PREFIX VERSUS POSTFIX IN C++

Tom Nealis

October 2015

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

Approved for public release; distribution is unlimited.
The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy this report when no longer needed by any method that will prevent disclosure of its contents or reconstruction of the document. Do not return to the originator.
Many coders today do not take the time to consider the implications of the code they write. Not all code is created equal, and something as seemingly harmless as incrementing or decrementing via prefix instead of a postfix notation can have a considerable effect on performance. Modern day compilers can and do optimize certain common instances of code involving this notation, but it should not be relied upon in a well-developed and maintained code base.
CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Methodology</td>
<td>1</td>
</tr>
<tr>
<td>Conclusions</td>
<td>3</td>
</tr>
<tr>
<td>Distribution List</td>
<td>5</td>
</tr>
</tbody>
</table>
INTRODUCTION

Compilers today have become very good at optimizing code that has not been written in the most efficient manner possible. Many coders often take this for granted and do not spend time concerning themselves with the performance of their code and mistakenly rely on compilers to detect and correct inefficiencies. One simple example of why coders should pay attention and not rely on compilers to do the thinking for them is when to use prefix or postfix in their code.

Most coders coming out of school today all know the basic difference between these two lines of code:

```cpp
function(++variable);
function(variable++);
```

The basic difference is that the first function call will be sent an incremented variable, whereas the second one will receive the current value of the variable and then the variable will be incremented upon return from the function. So, many coders will be comfortable with that knowledge but not think there is any difference between the next two lines of code:

```cpp
variable++;
++variable;
```

In the end, both of these lines of code will increment the variable, but the concern is how.

METHODOLOGY

In order to understand the difference between these two notations, what is produced by the compiler must be discussed. Without optimization, the compiler must create a copy in order to accomplish a postfix increment or decrement. The prefix does not require this and is, therefore, more efficient. Most modern compilers can detect and optimize the simple cases like the cases involving basic built-in types. This should not be relied upon and it should be a habit to always use prefix unless specifically needed to postfix. Take for example the following code:

```cpp
for(int i = 0; i < SomeNum; i++) { doAnything; }
```

Most college professors and books will show loops written in this way. So, coders that have seen loops mostly written in this way will continue to write them in the same fashion. It is not necessary to postfix increment for this loop. Even though most compilers will optimize this properly in most cases, this should always be written for loop:

```cpp
for(int i = 0; i < SomeNum; ++i) { doAnything; }
```

So let's take a look at some assembly. Modern compilers will produce the following after they optimize this code:

```
//prefix built in type
; 21 : for(auto i = 0u; i < 10000; ++i)
mov DWORD PTR _i$1[ebp], 0
jmp SHORT $LN3@wmain
mov eax, DWORD PTR _i$1[ebp]
add eax, 1
mov DWORD PTR _i$1[ebp], eax
cmp DWORD PTR _i$1[ebp], 10000; 00002710H
jae SHORT $LN1@wmain
```

Approved for public release; distribution is unlimited.
As one can see, the optimized code is exactly the same. The following loops are an example of code that is a little trickier for the compiler to optimize:

```cpp
auto& it = my_ints.begin();
while(it != my_ints.end())
    it++;
```

The variable ‘it’ is a vector iterator. The prefix and postfix increment line of code produces the following assembly code:

```
//iterator prefix
00F755E2  mov         ecx, dword ptr[it]
00F755E5  call        std::_Vector_iterator<std::_Vector_val<std::_Simple_types<unsigned int> > >::operator++ (0F711F9h)
00F755EA  jmp         wmain + 0EEh (0F7558Eh)
```

```
//iterator postfix
002E5A12  push        0
002E5A14  lea         eax, [ebp - 17Ch]
002E5A18  push        eax
002E5A1B  mov         ecx, dword ptr[it]
002E5A1E  call        std::_Vector_iterator<std::_Vector_val<std::_Simple_types<unsigned int> > >::operator++ (02E10FFh)
002E5A23  lea         ecx, [ebp - 17Ch]
002E5A29  call        std::_Vector_iterator<std::_Vector_val<std::_Simple_types<unsigned int> > > >::operator++ (02E119Ah)
002E5A2E  jmp         wmain + 0EEh (02E59BEh)
```

As one can clearly see, the compiler was unable to optimize the postfix. It had to create the copy. Figure 1 displays how long it takes to run through the previous code for a certain number of iterations.
The C++ coders need to take the time to understand implications of the code that they create. Some of the most benign looking code can have a significant impact on the performance of a piece of software that can, in turn, affect the device/system that is running it. An easily addressable example of this is the prefix and postfix notation. A coder should always use prefix notation unless they have to use postfix.
DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
    RDAR-WSF-M, T. Nealis
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org
REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

Prefix Vs. Postfix in C++
Title
Thomas M. Nealis
Author/Project Engineer
X8048 31

Extension Building

Date received by LCSD
Report number (to be assigned by LCSD)
RDAR-WSF-M

Author's/Project Engineers Office (Division, Laboratory, Symbol)

PART 1. Must be signed before the report can be edited.

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A_X___, B___, C___, D___, E___, F___ or X___ for the reason checked on the continuation of this form. Reason: Operational Use

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

Division Chief (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Division Chief (Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 05

Patricia Alameda
Patricia Alameda
Andrew Pskowski