**1-4 Activity: C++ to Assembly**

Dylan Jeffery

SNHU

CS-410

Brian West

1/9/2022

# **CS 410 C++ to Assembly Activity**

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

## C++ Code Functionality

| **C++ Line of Code** | **Explanation of Functionality** |
| --- | --- |
| #include<iostream> | Header: Includes input/output stream |
| using namespace std; | Use std namespace (don’t need std:: can just put cout) |
| int main() | Main method |
| { | Opening bracket of main method |
| int width=10; | Set int width equal to 10 |
| int height=5; | Set int height equal to 5 |
| int area; | Initialize variable area(int); |
| area = width \* height; | Calculates value of area based on width and height variables |
| cout<<endl<< area; | Console out  End line  Print area |
| return 0; | Indicates normal exit status |
| } | Close main method |

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

.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

**Step 3:** Align each line of C++ code with the corresponding blocks of assembly code.

## C++ to Assembly Alignment

Individual blocks here are from incrementally creating assembly line by line for comparison. Final register values are reflected correctly in the code block (~line 18) above.

| **C++ Line of Code** | **Blocks of Assembly Code** |
| --- | --- |
| int width=10; | movl $10, -4(%rbp)  movl $0, %eax  popq %rbp  .cfi\_def\_cfa 7, 8 |
| int height=5; | movl $5, -4(%rbp)  movl $0, %eax  popq %rbp  .cfi\_def\_cfa 7, 8 |
| int area;//init alone has no fx  area = width \* height; | movl $10, -12(%rbp)  movl $5, -8(%rbp)  movl -12(%rbp), %eax  imull -8(%rbp), %eax  movl %eax, -4(%rbp) |
| cout<<endl<< area; | subq $16, %rsp  …  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 |

**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** |
| --- | --- |
| movl $10, -12(%rbp) | Move 10, 12 bytes above register %rgp |
| movl $5, -8(%rbp) | Move 5, 8 bytes above register %rbp |
| movl -12(%rbp), %eax  imull -8(%rbp), %eax  movl %eax, -4(%rbp) | Move contents of rbp @12 bytes to %eax  Multiply contents of rbp @8 by contents of %eax  Move contents of %eax 4 bytes above register %rbp |
| movq \_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_@GOTPCREL(%rip), %rax  movq %rax, %rsi | Move (endl, char variables?) vars filled by dynamic linker to %rax  Move %rax -> %rsi  movq is used in place of movl when 64-bit data needs to be moved instead of 32-bit data |
| leaq \_ZSt4cout(%rip), %rdi  call \_ZNSolsEPFRSoS\_E@PLT  movq %rax, %rdx | LEA is short for Load Effective Address which acts as a pointer.  I think this call is referring to some piecewise object but I’m unsure how it translates (Obviously related to cout line but beyond that I am unsure)?  Then (64-bit) move %rax to %rdx |
| movl -4(%rbp), %eax | Move value from 4 bytes above %rbp to %eax |
| movl %eax, %esi | Move contents of register %eax to register %esi |
| movq %rdx, %rdi | (64-bit) Move contents of %rdx to %rdi |
| call \_ZNSolsEi@PLT | This call is something I’m unsure of. I believe PLT refers to a procedure linkage table. Used for external function/procedure calls resolved by the linker at run time. |
| movl $0, %eax  leave | Code is done, terminate program. |