# **CS 410 C++ to Assembly With Loops Activity Template**

**Step 1:** Explain the functionality of the C++ code.

## C++ Code Functionality

|  |  |
| --- | --- |
| **C++ Line of Code** | **Explanation of Functionality** |
| int main() {  int num, i;  int product = 1; | These instructions create memory for local variables and build up the stack frame for the main() function. This occurs at the start of the C++ code's main() function:  32 bytes are reserved on the stack by the sub rsp,0x20 for the local variables (num, i, and product). |
| //Compiler Code | A stack canary, which is used to identify stack buffer overflows, is set up by following these instructions. Although it doesn't correspond to a particular line in the C++ code, this is a standard security feature in contemporary compiled programs. |
| int product = 1; | This is equivalent to setting the C++ product variable's initial value to 1: |
| cout << "Enter a number:\n"; | This sequence prints "Enter a number:" by using cout. For std::ostream, the function 0x5555555550d0 calls the operator\\. It is equivalent to: |
| cin >> num; | This is a call to cin >> num; the num variable contains the user input. In particular, the implementation of cin >> num involves calling the input stream operator function, 0x5555555550b0. |
| for (i = num; i > 0; i--) | The value of num is moved by these instructions to a temporary place ([rbp-0x10]). This is equivalent to initializing the C++ code's loop variable, i = num:  determines if i > 0. The loop terminates if false; if true, it continues. This is consistent with the for loop's condition: |
| product = product \* i; | The multiplication operation product = product \* i; is carried out by this sequence, and it immediately translates to the C++ code: |
| i--; | According to the post-decrement in the for loop, this instruction decreases i by 1: |
| } // end of the loop | This loops back to the beginning if i is greater than zero and compares i to 0 once again. |
| cout << "The factorial for " << num << " is: " << product << endl; | These commands line up with the cout calls made to print the outcome: |
| // compiler code | These instructions make sure there was no stack buffer overflow by checking the stack canary once again. Since this security feature is introduced automatically by the compiler, it does not transfer directly to C++ code. |
| return 0; | After clearing the stack, these instructions exit the main() function. This is equivalent to the C++ code's return 0; statement: |

**Step 2:** Convert the C++ file into assembly code.

**Step 3:** Align each line of C++ code with the corresponding blocks of assembly code.

## C++ to Assembly Alignment

|  |  |
| --- | --- |
| **C++ Line of Code** | **Blocks of Assembly Code** |
| int main() {  int num, i;  int product = 1; | 0x0000555555555209 <+0>: endbr64  0x000055555555520d <+4>: push rbp  0x000055555555520e <+5>: mov rbp,rsp  0x0000555555555211 <+8>: sub rsp,0x20 |
| //Compiler Code | 0x0000555555555215 <+12>: mov rax,QWORD PTR fs:0x28 0x000055555555521e <+21>: mov QWORD PTR [rbp-0x8],rax 0x0000555555555222 <+25>: xor eax,eax |
| int product = 1; | 0x0000555555555224 <+27>: mov DWORD PTR [rbp-0xc],0x1 |
| cout << "Enter a number:\n"; | 0x0000555555555232 <+41>: mov rsi,rax  0x0000555555555235 <+44>: lea rax,[rip+0x2e04] # 0x555555558040 <\_ZSt4cout@GLIBCXX\_3.4>  0x000055555555523c <+51>: mov rdi,rax  0x000055555555523f <+54>: call 0x5555555550d0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> |
| cin >> num; | 0x0000555555555244 <+59>: lea rax,[rbp-0x14]  0x0000555555555248 <+63>: mov rsi,rax  0x000055555555524b <+66>: lea rax,[rip+0x2f0e] # 0x555555558160 <\_ZSt3cin@GLIBCXX\_3.4>  0x0000555555555252 <+73>: mov rdi,rax  0x0000555555555255 <+76>: call 0x5555555550b0 <\_ZNSirsERi@plt> |
| for (i = num; i > 0; i--) | 0x000055555555525a <+81>: mov eax,DWORD PTR [rbp-0x14]  0x000055555555525d <+84>: mov DWORD PTR [rbp-0x10],eax  0x0000555555555270 <+103>: cmp DWORD PTR [rbp-0x10],0x0  0x0000555555555274 <+107>: jg 0x555555555262 <main()+89> |
| product = product \* i; | 0x0000555555555262 <+89>: mov eax,DWORD PTR [rbp-0xc]  0x0000555555555265 <+92>: imul eax,DWORD PTR [rbp-0x10]  0x0000555555555269 <+96>: mov DWORD PTR [rbp-0xc],eax |
| i--; | 0x000055555555526c <+99>: sub DWORD PTR [rbp-0x10],0x1 |
| } // end of the loop | 0x0000555555555270 <+103>: cmp DWORD PTR [rbp-0x10],0x0  0x0000555555555274 <+107>: jg 0x555555555262 <main()+89> |
| cout << "The factorial for " << num << " is: " << product << endl; | 0x000055555555528a <+129>: call 0x5555555550d0  0x000055555555529a <+145>: call 0x555555555110 |
| // compiler code | 0x00005555555552db <+210>: mov rdx,QWORD PTR [rbp-0x8]  0x00005555555552df <+214>: sub rdx,QWORD PTR fs:0x28  0x00005555555552e8 <+223>: je 0x5555555552ef <main()+230>  0x00005555555552ea <+225>: call 0x5555555550f0 <\_\_stack\_chk\_fail@plt> |
| return 0; | 0x00005555555552ef <+230>: leave  0x00005555555552f0 <+231>: ret |

**Step 4:** Explain how the blocks of assembly code perform the same tasks as the C++ code.

## Assembly Functionality

|  |  |
| --- | --- |
| **Blocks of Assembly Code** | **Explanation of Functionality** |
| 0x0000555555555209 <+0>: endbr64  0x000055555555520d <+4>: push rbp  0x000055555555520e <+5>: mov rbp,rsp  0x0000555555555211 <+8>: sub rsp,0x20 | These instructions create memory for local variables and build up the stack frame for the main() function. This occurs at the start of the C++ code's main() function:  32 bytes are reserved on the stack by the sub rsp,0x20 for the local variables (num, i, and product). |
| 0x0000555555555215 <+12>: mov rax,QWORD PTR fs:0x28  0x000055555555521e <+21>: mov QWORD PTR [rbp-0x8],rax  0x0000555555555222 <+25>: xor eax,eax | A stack canary, which is used to identify stack buffer overflows, is set up by following these instructions. Although it doesn't correspond to a particular line in the C++ code, this is a standard security feature in contemporary compiled programs. |
| 0x0000555555555224 <+27>: mov DWORD PTR [rbp-0xc],0x1 | This is equivalent to setting the C++ product variable's initial value to 1: |
| 0x0000555555555232 <+41>: mov rsi,rax  0x0000555555555235 <+44>: lea rax,[rip+0x2e04] # 0x555555558040 <\_ZSt4cout@GLIBCXX\_3.4>  0x000055555555523c <+51>: mov rdi,rax  0x000055555555523f <+54>: call 0x5555555550d0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | This sequence prints "Enter a number:" by using cout. For std::ostream, the function 0x5555555550d0 calls the operator\\. It is equivalent to: |
| 0x0000555555555244 <+59>: lea rax,[rbp-0x14]  0x0000555555555248 <+63>: mov rsi,rax  0x000055555555524b <+66>: lea rax,[rip+0x2f0e] # 0x555555558160 <\_ZSt3cin@GLIBCXX\_3.4>  0x0000555555555252 <+73>: mov rdi,rax  0x0000555555555255 <+76>: call 0x5555555550b0 <\_ZNSirsERi@plt> | This is a call to cin >> num; the num variable contains the user input. In particular, the implementation of cin >> num involves calling the input stream operator function, 0x5555555550b0. |
| 0x000055555555525a <+81>: mov eax,DWORD PTR [rbp-0x14]  0x000055555555525d <+84>: mov DWORD PTR [rbp-0x10],eax  0x0000555555555270 <+103>: cmp DWORD PTR [rbp-0x10],0x0  0x0000555555555274 <+107>: jg 0x555555555262 <main()+89> | The value of num is moved by these instructions to a temporary place ([rbp-0x10]). This is equivalent to initializing the C++ code's loop variable, i = num:  determines if i > 0. The loop terminates if false; if true, it continues. This is consistent with the for loop's condition: |
| 0x0000555555555262 <+89>: mov eax,DWORD PTR [rbp-0xc]  0x0000555555555265 <+92>: imul eax,DWORD PTR [rbp-0x10]  0x0000555555555269 <+96>: mov DWORD PTR [rbp-0xc],eax | This sequence performs the multiplication product = product \* i;, which corresponds directly to the C++ code: |
| 0x000055555555526c <+99>: sub DWORD PTR [rbp-0x10],0x1 | According to the post-decrement in the for loop, this instruction decreases i by 1: |
| 0x0000555555555270 <+103>: cmp DWORD PTR [rbp-0x10],0x0  0x0000555555555274 <+107>: jg 0x555555555262 <main()+89> | This loops back to the beginning if i is greater than zero and compares i to 0 once again. |
| 0x000055555555528a <+129>: call 0x5555555550d0  0x000055555555529a <+145>: call 0x555555555110 | These commands line up with the cout calls made to print the outcome: |
| 0x00005555555552db <+210>: mov rdx,QWORD PTR [rbp-0x8]  0x00005555555552df <+214>: sub rdx,QWORD PTR fs:0x28  0x00005555555552e8 <+223>: je 0x5555555552ef <main()+230>  0x00005555555552ea <+225>: call 0x5555555550f0 <\_\_stack\_chk\_fail@plt> | These instructions make sure there was no stack buffer overflow by checking the stack canary once again. Since this security feature is introduced automatically by the compiler, it does not transfer directly to C++ code. |
| 0x00005555555552ef <+230>: leave  0x00005555555552f0 <+231>: ret | After clearing the stack, these instructions exit the main() function. This is equivalent to the C++ code's return 0; statement: |