15 Result is: 3.1416
```

Types of Macros: Chain Macros

- Chain macros are defined as the macros inside the macros. The parent macro is expanded in the first instance and then the child macro is expanded.

```
1  #include <iostream>
2  using namespace std;
3  #define PRIME CSE //Parent Macros
4  #define CSE "It's a department" //Child Macros
5
6  int main()
7  {
8      cout <<"The value of CSE is: "<<CSE<<endl;
9      return 0;
10 }
11 /*
12 ======Output=====
13 The value of CSE is: It's a department
14 */
```

Types of Macros: Object-like Macros

- An object-like macro is like an object in code that uses it. It is popularly used to replace a symbolic name with numerical/variable represented as a constant.

```
1  #include <iostream>
2  using namespace std;
3  #define DATE 1 //Definition of Macro
4  int main()
5  {
6      cout <<"Fall semester will be started at: "<<DATE<<" October"<<endl;
7      return 0;
8  }
9  /*
10 ======Output=====
11 Fall semester will be started at: 1 October
12 */
```

Types of Macros: Function-like Macros

- Function-like macros work the same as the function call. It is used to replace the entire code instead of the function name.

```
1  #include <iostream>
2  using namespace std;
3  #define PI 3.1416
4  #define AREA(r)(PI*(r)*(r))
5
6  int main() {
7      double Radius = 3.5;
8      cout<<"Area of Circle is: "<<AREA(Radius)<<endl;
9      return 0;
10 }
11 /*
12 ======Output=====
13 Area of Circle is: 38.4846
14 */
```

Types of Macros: Multi-line Macros

- An object-line macro may be of multiple lines. Therefore, if we want to create a multi-line macro, we have to use the backslash-newline.

```
#include <iostream>
using namespace std;
 #define values 10,\
                20,\
                30,\
                40,\
                50,\

int main()
{
    int arr[] = {values};
    cout<<"Elements of Values: ";

    for (int i = 0; i < 5; i++) {
        cout << arr[i] << ' ';
    }
    cout<<endl;
    return 0;
}
/*
======Output=====
Elements of Values: 10 20 30 40 50
*/
```

Types of Macros: Multi-line Macros

- Speed versus size The main benefit of using macros is faster execution time. During preprocessing, a macro is expanded (replaced by its definition) inline each time it is used. A function definition occurs only once regardless of how many times it is called. Macros may increase code size but do not have the overhead associated with function calls.
- Function evaluation A function evaluates to an address; a macro does not. Thus you cannot use a macro name in contexts requiring a pointer. For instance, you can declare a pointer to a function, but not a pointer to a macro.
- Type-checking When you declare a function, the compiler can check the argument types. Because you cannot declare a macro, the compiler cannot check macro argument types; although it can check the number of arguments you pass to a macro.

Inline Function

- Inline function is a function that is expanded in line when it is called.
- When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call.
- This substitution is performed by the C++ compiler at compile time.

Inline Function

```
1  #include <iostream>
2  using namespace std;
3  inline int sum(int num1, int num2)
4  {
5      int res = num1 + num2;
6      return res;
7  }
8  int main()
9  {
10     cout<<"Summation is: "<<sum(10, 20)<<endl;
11 }
12 /*
13 ======Output=====
14 Summation is: 30
15 */
```

Circumstances when compiler not working

- If a function contains a loop. (for, while, do-while)
- If a function contains static variables.
- If a function is recursive.
- If a function return type is other than void, and the return statement doesn't exist in function body.
- If a function contains switch or goto statement.

Advantages of Inline Function

- Function call overhead doesn't occur.
- It also saves the overhead of push/pop variables on the stack when function is called.
- It also saves overhead of a return call from a function.
- When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context.
- Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

Good News about Inline Function in C++

- All the functions defined inside the class are implicitly inline.
- ***Inline function is best without using macros.***

Thank You