techdifferences.com

Difference Between Inline and Macro in C++ (with Comparison Chart) - Tech Differences

7-9 minutes

Macro is an instruction which expands at the time of its invocation. Functions can also be defined, like macros. Similarly, the inline functions also expand at the point of its invocation. One primary difference between inline and macro function is that the **inline functions** are expanded during **compilation**, and the **macros** are expanded when the program is processed by the **preprocessor**.

Let’s study the difference between inline and macro with the help of a comparison chart.

Content: Inline Vs Macro

Comparison Chart

Basis for Comparison	Inline	Macro
Basic	Inline functions are parsed by the compiler.	Macros are expanded by the preprocessor.
Syntax	inline return_type funct_name (parameters){ . . . }	#define macro_name char_sequence
Keywords Used	inline	#define
Defined	It can be defined inside or outside the class.	It is always defined at the start of the program.
Evaluation	It evaluates the argument only once.	It evaluates the argument each time it is used in the code.
Expansion	The compiler may not inline and expand all the functions.	Macros are always expanded.
Automation	The short functions, defined inside the class are automatically made onto inline functions.	Macros should be defined specifically.
Accessing	An inline member function can access the	Macros can never be the members of the

Basis for Comparison	Inline	Macro
	data members of the class.	class and can not access the data members of the class.
Termination	Definition of inline function terminates with the curly brackets at the end of the inline function.	Definition of macro terminates with the new line.
Debugging	Debugging is easy for an inline function as error checking is done during compilation.	Debugging becomes difficult for macros as error checking does not occur during compilation.
Binding	An inline function binds all the statements in the body of the function very well as the body of the function start and ends with the curly brackets.	A macro faces the binding problem if it has more than one statement, as it has no termination symbol.

Definition of Inline

An inline function looks like a regular function but, is preceded by the keyword “**inline**“. Inline functions are short length functions which are expanded at the point of its invocation, instead of being called. Let’s understand inline functions with an example.

inline void initialize(int x, int y){ a=x; b=y }

void display(){ cout<< a <<" "<<b << \n; } //automatic inline

#include <iostream> using namespace std; class example{ int a, b; public: inline void initialize(int x, int y){ a=x; b=y } void display(){ cout<< a <<" "<<b << \n; } //automatic inline }; int main( ) { example e; e. initialize(10, 20); e.display(); }

```
#include <iostream>
using namespace std;
class example{
int a, b;
public:
inline void initialize( int x, int y){ a=x; b=y }
void display(){ cout<< a <<" "<<b << \n; }
//automatic inline
};
int main( )
{
example e;
e. initialize(10, 20);
e.display();
}
```

In above program, I declared and defined, the function initialize( ), as an inline function in the class “example”. The code of the initialization( ) function will expand where it is invoked by the object of the class “example”.The function display( ), defined in the class example is not declared inline but it may be considered inline by the compiler, as in C++ the function defined inside the class are automatically made inline by the compiler considering the length of the function.

- The inline function reduces the overhead of function calling and returning which in turn reduces the time of execution of the program. Also, the arguments are pushed onto the stack and registers are saved when a function is called and reset when the function return, which takes time, this is avoided by the inline functions as there is no need of creating local variables and formal parameters each time.
- Inline functions can be a member of the class and can also access the data member of the class.
- Inline function reduces the time of execution of the program but, sometimes if the length of the inline function is greater then, the size of the program will also increase because of the duplicated code. Hence, it is a good practice to inline very small functions.
- The inline function's argument is evaluated only once.

Definition of Macro

Macro is a "preprocessors directive". Before compilation, the program is examined by the preprocessor and where ever it finds the macro in the program, it replaces that macro by its definition. Hence, the macro is considered as the "text replacement". Let us study macro with an example.

```
#define GREATER(a, b) ((a < b) ? b : a)

cout << "Greater of 10 and 20 is " << GREATER("20", "10") <<
"\n";
```

#include <stdio.h> #define GREATER(a, b) ((a < b) ? b : a) int main(void) { cout << "Greater of 10 and 20 is " << GREATER("20", "10") << "\n"; return 0; }

```
#include <stdio.h>
#define GREATER(a, b) ((a < b) ? b : a)
int main( void)
{
  cout << "Greater of 10 and 20 is " <<
GREATER("20", "10") << "\n";
    return 0;
}
```

In above code, I declared a macro function GREATER( ), which compares and find the greater number of both the parameters. You can observe that there is no semicolon to terminate the macro as the macro is terminated only by the new line. As a macro is a just a text replacement, it will expand the code of macro where it is invoked.

- The macros are always defined in the capital letters just to make it easy for the programmers to identify all the macros in the program while reading.
- The macro can never be a class's member function, nor it can access the data members of any class.
- The macro function evaluates the argument each time it appears in its definition, which results in an unexpected result.
- Macro must be of a smaller size as the larger macros will unnecessarily increase the size of the code.

1. The basic difference between inline and macro is that an inline

functions are parsed by the compiler whereas, the macros in a program are expanded by preprocessor.

2. The keyword used to define an inline function is “**inline**” whereas, the keyword used to define a macro is “**#define**“.
3. Once the inline function is declared inside a class, it can be defined either inside a class or outside a class. On the other hand, a macro is always defined at the start of the program.
4. The argument passed to the inline functions are evaluated only once while compilation whereas, the macros argument are evaluated each time a macro is used in the code.
5. The compiler may not inline and expand all the functions defined inside a class. On the other hand, macros are always expanded.
6. The short function that are defined inside a class without inline keyword are automatically made inline functions. On the other hand, Macro should be defined specifically.
7. A function that is inline can access the members of the class, whereas, a macro can never access the members of the class.
8. To terminate an inline function, a closing curly brace is required whereas, a macro is terminated with the start of a new line.
9. Debugging become easy for inline function as it is checked during compilation for any error. On the other hand, a macro is not checked while compilation so, debugging a macro becomes difficult.
10. Being a function an inline function bind its members within a start and closing curly braces. On the other hand, macro does not have any termination symbol so; binding becomes difficult when macro contains more that one statement.

Conclusion

The inline functions are far more convincing than macro function. C++ also provides a better way to defining a constant, which uses a “const” keyword.