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

Janera Dobson

Southern New Hampshire University

CS-410-T3249

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

## C++ Code Functionality

| **C++ Line of Code** | **Explanation of Functionality** |
| --- | --- |
| #include<iostream> | Tells preprocessor to include content of iostream at the very beginning of the program before compiler starts the actual compilation of the code. |
| using namespace std; | Tells the compiler that symbol names defined in the std namespace are brought into the programs scope. Therefore, you can omit the namespace qualifier. |
| int main()  { | Tells the compiler that our function needs to return some integer at the end of the execution, and we do so by returning 0 at the end of the program.  Tells the compiler it’s the opening of the scope to the main method. |
| int num, i; | Declares integer variables “num” and “i” |
| int product =1; | Declares integer variable “product” with the value of 1 |
| cout<<"Enter a number:\n"<< endl; | Prints “Enter a number:” followed by a newline |
| cin>>num; | Console prompts user to input a value for int “num” |
| for(i=num;i>0; i--)  product = product \* i; | Loop initiates “i” as “num”, checks to see if “i” is greater than 0, and then subtract 1 from i. Then the value of “product” is multiplied by “i” and store it back in the variable product. |
| cout<<"The factorial for " << num << "is: \n"<< product; | Tells the compiler to print out the String “The factorial for “, insert the value of “num”, the String “is:”, and a newline followed by the new values stored in the variable named “product”. |
| return 1; | The return keyword is used to return some value and here it’s 1 which means success and will terminate the main function with this return statement but returning a non-zero number means failure. |
| } | Tells the compiler this is the end of the main methods scope. |

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

**.file "assignment2\_1.cpp"**

**.text**

**.section .rodata**

**.type \_ZStL19piecewise\_construct, @object**

**.size \_ZStL19piecewise\_construct, 1**

**\_ZStL19piecewise\_construct:**

**.zero 1**

**.local \_ZStL8\_\_ioinit**

**.comm \_ZStL8\_\_ioinit,1,1**

**.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**

**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**

**.cfi\_endproc**

**.LFE1493:**

**.size main, .-main**

**.type \_Z41\_\_static\_initialization\_and\_destruction\_0ii, @function**

**\_Z41\_\_static\_initialization\_and\_destruction\_0ii:**

**.LFB1982:**

**.cfi\_startproc**

**pushq %rbp**

**.cfi\_def\_cfa\_offset 16**

**.cfi\_offset 6, -16**

**movq %rsp, %rbp**

**.cfi\_def\_cfa\_register 6**

**subq $16, %rsp**

**movl %edi, -4(%rbp)**

**movl %esi, -8(%rbp)**

**cmpl $1, -4(%rbp)**

**jne .L8**

**cmpl $65535, -8(%rbp)**

**jne .L8**

**leaq \_ZStL8\_\_ioinit(%rip), %rdi**

**call \_ZNSt8ios\_base4InitC1Ev@PLT**

**leaq \_\_dso\_handle(%rip), %rdx**

**leaq \_ZStL8\_\_ioinit(%rip), %rsi**

**movq \_ZNSt8ios\_base4InitD1Ev@GOTPCREL(%rip), %rax**

**movq %rax, %rdi**

**call \_\_cxa\_atexit@PLT**

**.L8:**

**nop**

**leave**

**.cfi\_def\_cfa 7, 8**

**ret**

**.cfi\_endproc**

**.LFE1982:**

**.size \_Z41\_\_static\_initialization\_and\_destruction\_0ii, .-\_Z41\_\_static\_initialization\_and\_destruction\_0ii**

**.type \_GLOBAL\_\_sub\_I\_main, @function**

**\_GLOBAL\_\_sub\_I\_main:**

**.LFB1983:**

**.cfi\_startproc**

**pushq %rbp**

**.cfi\_def\_cfa\_offset 16**

**.cfi\_offset 6, -16**

**movq %rsp, %rbp**

**.cfi\_def\_cfa\_register 6**

**movl $65535, %esi**

**movl $1, %edi**

**call \_Z41\_\_static\_initialization\_and\_destruction\_0ii**

**popq %rbp**

**.cfi\_def\_cfa 7, 8**

**ret**

**.cfi\_endproc**

**.LFE1983:**

**.size \_GLOBAL\_\_sub\_I\_main, .-\_GLOBAL\_\_sub\_I\_main**

**.section .init\_array,"aw"**

**.align 8**

**.quad \_GLOBAL\_\_sub\_I\_main**

**.hidden \_\_dso\_handle**

**.ident "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"**

**.section .note.GNU-stack,"",@progbits**

**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** |
| --- | --- |
| "Enter a number:\n"  "The factorial for "  "is: \n" | .LC0:  .string "Enter a number:\n"  .LC1:  .string "The factorial for "  .LC2:  .string "is: \n" |
| int num, i; | movq %rax, -8(%rbp) |
| int product =1; | movl $1, -12(%rbp) |
| cout<<"Enter a number:\n"<< endl;  cin>>num; | leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call puts  leaq -20(%rbp), %rax  movq %rax, %rsi  leaq \_ZSt3cin(%rip), %rdi |
| for(i=num;i>0; i--)  product = product \* i; | .L2:  cmpl $1, -4(%rbp)  je .L5  .L3:  cmpl $0, -16(%rbp)  jle .L2  movl -12(%rbp), %eax  imull -16(%rbp), %eax  movl %eax, -12(%rbp)  subl $1, -16(%rbp) |
| cout<<"The factorial for " << num << "is: \n"<< product; | movl -20(%rbp), %eax  movl -12(%rbp), %eax  movl %eax, %esi  leaq .LC2(%rip), %rsi  movl -12(%rbp), %eax  call cout |
| return 1; | 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** |
| --- | --- |
| .LC0:  .string "Enter a number:\n"  .LC1:  .string "The factorial for "  .LC2:  .string "is: \n" | Declares three String variables used to display messages to the user. |
| movq %rax, -8(%rbp) | Declares int values in register |
| leaq .LC0(%rip), %rsi  leaq \_ZSt4cout(%rip), %rdi  call puts  leaq -20(%rbp), %rax  movq %rax, %rsi  leaq \_ZSt3cin(%rip), %rdi | The value of .LC0 is now moved to %rsi by calling puts.  Reads the next value into the console using cin. This value is then |
| .L2:  cmpl $1, -4(%rbp)  je .L5  .L3:  cmpl $0, -16(%rbp)  jle .L2  movl -12(%rbp), %eax  imull -16(%rbp), %eax  movl %eax, -12(%rbp)  subl $1, -16(%rbp) | We compile here first to check and compare if the value in -4(%rbp) to 1. If the comparison result is less than, we jump to .L5. If the comparison is greater than 1, that means we need to execute the code inside the loop, so jump to .L3. |
| movl -20(%rbp), %eax  movl -12(%rbp), %eax  movl %eax, %esi  leaq .LC2(%rip), %rsi  movl -12(%rbp), %eax  call cout | Here we move the value -20(%rbp) to be be stored to %eax. Then the value-12(%rbp) must be moved to eax in order to prints the String and value of .L2 to the screen by calling cout. |
| ret | Code done and terminates the program. |

Reference:

“Guide to Software Reverse Engineering.” <https://learn.snhu.edu/content/enforced/638373-CS-410-T3249-OL-TRAD-UG.21EW3/course_documents/CS%20410%20Guide%20to%20Software%20Reverse%20Engineering.pdf?_&d2lSessionVal=uWMB6LZaGTKjT4uGNTpajSZHY&ou=638373>.