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

## **File One**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 0: push %rbp to 4: sub $0x10,%rsp | This sets up the stack frame for the main() function. It's just setting aside space to work with local variables. |
| 8: movl $0x1,-0x8(%rbp) | Stores the number 1 into a local variable. This will be used as the first number in the outer loop (let’s call it a). |
| f: cmpl $0x9,-0x8(%rbp) to 13: jg a3 | Checks if a > 9. If true, it jumps to the end of the program. Otherwise, it starts the nested loop. |
| 19: movl $0x1,-0xc(%rbp) | Initializes another variable i = 1 for the inner loop. |
| 20: cmpl $0x9,-0xc(%rbp) to 24: jg 9a | Checks if i > 9. If true, it exits the inner loop and increments a. |
| 26: mov -0x8(%rbp),%eax  29: imul -0xc(%rbp),%eax  2d: mov %eax,-0x4(%rbp) | Multiplies a \* i and stores the result in a third local variable (let’s call it x). |
| 30–8f (a series of mov, lea, and callq) | These are preparing and making calls to std::cout functions. Specifically, they print the multiplication expression and result in the format a \* i = x. |
| 94: addl $0x1,-0xc(%rbp)  98: jmp 20 | Adds 1 to i and jumps back to the comparison to keep looping from 1 to 9. |
| 9a: addl $0x1,-0x8(%rbp)  9e: jmp f | Adds 1 to a and jumps back to the outer loop condition. |
| a3: mov $0x0,%eax  a8: leaveq  a9: retq | Ends the function and returns 0, which is normal for main() in C++. |

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

#include <iostream>

using namespace std;

int main() {

int a, i, 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** |
| --- | --- | --- |
| 8: movl $0x1,-0x8(%rbp) | int a = 1; | Initializes outer loop variable a to 1. |
| f: cmpl $0x9,-0x8(%rbp)  13: jg a3 | for (a = 1; a <= 9; a++) | Begins a loop that continues while a is less than or equal to 9. |
| 19: movl $0x1,-0xc(%rbp)  20: cmpl $0x9,-0xc(%rbp)  24: jg 9a | for (i = 1; i <= 9; i++) | Starts an inner loop from 1 to 9 using variable i. |
| 26: mov -0x8(%rbp),%eax  29: imul -0xc(%rbp),%eax  2d: mov %eax,-0x4(%rbp) | x = a \* i; | Multiplies a and i, storing the result in x. |
| All callq blocks between 30 and 8f | cout << a << " \* " << i << " = " << x << endl; | Prints the multiplication result in a readable format. |
| 94: addl $0x1,-0xc(%rbp)  98: jmp 20 | i++ | Increments the inner loop variable. |
| 9a: addl $0x1,-0x8(%rbp)  9e: jmp f | a++ | Increments the outer loop variable. |
| a3: mov $0x0,%eax  a8: leaveq  a9: retq | return 0; | Ends the program and returns 0 from main(). |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 0: push %rbp  1: mov %rsp,%rbp  4: sub $0x30,%rsp | Sets up the stack frame for the main() function and reserves 48 bytes of local space |
| 8: mov %fs:0x28,%rax  11: mov %rax,-0x8(%rbp)  15: xor %eax,%eax | Stack canary setup for security and clears the eax register. |
| 17–25: lea/callq | Loads addresses of format strings and calls a function (likely std::cout) to prompt the user to "Enter a number:". |
| 2a: mov %rax,%rdx  2d–3a: load more strings/call | Likely more setup for formatted output or flushing output before input. |
| 3f: lea -0x14(%rbp),%rax  43: mov %rax,%rsi  46: lea ..., %rdi  4d: callq | Gets address for user input and calls cin or equivalent to store an integer into -0x14(%rbp) (our variable num). |
| 52: mov -0x14(%rbp),%edx  55: mov -0x14(%rbp),%eax  58: imul %eax,%edx  5b: mov -0x14(%rbp),%eax  5e: imul %edx,%eax  61: mov %eax,-0x14(%rbp) | Cubes the input: x \* x \* x, and stores the result back into the same memory space. |
| 64: mov -0x14(%rbp),%eax  67: cvtsi2sd %eax,%xmm0 | Converts the cubed integer into a double (xmm0). |
| 6b–73: movsd / mulsd | Loads a constant double (e.g., 1.23) into xmm1, multiplies it with the double in xmm0. |
| 77: movsd %xmm0,-0x10(%rbp) | Stores the final double result into local variable result. |
| 7c–8a: lea / callq | Prepares and calls cout function again to print "Final result:". |
| 8f: mov %rax,%rdx  92: mov -0x10(%rbp),%rax  96: mov %rax,-0x28(%rbp) | Moves final result into a second temporary variable, likely for formatting or printing. |
| 9a: movsd -0x28(%rbp),%xmm0  9f: mov %rdx,%rdi  a2: callq | Loads the double value into a register and calls another function to print it out (probably std::cout << result). |
| a7: mov $0x0,%eax | Sets return value of main() to 0. |
| ac–b9: stack check / exit | Verifies stack integrity (canary value) and exits using leaveq and retq. |

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

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

num = num \* num \* num;

double result = static\_cast<double>(num) \* 1.23;

cout << "Final result: " << result << 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** |
| --- | --- | --- |
| 17: lea ... %rsi  1e: lea ... %rdi  25: callq | cout << "Enter a number: "; | Loads and prints the message prompting the user to enter a number. These instructions set up the string and call the print function (likely std::cout). |
| 3f: lea -0x14(%rbp),%rax  43: mov %rax,%rsi  46: lea ...,%rdi  4d: callq | cin >> num; | Prepares the address of the variable where user input will be stored and calls a function to read the integer (likely std::cin). |
| 52: mov -0x14(%rbp),%edx  55: mov -0x14(%rbp),%eax  58: imul %eax,%edx  5b: mov -0x14(%rbp),%eax  5e: imul %edx,%eax  61: mov %eax,-0x14(%rbp) | num = num \* num \* num; | Loads the user’s input into registers and performs two imul operations to cube the value (num^3). Then stores the result back into the same local variable. |
| 64: mov -0x14(%rbp),%eax  67: cvtsi2sd %eax,%xmm0  6b: movsd ..., %xmm1  73: mulsd %xmm1,%xmm0  77: movsd %xmm0,-0x10(%rbp) | result = static\_cast<double>(num) \* multiplier; | Converts the cubed integer value into a double, loads a floating-point constant (like 1.23), multiplies them, and stores the result. |
| 7c: lea ..., %rsi  83: lea ..., %rdi  8a: callq  8f: mov %rax,%rdx  92: mov -0x10(%rbp),%rax  96: mov %rax,-0x28(%rbp)  9a: movsd -0x28(%rbp),%xmm0  9f: mov %rdx,%rdi  a2: callq | cout << "Final result: " << result << endl; | Sets up output formatting for the floating-point result and calls the function to print the final value. The value is moved into xmm0 for display using the floating-point print function. |

**File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functioty** |
| --- | --- |
| 0: push %rbp to 15: xor %eax,%eax | Standard function setup: prepares the stack frame and clears registers. |
| 17: movl $0x1,-0xc(%rbp) | Initializes loop control variable at -0xc(%rbp) (likely limit = 1). |
| 1e: lea ... %rsi  25: lea ... %rdi  2c: callq | Loads and displays the prompt message (e.g., "Enter a number:"). |
| 46: lea -0x18(%rbp),%rax  4a: mov %rax,%rsi  4d: lea ..., %rdi  54: callq | Prepares to receive user input and stores the input integer at -0x18(%rbp). |
| 59: mov -0x18(%rbp),%eax  5c: sub $0x1,%eax  5f: mov %eax,-0xc(%rbp) | Subtracts 1 from the user input and stores it at -0xc(%rbp) — likely for controlling range in a loop. |
| 62: movl $0x1,-0x10(%rbp) | Initializes outer loop variable i = 1 at -0x10(%rbp). |
| 69: mov -0x18(%rbp),%eax  6c: cmp %eax,-0x10(%rbp)  6f: jg e3 | Compares loop variable i to the input. If i > input, jump to cleanup. Otherwise, enter the first loop body. |
| 71: movl $0x1,-0x14(%rbp) | Inner loop variable j = 1. |
| 78: mov -0x14(%rbp),%eax  7b: cmp -0xc(%rbp),%eax  7e: jg 99 | Compares j to limit (stored at -0xc(%rbp)); if j > limit, jump to increment. |
| 80–8e: lea / callq | Likely prints one character |
| 93: addl $0x1,-0x14(%rbp)  97: jmp 78 | Increments j and loops again if j <= limit. |
| 99: subl $0x1,-0xc(%rbp) | Decreases limit by 1. |
| 9d: movl $0x1,-0x14(%rbp) | Resets j = 1. |
| a4: mov -0x10(%rbp),%eax  a7: add %eax,%eax  a9: sub $0x1,%eax  ac: cmp %eax,-0x14(%rbp) | Calculates value. Stores it in eax. |
| af: jg ca | Loop: if j > (2\*i - 1), break; else continue printing. |
| b1: lea ..., %rsi b8: lea ..., %rdi bf: callq | Prints second set of characters |
| c4: addl $0x1,-0x14(%rbp) c8: jmp a4 | Increments j and continues loop. |
| ca: lea ..., %rsi d1: lea ..., %rdi d8: callq | Prints a newline or separator after a row |
| dd: addl $0x1,-0x10(%rbp) e1: jmp 69 | Increments i and loops to the next row. |
| e3: movl $0x1,-0xc(%rbp)  ea: movl $0x1,-0x10(%rbp) | Starts second pattern: resets limit = 1 and i = 1. |
| f1: mov -0x18(%rbp),%eax f4: sub $0x1,%eax f7: cmp %eax,-0x10(%rbp) fa: jg 171 | Checks if i <= input - 1; if not, skip second pattern. |
| fc: movl $0x1,-0x14(%rbp) | Sets j = 1 for inner loop of second pattern. |
| 103: mov -0x14(%rbp),%eax 106: cmp -0xc(%rbp),%eax 109: jg 124 | Loop while j <= limit to print spacing or pattern start. |
| 10b: lea ..., %rsi 112: lea ..., %rdi 119: callq | Prints character during second pattern setup. |
| 11e: addl $0x1,-0x14(%rbp) 122: jmp 103 | Increments j and continues loop. |
| 124: addl $0x1,-0xc(%rbp) 128: movl $0x1,-0x14(%rbp) | Increases limit and resets j = 1. |
| 12f: mov -0x18(%rbp),%eax 132: sub -0x10(%rbp),%eax 135: add %eax,%eax 137: sub $0x1,%eax | Calculates 2 \* (input - i) - 1. |
| 13a: cmp %eax,-0x14(%rbp) 13d: jg 158 | Loops while j <= result. |
| 13f: lea ..., %rsi 146: lea ..., %rdi 14d: callq | Prints characters for second pattern body. |
| 152: addl $0x1,-0x14(%rbp) 156: jmp 12f | Increments j, continues loop. |
| 158: lea ..., %rsi 15f: lea ..., %rdi 166: callq | Prints a newline after each row of the second pattern. |
| 16b: addl $0x1,-0x10(%rbp) 16f: jmp f1 | Increments i, begins next row of the second pattern. |
| 171: mov $0x1,%eax 176: mov -0x8(%rbp),%rcx 17a: xor %fs:0x28,%rcx 183: je 18a 185: callq 18a: leaveq 18b: retq | Exits cleanly by returning 1, validating the canary, and leaving main(). |

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

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

int limit = num - 1;

int i = 1;

while (i <= num) {

int j = 1;

while (j <= limit) {

cout << "\*";

j++;

}

limit--;

j = 1;

int range = 2 \* i - 1;

while (j <= range) {

cout << "\*";

j++;

}

cout << endl;

i++;

}

limit = 1;

i = 1;

while (i <= num - 1) {

int j = 1;

while (j <= limit) {

cout << "\*";

j++;

}

limit++;

j = 1;

int range = 2 \* (num - i) - 1;

while (j <= range) {

cout << "\*";

j++;

}

cout << endl;

i++;

}

return 1;

}

**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** |
| --- | --- | --- |
| 17: movl $0x1,-0xc(%rbp) | int limit = 1; | Initializes limit for controlling spaces in the first triangle. |
| 1e: lea ... %rsi 25: lea ... %rdi 2c: callq | cout << "Enter a number: "; | Prints the prompt message to the user. |
| 46: lea -0x18(%rbp),%rax 4a: mov %rax,%rsi 4d: lea ... %rdi 54: callq | cin >> num; | Reads the user's input number into variable num. |
| 59: mov -0x18(%rbp),%eax 5c: sub $0x1,%eax 5f: mov %eax,-0xc(%rbp) | limit = num - 1; | Adjusts limit to one less than user input for formatting. |
| 62: movl $0x1,-0x10(%rbp) | int i = 1; | Initializes outer loop index i. |
| 69: mov -0x18(%rbp),%eax 6c: cmp %eax,-0x10(%rbp) 6f: jg e3 | while (i <= num) | Starts outer loop that builds the top triangle. |
| 71: movl $0x1,-0x14(%rbp) | int j = 1; | Inner loop index initialized for spacing. |
| 78: mov -0x14(%rbp),%eax 7b: cmp -0xc(%rbp),%eax 7e: jg 99 | while (j <= limit) | Loop for printing leading stars/spaces. |
| 80: lea ... %rsi 87: lea ... %rdi 8e: callq | cout << "\*"; | Prints an asterisk (or a space if used for alignment). |
| 93: addl $0x1,-0x14(%rbp) 97: jmp 78 | j++; | Increments inner loop counter. |
| 99: subl $0x1,-0xc(%rbp) 9d: movl $0x1,-0x14(%rbp) | limit--; j = 1; | Decreases spacing limit and resets j. |
| a4: mov -0x10(%rbp),%eax a7: add %eax,%eax a9: sub $0x1,%eax | int range = 2 \* i - 1; | Computes how many characters in the row. |
| ac: cmp %eax,-0x14(%rbp) af: jg ca | while (j <= range) | Loop for printing centered stars. |
| b1: lea ... %rsi b8: lea ... %rdi bf: callq | cout << "\*"; | Prints one symbol in the center part of the row. |
| c4: addl $0x1,-0x14(%rbp) c8: jmp a4 | j++; | Increments j for the center pattern. |
| ca: lea ... %rsi d1: lea ... %rdi d8: callq | cout << endl; | Ends the current row with a newline. |
| dd: addl $0x1,-0x10(%rbp) e1: jmp 69 | i++; | |  | | --- | |  |  |  | | --- | | Increments outer loop index. | |
| e3: movl $0x1,-0xc(%rbp)  ea: movl $0x1,-0x10(%rbp) | limit = 1; i = 1; | Resets loop variables for bottom triangle. |
| f1: mov -0x18(%rbp),%eax f4: sub $0x1,%eax f7: cmp %eax,-0x10(%rbp) fa: jg 171 | while (i <= num - 1) | Begins outer loop for the bottom triangle. |
| fc: movl $0x1,-0x14(%rbp) | j = 1; | Inner loop index reset. |
| 103: mov -0x14(%rbp),%eax 106: cmp -0xc(%rbp),%eax 109: jg 124 | while (j <= limit) | Controls initial leading characters of each row. |
| 10b: lea ... %rsi 112: lea ... %rdi 119: callq | cout << "\*"; | Prints leading characters in bottom half. |
| 11e: addl $0x1,-0x14(%rbp) 122: jmp 103 | j++; | Increments j. |
| 124: addl $0x1,-0xc(%rbp) 128: movl $0x1,-0x14(%rbp) | limit++; j = 1; | Increases spacing and resets j. |
| 12f: mov -0x18(%rbp),%eax 132: sub -0x10(%rbp),%eax 135: add %eax,%eax 137: sub $0x1,%eax | int range = 2 \* (num - i) - 1; | Calculates number of symbols in the row |
| 13a: cmp %eax,-0x14(%rbp) 13d: jg 158 | while (j <= range) | Loop for bottom triangle character printing. |
| 13f: lea ... %rsi 146: lea ... %rdi 14d: callq | cout << "\*"; | Prints symbol in bottom row. |
| 152: addl $0x1,-0x14(%rbp) 156: jmp 12f | j++; | Increments inner loop variable. |
| 158: lea ... %rsi 15f: lea ... %rdi 166: callq | cout << endl; | Ends the row with a newline. |
| 16b: addl $0x1,-0x10(%rbp) 16f: jmp f1 | i++; | Increments outer loop index. |
| 171: mov $0x1,%eax 176: mov -0x8(%rbp),%rcx 17a: xor %fs:0x28,%rcx 183: je 18a 185: callq 18a: leaveq 18b: retq | Return 1; | Exits cleanly by returning 1 |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 17: movq $0x0,-0x20(%rbp) 1f: movq $0x1,-0x18(%rbp) | Initializes two variables: total = 0 and multiplier = 1. |
| 27: lea ..., %rsi 2e: lea ..., %rdi 35: callq ... | Displays a message prompting the user to enter a number. |
| 4f: lea -0x28(%rbp), %rax 53: mov %rax, %rsi 56: lea ..., %rdi 5d: callq ... | Calls input function to store the number at -0x28(%rbp). |
| 62: mov -0x28(%rbp), %rax 66: test %rax, %rax 69: je f2 | If user input is 0, skip the loop and go to final output. |
| 6f: mov -0x28(%rbp), %rcx 73: movabs ..., %rdx 7d: mov %rcx, %rax 80: imul %rdx 83: sar $0x2, %rdx 87: mov %rcx, %rax 8a: sar $0x3f, %rax 8e: sub %rax, %rdx 91: mov %rdx, %rax 94: mov %rax, -0x10(%rbp) | Divides input by 10 using optimized arithmetic; stores result in quotient (-0x10(%rbp)). |
| 98: mov -0x10(%rbp), %rdx 9c: mov %rdx, %rax 9f: shl $0x2, %rax a3: add %rdx, %rax a6: add %rax, %rax a9: sub %rax, %rcx ac: mov %rcx, %rax af: mov %rax, -0x10(%rbp) | Calculates input % 10 using input and stores it in -0x10(%rbp). |
| b3: mov -0x10(%rbp), %rax b7: imul -0x18(%rbp), %rax bc: add %rax, -0x20(%rbp) | Multiplies the current digit by the multiplier and adds it to the running total. |
| c0: shlq -0x18(%rbp) | Multiplies the multiplier by 2 |
| c4: mov -0x28(%rbp), %rcx c8: movabs ..., %rdx d2: mov %rcx, %rax d5: imul %rdx d8: sar $0x2, %rdx dc: mov %rcx, %rax df: sar $0x3f, %rax e3: sub %rax, %rdx e6: mov %rdx, %rax e9: mov %rax, -0x28(%rbp) | Updates input: divides input by 10 to prepare for the next digit extraction. |
| ed: jmpq 62 | Loops back to check if the new input is zero. |
| f2: lea ..., %rsi f9: lea ..., %rdi 100: callq ... | Prepares to print the final total message. |
| 108: mov -0x20(%rbp), %rax 10c: mov %rax, %rsi 10f: mov %rdx, %rdi 112: callq ... | Sends the final result (total) to output. |
| 117: mov %rax, %rdx 11a: mov ..., %rax 121: mov %rax, %rsi 124: mov %rdx, %rdi 127: callq ... | Final formatting/cleanup for output |
| 12c: mov $0x0, %eax | Sets the return value to 0. |
| 131–146 | Checks stack canary and exits the function cleanly. |

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

#include <iostream>

using namespace std;

int main() {

long total = 0;

long multiplier = 1;

long input;

cout << "Enter a number: ";

cin >> input;

while (input != 0) {

long quotient = input / 10;

long remainder = input - (quotient \* 10);

total += remainder \* multiplier;

multiplier \*= 2;

input = quotient;

}

cout << "Total: " << total << 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** |
| --- | --- | --- |
| 17: movq $0x0,-0x20(%rbp) 1f: movq $0x1,-0x18(%rbp) | long total = 0; long multiplier = 1; | Initializes total to accumulate the result and multiplier to weight each digit. |
| 27: lea ..., %rsi 2e: lea ..., %rdi 35: callq ... | cout << "Enter a number: "; | Displays a prompt message to request user input. |
| 4f: lea -0x28(%rbp), %rax 53: mov %rax, %rsi 56: lea ..., %rdi 5d: callq ... | cin >> input; | Takes user input and stores it in variable input. |
| 62: mov -0x28(%rbp), %rax 66: test %rax, %rax 69: je f2 | while (input != 0) | Begins loop that processes each digit until the input becomes 0. |
| 6f: mov -0x28(%rbp), %rcx 73: movabs ..., %rdx 7d: mov %rcx, %rax 80: imul %rdx 83: sar $0x2, %rdx 87: mov %rcx, %rax 8a: sar $0x3f, %rax 8e: sub %rax, %rdx 91: mov %rdx, %rax 94: mov %rax, -0x10(%rbp) | long quotient = input / 10; | Calculates the quotient using optimized division and stores it. |
| 98: mov -0x10(%rbp), %rdx 9c: mov %rdx, %rax 9f: shl $0x2, %rax a3: add %rdx, %rax a6: add %rax, %rax a9: sub %rax, %rcx ac: mov %rcx, %rax af: mov %rax, -0x10(%rbp) | long remainder = input - (quotient \* 10); | Computes the remainder (modulo 10) and stores it. |
| b3: mov -0x10(%rbp), %rax b7: imul -0x18(%rbp), %rax bc: add %rax, -0x20(%rbp) | total += remainder \* multiplier; | Adds the weighted digit to the total. |
| c0: shlq -0x18(%rbp) | multiplier \*= 2; | Doubles the multiplier for the next digit. |
| c4: mov -0x28(%rbp), %rcx c8: movabs ..., %rdx d2: mov %rcx, %rax d5: imul %rdx d8: sar $0x2, %rdx dc: mov %rcx, %rax df: sar $0x3f, %rax e3: sub %rax, %rdx e6: mov %rdx, %rax e9: mov %rax, -0x28(%rbp) | input /= 10; | Updates the input to remove the last digit. |
| ed: jmpq 62 | } | |  | | --- | |  |  |  | | --- | | Jumps back to start of the loop. | |
| f2: lea ..., %rsi f9: lea ..., %rdi 100: callq ... | cout << "Total: "; | Prepares to print the final result. |
| 108: mov -0x20(%rbp), %rax 10c: mov %rax, %rsi 10f: mov %rdx, %rdi 112: callq ... | cout << total; | Prints the total to the console. |
| 117: mov %rax, %rdx 11a: mov ..., %rax 121: mov %rax, %rsi 124: mov %rdx, %rdi 127: callq ... | cout << endl; | |  | | --- | |  |  |  | | --- | | Finishes the line. | |
| 12c: mov $0x0, %eax | return 0; | Exits main() successfully. |