# **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> |  |
| using namespace std; |  |
| int main() | The entry point of the program |
| { | Open braces for the main function |
| int num, i; | Declare two integer variables num,i without assigning a value. |
| int product =1; | Declare int variable product and assign a value 1 to it. |
| cout<<"Enter a number:\n"<< endl; | Printout Enter a number: and move the console cursor to next line. |
| cin>>num; | Fetch a value from the user and assign it to num |
| for(i=num;i>0; i--) | For loop from entered number to 1 |
| product = product \* i; | Multiply product by the i value and assign the result in product |
| cout<<"The factorial for " << num << "is: \n"<< product; | Printout The Factional for num is: and move to next line and printout the product value |
| return 1; | Return 1 to the caller |
| } |  |

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

**.LC0:**

**.string "Enter a number:\n"**

**.LC1:**

**.string "The factorial for "**

**.LC2:**

**.string "is: \n"**

**subq $32, %rsp**

**movq %fs:40, %rax**

**movq %rax, -8(%rbp)**

**xorl %eax, %eax**

**movl $1, -12(%rbp)**

**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**

**leaq -20(%rbp), %rax**

**movq %rax, %rsi**

**leaq \_ZSt3cin(%rip), %rdi**

**call \_ZNSirsERi@PLT**

**movl -20(%rbp), %eax**

**movl %eax, -16(%rbp)**

**.L3:**

**cmpl $0, -16(%rbp)**

**jle .L2**

**movl -12(%rbp), %eax**

**imull -16(%rbp), %eax**

**movl %eax, -12(%rbp)**

**subl $1, -16(%rbp)**

**jmp .L3**

**.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**

**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**

**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> | Before calling the main function the disassembler created data segment where global initialed variables are declared.  Gnu g++ creates global directives for each string in the program  .LC0:  .string "Enter a number:\n"  .LC1:  .string "The factorial for "  .LC2:  .string "is: \n" |
| using namespace std; |  |
| int main() |  |
| { | subq $32, %rsp  movq %fs:40, %rax |
| int num, i; | movq %rax, -8(%rbp) |
| int product =1; | xorl %eax, %eax  movl $1, -12(%rbp) |
| cout<<"Enter a number:\n"<< endl; | leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt11char\_traitsIcEERSt13basic\_ PKc@PLT  movq %rax, %rdx  movq \_ZSt4endlIcSt11char\_traitsIcE…\_ES6\_REL(%rip), %rax  movq %rax, %rsi  movq %rdx, %rdi  call \_ZNSolsEPFRSoS\_E@PLT  leaq -20(%rbp), %rax  movq %rax, %rsi |
| cin>>num; | leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT  movl -20(%rbp), %eax  movl %eax, -16(%rbp) |
| for(i=num;i>0; i--)   product = product \* i; | .L3:  cmpl $0, -16(%rbp)  jle .L2  movl -12(%rbp), %eax  imull -16(%rbp), %eax  movl %eax, -12(%rbp)  subl $1, -16(%rbp)  jmp .L3 |
|  |
| cout<<"The factorial for " << num << "is: \n"<< product; | .L2:  leaq .LC1(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt1113basic\_ostreamIcT\_ES5....@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\_traitsIcEER….\_ES5\_PKc@PLT  movq %rax, %rdx  movl -12(%rbp), %eax  movl %eax, %esi  movq %rdx, %rdi  call \_ZNSolsEi@PLT  movl $1, %eax  movq -8(%rbp), %rcx  xorq %fs:40, %rcx  je .L5  call \_\_stack\_chk\_fail@PLT |
| return 1; | .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" | Store the string data we are using in the program  With different directives for each statement |
| subq $32, %rsp  movq %fs:40, %rax | Subtract 32 bytes from Stack Pointer which is convention in x86-64 (Spell Space)  Move value stored at offset 40 into RAX register |
| movq %rax, -8(%rbp) | Move the previous value in RAX into a memory location at offset -8 from base pointer (RBP) |
| xorl %eax, %eax  movl $1, -12(%rbp) | Xor will set EAX to zero.  Move the value 1 to base register address -12 bytes |
| leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt11char\_traibasic\_ PKc@PLT  movq %rax, %rdx  movq \_ZSt4endlIcSt11…\_ES6\_REL(%rip), %rax  movq %rax, %rsi  movq %rdx, %rdi  call \_ZNSolsEPFRSoS\_E@PLT  leaq -20(%rbp), %rax  movq %rax, %rsi | Store the address of “Enter a number:\n” address into RSI  I guess the compiler during the compile time will have the address of all standard functions and reference to them through address and must make sure it’s unique across the whole binary (PLT, Procedure Linkage Table)  Call cout class with << operator overloading to printout “Enter a number ” and then call endl |
| leaq \_ZSt3cin(%rip), %rdi  call \_ZNSirsERi@PLT  movl -20(%rbp), %eax  movl %eax, -16(%rbp) | 1. Fetch the address of cin into %rdi 2. Call cin function 3. Store the value at offset -20 from base pointer into EAX register 4. Store EAX value into memory address -16(%rbp) |
| .L3:  cmpl $0, -16(%rbp)  jle .L2  movl -12(%rbp), %eax  imull -16(%rbp), %eax  movl %eax, -12(%rbp)  subl $1, -16(%rbp)  jmp .L3 | Loop code with Label L3 so we can perform goto.  check if Mem Address 16(%rbp) is zero  Jump to L2 if Mem Address less/equal zero  Assign Product value to EAX  Multiply Mem Address by EAX and store in EAX  Move EAX value to -12 (%RBP)  Decrement Mem value by 1, we can use DEC in MASM  Jump to the beginning of the loop. |
| .L2:  leaq .LC1(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZStlsISt1113basT\_ES5....@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 \_ZStlsISttsIcEER….\_ES5\_PKc@PLT  movq %rax, %rdx  movl -12(%rbp), %eax  movl %eax, %esi  movq %rdx, %rdi  call \_ZNSolsEi@PLT  movl $1, %eax  movq -8(%rbp), %rcx  xorq %fs:40, %rcx  je .L5  call \_\_stack\_chk\_fail@PLT | Store the address of LC1 “The factorial for” in rsi register and call cout function  And print out the value of num which -20(%rbp)  And then get the address of string “"is: \n" and print it out.  Print out the value from -12(%rbp) which is stored into EAX register and print out the value which is the product.  Finally assign the value 1 to EAX because EAX is the default register to store the return value from procedure  Jump to .L5 where ret instruction return from the proc to caller |
| .L5:  leave  .cfi\_def\_cfa 7, 8  ret  .cfi\_endproc | Return from the procedure |

**NOTE: I have shortened most of the function symbols to fit into the table.**