# **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() | Main function |
| int num, i; | Declare two integers: num and i |
| int product =1; | Declare integer product and initilize to 1 |
| cout<<"Enter a number:\n"<< endl; | Print the message to the console with a new line |
| cin>>num; | Take user input and assign to variable num |
| for(i=num;i>0; i--)  product = product \* i; | Loop as many times num is equal to. For each loop assign the value of product multiplied by i to product. |
| cout<<"The factorial for " << num << "is: \n"<< product; | Print out a message with the factorial of variable num. |

**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 () | .globl main  .type main, @function  main: |
| int product =1; | movl $1, -4(%rbp) |
| cout<<"Enter a number:\n"<< endl; | .LC0:  .string "Enter a number:\n"  subq $16, %rsp  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; | movq %fs:40, %rax  movq %rax, -8(%rbp)  xorl %eax, %eax  leaq -16(%rbp), %rax  movq %rax, %rsi  leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT  movq -8(%rbp), %rcx  xorq %fs:40, %rcx  je .L3  call \_\_stack\_chk\_fail@PLT |
| for(i=num;i>0; i--){} | movl -20(%rbp), %eax  movl %eax, -16(%rbp)  .L3:  cmpl $0, -16(%rbp)  jle .L2  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; | .LC1:  .string "The factorial for "  .LC2:  .string "is: \n"  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 |

**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** |
| .globl main  .type main, @function  main: | The global directive tells the assembler to add the main label. The type directive tells the assembler that main is executable code. Main: is the entry point for the function. |
| movl $1, -4(%rbp) | Rbp -4 is assigned value 1 |
| .LC0:  .string "Enter a number:\n"  subq $16, %rsp  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 | Enter number string is stored at .LC0. Stack space is allocated. Pointers are stored for various values on the stack. Then the cin function is called. Movq is used to move values to optimize stack. movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax is moving the value taken from cin. More movement is done and then a call to std::endl is done. |
| movq %fs:40, %rax  movq %rax, -8(%rbp)  xorl %eax, %eax  leaq -16(%rbp), %rax  movq %rax, %rsi  leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT  movq -8(%rbp), %rcx  xorq %fs:40, %rcx  je .L3  call \_\_stack\_chk\_fail@PLT | Movement is done along the registers to optimize the stack and use values for functions. Xorl is used to zero out a register. A call is made to the cin function and that value is stored. More movement and zeroing is done. The assembler then jumps to .L3 if the conditions are met and a call to stack\_chk\_ fail is done using the procedure linkage table. |
| movl -20(%rbp), %eax  movl %eax, -16(%rbp)  .L3:  cmpl $0, -16(%rbp)  jle .L2  subl $1, -16(%rbp)  jmp .L3 | Int i is set to num and the value is moved around in the registers. The assembler then compares the value at -16 rbp against 0 to see if the loop is finished. If not then 1 is subtracted from i and it loops again. |
| movl -12(%rbp), %eax  imull -16(%rbp), %eax  movl %eax, -12(%rbp) | Inside of the loop the product moves registers. It is multiplied with i and that value is then stored back at -12 rbp. |
| .LC1:  .string "The factorial for "  .LC2:  .string "is: \n"  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 | The constant strings are stored at .LC1 and .LC2. The values move registers to where they can be used and then each one is passed to cout. |