# **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** |
| 0000000000000000 <main>: 0: 55 push %rbp 1: 48 89 e5 mov %rsp,%rbp 4: 48 83 ec 10 sub $0x10,%rsp | Function prologue: Saves the base pointer (%rbp) to the stack, sets up a new stack frame by copying the stack pointer (%rsp) to %rbp, and reserves 16 bytes (0x10) for local variables (a at -0x8, i at -0xc, and x at -0x4). |
| 8: c7 45 f8 01 00 00 00 movl $0x1,-0x8(%rbp) f: 83 7d f8 09 cmpl $0x9,-0x8(%rbp) 13: 0f 8f 8a 00 00 00 jg a3 <main+0xa3> | Outer loop initialization: Initializes a to 1 (stored at -0x8(%rbp)), compares a with 9, and jumps to the end of the loop if a > 9. |
| 19: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp) 20: 83 7d f4 09 cmpl $0x9,-0xc(%rbp) 24: 7f 74 jg 9a <main+0x9a> | Inner loop initialization: Initializes i to 1 (stored at -0xc(%rbp)), compares i with 9, and jumps to increment the outer loop if i > 9. |
| 26: 8b 45 f8 mov -0x8(%rbp),%eax 29: 0f af 45 f4 imul -0xc(%rbp),%eax 2d: 89 45 fc mov %eax,-0x4(%rbp) | Multiplication: Loads a into %eax, multiplies it by i, and stores the result in x (at -0x4(%rbp)). |
| 30: 8b 45 f8 mov -0x8(%rbp),%eax 33: 89 c6 mov %eax,%esi 35: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 3c: e8 00 00 00 00 callq 41 <main+0x41> | Begins output: Loads a into %eax for printing, prepares cout parameters. |
| 41: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 48: 48 89 c7 mov %rax,%rdi 4b: e8 00 00 00 00 callq 50 <main+0x50> | Prints " \* ": Loads the string address for " \* ", prepares cout parameters, and calls the string output operator. |
| 50: 48 89 c2 mov %rax,%rdx 53: 8b 45 f4 mov -0xc(%rbp),%eax 56: 89 c6 mov %eax,%esi 58: 48 89 d7 mov %rdx,%rdi 5b: e8 00 00 00 00 callq 60 <main+0x60> | Prints i: Loads i into %eax for printing, prepares parameters, and outputs the value. |
| 60: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 67: 48 89 c7 mov %rax,%rdi 6a: e8 00 00 00 00 callq 6f <main+0x6f> | Prints " = ": Loads the string address for " = ", prepares cout parameters. |
| 6f: 48 89 c2 mov %rax,%rdx 72: 8b 45 fc mov -0x4(%rbp),%eax 75: 89 c6 mov %eax,%esi 77: 48 89 d7 mov %rdx,%rdi 7a: e8 00 00 00 00 callq 7f <main+0x7f> | Prints x: Loads the multiplication result (x) into %eax, prepares for output, and calls the output operator. |
| 7f: 48 89 c2 mov %rax,%rdx 82: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax 89: 48 89 c6 mov %rax,%esi 8c: 48 89 d7 mov %rdx,%rdi 8f: e8 00 00 00 00 callq 94 <main+0x94> | Prints a newline: Sets up endl for printing, prepares parameters, and calls the newline output function. |
| 94: 83 45 f4 01 addl $0x1,-0xc(%rbp) 98: eb 86 jmp 20 <main+0x20> | Inner loop increment: Increments i and jumps back to check the inner loop condition. |
| 9a: 83 45 f8 01 addl $0x1,-0x8(%rbp) 9e: e9 6c ff ff ff jmpq f <main+0xf> | Outer loop increment: Increments a and jumps back to check the outer loop condition. |
| a3: b8 00 00 00 00 mov $0x0,%eax a8: c9 leaveq a9: c3 retq | Function cleanup: Sets the return value to 0, restores the stack frame, and returns from the function. |

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

#include <iostream>

using namespace std;

int main() {

int a, i, x; // Variables `a`, `i`, `x` correspond to memory locations -0x8(%rbp), -0xc(%rbp), and -0x4(%rbp).

for (a = 1; a <= 9; a++) { // Outer loop

for (i = 1; i <= 9; i++) { // Inner loop

x = a \* i; // Multiplication

cout << a << " \* " << i << " = " << x << endl; // Print result

}

}

return 0; // Return from function

}

**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** |
| Function setup (prologue) | int main() { | Assembly sets up the stack manually, while C++ relies on the compiler to allocate variables. |
| Outer loop initialization | for (a = 1; a <= 9; a++) | Assembly stores 1 in a and checks the condition using cmpl. |
| Inner loop initialization | for (i = 1; i <= 9; i++) | Assembly stores 1 in i and checks the condition using cmpl. |
| Multiplication | x = a \* i; | Assembly uses imul to multiply a and i. |
| Printing | cout << a << " \* " << i << " = " << x << endl; | Assembly prepares cout parameters and calls output operators for each part of the message. |
| Loop control | } (end of loops) | Assembly increments loop variables (addl) and uses jumps to return to the condition check. |
| Return from function | return 0; } | Assembly sets the return value and restores the stack manually. |

**File Two**

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

|  |  |
| --- | --- |
| **Blocks of Assembly Code** | **Explanation of Functionality** |
| **0000000000000000 <main>: 0: 55 push %rbp 1: 48 89 e5 mov %rsp,%rbp 4: 48 83 ec 30 sub $0x30,%rsp** | **Function prologue: Saves the base pointer (%rbp) on the stack, sets up a new stack frame, and reserves 48 bytes of local storage.** |
| **8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax 11: 48 89 45 f8 mov %rax,-0x8(%rbp)** | **Sets up stack canary to protect against stack smashing.** |
| **17: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 1e: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 25: e8 00 00 00 00 callq 2a <main+0x2a>** | **Prints "Enter Radius:" prompt by loading the string address and calling the output function.** |
| **3f: 48 8d 45 ec lea -0x14(%rbp),%rax 46: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 4d: e8 00 00 00 00 callq 52 <main+0x52>** | **Reads the radius from the user and stores it at -0x14(%rbp).** |
| **52: 8b 55 ec mov -0x14(%rbp),%edx 55: 8b 45 ec mov -0x14(%rbp),%eax 58: 0f af d0 imul %eax,%edx** | **Calculates radius^2 by loading the radius and multiplying it by itself.** |
| **5b: 8b 45 ec mov -0x14(%rbp),%eax 5e: 0f af c2 imul %edx,%eax 61: 89 45 ec mov %eax,-0x14(%rbp)** | **Calculates radius^3 and stores it back into -0x14(%rbp).** |
| **67: f2 0f 10 0d 00 00 00 movsd 0x0(%rip),%xmm1 6b: f2 0f 59 c1 mulsd %xmm1,%xmm0** | **Multiplies radius^3 by (4/3) \* π to calculate the volume.** |
| **7c: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 8a: e8 00 00 00 00 callq 8f <main+0x8f>** | **Prints "The volume is:".** |
| **8f: f2 0f 10 45 f0 movsd -0x10(%rbp),%xmm0 a2: e8 00 00 00 00 callq a7 <main+0xa7>** | **Prints the calculated volume.** |
| **b9: 74 05 je c0 <main+0xc0>** | **Verifies the stack canary to ensure no corruption occurred.** |
| **c0: c9 leaveq c1: c3 retq** | **Function cleanup and return** |

**Step 4:** Convert the assembly code to C++ code.  
  
#include <iostream>

#include <cmath> // For M\_PI

using namespace std;

int main() {

double radius, volume;

cout << "Enter Radius:" << endl;

cin >> radius;

// Calculate volume: (4/3) \* π \* radius^3

volume = (4.0 / 3.0) \* M\_PI \* pow(radius, 3);

cout << "The volume is: " << volume << 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** |
| Function setup (prologue) | int main() { | Assembly manually sets up the stack, while C++ relies on the compiler. |
| Stack canary setup | // Compiler handles stack protection implicitly | Assembly explicitly sets up stack canaries, C++ does this automatically. |
| User prompt | cout << "Enter Radius:" << endl; | Assembly loads the string address for "Enter Radius:" and calls the output function, while C++ uses cout. |
| Read radius | cin >> radius; | Assembly loads the address for radius and reads user input into that address. |
| Volume calculation | volume = (4.0 / 3.0) \* M\_PI \* pow(radius, 3); | Assembly calculates radius^3 and multiplies by (4/3) \* π. |
| Print volume | cout << "The volume is: " << volume << endl; | Assembly outputs the calculated volume in a similar manner. |
| Function cleanup | return 0; } | Assembly verifies the stack canary, restores the stack, and returns, while C++ handles this through compiler-generated code. |

**File Three**

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

|  |  |
| --- | --- |
| **Blocks of Assembly Code** | **Explanation of Functionality** |
| 0000000000000000 <main>: 0: 55 push %rbp 1: 48 89 e5 mov %rsp,%rbp 4: 48 83 ec 20 sub $0x20,%rsp | Function prologue: Saves the base pointer (%rbp) on the stack, sets up a new stack frame, and reserves 32 bytes for local variables. |
| 8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax 11: 48 89 45 f8 mov %rax,-0x8(%rbp) | Stack canary setup for protection. |
| 17: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp) | Initializes row to 1 (stored at -0xc(%rbp)) to begin the pattern generation loop. |
| 1e: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 25: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 2c: e8 00 00 00 00 callq 31 <main+0x31> | Prints a user prompt to enter the number of rows. |
| 46: 48 8d 45 e8 lea -0x18(%rbp),%rax 4d: e8 00 00 00 00 callq 52 <main+0x52> | Reads the number of rows from the user and stores it at -0x18(%rbp). |
| 59: 8b 45 e8 mov -0x18(%rbp),%eax 5c: 83 e8 01 sub $0x1,%eax | Calculates the upper limit for the first loop by subtracting 1 from the entered number of rows. |
| 62: c7 45 f0 01 00 00 00 movl $0x1,-0x10(%rbp) 69: 8b 45 e8 mov -0x18(%rbp),%eax 6c: 39 45 f0 cmp %eax,-0x10(%rbp) 6f: 7f 72 jg e3 <main+0xe3> | Outer loop: Initializes and compares the current row number (-0x10(%rbp)) with the total number of rows (-0x18(%rbp)). Jumps out of the loop if the condition is met. |
| 71: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp) | Initializes the inner loop counter for the first pattern. |
| 78: 8b 45 ec mov -0x14(%rbp),%eax 7b: 3b 45 f4 cmp -0xc(%rbp),%eax 7e: 7f 19 jg 99 <main+0x99> | Inner loop: Compares the inner loop counter with the current row number and exits if the condition is met. |
| 80: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 8e: e8 00 00 00 00 callq 93 <main+0x93> | Prints a star (\*) for the current row in the first pattern. |
| 93: 83 45 ec 01 addl $0x1,-0x14(%rbp) 97: eb df jmp 78 <main+0x78> | Increments the inner loop counter and jumps back to check the condition. |
| 99: 83 6d f4 01 subl $0x1,-0xc(%rbp) 9d: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp) | Decreases the pattern spacing and resets the inner loop counter. |
| ca: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi d8: e8 00 00 00 00 callq dd <main+0xdd> | Prints a newline after finishing the stars for the current row. |
| dd: 83 45 f0 01 addl $0x1,-0x10(%rbp) e1: eb 86 jmp 69 <main+0x69> | Increments the row counter and jumps back to check the outer loop condition. |
| f1: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp) ea: c7 45 f0 01 00 00 00 movl $0x1,-0x10(%rbp) | Resets the row and spacing counters for the next part of the pattern. |
| 176: 48 8b 4d f8 mov -0x8(%rbp),%rcx 183: 74 05 je 18a <main+0x18a> | Verifies the stack canary before exiting. |
| 18a: c9 leaveq 18b: c3 retq | Cleans up and returns from the function. |

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

#include <iostream>

using namespace std;

int main() {

int rows;

cout << "Enter the number of rows: " << endl;

cin >> rows;

// First pattern

for (int i = 1; i <= rows; i++) {

for (int j = 1; j <= i; j++) {

cout << "\*";

}

cout << 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** |
| Function setup | int main() { | Sets up the function and reserves space for local variables. |
| Prompt for input | cout << "Enter the number of rows: " << endl; | Outputs the prompt to the user. |
| Read input | cin >> rows; | Reads the number of rows from the user. |
| Outer loop | for (int i = 1; i <= rows; i++) | Initializes and iterates through each row for the pattern. |
| Inner loop | for (int j = 1; j <= i; j++) | Outputs stars for the current row. |
| Print newline | cout << endl; | Adds a newline after each row of stars. |
| Function return | return 0; } | Cleans up and exits the function. |

**File Four**

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

|  |  |
| --- | --- |
| **Blocks of Assembly Code** | **Explanation of Functionality** |
| 0000000000000000 <main>: 0: 55 push %rbp 1: 48 89 e5 mov %rsp,%rbp 4: 48 83 ec 30 sub $0x30,%rsp | Function prologue: Saves the base pointer (%rbp) on the stack, sets up a new stack frame, and reserves 48 bytes for local variables. |
| 8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax 11: 48 89 45 f8 mov %rax,-0x8(%rbp) | Stack canary setup for stack protection. |
| 17: 48 c7 45 e0 00 00 00 movq $0x0,-0x20(%rbp) | Initializes result to 0 (stored at -0x20(%rbp)) for binary-to-hex conversion. |
| 1f: 48 c7 45 e8 01 00 00 movq $0x1,-0x18(%rbp) | Initializes the multiplier to 1 (stored at -0x18(%rbp)) for position tracking. |
| 27: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi 2e: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 35: e8 00 00 00 00 callq 3a <main+0x3a> | Prints a prompt for binary input. |
| 4f: 48 8d 45 d8 lea -0x28(%rbp),%rax 56: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 5d: e8 00 00 00 00 callq 62 <main+0x62> | Reads a binary number from the user and stores it at -0x28(%rbp). |
| 62: 48 8b 45 d8 mov -0x28(%rbp),%rax 66: 48 85 c0 test %rax,%rax 69: 0f 84 83 00 00 00 je f2 <main+0xf2> | Checks if the input binary number is zero; if so, jumps to the end of the conversion. |
| 6f: 48 8b 4d d8 mov -0x28(%rbp),%rcx 80: 48 f7 ea imul %rdx | Processes the binary number one digit at a time. |
| b3: 48 8b 45 f0 mov -0x10(%rbp),%rax b7: 48 0f af 45 e8 imul -0x18(%rbp),%rax bc: 48 01 45 e0 add %rax,-0x20(%rbp) | Updates the result by adding the converted digit multiplied by the current positional multiplier. |
| c4: 48 d1 65 e8 shlq -0x18(%rbp) | Doubles the multiplier for the next binary digit. |
| e9: e9 70 ff ff ff jmpq 62 <main+0x62> | Jumps back to process the next binary digit. |
| f2: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi f9: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi 100: e8 00 00 00 00 callq 105 <main+0x105> | Prints the result message before displaying the hexadecimal number. |
| 105: 48 89 c2 mov %rax,%rdx 108: 48 8b 45 e0 mov -0x20(%rbp),%rax 10c: e8 00 00 00 00 callq 112 <main+0x112> | Outputs the final hexadecimal result. |
| 117: 48 89 c2 mov %rax,%rdx 121: 48 89 d7 mov %rdx,%rdi 127: e8 00 00 00 00 callq 12c <main+0x12c> | Prints a newline after the result. |
| 131: 48 8b 75 f8 mov -0x8(%rbp),%rsi 135: 64 48 33 34 25 28 00 xor %fs:0x28,%rsi 13e: 74 05 je 145 <main+0x145> | Verifies the stack canary to ensure no corruption occurred. |
| 145: c9 leaveq 146: c3 retq | Cleans up the stack and returns from the function. |

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

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

int main() {

string binary;

int result = 0, multiplier = 1;

cout << "Enter a binary number: ";

cin >> binary;

for (int i = binary.length() - 1; i >= 0; i--) {

if (binary[i] == '1') {

result += multiplier;

}

multiplier \*= 2;

}

cout << "Hexadecimal: " << hex << 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** |
| Function setup | int main() { | Sets up the function and reserves space for local variables. |
| Input prompt | cout << "Enter a binary number: "; | Outputs the prompt to the user. |
| Read input | cin >> binary; | Reads a binary string from the user. |
| Conversion loop | for (int i = binary.length() - 1; i >= 0; i--) { ... } | Iterates through each digit of the binary number to convert it to decimal. |
| Update result | result += multiplier; | Adds the positional value to the result if the current binary digit is 1. |
| Update multiplier | multiplier \*= 2; | Doubles the multiplier for the next positional value. |
| Print result | cout << "Hexadecimal: " << hex << result << endl; | Outputs the result in hexadecimal format. |
| Function return | return 0; } | Cleans up and exits the function. |