# **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** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | This block of code pushes %rbp onto the stack, it then moves the stack pointer into it, and finally subtracts 10 from the stack pointer |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163> | This block of code stores the value of 1 in -0x8(%rbp), then compares -0x8(%rbp) to 9 and if it is greater than it, it will jump to 0xa3 |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154> | This block of code stores 1 in -0xc(%rbp), then compares if it is greater than 9, if it is jump to 0x9a |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x0(%rip),%rdi # 0x3c <main+60>  callq 0x41 <main+65> | This block of code starts by taking and copying the value -0x8(%rbp) into %eax, following this it multiplies it by the value held by -0xc(%rbp) and stores the product in -0x4(%rbp). It then copies the value held by -0x8(%rbp) into %eax again, then again into %esi, and finally prints the value held in %esi. |
| lea 0x0(%rip),%rsi # 0x48 <main+72>  mov %rax,%rdi  callq 0x50 <main+80> | This block of code prints from %rdi which currently points to 0x0(%rip) |
| mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96> | This block of code copies the value held in -0xc(%rbp) into %eax and then copies it again into %esi it then prints the value in %esi |
| lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111> | This block of code prints from %rdi which currently points to 0x0(%rip) |
| mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7f <main+127> | This block of code starts by copying the value from -0x4(%rbp) into %eax and then copies it again into %esi then prints the value in %esi |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x89 <main+137>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x94 <main+148> | This block of code prints a newline and clears the IOstream |
| addl $0x1,-0xc(%rbp)  jmp 0x20 <main+32> | This block of code increments the value in -0xc(%rbp) by 1 and jumps to 0x20 |
| addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | This block of code increments the value in -0x8(%rbp) by 1 and jumps to 0xf |
| mov $0x0,%eax  leaveq  retq | Returns 0 and exits the program |

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

See Attached files

**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** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | Int main (){  Int I, j, k, l; | Declares 4 Variables |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163> | For (I = 1; I <= 9; ++i){ | Defines the first variable and creates a loop that will repeat until I is greater than 9 |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154> | For (j = 1; j <= 9; ++j){ | Defines the second variable and creates a loop that will repeat until j is greater than 9 |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp) | K = I \* j | Multiplies I and J and stores the value in k |
| mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x0(%rip),%rdi # 0x3c <main+60>  callq 0x41 <main+65> | Cout << i | Prints the first variable |
| lea 0x0(%rip),%rsi # 0x48 <main+72>  mov %rax,%rdi  callq 0x50 <main+80> | Cout << “\*” | Prints the \* character |
| mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96> | Cout << j | Prints the second variable |
| lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111> | Cout << “=” | Prints the = character |
| mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7f <main+127> | Cout << k | Prints the third variable |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x89 <main+137>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x94 <main+148> | Cout << endl | Prints a newline |
| mov $0x0,%eax  leaveq  retq | Return 0; | Exits the program |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | This block of code pushes %rbp onto the stack, Subtracts 48 bytes from the stack pointer |
| lea 0x0(%rip),%rsi # 0x1e <main+30>  lea 0x0(%rip),%rdi # 0x25 <main+37>  callq 0x2a <main+42> | This block of code prints from %rdi which currently points to 0x0(%rip) |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x34 <main+52>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x3f <main+63> | This block of code prints end line |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x4d <main+77>  callq 0x52 <main+82> | This block of code reads input into %rsi which points to -0x14(%rbp) |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp) | This block of code copies the input from -0x14(%rbp) to %edx and %eax  Multiplies %eax by %edx and stores the result in %edx  Copies the value from -0x14(%rbp) into %eax again, then multiplies %edx by %eax and stores the product in -0x14(%rbp) |
| mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1 # 0x73 <main+115>  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | This block of code copies the value in -0x14(%rbp) to %eax, converts it to a double and stores the double in %xmm1, the copies the double in 0x0(%rip) to %xmm1, and then multiplies it by %xmm0 and stores the product in -0x10(%rbp) |
| lea 0x0(%rip),%rsi # 0x83 <main+131>  lea 0x0(%rip),%rdi # 0x8a <main+138>  callq 0x8f <main+143> | This block of code prints from %rdi which currently points to 0x0(%rip) |
| mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq 0xa7 <main+167> | This block of code prints the double stored in %xmm0 |
| mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0xc0 <main+192>  callq 0xc0 <main+192>  leaveq  retq | Manages the stack and returns 0 and exits the program |

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

See attached files

**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** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Int main() {  Int user\_in;  Int a;  Int b;  Double c;  Double pi; | Declares the local variables to be used in the program |
| lea 0x0(%rip),%rsi # 0x1e <main+30>  lea 0x0(%rip),%rdi # 0x25 <main+37>  callq 0x2a <main+42> | Cout << “Enter a number” | Prints the string “Enter a number” in order to obtain an input from a user |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x34 <main+52>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x3f <main+63> | Cout << endl | Prints a newline |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x4d <main+77>  callq 0x52 <main+82> | Cin >> user\_in; | Reads the input from the user into user\_in |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp) | A = user\_in;  B = user\_in;  A = a \* b;  B = b \* a; | Assigns the user input value to both the a and b variables, following that multiplies the input by itself 3 times and stores the final product of it in b |
| mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1 # 0x73 <main+115>  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | C = (double)b;  Pi = 3.14;  C = c \* pi; | Coverts the variable b into a double and stores it in variable c. Assigns pi with the value of 3.14, and then multiplies c and pi and assigns the product to c |
| lea 0x0(%rip),%rsi # 0x83 <main+131>  lea 0x0(%rip),%rdi # 0x8a <main+138>  callq 0x8f <main+143> | Cout<< “The result is” | Prints the string “The result is” |
| mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq 0xa7 <main+167> | Cout << c; | Prints out the result or the c variable |
| mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0xc0 <main+192>  callq 0xc0 <main+192>  leaveq  retq | Cout<< endl;  Return 0; | Prints a newline  Returns 0  Exits the program |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x20,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | This block of code pushes %rbp onto the stack, Subtracts 32 bytes from the stack pointer |
| movl $0x1,-0xc(%rbp)  lea 0x0(%rip),%rsi # 0x25 <main+37>  lea 0x0(%rip),%rdi # 0x2c <main+44>  callq 0x31 <main+49> | This block of code prints from %rdi which points to 0x0(%rip) |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x3b <main+59>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x46 <main+70> | This block of code prints a newline |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | This block of code reads the user input into %rsi which points to -0x18(%rbp) |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  mov %eax,-0xc(%rbp) | This block of code copies the value of 0x18(%rbp) to %eax, then subtract 1 from %eax and copy that new value into -0xc(%rbp) |
| movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227> | This block of code stores the value of 1 in -0x10(%rbp), then copies the value from -0x18($rbp) to %eax, then compares -0x10(%rbp) to %eax and if it is greater jump to 0xe3 |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153> | This block of code is for if the value is less than, it will store 1 in -0x14(%rbp), copy it to %eax, then compare %eax to -0xc(%rbp) and if it is greater then jump to 0x99 |
| lea 0x0(%rip),%rsi # 0x87 <main+135>  lea 0x0(%rip),%rdi # 0x8e <main+142>  callq 0x93 <main+147> | This block of code is for it is less than, in which case it will print from %rdi which points to 0x0(%rip) |
| addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120> | This block of code adds 1 to the value in -0x14(%rbp) and jump to 0x78 |
| subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp) | This block of code subtracts 1 from the value in -0xc(%rbp) and store the value of 1 in -0x14(%rbp) |
| mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202> | This block of code copies the value from -0x10(%rbp) to %eax, then double the value of %eax and subtract 1 from it. Then compare the value in -0x14(%rbp) to %eax and if it is greater jump to 0xca |
| lea 0x0(%rip),%rsi # 0xb8 <main+184>  lea 0x0(%rip),%rdi # 0xbf <main+191>  callq 0xc4 <main+196> | This block of code is for if the value is less than, print from %rdi which points to 0x0(%rip) |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164> | This block of code adds 1 to -0x14(%rbp) and jumps to 0xa4 |
| lea 0x0(%rip),%rsi # 0xd1 <main+209>  lea 0x0(%rip),%rdi # 0xd8 <main+216>  callq 0xdd <main+221> | This block of code prints from %rdi which points to 0x0(%rip) |
| addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105> | This block of code adds 1 to -0x10(%rbp) and jumps to 0x69 |
| movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> | This block of code makes -0xc(%rbp) and -0x10(%rbp) 1, then the value held by -0x18(%rbp) is copied by %eax, 1 is then subtracted from %eax, %eax is then compared against -0x10(%rbp) if the value is greater jump to 0x171 |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292> | This block of code makes -0x14(%rbp) 1 and copies the value held by -0x14(%rbp) to %eax then compares -0xc(%rbp) and %eax and if it is greater jump to 0x124 |
| lea 0x0(%rip),%rsi # 0x112 <main+274>  lea 0x0(%rip),%rdi # 0x119 <main+281>  callq 0x11e <main+286> | Prints out a string |
| addl $0x1,-0x14(%rbp)  jmp 0x103 <main+259> | adds 1 to -0x14(rbp), then jumps to 0x103 |
| addl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344> | Adds 1 to -0xc(%rbp) then stores 1 in -0x14(%rbp) and then copies the value in -0x18(%rbp) into %eax, subtracts the value held by -0x10(%rbp) by %eax, then doubles %eax, subtracts one from %eax, then compares %eax and -0x14(%rbp) and if it is greater jump to 0x158 |
| lea 0x0(%rip),%rsi # 0x146 <main+326>  lea 0x0(%rip),%rdi # 0x14d <main+333>  callq 0x152 <main+338> | Prints out a string |
| addl $0x1,-0x14(%rbp)  jmp 0x12f <main+303> | Adds 1 to -0x14(%rbp) and jumps to 0x12f |
| lea 0x0(%rip),%rsi # 0x15f <main+351>  lea 0x0(%rip),%rdi # 0x166 <main+358>  callq 0x16b <main+363> | Prints out a string |
| addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241> | Adds 1 to -0x10(%rbp) and then jumps to 0xf1 |
| mov $0x1,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0x18a <main+394>  callq 0x18a <main+394>  leaveq  retq | Return 0  Exits the program |

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

See attached files

**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** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x20,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  movl $0x1,-0xc(%rbp)  lea 0x0(%rip),%rsi # 0x25 <main+37>  lea 0x0(%rip),%rdi # 0x2c <main+44>  callq 0x31 <main+49> | Int a, b,c; | Creates three variables |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x3b <main+59>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x46 <main+70> | Cout << “enter a number” << endl; | Prints out a prompt for a user to input a number |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | Cin >> c | Gets user input and puts it under c |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  mov %eax,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227> | For(b=1; b<= c-a; b++){  For (b=1; b<=2\*a-1; b++)  Cout<< ”\*” << endl; | First loop to set amount of stars in a row |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  lea 0x0(%rip),%rsi # 0x87 <main+135>  lea 0x0(%rip),%rdi # 0x8e <main+142>  callq 0x93 <main+147> | For(a =1; a <=c -1; a++){ | Start of the second half of the diamond |
| addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120>  subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202>  lea 0x0(%rip),%rsi # 0xb8 <main+184>  lea 0x0(%rip),%rdi # 0xbf <main+191>  callq 0xc4 <main+196> | For(b=1; b<=a; b++){  Cout << “ “;  Cout << endl; |  |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164>  lea 0x0(%rip),%rsi # 0xd1 <main+209>  lea 0x0(%rip),%rdi # 0xd8 <main+216>  callq 0xdd <main+221>  addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105>  movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  lea 0x0(%rip),%rsi # 0x112 <main+274>  lea 0x0(%rip),%rdi # 0x119 <main+281>  callq 0x11e <main+286> | For(a =1; a<=c-1; a++){  For(d =1; d<= a; ++d){  Cout<<(“\*”}  } |  |
| addl $0x1,-0x14(%rbp)  jmp 0x103 <main+259>  addl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344>  lea 0x0(%rip),%rsi # 0x146 <main+326>  lea 0x0(%rip),%rdi # 0x14d <main+333>  callq 0x152 <main+338>  addl $0x1,-0x14(%rbp)  jmp 0x12f <main+303>  addl $0x1,-0x14(%rbp)  jmp 0x12f <main+303>  lea 0x0(%rip),%rsi # 0x15f <main+351>  lea 0x0(%rip),%rdi # 0x166 <main+358>  callq 0x16b <main+363> | for(b = 1; b< 2\*(c-a)-1; ++b){  cout << (“\*”);  }  Cout << endl; |  |
| addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241>  mov $0x1,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0x18a <main+394>  callq 0x18a <main+394>  leaveq  retq | Return 0; |  |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Pushes %rbp onto the stack, Subtracts 48 bytes from the stack pointer |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | Store value of 0 in -0x20(%rbp)  Store value of 1 in -0x18(%rbp) |
| lea 0x0(%rip),%rsi # 0x2e <main+46>  lea 0x0(%rip),%rdi # 0x35 <main+53>  callq 0x3a <main+58> | Prints from %rdi which points to 0x0(%rip) |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x44 <main+68>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | Prints a new line |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x5d <main+93>  callq 0x62 <main+98> | Reads input into %rsi which points to -0x28(%rbp) |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | Moves the input to %rax, tests it to see if it is >= 0 and if it does jumps to 0xf2 |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax | If %rax doesn’t equal 0 copies the value from –0x28(%rbp) into %rcx, then copies it to %rax. Moves the hex value of $0x6666666666666667 into %rdx |
| imul %rdx  sar $0x2,%rdx | Multiplies the values in %rdx by the value in %rax storing the product of the multiplication in %rdx, then divides %rdx by 4 |
| mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx | Copies the value in %rcx into %rax, shifts %rax to the right by 0x3f (63) bits and then subtracts %rax from %rdx, storing the result in %rdx, copies %rdx to %rax and -0x10(%rbp) |
| mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax | Copies the value of %rdx to %rax. Multiply %rax by 4, double the value in %rax and subtract it frm %rcx, storing the result in %rax, copies %rax to-0x10(%rbp) |
| imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp)  shlq -0x18(%rbp)  mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x28(%rbp)  jmpq 0x62 <main+98> | Multiply -0x18(%rbp) by %rax storing the product in %rax. Add %rax to the value in -0x20(%rbp), shift -18(%rbp), move -0x28(%rbp) into %rcx,  Moves the hex value of $0x6666666666666667 into %rdx, copies %rcx into %rax, multiplies %rdx, divides %rdx by 4, copy %rcx into %rax, subtract %rax and %rdx, copy the value in %rdx into %rax, copies %rax into -0x28(%rbp), then jumps to 0x62 |
| lea 0x0(%rip),%rsi # 0xf9 <main+249>  lea 0x0(%rip),%rdi # 0x100 <main+256>  callq 0x105 <main+261> | Prints a string “result” |
| mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279> | Calls the %rax variable to be printed |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x121 <main+289>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300> | Calls a new line to be printed |
| mov $0x0,%eax  mov -0x8(%rbp),%rsi  xor %fs:0x28,%rsi  je 0x145 <main+325>  callq 0x145 <main+325>  leaveq  retq | Return 0 and exits the program |

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

See Attached files

**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** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | A = 0;  B = 1;  C; | Initializes two variables with their own value, and makes another that is used for the user input |
| lea 0x0(%rip),%rsi # 0x2e <main+46>  lea 0x0(%rip),%rdi # 0x35 <main+53>  callq 0x3a <main+58>   mov %rax,%rdx  mov 0x0(%rip),%rax # 0x44 <main+68>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79>   lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x5d <main+93>  callq 0x62 <main+98> | Cout << “enter a number” << endl;  Cin>> c | Prompts the user to enter a number that will be stored, prints a new line and then accepts the user’s input under a variable |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | While(c != 0){ | Begins the while test and if it is found as equal it will jump to 0xf2 |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp)  shlq -0x18(%rbp)  mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x28(%rbp)  jmpq 0x62 <main+98> | Int d = c % 10;  A += d \* b;  B \*= 2;  C /= 10; |  |
| lea 0x0(%rip),%rsi # 0xf9 <main+249>  lea 0x0(%rip),%rdi # 0x100 <main+256>  callq 0x105 <main+261> | Cout << “Result: “ | First part of the print statement |
| mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279> | Cout << a; | Prints the variable |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x121 <main+289>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300> | Cout << endl; | Prints a new line |
| mov $0x0,%eax  mov -0x8(%rbp),%rsi  xor %fs:0x28,%rsi  je 0x145 <main+325>  callq 0x145 <main+325>  leaveq  retq | Return 0; | Returns 0 and exits the program |