# **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** |
| --- | --- |
| Dump of assembler code for function main: |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x10,%rsp  <+8>: movl $0x1,-0x8(%rbp) | First two instructions: Function prologues. They push the old base pointer onto the stack, which is saved for a later time. The value is initialized at %rbp. |
| <+15>: cmpl $0x9,-0x8(%rbp)  <+19>: jg 0x9ad <main+163>  <+25>: movl $0x1,-0xc(%rbp)  <+32>: cmpl $0x9,-0xc(%rbp)  <+36>: jg 0x9a4 <main+154> | The operator compares the value 9 to see if the values are greater than those in %rbp. If the value is larger, it’ll jump to the end. The value increments by 1 and then jumps back to the previous block. The comparison occurs twice. |
| <+38>: mov -0x8(%rbp),%eax  <+41>: imul -0xc(%rbp),%eax | The two values for “a” and “i” are multiplied. |
| <+45>: mov %eax,-0x4(%rbp)  <+48>: mov -0x8(%rbp),%eax  <+51>: mov %eax,%esi  <+53>: lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+60>: callq 0x7e0 <\_ZNSolsEi@plt>  <+65>: lea 0x153(%rip),%rsi # 0xaa5  <+72>: mov %rax,%rdi  <+75>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+80>: mov %rax,%rdx  <+83>: mov -0xc(%rbp),%eax  <+86>: mov %eax,%esi  <+88>: mov %rdx,%rdi  <+91>: callq 0x7e0 <\_ZNSolsEi@plt>  <+96>: lea 0x138(%rip),%rsi # 0xaa9  <+103>: mov %rax,%rdi  <+106>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+111>: mov %rax,%rdx  <+114>: mov -0x4(%rbp),%eax  <+117>: mov %eax,%esi  <+119>: mov %rdx,%rdi  <+122>: callq 0x7e0 <\_ZNSolsEi@plt>  <+127>: mov %rax,%rdx  <+130>: mov 0x20063d(%rip),%rax # 0x200fd0  <+137>: mov %rax,%rsi  <+140>: mov %rdx,%rdi  <+143>: callq 0x7c0 <\_ZNSolsEPFRSoS\_E@plt> | The multiplied values listed above are printed. This group calls cout to print the output of the multiplication. |
| <+148>: addl $0x1,-0xc(%rbp)  <+152>: jmp 0x92a <main+32>  <+154>: addl $0x1,-0x8(%rbp)  <+158>: jmpq 0x919 <main+15>  <+163>: mov $0x0,%eax  <+168>: leaveq  <+169>: retq | The compared values above are iterated by 1 and then will jump to the previous block. If the compared values reach 9, the program will exit. |
| End of assembler dump |  |

**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** |
| --- | --- | --- |
| Dump of assembler code for function main: |  |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x10,%rsp  <+8>: movl $0x1,-0x8(%rbp) | a = 1 | The variable “a” is initialized as 1 |
| <+15>: cmpl $0x9,-0x8(%rbp)  <+19>: jg 0x9ad <main+163>  <+25>: movl $0x1,-0xc(%rbp)  <+32>: cmpl $0x9,-0xc(%rbp)  <+36>: jg 0x9a4 <main+154> | for (a = 1; a <= 9; a++){  for (i = 1; i <= 9; i++){ | The variable “i” is initialized as 1. For loops are created for the comparison.  The iteration of “a” and “I” will occur later in the code. |
| <+38>: mov -0x8(%rbp),%eax  <+41>: imul -0xc(%rbp),%eax | x = a \* i; | The variables “a” and “i” are multiplied to the value “x”. |
| <+45>: mov %eax,-0x4(%rbp)  <+48>: mov -0x8(%rbp),%eax  <+51>: mov %eax,%esi  <+53>: lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+60>: callq 0x7e0 <\_ZNSolsEi@plt>  <+65>: lea 0x153(%rip),%rsi # 0xaa5  <+72>: mov %rax,%rdi  <+75>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+80>: mov %rax,%rdx  <+83>: mov -0xc(%rbp),%eax  <+86>: mov %eax,%esi  <+88>: mov %rdx,%rdi  <+91>: callq 0x7e0 <\_ZNSolsEi@plt>  <+96>: lea 0x138(%rip),%rsi # 0xaa9  <+103>: mov %rax,%rdi  <+106>: callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+111>: mov %rax,%rdx  <+114>: mov -0x4(%rbp),%eax  <+117>: mov %eax,%esi  <+119>: mov %rdx,%rdi  <+122>: callq 0x7e0 <\_ZNSolsEi@plt>  <+127>: mov %rax,%rdx  <+130>: mov 0x20063d(%rip),%rax # 0x200fd0  <+137>: mov %rax,%rsi  <+140>: mov %rdx,%rdi  <+143>: callq 0x7c0 <\_ZNSolsEPFRSoS\_E@plt> | cout << a << " \* " << i << " = " << x << endl; | The variables “a” and “I” are printed with “cout” and the output is the result of the multiplication. |
| <+148>: addl $0x1,-0xc(%rbp)  <+152>: jmp 0x92a <main+32>  <+154>: addl $0x1,-0x8(%rbp)  <+158>: jmpq 0x919 <main+15>  <+163>: mov $0x0,%eax  <+168>: leaveq  <+169>: retq | a++  and  i++ | The iteration mentioned above occurs. The loop will finish, and the program will exit. |
| End of assembler dump |  |  |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| Dump of assembler code for function main: |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x30,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: lea 0x191(%rip),%rsi # 0xba9  <+30>: lea 0x201601(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+37>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+42>: mov %rax,%rdx  <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  <+52>: mov %rax,%rsi  <+55>: mov %rdx,%rdi  <+58>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt>  <+63>: lea -0x14(%rbp),%rax  <+67>: mov %rax,%rsi  <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+77>: callq 0x870 <\_ZNSirsERi@plt> | First two instructions: Function prologues. They push the old base pointer onto the stack, which is saved for a later time, meaning the stack is initialized.  The variable is declared  The string, “Enter radius” is printed  User input is requested |
| <+82>: mov -0x14(%rbp),%edx  <+85>: mov -0x14(%rbp),%eax  <+88>: imul %eax,%edx  <+91>: mov -0x14(%rbp),%eax  <+94>: imul %edx,%eax  <+97>: mov %eax,-0x14(%rbp)  <+100>: mov -0x14(%rbp),%eax  <+103>: cvtsi2sd %eax,%xmm0  <+107>: movsd 0x15b(%rip),%xmm1 # 0xbc8  <+115>: mulsd %xmm1,%xmm0  <+119>: movsd %xmm0,-0x10(%rbp)  <+124>: lea 0x13a(%rip),%rsi # 0xbb7  <+131>: lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4> | The input requested is stored. The value is multiplied by itself.  The value of π is moved.  That value is then multiplied with the value previously requested.  The int is converted to a float. |
| <+138>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+143>: mov %rax,%rdx  <+146>: mov -0x10(%rbp),%rax  <+150>: mov %rax,-0x28(%rbp)  <+154>: movsd -0x28(%rbp),%xmm0  <+159>: mov %rdx,%rdi  <+162>: callq 0x8d0 <\_ZNSolsEd@plt>  <+167>: mov $0x0,%eax  <+172>: mov -0x8(%rbp),%rcx  <+176>: xor %fs:0x28,%rcx  <+185>: je 0xaba <main+192>  <+187>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  <+192>: leaveq  <+193>: retq | The other string captured using the bless command is “The volume is: “. The value is printed from the calculation from the previous code. The program then exits. |
| End of assembler dump. |  |

**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** |
| --- | --- | --- |
| Dump of assembler code for function main: |  |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x30,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: lea 0x191(%rip),%rsi # 0xba9  <+30>: lea 0x201601(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+37>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+42>: mov %rax,%rdx  <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  <+52>: mov %rax,%rsi  <+55>: mov %rdx,%rdi  <+58>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt>  <+63>: lea -0x14(%rbp),%rax  <+67>: mov %rax,%rsi  <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+77>: callq 0x870 <\_ZNSirsERi@plt> | int r = 0;  std::cout << "Enter radius: "  << std::endl;  std::cin >> r; | The variable “r” is declared  The string, “Enter radius” is printed  User input is requested |
| <+82>: mov -0x14(%rbp),%edx  <+85>: mov -0x14(%rbp),%eax  <+88>: imul %eax,%edx  <+91>: mov -0x14(%rbp),%eax  <+94>: imul %edx,%eax  <+97>: mov %eax,-0x14(%rbp)  <+100>: mov -0x14(%rbp),%eax  <+103>: cvtsi2sd %eax,%xmm0  <+107>: movsd 0x15b(%rip),%xmm1 # 0xbc8  <+115>: mulsd %xmm1,%xmm0  <+119>: movsd %xmm0,-0x10(%rbp)  <+124>: lea 0x13a(%rip),%rsi # 0xbb7  <+131>: lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4> | r = r \* r;  double PI = std::atan(1) \* 4;  double volume = r \* PI; | The input requested is stored. The value of “r” is multiplied by itself. This method squares the value.  The value of π is moved and initialized.  That squared value is then multiplied with the previously calculation to find the volume of the cylinder. The int is converted to a float. |
| <+138>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+143>: mov %rax,%rdx  <+146>: mov -0x10(%rbp),%rax  <+150>: mov %rax,-0x28(%rbp)  <+154>: movsd -0x28(%rbp),%xmm0  <+159>: mov %rdx,%rdi  <+162>: callq 0x8d0 <\_ZNSolsEd@plt>  <+167>: mov $0x0,%eax  <+172>: mov -0x8(%rbp),%rcx  <+176>: xor %fs:0x28,%rcx  <+185>: je 0xaba <main+192>  <+187>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  <+192>: leaveq  <+193>: retq | std::cout << "The volume is: " << volume << std::endl; | The string, “The volume is: “, is printed using cout.  The value of volume is printed from the calculation from the previous code.  The program then exits. |
| End of assembler dump. |  |  |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| Dump of assembler code for function main: |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x20,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: movl $0x1,-0xc(%rbp)  <+30>: lea 0x256(%rip),%rsi # 0xc35  <+37>: lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+44>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+49>: mov %rax,%rdx  <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  <+59>: mov %rax,%rsi  <+62>: mov %rdx,%rdi  <+65>: callq 0x870 <\_ZNSolsEPFRSoS\_E@plt>  <+70>: lea -0x18(%rbp),%rax  <+74>: mov %rax,%rsi  <+77>: lea 0x201732(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+84>: callq 0x840 <\_ZNSirsERi@plt> | First two instructions: Function prologues. They push the old base pointer onto the stack, which is saved for a later time, meaning the stack is initialized.  The strings are called and stored within the registers.  The string, “Enter number of rows”, is printed.  User request using “cin” for user input of rows. |
| <+89>: mov -0x18(%rbp),%eax  <+92>: sub $0x1,%eax  <+95>: mov %eax,-0xc(%rbp)  <+98>: movl $0x1,-0x10(%rbp)  <+105>: mov -0x18(%rbp),%eax  <+108>: cmp %eax,-0x10(%rbp)  <+111>: jg 0xa9d <main+227>  <+113>: movl $0x1,-0x14(%rbp)  <+120>: mov -0x14(%rbp),%eax  <+123>: cmp -0xc(%rbp),%eax  <+126>: jg 0xa53 <main+153>  <+128>: lea 0x209(%rip),%rsi # 0xc4a  <+135>: lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+142>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+147>: addl $0x1,-0x14(%rbp)  <+151>: jmp 0xa32 <main+120>  <+153>: subl $0x1,-0xc(%rbp)  <+157>: movl $0x1,-0x14(%rbp)  <+164>: mov -0x10(%rbp),%eax  <+167>: add %eax,%eax  <+169>: sub $0x1,%eax  <+172>: cmp %eax,-0x14(%rbp)  <+175>: jg 0xa84 <main+202>  <+177>: lea 0x1da(%rip),%rsi # 0xc4c  <+184>: lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+191>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+196>: addl $0x1,-0x14(%rbp)  <+200>: jmp 0xa5e <main+164>  <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+216>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+221>: addl $0x1,-0x10(%rbp)  <+225>: jmp 0xa23 <main+105>  <+227>: movl $0x1,-0xc(%rbp)  <+234>: movl $0x1,-0x10(%rbp)  <+241>: mov -0x18(%rbp),%eax  <+244>: sub $0x1,%eax  <+247>: cmp %eax,-0x10(%rbp)  <+250>: jg 0xb2b <main+369>  <+252>: movl $0x1,-0x14(%rbp)  <+259>: mov -0x14(%rbp),%eax  <+262>: cmp -0xc(%rbp),%eax  <+265>: jg 0xade <main+292>  <+267>: lea 0x17e(%rip),%rsi # 0xc4a  <+274>: lea 0x20154d(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+281>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+286>: addl $0x1,-0x14(%rbp)  <+290>: jmp 0xabd <main+259>  <+292>: addl $0x1,-0xc(%rbp)  <+296>: movl $0x1,-0x14(%rbp)  <+303>: mov -0x18(%rbp),%eax  <+306>: sub -0x10(%rbp),%eax  <+309>: add %eax,%eax  <+311>: sub $0x1,%eax  <+314>: cmp %eax,-0x14(%rbp)  <+317>: jg 0xb12 <main+344>  <+319>: lea 0x14c(%rip),%rsi # 0xc4c  <+326>: lea 0x201519(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+333>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+338>: addl $0x1,-0x14(%rbp)  <+342>: jmp 0xae9 <main+303>  <+344>: lea 0x135(%rip),%rsi # 0xc4e  <+351>: lea 0x201500(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+358>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+363>: addl $0x1,-0x10(%rbp)  <+367>: jmp 0xaab <main+241> | User input is stored  The program loops creating more rows until the number of rows defined by the user is reached, while placing a star for each row.  The program moves to the initial row then moves to the next and so on, while placing a star on each side of the row until the output is a diamond. |
| <+369>: mov $0x1,%eax  <+374>: mov -0x8(%rbp),%rcx  <+378>: xor %fs:0x28,%rcx  <+387>: je 0xb44 <main+394>  <+389>: callq 0x880 <\_\_stack\_chk\_fail@plt>  <+394>: leaveq  <+395>: retq | The stack is requested to be cleared    The program now exits |
| End of assembler dump. |  |

**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** |
| --- | --- | --- |
| Dump of assembler code for function main: |  |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x20,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: movl $0x1,-0xc(%rbp)  <+30>: lea 0x256(%rip),%rsi # 0xc35  <+37>: lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+44>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+49>: mov %rax,%rdx  <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  <+59>: mov %rax,%rsi  <+62>: mov %rdx,%rdi  <+65>: callq 0x870 <\_ZNSolsEPFRSoS\_E@plt>  <+70>: lea -0x18(%rbp),%rax  <+74>: mov %rax,%rsi  <+77>: lea 0x201732(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+84>: callq 0x840 <\_ZNSirsERi@plt> | int main(){    int rows;    int space;    int stars;    int width;    std::string spaceString;    std::string starString;  std::cout << "Enter the number of rows" << std::endl;    std::cin >> rows; | After the stack is initialized, the variables are declared.  The string “Enter the number of rows” is printed and populated in the terminal.  The user input is requested for the number of rows defined by the user input. |
| <+89>: mov -0x18(%rbp),%eax  <+92>: sub $0x1,%eax  <+95>: mov %eax,-0xc(%rbp)  <+98>: movl $0x1,-0x10(%rbp)  <+105>: mov -0x18(%rbp),%eax  <+108>: cmp %eax,-0x10(%rbp)  <+111>: jg 0xa9d <main+227>  <+113>: movl $0x1,-0x14(%rbp)  <+120>: mov -0x14(%rbp),%eax  <+123>: cmp -0xc(%rbp),%eax  <+126>: jg 0xa53 <main+153>  <+128>: lea 0x209(%rip),%rsi # 0xc4a  <+135>: lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+142>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+147>: addl $0x1,-0x14(%rbp)  <+151>: jmp 0xa32 <main+120>  <+153>: subl $0x1,-0xc(%rbp)  <+157>: movl $0x1,-0x14(%rbp)  <+164>: mov -0x10(%rbp),%eax  <+167>: add %eax,%eax  <+169>: sub $0x1,%eax  <+172>: cmp %eax,-0x14(%rbp)  <+175>: jg 0xa84 <main+202>  <+177>: lea 0x1da(%rip),%rsi # 0xc4c  <+184>: lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+191>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+196>: addl $0x1,-0x14(%rbp)  <+200>: jmp 0xa5e <main+164>  <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+216>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+221>: addl $0x1,-0x10(%rbp)  <+225>: jmp 0xa23 <main+105>  <+227>: movl $0x1,-0xc(%rbp)  <+234>: movl $0x1,-0x10(%rbp)  <+241>: mov -0x18(%rbp),%eax  <+244>: sub $0x1,%eax  <+247>: cmp %eax,-0x10(%rbp)  <+250>: jg 0xb2b <main+369>  <+252>: movl $0x1,-0x14(%rbp)  <+259>: mov -0x14(%rbp),%eax  <+262>: cmp -0xc(%rbp),%eax  <+265>: jg 0xade <main+292>  <+267>: lea 0x17e(%rip),%rsi # 0xc4a  <+274>: lea 0x20154d(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+281>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+286>: addl $0x1,-0x14(%rbp)  <+290>: jmp 0xabd <main+259>  <+292>: addl $0x1,-0xc(%rbp)  <+296>: movl $0x1,-0x14(%rbp)  <+303>: mov -0x18(%rbp),%eax  <+306>: sub -0x10(%rbp),%eax  <+309>: add %eax,%eax  <+311>: sub $0x1,%eax  <+314>: cmp %eax,-0x14(%rbp)  <+317>: jg 0xb12 <main+344>  <+319>: lea 0x14c(%rip),%rsi # 0xc4c  <+326>: lea 0x201519(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+333>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+338>: addl $0x1,-0x14(%rbp)  <+342>: jmp 0xae9 <main+303>  <+344>: lea 0x135(%rip),%rsi # 0xc4e  <+351>: lea 0x201500(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+358>: callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+363>: addl $0x1,-0x10(%rbp)  <+367>: jmp 0xaab <main+241> | width = rows \* 2 - 1;  for (int i = 0; i < rows; i++) {  stars = i \* 2 + 1;  space = width - stars;  for (int i = rows - 2; i >= 0; i--) {  stars = i \* 2 + 1;  space = width - stars;  for (int j = 0; j < space / 2; j++) {  spaceString = spaceString + ' ';  for (int j = 0; j < stars; j++) {  starString = starString + '\*'; | The user will define the number of rows from the previous code and is stored for the creation of the diamond.  The calculation occurs after the user input and the program will determine the number of stars and space needed to create a diamond.  The loops are for the first half and the second half of the diamond  The calculation prints in the form of a diamond. |
| <+369>: mov $0x1,%eax  <+374>: mov -0x8(%rbp),%rcx  <+378>: xor %fs:0x28,%rcx  <+387>: je 0xb44 <main+394>  <+389>: callq 0x880 <\_\_stack\_chk\_fail@plt>  <+394>: leaveq  <+395>: retq | Ret | The program now will exit after the stack is clear |
| End of assembler dump. |  |  |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| Dump of assembler code for function main: |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x30,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: movq $0x0,-0x20(%rbp)  <+31>: movq $0x1,-0x18(%rbp)  <+39>: lea 0x201(%rip),%rsi # 0xc29  <+46>: lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+53>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+58>: mov %rax,%rdx  <+61>: mov 0x201592(%rip),%rax # 0x201fd0  <+68>: mov %rax,%rsi  <+71>: mov %rdx,%rdi  <+74>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt>  <+79>: lea -0x28(%rbp),%rax  <+83>: mov %rax,%rsi  <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+93>: callq 0x870 <\_ZNSirsERl@plt> | First two instructions: Function prologues. They push the old base pointer onto the stack, which is saved for a later time, meaning the stack is initialized.  The variables are declared.  The provided string, “Enter the binary number “ is printed using “cout.”  User input is requested using the “cin” function and the value will be stored. |
| <+98>: mov -0x28(%rbp),%rax  <+102>: test %rax,%rax  <+105>: je 0xaec <main+242>  <+111>: mov -0x28(%rbp),%rcx  <+115>: movabs $0x6666666666666667,%rdx  <+125>: mov %rcx,%rax  <+128>: imul %rdx  <+131>: sar $0x2,%rdx  <+135>: mov %rcx,%rax  <+138>: sar $0x3f,%rax  <+142>: sub %rax,%rdx  <+145>: mov %rdx,%rax  <+148>: mov %rax,-0x10(%rbp)  <+152>: mov -0x10(%rbp),%rdx  <+156>: mov %rdx,%rax  <+159>: shl $0x2,%rax  <+163>: add %rdx,%rax  <+166>: add %rax,%rax  <+169>: sub %rax,%rcx  <+172>: mov %rcx,%rax  <+175>: mov %rax,-0x10(%rbp)  <+179>: mov -0x10(%rbp),%rax  <+183>: imul -0x18(%rbp),%rax  <+188>: add %rax,-0x20(%rbp)  <+192>: shlq -0x18(%rbp)  <+196>: mov -0x28(%rbp),%rcx  <+200>: movabs $0x6666666666666667,%rdx  <+210>: mov %rcx,%rax  <+213>: imul %rdx  <+216>: sar $0x2,%rdx  <+220>: mov %rcx,%rax  <+223>: sar $0x3f,%rax  <+227>: sub %rax,%rdx  <+230>: mov %rdx,%rax  <+233>: mov %rax,-0x28(%rbp)  <+237>: jmpq 0xa5c <main+98>  <+242>: lea 0x155(%rip),%rsi # 0xc48  <+249>: lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+256>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+261>: mov %rax,%rdx  <+264>: mov -0x20(%rbp),%rax  <+268>: mov %rax,%rsi  <+271>: mov %rdx,%rdi  <+274>: callq 0x8d0 <\_ZNSolsEl@plt>  <+279>: mov %rax,%rdx  <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  <+289>: mov %rax,%rsi  <+292>: mov %rdx,%rdi  <+295>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | The stored user input from previous code request is defined as an int.  The program searches the values from left to right to determine the binary value.  These values are calculated to find the total binary value.  The provided string, “Equivalent hexadecimal value: “ is then printed using “cout” function. The value previously calculated will print after the string. |
| <+300>: mov $0x0,%eax  <+305>: mov -0x8(%rbp),%rsi  <+309>: xor %fs:0x28,%rsi  <+318>: je 0xb3f <main+325>  <+320>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  <+325>: leaveq  <+326>: retq | The stack will then clear and the program will exit. |
| End of assembler dump. |  |

**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** |
| --- | --- | --- |
| Dump of assembler code for function main: |  |  |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x30,%rsp  <+8>: mov %fs:0x28,%rax  <+17>: mov %rax,-0x8(%rbp)  <+21>: xor %eax,%eax  <+23>: movq $0x0,-0x20(%rbp)  <+31>: movq $0x1,-0x18(%rbp)  <+39>: lea 0x201(%rip),%rsi # 0xc29  <+46>: lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+53>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+58>: mov %rax,%rdx  <+61>: mov 0x201592(%rip),%rax # 0x201fd0  <+68>: mov %rax,%rsi  <+71>: mov %rdx,%rdi  <+74>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt>  <+79>: lea -0x28(%rbp),%rax  <+83>: mov %rax,%rsi  <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+93>: callq 0x870 <\_ZNSirsERl@plt> | int main() {    long binary;    long factor = 1;    long total = 0;  std::cout << "Enter the binary number: " << std::endl;  std::cin >> binary; | After the stack is prepared  The variables are declared.  The provided string, “Enter the binary number “ is printed using “cout.”  User input is requested using the “cin” function and the value will be stored. |
| <+98>: mov -0x28(%rbp),%rax  <+102>: test %rax,%rax  <+105>: je 0xaec <main+242>  <+111>: mov -0x28(%rbp),%rcx  <+115>: movabs $0x6666666666666667,%rdx  <+125>: mov %rcx,%rax  <+128>: imul %rdx  <+131>: sar $0x2,%rdx  <+135>: mov %rcx,%rax  <+138>: sar $0x3f,%rax  <+142>: sub %rax,%rdx  <+145>: mov %rdx,%rax  <+148>: mov %rax,-0x10(%rbp)  <+152>: mov -0x10(%rbp),%rdx  <+156>: mov %rdx,%rax  <+159>: shl $0x2,%rax  <+163>: add %rdx,%rax  <+166>: add %rax,%rax  <+169>: sub %rax,%rcx  <+172>: mov %rcx,%rax  <+175>: mov %rax,-0x10(%rbp)  <+179>: mov -0x10(%rbp),%rax  <+183>: imul -0x18(%rbp),%rax  <+188>: add %rax,-0x20(%rbp)  <+192>: shlq -0x18(%rbp)  <+196>: mov -0x28(%rbp),%rcx  <+200>: movabs $0x6666666666666667,%rdx  <+210>: mov %rcx,%rax  <+213>: imul %rdx  <+216>: sar $0x2,%rdx  <+220>: mov %rcx,%rax  <+223>: sar $0x3f,%rax  <+227>: sub %rax,%rdx  <+230>: mov %rdx,%rax  <+233>: mov %rax,-0x28(%rbp)  <+237>: jmpq 0xa5c <main+98>  <+242>: lea 0x155(%rip),%rsi # 0xc48  <+249>: lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  <+256>: callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  <+261>: mov %rax,%rdx  <+264>: mov -0x20(%rbp),%rax  <+268>: mov %rax,%rsi  <+271>: mov %rdx,%rdi  <+274>: callq 0x8d0 <\_ZNSolsEl@plt>  <+279>: mov %rax,%rdx  <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  <+289>: mov %rax,%rsi  <+292>: mov %rdx,%rdi  <+295>: callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | while (binary != 0) {  total += (binary % 10) \* factor;    binary /= 10;  factor \*= 2;    }  std::cout << "Equivalent hexadecimal value: " << total; | Using a while loop, the user input is used, and the total binary value is calculated and stored in “total”.  The output of the total value is printed after the string, “Equivalent hexadecimal value: “ |
| <+300>: mov $0x0,%eax  <+305>: mov -0x8(%rbp),%rsi  <+309>: xor %fs:0x28,%rsi  <+318>: je 0xb3f <main+325>  <+320>: callq 0x8b0 <\_\_stack\_chk\_fail@plt>  <+325>: leaveq  <+326>: retq | Ret | The program now will exit after the stack is clear |
| End of assembler dump. |  |  |