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

## **File One**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163>  addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | Assign value 1 to -8(%rbp)  Compare value in -8(%rbp) with 9 Conditional jump <main+163> |
| cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154> | Compare value 1 to 9 in %rbp  Conditional jump <main+ 152> |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp) | Move value of x at -8(%rbp) to %eax Multiply at -8(%rbp) by %eax  Move %eax value to -4(%rbp) |
| mov -0x8(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96>  lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %rdx,%rdi  callq 0x7f <main+127>  mov %rax,%rdx  mov 0x0(%rip),%rax <main+137>  mov %rdx,%rdi  callq 0x94 <main+148> # 0x89 | Move -8(%rbp) to %eax  Move %eax to %esi to print  Callq prints statement x = a \* i  Loop again in which values are moved to %rdx, then %eax, to %esi, to call callq function to print |
| mov $0x0,%eax  leaveq  retq | Return exit program |

**Step 4:** Convert the assembly code to C++ code.  
  
*int main()*

*{*

*int number, i, a, x;*

*for (a = 1; a <= 9; a++)*

*{*

*for (i = 1; i <= 9; i++){*

*x = a \* i;*

*cout << a << " \* " << i << " = " << x << endl;*

*}*

*}*

*return 0;*

*}*

**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** |
| --- | --- | --- |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163>  addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | for (a = 1; a <= 9; a++) | initializes a to 1, checks if a is less than or equal to 9 before each iteration, and increments a by 1 at the end of each iteration. |
| cmpl $0x9,-0xc(%rbp)  jg 0x mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)9a <main+154> | for (i = 1; i <= 9; i++) x = a \* i; | loop initializes i to 1 (from previous context), checks if i is less than or equal to 9 before each iteration, and in each iteration, multiplies a with i to compute x. |
| mov -0x8(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96>  lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %rdx,%rdi  callq 0x7f <main+127>  mov %rax,%rdx  mov 0x0(%rip),%rax <main+137>  mov %rdx,%rdi  callq 0x94 <main+148> # 0x89 | cout << a << " \* " << i << " = " << x << endl; | retrieves values of variables a, i, and x and prepares them for output. Through a series of function calls, it is likely formatting and displaying the string "a times i equals x" to the console. The corresponding C++ line that matches this behavior is the statement that prints a, i, and x using the cout function. |
| mov $0x0,%eax  leaveq  retq | return 0; | preparing to exit the function. |
|  |  |  |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi  callq 0x52 <main+82> | Load -14(%rbp) in %rax  Move %rax in %rsi  Load %rdi  Calls callq |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax | Move -14(%rbp) into %edx  Move -14(%rbp) into %eax  Multiply %eax and %edx  Move -14(%rbp) into %eax |
| imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x8f <main+143> | Multiply %edx,%eax  Move %eax to -14(%rbp)  Move -14(%rbp) in %eax  Convert double %eax to %xmm0 register Move double to %xmm1 register  Mulitply  Move double %xmm0 to -10(%rbp)  Load 0 into %rsi  Load 0 in %rdi  Calls callq |
| mov %rdx,%rdi  callq 0xa7 <main+167> | Move %rdx in %rdi  Call callq |
| leaveq  retg | Return Exit |

**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** |
| --- | --- | --- |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi  callq 0x52 <main+82> | cout << "Enter Radius:\n";  cin >> radius; | prepares for a function call by loading the address of the variable (likely radius) into %rax and then moving it to %rsi. The string "Enter Radius:\n" is likely associated with the %rdi register. The callq instruction is then used to either display the string or get the input for the radius. |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax | int squaredValue = radius \* radius;  radius = squaredValue \* radius; | fetches the value of radius from memory and stores it in both %edx and %eax. It then multiplies these values, effectively squaring radius, and stores the result in %edx. Subsequently, it multiplies the squared value with the original value of radius, updating the value of radius. |
| imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x8f <main+143> mov %rdx,%rdi  callq 0xa7 <main+167> | area = PI \* (radius \* radius); | multiplies the value in %edx (which holds the value of radius) with the value in %eax (also radius), effectively squaring it. It then converts this squared integer value to a floating-point number and multiplies it with a floating-point constant (likely representing PI). The result is stored as area.  Blocks of Assembly Code:  perl  Copy code  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq SOME\_ADDRESS  mov %rdx,%rdi  callq SOME\_OTHER\_ADDRESS  C++ Code:  cpp  Copy code  cout << endl;  Explanation of Functionality:  The assembly code is preparing for and making function calls, which appear to be related to output operations. Given the context, it's likely printing a newline character (endl), which corresponds to the cout << endl; statement in C++.  This is a high-level translation based on the provided assembly. Some specifics, especially regarding exact function addresses and certain constants, are inferred from typical patterns and may not be exact. |
| leaveq  retg | Return 0; | Returns and exits the program. |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | Load -18(%rbp) into %rax  Move %rax into %rsi  Load 0 in %rdi  Calls callq |
| mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227>  movl $0x1,-0x14(%rbp)  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292> | Move -18(%rbp) into %eax  Compares %eax to the -10(%rbp) register Conditional jumps to <main+227>  Move 1 into -14(%rbp)  Add 1 into -10(%rbp)  Move -14(%rbp) into %eax  Compares values in %rbp to %eax Conditional jump to <main+292> |
| mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120>  subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202>  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292> callq 0xc4 <main+196> | Move -14(%rbp) into %eax  Compare %rbp to %eax  Conditional jump to <main+153>  Add 1 to -14(%rbp) register Jumps to <main+120>  Subtract 1 from %rbp  Move 1 into -14(%rbp)  Move -10(%rbp) to %eax  Add %eax to %eax  Sub 1 from %eax  Comapre %eax to -14(%rbp)  Conditional jump to <main+202> Add 1 to -10(%rbp)  Move -14(%rbp) to %eax  Compare %rbp to %eax  Conditional jump to <main+292>  Calls callq |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164>  addl $0x1,-0x10(%rbp) j  mp 0x69 <main+105>  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi callq 0x11e <main+286> | Add 1 to -14(%rbp)  Jumps to <main+164>  Add 1 to -10(%rbp)  Jumps to <main+105>  Compare %eax to -10(%rbp) Conditional jump to <main+369> Load 0 in %rsi and the %rdi  Call callq |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> | Move -18(%rbp) to %eax S  ubtract 1 from %eax  Compare %eax to -10(%rbp)  Conditional jump to <main+369> |

**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** |
| --- | --- | --- |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | cout << "Enter number of rows\n";  cin >> r; | prepares for and executes function calls, likely to display a message prompting the user to input the number of rows and then to read that input into the variable r. |
| mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227>  movl $0x1,-0x14(%rbp)  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120>  subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202>  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292> callq 0xc4 <main+196> | for(i=0;i<=r;i++) {  for(j=1;j<=r-i;j++) cout<<" "; | initializes i to 0 and loops until i is less than or equal to r. Inside the main loop, there's a nested loop that appears to print spaces (r-i times). |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164>  addl $0x1,-0x10(%rbp) j  mp 0x69 <main+105>  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi callq 0x11e <main+286> | for(j=1;j<=2\*i-1;j++) cout<<"\*";  cout<<endl; } | increments the value located at -0x14(%rbp) (likely representing j) and then jumps to a location (0xa4). It then increments the value at -0x10(%rbp) (likely representing i).  The loop condition checks if j is less than or equal to 2 \* i - 1. If this condition is true, the code prints the asterisk symbol \*. Once the inner loop finishes, the code prints a newline character (endl). |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> | for(i=r-1;i>=1;i--) {  for(j=1;j<=r-i;j++) | loop where i starts from r-1 and decrements in each iteration until it reaches 1. Inside this loop, there's another nested loop that will iterate based on the value of i and r. |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp)  callq 0x3a <main+58>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | Move 0 in -20(%rbp)  Move 1 in -18(%rbp)  Calls callq  Move %rax in %rsi  Move %rdx in %rdi  Calls calls |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi | Load -28(%rbp) in %rax  Move %rax in %rsi  Load 0 in %rdi |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | Move -28(%rbp) in %rax  Test %rax in %rax  Jump if equal to <main+242> |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp)  shlq -0x18(%rbp)  mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax | Move -28(%rbp) in %rcx  Move abs value 6666666666666667 in %rdx  Move %rcx in %rax  Multiply %rdx  Shift right 2 in %rx  Move %rcx in %rax  Shift right 3 in %rax  Subtract %rax in %rdx  Move %rdx in %rax  Move %rax in -10(%rbp)  Move -10(%rbp) in %rdx  Move %rdx in %rax  Shift left 2 in %rax  Add %rdx in %rax  Add %rax in %rax  Subtract %rax in %rcx  Move %rcx in %rax  Move %rax in -10(%rbp)  Move -18(%rbp) in %rax  Multiply -18(%rbp) iin %rax  Add %rax in -20(%rbp)  Shift -18(%rbp)  Move -28(%rbp) in %rcx  Move abs value 6666666666666667 in %rdx Move %rcx in %rax  Multiply %rdx  Shift right 2 in %rdx  Move %rcx in %rax |
| mov %rax,%rdx  mov -0x20(%rbp),%rax mov %rax,%rsi  mov %rdx,%rdi | Move %rax in %rdx  Move -20(%rbp) in %rax  Move %rax in %rsi  Move %rdx in %rdi |

**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** |
| --- | --- | --- |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp)  callq 0x3a <main+58>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | cout << "Enter the binary number:\n";  cin >> num; | prepares for output by initializing some memory locations and then makes function calls, likely related to displaying the prompt message and reading the input into the variable num |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi | decimalNum = 0; i = 0; | sequence initializes two variables, decimalNum and i, to zero. |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242>  mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp)  shlq -0x18(%rbp)  mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax | while (num != 0) {  rem = num % 10; num /= 10;  decimalNum += rem \* pow(2, i);  ++i; | Loop that continues as long as num is not zero. Within the loop, the least significant digit of the binary number (num) is extracted (as rem). The binary number (num) is then divided by 10 to shift its digits to the right. Subsequently, the decimal equivalent of the binary number is computed using powers of 2 and accumulated in decimalNum. |
| mov %rax,%rdx  mov -0x20(%rbp),%rax mov %rax,%rsi  mov %rdx,%rdi | **cout << "Equivalent hexadecimal value: " <<** | executes a function call, likely related to displaying the message "Equivalent hexadecimal value:" followed by the value of decimalNum. |