# CS 410 Project One Proficiency Test Template

## Explain the functionality of the blocks of assembly code.

### “main” function”

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| 1. 0x0000000000000000 <+0>: push %rbp 2. 0x0000000000000001 <+1>: mov %rsp,%rbp | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. |
| 1. 0x0000000000000004 <+4>:   lea 0x0(%rip),%rsi # 0xb <main+11>   1. 0x000000000000000b <+11>:   lea 0x0(%rip),%rdi # 0x12 <main+18>   1. 0x0000000000000012 <+18>: callq 0x17 <main+23> 2. 0x0000000000000017 <+23>: callq 0x1c <main+28> | 1. This loads the effective address 0x0(%rip), the next instruction into the %rsi register, the second argument register used in the output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following output function call. 3. This calls the function to output the string: “Welcome to our investment Company.” 4. This calls the output function to output a new line. |
| 1. 0x000000000000001c <+28>: mov %eax,0x0(%rip) # 0x22 <main+34> 2. 0x0000000000000022 <+34>: mov 0x0(%rip),%eax # 0x28 <main+40> 3. 0x0000000000000028 <+40>: cmp $0x1,%eax 4. 0x000000000000002b <+43>:   je 0x40 <main+64> | 1. This moves the value stored in the %eax register (the value of a variable to determine whether the user is a valid user) and stores this value in the 0x0%(rip) register. 2. This moves the value stored at the location 0x0(%rip) into the %eax register. 3. This instruction compares the value 1 with the value stored in the %eax register. 4. Based on the previously made comparison, this instruction jumps the execution of the program to address 0x40, which is line 64 of the main function. |
| 1. 0x000000000000002d <+45>: lea 0x0(%rip),%rsi # 0x34 <main+52> 2. 0x0000000000000034 <+52>: lea 0x0(%rip),%rdi # 0x3b <main+59> 3. 0x000000000000003b <+59>: callq 0x40 <main+64> | 1. This loads the effective address 0x0(%rip), the next instruction, and stores its value into the %rsi, the second argument register to be used in the function call to CheckUserPermissionAccess(). 2. This loads the the address 0x0(%rip), the next instruction, into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function CheckUserPermissionAccess and stores its result in the variable validUser. This is a while loop and continues until the user is successfully authenticated. |
| 1. 0x0000000000000040 <+64>: mov 0x0(%rip),%eax # 0x46 <main+70> 2. 0x0000000000000046 <+70>: cmp $0x1,%eax 3. 0x0000000000000049 <+73>:   je 0x4d <main+77> | 1. This moves the value at the address 0x0(%rip), the next instruction and is also the value returned by the previous function call inside the while loop, into the %eax register to be used in the following comparison to determine whether the while loop should be exited. 2. This compares the value 1 with the value previously stored in the %eax register. 3. Based on the previous comparison, if the value stored in the %eax register is equal to 1, this line instructs the program’s execution to jump to address 0x4d, which is line 77 of the main function, essentially exiting the first program loop. |
| 1. 0x000000000000004b <+75>: jmp 0x17 <main+23> | 1. This line is reached if the previous comparison did not equal 1. This instructs the program’s execution to jump to address 0x17, which is line 23 of the main function. This is the beginning of the first while loop for the program to continue calling the function CheckUserPermissionAccess. |
| 1. 0x000000000000004d <+77>: lea 0x0(%rip),%rsi # 0x54 <main+84> 2. 0x0000000000000054 <+84>: lea 0x0(%rip),%rdi # 0x5b <main+91> 3. 0x000000000000005b <+91>: callq 0x60 <main+96> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the function to output the following string: “What would you like to do? “ |
| 1. 0x0000000000000060 <+96>: lea 0x0(%rip),%rsi # 0x67 <main+103> 2. 0x0000000000000067 <+103>: lea 0x0(%rip),%rdi # 0x6e <main+110> 3. 0x000000000000006e <+110>: callq 0x73 <main+115> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the function to output the following string: “DISPLAY the client’s list (enter 1)” |
| 1. 0x0000000000000073 <+115>: lea 0x0(%rip),%rsi # 0x7a <main+122> 2. 0x000000000000007a <+122>: lea 0x0(%rip),%rdi # 0x81 <main+129> 3. 0x0000000000000081 <+129>: callq 0x86 <main+134> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the function to output the following string: “CHANGE a client’s choice (enter 2)” |
| 1. 0x0000000000000086 <+134>: lea 0x0(%rip),%rsi # 0x8d <main+141> 2. 0x000000000000008d <+141>: lea 0x0(%rip),%rdi # 0x94 <main+148> 3. 0x0000000000000094 <+148>: callq 0x99 <main+153> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the function to output the following string: “EXIT the program (enter 3)” |
| 1. 0x0000000000000099 <+153>: lea 0x0(%rip),%rsi # 0xa0 <main+160> 2. 0x00000000000000a0 <+160>: lea 0x0(%rip),%rdi # 0xa7 <main+167> 3. 0x00000000000000a7 <+167>: callq 0xac <main+172> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the output function at the address 0xac |
| 1. 0x00000000000000ac <+172>: lea 0x0(%rip),%rsi # 0xb3 <main+179> 2. 0x00000000000000b3 <+179>: lea 0x0(%rip),%rdi # 0xba <main+186> 3. 0x00000000000000ba <+186>: callq 0xbf <main+191> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, for use in the next output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following output function call. 3. This calls the output function at the address 0xbf |
| 1. 0x00000000000000bf <+191>: mov %rax,%rdx 2. 0x00000000000000c2 <+194>: mov 0x0(%rip),%eax # 0xc8 <main+200> 3. 0x00000000000000c8 <+200>: mov %eax,%esi 4. 0x00000000000000ca <+202>: mov %rdx,%rdi 5. 0x00000000000000cd <+205>: callq 0xd2 <main+210> | 1. This moves the value stored in the %rax register into the %rdx register to prepare for the input function call. 2. This moves the value stored at address 0x0(%rip), the next instruction, and the variable for the user choice, into the %eax register to prepare for the input function call. 3. This instruction moves the value stored in the %eax register into the %esi register, the source index in preparation for the input function call. 4. This instruction moves the value stored in the %rdx register into the %rdi register, the first argument register used in the following input function call. 5. This calls the function to receive from input the value for the variable choice. |
| 1. 0x00000000000000d2 <+210>: mov %rax,%rdx 2. 0x00000000000000d5 <+213>: mov 0x0(%rip),%rax # 0xdc <main+220> 3. 0x00000000000000dc <+220>: mov %rax,%rsi 4. 0x00000000000000df <+223>: mov %rdx,%rdi 5. 0x00000000000000e2 <+226>: callq 0xe7 <main+231> | 1. This moves the value stored in the %rax register into the %rdx register to be used in the output function call. 2. This moves the value stored at the address 0x0(%rip), the next instruction, intot he %eax register. 3. This moves the value previously stored in the %eax register into the %esi register, the source index register in preparation for the output function call. 4. This instruction moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This instruction calls the output function to output the string “You chose “ followed by the value taken in from the previous input function call. |
| 1. 0x00000000000000e7 <+231>: mov 0x0(%rip),%eax # 0xed <main+237> 2. 0x00000000000000ed <+237>: cmp $0x1,%eax 3. 0x00000000000000f0 <+240>: jne 0xf9 <main+249> | 1. This instruction moves the value stored at the address 0c0(%rip), the next instruction, into the %eax register to be used in the following comparison. This is the value of the choice variable. 2. This compares the value 1 with the value previously stored in the %eax register. 3. Based on the previous comparison, this instruction jumps the program’s execution if the comparison does not equal the value 1, to the address 0xf9, which is line 249 of the main function. |
| 1. 0x00000000000000f2 <+242>: callq 0xf7 <main+247> 2. 0x00000000000000f7 <+247>: jmp 0x109 <main+265> | 1. If based on the previous comparison, the value did equal 1, this line is reached and calls the function at the address 0xf7, which is the Displayinfo function. 2. This line is returned to once the previous function call has been completed. This line instructs the program’s execution to jump to address 0x109, which is line 265 of the main function. This is where the next comparison will be made to determine whether or not the second program loop should be exited. |
| 1. 0x00000000000000f9 <+249>: mov 0x0(%rip),%eax # 0xff <main+255> 2. 0x00000000000000ff <+255>: cmp $0x2,%eax 3. 0x0000000000000102 <+258>: jne 0x109 <main+265> | 1. This instruction moves the value at the address 0x0(%rip), the next instruction, into the %eax register to be used in the following comparison. This is the value of the choice variable. 2. This instruction compares the value 2 with the value previously stored in the %eax register. 3. Based on the previous comparison, this instruction jumps the program’s execution if the comparison did not equal the value 2, to the address 0x109, which is line 265 of the main function. |
| 1. 0x0000000000000104 <+260>: callq 0x109 <main+265> | 1. This line is reached if the value from the previous comparison does equal to. This instruction calls the function ChangeCustomerChoice(). One the execution of the ChangeCustomerChoice function completes, the program returns to the following instruction at the address 0x109 or line 265 of the main function. |
| 1. 0x0000000000000109 <+265>: mov 0x0(%rip),%eax # 0x10f <main+271> 2. 0x000000000000010f <+271>: cmp $0x3,%eax 3. 0x0000000000000112 <+274>: je 0x119 <main+281> | 1. This moves the value at the address 0x0(%rip), the next instruction, into the %eax register, which is the value of the choice variable, in preparation for the next comparison. 2. This compares the value 3 with the value stored in the %eax register. 3. If based from the previous comparison the value in the %eax register is equal to the value 3, this instructs the program’s execution to jump to address 0x119, which is line 281 of the main function, effectively exiting the second while loop. |
| 1. 0x0000000000000114 <+276>: jmpq 0x4d <main+77> | 1. If from the previous comparison, the value in the %eax register did not equal three, this instruction is then executed. This is an unconditional instruction to jump to the address 0x4d, which is line 77 of the main function, which is also the beginning of the second control loop or while loop for the main function. |
| 1. 0x0000000000000119 <+281>: mov $0x0,%eax 2. 0x000000000000011e <+286>: pop %rbp 3. 0x000000000000011f <+287>: retq | 1. This moves the value 0 into the %eax register, setting the main functions return value. 2. This instruction restored the value of the base pointer register. 3. This instruction returns from the main function. |

### ChangeCustomerChoice function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| 1. 0x000000000000042d <+0>:   push %rbp   1. 0x000000000000042e <+1>:   mov %rsp,%rbp | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. |
| 1. 0x0000000000000431 <+4>:   lea 0x0(%rip),%rsi # 0x438 <\_Z20ChangeCustomerChoicev+11>   1. 0x0000000000000438 <+11>:   lea 0x0(%rip),%rdi # 0x43f <\_Z20ChangeCustomerChoicev+18>   1. 0x000000000000043f <+18>:   callq 0x444 <\_Z20ChangeCustomerChoicev+23> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register in preparation for the output function call. 2. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register in preparation for the following output function call. 3. This calls the function to output the string: “Enter the number of the client that you wish to change.” follwoed by a new line. |
| 1. 0x0000000000000444 <+23>:   lea 0x0(%rip),%rsi # 0x44b <\_Z20ChangeCustomerChoicev+30>   1. 0x000000000000044b <+30>:   lea 0x0(%rip),%rdi # 0x452 <\_Z20ChangeCustomerChoicev+37>   1. 0x0000000000000452 <+37>:   callq 0x457 <\_Z20ChangeCustomerChoicev+42> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used with the input function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used in the following input function call. 3. This calls the input function to receive from input the value for the clientNumber variable. |
| 1. 0x0000000000000457 <+42>:   lea 0x0(%rip),%rsi # 0x45e <\_Z20ChangeCustomerChoicev+49>   1. 0x000000000000045e <+49>:   lea 0x0(%rip),%rdi # 0x465 <\_Z20ChangeCustomerChoicev+56>   1. 0x0000000000000465 <+56>:   callq 0x46a <\_Z20ChangeCustomerChoicev+61> | 1. This instruction loads the effective address 0x0(%rip) into the %rsi register, the second argument register, to be used in the following output function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register, to be used in the following output function call. 3. This calls the output function to display the string: “Please enter the client’s new service choice (1 = brokerage, 2 = Retirement)”, followed by a new line. |
| 1. 0x000000000000046a <+61>:   lea 0x0(%rip),%rsi # 0x471 <\_Z20ChangeCustomerChoicev+68>   1. 0x0000000000000471 <+68>:   lea 0x0(%rip),%rdi # 0x478 <\_Z20ChangeCustomerChoicev+75>   1. 0x0000000000000478 <+75>:   callq 0x47d <\_Z20ChangeCustomerChoicev+80> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register, to be used with the following input function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used with the following input function call. 3. This calls the input function to receive from input the value for the serviceNumber variable. |
| 1. 0x000000000000047d <+80>:   mov 0x0(%rip),%eax # 0x483 <\_Z20ChangeCustomerChoicev+86>   1. 0x0000000000000483 <+86>:   cmp $0x1,%eax   1. 0x0000000000000486 <+89>:   jne 0x496 <\_Z20ChangeCustomerChoicev+105> | 1. This moves the value stored at 0x0(rip), the next instruction, which is also the clientNumber variable, into the %eax register to prepare for the following comparison. 2. This instruction comapres the value 1 with the value previously stored in the %eax register, the clientNumber variable. 3. Based on the previously made comparison, if the value is not equal to 1, this instruction jumps the programs execution to address 0x496, which is line 105 of the ChangeCustomerChoice function. |
| 1. 0x0000000000000488 <+91>:   mov 0x0(%rip),%eax # 0x48e <\_Z20ChangeCustomerChoicev+97>   1. 0x000000000000048e <+97>:   mov %eax,0x0(%rip) # 0x494 <\_Z20ChangeCustomerChoicev+103>   1. 0x0000000000000494 <+103>:   jmp 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. If from the previous comparison, the value in the %eax register (the clientNumber variable) was equal to 1, this instruction is reached. This moves the value stored at the address 0x0(%rip) into the %eax register, to prepare for use in the following instruction. 2. The client associated with the clientNumber that was given in input, is assigned the new serviceNumber that was given at input. This value is then taken from the %eax register, and moved back to the address at 0x0(%rip). 3. Once the previous assignment has been made, this instruction jumps the program’s execution to address 0x4f8, which is line 203 of the ChangeCustomerChoice function, where the function ends. |
| 1. 0x0000000000000496 <+105>:   mov 0x0(%rip),%eax # 0x49c <\_Z20ChangeCustomerChoicev+111>   1. 0x000000000000049c <+111>:   cmp $0x2,%eax   1. 0x000000000000049f <+114>:   jne 0x4af <\_Z20ChangeCustomerChoicev+130> | 1. This instruction moves the value at address 0x0(%rip), the next instruction into the %eax register (the clientNumber variable), in preparation for use in the following comparison. 2. This compares the value 2 with the value previously stored in the %eax register. 3. If from the previous comparison, the value in the %eax register is not equal to the value 2, this instruction jumps the program’s execution to address 0x4af, which is line 130 of the ChangeCustomerChoice function, where the next comparison will be prepared. |
| 1. 0x00000000000004a1 <+116>:   mov 0x0(%rip),%eax # 0x4a7 <\_Z20ChangeCustomerChoicev+122>   1. 0x00000000000004a7 <+122>:   mov %eax,0x0(%rip) # 0x4ad <\_Z20ChangeCustomerChoicev+128>   1. 0x00000000000004ad <+128>:   jmp 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. If from the previous comparison, the value in the %eax register (the clientNumber variable) was equal to 2, this instruction is reached. This moves the value stored at the address 0x0(%rip) into the %eax register, to prepare for use in the following instruction. 2. The client associated with the clientNumber that was given in input, is assigned the new serviceNumber that was given at input. This value is then taken from the %eax register, and moved back to the address at 0x0(%rip). 3. Once the previous assignment has been made, this instruction jumps the program’s execution to address 0x4f8, which is line 203 of the ChangeCustomerChoice function, where the function ends. |
| 1. 0x00000000000004af <+130>:   mov 0x0(%rip),%eax # 0x4b5 <\_Z20ChangeCustomerChoicev+136>   1. 0x00000000000004b5 <+136>:   cmp $0x3,%eax   1. 0x00000000000004b8 <+139>:   jne 0x4c8 <\_Z20ChangeCustomerChoicev+155> | 1. This instruction moves the value at address 0x0(%rip), the next instruction into the %eax register (the clientNumber variable), in preparation for use in the following comparison. 2. This compares the value 3 with the value previously stored in the %eax register. 3. If from the previous comparison, the value in the %eax register is not equal to the value 3, this instruction jumps the program’s execution to address 0x4c8, which is line 155 of the ChangeCustomerChoice function, where the next comparison will be prepared. |
| 1. 0x00000000000004ba <+141>:   mov 0x0(%rip),%eax # 0x4c0 <\_Z20ChangeCustomerChoicev+147>   1. 0x00000000000004c0 <+147>:   mov %eax,0x0(%rip) # 0x4c6 <\_Z20ChangeCustomerChoicev+153>   1. 0x00000000000004c6 <+153>:   jmp 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. If from the previous comparison, the value in the %eax register (the clientNumber variable) was equal to 3, this instruction is reached. This moves the value stored at the address 0x0(%rip) into the %eax register, to prepare for use in the following instruction. 2. The client associated with the clientNumber that was given in input, is assigned the new serviceNumber that was given at input. This value is then taken from the %eax register, and moved back to the address at 0x0(%rip). 3. Once the previous assignment has been made, this instruction jumps the program’s execution to address 0x4f8, which is line 203 of the ChangeCustomerChoice function, where the function ends. |
| 1. 0x00000000000004c8 <+155>:   mov 0x0(%rip),%eax # 0x4ce <\_Z20ChangeCustomerChoicev+161>   1. 0x00000000000004ce <+161>:   cmp $0x4,%eax   1. 0x00000000000004d1 <+164>:   jne 0x4e1 <\_Z20ChangeCustomerChoicev+180> | 1. This instruction moves the value at address 0x0(%rip), the next instruction into the %eax register (the clientNumber variable), in preparation for use in the following comparison. 2. This compares the value 4 with the value previously stored in the %eax register. 3. If from the previous comparison, the value in the %eax register is not equal to the value 4, this instruction jumps the program’s execution to address 0x4e1, which is line 180 of the ChangeCustomerChoice function, where the next comparison will be prepared. |
| 1. 0x00000000000004d3 <+166>:   mov 0x0(%rip),%eax # 0x4d9 <\_Z20ChangeCustomerChoicev+172>   1. 0x00000000000004d9 <+172>:   mov %eax,0x0(%rip) # 0x4df <\_Z20ChangeCustomerChoicev+178>   1. 0x00000000000004df <+178>:   jmp 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. If from the previous comparison, the value in the %eax register (the clientNumber variable) was equal to 4, this instruction is reached. This moves the value stored at the address 0x0(%rip) into the %eax register, to prepare for use in the following instruction. 2. The client associated with the clientNumber that was given in input, is assigned the new serviceNumber that was given at input. This value is then taken from the %eax register, and moved back to the address at 0x0(%rip). 3. Once the previous assignment has been made, this instruction jumps the program’s execution to address 0x4f8, which is line 203 of the ChangeCustomerChoice function, where the function ends. |
| 1. 0x00000000000004e1 <+180>:   mov 0x0(%rip),%eax # 0x4e7 <\_Z20ChangeCustomerChoicev+186>   1. 0x00000000000004e7 <+186>:   cmp $0x5,%eax   1. 0x00000000000004ea <+189>:   jne 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. This instruction moves the value at address 0x0(%rip), the next instruction into the %eax register (the clientNumber variable), in preparation for use in the following comparison. 2. This compares the value 5 with the value previously stored in the %eax register. 3. If from the previous comparison, the value in the %eax register is not equal to the value 5, this instruction jumps the program’s execution to address 0x4f8, which is line 203 of the ChangeCustomerChoice function, where no further comparisons will be made, and the function will end. |
| 1. 0x00000000000004ec <+191>: mov 0x0(%rip),%eax # 0x4f2 <\_Z20ChangeCustomerChoicev+197> 2. 0x00000000000004f2 <+197>: mov %eax,0x0(%rip) # 0x4f8 <\_Z20ChangeCustomerChoicev+203> | 1. If from the previous comparison, the value in the %eax register (the clientNumber variable) was equal to 5, this instruction is reached. This moves the value stored at the address 0x0(%rip) into the %eax register, to prepare for use in the following instruction. 2. The client associated with the clientNumber that was given in input, is assigned the new serviceNumber that was given at input. This value is then taken from the %eax register, and moved back to the address at 0x0(%rip). |
| 1. 0x00000000000004f8 <+203>:   nop   1. 0x00000000000004f9 <+204>:   pop %rbp   1. 0x00000000000004fa <+205>:   retq | 1. This instruction is a No Operation instruction, which eassentially does nothing, but instructs the execution to move to the next instruction without making any changes to register values, memory locations or flags. 2. This instruction restored the value of the base pointer register. 3. This instruction returns from the ChangeCustomerChoice function. |

### CheckUserPermissonAccess Function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| 1. 0x0000000000000120 <+0>:   push %rbp   1. 0x0000000000000121 <+1>:   mov %rsp,%rbp   1. 0x0000000000000124 <+4>:   push %rbx   1. 0x0000000000000125 <+5>:   sub $0x48,%rsp   1. 0x0000000000000129 <+9>:   mov %fs:0x28,%rax   1. 0x0000000000000132 <+18>:   mov %rax,-0x18(%rbp)   1. 0x0000000000000136 <+22>:   xor %eax,%eax   1. 0x0000000000000138 <+24>:   lea -0x45(%rbp),%rax   1. 0x000000000000013c <+28>:   mov %rax,%rdi   1. 0x000000000000013f <+31>:   callq 0x144 <\_Z25CheckUserPermissionAccessv+36> | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. 3. This instruction pushes the %rbx register onto the stack to save its value. 4. This subtracts (0x48) which in decimal for is 72 from the stack pointer, allocating 72 bytes of memory for local variables. 5. This moves the value at the offset 0x28 in the FS segment register into the %rax register. 6. This moves the value in the %rax register to the address -0x18(%rbp), (0x18 in decimal form is 24), -24 offset or 24 bytes above the base pointer register. 7. This instruction makes an XOR comparison on the value stored in the %eax register with itself, effectively clearing out the %eax register. 8. This instruction loads the effective address -0x45(%rbp), (0x45) in decimal is 69, which is an -69 offset or 69 bytes above the base pointer register, into the %rax register. 9. This moves the previously stored value in the %rax register into the %rdi register, the first argument register for the following function call. 10. This calls the function located at address 0x144 in preparation for input and output function calls. |
| 1. 0x0000000000000144 <+36>:   lea -0x45(%rbp),%rdx   1. 0x0000000000000148 <+40>:   lea -0x40(%rbp),%rax   1. 0x000000000000014c <+44>:   lea 0x0(%rip),%rsi # 0x153 <\_Z25CheckUserPermissionAccessv+51>   1. 0x0000000000000153 <+51>:   mov %rax,%rdi   1. 0x0000000000000156 <+54>:   callq 0x15b <\_Z25CheckUserPermissionAccessv+59> | These instructions involve loading addresses and making function calls, likely related to input/output operations for username and password   1. This loads the effective address -0x45(%rbp) into the %rdx register. This is the global variable username. 2. This loads the effective address -0x40(%rbp) into the %rax register to prepare it for use. This is the variable containing the correctPassword. 3. This moves the value at address 0x0(%rip), the next instruction into the %esi register, the second argument register, which will be used in the following function call. 4. This moves the value stored in the %rax register into the %rdi register, the first argument register to be used in the following function call. 5. This calls the function at address 0x15b to output the string “Enter your username: “. |
| 1. 0x000000000000015b <+59>:   lea -0x45(%rbp),%rax   1. 0x000000000000015f <+63>:   mov %rax,%rdi   1. 0x0000000000000162 <+66>:   callq 0x167 <\_Z25CheckUserPermissionAccessv+71> | 1. This instruction loads the effective address -0x45(%rbp) into the %rax register. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register in preparation for the following function call. 3. This instruction calls the input function to receive input and store the value in the username variable. |
| 1. 0x0000000000000167 <+71>:   movl $0x0,-0x44(%rbp) | 1. This instruction moves the value 0 into the address -0x44(%rbp). This is essentially initializing the password variable that will be taken in from user input. |
| 1. 0x000000000000016e <+78>:   lea 0x0(%rip),%rsi # 0x175 <\_Z25CheckUserPermissionAccessv+85>   1. 0x0000000000000175 <+85>:   lea 0x0(%rip),%rdi # 0x17c <\_Z25CheckUserPermissionAccessv+92>   1. 0x000000000000017c <+92>:   callq 0x181 <\_Z25CheckUserPermissionAccessv+97> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register to prepare it for use in the following function call. 2. This instruction loads the address 0x0(%rip), the next instruction, into the %rdi register, the first argument register, for the following function call. 3. Calls the output function to display the string “Enter your password: “. |
| 1. 0x0000000000000181 <+97>:   lea 0x0(%rip),%rsi # 0x188 <\_Z25CheckUserPermissionAccessv+104>   1. 0x0000000000000188 <+104>:   lea 0x0(%rip),%rdi # 0x18f <\_Z25CheckUserPermissionAccessv+111>   1. 0x000000000000018f <+111>:   callq 0x194 <\_Z25CheckUserPermissionAccessv+116> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register for the following function call. 2. This loads the effective address 0x0(%rip) into the %rdi register, the first argument register for use with the next function call. 3. Calls the input function to receive input from the user and stores it in the password variable. |
| 1. 0x00000000000001a7 <+135>:   lea -0x40(%rbp),%rax   1. 0x00000000000001ab <+139>:   mov %rax,%rsi   1. 0x00000000000001ae <+142>:   lea 0x0(%rip),%rdi # 0x1b5 <\_Z25CheckUserPermissionAccessv+149>   1. 0x00000000000001b5 <+149>:   callq 0x1ba <\_Z25CheckUserPermissionAccessv+154>   1. 0x00000000000001ba <+154>:   lea -0x40(%rbp),%rax   1. 0x00000000000001be <+158>:   lea 0x0(%rip),%rsi # 0x1c5 <\_Z25CheckUserPermissionAccessv+165>   1. 0x00000000000001c5 <+165>:   mov %rax,%rdi   1. 0x00000000000001c8 <+168>:   callq 0x1cd <\_Z25CheckUserPermissionAccessv+173> | 1. This loads the effective address -0x40(%rbp) into the %rax register in preparation of the next function call. 2. This moves the value previously stored in the %rax register into the %rsi register, the second argument register for the next function call. 3. This instruction loads the effective address 0x0(%rip), the next instruction into the %rdi register, the first argument register for the following function call. 4. Calls the function at address 0x1ba. 5. Loads the effective address -0x40(%rbp) into the %rax register to prepare for use in the next function call. 6. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register in preparation for the next function call. 7. This moves the value previously stored in the %rax register into the %rdi register. 8. Calls the function at address 0x1cd. |
| 1. 0x00000000000001cd <+173>:   mov %eax,-0x44(%rbp)   1. 0x00000000000001d0 <+176>:   cmpl $0x0,-0x44(%rbp)   1. 0x00000000000001d4 <+180>:   jne 0x1dd <\_Z25CheckUserPermissionAccessv+189> | 1. This moves the value stored in the %eax register to the location -0x44(%rbp). 2. This instruction compares the value 0 with the value stored at location -0x44(%rbp). This is making the comparison to determine if the password entered by the user, matches the correctPassword variable. 3. Based on the previously made comparison, this instructs the program’s execution to jump to address 0x1dd, which is line 180 of the CheckUserPermissionAccess function, if the two values do not equal each other. |
| 1. 0x00000000000001d6 <+182>:   mov $0x1,%ebx   1. 0x00000000000001db <+187>:   jmp 0x1e2 <\_Z25CheckUserPermissionAccessv+194> | 1. If from the previously made comparison, the two values do equal each other, this line is reached and the value 1 is moved into the %ebx register to be used as the return value to indicate successful authentication. 2. This instruction jumps the program’s execution to address 0x1e2, which is line 194 of the CheckUserPermissionAccess function. |
| 1. 0x00000000000001dd <+189>:   mov $0x2,%ebx | 1. This instruction is reached from the previously made comparison when the two values compared do not equal each other. This instruction moves the value 2 into the %ebx register to be used as the return value to indicate successful authentication. |
| 1. 0x00000000000001e2 <+194>:   lea -0x40(%rbp),%rax   1. 0x00000000000001e6 <+198>:   mov %rax,%rdi   1. 0x00000000000001e9 <+201>:   callq 0x1ee <\_Z25CheckUserPermissionAccessv+206> | 1. This loads the effective address -0x40(%rbp) into the %rax register in preparation for the following function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register used in the following function call. 3. This calls the output function to display the strings: “Invalid Password. Please try again.” |
| 1. 0x00000000000001ee <+206>:   mov %ebx,%eax   1. 0x00000000000001f0 <+208>:   mov -0x18(%rbp),%rcx   1. 0x00000000000001f4 <+212>:   xor %fs:0x28,%rcx   1. 0x00000000000001fd <+221>:   je 0x23a <\_Z25CheckUserPermissionAccessv+282>   1. 0x00000000000001ff <+223>:   jmp 0x235 <\_Z25CheckUserPermissionAccessv+277> | Used for checking the stack for errors and cleanup   1. This moves the value stored in the %ebx register into the %eax register. 2. This moves the value stored at -0x18(%rbp) into the %rcx register to be used in the following XOR comparison. 3. This instruction makes an XOR comparison against the value stored %fs:0x28 segment register, with the value previously stored in the %rcx register, and stores this new value in the %rcx register. This is typically used for checking the stack for errors and cleanup. 4. Based on the previous comparison, if the two values equal each other, this instruction jumps the program’s execution to addredd 0x23a, which is line 282 of the CheckUserPermissionAccess function. 5. This line is reached if from the previous comparison the two values were not equal to each other. This instruction jumps the program’s eution to address 0x235, which is line 277 of the CheckUserPermissionAccess function. |
| 1. 0x0000000000000235 <+277>:   callq 0x23a <\_Z25CheckUserPermissionAccessv+282>   1. 0x000000000000023a <+282>:   add $0x48,%rsp   1. 0x000000000000023e <+286>:   pop %rbx | 1. This instruction calls the address 0x23a, which is the next instruction of the CheckUserPermissionAccess function. 2. This adds the value (0x48), which is 72 in decimal form, to the stack pointer, effectively deallocating 72 bytes of stack space. 3. This instruction pops the top value from the stack into the %rbx register, effectively restoring the value that was saved when this function was first called. |
| 1. 0x000000000000023f <+287>:   pop %rbp   1. 0x0000000000000240 <+288>:   retq | 1. This instruction restored the value of the base pointer register. 2. This instruction returns from the CheckUserPermissionAccess function. |

### DisplayInfo Function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| 1. 0x0000000000000241 <+0>:   push %rbp   1. 0x0000000000000242 <+1>:   mov %rsp,%rbp | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. |
| 1. 0x0000000000000245 <+4>:   lea 0x0(%rip),%rsi # 0x24c <\_Z11DisplayInfov+11>   1. 0x000000000000024c <+11>:   lea 0x0(%rip),%rdi # 0x253 <\_Z11DisplayInfov+18>   1. 0x0000000000000253 <+18>:   callq 0x258 <\_Z11DisplayInfov+23> | 1. This instruction loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register for the output function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register, for use with the following output function call. 3. This calls the output function to display the string “Clinet’s Name: “ |
| 1. 0x0000000000000258 <+23>:   mov %rax,%rdx   1. 0x000000000000025b <+26>:   mov 0x0(%rip),%rax # 0x262 <\_Z11DisplayInfov+33>   1. 0x0000000000000262 <+33>:   mov %rax,%rsi   1. 0x0000000000000265 <+36>:   mov %rdx,%rdi   1. 0x0000000000000268 <+39>:   callq 0x26d <\_Z11DisplayInfov+44> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function to output the string of the first client’s name |
| 1. 0x000000000000026d <+44>:   lea 0x0(%rip),%rsi # 0x274 <\_Z11DisplayInfov+51>   1. 0x0000000000000274 <+51>:   lea 0x0(%rip),%rdi # 0x27b <\_Z11DisplayInfov+58>   1. 0x000000000000027b <+58>:   callq 0x280 <\_Z11DisplayInfov+63> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used for the following function call. 3. This calls the function to output the string: “Service Selected (1 = Brokerage, 2 = Retirement)” |
| 1. 0x0000000000000280 <+63>:   lea 0x0(%rip),%rsi # 0x287 <\_Z11DisplayInfov+70>   1. 0x0000000000000287 <+70>:   mov %rax,%rdi   1. 0x000000000000028a <+73>:   callq 0x28f <\_Z11DisplayInfov+78> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string: “Selected Option “. |
| 1. 0x000000000000028f <+78>:   lea 0x0(%rip),%rsi # 0x296 <\_Z11DisplayInfov+85>   1. 0x0000000000000296 <+85>:   mov %rax,%rdi   1. 0x0000000000000299 <+88>:   callq 0x29e <\_Z11DisplayInfov+93> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string for the first client’s selected option. |
| 1. 0x000000000000029e <+93>:   mov %rax,%rdx   1. 0x00000000000002a1 <+96>:   mov 0x0(%rip),%eax # 0x2a7 <\_Z11DisplayInfov+102>   1. 0x00000000000002a7 <+102>:   mov %eax,%esi   1. 0x00000000000002a9 <+104>:   mov %rdx,%rdi   1. 0x00000000000002ac <+107>:   callq 0x2b1 <\_Z11DisplayInfov+112> | 1. This instruction moves the value stored in the %rax register into the %rdx register in preparation for the next function call. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %eax register. 3. This moves the value previously stored in the %eax register, into the %esi register, a first argument register for the next function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register, for use in the next function call. 5. This calls the output function to display the string: “Client’s Name : “ |
| 1. 0x00000000000002b1 <+112>:   mov %rax,%rdx   1. 0x00000000000002b4 <+115>:   mov 0x0(%rip),%rax # 0x2bb <\_Z11DisplayInfov+122>   1. 0x00000000000002bb <+122>:   mov %rax,%rsi   1. 0x00000000000002be <+125>:   mov %rdx,%rdi   1. 0x00000000000002c1 <+128>:   callq 0x2c6 <\_Z11DisplayInfov+133> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function to output the string of the second client’s name |
| 1. 0x00000000000002c6 <+133>:   lea 0x0(%rip),%rsi # 0x2cd <\_Z11DisplayInfov+140>   1. 0x00000000000002cd <+140>:   lea 0x0(%rip),%rdi # 0x2d4 <\_Z11DisplayInfov+147>   1. 0x00000000000002d4 <+147>:   callq 0x2d9 <\_Z11DisplayInfov+152> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used for the following function call. 3. This calls the function to output the string: “Service Selected (1 = Brokerage, 2 = Retirement)” |
| 1. 0x00000000000002d9 <+152>:   lea 0x0(%rip),%rsi # 0x2e0 <\_Z11DisplayInfov+159>   1. 0x00000000000002e0 <+159>:   mov %rax,%rdi   1. 0x00000000000002e3 <+162>:   callq 0x2e8 <\_Z11DisplayInfov+167> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string: “Selected Option “. |
| 1. 0x00000000000002e8 <+167>:   lea 0x0(%rip),%rsi # 0x2ef <\_Z11DisplayInfov+174>   1. 0x00000000000002ef <+174>:   mov %rax,%rdi   1. 0x00000000000002f2 <+177>:   callq 0x2f7 <\_Z11DisplayInfov+182> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string for the second client’s selected option. |
| 1. 0x00000000000002f7 <+182>:   mov %rax,%rdx   1. 0x00000000000002fa <+185>:   mov 0x0(%rip),%eax # 0x300 <\_Z11DisplayInfov+191>   1. 0x0000000000000300 <+191>:   mov %eax,%esi   1. 0x0000000000000302 <+193>:   mov %rdx,%rdi   1. 0x0000000000000305 <+196>:   callq 0x30a <\_Z11DisplayInfov+201> | 1. This instruction moves the value stored in the %rax register into the %rdx register in preparation for the next function call. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %eax register. 3. This moves the value previously stored in the %eax register, into the %esi register, a first argument register for the next function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register, for use in the next function call. 5. This calls the output function to display the string: “Client’s Name : “ |
| 1. 0x000000000000030a <+201>:   mov %rax,%rdx   1. 0x000000000000030d <+204>:   mov 0x0(%rip),%rax # 0x314 <\_Z11DisplayInfov+211>   1. 0x0000000000000314 <+211>:   mov %rax,%rsi   1. 0x0000000000000317 <+214>:   mov %rdx,%rdi   1. 0x000000000000031a <+217>:   callq 0x31f <\_Z11DisplayInfov+222> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function to output the string of the third client’s name. |
| 1. 0x000000000000031f <+222>:   lea 0x0(%rip),%rsi # 0x326 <\_Z11DisplayInfov+229>   1. 0x0000000000000326 <+229>:   lea 0x0(%rip),%rdi # 0x32d <\_Z11DisplayInfov+236>   1. 0x000000000000032d <+236>:   callq 0x332 <\_Z11DisplayInfov+241> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used for the following function call. 3. This calls the function to output the string: “Service Selected (1 = Brokerage, 2 = Retirement)” |
| 1. 0x0000000000000332 <+241>:   lea 0x0(%rip),%rsi # 0x339 <\_Z11DisplayInfov+248>   1. 0x0000000000000339 <+248>:   mov %rax,%rdi   1. 0x000000000000033c <+251>:   callq 0x341 <\_Z11DisplayInfov+256> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string: “Selected Option “. |
| 1. 0x0000000000000341 <+256>:   lea 0x0(%rip),%rsi # 0x348 <\_Z11DisplayInfov+263>   1. 0x0000000000000348 <+263>:   mov %rax,%rdi   1. 0x000000000000034b <+266>:   callq 0x350 <\_Z11DisplayInfov+271> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string for the third client’s selected option. |
| 1. 0x0000000000000350 <+271>:   mov %rax,%rdx   1. 0x0000000000000353 <+274>:   mov 0x0(%rip),%eax # 0x359 <\_Z11DisplayInfov+280>   1. 0x0000000000000359 <+280>:   mov %eax,%esi   1. 0x000000000000035b <+282>:   mov %rdx,%rdi   1. 0x000000000000035e <+285>:   callq 0x363 <\_Z11DisplayInfov+290> | 1. This instruction moves the value stored in the %rax register into the %rdx register in preparation for the next function call. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %eax register. 3. This moves the value previously stored in the %eax register, into the %esi register, a first argument register for the next function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register, for use in the next function call. 5. This calls the output function to display the string: “Client’s Name : “ |
| 1. 0x0000000000000363 <+290>:   mov %rax,%rdx   1. 0x0000000000000366 <+293>:   mov 0x0(%rip),%rax # 0x36d <\_Z11DisplayInfov+300>   1. 0x000000000000036d <+300>:   mov %rax,%rsi   1. 0x0000000000000370 <+303>:   mov %rdx,%rdi   1. 0x0000000000000373 <+306>:   callq 0x378 <\_Z11DisplayInfov+311> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function to output the string of the fourth client’s name. |
| 1. 0x0000000000000378 <+311>:   lea 0x0(%rip),%rsi # 0x37f <\_Z11DisplayInfov+318>   1. 0x000000000000037f <+318>:   lea 0x0(%rip),%rdi # 0x386 <\_Z11DisplayInfov+325>   1. 0x0000000000000386 <+325>:   callq 0x38b <\_Z11DisplayInfov+330> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used for the following function call. 3. This calls the function to output the string: “Service Selected (1 = Brokerage, 2 = Retirement)” |
| 1. 0x000000000000038b <+330>:   lea 0x0(%rip),%rsi # 0x392 <\_Z11DisplayInfov+337>   1. 0x0000000000000392 <+337>:   mov %rax,%rdi   1. 0x0000000000000395 <+340>:   callq 0x39a <\_Z11DisplayInfov+345> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string: “Selected Option “. |
| 1. 0x000000000000039a <+345>:   lea 0x0(%rip),%rsi # 0x3a1 <\_Z11DisplayInfov+352>   1. 0x00000000000003a1 <+352>:   mov %rax,%rdi   1. 0x00000000000003a4 <+355>:   callq 0x3a9 <\_Z11DisplayInfov+360> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string for the fourth client’s selected option. |
| 1. 0x00000000000003a9 <+360>:   mov %rax,%rdx   1. 0x00000000000003ac <+363>:   mov 0x0(%rip),%eax # 0x3b2 <\_Z11DisplayInfov+369>   1. 0x00000000000003b2 <+369>:   mov %eax,%esi   1. 0x00000000000003b4 <+371>:   mov %rdx,%rdi   1. 0x00000000000003b7 <+374>:   callq 0x3bc <\_Z11DisplayInfov+379> | 1. This instruction moves the value stored in the %rax register into the %rdx register in preparation for the next function call. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %eax register. 3. This moves the value previously stored in the %eax register, into the %esi register, a first argument register for the next function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register, for use in the next function call. 5. This calls the output function to display the string: “Client’s Name : “ |
| 1. 0x00000000000003bc <+379>:   mov %rax,%rdx   1. 0x00000000000003bf <+382>:   mov 0x0(%rip),%rax # 0x3c6 <\_Z11DisplayInfov+389>   1. 0x00000000000003c6 <+389>:   mov %rax,%rsi   1. 0x00000000000003c9 <+392>:   mov %rdx,%rdi   1. 0x00000000000003cc <+395>:   callq 0x3d1 <\_Z11DisplayInfov+400> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function to output the string of the fifth client’s name. |
| 1. 0x00000000000003d1 <+400>:   lea 0x0(%rip),%rsi # 0x3d8 <\_Z11DisplayInfov+407>   1. 0x00000000000003d8 <+407>:   lea 0x0(%rip),%rdi # 0x3df <\_Z11DisplayInfov+414>   1. 0x00000000000003df <+414>:   callq 0x3e4 <\_Z11DisplayInfov+419> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register used for the following function call. 3. This calls the function to output the string: “Service Selected (1 = Brokerage, 2 = Retirement)” |
| 1. 0x00000000000003e4 <+419>:   lea 0x0(%rip),%rsi # 0x3eb <\_Z11DisplayInfov+426>   1. 0x00000000000003eb <+426>:   mov %rax,%rdi   1. 0x00000000000003ee <+429>:   callq 0x3f3 <\_Z11DisplayInfov+434> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string: “Selected Option “. |
| 1. 0x00000000000003f3 <+434>:   lea 0x0(%rip),%rsi # 0x3fa <\_Z11DisplayInfov+441>   1. 0x00000000000003fa <+441>:   mov %rax,%rdi   1. 0x00000000000003fd <+444>:   callq 0x402 <\_Z11DisplayInfov+449> | 1. This loads the effective address 0x0(%rip), the next instruction, into the %rsi register, the second argument register used in the next function call. 2. This moves the value stored in the %rax register into the %rdi register, the first argument register, to be used in the following function call. 3. This calls the function to output the string for the fifth client’s selected option. |
| 1. 0x0000000000000402 <+449>:   mov %rax,%rdx   1. 0x0000000000000405 <+452>:   mov 0x0(%rip),%eax # 0x40b <\_Z11DisplayInfov+458>   1. 0x000000000000040b <+458>:   mov %eax,%esi   1. 0x000000000000040d <+460>:   mov %rdx,%rdi   1. 0x0000000000000410 <+463>:   callq 0x415 <\_Z11DisplayInfov+468> | 1. This instruction moves the value stored in the %rax register into the %rdx register in preparation for the next function call. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %eax register. 3. This moves the value previously stored in the %eax register, into the %esi register, a first argument register for the next function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register, for use in the next function call. 5. This calls the function at the address 0x415 |
| 1. 0x0000000000000415 <+468>:   mov %rax,%rdx   1. 0x0000000000000418 <+471>:   mov 0x0(%rip),%rax # 0x41f <\_Z11DisplayInfov+478>   1. 0x000000000000041f <+478>:   mov %rax,%rsi   1. 0x0000000000000422 <+481>:   mov %rdx,%rdi   1. 0x0000000000000425 <+484>:   callq 0x42a <\_Z11DisplayInfov+489> | 1. This moves the value stored in the %rax register into the %rdx register. 2. This instruction moves the value at the address 0x0%rip), the next instruction, into the %rax register in preparation for the next function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register for the use in the following output function call. 4. This moves the value previously stored in the %rdx register, into the %rdi register, the first argument register for the following output function call. 5. This calls the function located at the address 0x42a. |
| 1. 0x000000000000042a <+489>:   nop   1. 0x000000000000042b <+490>:   pop %rbp   1. 0x000000000000042c <+491>:   retq | 1. This instruction is a No Operation instruction, which eassentially does nothing, but instructs the execution to move to the next instruction without making any changes to register values, memory locations or flags. 2. This instruction restored the value of the base pointer register. 3. This instruction returns from the DisplayInfo function. |