Technical Report ARWSE-TR-14023

CSTRING CONCATENATION

Tom Nealis

September 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.
**REPORT DOCUMENTATION PAGE**

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden to Department of Defense, Washington Headquarters Services Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

<table>
<thead>
<tr>
<th>1. REPORT DATE (DD-MM-YYYY)</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED (From - To)</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 2015</td>
<td>Final</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
<th>5a. CONTRACT NUMBER</th>
<th>5b. GRANT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSTRING CONCATENATION</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHORS</th>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom Nealis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S. Army ARDEC, WSEC</td>
<td></td>
</tr>
<tr>
<td>Fire Control Systems &amp; Technology Directorate</td>
<td></td>
</tr>
<tr>
<td>(RDAR-WSF-M)</td>
<td></td>
</tr>
<tr>
<td>Picatinny Arsenal, NJ 07806-5000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</th>
<th>10. SPONSOR/MONITOR’S ACRONYM(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S. Army ARDEC, ESIC</td>
<td></td>
</tr>
<tr>
<td>Knowledge &amp; Process Management (RDAR-EIK)</td>
<td></td>
</tr>
<tr>
<td>Picatinny Arsenal, NJ 07806-5000</td>
<td></td>
</tr>
</tbody>
</table>

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

Technical Report ARWSE-TR-14023

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Concatenating two or more strings together while developing a C++ application is a very common task. For CStrings, there are two primary ways to concatenate strings. The first is to use the += operator to concatenate two strings, and the second is to use the + operator. This report compares the two operations.

15. SUBJECT TERMS

std::string Append

16. SECURITY CLASSIFICATION OF:

<table>
<thead>
<tr>
<th>a. REPORT U</th>
<th>b. ABSTRACT U</th>
<th>c. THIS PAGE U</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAR</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

17. LIMITATION OF ABSTRACT

SAR

18. NUMBER OF PAGES

13

19a. NAME OF RESPONSIBLE PERSON

Tom Nealis

19b. TELEPHONE NUMBER (Include area code)

(973) 724-8048

Standard Form 298 (Rev. 8/98)  
Prescribed by ANSI Std. Z39.18
## 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>6</td>
</tr>
<tr>
<td>References</td>
<td>7</td>
</tr>
<tr>
<td>Distribution List</td>
<td>9</td>
</tr>
</tbody>
</table>
INTRODUCTION

Concatenating two or more strings together while developing a C++ application is a very common task. CStrings provide two operators for concatenating strings. The first method is to use the += operator, and the second is to use the + operator. This report will analyze and compare the two operations.

In another report on appending std::strings together, it was found that it was more efficient to use the += operator instead of the + operator (ref. 1). This then led to the question of whether the CString class operated the same. It turns out that the CString performs about the same for both operators when only dealing with about 3 to 4 strings. Once there are more strings, then the += operator starts to outperform the + operator.

METHODOLOGY

In order to acquire data for this report, the following program was written, which would concatenate a certain number of strings using the += operator and also concatenate the same strings using the + operator. I collected data for concatenating 2 to 10 strings. The source code for this program is shown on the following pages:

```cpp
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
    int nRetCode = 0;

    HMODULE hModule = ::GetModuleHandle(NULL);

    if(hModule != NULL)
    {
        // initialize MFC and print and error on failure
        if(!AfxWinInit(hModule, NULL, ::GetCommandLine(), 0))
        {
            // TODO: change error code to suit your needs
            _tprintf(_T("Fatal Error: MFC initialization failed\n"));
            nRetCode = 1;
        }
    }
    else
    {
        LARGE_INTEGER frequency;
        QueryPerformanceFrequency(&frequency);

        LARGE_INTEGER starting_time, ending_time, elapsed_microseconds;

        //std::ofstream a_file("outfile2.txt");
        //std::ofstream a_file("outfile3.txt");
        //std::ofstream a_file("outfile4.txt");
        //std::ofstream a_file("outfile5.txt");
        //std::ofstream a_file("outfile6.txt");
        //std::ofstream a_file("outfile7.txt");
        //std::ofstream a_file("outfile8.txt");
        //std::ofstream a_file("outfile9.txt");
        std::ofstream a_file("outfile9.txt");
    }

    return nRetCode;
}
```
//setup strings here
std::vector<CString> my_strings;
my_strings.push_back(_T("This is the first.");
my_strings.push_back(_T("This is the second.");
my_strings.push_back(_T("This is the third.");
my_strings.push_back(_T("This is the fourth.");
my_strings.push_back(_T("This is the fifth.");
my_strings.push_back(_T("This is the sixth.");
my_strings.push_back(_T("This is the seventh.");
my_strings.push_back(_T("This is the eighth.");
my_strings.push_back(_T("This is the nineth.");
my_strings.push_back(_T("This is the tenth.");

CString plus_equal;
CString plus_plus;

for(auto i = 0u; i < 10; ++i)
{
    plus_equal = _T("");
    QueryPerformanceCounter(&starting_time);

    //code to measure here
    plus_equal = my_strings[0];
    plus_equal += my_strings[1];
    plus_equal += my_strings[2];
    plus_equal += my_strings[3];
    plus_equal += my_strings[4];
    plus_equal += my_strings[5];
    plus_equal += my_strings[6];
    plus_equal += my_strings[7];
    plus_equal += my_strings[8];
    plus_equal += my_strings[9];

    QueryPerformanceCounter(&ending_time);
    elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    //this time is in micro seconds
    auto te_plus_equal = static_cast<double>((elapsed_microseconds.QuadPart * 
        1000000.0) / frequency.QuadPart);

    plus_plus = _T("");
    QueryPerformanceCounter(&starting_time);

    //code to measure here
    //plus_plus = my_strings[0] + my_strings[1];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2];
    //    my_strings[4];
    //    my_strings[4] + my_strings[5];
The code is very straightforward. Sections need to be commented out depending on the results that are desired. The built-in high resolution counters are used in order to measure how long the concatenation took. The results are logged to the output file for later processing.

After running this program for each of the results desired, the results are shown in figure 1.
Figure 1 shows that for only a few CStrings, there was a negligible effect on performance. Once there was about four CStrings, there was a noticeable difference starting to emerge. As with the std::string, one would tend to use the += instead of the + operator.

Let's take a look at the compiler generated assembly code in order to get a better idea why the measured results were received. For appending three CStrings, the assembly code is as follows:

```assembly
plus_equal = my_strings[0];
00EEE006 push 0
00EEE008 lea ecx,[ebp-128h]
00EEE00E call std::vector<ATL::CStringT<wchar_t,StrTraitMFC_DLL<wchar_t,ATL::ChTraitsCRT<wchar_t> > >,
   std::allocator<ATL::CStringT<wchar_t,StrTraitMFC_DLL<wchar_t,ATL::ChTraitsCRT<wchar_t> > > >::operator[] (0EE1258h)
00EEE013 mov esi,esp
00EEE015 push eax
00EEE016 lea ecx,[ebp-134h]
00EEE01C call dword ptr ds:[OF0541Ch]
00EEE022 cmp esi,esp
00EEE024 call _RTC_CheckEsp (0EE1843h)
   plus_equal += my_strings[1];
00EEE029 push 1
00EEE02B lea ecx,[ebp-128h]
00EEE031 call std::vector<ATL::CStringT<wchar_t,StrTraitMFC_DLL<wchar_t,ATL::ChTraitsCRT<wchar_t> > >,
   std::allocator<ATL::CStringT<wchar_t,StrTraitMFC_DLL<wchar_t,ATL::ChTraitsCRT<wchar_t> > > >::operator[] (0EE1258h)
00EEE036 mov esi,esp
00EEE038 push eax
00EEE039 lea ecx,[ebp-134h]
00EEE03F call dword ptr ds:[OF05420h]
```

Approved for public release; distribution is unlimited.
plus_equal += my_strings[2];

plus_plus = my_strings[0] + my_strings[1] + my_strings[2];
The += concatenate created 27 lines of machine code versus the 49 lines of machine code generated by the + operator. So just by the number of instructions created, it can be seen that the + operator will take longer. Looking deeper into the assembly, one can see that the + operator is returning a new buffer for each +, whereas the += operator is doing an actual concatenation on the current CString.

CONCLUSIONS

It’s very important for a developer to understand the complexities of writing code in one way versus another. This report shows that the more efficient way to concatenate CStrings is to use the += operator. Although the performance is not very different when only a few strings are involved, it would be better to just always use the more efficient version.
REFERENCES

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