# **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 | Reserve 16 bytes |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg a3 <main+0xa3>  movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 9a <main+0x9a>  mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax | Move 1 into the variable and compare to create a loop and compare the value with 9.  Create inner loop compare the value with 9  Compute the multiplication of the two values |
| mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x0(%rip),%rdi # 3c <main+0x3c>  callq 41 <main+0x41>  lea 0x0(%rip),%rsi # 48 <main+0x48>  mov %rax,%rdi  callq 50 <main+0x50>  mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 60 <main+0x60>  lea 0x0(%rip),%rsi # 67 <main+0x67>  mov %rax,%rdi  callq 6f <main+0x6f>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 7f <main+0x7f> | Get the address of %d + %d = %d and print out the multilocation result of the 2 variables.  Increment the inner loop variable and compare if it reached 9, repeat the logic.  Jump to the address of the Main loop and repeat the logic. |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 89 <main+0x89>  mov %rax,%rsi  mov %rdx,%rdi  callq 94 <main+0x94>  addl $0x1,-0xc(%rbp)  jmp 20 <main+0x20>  addl $0x1,-0x8(%rbp)  jmpq f <main+0xf>  mov $0x0,%eax |  |
| leaveq  retq |  |

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

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**int number, i, a, x;**

**for (a = 1; a <= 9; a++)**

**{**

**for (i = 1; i <= 9; i++){**

**x = a \* i;**

**cout << a << " \* " << i << " = " << x << endl;**

**}**

**}**

**return 0;**

**}**

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | **int number, i, a, x;** | Reserve 16 bytes |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg a3 <main+0xa3>  movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 9a <main+0x9a>  mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x0(%rip),%rdi # 3c <main+0x3c>  callq 41 <main+0x41>  lea 0x0(%rip),%rsi # 48 <main+0x48>  mov %rax,%rdi  callq 50 <main+0x50>  mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 60 <main+0x60>  lea 0x0(%rip),%rsi # 67 <main+0x67>  mov %rax,%rdi  callq 6f <main+0x6f>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 7f <main+0x7f>  mov %rax,%rdx  mov 0x0(%rip),%rax # 89 <main+0x89>  mov %rax,%rsi  mov %rdx,%rdi  callq 94 <main+0x94>  addl $0x1,-0xc(%rbp)  jmp 20 <main+0x20>  addl $0x1,-0x8(%rbp)  jmpq f <main+0xf> | **for (a = 1; a <= 9; a++)**  **{**  **for (i = 1; i <= 9; i++){**  **x = a \* i;**  **cout << a << " \* " << i << " = " << x << endl;**  **}**  **}** | Assign the values and start the main loop and repeat the inner loop logic print out the multiplication result. |
| mov $0x0,%eax | **return 0;** |  |
| leaveq  retq | } |  |

## **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) | Allocate 48 bytes |
| xor %eax,%eax  lea 0x0(%rip),%rsi # 1e <main+0x1e>  lea 0x0(%rip),%rdi # 25 <main+0x25>  callq 2a <main+0x2a>  mov %rax,%rdx  mov 0x0(%rip),%rax # 34 <main+0x34>  mov %rax,%rsi  mov %rdx,%rdi  callq 3f <main+0x3f>  lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 4d <main+0x4d>  callq 52 <main+0x52> | Get the address of “Enter Radius:” and read the value from the user and store it |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1 # 73 <main+0x73> | Move or Merge Scalar double precision  b = b \* a \* a \* 10; |
| mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi # 83 <main+0x83>  lea 0x0(%rip),%rdi # 8a <main+0x8a>  callq 8f <main+0x8f>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq a7 <main+0xa7>  mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je c0 <main+0xc0> |  |
| callq c0 <main+0xc0>  leaveq  retq | End of procedure |

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

**#include <iostream>**

**using namespace std;**

**void main(){**

**double a = 0, b = 3.14;**

**cout << “Enter Radius:”<< endl;**

**cin >> a;**

**b = b \* a \* a \* 10;**

**cout << “The volume is: “ << b << endl;**

**}**

**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) | double a = 0, b = 3.14; | Reserve double variables. |
| xor %eax,%eax  lea 0x0(%rip),%rsi # 1e <main+0x1e>  lea 0x0(%rip),%rdi # 25 <main+0x25>  callq 2a <main+0x2a>  mov %rax,%rdx  mov 0x0(%rip),%rax # 34 <main+0x34>  mov %rax,%rsi  mov %rdx,%rdi  callq 3f <main+0x3f>  lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 4d <main+0x4d>  callq 52 <main+0x52> | cout << “Enter Radius:”<< endl;   cin >> a; | Print out and fetch the value from the user. |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1 # 73 <main+0x73>   mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi # 83 <main+0x83>  lea 0x0(%rip),%rdi # 8a <main+0x8a>  callq 8f <main+0x8f>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq a7 <main+0xa7>  mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je c0 <main+0xc0>   callq c0 <main+0xc0>  leaveq  retq | b = b \* a \* a \* 10;   cout << “The volume is: “ << b << endl; | Compute the volume and print out the value and finish the execution. |

## **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 | Reserve 32 bytes |
| mov %rax,-0x8(%rbp)  xor %eax,%eax  movl $0x1,-0xc(%rbp)  lea 0x0(%rip),%rsi # 25 <main+0x25>  lea 0x0(%rip),%rdi # 2c <main+0x2c>  callq 31 <main+0x31>  mov %rax,%rdx  mov 0x0(%rip),%rax # 3b <main+0x3b>  mov %rax,%rsi  mov %rdx,%rdi  callq 46 <main+0x46> | Assign 1 to a variable and print out “Enter the number of rows”  Fetch the value from the user. |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 54 <main+0x54>  callq 59 <main+0x59>  mov -0x18(%rbp),%eax  sub $0x1,%eax  mov %eax,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg e3 <main+0xe3>  movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 99 <main+0x99>  lea 0x0(%rip),%rsi # 87 <main+0x87>  lea 0x0(%rip),%rdi # 8e <main+0x8e>  callq 93 <main+0x93> |  |
| addl $0x1,-0x14(%rbp)  jmp 78 <main+0x78>  subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg ca <main+0xca>  lea 0x0(%rip),%rsi # b8 <main+0xb8>  lea 0x0(%rip),%rdi # bf <main+0xbf>  callq c4 <main+0xc4>  addl $0x1,-0x14(%rbp)  jmp a4 <main+0xa4>  lea 0x0(%rip),%rsi # d1 <main+0xd1>  lea 0x0(%rip),%rdi # d8 <main+0xd8>  callq dd <main+0xdd> |  |
| addl $0x1,-0x10(%rbp)  jmp 69 <main+0x69>  movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 171 <main+0x171>  movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 124 <main+0x124>  lea 0x0(%rip),%rsi # 112 <main+0x112>  lea 0x0(%rip),%rdi # 119 <main+0x119>  callq 11e <main+0x11e>  addl $0x1,-0x14(%rbp)  jmp 103 <main+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 158 <main+0x158>  lea 0x0(%rip),%rsi # 146 <main+0x146>  lea 0x0(%rip),%rdi # 14d <main+0x14d>  callq 152 <main+0x152> |  |
| addl $0x1,-0x14(%rbp)  jmp 12f <main+0x12f>  lea 0x0(%rip),%rsi # 15f <main+0x15f>  lea 0x0(%rip),%rdi # 166 <main+0x166>  callq 16b <main+0x16b>  addl $0x1,-0x10(%rbp)  jmp f1 <main+0xf1>  mov $0x1,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 18a <main+0x18a>  callq 18a <main+0x18a>  leaveq  retq |  |

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

#include <iostream>

using namespace std;

void main(){

int a = 1;

cout << “Enter number of rows” << endl;

cin >> a;

}

**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** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## **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 | Reserve 48 bytes |
| mov %rax,-0x8(%rbp)  xor %eax,%eax  movq $0x0,-0x20(%rbp) | Assign 0 to long variable |
| movq $0x1,-0x18(%rbp)  lea 0x0(%rip),%rsi # 2e <main+0x2e>  lea 0x0(%rip),%rdi # 35 <main+0x35>  callq 3a <main+0x3a>  mov %rax,%rdx  mov 0x0(%rip),%rax # 44 <main+0x44>  mov %rax,%rsi  mov %rdx,%rdi  callq 4f <main+0x4f>  lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 5d <main+0x5d>  callq 62 <main+0x62> | Assign 1 to variable2  Print out “Enter the binary number:”  store the value in another variable |
| mov -0x28(%rbp),%rax  test %rax,%rax  je f2 <main+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 | Store 6666…. Into RDX  Multiple and shit right store shift right subtract |
| mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x28(%rbp)  jmpq 62 <main+0x62>  lea 0x0(%rip),%rsi # f9 <main+0xf9>  lea 0x0(%rip),%rdi # 100 <main+0x100>  callq 105 <main+0x105>  mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 117 <main+0x117>  mov %rax,%rdx  mov 0x0(%rip),%rax # 121 <main+0x121>  mov %rax,%rsi  mov %rdx,%rdi  callq 12c <main+0x12c>  mov $0x0,%eax  mov -0x8(%rbp),%rsi  xor %fs:0x28,%rsi |  |
| je 145 <main+0x145>  callq 145 <main+0x145>  leaveq  retq | Exit the proc |

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

#include <iostream>

using namespace std;

void main(){

long num =0,res=1 ;

cout << “Enter the binary number:” << endl;

cin >> num;

// convertion logic

cout << endl;

cout << “Equivalent hexadecimal value: << value;

}

**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** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## NOTE: I was able to extract the rdata (Data Segment) where string data used in the program.