Factorial with Assembly Generated code

Bello Melido

11/08/2023

CSC 210

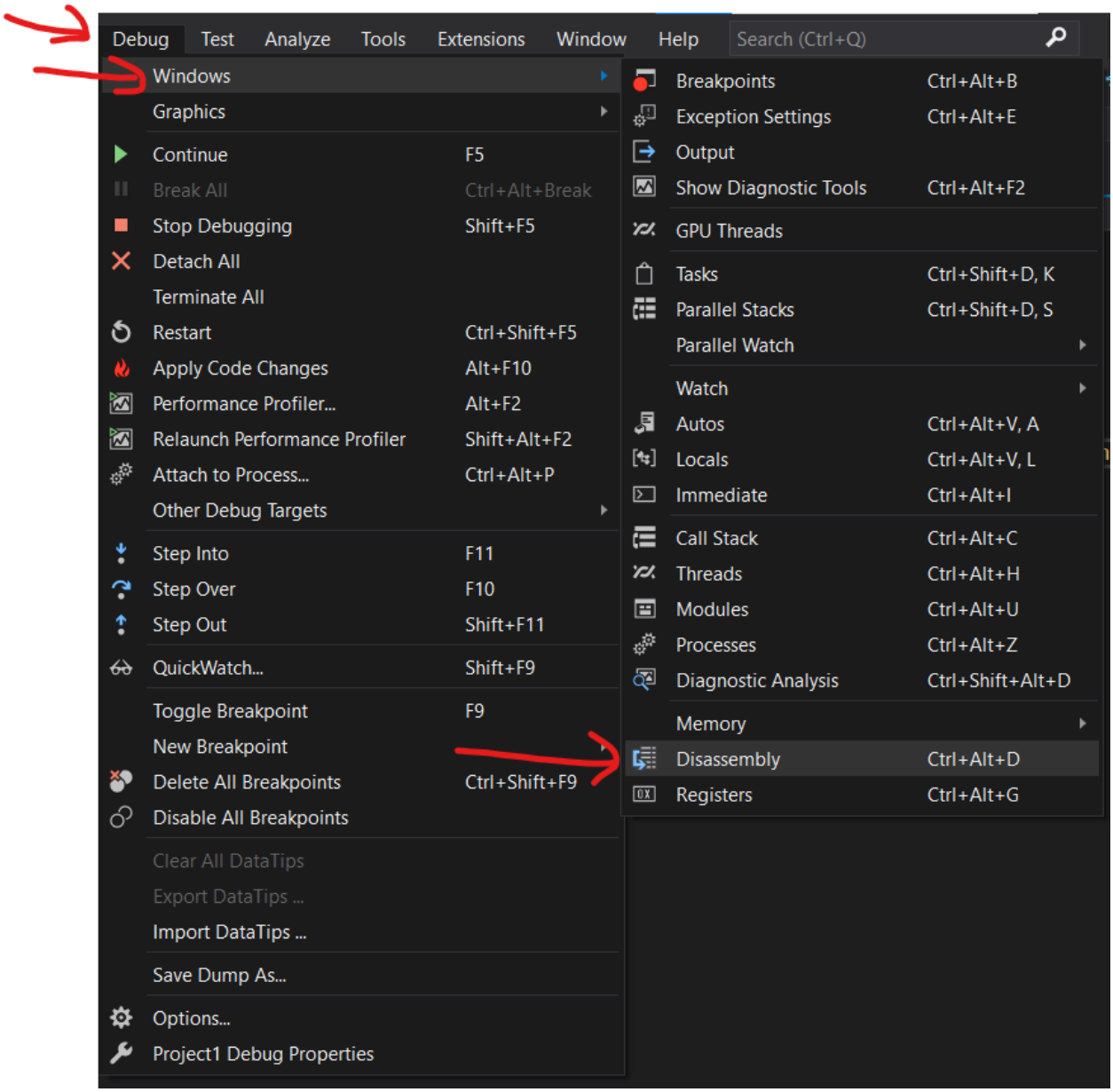
C++ Code for factorial with QueryPerformanceFrequency (visual studio 2019)

```cpp
#include <windows.h>
#include <iostream>

using namespace std;

int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    // Get the frequency.
    LARGE_INTEGER frequency;
    if (!QueryPerformanceFrequency(&frequency)) {
        cerr << "QueryPerformanceFrequency not supported" << endl;
        return 1;
    }

    // Prepare variables for the loop.
    const int iterations = 10000;
    double totalExecutionTime = 0;

    for (int i = 0; i < iterations; ++i) {
        // Get the start time
        LARGE_INTEGER startTime;
        QueryPerformanceCounter(&startTime);

        // Call the factorial function.
        int result = factorial(10);  // Using 10 as an example

        // Get the end time.
        LARGE_INTEGER endTime;
        QueryPerformanceCounter(&endTime);

        // Calculate the execution time for this iteration.
        totalExecutionTime += (endTime.QuadPart - startTime.QuadPart) / (double)frequency.QuadPart;
    }

    // Calculate the average execution time.
    double averageExecutionTime = totalExecutionTime / iterations;

    // Print the average execution time.
    printf("The average execution time of the factorial function over %d iterations is %f milliseconds.\n", iterations, averageExecutionTime * 1000);

    // Wait for the user to press a key before exiting.
    system("pause");
    return 0;
}
```

Result

```
C:\Users\enrri\source\repos\Project1\Debug\Project1.exe
The average execution time of the factorial function over 10000 iterations is 0.000184 milliseconds.
Press any key to continue . . .
```

To get the assembly



Assembly code

```
--- C:\Users\enrri\source\repos\Project1\Project1\Source.cpp -------------------

int main() {
00242710  push        ebp
00242711  mov         ebp,esp
00242713  sub         esp,140h
00242719  push        ebx
0024271A  push        esi
0024271B  push        edi
0024271C  lea         edi,[ebp-80h]
0024271F  mov         ecx,20h
00242724  mov         eax,0CCCCCCCCh
00242729  rep stos    dword ptr es:[edi]
0024272B  mov         eax,dword ptr [__security_cookie (024C004h)]
00242730  xor         eax,ebp
00242732  mov         dword ptr [ebp-4],eax
00242735  mov         ecx,offset _80D703B7_Source@cpp (024F035h)
0024273A  call        @__CheckForDebuggerJustMyCode@4 (02413BBh)
```

```
    // Get the frequency.
    LARGE_INTEGER frequency;
    if (!QueryPerformanceFrequency(&frequency)) {
0024273F  mov         esi,esp
00242741  lea         eax,[frequency]
00242744  push        eax
00242745  call        dword ptr [__imp__QueryPerformanceFrequency@4 (024D004h)]
0024274B  cmp         esi,esp
0024274D  call        __RTC_CheckEsp (02412B2h)
00242752  test        eax,eax
00242754  jne         __$EncStackInitStart+6Dh (0242789h)
        cerr << "QueryPerformanceFrequency not supported" << endl;
00242756  mov         esi,esp
00242758  push        offset std::endl<char,std::char_traits<char> > (024103Ch)
0024275D  push        offset string "QueryPerformanceFrequency not s@"... (0249B30h)
00242762  mov         eax,dword ptr [__imp_std::cerr (024D09Ch)]
00242767  push        eax
00242768  call        std::operator<<<std::char_traits<char> > (02411C2h)
0024276D  add         esp,8
00242770  mov         ecx,eax
00242772  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (024D0A8h)]
00242778  cmp         esi,esp
0024277A  call        __RTC_CheckEsp (02412B2h)
        return 1;
0024277F  mov         eax,1
00242784  jmp         __$EncStackInitStart+15Bh (0242877h)
    }
```

```
    // Prepare variables for the loop.
    const int iterations = 10000;
00242789  mov         dword ptr [iterations],2710h
    double totalExecutionTime = 0;
00242790  xorps       xmm0,xmm0
00242793  movsd       mmword ptr [totalExecutionTime],xmm0

    for (int i = 0; i < iterations; ++i) {
00242798  mov         dword ptr [ebp-38h],0
0024279F  jmp         __$EncStackInitStart+8Eh (02427AAh)
002427A1  mov         eax,dword ptr [ebp-38h]
002427A4  add         eax,1
002427A7  mov         dword ptr [ebp-38h],eax
002427AA  cmp         dword ptr [ebp-38h],2710h
002427B1  jge         __$EncStackInitStart+109h (0242825h)
        // Get the start time
        LARGE_INTEGER startTime;
        QueryPerformanceCounter(&startTime);
002427B3  mov         esi,esp
002427B5  lea         eax,[ebp-48h]
002427B8  push        eax
002427B9  call        dword ptr [__imp__QueryPerformanceCounter@4 (024D000h)]
002427BF  cmp         esi,esp
002427C1  call        __RTC_CheckEsp (02412B2h)

        // Call the factorial function.
        int result = factorial(10);  // Using 10 as an example
002427C6  push        0Ah
002427C8  call        factorial (02412FDh)
002427CD  add         esp,4
002427D0  mov         dword ptr [ebp-54h],eax
```

```
        // Get the end time.
        LARGE_INTEGER endTime;
        QueryPerformanceCounter(&endTime);
002427D3  mov         esi,esp
002427D5  lea         eax,[ebp-64h]
002427D8  push        eax
002427D9  call        dword ptr [__imp__QueryPerformanceCounter@4 (024D000h)]
002427DF  cmp         esi,esp
002427E1  call        __RTC_CheckEsp (02412B2h)

        // Calculate the execution time for this iteration.
        totalExecutionTime += (endTime.QuadPart - startTime.QuadPart) / (double)frequency.QuadPart;
002427E6  mov         ecx,dword ptr [ebp-64h]
002427E9  sub         ecx,dword ptr [ebp-48h]
002427EC  mov         edx,dword ptr [ebp-60h]
002427EF  sbb         edx,dword ptr [ebp-44h]
002427F2  call        __ltod3 (0241479h)
002427F7  mov         edx,dword ptr [ebp-0Ch]
002427FA  mov         ecx,dword ptr [frequency]
002427FD  movsd       mmword ptr [ebp-140h],xmm0
00242805  call        __ltod3 (0241479h)
0024280A  movsd       xmm1,mmword ptr [ebp-140h]
00242812  divsd       xmm1,xmm0
00242816  addsd       xmm1,mmword ptr [totalExecutionTime]
0024281B  movsd       mmword ptr [totalExecutionTime],xmm1
    }
00242820  jmp         __$EncStackInitStart+85h (02427A1h)
```

```
    // Calculate the average execution time.
    double averageExecutionTime = totalExecutionTime / iterations;
00242825  movsd       xmm0,mmword ptr [totalExecutionTime]
0024282A  divsd       xmm0,mmword ptr [string "The execution time of the facto@"... (0249B60h)]
00242832  movsd       mmword ptr [averageExecutionTime],xmm0

    // Print the average execution time.
    printf("The average execution time of the factorial function over %d iterations is %f milliseconds.\n", iterations, averageExecutionTime * 1000);
00242837  movsd       xmm0,mmword ptr [averageExecutionTime]
0024283C  mulsd       xmm0,mmword ptr [__real@3ff0000000000000 (0249B58h)]
00242844  sub         esp,8
00242847  movsd       mmword ptr [esp],xmm0
0024284C  push        2710h
00242851  push        offset string "The average execution time of t@"... (024A678h)
00242856  call        _printf (02410FAh)
0024285B  add         esp,10h

    // Wait for the user to press a key before exiting.
    system("pause");
0024285E  mov         esi,esp
00242860  push        offset string "pause" (0249B68h)
00242865  call        dword ptr [__imp__system (024D208h)]
0024286B  add         esp,4
0024286E  cmp         esi,esp
00242870  call        __RTC_CheckEsp (02412B2h)
    return 0;
00242875  xor         eax,eax
}
```

Improved code C++

```cpp
#include <windows.h>
#include <iostream>

using namespace std;

// Iterative factorial function.
unsigned long long factorial(int n) {
    unsigned long long result = 1;
    for (int i = 2; i <= n; ++i) { // Start from 2 as multiplying by 1 is unnecessary
        result *= i;
    }
    return result;
}

int main() {
    // Get the frequency.
    LARGE_INTEGER frequency;
    if (!QueryPerformanceFrequency(&frequency)) {
        cerr << "QueryPerformanceFrequency not supported" << endl;
        return 1;
    }

    // Pre-calculate factorial of 10.
    unsigned long long precalculatedFactorial = factorial(1000);

    // Prepare variables for the loop.
    const int iterations = 10000;
    double totalExecutionTimeInSeconds = 0.0; // Use double to avoid integer division issues

    for (int i = 0; i < iterations; ++i) {
        // Get the start time
        LARGE_INTEGER startTime;
        QueryPerformanceCounter(&startTime);

        // Simulate the calculation of factorial (we already have the result).
        volatile unsigned long long result = precalculatedFactorial; // use volatile to prevent optimizations

        // Get the end time.
        LARGE_INTEGER endTime;
        QueryPerformanceCounter(&endTime);

        // Calculate the execution time for this iteration in seconds.
        totalExecutionTimeInSeconds += static_cast<double>(endTime.QuadPart - startTime.QuadPart) / frequency.QuadPart;
    }

    // Calculate the average execution time in seconds.
    double averageExecutionTimeInSeconds = totalExecutionTimeInSeconds / iterations;

    // Convert the average time to milliseconds.
    double averageExecutionTimeInMilliseconds = averageExecutionTimeInSeconds * 1000.0;

    // Print the average execution time.
    cout << "The average execution time of the factorial function over " << iterations
        << " iterations is " << averageExecutionTimeInMilliseconds << " milliseconds." << endl;

    return 0;
}
```

Result

```
C:\Users\enrri\source\repos\Project1\Debug\Project1.exe
The average execution time of the factorial function over 10000 iterations is 3.965e-05 milliseconds.
```

Generated assembly code

```
int main() {
00022660  push        ebp
00022661  mov         ebp,esp
00022663  sub         esp,164h
00022669  push        ebx
0002266A  push        esi
0002266B  push        edi
0002266C  lea         edi,[ebp-0E4h]
00022672  mov         ecx,39h
00022677  mov         eax,0CCCCCCCCh
0002267C  rep stos    dword ptr es:[edi]
0002267E  mov         eax,dword ptr [__security_cookie (02C004h)]
00022683  xor         eax,ebp
00022685  mov         dword ptr [ebp-4],eax
00022688  mov         ecx,offset _80D703B7_Source@cpp (02F035h)
0002268D  call        @__CheckForDebuggerJustMyCode@4 (0213BBh)
    // Get the frequency.
    LARGE_INTEGER frequency;
    if (!QueryPerformanceFrequency(&frequency)) {
00022692  mov         esi,esp
00022694  lea         eax,[frequency]
00022697  push        eax
00022698  call        dword ptr [__imp__QueryPerformanceFrequency@4 (02D004h)]
0002269E  cmp         esi,esp
000226A0  call        __RTC_CheckEsp (0212B7h)
000226A5  test        eax,eax
000226A7  jne         __$EncStackInitStart+70h (0226DCh)
        cerr << "QueryPerformanceFrequency not supported" << endl;
000226A9  mov         esi,esp
000226AB  push        offset std::endl<char,std::char_traits<char> > (02103Ch)
000226B0  push        offset string "QueryPerformanceFrequency not s@"... (029B30h)
000226B5  mov         eax,dword ptr [__imp_std::cerr (02D09Ch)]
```

```
000226BA  push        eax
000226BB  call        std::operator<<<std::char_traits<char> > (0211CCh)
000226C0  add         esp,8
000226C3  mov         ecx,eax
000226C5  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (02D0B4h)]
000226CB  cmp         esi,esp
000226CD  call        __RTC_CheckEsp (0212B7h)
        return 1;
000226D2  mov         eax,1
000226D7  jmp         __$EncStackInitStart+1DBh (022847h)
    }

    // Pre-calculate factorial of 10.
    unsigned long long precalculatedFactorial = factorial(1000);
000226DC  push        3E8h
000226E1  call        factorial (021104h)
000226E6  add         esp,4
000226E9  mov         dword ptr [precalculatedFactorial],eax
000226EC  mov         dword ptr [ebp-1Ch],edx

    // Prepare variables for the loop.
    const int iterations = 10000;
000226EF  mov         dword ptr [iterations],2710h
    double totalExecutionTimeInSeconds = 0.0; // Use double to avoid integer division issues
000226F6  xorps       xmm0,xmm0
000226F9  movsd       mmword ptr [totalExecutionTimeInSeconds],xmm0

    for (int i = 0; i < iterations; ++i) {
00022701  mov         dword ptr [ebp-48h],0
00022708  jmp         __$EncStackInitStart+0A7h (022713h)
0002270A  mov         eax,dword ptr [ebp-48h]
0002270D  add         eax,1
00022710  mov         dword ptr [ebp-48h],eax
```

```
00022713  cmp         dword ptr [ebp-48h],2710h
0002271A  jge         __$EncStackInitStart+12Dh (022799h)
        // Get the start time
        LARGE_INTEGER startTime;
        QueryPerformanceCounter(&startTime);
0002271C  mov         esi,esp
0002271E  lea         eax,[ebp-58h]
00022721  push        eax
00022722  call        dword ptr [__imp__QueryPerformanceCounter@4 (02D000h)]
00022728  cmp         esi,esp
0002272A  call        __RTC_CheckEsp (0212B7h)

        // Simulate the calculation of factorial (we already have the result).
        volatile unsigned long long result = precalculatedFactorial; // use volatile to prevent optimizations
0002272F  mov         eax,dword ptr [precalculatedFactorial]
00022732  mov         ecx,dword ptr [ebp-1Ch]
00022735  mov         dword ptr [ebp-0B8h],eax
0002273B  mov         dword ptr [ebp-0B4h],ecx

        // Get the end time.
        LARGE_INTEGER endTime;
        QueryPerformanceCounter(&endTime);
00022741  mov         esi,esp
00022743  lea         eax,[ebp-78h]
00022746  push        eax
00022747  call        dword ptr [__imp__QueryPerformanceCounter@4 (02D000h)]
0002274D  cmp         esi,esp
0002274F  call        __RTC_CheckEsp (0212B7h)

        // Calculate the execution time for this iteration in seconds.
        totalExecutionTimeInSeconds += static_cast<double>(endTime.QuadPart - startTime.QuadPart) / frequency.QuadPart;
00022754  mov         ecx,dword ptr [ebp-78h]
00022757  sub         ecx,dword ptr [ebp-58h]
```

```
0002275A  mov         edx,dword ptr [ebp-74h]
0002275D  sbb         edx,dword ptr [ebp-54h]
00022760  call        __ltod3 (021474h)
00022765  mov         edx,dword ptr [ebp-0Ch]
00022768  mov         ecx,dword ptr [frequency]
0002276B  movsd       mmword ptr [ebp-164h],xmm0
00022773  call        __ltod3 (021474h)
00022778  movsd       xmm1,mmword ptr [ebp-164h]
00022780  divsd       xmm1,xmm0
00022784  addsd       xmm1,mmword ptr [totalExecutionTimeInSeconds]
0002278C  movsd       mmword ptr [totalExecutionTimeInSeconds],xmm1
    }
00022794  jmp         __$EncStackInitStart+9Eh (02270Ah)

    // Calculate the average execution time in seconds.
    double averageExecutionTimeInSeconds = totalExecutionTimeInSeconds / iterations;
00022799  movsd       xmm0,mmword ptr [totalExecutionTimeInSeconds]
000227A1  divsd       xmm0,mmword ptr [__real@40c3880000000000 (029BE0h)]
000227A9  movsd       mmword ptr [averageExecutionTimeInSeconds],xmm0

    // Convert the average time to milliseconds.
    double averageExecutionTimeInMilliseconds = averageExecutionTimeInSeconds * 1000.0;
000227B1  movsd       xmm0,mmword ptr [averageExecutionTimeInSeconds]
000227B9  mulsd       xmm0,mmword ptr [__real@408f400000000000 (029BD0h)]
000227C1  movsd       mmword ptr [averageExecutionTimeInMilliseconds],xmm0

    // Print the average execution time.
    cout << "The average execution time of the factorial function over " << iterations
000227C9  mov         esi,esp
000227CB  push        offset std::endl<char,std::char_traits<char> > (02103Ch)
000227D0  push        offset string " milliseconds." (029B60h)
000227D5  mov         edi,esp
000227D7  sub         esp,8
```

```
000227DA  movsd       xmm0,mmword ptr [averageExecutionTimeInMilliseconds]
000227E2  movsd       mmword ptr [esp],xmm0
000227E7  push        offset string " iterations is " (029B74h)
000227EC  mov         ebx,esp
000227EE  push        2710h
000227F3  push        offset string "The average execution time of t@"... (029B88h)
000227F8  mov         eax,dword ptr [__imp_std::cout (02D0A0h)]
000227FD  push        eax
000227FE  call        std::operator<<<std::char_traits<char> > (0211CCh)
00022803  add         esp,8
00022806  mov         ecx,eax
00022808  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (02D0B0h)]
0002280E  cmp         ebx,esp
00022810  call        __RTC_CheckEsp (0212B7h)
00022815  push        eax
00022816  call        std::operator<<<std::char_traits<char> > (0211CCh)
0002281B  add         esp,8
0002281E  mov         ecx,eax
00022820  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (02D0ACh)]
00022826  cmp         edi,esp
00022828  call        __RTC_CheckEsp (0212B7h)
0002282D  push        eax
0002282E  call        std::operator<<<std::char_traits<char> > (0211CCh)
00022833  add         esp,8
00022836  mov         ecx,eax
00022838  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (02D0B4h)]
0002283E  cmp         esi,esp
00022840  call        __RTC_CheckEsp (0212B7h)
        << " iterations is " << averageExecutionTimeInMilliseconds << " milliseconds." << endl;

    return 0;
00022845  xor         eax,eax
}
```