# **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> | Includes the iostream header file for inputs and outputs |
| using namespace std; | Used to set the namespace to std. can use cout instead of std::cout |
| int main() | Defines the main function, no arguments and returns an int |
| { | Beginning of function body |
| int num, i; | Declares integer variables num and i |
| int product =1; | Declares integer variable product and assigns 1 |
| cout<<"Enter a number:\n"<< endl; | Output to console |
| cin>>num; | User input of num |
| for(i=num;i>0; i--) | For loop that loops num times starting from num down to 1 |
| product = product \* i; | Multiplies product by i and saves it in product |
| cout<<"The factorial for " << num << "is: \n"<< product; | Output to console using strings, num, and product |
| return 1; | Returns a 1 essentially ending the function |
| } | End of function body |

**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> | Gets processed by the preprocessor |
| using namespace std; | Used in compilation but doesn’t have assembly code |
| int main() { | .LC0:    .string "Enter a number:\n"  .LC1:    .string "The factorial for "  .LC2:    .string "is: \n"    .text    .globl  main    .type main, @function  main:  .LFB1493:    .cfi\_startproc    pushq %rbp    .cfi\_def\_cfa\_offset 16    .cfi\_offset 6, -16    movq  %rsp, %rbp    .cfi\_def\_cfa\_register 6    subq  $32, %rsp    movq  %fs:40, %rax    movq  %rax, -8(%rbp)    xorl  %eax, %eax |
| int num, i; |  |
| 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    movq  %rax, %rdx    movq  \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax    movq  %rax, %rsi    movq  %rdx, %rdi    call  \_ZNSolsEPFRSoS\_E@PLT |
| cin>>num; | leaq  -20(%rbp), %rax    movq  %rax, %rsi    leaq  \_ZSt3cin(%rip), %rdi    call  \_ZNSirsERi@PLT |
| for(i=num;i>0; i--) | movl  -20(%rbp), %eax    movl  %eax, -16(%rbp)  .L3:    cmpl  $0, -16(%rbp)    jle .L2  (body of loop goes here)    subl  $1, -16(%rbp)    jmp .L3 |
| product = product \* i; | movl  -12(%rbp), %eax    imull -16(%rbp), %eax    movl  %eax, -12(%rbp) |
| cout<<"The factorial for " << num << "is: \n"<< product; | .L2:    leaq  .LC1(%rip), %rsi    leaq  \_ZSt4cout(%rip), %rdi    call  \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT    movq  %rax, %rdx    movl  -20(%rbp), %eax    movl  %eax, %esi    movq  %rdx, %rdi    call  \_ZNSolsEi@PLT    leaq  .LC2(%rip), %rsi    movq  %rax, %rdi    call  \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT    movq  %rax, %rdx    movl  -12(%rbp), %eax    movl  %eax, %esi    movq  %rdx, %rdi    call  \_ZNSolsEi@PLT |
| return 1; } | movl  $1, %eax    movq  -8(%rbp), %rcx    xorq  %fs:40, %rcx    je  .L5    call  \_\_stack\_chk\_fail@PLT  .L5:    leave    .cfi\_def\_cfa 7, 8    ret    .cfi\_endproc |

**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** |
| --- | --- |
| .LC0:    .string "Enter a number:\n"  .LC1:    .string "The factorial for "  .LC2:    .string "is: \n" | These are the strings that are used in the program |
| .text    .globl  main    .type main, @function  main:  .LFB1493:    .cfi\_startproc    pushq %rbp    .cfi\_def\_cfa\_offset 16    .cfi\_offset 6, -16    movq  %rsp, %rbp    .cfi\_def\_cfa\_register 6    subq  $32, %rsp    movq  %fs:40, %rax    movq  %rax, -8(%rbp)    xorl  %eax, %eax | .text is the beginning of the code section  Main is a global symbol  Main is a function  Start of main function  This is from the compiler for the most part. Creates a main function, allocates 32 bytes for local variables, and creates some sort of buffer for overflow |
| movl  $1, -12(%rbp) | Moves the value of 1 to an offset of 12 from the base pointer. |
| leaq  .LC0(%rip), %rsi    leaq  \_ZSt4cout(%rip), %rdi    call  \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT    movq  %rax, %rdx    movq  \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax    movq  %rax, %rsi    movq  %rdx, %rdi    call  \_ZNSolsEPFRSoS\_E@PLT | This section grabs the string LC0 and outputs it by attaching the string to cout and then calling couting and then calling endline |
| leaq  -20(%rbp), %rax    movq  %rax, %rsi    leaq  \_ZSt3cin(%rip), %rdi    call  \_ZNSirsERi@PLT | This sets the pointer for a location to store user input and then passes that pointer to a call for cin and stores the input (num) in that location |
| movl  -20(%rbp), %eax    movl  %eax, -16(%rbp) | This is the beginning of the for loop where I is set to num |
| .L3:    cmpl  $0, -16(%rbp)    jle .L2 | This is the comparison in the for loop where it compares if I > 0 but does it opposite by comparing if I <= 0. If I <= 0 it jumps to .L2 ending the loop. |
| movl  -12(%rbp), %eax    imull -16(%rbp), %eax    movl  %eax, -12(%rbp) | This is the body of the loop.  Moves the value of offset of 12 from base pointer (product) to eax register. Then multiplies 16 from the base pointer (i) by eax (product). Then moves that total into offset of 12 from base pointer (product). |
| subl  $1, -16(%rbp)    jmp .L3 | This increments I for the loop and jumps back to the comparison part again. |
| .L2:    leaq  .LC1(%rip), %rsi    leaq  \_ZSt4cout(%rip), %rdi    call  \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT    movq  %rax, %rdx    movl  -20(%rbp), %eax    movl  %eax, %esi    movq  %rdx, %rdi    call  \_ZNSolsEi@PLT    leaq  .LC2(%rip), %rsi    movq  %rax, %rdi    call  \_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@PLT    movq  %rax, %rdx    movl  -12(%rbp), %eax    movl  %eax, %esi    movq  %rdx, %rdi    call  \_ZNSolsEi@PLT | This part is for the last cout statement.  String .LC1 is attached to cout and called.  The value of offset of 20 from the base pointer  (num) is attached to cout and called.  String .LC2 is attached to cout and called.  The value of offset of 12 from the base pointer  (product) is attached to cout and called. |
| movl  $1, %eax    movq  -8(%rbp), %rcx    xorq  %fs:40, %rcx    je  .L5    call  \_\_stack\_chk\_fail@PLT  .L5:    leave    .cfi\_def\_cfa 7, 8    ret    .cfi\_endproc | There is some sort of overflow check here that I don’t quite understand with xor on fs:40 and rcx and if they aren’t equal call to\_\_stack\_chk\_fail@PLT, but after that it ends the function and returns 1. |