# **CS 410 Binary to C++ Activity Template**

## **File One: assignment4\_1.o**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| main:   1. push %rbp 2. mov %rsp, %rbp 3. sub 0x10, %rsp, 4. mov 0x1, -0x8 (%rbp) 5. cmp %rbp, 0x9 6. jg 0xa3 7. mov %rbp, 0x1 8. cmp %rbp, 0x9 9. jg 0x9a 10. mov %eax, -0x8 11. imul %eax, -0xc 12. mov -0x4, %eax 13. mov %eax, -0x8 14. mov %esi, %eax 15. lea %rdi, std::cout 16. call put 17. mov %rdi, %rax 18. call put 19. mov %rdx, %rax 20. mov %eax, -0x4 21. mov %esi, %eax 22. mov %rdi, %rdx 23. call put 24. mov %rdx, %rax 25. mov %rax, -0x4 26. mov %rax, std::endl 27. mov %rsi, %rax 28. mov %rdi, %rdx 29. call put 30. add -0xc, 0x1 31. jmp 0x20 | Sets variables a, i and x  Move 1 into register -0x8 (%rbp)  Compare values of %rbp to 9  If value is greater than 9 jump to 0xa3  If value is less than or equal to 9  Move 1 into %rbp  Compare values of %rbp to 9  If value is greater than 9 jump 0x9a  If value is less than or equal to 9  Multiply variables a and i  Output value of a \* value of i = result of multiplications |
| 0xa3:   1. mov %eax, 0x0 2. leave %rbp 3. retn | Move 0 int eax  Leave  Return 0 |
| 0x9a:   1. add -0x8, 0x1 2. jmp 0xf | Add 1 to variable i  Jump back to second loop |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main:   1. push %rbp 2. mov %rsp, %rbp 3. sub 0x10, %rsp, 4. mov 0x1, -0x8 (%rbp) 5. cmp %rbp, 0x9 6. jg 0xa3 7. mov %rbp, 0x1 8. cmp %rbp, 0x9 9. jg 0x9a 10. mov %eax, -0x8 11. imul %eax, -0xc 12. mov -0x4, %eax 13. mov %eax, -0x8 14. mov %esi, %eax 15. lea %rdi, std::cout 16. call put 17. mov %rdi, %rax 18. call put 19. mov %rdx, %rax 20. mov %eax, -0x4 21. mov %esi, %eax 22. mov %rdi, %rdx 23. call put 24. mov %rdx, %rax 25. mov %rax, -0x4 26. mov %rax, std::endl 27. mov %rsi, %rax 28. mov %rdi, %rdx 29. call put 30. add -0xc, 0x1   jmp 0x20 | int main() {  int i, a, x;  for (a = 1; a <= 9; a++) {  for (i = 1; i <= 9; i++) {  x = a \* i;  cout << a << “ \* “ << i << “ = “ << x << endl;  }  } | Declare number, i, a, x  For a = 1, if a is less than or equal to 9 continue  Otherwise exit the for loop  For i = 1, if i is less than or equal to 9 continue  Otherwise exit the for loop  x = a \* i  Output a “ \* “ i “ = “ x |
| 0xa3:   1. mov %eax, 0x0 2. leave %rbp 3. retn | return 0; | return 0 |
| 0x9a:   1. add -0x8, 0x1 2. jmp 0xf | i++ | Add 1 to i and then loop |
|  |  |  |
|  |  |  |

## **File Two: assignment4\_2.o**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 1. pushq %rbp 2. movq %rsp, %rbp 3. subq 0x10, %rsp 4. movl 0x0, -0x4 (%rbp) 5. movss .long, %xmm0 6. movss %xmm0, -0x8 (%rbp) 7. leaq “Enter Radius”(%rip), %rsi 8. callq put 9. movq std::cin(%rip), %rdi 10. leaq -0x12(%rbp), %rsi 11. cmpl 0x0, -0x12(%rbp) 12. jne .LBB2 13. movl 0x0, -0x4 (%rbp) 14. jmp .LBB4 | 1. Create variable num 2. Copies value rbp to rsp 3. Extend rsp to register location 0x10 4. Move 0 to location 0x4 5. Move 3.14000001 to memory location xmm0 6. Move from xmm0 to local 0x8 7. Store Enter Radius into rsi 8. Output Enter Radius 9. Copies value of rdi to rax 10. Create and store value from input to 0x12 11. Compare 0 to the number inputted 12. If inputted number not equal zero, jump 13. Move 0 to 0x4 location 14. Jump to LBB4 |
| .LBB2   1. movl -0x12(%rbp), %eax 2. imull -0x12(%rbp), %eax 3. imull -0x12(%rbp), %eax 4. cvtsi2ss %eax, %xmm0 5. mulss -0x8 (%rbp), %xmm0 6. cvttss2si %xmm0, %eax 7. movl %eax, -0x12(%rbp) 8. leaq “The volume is: “, %rsi 9. callq put 10. movq %rax, %rdi 11. movl -0x12(%rbp), %esi 12. callq put 13. movq %rax, %rdi 14. callq %rsi | 1. Move value of 0x12 to eax 2. Multiply values of eax and 0x12 3. Multiply values of eax and 0x12 4. Convert eax to a float and store in memory 5. Multiply values in 0x8 and xmm0 6. Convert xmm0 from float to integer and store in eax 7. Move eax into location 0x12 8. Store “The volume is: “ in rsi 9. Call cout 10. Copies value of rdi into rax 11. Moves the value in 0x12 to esi 12. Call << 13. Copies value of rdi into rax 14. Output rsi and value in esi |
| .LBB4   1. movl 0x0, -0x4 2. leave 3. retn 0x4 | 1. Move 0 to location 0x4 2. Exit 3. Return 0 |
|  |  |
|  |  |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| 1. pushq %rbp 2. movq %rsp, %rbp 3. subq 0x10, %rsp 4. movl 0x0, -0x4 (%rbp) 5. movss .long, %xmm0 6. movss %xmm0, -0x8 (%rbp) 7. leaq “Enter Radius”(%rip), %rsi 8. callq put 9. movq std::cin(%rip), %rdi 10. leaq -0x12(%rbp), %rsi 11. cmpl 0x0, -0x12(%rbp) 12. jne .LBB2 13. movl 0x0, -0x4 (%rbp) 14. jmp .LBB4 | int main() {  float pi = 3.14000001;  int num;  cout << “Enter Radius: “ << endl;  cin >> num;  if (num == 0) {  return 0;  } else {  Jump to LBB2  }  } | Create float variable pi  Create integer variable num  Output Enter Radius  Take input and assign it to variable num  If num equals 0 jump to LBB4 which returns 0  If num does not equal 0 jump to LBB2 |
| .LBB2   1. movl -0x12(%rbp), %eax 2. imull -0x12(%rbp), %eax 3. imull -0x12(%rbp), %eax 4. cvtsi2ss %eax, %xmm0 5. mulss -0x8 (%rbp), %xmm0 6. cvttss2si %xmm0, %eax 7. movl %eax, -0x12(%rbp) 8. leaq “The volume is: “, %rsi 9. callq put 10. movq %rax, %rdi 11. movl -0x12(%rbp), %esi 12. callq put 13. movq %rax, %rdi 14. callq %rsi | num = (num \* num \* num) \* pi  cout << “The volume is: “ << num << endl; | num equals num times num times num times pi  Output “The volume is: “num |
| .LBB4 |  |  |
| 1. movl 0x0, -0x4 2. Leave 3. retn 0x4 | return 0 | Returns 0; |
|  |  |  |

## **File Three: assignment4\_3.o**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 1. push %rbp 2. mov %rsp,%rbp 3. sub 0x20,%rsp 4. mov %fs:0x28,%rax 5. mov %rax,-0x8(%rbp) 6. xor %eax,%eax 7. movl 0x1,-0xc(%rbp) 8. lea 0x0(%rip),%rsi 9. lea 0x0(%rip),%rdi 10. call 0x31 11. mov %rax,%rdx 12. mov 0x0(%rip),%rax # 0x3b 13. mov %rax,%rsi 14. mov %rdx,%rdi 15. call 0x46 16. lea -0x18(%rbp),%rax 17. mov %rax,%rsi 18. lea 0x0(%rip),%rdi # 0x54 19. call 0x59 20. mov -0x18(%rbp),%eax 21. sub 0x1,%eax 22. mov %eax,-0xc(%rbp) 23. movl 0x1,-0x10(%rbp) | 1. Declare rows, input, i, j 2. Move rbp into rsp 3. Gets “Enter number of rows” from stack 4. Move fsbase into rax 5. Move from rax into 0x8 6. Set eax to 0 7. Move 1 into 0xc 8. Store 0 in rsi 9. Store 0 in rdi 10. Output “Enter number of rows 11. Move rax to rdx 12. Move 0 to rax 13. Move rax to rsi 14. Move rdx to rdi 15. Call << 16. Store rax into 0x18 17. Move rax into rsi 18. Assign std::cin >> to rdi 19. Call rdi 20. Move 0x18 to eax 21. Assigned 0x18 to value of 1 22. Move 0xc to eax 23. Move 0x10 to value 1 |
| 1. mov -0x18(%rbp),%eax 2. cmp %eax,-0x10(%rbp) 3. jg 0xe3 | 1. Move eax to 0x18 2. Compare 0x10 to eax 3. If 0x18 is greater than eax jump to 0xe3 |
| 1. mov 0x14, 0x1 | 1. Move 1 into 0x14 |
| 1. mov eax, 0x14 2. cmp eax, 0xc 3. jg 0x99 | 1. Move 0x14 into eax 2. Compare eax to 0xc 3. If 0xc is greater than eax jump to 0x99 |
| 1. lea 0x0(%rip),%rsi # 0x87 2. lea 0x0(%rip),%rdi # 0x8e 3. call 0x93 4. addl 0x1,-0x14(%rbp) 5. jmp 0x78 | 1. Store “ “ to rsi 2. Store cout to rdi 3. Output cout << “ “ 4. Add 1 to 0x14 5. Jump to 0x78 |
| 1. subl 0x1,-0xc(%rbp) 2. movl 0x1,-0x14(%rbp) 3. mov -0x10(%rbp),%eax 4. add %eax,%eax 5. sub 0x1,%eax 6. cmp %eax,-0x14(%rbp) 7. jg 0xca | 1. Subtract 1 from 0xc 2. Move 1 to 0x14 3. Move 0x10 to eax 4. Add eax to eax 5. Get eax from stack 6. Compare 0x14 to eax 7. If 0x14 is greater than eax jump to 0xca |
| 1. lea 0x0(%rip),%rsi # 0xb8 2. lea 0x0(%rip),%rdi # 0xbf 3. call 0xc4 4. addl 0x1,-0x14(%rbp) 5. jmp 0xa4 | 1. Assign “ \* “ to rsi 2. Assign cout to rdi 3. Output cout << “ “ 4. Add 1 to 0x14 5. Jump to 0xa4 |
| 1. lea 0x0(%rip),%rsi # 0xd1 2. lea 0x0(%rip),%rdi # 0xd8 3. call 0xdd 4. addl 0x1,-0x10(%rbp) 5. jmp 0x69 | 1. Assign “\n” to rsi 2. Assign cout to rdi 3. Output cout << “\n” 4. Add 1 to 0x10 5. Jump to 0x69 |
| 1. movl 0x1,-0xc(%rbp) 2. movl 0x1,-0x10(%rbp) 3. mov -0x18(%rbp),%eax 4. sub 0x1,%eax 5. cmp %eax,-0x10(%rbp) 6. jg 0x171 | 1. Move 1 into 0xc 2. Move 1 into 0x10 3. Move 0x18 into eax 4. Gets and uses eax 5. Compare 0x10 and eax 6. If 0x10 is greater than eax jump to 0x171 |
| 1. movl 0x1,-0x14(%rbp) 2. mov -0x14(%rbp),%eax 3. cmp -0xc(%rbp),%eax 4. jg 0x124 | 1. Move 0x1 into 0x14 2. Move 0x14 into eax 3. Compare 0xc and eax 4. If 0x14 is greater than eax jump to 0x124 |
| 1. lea 0x0(%rip),%rsi # 0x112 2. lea 0x0(%rip),%rdi # 0x119 3. call 0x11e 4. addl 0x1,-0x14(%rbp) 5. jmp 0x103 | 1. Store “ “ into rsi 2. Store cout into rdi 3. Output cout << “ “ 4. Add 1 to 0x14 5. Jump to 0x103 |
| 1. addl 0x1,-0xc(%rbp) 2. movl 0x1,-0x14(%rbp) 3. mov -0x18(%rbp),%eax 4. sub -0x10(%rbp),%eax 5. add %eax,%eax 6. sub 0x1,%eax 7. cmp %eax,-0x14(%rbp) 8. jg 0x158 | 1. Add 1 to 0xc 2. Move 1 into 0x14 3. Move 0x18 into eax 4. Gets 0x10 from stack and use it in eax 5. Adds eax to eax 6. Gets 0x1 from stack and use in eax 7. Compare 0x14 and eax 8. If 0x14 is greater than eax jump to 0x158 |
| 1. lea 0x0(%rip),%rsi # 0x146 2. lea 0x0(%rip),%rdi # 0x14d 3. call 0x152 4. addl 0x1,-0x14(%rbp) 5. jmp 0x12f | 1. Assign “ \* “ to rsi 2. Assign cout to rdi 3. Output cout << “ \* “ 4. Add 1 to 0x14 5. Jump to 0x12f |
| 1. lea 0x0(%rip),%rsi # 0x15f 2. lea 0x0(%rip),%rdi # 0x166 3. call 0x16b 4. addl 0x1,-0x10(%rbp) 5. jmp 0xf1 | 1. Assign “\n” to rsi 2. Assign cout to rdi 3. Output cout << “\n” 4. Add 1 to 0x10 5. Jump to 0xf1 |
| 1. mov 0x1,%eax 2. mov -0x8(%rbp),%rcx 3. xor %fs:0x28,%rcx 4. je 0x18a | 1. Move 0x1 into eax 2. Move 0x8 into rcx 3. Sets 0x28 to rcx 4. If 0x28 is equal to rcx jump to 0x18a |
| 1. leave 2. ret | Exit  Return 1 |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| 1. push %rbp 2. mov %rsp,%rbp 3. sub 0x20,%rsp 4. mov %fs:0x28,%rax 5. mov %rax,-0x8(%rbp) 6. xor %eax,%eax 7. movl 0x1,-0xc(%rbp) 8. lea 0x0(%rip),%rsi 9. lea 0x0(%rip),%rdi 10. call 0x31 11. mov %rax,%rdx 12. mov 0x0(%rip),%rax # 0x3b 13. mov %rax,%rsi 14. mov %rdx,%rdi 15. call 0x46 16. lea -0x18(%rbp),%rax 17. mov %rax,%rsi 18. lea 0x0(%rip),%rdi # 0x54 19. call 0x59 20. mov -0x18(%rbp),%eax 21. sub 0x1,%eax 22. mov %eax,-0xc(%rbp) 23. movl 0x1,-0x10(%rbp) | #include <iostream>  using namespace std;  int input = 0;  int main() {  int rows, i, j;    cout << "Enter number of rows" << endl;  cin >> input;  rows = input - 1;  j = 1; | Declare variables rows, i, j, input  Set input = 0  Output Enter number of rows  Take input and assign to input variable  Assign rows to input minus 1  Assign j = 1 |
| 1. mov -0x18(%rbp),%eax 2. cmp %eax,-0x10(%rbp) 3. jg 0xe3 | For (j = 1; j <= input; j++) { | For loop if j is less than equal to input |
| 1. mov 0x14, 0x1 | i = 1 | i equal 1 |
| 1. mov eax, 0x14 2. cmp eax, 0xc 3. jg 0x99 | For (i =1; I <= rows; i++) { | For loop if i is less than equal to rows |
| 1. lea 0x0(%rip),%rsi 2. lea 0x0(%rip),%rdi 3. call 0x93 4. addl 0x1,-0x14(%rbp) 5. jmp 0x78 | Cout << “ “ << endl;  i++ | Output space  Increment i |
| 1. subl 0x1,-0xc(%rbp) 2. movl 0x1,-0x14(%rbp) 3. mov -0x10(%rbp),%eax 4. add %eax,%eax 5. sub 0x1,%eax 6. cmp %eax,-0x14(%rbp) 7. jg 0xca | rows—  for (i = 1; i <= j + (j – 1); I ++) { | Decrement rows  For loop if i is less than equal to j + j - 1 |
| 1. lea 0x0(%rip),%rsi 2. lea 0x0(%rip),% 3. call 0xc4 4. addl 0x1,-0x14(%rbp) 5. jmp 0xa4 | Cout << “ \* “ << endl;  i++ | Output “ \* “  Increment i |
| 1. lea 0x0(%rip),%rsi 2. lea 0x0(%rip),% 3. call 0xdd 4. addl 0x1,-0x10(%rbp) 5. jmp 0x69 | Cout << “\n” << endl;  j++ | Insert newline  Increment j |
| 1. movl 0x1,-0xc(%rbp) 2. movl 0x1,-0x10(%rbp) 3. mov -0x18(%rbp),%eax 4. sub 0x1,%eax 5. cmp %eax,-0x10(%rbp) 6. jg 0x171 | rows = 1;  for (j = 1; j <= (input – 1); j++) { | Rows equal to 1  For loop j is less than equal to input -1 |
| 1. movl 0x1,-0x14(%rbp) 2. mov -0x14(%rbp),%eax 3. cmp -0xc(%rbp),%eax 4. jg 0x124 | for (i = 1; i <= rows; i++) { | For loop i is less than equal to rows |
| 1. lea 0x0(%rip),% 2. lea 0x0(%rip),% 3. call 0x11e 4. addl 0x1,-0x14(%rbp) 5. jmp 0x103 | Cout << “ “ << endl;  i++; | Output space  Increment i |
| 1. addl 0x1,-0xc(%rbp) 2. movl 0x1,-0x14(%rbp) 3. mov -0x18(%rbp),%eax 4. sub -0x10(%rbp),%eax 5. add %eax,%eax 6. sub 0x1,%eax 7. cmp %eax,-0x14(%rbp) 8. jg 0x158 | Rows++;  i = 1;  while (true) {  rows = input – j;  if (i > (rows + (rows – 1))) {  break | Increment rows  i = 1  while loop is true  rows = input – j  If i is greater than rows plus rows minus 1 break |
| 1. lea 0x0(%rip),% 2. lea 0x0(%rip),% 3. call 0x152 4. addl 0x1,-0x14(%rbp) 5. jmp 0x12f | Else {  Cout << “ \* “ << endl;  i ++; | Else Output “ \* “  Increment i |
| 1. lea 0x0(%rip),% 2. lea 0x0(%rip),% 3. call 0x16b 4. addl 0x1,-0x10(%rbp) 5. jmp 0xf1 | Cout << “\n” << endl;  j++ | Output newline  Increment j |
| 1. mov 0x1,%eax 2. mov -0x8(%rbp),%rcx 3. xor %fs:0x28,%rcx 4. je 0x18a | If (rows \* (input + 1)) {  Return 1  } | If rows times input plus 1 return 1 |
|  |  |  |

## **File Four**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |