# **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** |
| --- | --- |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x10,%rsp  <+8>: movl $0x1,-0x8(%rbp) | The stack is initilized and the value one is initialized at rbp. |
| <+15>: cmpl $0x9,-0x8(%rbp)  <+19>: jg 0x5555555549ad <main+163>  <+25>: movl $0x1,-0xc(%rbp)  <+32>: cmpl $0x9,-0xc(%rbp)  <+36>: jg 0x5555555549a4 <main+154> | The compare operator is used to see if 9 is greater than the values at rbp. If it is then it jumps to the end of the program where it increments by one and jumps back. This is done twice in this block. |
| <+38>: mov -0x8(%rbp),%eax  <+41>: imul -0xc(%rbp),%eax | Two values are multiplied by each other |
| <+45>: mov %eax,-0x4(%rbp)  <+48>: mov -0x8(%rbp),%eax  <+51>: mov %eax,%esi  <+53>: lea 0x2006da(%rip),%rdi # 0x555555755020  <+60>: callq 0x5555555547e0 <\_ZNSolsEi@plt>  <+65>: lea 0x153(%rip),%rsi # 0x555555554aa5  <+72>: mov %rax,%rdi  <+75>: callq 0x5555555547b0 <\_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 0x5555555547e0 <\_ZNSolsEi@plt>  <+96>: lea 0x138(%rip),%rsi # 0x555555554aa9  <+103>: mov %rax,%rdi  <+106>: callq 0x5555555547b0 <\_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 0x5555555547e0 <\_ZNSolsEi@plt>  <+127>: mov %rax,%rdx  <+130>: mov 0x20063d(%rip),%rax # 0x555555754fd0  <+137>: mov %rax,%rsi  <+140>: mov %rdx,%rdi  <+143>: callq 0x5555555547c0 <\_ZNSolsEPFRSoS\_E@plt> | The two values that were multiplied by each other are printed to the terminal along with the result of the multiplication. Strings are also used so that the form in the terminal looks like # \* # = #. |
| <+148>: addl $0x1,-0xc(%rbp)  <+152>: jmp 0x55555555492a <main+32>  <+154>: addl $0x1,-0x8(%rbp)  <+158>: jmpq 0x555555554919 <main+15>  <+163>: mov $0x0,%eax  <+168>: leaveq  <+169>: retq | The values that were previously compared against 9 are iterated by one here and jump back to the previous block. Once both values reach 9 the program terminates. |

**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** |
| --- | --- | --- |
| <+0>: push %rbp  <+1>: mov %rsp,%rbp  <+4>: sub $0x10,%rsp  <+8>: movl $0x1,-0x8(%rbp) | i = 1; | i is initilized as 1. |
| <+15>: cmpl $0x9,-0x8(%rbp)  <+19>: jg 0x5555555549ad <main+163>  <+25>: movl $0x1,-0xc(%rbp)  <+32>: cmpl $0x9,-0xc(%rbp)  <+36>: jg 0x5555555549a4 <main+154> | For (i; i <= 9; i++){  for (int j = 1; j <= 9; j++) {  }  } | J is also initilized as 1 and the two for loops are created. The increments happen in the last block of code. |
| <+38>: mov -0x8(%rbp),%eax  <+41>: imul -0xc(%rbp),%eax | Int k = i \* j; | I and j are multiplied by each other. |
| <+45>: mov %eax,-0x4(%rbp)  <+48>: mov -0x8(%rbp),%eax  <+51>: mov %eax,%esi  <+53>: lea 0x2006da(%rip),%rdi # 0x555555755020  <+60>: callq 0x5555555547e0 <\_ZNSolsEi@plt>  <+65>: lea 0x153(%rip),%rsi # 0x555555554aa5  <+72>: mov %rax,%rdi  <+75>: callq 0x5555555547b0 <\_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 0x5555555547e0 <\_ZNSolsEi@plt>  <+96>: lea 0x138(%rip),%rsi # 0x555555554aa9  <+103>: mov %rax,%rdi  <+106>: callq 0x5555555547b0 <\_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 0x5555555547e0 <\_ZNSolsEi@plt>  <+127>: mov %rax,%rdx  <+130>: mov 0x20063d(%rip),%rax # 0x555555754fd0  <+137>: mov %rax,%rsi  <+140>: mov %rdx,%rdi  <+143>: callq 0x5555555547c0 <\_ZNSolsEPFRSoS\_E@plt> | Cout << i << ‘\*” << j << ‘=’ << k << endl; | I and j are printed out as a multiplication problem to the terminal for each iteration of the loop in the format # \* # = #. |
| <+148>: addl $0x1,-0xc(%rbp)  <+152>: jmp 0x55555555492a <main+32>  <+154>: addl $0x1,-0x8(%rbp)  <+158>: jmpq 0x555555554919 <main+15>  <+163>: mov $0x0,%eax  <+168>: leaveq  <+169>: retq | I++  j++ | This is where each loop iteration happens. Once the loops are finished the program terminates. |

## **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  <+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 0x555555554890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+42>: mov %rax,%rdx  <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  <+52>: mov %rax,%rsi  <+55>: mov %rdx,%rdi  <+58>: callq 0x5555555548a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+63>: lea -0x14(%rbp),%rax  <+67>: mov %rax,%rsi  <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+77>: callq 0x555555554870 <std::istream::operator>>(int&)@plt> | The stack is initilized. Strings are pulled from memory and printed to say “Enter Radius:”. Input is then taken. |
| <+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 is stored and multiplied by itself. The value of pie is then moved and multiplied by the previous value. An integer to floating point conversion is also done. |
| <+138>: callq 0x555555554890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 string, “The volume is: “ is printed along with the value calcluated earlier. The program then terminates. |
|  |  |
|  |  |

**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** |
| --- | --- | --- |
| <+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 0x555555554890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+42>: mov %rax,%rdx  <+45>: mov 0x2015a2(%rip),%rax # 0x201fd0  <+52>: mov %rax,%rsi  <+55>: mov %rdx,%rdi  <+58>: callq 0x5555555548a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+63>: lea -0x14(%rbp),%rax  <+67>: mov %rax,%rsi  <+70>: lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  <+77>: callq 0x555555554870 <std::istream::operator>>(int&)@plt> | Int r;  Cout << “Enter radius: “ << endl;  cin >> r; | The string “Enter radius: “ is printed to the terminal with an end line. Cin is then called to take input into variable r. |
| <+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;  const double PI = std::atan(1) \* 4;float vol = r \* pi;  double volume = r \* PI; | The radius is multiplied by itself to get the squared value. The value for pi is initialized. The volume of the cylinder is determined by multiplying r squared by pi. |
| <+138>: callq 0x555555554890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 | Cout << “The volume is: “ << vol << endl; | The string “The volume is: “ is printed to the terminal with the value of the volume calculated earlier. The program then terminates. |
|  |  |  |
|  |  |  |

## **File Three**

**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 $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 <std::cout@@GLIBCXX\_3.4>  <+44>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+49>: mov %rax,%rdx  <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  <+59>: mov %rax,%rsi  <+62>: mov %rdx,%rdi  <+65>: callq 0x870 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+70>: lea -0x18(%rbp),%rax  <+74>: mov %rax,%rsi  <+77>: lea 0x201732(%rip),%rdi # 0x202140 <std::cin@@GLIBCXX\_3.4> | The stack is initialized and the strings are loaded into the registers. The strings are then printed out to ther terminal to say “Enter number 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)  <+142>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+191>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+196>: addl $0x1,-0x14(%rbp)  <+200>: jmp 0xa5e <main+164>  <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <std::cout@@GLIBCXX\_3.4>  <+216>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+281>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+333>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+338>: addl $0x1,-0x14(%rbp)  <+342>: jmp 0xae9 <main+303>  <+344>: lea 0x135(%rip),%rsi # 0xc4e  <+351>: lea 0x201500(%rip),%rdi # 0x202020 <std::cout@@GLIBCXX\_3.4>  <+358>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+363>: addl $0x1,-0x10(%rbp)  <+367>: jmp 0xaab <main+241> | The input is taken and stored. A row of stars are outputted until the number of rows entered is reached. Then it increments back down to the initial row. The initial row has only one start and a start is added to each side for every additional row. When the max is reach is increments down and a start is taken away from each side so that the end result looks like 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 cleared and the program terminates. |
|  |  |
|  |  |

**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** |
| --- | --- | --- |
| <+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 <std::cout@@GLIBCXX\_3.4>  <+44>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+49>: mov %rax,%rdx  <+52>: mov 0x2015db(%rip),%rax # 0x201fd0  <+59>: mov %rax,%rsi  <+62>: mov %rdx,%rdi  <+65>: callq 0x870 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+70>: lea -0x18(%rbp),%rax  <+74>: mov %rax,%rsi  <+77>: lea 0x201732(%rip),%rdi # 0x202140 <std::cin@@GLIBCXX\_3.4> | std::cout << "Enter the number off rows" << std::endl; | The string “Enter the number of rows” is printed to the terminal with an end line. |
| <+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)  <+142>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+191>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+196>: addl $0x1,-0x14(%rbp)  <+200>: jmp 0xa5e <main+164>  <+202>: lea 0x1c3(%rip),%rsi # 0xc4e  <+209>: lea 0x20158e(%rip),%rdi # 0x202020 <std::cout@@GLIBCXX\_3.4>  <+216>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+281>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@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 <std::cout@@GLIBCXX\_3.4>  <+333>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+338>: addl $0x1,-0x14(%rbp)  <+342>: jmp 0xae9 <main+303>  <+344>: lea 0x135(%rip),%rsi # 0xc4e  <+351>: lea 0x201500(%rip),%rdi # 0x202020 <std::cout@@GLIBCXX\_3.4>  <+358>: callq 0x860 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+363>: addl $0x1,-0x10(%rbp)  <+367>: jmp 0xaab <main+241> | int rows;  cin >> rows;    for (int i = 0; i < rows; i++) whiteSpaceString = "";  starString = "";  stars = i \* 2 + 1;  whiteSpace = width - stars;  for (int j = 0; j <whiteSpace/2; j++) {  whiteSpaceString = whiteSpaceString + ' ';  }  for (int j = 0; j < stars; j++) {  starString = starString + '\*';  }  std::cout << whiteSpaceString << starString << whiteSpaceString << std::endl;  }  for (int i = rows - 2; i >= 0; i--) {  whiteSpaceString = "";  starString = "";  stars = i \* 2 + 1;  whiteSpace = width - stars;  for (int j = 0; j < whiteSpace / 2; j++) {  whiteSpaceString = whiteSpaceString + ' ';  }  for (int j = 0; j < stars; j++) starString = starString + '\*';  }  std::cout << whiteSpaceString << starString << whiteSpaceString << std::endl;  } | The user inputs the number of rows for the diamond. The program then iterates for each row and calculates the number of white space and starts. It then outputs to the terminal the row. |
| <+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 | return -1; | The program terminates |
|  |  |  |
|  |  |  |

## **File Four**

**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  <+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 <std::cout@@GLIBCXX\_3.4>  <+53>: callq 0x890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+58>: mov %rax,%rdx  <+61>: mov 0x201592(%rip),%rax # 0x201fd0  <+68>: mov %rax,%rsi  <+71>: mov %rdx,%rdi  <+74>: callq 0x8a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+79>: lea -0x28(%rbp),%rax  <+83>: mov %rax,%rsi  <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <std::cin@@GLIBCXX\_3.4>  <+93>: callq 0x870 <std::istream::operator>>(long&)@plt> | The stack is initialized. The string “Enter the binary number:” is printed to the terminal. Cin is then called to take the input and store it. |
| <+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 <std::cout@@GLIBCXX\_3.4>  <+256>: callq 0x890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+261>: mov %rax,%rdx  <+264>: mov -0x20(%rbp),%rax  <+268>: mov %rax,%rsi  <+271>: mov %rdx,%rdi  <+274>: callq 0x8d0 <std::ostream::operator<<(long)@plt>  <+279>: mov %rax,%rdx  <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  <+289>: mov %rax,%rsi  <+292>: mov %rdx,%rdi  <+295>: callq 0x8a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt> | The input is taken as a long integer. The bits are shifted to the right and left in order to determine the values of each section of binary. The values are then added and subtracted in order to determine the total value. The string “Equivalent hexadecimal value: “ is printed with the value determined earlier. |
| <+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 program terminates |
|  |  |
|  |  |

**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** |
| --- | --- | --- |
| <+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 <std::cout@@GLIBCXX\_3.4>  <+53>: callq 0x890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+58>: mov %rax,%rdx  <+61>: mov 0x201592(%rip),%rax # 0x201fd0  <+68>: mov %rax,%rsi  <+71>: mov %rdx,%rdi  <+74>: callq 0x8a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt>  <+79>: lea -0x28(%rbp),%rax  <+83>: mov %rax,%rsi  <+86>: lea 0x2016e9(%rip),%rdi # 0x202140 <std::cin@@GLIBCXX\_3.4>  <+93>: callq 0x870 <std::istream::operator>>(long&)@plt> | std::cout << "Enter the binary number: " << std::endl; | The program prints out the string “Enter the binary number: “ and then prints an endline. |
| <+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 <std::cout@@GLIBCXX\_3.4>  <+256>: callq 0x890 <std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)@plt>  <+261>: mov %rax,%rdx  <+264>: mov -0x20(%rbp),%rax  <+268>: mov %rax,%rsi  <+271>: mov %rdx,%rdi  <+274>: callq 0x8d0 <std::ostream::operator<<(long)@plt>  <+279>: mov %rax,%rdx  <+282>: mov 0x2014b5(%rip),%rax # 0x201fd0  <+289>: mov %rax,%rsi  <+292>: mov %rdx,%rdi  <+295>: callq 0x8a0 <std::ostream::operator<<(std::ostream& (\*)(std::ostream&))@plt> | long binary;  Long factor = 1;  Long total = 0;  std::cin >> binary;  while (binary != 0) {  total += (binary % 10) \* factor;  binary /= 10;  factor \*= 2;  }  std::cout << "Equivalent hexadecimal value: " << total; | The input is taken as a long integer. The value of the binary is determined and stored in total. The string is then outputted “Equivalent hexadecimal value: “ and then the total is printed. |
| <+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 | return -1; | Terminates program. |
|  |  |  |
|  |  |  |