# **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** |
| --- | --- |
| 0x000000000000090a <+0>: push %rbp  0x000000000000090b <+1>: mov %rsp,%rbp  0x000000000000090e <+4>: sub $0x10,%rsp | sets up main function and allocates for local variables |
| 0x0000000000000912 <+8>: movl $0x1,-0x8(%rbp) | Initializes outer loop variable to 1 |
| 0x0000000000000919 <+15>: cmpl $0x9,-0x8(%rbp)  0x000000000000091d <+19>: jg 0x9ad <main+163> | Loop statement. Compares variable from above to 9 and jumps to end if variable is greater than 9 |
| 0x0000000000000923 <+25>: movl $0x1,-0xc(%rbp) | Initialzes inner loop variable to 1 |
| 0x000000000000092a <+32>: cmpl $0x9,-0xc(%rbp)  0x000000000000092e <+36>: jg 0x9a4 <main+154> | Compares variable from above to 9 and jumps to increment for outer loop if variable is greater than 9 |
| 0x0000000000000930 <+38>: mov -0x8(%rbp),%eax  0x0000000000000933 <+41>: imul -0xc(%rbp),%eax  0x0000000000000937 <+45>: mov %eax,-0x4(%rbp) | Body of the loop  First the iterators are multiplied together then stored in -4(%rbp) |
| 0x000000000000093a <+48>: mov -0x8(%rbp),%eax  0x000000000000093d <+51>: mov %eax,%esi  0x000000000000093f <+53>: lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000946 <+60>: callq 0x7e0 <\_ZNSolsEi@plt>  0x000000000000094b <+65>: lea 0x153(%rip),%rsi # 0xaa5  0x0000000000000952 <+72>: mov %rax,%rdi  0x0000000000000955 <+75>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Body of the loop  Then through line 143 basically uses the value of the loops iterators and outputs the outer iterator to the terminal, then a “ \* “ then the inner iterator, then a “ = “ then the value from -4(%rbp) (the iterators multiplied together) |
| 0x000000000000095a <+80>: mov %rax,%rdx  0x000000000000095d <+83>: mov -0xc(%rbp),%eax  0x0000000000000960 <+86>: mov %eax,%esi  0x0000000000000962 <+88>: mov %rdx,%rdi  0x0000000000000965 <+91>: callq 0x7e0 <\_ZNSolsEi@plt>  0x000000000000096a <+96>: lea 0x138(%rip),%rsi # 0xaa9  0x0000000000000971 <+103>: mov %rax,%rdi  0x0000000000000974 <+106>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000979 <+111>: mov %rax,%rdx  0x000000000000097c <+114>: mov -0x4(%rbp),%eax  0x000000000000097f <+117>: mov %eax,%esi  0x0000000000000981 <+119>: mov %rdx,%rdi  0x0000000000000984 <+122>: callq 0x7e0 <\_ZNSolsEi@plt>  0x0000000000000989 <+127>: mov %rax,%rdx  0x000000000000098c <+130>: mov 0x20063d(%rip),%rax # 0x200fd0  0x0000000000000993 <+137>: mov %rax,%rsi  0x0000000000000996 <+140>: mov %rdx,%rdi  0x0000000000000999 <+143>: callq 0x7c0 <\_ZNSolsEPFRSoS\_E@plt> |  |
| 0x000000000000099e <+148>: addl $0x1,-0xc(%rbp)  0x00000000000009a2 <+152>: jmp 0x92a <main+32> | Increment inner loop and jump to condition |
| 0x00000000000009a4 <+154>: addl $0x1,-0x8(%rbp)  0x00000000000009a8 <+158>: jmpq 0x919 <main+15> | Increment outer loop and jump to condition |
| 0x00000000000009ad <+163>: mov $0x0,%eax  0x00000000000009b2 <+168>: leaveq  0x00000000000009b3 <+169>: retq | Ends function and returns |

**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** |
| --- | --- | --- |
| 0x000000000000090a <+0>: push %rbp  0x000000000000090b <+1>: mov %rsp,%rbp  0x000000000000090e <+4>: sub $0x10,%rsp | int main() {  int i, j, x; | Sets up main function and reserves space for local variables |
| 0x0000000000000912 <+8>: movl $0x1,-0x8(%rbp) | The i = 1; in  for (i = 1; i <= 9; i++) { | Initializes the outer loop iterator as 1 |
| 0x0000000000000919 <+15>: cmpl $0x9,-0x8(%rbp)  0x000000000000091d <+19>: jg 0x9ad <main+163> | The i <= 9; in  for (i = 1; i <= 9; i++) { | Checks condition of loop |
| 0x0000000000000923 <+25>: movl $0x1,-0xc(%rbp) | The j = 1; in  for (j = 1; j <= 9; j++) { | Initializes the inner loop iterator as 1 |
| 0x000000000000092a <+32>: cmpl $0x9,-0xc(%rbp)  0x000000000000092e <+36>: jg 0x9a4 <main+154> | The j <= 9; in  for (j = 1; j <= 9; j++) { | Checks condition of the loop |
| 0x0000000000000930 <+38>: mov -0x8(%rbp),%eax  0x0000000000000933 <+41>: imul -0xc(%rbp),%eax  0x0000000000000937 <+45>: mov %eax,-0x4(%rbp) | x = i \* j; | Multiplies the iterators and stores the result |
| 0x000000000000093a <+48>: mov -0x8(%rbp),%eax  0x000000000000093d <+51>: mov %eax,%esi  0x000000000000093f <+53>: lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000946 <+60>: callq 0x7e0 <\_ZNSolsEi@plt>  0x000000000000094b <+65>: lea 0x153(%rip),%rsi # 0xaa5 | cout << i << " \* " << j << " = " << x << endl; | Print statement |
| 0x0000000000000952 <+72>: mov %rax,%rdi  0x0000000000000955 <+75>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x000000000000095a <+80>: mov %rax,%rdx  0x000000000000095d <+83>: mov -0xc(%rbp),%eax  0x0000000000000960 <+86>: mov %eax,%esi  0x0000000000000962 <+88>: mov %rdx,%rdi  0x0000000000000965 <+91>: callq 0x7e0 <\_ZNSolsEi@plt>  0x000000000000096a <+96>: lea 0x138(%rip),%rsi # 0xaa9  0x0000000000000971 <+103>: mov %rax,%rdi  0x0000000000000974 <+106>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000979 <+111>: mov %rax,%rdx  0x000000000000097c <+114>: mov -0x4(%rbp),%eax  0x000000000000097f <+117>: mov %eax,%esi  0x0000000000000981 <+119>: mov %rdx,%rdi  0x0000000000000984  <+122>: callq 0x7e0 <\_ZNSolsEi@plt>  0x0000000000000989 <+127>: mov %rax,%rdx  0x000000000000098c <+130>: mov 0x20063d(%rip),%rax # 0x200fd0  0x0000000000000993 |  | Print statement (cont.) |
| <+137>: mov %rax,%rsi  0x0000000000000996 <+140>: mov %rdx,%rdi  0x0000000000000999 <+143>: callq 0x7c0 <\_ZNSolsEPFRSoS\_E@plt> |  | Print statement (cont.) |
| 0x000000000000099e <+148>: addl $0x1,-0xc(%rbp)  0x00000000000009a2 <+152>: jmp 0x92a <main+32> | The j++; in  for (j = 1; j <= 9; j++) { | Increment the inner loop iterator |
| 0x00000000000009a4 <+154>: addl $0x1,-0x8(%rbp)  0x00000000000009a8 <+158>: jmpq 0x919 <main+15> | The i++; in  for (i = 1; i <= 9; i++) { | Increment the outer loop iterator |
| 0x00000000000009ad <+163>: mov $0x0,%eax  0x00000000000009b2 <+168>: leaveq  0x00000000000009b3 <+169>: retq | return 0;  } | Ends fuction and returns 0 |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 0x00000000000009fa <+0>: push %rbp  0x00000000000009fb <+1>: mov %rsp,%rbp  0x00000000000009fe <+4>: sub $0x30,%rsp    0x0000000000000a02 <+8>: mov %fs:0x28,%rax  0x0000000000000a0b <+17>: mov %rax,-0x8(%rbp)  0x0000000000000a0f <+21>: xor %eax,%eax | Sets up main function and allocates local variables and sets up overflow detection |
| 0x0000000000000a11 <+23>: lea 0x191(%rip),%rsi # 0xba9  0x0000000000000a18 <+30>: lea 0x201601(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a1f <+37>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a24 <+42>: mov %rax,%rdx  0x0000000000000a27 <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  0x0000000000000a2e <+52>: mov %rax,%rsi  0x0000000000000a31 <+55>: mov %rdx,%rdi  0x0000000000000a34 <+58>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | Prints to console “Enter Radius:” |
| 0x0000000000000a39 <+63>: lea -0x14(%rbp),%rax  0x0000000000000a3d <+67>: mov %rax,%rsi  0x0000000000000a40 <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a47 <+77>: callq 0x870 <\_ZNSirsERi@plt> | Gets input from user |
| 0x0000000000000a4c <+82>: mov -0x14(%rbp),%edx  0x0000000000000a4f <+85>: mov -0x14(%rbp),%eax  0x0000000000000a52 <+88>: imul %eax,%edx   0x0000000000000a55 <+91>: mov -0x14(%rbp),%eax  0x0000000000000a58 <+94>: imul %edx,%eax  0x0000000000000a5b <+97>: mov %eax,-0x14(%rbp) | This loads the user input and multiplies it by itself, then multiplies it by itself again (basically input^3) and stores the result in the input variable space. |
| 0x0000000000000a5e <+100>: mov -0x14(%rbp),%eax  0x0000000000000a61 <+103>: cvtsi2sd %eax,%xmm0  0x0000000000000a65 <+107>: movsd 0x15b(%rip),%xmm1 # 0xbc8  0x0000000000000a6d <+115>: mulsd %xmm1,%xmm0  0x0000000000000a71 <+119>: movsd %xmm0,-0x10(%rbp) | This takes input (cubed) and converts it to a double and loads it in %xmm0. Then loads a double value already stored (for us it is 3.14) into %xmm1. Then it multiplies them together and then stores the result in -0x10(%rbp) |
| 0x0000000000000a76 <+124>: lea 0x13a(%rip),%rsi # 0xbb7  0x0000000000000a7d <+131>: lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a84 <+138>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a89 <+143>: mov %rax,%rdx  0x0000000000000a8c <+146>: mov -0x10(%rbp),%rax  0x0000000000000a90 <+150>: mov %rax,-0x28(%rbp)  0x0000000000000a94 <+154>: movsd -0x28(%rbp),%xmm0  0x0000000000000a99 <+159>: mov %rdx,%rdi  0x0000000000000a9c <+162>: callq 0x8d0 <\_ZNSolsEd@plt> | This prints the string “The volume is: “ to the console followed by |
| 0x0000000000000aa1 <+167>: mov $0x0,%eax  0x0000000000000aa6 <+172>: mov -0x8(%rbp),%rcx  0x0000000000000aaa <+176>: xor %fs:0x28,%rcx  0x0000000000000ab3 <+185>: je 0xaba <main+192>  0x0000000000000ab5 <+187>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  0x0000000000000aba <+192>: leaveq  0x0000000000000abb <+193>: retq | Checks for overflow, ends function, and returns 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** |
| --- | --- | --- |
| 0x00000000000009fa <+0>: push %rbp  0x00000000000009fb <+1>: mov %rsp,%rbp  0x00000000000009fe <+4>: sub $0x30,%rsp    0x0000000000000a02 <+8>: mov %fs:0x28,%rax  0x0000000000000a0b <+17>: mov %rax,-0x8(%rbp)  0x0000000000000a0f <+21>: xor %eax,%eax | int main() {  int input; | Set up main and reserve space for local variables and set up overflow detection |
| 0x0000000000000a11 <+23>: lea 0x191(%rip),%rsi # 0xba9  0x0000000000000a18 <+30>: lea 0x201601(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a1f <+37>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a24 <+42>: mov %rax,%rdx  0x0000000000000a27 <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  0x0000000000000a2e <+52>: mov %rax,%rsi  0x0000000000000a31 <+55>: mov %rdx,%rdi  0x0000000000000a34 <+58>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Enter Radius:" << endl; | Print statement |
| 0x0000000000000a39 <+63>: lea -0x14(%rbp),%rax  0x0000000000000a3d <+67>: mov %rax,%rsi  0x0000000000000a40 <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a47 <+77>: callq 0x870 <\_ZNSirsERi@plt> | cin >> input; | Get user input |
| 0x0000000000000a4c <+82>: mov -0x14(%rbp),%edx  0x0000000000000a4f <+85>: mov -0x14(%rbp),%eax  0x0000000000000a52 <+88>: imul %eax,%edx   0x0000000000000a55 <+91>: mov -0x14(%rbp),%eax  0x0000000000000a58 <+94>: imul %edx,%eax  0x0000000000000a5b <+97>: mov %eax,-0x14(%rbp) | input = input \* input \* input; | Cube input |
| 0x0000000000000a5e <+100>: mov -0x14(%rbp),%eax  0x0000000000000a61 <+103>: cvtsi2sd %eax,%xmm0  0x0000000000000a65 <+107>: movsd 0x15b(%rip),%xmm1 # 0xbc8  0x0000000000000a6d <+115>: mulsd %xmm1,%xmm0  0x0000000000000a71 <+119>: movsd %xmm0,-0x10(%rbp) | double result = static\_cast<double>(input) \* 3.14; | Turn input into a double then multiply by 3.14. Save result to result. |
| 0x0000000000000a76 <+124>: lea 0x13a(%rip),%rsi # 0xbb7  0x0000000000000a7d <+131>: lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a84 <+138>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a89 <+143>: mov %rax,%rdx  0x0000000000000a8c <+146>: mov -0x10(%rbp),%rax  0x0000000000000a90 <+150>: mov %rax,-0x28(%rbp)  0x0000000000000a94 <+154>: movsd -0x28(%rbp),%xmm0  0x0000000000000a99 <+159>: mov %rdx,%rdi  0x0000000000000a9c <+162>: callq 0x8d0 <\_ZNSolsEd@plt> | cout << "The volume is: " << result << endl; | Print statement and print result |
| 0x0000000000000aa1 <+167>: mov $0x0,%eax  0x0000000000000aa6 <+172>: mov -0x8(%rbp),%rcx  0x0000000000000aaa <+176>: xor %fs:0x28,%rcx  0x0000000000000ab3 <+185>: je 0xaba <main+192>  0x0000000000000ab5 <+187>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  0x0000000000000aba <+192>: leaveq  0x0000000000000abb <+193>: retq | return 0;  } | Check for overflow, end function, and return 0; |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 0x00000000000009ba <+0>: push %rbp  0x00000000000009bb <+1>: mov %rsp,%rbp  0x00000000000009be <+4>: sub $0x20,%rsp  0x00000000000009c2 <+8>: mov %fs:0x28,%rax  0x00000000000009cb <+17>: mov %rax,-0x8(%rbp)  0x00000000000009cf <+21>: xor %eax,%eax | Sets up main function and allocates local variables and sets up overflow detection |
| 0x00000000000009d1 <+23>: movl $0x1,-0xc(%rbp) | Stores the value of 1 in a variable -0xc(%rbp) |
| 0x00000000000009d8 <+30>: lea 0x256(%rip),%rsi # 0xc35  0x00000000000009df <+37>: lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x00000000000009e6 <+44>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x00000000000009eb <+49>: mov %rax,%rdx  0x00000000000009ee <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  0x00000000000009f5 <+59>: mov %rax,%rsi  0x00000000000009f8 <+62>: mov %rdx,%rdi  0x00000000000009fb <+65>: callq 0x870 <\_ZNSolsEPFRSoS\_E@plt> | This outputs “Enter number of rows” to the console |
| 0x0000000000000a00 <+70>: lea -0x18(%rbp),%rax  0x0000000000000a04 <+74>: mov %rax,%rsi  0x0000000000000a07 <+77>: lea 0x201732(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a0e <+84>: callq 0x840 <\_ZNSirsERi@plt> | This gets a user input and stores it in a variable -0x18(%rbp). We’ll call it input. |
| 0x0000000000000a13 <+89>: mov -0x18(%rbp),%eax  0x0000000000000a16 <+92>: sub $0x1,%eax    0x0000000000000a19 <+95>: mov %eax,-0xc(%rbp) | Subtracts 1 from input and stores in variable from line 23 -0xc(%rbp). |
| 0x0000000000000a1c <+98>: movl $0x1,-0x10(%rbp) | Stores 1 in another variable -0x10(%rbp) |
| 0x0000000000000a23 <+105>: mov -0x18(%rbp),%eax  0x0000000000000a26 <+108>: cmp %eax,-0x10(%rbp)  0x0000000000000a29 <+111>: jg 0xa9d <main+227> | If input > -0x10(%rbp) jump to line 227 |
| 0x0000000000000a2b <+113>: movl $0x1,-0x14(%rbp) | Stores 1 in another variable -0x14(%rbp) |
| 0x0000000000000a32 <+120>: mov -0x14(%rbp),%eax  0x0000000000000a35 <+123>: cmp -0xc(%rbp),%eax  0x0000000000000a38 <+126>: jg 0xa53 <main+153> | If -0xc(%rbp) > -0x14(%rbp) jump to line 153 |
| 0x0000000000000a3a <+128>: lea 0x209(%rip),%rsi # 0xc4a  0x0000000000000a41 <+135>: lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a48 <+142>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | This prints “ ” to the console. |
| 0x0000000000000a4d <+147>: addl $0x1,-0x14(%rbp)  0x0000000000000a51 <+151>: jmp 0xa32 <main+120> | Increments -0x14(%rbp) and jumps to line 120 |
| 0x0000000000000a53 <+153>: subl $0x1,-0xc(%rbp) | Decrement -0xc(%rbp) |
| 0x0000000000000a57 <+157>: movl $0x1,-0x14(%rbp) | Sets -0x14(%rbp) to 1; |
| 0x0000000000000a5e <+164>: mov -0x10(%rbp),%eax  0x0000000000000a61 <+167>: add %eax,%eax  0x0000000000000a63 <+169>: sub $0x1,%eax  0x0000000000000a66 <+172>: cmp %eax,-0x14(%rbp)  0x0000000000000a69 <+175>: jg 0xa84 <main+202> | Moves -0x10(%rbp) into %eax, doubles it, subtracts 1 and if it is > -0x14(%rbp) jump to line 202 |
| 0x0000000000000a6b <+177>: lea 0x1da(%rip),%rsi # 0xc4c  0x0000000000000a72 <+184>: lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>    0x0000000000000a79 <+191>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Prints “\*” to console. |
| 0x0000000000000a7e <+196>: addl $0x1,-0x14(%rbp)  0x0000000000000a82 <+200>: jmp 0xa5e <main+164> | Increment -0x14(%rbp)  Jump to line 164 |
| 0x0000000000000a84 <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  0x0000000000000a8b <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a92 <+216>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Prints endl to console. |
| 0x0000000000000a97 <+221>: addl $0x1,-0x10(%rbp)  0x0000000000000a9b <+225>: jmp 0xa23 <main+105> | Increment -0x10(%rbp)  Jump to line 105 |
| 0x0000000000000a9d <+227>: movl $0x1,-0xc(%rbp) | Set -0xc(%rbp) to 1 |
| 0x0000000000000aa4 <+234>: movl $0x1,-0x10(%rbp) | Set -0x10(%rbp) to 1 |
| 0x0000000000000aab <+241>: mov -0x18(%rbp),%eax  0x0000000000000aae <+244>: sub $0x1,%eax  0x0000000000000ab1 <+247>: cmp %eax,-0x10(%rbp)  0x0000000000000ab4 <+250>: jg 0xb2b <main+369> | Move -0x18(%rbp) to eax register, subtract 1, and if > -0x10(%rbp) jump to line 369 |
| 0x0000000000000ab6 <+252>: movl $0x1,-0x14(%rbp) | Set -0x14(%rbp) to 1. |
| 0x0000000000000abd <+259>: mov -0x14(%rbp),%eax  0x0000000000000ac0 <+262>: cmp -0xc(%rbp),%eax  0x0000000000000ac3 <+265>: jg 0xade <main+292> | Moves -0x14(%rbp) to register eax and if 0xc(%rbp) > %eax jump to line 292 |
| 0x0000000000000ac5 <+267>: lea 0x17e(%rip),%rsi # 0xc4a  0x0000000000000acc <+274>: lea 0x20154d(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000ad3 <+281>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Prints “ ” to console |
| 0x0000000000000ad8 <+286>: addl $0x1,-0x14(%rbp)    0x0000000000000adc <+290>: jmp 0xabd <main+259> | Increment -0x14(%rbp)  Jump to line 259 |
| 0x0000000000000ade <+292>: addl $0x1,-0xc(%rbp) | Increment -0xc(%rbp) |
| 0x0000000000000ae2 <+296>: movl $0x1,-0x14(%rbp) | set -0x14(%rbp) to 1 |
| 0x0000000000000ae9 <+303>: mov -0x18(%rbp),%eax  0x0000000000000aec <+306>: sub -0x10(%rbp),%eax  0x0000000000000aef <+309>: add %eax,%eax  0x0000000000000af1 <+311>: sub $0x1,%eax  0x0000000000000af4 <+314>: cmp %eax,-0x14(%rbp)  0x0000000000000af7 <+317>: jg 0xb12 <main+344> | Move -0x18(%rbp) into the register %eax, subtract -0x10(%rbp) and store in %eax, add %eax to itself (double it), subtract 1, and then if that is greater than -0x14(%rbp) jump to line 344 |
| 0x0000000000000af9 <+319>: lea 0x14c(%rip),%rsi # 0xc4c  0x0000000000000b00 <+326>: lea 0x201519(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000b07 <+333>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Prints “\*” to console |
| 0x0000000000000b0c <+338>: addl $0x1,-0x14(%rbp)  0x0000000000000b10 <+342>: jmp 0xae9 <main+303> | Increments -0x14(%rbp)  Jump to line 303 |
| 0x0000000000000b12 <+344>: lea 0x135(%rip),%rsi # 0xc4e  0x0000000000000b19 <+351>: lea 0x201500(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000b20 <+358>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Prints endl to console |
| 0x0000000000000b25 <+363>: addl $0x1,-0x10(%rbp)  0x0000000000000b29 <+367>: jmp 0xaab <main+241> | Increment -0x10(%rbp)  Jump to line 241 |
| 0x0000000000000b2b <+369>: mov $0x1,%eax  0x0000000000000b30 <+374>: mov -0x8(%rbp),%rcx  0x0000000000000b34 <+378>: xor %fs:0x28,%rcx  0x0000000000000b3d <+387>: je 0xb44 <main+394>  0x0000000000000b3f <+389>: callq 0x880 <\_\_stack\_chk\_fail@plt>  0x0000000000000b44 <+394>: leaveq  0x0000000000000b45 <+395>: retq | Checks for overflow, ends function, and returns 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** |
| --- | --- | --- |
| 0x00000000000009ba <+0>: push %rbp  0x00000000000009bb <+1>: mov %rsp,%rbp  0x00000000000009be <+4>: sub $0x20,%rsp  0x00000000000009c2 <+8>: mov %fs:0x28,%rax  0x00000000000009cb <+17>: mov %rax,-0x8(%rbp)  0x00000000000009cf <+21>: xor %eax,%eax | int main() {  int input, c, i, j; | Set up main function and declare variables |
| 0x00000000000009d1 <+23>: movl $0x1,-0xc(%rbp) | c = 1; | Set c = 1; |
| 0x00000000000009d8 <+30>: lea 0x256(%rip),%rsi # 0xc35  0x00000000000009df <+37>: lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x00000000000009e6 <+44>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x00000000000009eb <+49>: mov %rax,%rdx  0x00000000000009ee <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  0x00000000000009f5 <+59>: mov %rax,%rsi  0x00000000000009f8 <+62>: mov %rdx,%rdi  0x00000000000009fb <+65>: callq 0x870 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Enter number of rows" << endl; | Prints “Enter number of rows” to console |
| 0x0000000000000a00 <+70>: lea -0x18(%rbp),%rax  0x0000000000000a04 <+74>: mov %rax,%rsi  0x0000000000000a07 <+77>: lea 0x201732(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a0e <+84>: callq 0x840 <\_ZNSirsERi@plt> | cin >> input; | Gets user input and saves to input variable. |
| 0x0000000000000a13 <+89>: mov -0x18(%rbp),%eax  0x0000000000000a16 <+92>: sub $0x1,%eax    0x0000000000000a19 <+95>: mov %eax,-0xc(%rbp) | c = input - 1; | Assigns variable c to input - 1 |
| 0x0000000000000a1c <+98>: movl $0x1,-0x10(%rbp) | The i = 1 in  for (i = 1; i <= input; i++) { | Initializes i to 1 |
| 0x0000000000000a23 <+105>: mov -0x18(%rbp),%eax  0x0000000000000a26 <+108>: cmp %eax,-0x10(%rbp)  0x0000000000000a29 <+111>: jg 0xa9d <main+227> | The i <= input in  for (i = 1; i <= input; i++) { | Sets condition for first outer loop |
| 0x0000000000000a2b <+113>: movl $0x1,-0x14(%rbp) | The j = 1 in  for (j = 1; j <= c; j++) { | Initializes j to 1 |
| 0x0000000000000a32 <+120>: mov -0x14(%rbp),%eax  0x0000000000000a35 <+123>: cmp -0xc(%rbp),%eax  0x0000000000000a38 <+126>: jg 0xa53 <main+153> | The j <= c in  for (j = 1; j <= c; j++) { | Sets condition for first inner loop |
| 0x0000000000000a3a <+128>: lea 0x209(%rip),%rsi # 0xc4a  0x0000000000000a41 <+135>: lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a48 <+142>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << " "; | Prints a space to console |
| 0x0000000000000a4d <+147>: addl $0x1,-0x14(%rbp)  0x0000000000000a51 <+151>: jmp 0xa32 <main+120> | The j++ in  for (j = 1; j <= c; j++) { | Increment the iterator j |
| 0x0000000000000a53 <+153>: subl $0x1,-0xc(%rbp) | c--; | Decrement c |
| 0x0000000000000a57 <+157>: movl $0x1,-0x14(%rbp) | The j = 1 in  for (j = 1; j <= i \* 2 - 1; j++) { | Initialize j to 1 |
| 0x0000000000000a5e <+164>: mov -0x10(%rbp),%eax  0x0000000000000a61 <+167>: add %eax,%eax  0x0000000000000a63 <+169>: sub $0x1,%eax  0x0000000000000a66 <+172>: cmp %eax,-0x14(%rbp)  0x0000000000000a69 <+175>: jg 0xa84 <main+202> | The j <= i \* 2 - 1 in  for (j = 1; j <= i \* 2 - 1; j++) { | Sets condition for second inner loop. |
| 0x0000000000000a6b <+177>: lea 0x1da(%rip),%rsi # 0xc4c  0x0000000000000a72 <+184>: lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>    0x0000000000000a79 <+191>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << "\*"; | Prints “\*” to the console |
| 0x0000000000000a7e <+196>: addl $0x1,-0x14(%rbp)  0x0000000000000a82 <+200>: jmp 0xa5e <main+164> | The j++ in  for (j = 1; j <= i \* 2 - 1; j++) { | Increment the iterator j |
| 0x0000000000000a84 <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  0x0000000000000a8b <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a92 <+216>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << endl; | Start new line |
| 0x0000000000000a97 <+221>: addl $0x1,-0x10(%rbp)  0x0000000000000a9b <+225>: jmp 0xa23 <main+105> | The i++ in  for (i = 1; i <= input; i++) { | Increment the iterator i |
| 0x0000000000000a9d <+227>: movl $0x1,-0xc(%rbp) | c = 1; | Set c to 1 |
| 0x0000000000000aa4 <+234>: movl $0x1,-0x10(%rbp) | The i = 1 in  for (i = 1; i <= input - 1; i++) { | Initializes i to 1 |
| 0x0000000000000aab <+241>: mov -0x18(%rbp),%eax  0x0000000000000aae <+244>: sub $0x1,%eax  0x0000000000000ab1 <+247>: cmp %eax,-0x10(%rbp)  0x0000000000000ab4 <+250>: jg 0xb2b <main+369> | The i <= input - 1 in  for (i = 1; i <= input - 1; i++) { | Set condition for second outer loop |
| 0x0000000000000ab6 <+252>: movl $0x1,-0x14(%rbp) | The j = 1 in  for (j = 1; j <= c; j++) { | Initializes j to 1 |
| 0x0000000000000abd <+259>: mov -0x14(%rbp),%eax  0x0000000000000ac0 <+262>: cmp -0xc(%rbp),%eax  0x0000000000000ac3 <+265>: jg 0xade <main+292> | The j <= c in  for (j = 1; j <= c; j++) { | Sets condition of first inner loop |
| 0x0000000000000ac5 <+267>: lea 0x17e(%rip),%rsi # 0xc4a  0x0000000000000acc <+274>: lea 0x20154d(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000ad3 <+281>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << " "; | Prints a space to the console |
| 0x0000000000000ad8 <+286>: addl $0x1,-0x14(%rbp)    0x0000000000000adc <+290>: jmp 0xabd <main+259> | The j++ in  for (j = 1; j <= c; j++) { | Increment the iterator j |
| 0x0000000000000ade <+292>: addl $0x1,-0xc(%rbp) | c++; | Increment c |
| 0x0000000000000ae2 <+296>: movl $0x1,-0x14(%rbp) | The j = 1 in  for (j = 1; j <= ((input - i) \* 2) - 1; j++) { | Initializes j to 1 |
| 0x0000000000000ae9 <+303>: mov -0x18(%rbp),%eax  0x0000000000000aec <+306>: sub -0x10(%rbp),%eax  0x0000000000000aef <+309>: add %eax,%eax  0x0000000000000af1 <+311>: sub $0x1,%eax  0x0000000000000af4 <+314>: cmp %eax,-0x14(%rbp)  0x0000000000000af7 <+317>: jg 0xb12 <main+344> | The j = <= ((input - i) \* 2) - 1; in  for (j = 1; j <= ((input - i) \* 2) - 1; j++) { | Sets the condition for the second inner loop |
| 0x0000000000000af9 <+319>: lea 0x14c(%rip),%rsi # 0xc4c  0x0000000000000b00 <+326>: lea 0x201519(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000b07 <+333>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << "\*"; | Prints “\*” to the console |
| 0x0000000000000b0c <+338>: addl $0x1,-0x14(%rbp)  0x0000000000000b10 <+342>: jmp 0xae9 <main+303> | The j++; in  for (j = 1; j <= ((input - i) \* 2) - 1; j++) { | Increment the iterator j |
| 0x0000000000000b12 <+344>: lea 0x135(%rip),%rsi # 0xc4e  0x0000000000000b19 <+351>: lea 0x201500(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000b20 <+358>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout << endl; | Start a new line |
| 0x0000000000000b25 <+363>: addl $0x1,-0x10(%rbp)  0x0000000000000b29 <+367>: jmp 0xaab <main+241> | The i++ in  for (i = 1; i <= input - 1; i++) { | Increment the iterator i |
| 0x0000000000000b2b <+369>: mov $0x1,%eax  0x0000000000000b30 <+374>: mov -0x8(%rbp),%rcx  0x0000000000000b34 <+378>: xor %fs:0x28,%rcx  0x0000000000000b3d <+387>: je 0xb44 <main+394>  0x0000000000000b3f <+389>: callq 0x880 <\_\_stack\_chk\_fail@plt>  0x0000000000000b44 <+394>: leaveq  0x0000000000000b45 <+395>: retq | return 1;  } | Checks for overflow, ends function, and returns 1 |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 0x00000000000009fa <+0>: push %rbp  0x00000000000009fb <+1>: mov %rsp,%rbp  0x00000000000009fe <+4>: sub $0x30,%rsp  0x0000000000000a02 <+8>: mov %fs:0x28,%rax  0x0000000000000a0b <+17>: mov %rax,-0x8(%rbp)  0x0000000000000a0f <+21>: xor %eax,%eax | Sets up main function and allocates local variables and sets up overflow detection |
| 0x0000000000000a11 <+23>: movq $0x0,-0x20(%rbp)  0x0000000000000a19 <+31>: movq $0x1,-0x18(%rbp) | Set -0x20(%rbp) to 0  Set -0x18(%rbp) to 1 |
| 0x0000000000000a21 <+39>: lea 0x201(%rip),%rsi # 0xc29  0x0000000000000a28 <+46>: lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a2f <+53>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a34 <+58>: mov %rax,%rdx  0x0000000000000a37 <+61>: mov 0x201592(%rip),%rax # 0x201fd0  0x0000000000000a3e <+68>: mov %rax,%rsi  0x0000000000000a41 <+71>: mov %rdx,%rdi  0x0000000000000a44 <+74>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | Print “Enter the binary number:” |
| 0x0000000000000a49 <+79>: lea -0x28(%rbp),%rax  0x0000000000000a4d <+83>: mov %rax,%rsi  0x0000000000000a50 <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a57 <+93>: callq 0x870 <\_ZNSirsERl@plt> | Get user input in -0x28(%rbp). We’ll call this variable input. |
| 0x0000000000000a5c <+98>: mov -0x28(%rbp),%rax  0x0000000000000a60 <+102>: test %rax,%rax    0x0000000000000a63 <+105>: je 0xaec <main+242> | This tests if 0 is entered by using AND operation. If input is 0 jump to 242. |
| 0x0000000000000a69 <+111>: mov -0x28(%rbp),%rcx  0x0000000000000a6d <+115>: movabs $0x6666666666666667,%rdx  0x0000000000000a77 <+125>: mov %rcx,%rax  0x0000000000000a7a <+128>: imul %rdx  0x0000000000000a7d <+131>: sar $0x2,%rdx  0x0000000000000a81 <+135>: mov %rcx,%rax  0x0000000000000a84 <+138>: sar $0x3f,%rax  0x0000000000000a88 <+142>: sub %rax,%rdx  0x0000000000000a8b <+145>: mov %rdx,%rax  0x0000000000000a8e <+148>: mov %rax,-0x10(%rbp) | This is a little confusing because of all the optimization done but it is basically setting a new variable -0x10(%rbp) to input % 10 (the last digit of the binary number). |
| 0x0000000000000a92 <+152>: mov -0x10(%rbp),%rdx  0x0000000000000a96 <+156>: mov %rdx,%rax  0x0000000000000a99 <+159>: shl $0x2,%rax  0x0000000000000a9d <+163>: add %rdx,%rax  0x0000000000000aa0 <+166>: add %rax,%rax  0x0000000000000aa3 <+169>: sub %rax,%rcx  0x0000000000000aa6 <+172>: mov %rcx,%rax  0x0000000000000aa9 <+175>: mov %rax,-0x10(%rbp) | Load -0x10(%rbp) into rdx and rax. Left shift rax by 2 (multiply by 4). Add rax and rdx (same as -0x10(%rbp) \* 5) in rax. Add rax to rax (same as -0x10(%rbp) \* 10) in rax. Subtract rax from rcx (-0x28(%rbp) input). Move rcx to rax and store rax in -0x10(%rbp) |
| 0x0000000000000aad <+179>: mov -0x10(%rbp),%rax  0x0000000000000ab1 <+183>: imul -0x18(%rbp),%rax  0x0000000000000ab6 <+188>: add %rax,-0x20(%rbp) | Replace -0x20(%rbp) with -0x10(%rbp) \*-0x18(%rbp) + -0x20(%rbp) |
| 0x0000000000000aba <+192>: shlq -0x18(%rbp) | Left shift -0x18(%rbp). Same as multiply by 2. |
| 0x0000000000000abe <+196>: mov -0x28(%rbp),%rcx  0x0000000000000ac2 <+200>: movabs $0x6666666666666667,%rdx  0x0000000000000acc <+210>: mov %rcx,%rax    0x0000000000000acf <+213>: imul %rdx  0x0000000000000ad2 <+216>: sar $0x2,%rdx  0x0000000000000ad6 <+220>: mov %rcx,%rax  0x0000000000000ad9 <+223>: sar $0x3f,%rax  0x0000000000000add <+227>: sub %rax,%rdx  0x0000000000000ae0 <+230>: mov %rdx,%rax  0x0000000000000ae3 <+233>: mov %rax,-0x28(%rbp)  0x0000000000000ae7 <+237>: jmpq 0xa5c <main+98> | Once again this is a little confusing because of optimization but this basically reassigning input to input / 10 (knocking off the last digit). Then it sends the program back to the conditional. |
| 0x0000000000000aec <+242>: lea 0x155(%rip),%rsi # 0xc48  0x0000000000000af3 <+249>: lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000afa <+256>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000aff <+261>: mov %rax,%rdx  0x0000000000000b02 <+264>: mov -0x20(%rbp),%rax  0x0000000000000b06 <+268>: mov %rax,%rsi  0x0000000000000b09 <+271>: mov %rdx,%rdi  0x0000000000000b0c <+274>: callq 0x8d0 <\_ZNSolsEl@plt>  0x0000000000000b11 <+279>: mov %rax,%rdx  0x0000000000000b14 <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  0x0000000000000b1b <+289>: mov %rax,%rsi  0x0000000000000b1e <+292>: mov %rdx,%rdi  0x0000000000000b21 <+295>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | Print out “Equivalent hexidecimal value: “ then -0x20(%rbp) then endl |
| 0x0000000000000b26 <+300>: mov $0x0,%eax  0x0000000000000b2b <+305>: mov -0x8(%rbp),%rsi  0x0000000000000b2f <+309>: xor %fs:0x28,%rsi    0x0000000000000b38 <+318>: je 0xb3f <main+325>  0x0000000000000b3a <+320>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  0x0000000000000b3f <+325>: leaveq  0x0000000000000b40 <+326>: retq | Check for overflow, end function, and 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** |
| --- | --- | --- |
| 0x00000000000009fa <+0>: push %rbp  0x00000000000009fb <+1>: mov %rsp,%rbp  0x00000000000009fe <+4>: sub $0x30,%rsp  0x0000000000000a02 <+8>: mov %fs:0x28,%rax  0x0000000000000a0b <+17>: mov %rax,-0x8(%rbp)  0x0000000000000a0f <+21>: xor %eax,%eax | int main() {  long binaryNumber, decimalNumber, powerOfTwo;  int lastDigit; | Sets up main function and allocates local variables and sets up overflow detection |
| 0x0000000000000a11 <+23>: movq $0x0,-0x20(%rbp)  0x0000000000000a19 <+31>: movq $0x1,-0x18(%rbp) | decimalNumber = 0;  powerOfTwo = 1; | Initialize variables |
| 0x0000000000000a21 <+39>: lea 0x201(%rip),%rsi # 0xc29  0x0000000000000a28 <+46>: lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000a2f <+53>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000a34 <+58>: mov %rax,%rdx  0x0000000000000a37 <+61>: mov 0x201592(%rip),%rax # 0x201fd0  0x0000000000000a3e <+68>: mov %rax,%rsi  0x0000000000000a41 <+71>: mov %rdx,%rdi  0x0000000000000a44 <+74>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Enter the binary number:" << endl; | Print “Enter the binary number:” |
| 0x0000000000000a49 <+79>: lea -0x28(%rbp),%rax  0x0000000000000a4d <+83>: mov %rax,%rsi  0x0000000000000a50 <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  0x0000000000000a57 <+93>: callq 0x870 <\_ZNSirsERl@plt> | cin >> binaryNumber; | Get user input and store in binaryNumber |
| 0x0000000000000a5c <+98>: mov -0x28(%rbp),%rax  0x0000000000000a60 <+102>: test %rax,%rax    0x0000000000000a63 <+105>: je 0xaec <main+242> | while (binaryNumber != 0) { | Condition for loop |
| 0x0000000000000a69 <+111>: mov -0x28(%rbp),%rcx  0x0000000000000a6d <+115>: movabs $0x6666666666666667,%rdx  0x0000000000000a77 <+125>: mov %rcx,%rax  0x0000000000000a7a <+128>: imul %rdx  0x0000000000000a7d <+131>: sar $0x2,%rdx  0x0000000000000a81 <+135>: mov %rcx,%rax  0x0000000000000a84 <+138>: sar $0x3f,%rax  0x0000000000000a88 <+142>: sub %rax,%rdx  0x0000000000000a8b <+145>: mov %rdx,%rax  0x0000000000000a8e <+148>: mov %rax,-0x10(%rbp) | lastDigit = binaryNumber % 10; | Get last digit of input |
| 0x0000000000000aad <+179>: mov -0x10(%rbp),%rax  0x0000000000000ab1 <+183>: imul -0x18(%rbp),%rax  0x0000000000000ab6 <+188>: add %rax,-0x20(%rbp) | decimalNumber += lastDigit \* powerOfTwo; | Add the base 10 value of last digit by multiplying it by the correct power of 2. |
| 0x0000000000000aba <+192>: shlq -0x18(%rbp) | powerOfTwo \*= 2; | Double power of 2 |
| 0x0000000000000abe <+196>: mov -0x28(%rbp),%rcx  0x0000000000000ac2 <+200>: movabs $0x6666666666666667,%rdx  0x0000000000000acc <+210>: mov %rcx,%rax    0x0000000000000acf <+213>: imul %rdx  0x0000000000000ad2 <+216>: sar $0x2,%rdx  0x0000000000000ad6 <+220>: mov %rcx,%rax  0x0000000000000ad9 <+223>: sar $0x3f,%rax  0x0000000000000add <+227>: sub %rax,%rdx  0x0000000000000ae0 <+230>: mov %rdx,%rax  0x0000000000000ae3 <+233>: mov %rax,-0x28(%rbp)  0x0000000000000ae7 <+237>: jmpq 0xa5c <main+98> | binaryNumber /= 10;  } | Get rid of last digit and return to loop conditional. |
| 0x0000000000000aec <+242>: lea 0x155(%rip),%rsi # 0xc48  0x0000000000000af3 <+249>: lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  0x0000000000000afa <+256>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  0x0000000000000aff <+261>: mov %rax,%rdx  0x0000000000000b02 <+264>: mov -0x20(%rbp),%rax  0x0000000000000b06 <+268>: mov %rax,%rsi  0x0000000000000b09 <+271>: mov %rdx,%rdi  0x0000000000000b0c <+274>: callq 0x8d0 <\_ZNSolsEl@plt>  0x0000000000000b11 <+279>: mov %rax,%rdx  0x0000000000000b14 <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  0x0000000000000b1b <+289>: mov %rax,%rsi  0x0000000000000b1e <+292>: mov %rdx,%rdi  0x0000000000000b21 <+295>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Equivalent hexadecimal value: " << decimalNumber << endl; | Print out “Equivalent hexidecimal value: “ then decimalNumber then endl |
| 0x0000000000000b26 <+300>: mov $0x0,%eax  0x0000000000000b2b <+305>: mov -0x8(%rbp),%rsi  0x0000000000000b2f <+309>: xor %fs:0x28,%rsi    0x0000000000000b38 <+318>: je 0xb3f <main+325>  0x0000000000000b3a <+320>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  0x0000000000000b3f <+325>: leaveq  0x0000000000000b40 <+326>: retq | return 0;  } | Check for overflow, end function, and return 0 |