C Inline Functions and Memory Use

Asked 16 years, 1 month ago Modified 15 years, 10 months ago

Viewed 10k times

7

If I use inline functions, does the memory usage increase?

c inline-method

Share

Improve this question

Follow

edited Nov 16, 2008 at 9:57

The Archetypal Paul

41.7k ● 20 ● 106 ● 135

asked Nov 11, 2008 at 6:38

Naveen kumar

In the general case this is un-answerable. If you have a specific example try it and see what happens. – Loki Astari Nov 11, 2008 at 7:16

8 Answers

Sorted by: Highest score (default)

10

There are two kinds of memory usage that inline functions will affect:

code size — in general, inlining code will increase how much memory is used to load your program. This is because there will be multiple copies of the generated code scattered around your program. However, this isn't always true -- if your inlined function was only used once, there's little change, and if the inlined function is very small, you could get a net reduction in code size by removing the function call overhead. Also, the function may be reduced in size by the optimizer that's able to remove code that's not used in the particular inline invocation.

stack usage — If your inlined functions have lots of local variables, then you may use more stack space. In C, the compiler usually allocates the stack space for a function once upon entry to the function. This has to be large enough to hold all the local variables that aren't stored in registers. If you call a function out-of-line, the stack for that function is used until it returns, when it's released again. If you inline the function, then that stack space will remain used for the whole life of the uber-function.

Inlining won't affect heap usage, as the same allocations and deallocations would occur for the inlined code as would occur for the non-inlined version.

Ben Combee
17.4k ● 6 ● 42 ● 42

6

There is another point you have to consider:

Using inline functions, the compiler is able to see where variables of the caller are going to be used as variables in the callee. The compiler can optimize out (often this is really many assembler lines that can be omitted. look out for the so called "aliasing problem") redundant code based on that knowledge. So your "code bloat" is often not all that big, especially if you have smaller functions it can even reduce bloat as Jim stated above.

Someone made a good point: Better make the compiler decide whether it inlines the function in question or not, since it knows the code it generates better than you ever would.

Share Improve this answer Follow

edited Nov 11, 2008 at 7:11

answered Nov 11, 2008 at 6:51

Johannes Schaub - litb
506k ● 131 ● 917 ● 1.2k

2

Depends on the function. Simple one-liners could have a memory reduction since no callstack needs to be setup and cleaned and no function call is made. If the function is larger than this overhead needed to call a function, then it will of course bloat the code.

Share Improve this answer Follow

answered Nov 11, 2008 at 6:39

Jim Buck
20.7k ● 11 ● 60 ● 75

2

That is really un-answerable in the general case.

To start with you do not generally have control over the in-lining. Even if you mark a function inline it is actually still up to the compiler wither it will actually do the in-lining (it is just a hint).

The compiler will do its best to optimize the code; using in-lining is just one tool in doing this. So inlining short functions will make the code smaller (as you don't need to set up the parameters for the call or retrieve the return value. But even with long functions the answer is not absolute.

If the compiler decides to inline a long function then you would think the code would get longer. But this is not generally the case; as this gives the compiler extra opportunities to apply other optimization techniques that could potentially make the code still smaller. If the compilers analysis finds that the resulting code bloat is detrimental to the code the in-lining will not be done.

Basically the compiler does its analysis and decides and the best course of action.

Conclusion. Don't worry about it. The compiler is smarter than you and will do the correct thing.

Share Improve this answer Follow

answered Nov 11, 2008 at 7:14

Loki Astari
264k ● 86 ● 342 ● 571

1

Inline functions definitely increase the size of your final executable(or binary), because they will be "copy-pasted" whereever you call them.

Share Improve this answer Follow

answered Nov 11, 2008 at 6:40

nullDev
11.5k ● 8 ● 37 ● 53

I'm sorry but your answer is incomplete and is included in the answers above. – Randy Stegbauer Nov 11, 2008 at 18:55

0

You program will in the general case become larger (I'm sure there are exceptions, though). The runtime memory consumption might go down, but not by much.

Why are you asking? Normally, you let the compiler determine whether a function should be inlined or not; it can usually make a better call given the size and complexity of the function.

Share Improve this answer

answered Nov 11, 2008 at 6:49

Follow

JesperE
64.3k ● 22 ● 142 ● 199

0

A function call requires several processor instructions.

You usually need a PUSH instruction for every argument to the function, a CALL instruction to call the function, and often another instruction that cleans up the stack after the function call.

Also, functions may modify the processor's registers, so the calling function may need more instructions to preserve registers or reload values that would otherwise still be in registers.

So if the function you're calling is just a few instructions long, inlining it can save memory *and* run faster.

That said, inlining is for when your profiler tells you that you should.

Share Improve this answer

Follow

answered Nov 11, 2008 at 6:57

Artelius
49k ● 13 ● 92 ● 106

C defines an ABI where all parameters are pushed to the stack. C++ does not define a specific ABI, allowing compiler implementations to use registers for parameter passing. So in most situations not all parameters will be pushed to the stack. – Loki Astari Nov 12, 2008 at 12:20

0

It sometimes so happens that we have functions scattered all over the program.In this case a function call causes the program to jumps to the address of the function and come back when the function call terminates. This takes away some precious time.

The above problem can be resolved with the use of inline functions. This causes the compiler to call the code directly from the source. No new memory instruction set is created for the inline function code.

Although the inline declaration in c++ is free and occurs automatically when the function is defined in the declaration, in c it is restricted by the following rules ::

1. In C, any function with internal linkage can be declared inline, but a function with external linkage is has restrictions on inline.
2. If the inline keyword is used in the function declaration, then the function definition should be present in the same translation unit.

inline datatype function_name(arguments)

This code runs upto 30% fastert than a non inline function, the rest depending on the prcessor speed.

Now comes the strategy part. You may use inline functions at your will but keeping in mind that inline functions can take much less time to execute but they have a high memory occupancy on the run. Also the compiler has always an option to overlook your inline declaration if the code declared inline is abnormally large compared to the code size.

Inline declaration although destroys the order of evaluation, does not make the function internal. The function is still external.

Share Improve this answer Follow

answered Nov 11, 2008 at 7:15

balaweblog

15.4k ● 28 ● 76 ● 95