# **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** |
| --- | --- |
| #include<iostream> | opens up the i/o stream so it can be used for cout and cin functionality |
| using namespace std; | declares the namespace to be the standard namespace |
| int main() | initializes the main function of the program |
| { |  |
| int num, i; | declares an integer variable called num and an integer variable called i |
| int product =1; | declares an integer variable called product with the value of 1 |
| cout<<"Enter a number:\n"<< endl; | prints “Enter a number: “ to the console and then moves to the next line and then ends that line |
| cin>>num; | takes input from the user and stores that variable in num |
| for(i=num;i>0; i--) | sets up a for loop that will run num number of times by setting i equal to num, checking if i is greater than 0, and then decreasing i by one |
| product = product \* i; | during each iteration of the for loop the product is set to the product multiplied by the value of i |
| cout<<"The factorial for " << num << "is: \n"<< product; | after the for loop has finished the system prints “The factorial for “value stored in num” is: “value stored in product” |
| return 1; | returns 1 to end the main function |
| } | closes the main function |

**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** |
| --- | --- |
| #include<iostream> |  |
| using namespace std; |  |
| int main() | main:  .LFB1493:  .cfi\_startproc  pushq %rbp |
| { |  |
| int num, i; | movq %fs:40, %rax  movq %rax, -8(%rbp)  xorl %eax, %eax |
| int product =1; | movl $1, -12(%rbp) |
| cout<<"Enter a number:\n"<< endl; | leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT |
| cin>>num; | movq %rax, %rdx  movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax  movq %rax, %rsi  movq %rdx, %rdi  call \_ZNSolsEPFRSoS\_E@PLT |
| for(i=num;i>0; i--) | leaq -20(%rbp), %rax |
| product = product \* i; | movq %rax, %rsi |
| cout<<"The factorial for " << num << "is: \n"<< product; | leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT |
| return 1; |  |
| } |  |

**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** |
| --- | --- |
|  |  |
|  |  |
| main:  .LFB1493:  .cfi\_startproc  pushq %rbp | initializes the main function of the program |
|  |  |
| movq %fs:40, %rax  movq %rax, -8(%rbp)  xorl %eax, %eax | prepares rbp for storing variables and sets them to be 8 bytes above the stack. I think because there is 4 for each variable since they are both integers (num and i) |
| movl $1, -12(%rbp) | sets rbp 12 bytes above the stack to 1 (this is the variable product) |
| leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT | places the variable stored in LC0 into rsi (this is the string “Enter a number:\n”) then places the value stored in \_ZSt4cout into rdi, then calls basic output stream. |
| movq %rax, %rdx  movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax  movq %rax, %rsi  movq %rdx, %rdi  call \_ZNSolsEPFRSoS\_E@PLT | moves the value in rax into rdx  moves the old output stream value into rax  moves rax into rsi  moves rdx into rdi  calls a function (in this case it is the cin function |
| leaq -20(%rbp), %rax | points the value stored 20 bytes above rbp into rax |
| movq %rax, %rsi | moves the value stored in rax into rsi |
| leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT | points the value stored in rip into rdo and calls a function (this time the final cout function of the program) |
|  |  |
|  |  |