Weifan Lin

Csc342 Section G

04/12/2015

Take-Home Exam

Title:

Clear Array

Objective:

The goal of this project is to compare the performance of clearing an array of integers using the index function and pointer function. In addition, the time required to clear an array using both methods are recorded into a graph.

Performance:

First, create a new console project on visual studio with the following files.

main.cpp

```
#include<Windows.h>
#include<iostream>
using namespace std;
void clear_array_pointer(int *, int);
void clear_array_index(int[], int);
int main()
```

```
{
	__int64 ctr1 = 0, ctr2 = 0, freq = 0;
	int i = 0;
	const int size = 10000;
	int array[size];
	for (i = 0; i < size; i++)
		array[i] = i+1;
	int *p = array;
	if (QueryPerformanceCounter((LARGE_INTEGER*)&ctr1) != 0)
	{
		clear_array_pointer(p, size);
		QueryPerformanceCounter((LARGE_INTEGER*)&ctr2);
		cout << "Start Value: " << ctr1 << endl;
		cout << "End Value: " << ctr2 << endl;
		QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
		cout << "QueryPerformanceCounter minimu resolution: 1/" << freq << "
	Seconds." << endl;
		cout << size<<" elements need time: " << ((ctr2 - ctr1)*1.0 / freq)
	*1000000 << "Microseconds." << endl;
	}
	else {
	DWORD dwError = GetLastError();
	cout << "Error value = " << dwError << endl;
	}
	return 0;
}
```

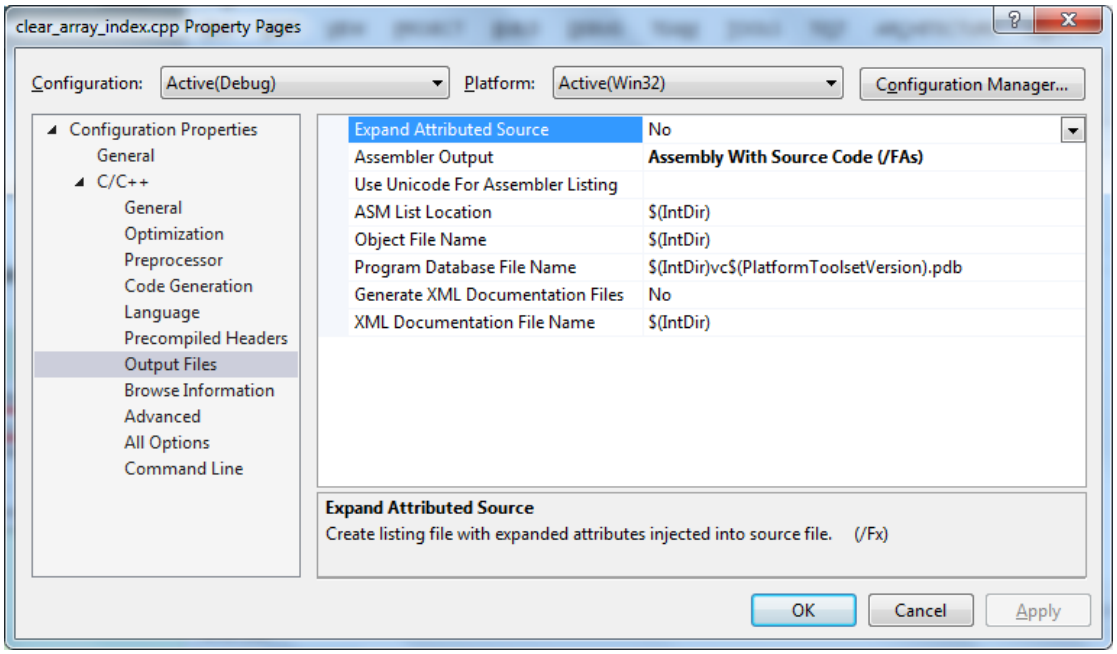
ClearArrayIndex.cpp

```
void clear_array_index(int ary[], int size)
{
	for (int k = 0; k < size; k++)
		ary[k] = 0;
}
```

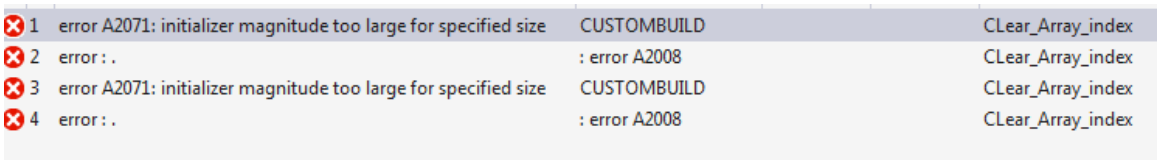
ClearArrayUsingPointer.cpp

```
void clear_array_pointer(int *ary, int size)
{
	int *p;
	for (p = &ary[0]; p < &ary[size]; p++)
		*p = 0;
}
```

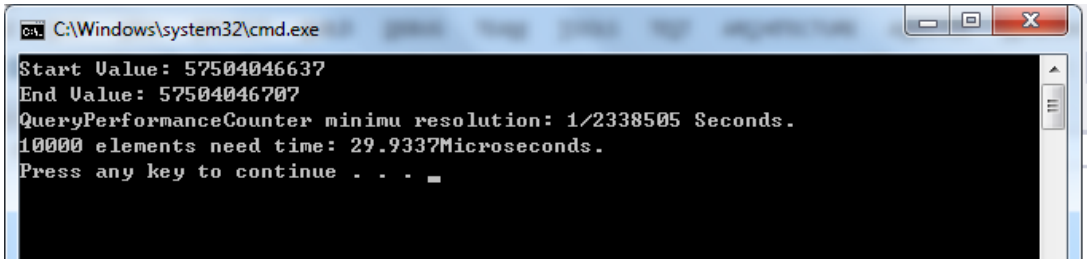
Then right click the files Clear_Array_index.cpp and ClearArrayPointer.cpp proerties and change the Assembly Output to Assembly With Source Code (/FAs). Compile the Clear_array_index.cpp. The file Clear_Array_index.asm is created in the project directory. Add this asm file to the source file, and remove the Clear_array_index.cpp file from the project.



Then, right click the asm file to Compile. Two error messages will show up.



Go to these lines in the asm file and comment them by adding ";" before the lines. Then Compile again. The following window pops up.



ASM file before optimization (ClearArrayIndex):

; Listing generated by Microsoft (R) Optimizing Compiler Version 18.00.21005.1

```
        TITLE C:\Users\Weifan\Desktop\343labs\ClearArray\ClearArray\clear.cpp
        .686P
        .XMM
        include listing.inc
        .model flat

INCLUDELIB MSVCRTD
INCLUDELIB OLDNAMES


PUBLIC ?clear_array_index@@YAXQAHH@Z            ; clear_array_index
EXTRN __RTC_InitBase:PROC
EXTRN __RTC_Shutdown:PROC
; COMDAT rtc$TMZ
rtc$TMZ SEGMENT
;__RTC_Shutdown.rtc$TMZ DD FLAT:__RTC_Shutdown
rtc$TMZ ENDS
; COMDAT rtc$IMZ
rtc$IMZ SEGMENT
;__RTC_InitBase.rtc$IMZ DD FLAT:__RTC_InitBase
rtc$IMZ ENDS
; Function compile flags: /Odtp /RTCsu /ZI
; File c:\users\Weifan\desktop\343labs\cleararray\cleararray\clear.cpp
; COMDAT ?clear_array_index@@YAXQAHH@Z
_TEXT SEGMENT
_k$1 = -8 ; size = 4
```

```
_ary$ = 8 ; size = 4
_size$ = 12 ; size = 4
?clear_array_index@@YAXQAHH@Z PROC ; clear_array_index, COMDAT
;2 :{
push ebp
mov ebp, esp
sub esp, 204 ; 000000ccH
push ebx
push esi
push edi
lea edi, DWORD PTR [ebp-204]
mov ecx, 51 ; 00000033H
mov eax, -858993460 ; ccccccccH
rep stosd
; 3
: for (int k = 0; k < size; k++)
mov DWORD PTR _k$1[ebp], 0
jmp SHORT $LN3@clear_arra
$LN2@clear_arra:
mov eax, DWORD PTR _k$1[ebp]
add eax, 1
mov DWORD PTR _k$1[ebp], eax
$LN3@clear_arra:
mov eax, DWORD PTR _k$1[ebp]
cmpeax, DWORD PTR _size$[ebp]
```

```
jge SHORT $LN4@clear_arra
; 4 : ary[k] = 0;
mov eax, DWORD PTR _k$1[ebp]
mov ecx, DWORD PTR _ary$[ebp]
mov DWORD PTR [ecx+eax*4], 0
jmp SHORT $LN2@clear_arra
$LN4@clear_arra:
;5 :}
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
ret 0
?clear_array_index@@YAXQAHH@Z ENDP ; clear_array_index
_TEXT ENDS
END
```

After Optimization:

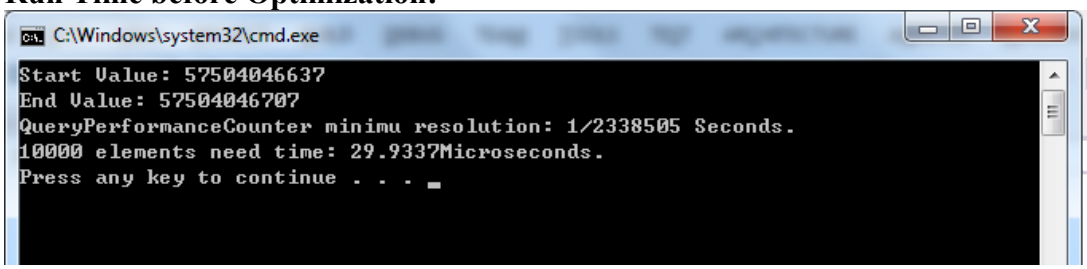
```
; Listing generated by Microsoft (R) Optimizing Compiler Version 18.00.21005.1
TITLE D:\342 and 343labs\ClearArray\ClearArrayProject\ClearArray\ClearArrayUsingPointer.cpp
.686P
.XMM
```

```
include listing.inc

.model flat

INCLUDELIB MSVCRTD

INCLUDELIB OLDNAMES

PUBLIC  ?clear_array_pointer@@YAXPAHH@Z ; clear_array_pointer

EXTRN __RTC_InitBase:PROC

EXTRN __RTC_Shutdown:PROC

; COMDAT rtc$TMZ

rtc$TMZ SEGMENT

;__RTC_Shutdown.rtc$TMZ DD FLAT:__RTC_Shutdown

rtc$TMZ ENDS

; COMDAT rtc$IMZ

rtc$IMZ SEGMENT

;__RTC_InitBase.rtc$IMZ DD FLAT:__RTC_InitBase

rtc$IMZ ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; COMDAT ?clear_array_pointer@@YAXPAHH@Z

_TEXT SEGMENT

_p$ = -8          ; size = 4 ;

_ary$ = 8         ; size = 4 ;

_size$ = 12       ; size = 4 ;

?clear_array_pointer@@YAXPAHH@Z PROC ;    clear_array_pointer,

COMDAT

; File d:\342 and
343labs\cleararray\cleararrayproject\cleararray\clearrarrayusingpointer.cpp ; Line 2

push ebp
```

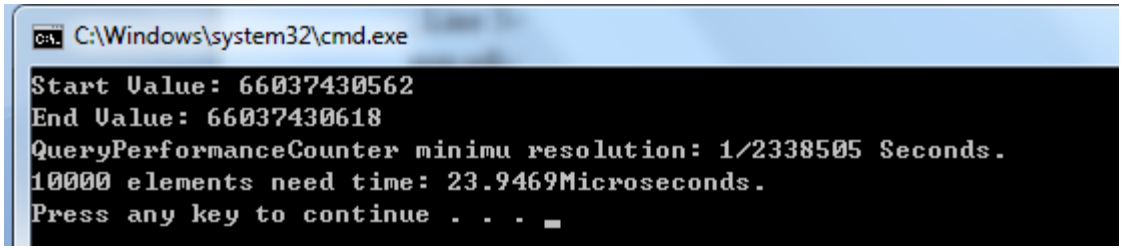
```
mov ebp,
esp sub esp, 204          ; 000000ccH
push ebx
push esi
push edi
lea edi,  DWORD PTR [ebp-204]
mov ecx, 51     ; 00000033H
mov eax, -858993460  ; ccccccccH
rep stosd
; Line 4
mov eax, DWORD PTR _ary$[ebp]
mov DWORD PTR _p$[ebp],
eax mov ebx, DWORD PTR _size$[ebp]
lea edx, DWORD PTR [ecx+eax*4]

jmp SHORT $LN3@clear_arra
$LN2@clear_arra:
add eax, 4
$LN3@clear_arra:
cmpeax, edx
jae SHORT $LN4@clear_arra
; Line 5
mov DWORD PTR [eax], 0
jmp SHORT $LN2@clear_arra
$LN4@clear_arra:
```

```
; Line 6
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
ret 0
?clear_array_pointer@@YAXPAHH@Z ENDP      ; clear_array_pointer
_TEXT ENDS
END
```

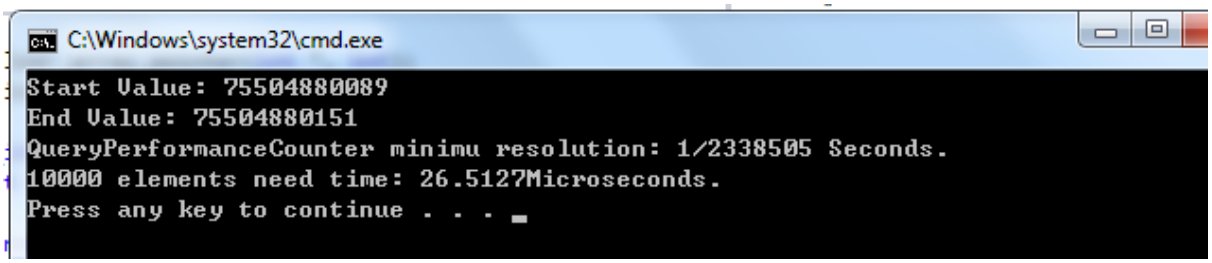
Run Time before Optimization:



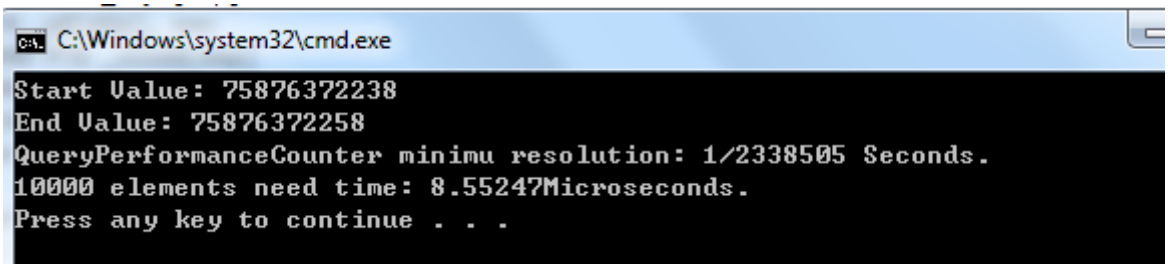
Run Time after Optimization:



Run Time before Optimization (pointer):

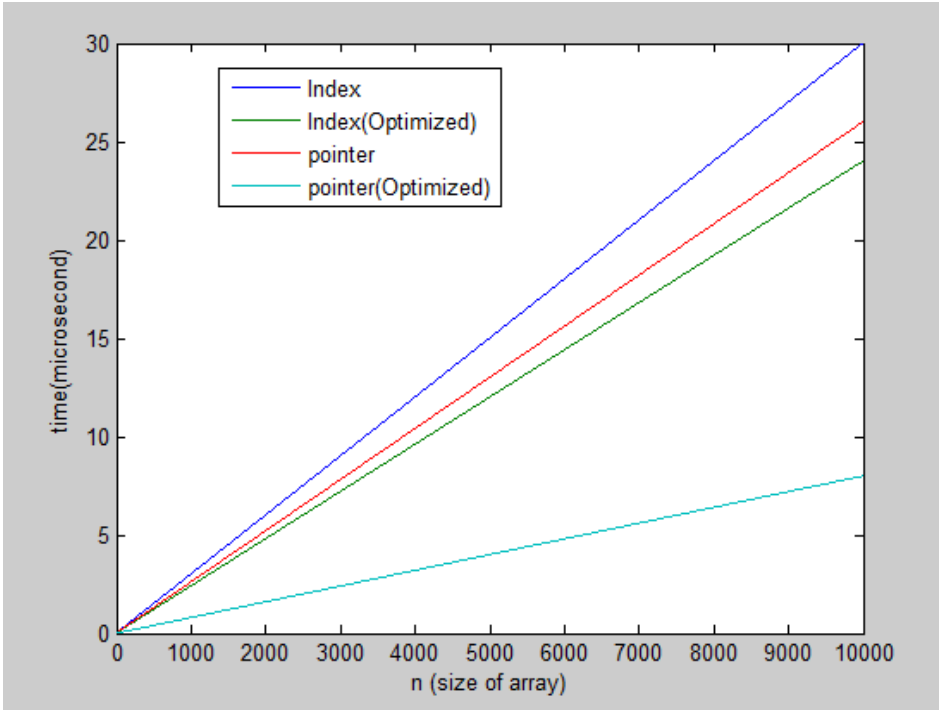


Run Time after Optimization (pointer):



Results:

A comparison of the timing for the four methods is shown below.



Conclusion:

The Optimized pointer method takes the shortest time to clear the array of size 10000. The Optimized IndexArray code is slightly faster than the pointer code. The IndexArray code is the slowest. Optimized pointer shows a significant run time improvement.