# **CS 410 Binary to C++ Activity Template**

Janera Dobson

Southern New Hampshire University

CS-410-T3249

## **File One**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163>  addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | Assign the value 1 to -8(%rbp)  Compare the value in -8(%rbp) with 9  Conditional jump to <main+163> |
| cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154> | Compare the value 1 to 9 in %rbp  Conditional jump to <main+ 152> |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp) | Move the value of x at -8(%rbp) to %eax  Multiply value at -8(%rbp) by %eax  Move %eax new value to -4(%rbp) |
| mov -0x8(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96>  lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %rdx,%rdi  callq 0x7f <main+127>  mov %rax,%rdx  mov 0x0(%rip),%rax # 0x89 <main+137>  mov %rdx,%rdi  callq 0x94 <main+148> | Move -8(%rbp) to %eax  Now move %eax to %esi to prepare for print  Callq prints statement on x = a \* i  This enters the loop again in which values are moved to %rdx, then %eax, to %esi, and then %rdi in order to call callq function to print values on stack console. |
| mov  $0x0,%eax  leaveq  retq | Return and exit program |

**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** |
| --- | --- | --- |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163>  addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | **for (a = 1; a <= 9; a++)** | Both codes go to show how 1 was moved into the register to compare it to 9 and if true it adds on but if not, it jumps to the next for loop. |
| cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154>  mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp) | **for (i = 1; i <= 9; i++)**  **x = a \* i;** | Both shows how the comparison happens again within the for loop and then moved to the location of x and multiplied and returned back to the register. |
| mov -0x8(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96>  lea 0x0(%rip),%rsi  mov %rax,%rdi  callq 0x6f <main+111>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %rdx,%rdi  callq 0x7f <main+127>  mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rdx,%rdi  callq 0x94 <main+148> | **cout << a << " \* " << i << " = " << x << endl;** | These codes show how each component of the cout called is moved to the registers and stored in %rdi to prepare for its final call of callq. |
| mov  $0x0,%eax  leaveq  retq | **return 0;** | Both moves 0 to the %eax and exits the program. |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi  callq 0x52 <main+82> | Load -14(%rbp) in %rax  Move %rax in %rsi  Load %rdi  Calls callq |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax | Move -14(%rbp) in %edx  Move -14(%rbp) in %eax  Multiply %eax and %edx  Move -14(%rbp) in %eax |
| imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x0(%rip),%xmm1  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x8f <main+143> | Multiply %edx,%eax  Move %eax to -14(%rbp)  Move -14(%rbp) in %eax  Convert double %eax to %xmm0 register  Move double to %xmm1 register  Mulitply %xmm1 to %xmm0  Move double %xmm0 to -10(%rbp)  Load 0 into %rsi  Load 0 in %rdi  Calls callq |
| mov %rdx,%rdi  callq 0xa7 <main+167> | Move %rdx in %rdi  Call callq |
| leaveq  retg | Return and exit program |

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

**#include <iostream>**

**#define PI 3.14159**

**using namespace std;**

**int main()**

**{**

**float radius, area, circum;**

**cout<<"Enter Radius:\n";**

**cin>>radius;**

**circum = 2\*PI\*radius;**

**area = PI\*(radius\*radius);**

**cout << 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** |
| --- | --- | --- |
| Load -14(%rbp) in %rax  Move %rax in %rsi  Load %rdi  Calls callq | **cout<<"Enter Radius:\n";**  **cin>>radius;** | Shows how the string is loaded in the %rax, then the value is the moved and loaded, then finally printed via callq. |
| Move -14(%rbp) in %edx  Move -14(%rbp) in %eax  Multiply %eax and %edx  Move -14(%rbp) in %eax | **circum = 2\*PI\*radius;** | Here the variable circum is given the value, which was moved from the parts of the equations, multiplied and replaced in the %eax. |
| Multiply %edx,%eax  Move %eax to -14(%rbp)  Move -14(%rbp) in %eax  Convert double %eax to %xmm0 register  Move double to %xmm1 register  Mulitply %xmm1 to %xmm0  Move double %xmm0 to -10(%rbp)  Load 0 into %rsi  Load 0 in %rdi  Calls callq  Move %rdx in %rdi  Call callq | **area = PI\*(radius\*radius);**  **cout << endl;** | Here another mathematical equation is formulated and what was multiplied by the %edx and %eax then converted to a double and moved to the %xmm0 register. Then another multiplications problem occurs doing the same thing, but this time multiplied by the %xmm1 to %xmm0. Once done, moved to -10(%rbp), loaded into the %rdi in order to be called by the callq. |
| leaveq  retg | **return 0;** | Both moves 0 to the %eax and exits the program. |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | Loads -18(%rbp) in %rax  Moves %rax to %rsi  Loads 0 in %rdi  Calls callq |
| mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227>  movl $0x1,-0x14(%rbp)  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  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>  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  callq 0xc4 <main+196> | Move -18(%rbp) to %eax  Compare %eax to the -10(%rbp) register  Conditional jump to <main+227>  Move 1 in -14(%rbp)  Add 1 in -10(%rbp)  Move -14(%rbp) to %eax  Compare values in %rbp to %eax  Conditional jump to <main+292>  Move -14(%rbp) to %eax  Compare %rbp to %eax  Conditional jump to <main+153>  Add 1 to -14(%rbp) register  Jumps to <main+120>  Subtract 1 from %rbp  Move 1 to -14(%rbp)  Move -10(%rbp) to %eax  Add %eax to %eax  Sub 1 from %eax  Comapre %eax to -14(%rbp)  Conditional jump to <main+202>  Add 1 to -10(%rbp)  Move -14(%rbp) to %eax  Compare %rbp to %eax  Conditional jump to <main+292>  Calls callq |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164>  addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105>  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x11e <main+286> | Add 1 to -14(%rbp)  Jumps to <main+164>  Add 1 to -10(%rbp)  Jumps to <main+105>  Compare %eax to -10(%rbp)  Conditional jump to <main+369>  Load 0 in %rsi and the %rdi  Call callq |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344>  addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241> | Move -18(%rbp) to %eax  Subtract 1 from %eax  Compare %eax to -10(%rbp)  Conditional jump to <main+369>  Subtract 1 to %eax  Compare %eax to -14(%rbp)  Conditional jump to <main+344>  Add 1 to -10(%rbp)  Jumps to <main+241> |

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

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int i,j,r;**

**cout << "Enter number of rows\n";**

**cin >> r;**

**for(i=0;i<=r;i++)**

**{**

**for(j=1;j<=r-i;j++)**

**cout<<" ";**

**for(j=1;j<=2\*i-1;j++)**

**cout<<"\*";**

**cout<<endl;**

**}**

**for(i=r-1;i>=1;i--)**

**{**

**for(j=1;j<=r-i;j++)**

**cout<<" ";**

**for(j=1;j<=2\*i-1;j++)**

**cout<<"\*";**

**cout<<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** |
| --- | --- | --- |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | **cout << "Enter number of rows\n";**  **cin >> r;** | Loads string in -18(%rbp) to %rax. Moves the new value of %rax to %rsi which is the cin value and the prints via callq. |
| mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227>  movl $0x1,-0x14(%rbp)  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  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>  addl $0x1,-0x10(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  callq 0xc4 <main+196> | **for(i=0;i<=r;i++)**  **{**  **for(j=1;j<=r-i;j++)**  **cout<<" ";** | Both codes show the functionality of these two for loops. Moving values in and comparing them followed by a test. If false, it jumps to the next conditional statement, if true you add and compare again until it reaches a false condition. The same goes for the next for loop and the conditions will continue to execute as long as the statement is true, but once false it jumps to the next conditional statement. Followed by callq to print cout statement. |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164>  addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105>  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x11e <main+286> | **for(j=1;j<=2\*i-1;j++)**  **cout<<"\*";**  **cout<<endl;**  **}** | Here the for loop adds the %rbp and jumps to the next conditional test. It does so again and compares the values added until it reaches false and prints cout via callq. |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344>  addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241> | **for(i=r-1;i>=1;i--)**  **{**  **for(j=1;j<=r-i;j++)**  **cout<<" ";**  **for(j=1;j<=2\*i-1;j++)**  **cout<<"\*";**  **cout<<endl;**  **}**  **}** | These codes show how the values in the for loop at -18(%rbp) was moved to %eax, the subtracted and compared to the new %eax. Then jumps to next conditional test which subtracts and then compares the loop and doing it again until it jumps back to <main+241>. |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp)  callq 0x3a <main+58>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | Move 0 in -20(%rbp)  Move 1 in -18(%rbp)  Calls callq  Move %rax in %rsi  Move %rdx in %rdi  Calls calls |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi | Load -28(%rbp) in %rax  Move %rax in %rsi  Load 0 in %rdi |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | Move -28(%rbp) in %rax  Test %rax in %rax  Jump if equal to <main+242> |
| 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 | Move -28(%rbp) in %rcx  Move abs value 6666666666666667 in %rdx  Move %rcx in %rax  Multiply %rdx  Shift right 2 in %rx  Move %rcx in %rax  Shift right 3 in %rax  Subtract %rax in %rdx  Move %rdx in %rax  Move %rax in -10(%rbp)  Move -10(%rbp) in %rdx  Move %rdx in %rax  Shift left 2 in %rax  Add %rdx in %rax  Add %rax in %rax  Subtract %rax in %rcx  Move %rcx in %rax  Move %rax in -10(%rbp)  Move -18(%rbp) in %rax  Multiply -18(%rbp) iin %rax  Add %rax in -20(%rbp)  Shift -18(%rbp)  Move -28(%rbp) in %rcx  Move abs value 6666666666666667 in %rdx  Move %rcx in %rax  Multiply %rdx  Shift right 2 in %rdx  Move %rcx in %rax |
| mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279>  mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300>  leaveq  retq | Move %rax in %rdx  Move -20(%rbp) in %rax  Move %rax in %rsi  Move %rdx in %rdi  Calls callq  Mov %rax in %rdx  Move 0 in %rax  Move %rax in %rsi  Move %rdx in %rdi  Calls callq  Return |

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

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**int main()**

**{**

**long long num;**

**int decimalNum, i, rem;**

**cout << "Enter the binary number:\n";**

**cin >> num;**

**decimalNum = 0;**

**i = 0;**

**//converting binary to decimal**

**while (num != 0)**

**{**

**rem = num % 10;**

**num /= 10;**

**decimalNum += rem \* pow(2, i);**

**++i;**

**}**

**cout << "Equivalent hexadecimal value: " << decimalNum;**

**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** |
| --- | --- | --- |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp)  callq 0x3a <main+58>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | **cout << "Enter the binary number:\n";**  **cin >> num;** | Codes moves the request string into %rbp and prints via callq. The moves values around in register to the print value provided by user. |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi | **decimalNum = 0;**  **i = 0;** | Initializes two variables with 0 in stores it in %rdi. |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | **while (num != 0)**  **{** | Begins while test and jumps if equal to <main+242>. |
| 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 | **rem = num % 10;**  **num /= 10;**  **decimalNum += rem \* pow(2, i);**  **++i;** | Here the codes are looking to move the values in the %rcx and find the absolute value in the %rdx regiater. From there %rdx is multiplied followed by some shifts in the registers in order to get the mathematical equations in position to move and shift again before trying to evaluate the absolute value of pow and moved into the %rax. |
| mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279>  mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300>  leaveq  retq | **cout << "Equivalent hexadecimal value: " << decimalNum;**  **return 0;** | Here the %rax is moved into %rdx which prepares the string to called and printed via allq. Lastly, the variable decimalNum id moved to the %rdi and printed via callq to complete the entire cout statement. Followed by return to exit the program. |

**gdb assignment4\_1.o**

(gdb) info functions

All defined functions:

Non-debugging symbols:

0x0000000000000000 main

0x00000000000000aa \_\_static\_initialization\_and\_destruction\_0(int, int)

0x00000000000000f3 \_GLOBAL\_\_sub\_I\_main

(gdb) disassemble main

Dump of assembler code for function main:

  0x0000000000000000 <+0>:  push %rbp

  0x0000000000000001 <+1>:  mov  %rsp,%rbp

  0x0000000000000004 <+4>:  sub  $0x10,%rsp

  0x0000000000000008 <+8>:  movl $0x1,-0x8(%rbp)

  0x000000000000000f <+15>:  cmpl $0x9,-0x8(%rbp)

  0x0000000000000013 <+19>:  jg  0xa3 <main+163>

  0x0000000000000019 <+25>:  movl $0x1,-0xc(%rbp)

  0x0000000000000020 <+32>:  cmpl $0x9,-0xc(%rbp)

  0x0000000000000024 <+36>:  jg  0x9a <main+154>

  0x0000000000000026 <+38>:  mov  -0x8(%rbp),%eax

  0x0000000000000029 <+41>:  imul -0xc(%rbp),%eax

  0x000000000000002d <+45>:  mov  %eax,-0x4(%rbp)

  0x0000000000000030 <+48>:  mov  -0x8(%rbp),%eax

  0x0000000000000033 <+51>:  mov  %eax,%esi

  0x0000000000000035 <+53>:  lea  0x0(%rip),%rdi    # 0x3c <main+60>

  0x000000000000003c <+60>:  callq 0x41 <main+65>

  0x0000000000000041 <+65>:  lea  0x0(%rip),%rsi    # 0x48 <main+72>

  0x0000000000000048 <+72>:  mov  %rax,%rdi

  0x000000000000004b <+75>:  callq 0x50 <main+80>

  0x0000000000000050 <+80>:  mov  %rax,%rdx

  0x0000000000000053 <+83>:  mov  -0xc(%rbp),%eax

  0x0000000000000056 <+86>:  mov  %eax,%esi

  0x0000000000000058 <+88>:  mov  %rdx,%rdi

  0x000000000000005b <+91>:  callq 0x60 <main+96>

  0x0000000000000060 <+96>:  lea  0x0(%rip),%rsi    # 0x67 <main+103>

  0x0000000000000067 <+103>: mov  %rax,%rdi

  0x000000000000006a <+106>: callq 0x6f <main+111>

  0x000000000000006f <+111>: mov  %rax,%rdx

  0x0000000000000072 <+114>: mov  -0x4(%rbp),%eax

  0x0000000000000075 <+117>: mov  %eax,%esi

  0x0000000000000077 <+119>: mov  %rdx,%rdi

  0x000000000000007a <+122>: callq 0x7f <main+127>

  0x000000000000007f <+127>: mov  %rax,%rdx

  0x0000000000000082 <+130>: mov  0x0(%rip),%rax    # 0x89 <main+137>

  0x0000000000000089 <+137>: mov  %rax,%rsi

  0x000000000000008c <+140>: mov  %rdx,%rdi

  0x000000000000008f <+143>: callq 0x94 <main+148>

  0x0000000000000094 <+148>: addl $0x1,-0xc(%rbp)

  0x0000000000000098 <+152>: jmp  0x20 <main+32>

  0x000000000000009a <+154>: addl $0x1,-0x8(%rbp)

  0x000000000000009e <+158>: jmpq 0xf <main+15>

  0x00000000000000a3 <+163>: mov  $0x0,%eax

  0x00000000000000a8 <+168>: leaveq

  0x00000000000000a9 <+169>: retq

---Type <return> to continue, or q <return> to quit---return

End of assembler dump.

(gdb)

**gdb assignment4\_2.o**

(gdb) info functions

All defined functions:

Non-debugging symbols:

0x0000000000000000 main

0x00000000000000c2 \_\_static\_initialization\_and\_destruction\_0(int, int)

0x000000000000010b \_GLOBAL\_\_sub\_I\_main

(gdb) disassemble main

Dump of assembler code for function main:

  0x0000000000000000 <+0>:  push %rbp

  0x0000000000000001 <+1>:  mov  %rsp,%rbp

  0x0000000000000004 <+4>:  sub  $0x30,%rsp

  0x0000000000000008 <+8>:  mov  %fs:0x28,%rax

  0x0000000000000011 <+17>:  mov  %rax,-0x8(%rbp)

  0x0000000000000015 <+21>:  xor  %eax,%eax

  0x0000000000000017 <+23>:  lea  0x0(%rip),%rsi    # 0x1e <main+30>

  0x000000000000001e <+30>:  lea  0x0(%rip),%rdi    # 0x25 <main+37>

  0x0000000000000025 <+37>:  callq 0x2a <main+42>

  0x000000000000002a <+42>:  mov  %rax,%rdx

  0x000000000000002d <+45>:  mov  0x0(%rip),%rax    # 0x34 <main+52>

  0x0000000000000034 <+52>:  mov  %rax,%rsi

  0x0000000000000037 <+55>:  mov  %rdx,%rdi

  0x000000000000003a <+58>:  callq 0x3f <main+63>

  0x000000000000003f <+63>:  lea  -0x14(%rbp),%rax

  0x0000000000000043 <+67>:  mov  %rax,%rsi

  0x0000000000000046 <+70>:  lea  0x0(%rip),%rdi    # 0x4d <main+77>

  0x000000000000004d <+77>:  callq 0x52 <main+82>

  0x0000000000000052 <+82>:  mov  -0x14(%rbp),%edx

  0x0000000000000055 <+85>:  mov  -0x14(%rbp),%eax

  0x0000000000000058 <+88>:  imul %eax,%edx

  0x000000000000005b <+91>:  mov  -0x14(%rbp),%eax

  0x000000000000005e <+94>:  imul %edx,%eax

  0x0000000000000061 <+97>:  mov  %eax,-0x14(%rbp)

  0x0000000000000064 <+100>: mov  -0x14(%rbp),%eax

  0x0000000000000067 <+103>: cvtsi2sd %eax,%xmm0

  0x000000000000006b <+107>: movsd 0x0(%rip),%xmm1    # 0x73 <main+115>

  0x0000000000000073 <+115>: mulsd %xmm1,%xmm0

  0x0000000000000077 <+119>: movsd %xmm0,-0x10(%rbp)

  0x000000000000007c <+124>: lea  0x0(%rip),%rsi    # 0x83 <main+131>

  0x0000000000000083 <+131>: lea  0x0(%rip),%rdi    # 0x8a <main+138>

  0x000000000000008a <+138>: callq 0x8f <main+143>

  0x000000000000008f <+143>: mov  %rax,%rdx

  0x0000000000000092 <+146>: mov  -0x10(%rbp),%rax

  0x0000000000000096 <+150>: mov  %rax,-0x28(%rbp)

  0x000000000000009a <+154>: movsd -0x28(%rbp),%xmm0

  0x000000000000009f <+159>: mov  %rdx,%rdi

  0x00000000000000a2 <+162>: callq 0xa7 <main+167>

  0x00000000000000a7 <+167>: mov  $0x0,%eax

  0x00000000000000ac <+172>: mov  -0x8(%rbp),%rcx

  0x00000000000000b0 <+176>: xor  %fs:0x28,%rcx

  0x00000000000000b9 <+185>: je  0xc0 <main+192>

  0x00000000000000bb <+187>: callq 0xc0 <main+192>

  0x00000000000000c0 <+192>: leaveq

---Type <return> to continue, or q <return> to quit---return

  0x00000000000000c1 <+193>: retq

End of assembler dump.

(gdb)

**gdb assignment4\_3.o**

(gdb) info functions

All defined functions:

Non-debugging symbols:

0x0000000000000000 main

0x000000000000018c \_\_static\_initialization\_and\_destruction\_0(int, int)

0x00000000000001d5 \_GLOBAL\_\_sub\_I\_main

(gdb) disassemble main

Dump of assembler code for function main:

  0x0000000000000000 <+0>:  push %rbp

  0x0000000000000001 <+1>:  mov  %rsp,%rbp

  0x0000000000000004 <+4>:  sub  $0x20,%rsp

  0x0000000000000008 <+8>:  mov  %fs:0x28,%rax

  0x0000000000000011 <+17>:  mov  %rax,-0x8(%rbp)

  0x0000000000000015 <+21>:  xor  %eax,%eax

  0x0000000000000017 <+23>:  movl $0x1,-0xc(%rbp)

  0x000000000000001e <+30>:  lea  0x0(%rip),%rsi    # 0x25 <main+37>

  0x0000000000000025 <+37>:  lea  0x0(%rip),%rdi    # 0x2c <main+44>

  0x000000000000002c <+44>:  callq 0x31 <main+49>

  0x0000000000000031 <+49>:  mov  %rax,%rdx

  0x0000000000000034 <+52>:  mov  0x0(%rip),%rax    # 0x3b <main+59>

  0x000000000000003b <+59>:  mov  %rax,%rsi

  0x000000000000003e <+62>:  mov  %rdx,%rdi

  0x0000000000000041 <+65>:  callq 0x46 <main+70>

  0x0000000000000046 <+70>:  lea  -0x18(%rbp),%rax

  0x000000000000004a <+74>:  mov  %rax,%rsi

  0x000000000000004d <+77>:  lea  0x0(%rip),%rdi    # 0x54 <main+84>

  0x0000000000000054 <+84>:  callq 0x59 <main+89>

  0x0000000000000059 <+89>:  mov  -0x18(%rbp),%eax

  0x000000000000005c <+92>:  sub  $0x1,%eax

  0x000000000000005f <+95>:  mov  %eax,-0xc(%rbp)

  0x0000000000000062 <+98>:  movl $0x1,-0x10(%rbp)

  0x0000000000000069 <+105>: mov  -0x18(%rbp),%eax

  0x000000000000006c <+108>: cmp  %eax,-0x10(%rbp)

  0x000000000000006f <+111>: jg  0xe3 <main+227>

  0x0000000000000071 <+113>: movl $0x1,-0x14(%rbp)

  0x0000000000000078 <+120>: mov  -0x14(%rbp),%eax

  0x000000000000007b <+123>: cmp  -0xc(%rbp),%eax

  0x000000000000007e <+126>: jg  0x99 <main+153>

  0x0000000000000080 <+128>: lea  0x0(%rip),%rsi    # 0x87 <main+135>

  0x0000000000000087 <+135>: lea  0x0(%rip),%rdi    # 0x8e <main+142>

  0x000000000000008e <+142>: callq 0x93 <main+147>

  0x0000000000000093 <+147>: addl $0x1,-0x14(%rbp)

  0x0000000000000097 <+151>: jmp  0x78 <main+120>

  0x0000000000000099 <+153>: subl $0x1,-0xc(%rbp)

  0x000000000000009d <+157>: movl $0x1,-0x14(%rbp)

  0x00000000000000a4 <+164>: mov  -0x10(%rbp),%eax

  0x00000000000000a7 <+167>: add  %eax,%eax

  0x00000000000000a9 <+169>: sub  $0x1,%eax

  0x00000000000000ac <+172>: cmp  %eax,-0x14(%rbp)

  0x00000000000000af <+175>: jg  0xca <main+202>

  0x00000000000000b1 <+177>: lea  0x0(%rip),%rsi    # 0xb8 <main+184>

  0x00000000000000b8 <+184>: lea  0x0(%rip),%rdi    # 0xbf <main+191>

---Type <return> to continue, or q <return> to quit---return

  0x00000000000000bf <+191>: callq 0xc4 <main+196>

  0x00000000000000c4 <+196>: addl $0x1,-0x14(%rbp)

  0x00000000000000c8 <+200>: jmp  0xa4 <main+164>

  0x00000000000000ca <+202>: lea  0x0(%rip),%rsi    # 0xd1 <main+209>

  0x00000000000000d1 <+209>: lea  0x0(%rip),%rdi    # 0xd8 <main+216>

  0x00000000000000d8 <+216>: callq 0xdd <main+221>

  0x00000000000000dd <+221>: addl $0x1,-0x10(%rbp)

  0x00000000000000e1 <+225>: jmp  0x69 <main+105>

  0x00000000000000e3 <+227>: movl $0x1,-0xc(%rbp)

  0x00000000000000ea <+234>: movl $0x1,-0x10(%rbp)

  0x00000000000000f1 <+241>: mov  -0x18(%rbp),%eax

  0x00000000000000f4 <+244>: sub  $0x1,%eax

  0x00000000000000f7 <+247>: cmp  %eax,-0x10(%rbp)

  0x00000000000000fa <+250>: jg  0x171 <main+369>

  0x00000000000000fc <+252>: movl $0x1,-0x14(%rbp)

  0x0000000000000103 <+259>: mov  -0x14(%rbp),%eax

  0x0000000000000106 <+262>: cmp  -0xc(%rbp),%eax

  0x0000000000000109 <+265>: jg  0x124 <main+292>

  0x000000000000010b <+267>: lea  0x0(%rip),%rsi    # 0x112 <main+274>

  0x0000000000000112 <+274>: lea  0x0(%rip),%rdi    # 0x119 <main+281>

  0x0000000000000119 <+281>: callq 0x11e <main+286>

  0x000000000000011e <+286>: addl $0x1,-0x14(%rbp)

  0x0000000000000122 <+290>: jmp  0x103 <main+259>

  0x0000000000000124 <+292>: addl $0x1,-0xc(%rbp)

  0x0000000000000128 <+296>: movl $0x1,-0x14(%rbp)

  0x000000000000012f <+303>: mov  -0x18(%rbp),%eax

  0x0000000000000132 <+306>: sub  -0x10(%rbp),%eax

  0x0000000000000135 <+309>: add  %eax,%eax

  0x0000000000000137 <+311>: sub  $0x1,%eax

  0x000000000000013a <+314>: cmp  %eax,-0x14(%rbp)

  0x000000000000013d <+317>: jg  0x158 <main+344>

  0x000000000000013f <+319>: lea  0x0(%rip),%rsi    # 0x146 <main+326>

  0x0000000000000146 <+326>: lea  0x0(%rip),%rdi    # 0x14d <main+333>

  0x000000000000014d <+333>: callq 0x152 <main+338>

  0x0000000000000152 <+338>: addl $0x1,-0x14(%rbp)

  0x0000000000000156 <+342>: jmp  0x12f <main+303>

  0x0000000000000158 <+344>: lea  0x0(%rip),%rsi    # 0x15f <main+351>

  0x000000000000015f <+351>: lea  0x0(%rip),%rdi    # 0x166 <main+358>

  0x0000000000000166 <+358>: callq 0x16b <main+363>

  0x000000000000016b <+363>: addl $0x1,-0x10(%rbp)

  0x000000000000016f <+367>: jmp  0xf1 <main+241>

  0x0000000000000171 <+369>: mov  $0x1,%eax

  0x0000000000000176 <+374>: mov  -0x8(%rbp),%rcx

  0x000000000000017a <+378>: xor  %fs:0x28,%rcx

  0x0000000000000183 <+387>: je  0x18a <main+394>

---Type <return> to continue, or q <return> to quit---

**gdb assignment4\_4.o**

(gdb) info function

All defined functions:

Non-debugging symbols:

0x0000000000000000 main

0x0000000000000147 \_\_static\_initialization\_and\_destruction\_0(int, int)

0x0000000000000190 \_GLOBAL\_\_sub\_I\_main

(gdb) disassemble main

Dump of assembler code for function main:

  0x0000000000000000 <+0>:  push %rbp

  0x0000000000000001 <+1>:  mov  %rsp,%rbp

  0x0000000000000004 <+4>:  sub  $0x30,%rsp

  0x0000000000000008 <+8>:  mov  %fs:0x28,%rax

  0x0000000000000011 <+17>:  mov  %rax,-0x8(%rbp)

  0x0000000000000015 <+21>:  xor  %eax,%eax

  0x0000000000000017 <+23>:  movq $0x0,-0x20(%rbp)

  0x000000000000001f <+31>:  movq $0x1,-0x18(%rbp)

  0x0000000000000027 <+39>:  lea  0x0(%rip),%rsi    # 0x2e <main+46>

  0x000000000000002e <+46>:  lea  0x0(%rip),%rdi    # 0x35 <main+53>

  0x0000000000000035 <+53>:  callq 0x3a <main+58>

  0x000000000000003a <+58>:  mov  %rax,%rdx

  0x000000000000003d <+61>:  mov  0x0(%rip),%rax    # 0x44 <main+68>

  0x0000000000000044 <+68>:  mov  %rax,%rsi

  0x0000000000000047 <+71>:  mov  %rdx,%rdi

  0x000000000000004a <+74>:  callq 0x4f <main+79>

  0x000000000000004f <+79>:  lea  -0x28(%rbp),%rax

  0x0000000000000053 <+83>:  mov  %rax,%rsi

  0x0000000000000056 <+86>:  lea  0x0(%rip),%rdi    # 0x5d <main+93>

  0x000000000000005d <+93>:  callq 0x62 <main+98>

  0x0000000000000062 <+98>:  mov  -0x28(%rbp),%rax

  0x0000000000000066 <+102>: test %rax,%rax

  0x0000000000000069 <+105>: je  0xf2 <main+242>

  0x000000000000006f <+111>: mov  -0x28(%rbp),%rcx

  0x0000000000000073 <+115>: movabs $0x6666666666666667,%rdx

  0x000000000000007d <+125>: mov  %rcx,%rax

  0x0000000000000080 <+128>: imul %rdx

  0x0000000000000083 <+131>: sar  $0x2,%rdx

  0x0000000000000087 <+135>: mov  %rcx,%rax

  0x000000000000008a <+138>: sar  $0x3f,%rax

  0x000000000000008e <+142>: sub  %rax,%rdx

  0x0000000000000091 <+145>: mov  %rdx,%rax

  0x0000000000000094 <+148>: mov  %rax,-0x10(%rbp)

  0x0000000000000098 <+152>: mov  -0x10(%rbp),%rdx

  0x000000000000009c <+156>: mov  %rdx,%rax

  0x000000000000009f <+159>: shl  $0x2,%rax

  0x00000000000000a3 <+163>: add  %rdx,%rax

  0x00000000000000a6 <+166>: add  %rax,%rax

  0x00000000000000a9 <+169>: sub  %rax,%rcx

  0x00000000000000ac <+172>: mov  %rcx,%rax

  0x00000000000000af <+175>: mov  %rax,-0x10(%rbp)

  0x00000000000000b3 <+179>: mov  -0x10(%rbp),%rax

  0x00000000000000b7 <+183>: imul -0x18(%rbp),%rax

  0x00000000000000bc <+188>: add  %rax,-0x20(%rbp)

---Type <return> to continue, or q <return> to quit---return

  0x00000000000000c0 <+192>: shlq -0x18(%rbp)

  0x00000000000000c4 <+196>: mov  -0x28(%rbp),%rcx

  0x00000000000000c8 <+200>: movabs $0x6666666666666667,%rdx

  0x00000000000000d2 <+210>: mov  %rcx,%rax

  0x00000000000000d5 <+213>: imul %rdx

  0x00000000000000d8 <+216>: sar  $0x2,%rdx

  0x00000000000000dc <+220>: mov  %rcx,%rax

  0x00000000000000df <+223>: sar  $0x3f,%rax

  0x00000000000000e3 <+227>: sub  %rax,%rdx

  0x00000000000000e6 <+230>: mov  %rdx,%rax

  0x00000000000000e9 <+233>: mov  %rax,-0x28(%rbp)

  0x00000000000000ed <+237>: jmpq 0x62 <main+98>

  0x00000000000000f2 <+242>: lea  0x0(%rip),%rsi    # 0xf9 <main+249>

  0x00000000000000f9 <+249>: lea  0x0(%rip),%rdi    # 0x100 <main+256>

  0x0000000000000100 <+256>: callq 0x105 <main+261>

  0x0000000000000105 <+261>: mov  %rax,%rdx

  0x0000000000000108 <+264>: mov  -0x20(%rbp),%rax

  0x000000000000010c <+268>: mov  %rax,%rsi

  0x000000000000010f <+271>: mov  %rdx,%rdi

  0x0000000000000112 <+274>: callq 0x117 <main+279>

  0x0000000000000117 <+279>: mov  %rax,%rdx

  0x000000000000011a <+282>: mov  0x0(%rip),%rax    # 0x121 <main+289>

  0x0000000000000121 <+289>: mov  %rax,%rsi

  0x0000000000000124 <+292>: mov  %rdx,%rdi

  0x0000000000000127 <+295>: callq 0x12c <main+300>

  0x000000000000012c <+300>: mov  $0x0,%eax

  0x0000000000000131 <+305>: mov  -0x8(%rbp),%rsi

  0x0000000000000135 <+309>: xor  %fs:0x28,%rsi

  0x000000000000013e <+318>: je  0x145 <main+325>

  0x0000000000000140 <+320>: callq 0x145 <main+325>

  0x0000000000000145 <+325>: leaveq

  0x0000000000000146 <+326>: retq

End of assembler dump.

(gdb)