# **CS 410 C++ to Assembly 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()  {  int width=10;  int height=5;  int area;  area = width \* height;    cout<<endl<< area;  return 0;  } | This simple c++ code starts by importing the input and output functionality. Then declares the main function with 3 variables and finally then puts the third variable equal to the the first and second variable multiplied. Then it prints to the terminal to end the line and display the variable on the terminal line. Lastly it ends the function by returning zero. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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; | .file "assignment1\_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  .text |
| int main(){  int width=10;  int height=5;  int area;  area = width \* height;  cout<<endl<< area;  return 0;  } | .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 $16, %rsp  movl $10, -12(%rbp)  movl $5, -8(%rbp)  movl -12(%rbp), %eax  imull -8(%rbp), %eax  movl %eax, -4(%rbp)  movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax  movq %rax, %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZNSolsEPFRSoS\_E@PLT  movq %rax, %rdx  movl -4(%rbp), %eax  movl %eax, %esi  movq %rdx, %rdi  call \_ZNSolsEi@PLT  movl $0, %eax  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:  .LFB1979:  .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 .L5  cmpl $65535, -8(%rbp)  jne .L5  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  .L5:  nop  leave  .cfi\_def\_cfa 7, 8  ret  .cfi\_endproc  .LFE1979:  .size \_Z41\_\_static\_initialization\_and\_destruction\_0ii, .-\_Z41\_\_static\_initialization\_and\_destruction\_0ii  .type \_GLOBAL\_\_sub\_I\_main, @function |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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** |
| --- | --- |
| .file "assignment1\_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  .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 $16, %rsp  movl $10, -12(%rbp)  movl $5, -8(%rbp)  movl -12(%rbp), %eax  imull -8(%rbp), %eax  movl %eax, -4(%rbp)  movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax  movq %rax, %rsi  leaq \_ZSt4cout(%rip), %rdi  call \_ZNSolsEPFRSoS\_E@PLT  movq %rax, %rdx  movl -4(%rbp), %eax  movl %eax, %esi  movq %rdx, %rdi  call \_ZNSolsEi@PLT  movl $0, %eax  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:  .LFB1979:  .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 .L5  cmpl $65535, -8(%rbp)  jne .L5  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  .L5:  nop  leave  .cfi\_def\_cfa 7, 8  ret  .cfi\_endproc  .LFE1979:  .size \_Z41\_\_static\_initialization\_and\_destruction\_0ii, .-\_Z41\_\_static\_initialization\_and\_destruction\_0ii  .type \_GLOBAL\_\_sub\_I\_main, @function  \_GLOBAL\_\_sub\_I\_main:  .LFB1980:  .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  .LFE1980:  .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 | To start we have the line .file which simply reads the name of our file. A little bit down we see the @funtion call which is where out int main would be. In assembly the program needs to be told when to start reading and that is what the .cfi\_startproc is doing. The pushq line is used for when you want to declare/push a certain amount of bytes to the stack. Moving down the the movl line is where we see in this assembly language that the variables are being declared comparatively to our cpp code. Something to note is that it can be different in almost every assembly language. All the assembly languages look almost completely different. Although the concepts are similar. Also the operating system process assembly differently. Although the languages look vastly different they do the same thing. CPP is broken down into assembly and then down again to binary. There is many layers into programming with only more to come in the future. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |