CS-499 Computer Science Capstone
Milestone Three
5/28/2021
Zane Russell Brown

Assembly Code Analysis

Explain the functionality of the blocks of assembly code.

"main" function

Assembly Code Block	Explanation of Functionality
0x0000000000000000 <+0>: **push %rbp** 0x0000000000000001 <+1>: **mov %rsp,%rbp**	1.) %rbp is pushed into the stack. 2.) %rsp is moved into %rbp. This seems to be a normal set up for the main function. We can see the stack being set up.
0x0000000000000004 <+4>: **lea 0x0(%rip),%rsi # 0xb <main+11>** 0x000000000000000b <+11>: **lea 0x0(%rip),%rdi # 0x12 <main+18>** 0x0000000000000012 <+18>: **callq 0x17 <main+23>** 0x0000000000000017 <+23>: **callq 0x1c <main+28>** 0x000000000000001c <+28>: **mov %eax,0x0(%rip) # 0x22 <main+34>** 0x0000000000000022 <+34>: **mov 0x0(%rip),%eax # 0x28 <main+40>** 0x0000000000000028 <+40>: **cmp $0x1,%eax** 0x000000000000002b <+43>: **je 0x40 <main+64>**	1.) **Lea** is pointing 0(%rip) into %rsi and %rdi. Two **callq** commands are then used to execute lines <main+23> and <main+28>. 2.) %eax is then moved into 0(%rip). 3.) 0(%rip) is moved into %eax. 4.) The value 1 is then compared with %eax. The program will then jump to line <main+64> if equal to 1. Else, it will run the next line. The use of **lea** pointers and calls usually represents a string output. This seems to be true in this program as well. This is more than likely outputting a string for the user.

Assembly Code Block	Explanation of Functionality
0x000000000000002d <+45>: **lea 0x0(%rip),%rsi # 0x34 <main+52>** 0x0000000000000034 <+52>: **lea 0x0(%rip),%rdi # 0x3b <main+59>** 0x000000000000003b <+59>: **callq 0x40 <main+64>** 0x0000000000000040 <+64>: **mov 0x0(%rip),%eax # 0x46 <main+70>** 0x0000000000000046 <+70>: **cmp $0x1,%eax** 0x0000000000000049 <+73>: **je 0x4d <main+77>** 0x000000000000004b <+75>: **jmp 0x17 <main+23>**	1.) **Lea** is pointing 0(%rip) into %rsi and %rdi. A **callq** command is then used to execute line <main+64> which moves 0(%rip) into %eax. 2.) %eax is then compared against the value 1. The program will then jump to line <main+77> if equal. Else it will run the next line which jumps back to <main+23> The mixture of this block of assembly and the previous block of assembly with both their **lea** pointers, **call**s and **cmp** commands tells me that this could be the program outputting some strings for the user to see while also possibly calling in the password verification function before outputting any other strings.
0x000000000000004d <+77>: **lea 0x0(%rip),%rsi # 0x54 <main+84>** 0x0000000000000054 <+84>: **lea 0x0(%rip),%rdi # 0x5b <main+91>** 0x000000000000005b <+91>: **callq 0x60 <main+96>** 0x0000000000000060 <+96>: **lea 0x0(%rip),%rsi # 0x67 <main+103>** 0x0000000000000067 <+103>: **lea 0x0(%rip),%rdi # 0x6e <main+110>** 0x000000000000006e <+110>: **callq 0x73 <main+115>** 0x0000000000000073 <+115>: **lea 0x0(%rip),%rsi # 0x7a <main+122>** 0x000000000000007a <+122>: **lea 0x0(%rip),%rdi # 0x81 <main+129>** 0x0000000000000081 <+129>: **callq 0x86 <main+134>** 0x0000000000000086 <+134>: **lea 0x0(%rip),%rsi # 0x8d <main+141>** 0x000000000000008d <+141>: **lea 0x0(%rip),%rdi # 0x94 <main+148>** 0x0000000000000094 <+148>: **callq 0x99 <main+153>** 0x0000000000000099 <+153>: **lea 0x0(%rip),%rsi # 0xa0 <main+160>** 0x00000000000000a0 <+160>: **lea 0x0(%rip),%rdi # 0xa7 <main+167>** 0x00000000000000a7 <+167>: **callq 0xac <main+172>** 0x00000000000000ac <+172>: **lea 0x0(%rip),%rsi # 0xb3 <main+179>** 0x00000000000000b3 <+179>: **lea 0x0(%rip),%rdi # 0xba <main+186>** 0x00000000000000ba <+186>: **callq 0xbf <main+191>** 0x00000000000000bf <+191>: **mov %rax,%rdx**	1.) **Lea** points 0(%rip) into %rsi and %rdi. A **callq** is then made to run lines <main+96> which runs two other **lea** commands that point 0(%rip) into %rsi and %rdi. 2.) Step 1 continues to perform the same **lea** command and **callq** (while calling for their next respective <main+..> line) until the <main+186> line. 3.) After the repeated pointers and calls, this block of assembly finally uses a **callq** to call line <main+191> which moves %rax into %rdx. This large block of assembly code seems to be repeating itself until the last two lines. This could be used for constant output of strings that output when the while loop is running

Assembly Code Block	Explanation of Functionality
0x00000000000000c2 <+194>: **mov 0x0(%rip),%eax # 0xc8 <main+200>** 0x00000000000000c8 <+200>: **mov %eax,%esi** 0x00000000000000ca <+202>: **mov %rdx,%rdi** 0x00000000000000cd <+205>: **callq 0xd2 <main+210>** 0x00000000000000d2 <+210>: **mov %rax,%rdx** 0x00000000000000d5 <+213>: **mov 0x0(%rip),%rax # 0xdc <main+220>** 0x00000000000000dc <+220>: **mov %rax,%rsi** 0x00000000000000df <+223>: **mov %rdx,%rdi** 0x00000000000000e2 <+226>: **callq 0xe7 <main+231>** 0x00000000000000e7 <+231>: **mov 0x0(%rip),%eax # 0xed <main+237>** 0x00000000000000ed <+237>: **cmp $0x1,%eax** 0x00000000000000f0 <+240>: **jne 0xf9 <main+249>** 0x00000000000000f2 <+242>: **callq 0xf7 <main+247>** 0x00000000000000f7 <+247>: **jmp 0x109 <main+265>**	1.) 0(%rip) is moved into %eax. 2.) %eax is then moved into %esi. 3.) %rdx is moved into %rdi. 4.) A **callq** is moved here to run <main+210> which moves %rax into %rdx. 5.) 0(%rip) is then moved into %rax. 6.) %rax is moved into %rsi. 7.) %rdx is then moved into %rdi. 8.) Another **callq** is made here to run <main+231> which moves 0(%rip) into %eax. 9.) The program then compares 1 with %eax. The program will jump to <main+249> if not equal. Else, it will continue to the next line which calls for <main+247> to run which jumps to <main+265>. We see a couple of registers being moved around here. Notably, we see %esi and %rsi being used which could indicate both cout and cin statements being used. This along with the **cmp** commands being used could determine that the program is asking the users to input some sort of value to interact with the program.

Assembly Code Block	Explanation of Functionality
0x00000000000000f9 <+249>: **mov 0x0(%rip),%eax # 0xff <main+255>** 0x00000000000000ff <+255>: **cmp $0x2,%eax** 0x0000000000000102 <+258>: **jne 0x109 <main+265>** 0x0000000000000104 <+260>: **callq 0x109 <main+265>** 0x0000000000000109 <+265>: **mov 0x0(%rip),%eax # 0x10f <main+271>** 0x000000000000010f <+271>: **cmp $0x3,%eax** 0x0000000000000112 <+274>: **je 0x119 <main+281>** 0x0000000000000114 <+276>: **jmpq 0x4d <main+77>**	1.) 0(%rip) is moved into %eax. 2.) This %eax is then compared with the value 2. The program will then jump to line <main+265> if not equal. Else it will run the next line that is a **callq** that executes <main+265> which moves 0(%rip) into %eax. 3.) A **cmp** command is used here to compare the value 3 with %eax. The program will jump to <main+281> if equal. Else it will run to the next line of code. Which jumps the program back to <main+77>. From what I can see here, the previous block of assembly and this block are actually combined. The previous block compared %eax with the value 1 and this block of assembly is comparing %eax with 2 and 3. This looks like the program is requesting the user to enter 1, 2 or 3 to execute some sort of function. This evidence is backed even more due to the previous block using %esi and %rsi which are used for cout and cin statements.
0x0000000000000119 <+281>: **mov $0x0,%eax** 0x000000000000011e <+286>: **pop %rbp** 0x000000000000011f <+287>: **retq**	1.) This last block of assembly is used for zeroing out %eax, popping %rbp out of the stack and the return statement. Essentially the termination of the program.

Main() function

```
int main() {

	int userSelection;

	cout << "Hello! Welcome to our Investment Company" << endl;

	CheckUserPermissionAccess();

	while (userSelection != 3)
	{
		cout << "What would you like to do?" << endl;
		cout << "DISPLAY the client list (enter 1)" << endl;
		cout << "CHANGE a client's choice (enter 2)" << endl;
		cout << "Print Creators Name (enter 4)" << endl;
```

```
                cout << "Exit the program.. (enter 3)" << endl;
                cin >> userSelection;

                if (userSelection == 1)
                {
                        DisplayInfo();
                }
                else if (userSelection == 2)
                {
                        ChangeCustomerChoice();
                }
                else if (userSelection == 4)
                {
                        cout << "Zane Brown whom belongs to the Slytherin house made this." << endl;
                }
        }

        return 0;
}
```

Code and Assembly

All assembly functionality has been discussed above. The purpose of the below table is to group the c++ program and its assembly together.

Block of Assembly	C++ Code	Explanation of Functionality
0x0000000000000000 <+0>: **push %rbp** 0x0000000000000001 <+1>: **mov %rsp,%rbp**	**int main**() {	This is the set up that nearly all programs start with.
0x0000000000000004 <+4>: **lea 0x0(%rip),%rsi** 0x000000000000000b <+11>: **lea 0x0(%rip),%rdi** 0x0000000000000012 <+18>: **callq 0x17** 0x0000000000000017 <+23>: **callq 0x1c** 0x000000000000001c <+28>: **mov %eax,0x0(%rip)** 0x0000000000000022 <+34>: **mov 0x0(%rip),%eax** 0x0000000000000028 <+40>: **cmp $0x1,%eax** 0x000000000000002b <+43>: **je 0x40** 0x000000000000002d <+45>: **lea 0x0(%rip),%rsi** 0x0000000000000034 <+52>: **lea 0x0(%rip),%rdi** 0x000000000000003b <+59>: **callq 0x40**	**int** userSelection; cout << "Hello! Welcome to our Investment Company" << **endl**; CheckUserPermissionAccess(); **while** (userSelection != 3) {	The assembly here is showing that the program is outputting some strings which is more than likely the set up for our first string here. We then see the program running a **cmp** command that compares 1 to %eax. This is probably a boolean value and thus we can use a while loop to run the contents until true. Note: the username and password function are also here because that is run before we enter the program.

0x0000000000000040 <+64>: **mov 0x0(%rip),%eax** 0x0000000000000046 <+70>: **cmp $0x1,%eax** 0x0000000000000049 <+73>: **je 0x4d** 0x000000000000004b <+75>: **jmp 0x17**		
0x000000000000004d <+77>: **lea 0x0(%rip),%rsi** 0x0000000000000054 <+84>: **lea 0x0(%rip),%rdi** 0x000000000000005b <+91>: **callq 0x60** 0x0000000000000060 <+96>: **lea 0x0(%rip),%rsi** 0x0000000000000067 <+103>: **lea 0x0(%rip),%rdi** 0x000000000000006e <+110>: **callq 0x73** 0x0000000000000073 <+115>: **lea 0x0(%rip),%rsi** 0x000000000000007a <+122>: **lea 0x0(%rip),%rdi** 0x0000000000000081 <+129>: **callq 0x86** 0x0000000000000086 <+134>: **lea 0x0(%rip),%rsi** 0x000000000000008d <+141>: **lea 0x0(%rip),%rdi** 0x0000000000000094 <+148>: **callq 0x99** 0x0000000000000099 <+153>: **lea 0x0(%rip),%rsi** 0x00000000000000a0 <+160>: **lea 0x0(%rip),%rdi** 0x00000000000000a7 <+167>: **callq 0xac** 0x00000000000000ac <+172>: **lea 0x0(%rip),%rsi** 0x00000000000000b3 <+179>: **lea 0x0(%rip),%rdi** 0x00000000000000ba <+186>: **callq 0xbf** 0x00000000000000bf <+191>: **mov %rax,%rdx**	cout << "What would you like to do?" << **endl**; cout << "DISPLAY the client list (enter 1)" << **endl**; cout << "CHANGE a client's choice (enter 2)" << **endl**; cout << "Print Creators Name (enter 4)" << **endl**; cout << "Exit the program.. (enter 3)" << **endl**; cin >> userSelection;	This assembly looks like it is only calling for cout and cin statements. This would correspond with the strings and inputs that are called within the while loop.

0x00000000000000c2 <+194>: **mov 0x0(%rip),%eax**
0x00000000000000c8 <+200>: **mov %eax,%esi**
0x00000000000000ca <+202>: **mov %rdx,%rdi**
0x00000000000000cd <+205>: **callq 0xd2**
0x00000000000000d2 <+210>: **mov %rax,%rdx**
0x00000000000000d5 <+213>: **mov 0x0(%rip),%rax**
0x00000000000000dc <+220>: **mov %rax,%rsi**
	if (userSelection == 1)
	{
	DisplayInfo();
	}
	
	else if (userSelection == 2)
	{
	
	ChangeCustomerChoice();
		We can see several **cmp** commands being used here that are taking the values 1,2 and 3. This corresponds with the program requesting the user enter a value to run a specific function or exit.

0x00000000000000df <+223>: **mov %rdx,%rdi**
0x00000000000000e2 <+226>: **callq 0xe7**
0x00000000000000e7 <+231>: **mov 0x0(%rip),%eax**
0x00000000000000ed <+237>: **cmp $0x1,%eax**
0x00000000000000f0 <+240>: **jne 0xf9**
0x00000000000000f2 <+242>: **callq 0xf7**
0x00000000000000f7 <+247>: **jmp 0x109**
0x00000000000000f9 <+249>: **mov 0x0(%rip),%eax**
0x00000000000000ff <+255>: **cmp $0x2,%eax**
0x0000000000000102 <+258>: **jne 0x109**
0x0000000000000104 <+260>: **callq 0x109**
0x0000000000000109 <+265>: **mov 0x0(%rip),%eax**
0x000000000000010f <+271>: **cmp $0x3,%eax**
0x0000000000000112 <+274>: **je 0x119**
0x0000000000000114 <+276>: **jmpq 0x4d**
	}
	
	else if (userSelection == 4)
	{
	cout << "Zane Brown whom belongs to the Slytherin house made this." << **endl**;
	}
	}
		Note: I added a fourth option to output my name for the assignment requirement.

0x0000000000000119 <+281>: **mov $0x0,%eax** 0x000000000000011e <+286>: **pop %rbp** 0x000000000000011f <+287>: **retq**	**return** 0; }	This runs the return statement and terminates the program.

ChangeCustomerChoice function

Assembly Code Block	Explanation of Functionality
0x000000000000042d <+0>: **push %rbp** 0x000000000000042e <+1>: **mov %rsp,%rbp**	1.) %rbp is pushed into the stack 2.) %rsp is moved into %rbp Basic set up to the function and setting up the stack.

Assembly Code Block	Explanation of Functionality
0x0000000000000431 <+4>: **lea 0x0(%rip),%rsi # 0x438 <_Z20ChangeCustomerChoicev+11>** 0x0000000000000438 <+11>: **lea 0x0(%rip),%rdi # 0x43f <_Z20ChangeCustomerChoicev+18>** 0x000000000000043f <+18>: **callq 0x444 <_Z20ChangeCustomerChoicev+23>** 0x0000000000000444 <+23>: **lea 0x0(%rip),%rsi # 0x44b <_Z20ChangeCustomerChoicev+30>** 0x000000000000044b <+30>: **lea 0x0(%rip),%rdi # 0x452 <_Z20ChangeCustomerChoicev+37>** 0x0000000000000452 <+37>: **callq 0x457 <_Z20ChangeCustomerChoicev+42>** 0x0000000000000457 <+42>: **lea 0x0(%rip),%rsi # 0x45e <_Z20ChangeCustomerChoicev+49>** 0x000000000000045e <+49>: **lea 0x0(%rip),%rdi # 0x465 <_Z20ChangeCustomerChoicev+56>** 0x0000000000000465 <+56>: **callq 0x46a <_Z20ChangeCustomerChoicev+61>** 0x000000000000046a <+61>: **lea 0x0(%rip),%rsi # 0x471 <_Z20ChangeCustomerChoicev+68>** 0x0000000000000471 <+68>: **lea 0x0(%rip),%rdi # 0x478 <_Z20ChangeCustomerChoicev+75>** 0x0000000000000478 <+75>: **callq 0x47d <_Z20ChangeCustomerChoicev+80>** 0x000000000000047d <+80>: **mov 0x0(%rip),%eax # 0x483 <_Z20ChangeCustomerChoicev+86>** 0x0000000000000483 <+86>: **cmp $0x1,%eax** 0x0000000000000486 <+89>: **jne 0x496 <_Z20ChangeCustomerChoicev+105>** 0x0000000000000488 <+91>: **mov 0x0(%rip),%eax # 0x48e <_Z20ChangeCustomerChoicev+97>** 0x000000000000048e <+97>: **mov %eax,0x0(%rip) # 0x494 <_Z20ChangeCustomerChoicev+103>** 0x0000000000000494 <+103>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**	Note we will be referring to <_Z20ChangeCustomerChoicev> to <CCC> just to make things easier for the explanations. 1.) **Lea** is pointing 0(%rip) into %rsi and %rdi. A **callq** command is then used to run <CCC+23> which is another set of **lea** commands that point 0(%rip) into %rsi and %rdi. 2.) The steps above continue while **callq** commands execute their respective lines until we reach <CCC+61> 3.) At line <CCC+61> we have additional **lea** commands that point 0(%rip) into %rsi and %rdi. 4.) Then a **callq** command used to execute <CCC+80> which moves 0(%rip) into %eax. 5.) The value 1 is then compared with %eax. The program then jumps to <CCC+105> if not equal. Else, it will continue to the next line. 6.) 0(%rip) is then moved to %eax. 7.) %eax is moved into 0(%rip). 8.) The program will then jump to the end of the function. Like previous blocks of assembly code, these **lea** commands pointing to %rsi and %rdi seem to be outputting strings while also possibly setting up cin statements for user input. We also then see 0(%rip) moved into %eax and then compared with 1 and a jump if equal/not equal. This gives me the assumption that the program is asking for user input to interact with the system.

Assembly Code Block	Explanation of Functionality
0x0000000000000496 <+105>: **mov 0x0(%rip),%eax # 0x49c <_Z20ChangeCustomerChoicev+111>** 0x000000000000049c <+111>: **cmp $0x2,%eax** 0x000000000000049f <+114>: **jne 0x4af <_Z20ChangeCustomerChoicev+130>** 0x00000000000004a1 <+116>: **mov 0x0(%rip),%eax # 0x4a7 <_Z20ChangeCustomerChoicev+122>** 0x00000000000004a7 <+122>: **mov %eax,0x0(%rip) # 0x4ad <_Z20ChangeCustomerChoicev+128>** 0x00000000000004ad <+128>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**	1.) 0(%rip) is moved into %eax. 2.) **Cmp** command is used here to compare 2 with %eax. The program then jumps to <CCC+130> if not equal. Else, the following lines continue. 3.) 0(%rip) is then moved to %eax. 4.) %eax is moved to 0(%rip). 5.) The program jumps to the end of the function <CCC+203>. Again, we see the program taking the user's input. This time seeing if the user inputs the value 2. If equal, the program will perform some **mov**s and jump to line <+203>. Else it will jump to <+130> if not equal.
0x00000000000004af <+130>: **mov 0x0(%rip),%eax # 0x4b5 <_Z20ChangeCustomerChoicev+136>** 0x00000000000004b5 <+136>: **cmp $0x3,%eax** 0x00000000000004b8 <+139>: **jne 0x4c8 <_Z20ChangeCustomerChoicev+155>** 0x00000000000004ba <+141>: **mov 0x0(%rip),%eax # 0x4c0 <_Z20ChangeCustomerChoicev+147>** 0x00000000000004c0 <+147>: **mov %eax,0x0(%rip) # 0x4c6 <_Z20ChangeCustomerChoicev+153>** 0x00000000000004c6 <+153>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**	1.) 0(%rip) is moved into %eax 2.) **Cmp** command is used to compare 3 with %eax. The program will then jump to <CCC+155> if not equal. Else it will continue to the next line. 3.) 0(%rip) is moved into %eax. 4.) %eax is moved into 0(%rip). 5.) The program the jumps to the end of the function <CCC+203>. This is again taking in the user input. This time seeing if the user inputs the value 3. If equal, the program will perform the following lines under the **jne** command. Else, if not equal, the program will jump to <+155>.
0x00000000000004c8 <+155>: **mov 0x0(%rip),%eax # 0x4ce <_Z20ChangeCustomerChoicev+161>** 0x00000000000004ce <+161>: **cmp $0x4,%eax** 0x00000000000004d1 <+164>: **jne 0x4e1 <_Z20ChangeCustomerChoicev+180>** 0x00000000000004d3 <+166>: **mov 0x0(%rip),%eax # 0x4d9 <_Z20ChangeCustomerChoicev+172>** 0x00000000000004d9 <+172>: **mov %eax,0x0(%rip) # 0x4df <_Z20ChangeCustomerChoicev+178>** 0x00000000000004df <+178>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**	1.) 0(%rip) is moved to %eax. 2.) **Cmp** is used again to compare %eax with the value 4. The program will jump to <CCC+180> if not equal. Else it will continue to the next line if equal. 3.) 0(%rip) is moved into %eax. 4.) %eax is moved to 0(%rip). 5.) The program jumps to the end of the function <CCC+203>. Same as the above blocks of assembly. The program is seeing if the user inputs the value 4. If equal, the program will perform a couple **mov**s and then jump to <CCC+203>. Else, if not equal, the program will jump to <180>.

Assembly Code Block	Explanation of Functionality
0x00000000000004e1 <+180>: **mov 0x0(%rip),%eax # 0x4e7 <_Z20ChangeCustomerChoicev+186>** 0x00000000000004e7 <+186>: **cmp $0x5,%eax** 0x00000000000004ea <+189>: **jne 0x4f8 <_Z20ChangeCustomerChoicev+203>** 0x00000000000004ec <+191>: **mov 0x0(%rip),%eax # 0x4f2 <_Z20ChangeCustomerChoicev+197>** 0x00000000000004f2 <+197>: **mov %eax,0x0(%rip) # 0x4f8 <_Z20ChangeCustomerChoicev+203>**	1.) 0(%rip) is moved to %eax. 2.) **Cmp** is used to compare the value 5 with %eax. If not equal, the program will jump to <CCC+203>. Else, it will continue to the next line. 3.) 0(%rip) is moved to %eax. 4.) %eax is moved to 0(%rip). Finally, we see the program taking the users final input which is 5. If not equal, then the program will jump to <CCC+203>. Else, if equal the program will run the last two **mov** commands.
0x00000000000004f8 <+203>: **nop** 0x00000000000004f9 <+204>: **pop %rbp** 0x00000000000004fa <+205>: **retq**	1.) **Nop** command is used here as a null check. This is to make sure that the function is not empty. 2.) **%rbp** is popped from the stack. 3.) The return statement is called. The function terminates.

ChangeCustomerChoice () function

```
void ChangeCustomerChoice()
{
	int clientNumber;
	int clientNewService;

	cout << "You chose 2" << endl;
	cout << "Enter the number of the client that you wish to change" << endl;
	cin >> clientNumber;
	cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;
	cin >> clientNewService;

	if (clientNumber == 1)
	{
		bobOption = clientNewService;
	}
	else if (clientNumber == 2)
	{
		sarahOption = clientNewService;
	}
	else if (clientNumber == 3)
	{
		amyOption = clientNewService;
	}
	else if (clientNumber == 4)
```

```
        {
                johnnyOption = clientNewService;
        }
        else if (clientNumber == 5)
        {
                carolOption = clientNewService;
        }

        return;
}
```

Code and Assembly

All assembly functionality has been discussed above. The purpose of the below table is to group the c++ program and its assembly together.

Block of Assembly	C++ Code	Functionality
0x000000000000042d <+0>: **push %rbp** 0x000000000000042e <+1>: **mov %rsp,%rbp**	**void ChangeCustomerChoice**() {	This is the basic startup to all functions.
0x0000000000000431 <+4>: **lea 0x0(%rip),%rsi # 0x438 <_Z20ChangeCustomerChoicev+11>** 0x0000000000000438 <+11>: **lea 0x0(%rip),%rdi # 0x43f <_Z20ChangeCustomerChoicev+18>** 0x000000000000043f <+18>: **callq 0x444 <_Z20ChangeCustomerChoicev+23>** 0x0000000000000444 <+23>: **lea 0x0(%rip),%rsi # 0x44b <_Z20ChangeCustomerChoicev+30>** 0x000000000000044b <+30>: **lea 0x0(%rip),%rdi # 0x452 <_Z20ChangeCustomerChoicev+37>** 0x0000000000000452 <+37>: **callq 0x457 <_Z20ChangeCustomerChoicev+42>** 0x0000000000000457 <+42>: **lea 0x0(%rip),%rsi # 0x45e <_Z20ChangeCustomerChoicev+49>** 0x000000000000045e <+49>: **lea 0x0(%rip),%rdi # 0x465 <_Z20ChangeCustomerChoicev+56>** 0x0000000000000465 <+56>: **callq 0x46a <_Z20ChangeCustomerChoicev+61>** 0x000000000000046a <+61>: **lea 0x0(%rip),%rsi # 0x471 <_Z20ChangeCustomerChoicev+68>** 0x0000000000000471 <+68>: **lea 0x0(%rip),%rdi # 0x478 <_Z20ChangeCustomerChoicev+75>** 0x0000000000000478 <+75>: **callq 0x47d <_Z20ChangeCustomerChoicev+80>** 0x000000000000047d <+80>: **mov 0x0(%rip),%eax # 0x483 <_Z20ChangeCustomerChoicev+86>** 0x0000000000000483 <+86>: **cmp $0x1,%eax**	**int** clientNumber; **int** clientNewService; cout << "You chose 2" << **endl**; cout << "Enter the number of the client that you wish to change" << **endl**; cin >> clientNumber; cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << **endl**; cin >> clientNewService;	This block of assembly is setting up and running the necessary statements that output the strings and save the users input that will be used later to change the customer's choice. Also, at the bottom, we have a compare statement that is the first compare that we see for our if/else statements that determine what to do when a user enters a specific value.

0x0000000000000486 <+89>: **jne 0x496 <_Z20ChangeCustomerChoicev+105>** 0x0000000000000488 <+91>: **mov 0x0(%rip),%eax # 0x48e <_Z20ChangeCustomerChoicev+97>** 0x000000000000048e <+97>: **mov %eax,0x0(%rip) # 0x494 <_Z20ChangeCustomerChoicev+103>** 0x0000000000000494 <+103>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**		

0x0000000000000496 <+105>: **mov 0x0(%rip),%eax # 0x49c <_Z20ChangeCustomerChoicev+111>**
0x000000000000049c <+111>: **cmp $0x2,%eax**
0x000000000000049f <+114>: **jne 0x4af <_Z20ChangeCustomerChoicev+130>**
0x00000000000004a1 <+116>: **mov 0x0(%rip),%eax # 0x4a7 <_Z20ChangeCustomerChoicev+122>**
0x00000000000004a7 <+122>: **mov %eax,0x0(%rip) # 0x4ad <_Z20ChangeCustomerChoicev+128>**
0x00000000000004ad <+128>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**
0x00000000000004af <+130>: **mov 0x0(%rip),%eax # 0x4b5 <_Z20ChangeCustomerChoicev+136>**
0x00000000000004b5 <+136>: **cmp $0x3,%eax**
0x00000000000004b8 <+139>: **jne 0x4c8 <_Z20ChangeCustomerChoicev+155>**
0x00000000000004ba <+141>: **mov 0x0(%rip),%eax # 0x4c0 <_Z20ChangeCustomerChoicev+147>**
0x00000000000004c0 <+147>: **mov %eax,0x0(%rip) # 0x4c6 <_Z20ChangeCustomerChoicev+153>**
0x00000000000004c6 <+153>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**
0x00000000000004c8 <+155>: **mov 0x0(%rip),%eax # 0x4ce <_Z20ChangeCustomerChoicev+161>**
0x00000000000004ce <+161>: **cmp $0x4,%eax**
0x00000000000004d1 <+164>: **jne 0x4e1 <_Z20ChangeCustomerChoicev+180>**
0x00000000000004d3 <+166>: **mov 0x0(%rip),%eax # 0x4d9 <_Z20ChangeCustomerChoicev+172>**
0x00000000000004d9 <+172>: **mov %eax,0x0(%rip) # 0x4df <_Z20ChangeCustomerChoicev+178>**
0x00000000000004df <+178>: **jmp 0x4f8 <_Z20ChangeCustomerChoicev+203>**
0x00000000000004e1 <+180>: **mov 0x0(%rip),%eax # 0x4e7 <_Z20ChangeCustomerChoicev+186>**
0x00000000000004e7 <+186>: **cmp $0x5,%eax**
0x00000000000004ea <+189>: **jne 0x4f8 <_Z20ChangeCustomerChoicev+203>**
0x00000000000004ec <+191>: **mov 0x0(%rip),%eax # 0x4f2 <_Z20ChangeCustomerChoicev+197>**
	if (clientNumber == 1)
	{
	bobOption = clientNewService;
	}
	else if (clientNumber == 2)
	{
	sarahOption = clientNewService;
	}
	else if (clientNumber == 3)
	{
	amyOption = clientNewService;
	}
	else if (clientNumber == 4)
	{
	johnnyOption = clientNewService;
	}
	else if (clientNumber == 5)
	{
	carolOption = clientNewService;
	}
		This block of assembly is using a number of **cmp** commands that are used to run the if/else statements that help the program determine what to do when the user inputs 1~5. Global variables are used here to change the variables so they can be used in several functions.

0x00000000000004f2 <+197>: **mov %eax,0x0(%rip) # 0x4f8 <_Z20ChangeCustomerChoicev+203>**		
0x00000000000004f8 <+203>: **nop** 0x00000000000004f9 <+204>: **pop %rbp** 0x00000000000004fa <+205>: **retq**	**return**; }	This runs the return statement and terminates the function.

CheckUserPermissonAccess Function

Assembly Code Block	Explanation of Functionality
0x0000000000000120 <+0>: **push %rbp** 0x0000000000000121 <+1>: **mov %rsp,%rbp** 0x0000000000000124 <+4>: **push %rbx** 0x0000000000000125 <+5>: **sub $0x48,%rsp** 0x0000000000000129 <+9>: **mov %fs:0x28,%rax** 0x0000000000000132 <+18>: **mov %rax,-0x18(%rbp)** 0x0000000000000136 <+22>: **xor %eax,%eax**	1.) %rbp is pushed into the stack. 2.) %rsp is moved into %rbp. 3.) %rbx is then pushed into the stack. 4.) 48 is subtracted from %rsp for buffer. 5.) A stack-guard check is performed here. 6.) %rax is moved into -18(%rbp) 7.) %eax is zeroed out. This is again the setup we see in all our functions. This function does contain a couple more commands than usual. For example, we have another **push** command that is pushing %rbx into the stack. This might indicate the password that will be stored for the user to enter at a later time.

```
0x0000000000000138 <+24>:  lea    -0x45(%rbp),%rax
0x000000000000013c <+28>:  mov    %rax,%rdi
0x000000000000013f <+31>:  callq  0x144
<_Z25CheckUserPermissionAccessv+36>
0x0000000000000144 <+36>:  lea    -0x45(%rbp),%rdx
0x0000000000000148 <+40>:  lea    -0x40(%rbp),%rax
0x000000000000014c <+44>:  lea    0x0(%rip),%rsi        # 0x153
<_Z25CheckUserPermissionAccessv+51>
0x0000000000000153 <+51>:  mov    %rax,%rdi
0x0000000000000156 <+54>:  callq  0x15b
<_Z25CheckUserPermissionAccessv+59>
0x000000000000015b <+59>:  lea    -0x45(%rbp),%rax
0x000000000000015f <+63>:  mov    %rax,%rdi
0x0000000000000162 <+66>:  callq  0x167
<_Z25CheckUserPermissionAccessv+71>
0x0000000000000167 <+71>:  movl   $0x0,-0x44(%rbp)
0x000000000000016e <+78>:  lea    0x0(%rip),%rsi        # 0x175
<_Z25CheckUserPermissionAccessv+85>
0x0000000000000175 <+85>:  lea    0x0(%rip),%rdi        # 0x17c
<_Z25CheckUserPermissionAccessv+92>
0x000000000000017c <+92>:  callq  0x181
<_Z25CheckUserPermissionAccessv+97>
0x0000000000000181 <+97>:  lea    0x0(%rip),%rsi        # 0x188
<_Z25CheckUserPermissionAccessv+104>
0x0000000000000188 <+104>: lea    0x0(%rip),%rdi        # 0x18f
<_Z25CheckUserPermissionAccessv+111>
0x000000000000018f <+111>:  callq  0x194
<_Z25CheckUserPermissionAccessv+116>
0x0000000000000194 <+116>: lea    0x0(%rip),%rsi        # 0x19b
<_Z25CheckUserPermissionAccessv+123>
0x000000000000019b <+123>: lea    0x0(%rip),%rdi        # 0x1a2
<_Z25CheckUserPermissionAccessv+130>
0x00000000000001a2 <+130>: callq  0x1a7
<_Z25CheckUserPermissionAccessv+135>
0x00000000000001a7 <+135>: lea    -0x40(%rbp),%rax
0x00000000000001ab <+139>: mov    %rax,%rsi
0x00000000000001ae <+142>: lea    0x0(%rip),%rdi        # 0x1b5
<_Z25CheckUserPermissionAccessv+149>
0x00000000000001b5 <+149>: callq  0x1ba
<_Z25CheckUserPermissionAccessv+154>
0x00000000000001ba <+154>: lea    -0x40(%rbp),%rax
```

8.) **Lea** points -45(%rbp) into %rax.
9.) %rax is then moved into %rdi.
10.) A call is made to run line <+36> which is another **lea** command that points -45(%rbp) to %rdx.
11.) **Lea** command then points -40(%rbp) into %rax.
12.) Another **lea** command is used to point 0(%rip) into %rsi.
13.) %rax is moved to %rdi.
14.) A **callq** statement is used to run line <+59> which is another **lea** command that points -45(%rbp) to %rax.
15.)%rax is moved to %rdi.
16.) A call is used to execute line <+71> which moves 0 into -44(%rbp).
17.) **Lea** points 0(%rip) into %rsi and %rdi.
18.) A call is used to run line <+97> which is another **lea** command that points 0(%rip) into %rsi and then into %rdi.
19.)A **callq** is used to run <+116> which is yet again another **lea** command that points 0(%rip) into %rsi and then into %rdi.
20.) Another **callq** is used to run line <+135> which is a **lea** command that points -40(%rbp) into %rax.
21.)%rax is then moved into %rsi
22.)0(%rip) is then pointed into %rdi
23.) A call is made her to run <+154> which points -40(%rbp) into %rax.
24.) 0(%rip) is then pointed to %rsi.
25.) %rax is moved to %rdi.
26.) A call is used to execute line <+173> which moves %eax into -44(%rbp).
27.) A **cmpl** is used to compare 0 with -44(%rbp). If not equal, the program will jump to line <+189>. Else, if equal the program will continue to the next line.
28.) 1 is moved into %ebx.
29.)The program then jumps to line <+194>.

We see several **lea** pointer commands, **mov** commands and **callq** commands here in this block of assembly. This could let us know that the program is outputting a string for the user to see. We know from the .out file that the first text that we see when the program is executed is the “Enter your username” and “password” strings. This could point to that output. Additionally, we have a **cmpl** which could indicate that this is taking in the users input for the “username” or “password.

0x00000000000001be <+158>: **lea 0x0(%rip),%rsi # 0x1c5 <_Z25CheckUserPermissionAccessv+165>** 0x00000000000001c5 <+165>: **mov %rax,%rdi** 0x00000000000001c8 <+168>: **callq 0x1cd <_Z25CheckUserPermissionAccessv+173>** 0x00000000000001cd <+173>: **mov %eax,-0x44(%rbp)** 0x00000000000001d0 <+176>: **cmpl $0x0,-0x44(%rbp)** 0x00000000000001d4 <+180>: **jne 0x1dd <_Z25CheckUserPermissionAccessv+189>** 0x00000000000001d6 <+182>: **mov $0x1,%ebx** 0x00000000000001db <+187>: **jmp 0x1e2 <_Z25CheckUserPermissionAccessv+194>**	

0x00000000000001dd <+189>: **mov $0x2,%ebx** 0x00000000000001e2 <+194>: **lea -0x40(%rbp),%rax** 0x00000000000001e6 <+198>: **mov %rax,%rdi** 0x00000000000001e9 <+201>: **callq 0x1ee** **<_Z25CheckUserPermissionAccessv+206>** 0x00000000000001ee <+206>: **mov %ebx,%eax** 0x00000000000001f0 <+208>: **mov -0x18(%rbp),%rcx** 0x00000000000001f4 <+212>: **xor %fs:0x28,%rcx** 0x00000000000001fd <+221>: **je 0x23a** **<_Z25CheckUserPermissionAccessv+282>** 0x00000000000001ff <+223>: **jmp 0x235** **<_Z25CheckUserPermissionAccessv+277>** 0x0000000000000201 <+225>: **mov %rax,%rbx** 0x0000000000000204 <+228>: **lea -0x45(%rbp),%rax** 0x0000000000000208 <+232>: **mov %rax,%rdi** 0x000000000000020b <+235>: **callq 0x210** **<_Z25CheckUserPermissionAccessv+240>** 0x0000000000000210 <+240>: **mov %rbx,%rax** 0x0000000000000213 <+243>: **mov %rax,%rdi** 0x0000000000000216 <+246>: **callq 0x21b** **<_Z25CheckUserPermissionAccessv+251>** 0x000000000000021b <+251>: **mov %rax,%rbx** 0x000000000000021e <+254>: **lea -0x40(%rbp),%rax** 0x0000000000000222 <+258>: **mov %rax,%rdi** 0x0000000000000225 <+261>: **callq 0x22a** **<_Z25CheckUserPermissionAccessv+266>** 0x000000000000022a <+266>: **mov %rbx,%rax** 0x000000000000022d <+269>: **mov %rax,%rdi** 0x0000000000000230 <+272>: **callq 0x235** **<_Z25CheckUserPermissionAccessv+277>** 0x0000000000000235 <+277>: **callq 0x23a** **<_Z25CheckUserPermissionAccessv+282>** 0x000000000000023a <+282>: **add $0x48,%rsp**	1.) The value 2 is moved into %ebx. 2.) -40(%rbp) is moved to %rax. 3.) %rax is then moved into %rdi. 4.) A **callq** command is then used to run line <+206> which moves %ebx into %eax. 5.) -18(%rbp) moves into %rcx. 6.) %rax is xored by 8 bytes and then checks if 0 for a zero flag. 7.) The program jumps to <+282> if equal. 8.) The program jumps to <+277>. 9.) %rax is moved into %rbx. 10.)-45(%rbp) is pointed into %rax. 11.)%rax is then moved to %rdi. 12.)A call is made to run <+240> which moves %rbx into rax. And then %rax into %rdi. 13.)Another call is made to run <+251> which moves %rax into %rdi. 14.) Another call is made to run <+266> which moves %rbx into %rax and then %rax into %rdi. 15.)Finally another call is made to run <+277> which calls to run <+282> which adds the hex number 48 (72) into %rsp. This seems to be memory being moved as well as outputs and additional cout and cin statements being used after the compare statement in the previous block jumps when equal to a certain value.
0x000000000000023e <+286>: **pop %rbx** 0x000000000000023f <+287>: **pop %rbp** 0x0000000000000240 <+288>: **retq**	1.) %rbx and %rbp are popped from the stack. 2.) The return statement is called. 3.) The program terminates.

CheckUserPermissionAccess () function

```
void CheckUserPermissionAccess()
{

	int userPassInput;
	string usernameInput;
	int Pass = 123;

	cout << "Enter your username: " << endl;
	cin >> usernameInput;
	cout << "Enter your password: " << endl;
	cin >> userPassInput;

	if (userPassInput != Pass)
	{
		while (userPassInput != Pass) {
			cout << "Invalid Password. Please try again" << endl;
			cout << "Enter your username: " << endl;
			cin >> usernameInput;
			cout << "Enter your password: " << endl;
			cin >> userPassInput;
		}

	}
	else
	{
		cout << "password accepted" << endl;
	}

	return;

}
```

Code and Assembly

All assembly functionality has been discussed above. The purpose of the below table is to group the c++ program and its assembly together.

Block of Assembly	C++ Code	Functionality
0x0000000000000120 <+0>: **push %rbp** 0x0000000000000121 <+1>: **mov %rsp,%rbp** 0x0000000000000124 <+4>: **push %rbx** 0x0000000000000125 <+5>: **sub $0x48,%rsp** 0x0000000000000129 <+9>: **mov %fs:0x28,%rax** 0x0000000000000132 <+18>: **mov %rax,-0x18(%rbp)** 0x0000000000000136 <+22>: **xor %eax,%eax**	**void CheckUserPermissionAccess**() {	This is the basic setup we see when we start our functions. A couple more lines of assembly are used here however.
0x0000000000000138 <+24>: **lea -0x45(%rbp),%rax** 0x000000000000013c <+28>: **mov %rax,%rdi** 0x000000000000013f <+31>: **callq 0x144 <_Z25CheckUserPermissionAccessv+36>** 0x0000000000000144 <+36>: **lea -0x45(%rbp),%rdx** 0x0000000000000148 <+40>: **lea -0x40(%rbp),%rax** 0x000000000000014c <+44>: **lea 0x0(%rip),%rsi # 0x153 <_Z25CheckUserPermissionAccessv+51>** 0x0000000000000153 <+51>: **mov %rax,%rdi** 0x0000000000000156 <+54>: **callq 0x15b <_Z25CheckUserPermissionAccessv+59>** 0x000000000000015b <+59>: **lea -0x45(%rbp),%rax** 0x000000000000015f <+63>: **mov %rax,%rdi** 0x0000000000000162 <+66>: **callq 0x167 <_Z25CheckUserPermissionAccessv+71>** 0x0000000000000167 <+71>: **movl $0x0,-0x44(%rbp)** 0x000000000000016e <+78>: **lea 0x0(%rip),%rsi # 0x175 <_Z25CheckUserPermissionAccessv+85>** 0x0000000000000175 <+85>: **lea 0x0(%rip),%rdi # 0x17c <_Z25CheckUserPermissionAccessv+92>** 0x000000000000017c <+92>: **callq 0x181 <_Z25CheckUserPermissionAccessv+97>** 0x0000000000000181 <+97>: **lea 0x0(%rip),%rsi # 0x188 <_Z25CheckUserPermissionAccessv+104>** 0x0000000000000188 <+104>: **lea 0x0(%rip),%rdi # 0x18f <_Z25CheckUserPermissionAccessv+111>** 0x000000000000018f <+111>: **callq 0x194 <_Z25CheckUserPermissionAccessv+116>** 0x0000000000000194 <+116>: **lea 0x0(%rip),%rsi # 0x19b <_Z25CheckUserPermissionAccessv+123>** 0x000000000000019b <+123>: **lea 0x0(%rip),%rdi # 0x1a2 <_Z25CheckUserPermissionAccessv+130>**	**int** userPassInput; string usernameInput; **int** Pass = 123; cout << "Enter your username: " << **endl**; cin >> usernameInput; cout << "Enter your password: " << **endl**; cin >> userPassInput; **if** (userPassInput != Pass) {	This large block of code makes up several different parts of our function. First, we can see in the assembly that some unique memory locations are being passed into registers (e.g. -45(%rbp), %rdx). This could indicate the variables like our hardcoded password are being created. We can also see the assembly code using **lea, mov** and **callq** commands being used to more than likely call the cout and cin statements for string output and input from the user. Finally, at the end of this assembly, we can see a **cmp** command being used. This is more than likely the set up of our if/else statement that is used to determine if the user entered the correct password.

0x00000000000001a2 <+130>: **callq 0x1a7 <_Z25CheckUserPermissionAccessv+135>** 0x00000000000001a7 <+135>: **lea -0x40(%rbp),%rax** 0x00000000000001ab <+139>: **mov %rax,%rsi** 0x00000000000001ae <+142>: **lea 0x0(%rip),%rdi # 0x1b5 <_Z25CheckUserPermissionAccessv+149>** 0x00000000000001b5 <+149>: **callq 0x1ba <_Z25CheckUserPermissionAccessv+154>** 0x00000000000001ba <+154>: **lea -0x40(%rbp),%rax** 0x00000000000001be <+158>: **lea 0x0(%rip),%rsi # 0x1c5 <_Z25CheckUserPermissionAccessv+165>** 0x00000000000001c5 <+165>: **mov %rax,%rdi** 0x00000000000001c8 <+168>: **callq 0x1cd <_Z25CheckUserPermissionAccessv+173>** 0x00000000000001cd <+173>: **mov %eax,-0x44(%rbp)** 0x00000000000001d0 <+176>: **cmpl $0x0,-0x44(%rbp)** 0x00000000000001d4 <+180>: **jne 0x1dd <_Z25CheckUserPermissionAccessv+189>** 0x00000000000001d6 <+182>: **mov $0x1,%ebx** 0x00000000000001db <+187>: **jmp 0x1e2 <_Z25CheckUserPermissionAccessv+194>**		

0x00000000000001dd <+189>: **mov $0x2,%ebx**
0x00000000000001e2 <+194>: **lea -0x40(%rbp),%rax**
0x00000000000001e6 <+198>: **mov %rax,%rdi**
0x00000000000001e9 <+201>: **callq 0x1ee <_Z25CheckUserPermissionAccessv+206>**
0x00000000000001ee <+206>: **mov %ebx,%eax**
0x00000000000001f0 <+208>: **mov -0x18(%rbp),%rcx**
0x00000000000001f4 <+212>: **xor %fs:0x28,%rcx**
0x00000000000001fd <+221>: **je 0x23a <_Z25CheckUserPermissionAccessv+282>**
0x00000000000001ff <+223>: **jmp 0x235 <_Z25CheckUserPermissionAccessv+277>**
0x0000000000000201 <+225>: **mov %rax,%rbx**
0x0000000000000204 <+228>: **lea -0x45(%rbp),%rax**
0x0000000000000208 <+232>: **mov %rax,%rdi**
0x000000000000020b <+235>: **callq 0x210 <_Z25CheckUserPermissionAccessv+240>**
0x0000000000000210 <+240>: **mov %rbx,%rax**
0x0000000000000213 <+243>: **mov %rax,%rdi**
0x0000000000000216 <+246>: **callq 0x21b <_Z25CheckUserPermissionAccessv+251>**
0x000000000000021b <+251>: **mov %rax,%rbx**
0x000000000000021e <+254>: **lea -0x40(%rbp),%rax**
0x0000000000000222 <+258>: **mov %rax,%rdi**
	while (userPassInput != Pass) {
	cout << "Invalid Password. Please try again" << **endl**;
	cout << "Enter your username: " << **endl**;
	cin >> usernameInput;
	cout << "Enter your password: " << **endl**;
	cin >> userPassInput;
	}
	
	}
	else
	{
	cout << "password accepted" << **endl**;
	}
		In this last block of assembly before the function terminates, we can see that more **lea, mov, xor, jmp, je, callq** and **cmp** commands are used. We can assume from the commands being used here that this is running the while loop for when the user enters the wrong password. The function then outputs strings in the while loop until the user enters the correct password.
		
		Note: The else statement with "password accepted" was used entirely for debugging purposes and to show that I worked on this program.

0x0000000000000225 <+261>: **callq 0x22a <_Z25CheckUserPermissionAccessv+266>** 0x000000000000022a <+266>: **mov %rbx,%rax** 0x000000000000022d <+269>: **mov %rax,%rdi** 0x0000000000000230 <+272>: **callq 0x235 <_Z25CheckUserPermissionAccessv+277>** 0x0000000000000235 <+277>: **callq 0x23a <_Z25CheckUserPermissionAccessv+282>** 0x000000000000023a <+282>: **add $0x48,%rsp**		
0x000000000000023e <+286>: **pop %rbx** 0x000000000000023f <+287>: **pop %rbp** 0x0000000000000240 <+288>: **retq**	**return**; }	This runs the return statement and then terminates the function.

DisplayInfo Function

Assembly Code Block	Explanation of Functionality
0x0000000000000241 <+0>: **push %rbp** 0x0000000000000242 <+1>: **mov %rsp,%rbp**	1.) %rbp is pushed to the stack. 2.) %rsp is moved into %rbp This is our usual start up to the function. We can see the stack being set up and a register moving into the register that is pushed into the stack.
0x0000000000000245 <+4>: **lea 0x0(%rip),%rsi # 0x24c <_Z11DisplayInfov+11>** 0x000000000000024c <+11>: **lea 0x0(%rip),%rdi # 0x253 <_Z11DisplayInfov+18>** 0x0000000000000253 <+18>: **callq 0x258 <_Z11DisplayInfov+23>** 0x0000000000000258 <+23>: **mov %rax,%rdx**	1.) The **lea** commands are pointing 0(%rip) into %rsi and %rdi. 2.) A call is then made to run <+23> which moves %rax into %rdx Not too many complicated things are happening here. In fact, these memory locations will continue to be pointed into %rsi %rdi as well as %esi later on in the assembly. Additionally, calls will continue to be used to run **mov** to move registers like %rax into %rdx as well as others. This is because this function's main purpose is to display strings for the user.
0x000000000000025b <+26>: **mov 0x0(%rip),%rax # 0x262 <_Z11DisplayInfov+33>** 0x0000000000000262 <+33>: **mov %rax,%rsi** 0x0000000000000265 <+36>: **mov %rdx,%rdi** 0x0000000000000268 <+39>: **callq 0x26d <_Z11DisplayInfov+44>** 0x000000000000026d <+44>: **lea 0x0(%rip),%rsi # 0x274 <_Z11DisplayInfov+51>** 0x0000000000000274 <+51>: **lea 0x0(%rip),%rdi # 0x27b <_Z11DisplayInfov+58>** 0x000000000000027b <+58>: **callq 0x280 <_Z11DisplayInfov+63>** 0x0000000000000280 <+63>: **lea 0x0(%rip),%rsi # 0x287 <_Z11DisplayInfov+70>** 0x0000000000000287 <+70>: **mov %rax,%rdi** 0x000000000000028a <+73>: **callq 0x28f <_Z11DisplayInfov+78>** 0x000000000000028f <+78>: **lea 0x0(%rip),%rsi # 0x296 <_Z11DisplayInfov+85>** 0x0000000000000296 <+85>: **mov %rax,%rdi** 0x0000000000000299 <+88>: **callq 0x29e <_Z11DisplayInfov+93>** 0x000000000000029e <+93>: **mov %rax,%rdx**	1.) 0(%rip) is moved into %rax. 2.) %rax is then moved into %rsi. 3.) %rdx is moved to %rdi. 4.) A **callq** is used here to run line <+44> which uses **lea** commands to point 0(%rip) into %rsi and %rdi. 5.) Another call is made to run line <_63> which uses **lea** to point 0(%rip) into %rsi. 6.) **Mov** is used to move %rax into %rdi. 7.) Another call is used to execute line <+78> which uses another **lea** to point 0(%rip) into %rsi again. 8.) %rax is moved into %rdi. 9.) Finally a **callq** is uses to run line <+93> which moves %rax into %rdx. Again, this block of code seems to be moving around memory registers and pointing memory locations into registers in order to set up the function to output strings for the user.

Assembly Code Block	Explanation of Functionality
0x00000000000002a1 <+96>: **mov 0x0(%rip),%eax # 0x2a7 <_Z11DisplayInfov+102>** 0x00000000000002a7 <+102>: **mov %eax,%esi** 0x00000000000002a9 <+104>: **mov %rdx,%rdi** 0x00000000000002ac <+107>: **callq 0x2b1 <_Z11DisplayInfov+112>** 0x00000000000002b1 <+112>: **mov %rax,%rdx** 0x00000000000002b4 <+115>: **mov 0x0(%rip),%rax # 0x2bb <_Z11DisplayInfov+122>** 0x00000000000002bb <+122>: **mov %rax,%rsi** 0x00000000000002be <+125>: **mov %rdx,%rdi** 0x00000000000002c1 <+128>: **callq 0x2c6 <_Z11DisplayInfov+133>** 0x00000000000002c6 <+133>: **lea 0x0(%rip),%rsi # 0x2cd <_Z11DisplayInfov+140>** 0x00000000000002cd <+140>: **lea 0x0(%rip),%rdi # 0x2d4 <_Z11DisplayInfov+147>** 0x00000000000002d4 <+147>: **callq 0x2d9 <_Z11DisplayInfov+152>** 0x00000000000002d9 <+152>: **lea 0x0(%rip),%rsi # 0x2e0 <_Z11DisplayInfov+159>** 0x00000000000002e0 <+159>: **mov %rax,%rdi** 0x00000000000002e3 <+162>: **callq 0x2e8 <_Z11DisplayInfov+167>** 0x00000000000002e8 <+167>: **lea 0x0(%rip),%rsi # 0x2ef <_Z11DisplayInfov+174>** 0x00000000000002ef <+174>: **mov %rax,%rdi** 0x00000000000002f2 <+177>: **callq 0x2f7 <_Z11DisplayInfov+182>** 0x00000000000002f7 <+182>: **mov %rax,%rdx**	1.) 0(%rip) is moved into %eax. 2.) %eax is then moved into %esi 3.) %rdx is moved to %rdi. 4.) A call is made to run line <+112> which moves %rax into %rsi. 5.) The next two lines use **mov** commands that move %rax into %rsi and then %rdx into %rdi. 6.) A call is made to run line <+133> which executes a **lea** command that points 0(%rip) into %rsi. Another **lea** command then points 0(%rip) into %rdi. 7.) A **callq** command is then called to run line <+152> which runs another **lea** which points 0(%rip) into %rsi. 8.) %rax is moved into %rdi. 9.) A call is made to run line <+167> which uses a **lea** command to move 0(%rip) into %rsi. 10.)%rax is moved into %rdi 11.)Finally another call is used to execute line <+182> which **mov**es %rax into %rdx. Like before, the use of the many **mov**e commands and **lea** pointers let us know that some sort of strings are being output. Here, we actually see **%eax** and **%esi** being used here which gives us better evidence that some sort of function is happening here. %eax is usually used before some kind of execution (though usually with operations) and %esi is usually used with **cout** statements.

Assembly Code Block	Explanation of Functionality
0x00000000000002fa <+185>: **mov 0x0(%rip),%eax # 0x300 <_Z11DisplayInfov+191>** 0x0000000000000300 <+191>: **mov %eax,%esi** 0x0000000000000302 <+193>: **mov %rdx,%rdi** 0x0000000000000305 <+196>: **callq 0x30a <_Z11DisplayInfov+201>** 0x000000000000030a <+201>: **mov %rax,%rdx** 0x000000000000030d <+204>: **mov 0x0(%rip),%rax # 0x314 <_Z11DisplayInfov+211>** 0x0000000000000314 <+211>: **mov %rax,%rsi** 0x0000000000000317 <+214>: **mov %rdx,%rdi** 0x000000000000031a <+217>: **callq 0x31f <_Z11DisplayInfov+222>** 0x000000000000031f <+222>: **lea 0x0(%rip),%rsi # 0x326 <_Z11DisplayInfov+229>** 0x0000000000000326 <+229>: **lea 0x0(%rip),%rdi # 0x32d <_Z11DisplayInfov+236>** 0x000000000000032d <+236>: **callq 0x332 <_Z11DisplayInfov+241>** 0x0000000000000332 <+241>: **lea 0x0(%rip),%rsi # 0x339 <_Z11DisplayInfov+248>** 0x0000000000000339 <+248>: **mov %rax,%rdi** 0x000000000000033c <+251>: **callq 0x341 <_Z11DisplayInfov+256>** 0x0000000000000341 <+256>: **lea 0x0(%rip),%rsi # 0x348 <_Z11DisplayInfov+263>** 0x0000000000000348 <+263>: **mov %rax,%rdi** 0x000000000000034b <+266>: **callq 0x350 <_Z11DisplayInfov+271>** 0x0000000000000350 <+271>: **mov %rax,%rdx**	1.) 0(%rip) is moved into %eax 2.) %eax is then moved into %esi 3.) %rdx is moved into %rdi. 4.) A call is made to run line <+201> which moves %rax into %rdx. 5.) The next line moves 0(%rip) into %rax. 6.) %rax is moved into %rsi. 7.) %rdx is moved into %rdi. 8.) A call is made to run line <+222> which executes a **lea** command that points 0(%rip) into %rsi. Another **lea** command then points 0(%rip) into %rdi. 9.) A **callq** command is then called to run line <+241> which runs another **lea** which points 0(%rip) into %rsi. 10.)%rax is moved into %rdi. 11.)A call is made to run line <+256> which uses a **lea** command to move 0(%rip) into %rsi. 12.)%rax is moved into %rdi 13.) Again, another call is used to execute line <+271> which **mov**es %rax into %rdx. Apologies for the redundancy, however, I think it is important to repeat this information for each block. Again, the use of the many **mov**e commands and **lea** pointers let us know that some sort of strings are being output. Here, we actually see **%eax** and **%esi** being used here which gives us better evidence that some sort output with **cout** is happening here.

Assembly Code Block	Explanation of Functionality
0x0000000000000353 <+274>: **mov 0x0(%rip),%eax # 0x359 <_Z11DisplayInfov+280>** 0x0000000000000359 <+280>: **mov %eax,%esi** 0x000000000000035b <+282>: **mov %rdx,%rdi** 0x000000000000035e <+285>: **callq 0x363 <_Z11DisplayInfov+290>** 0x0000000000000363 <+290>: **mov %rax,%rdx** 0x0000000000000366 <+293>: **mov 0x0(%rip),%rax # 0x36d <_Z11DisplayInfov+300>** 0x000000000000036d <+300>: **mov %rax,%rsi** 0x0000000000000370 <+303>: **mov %rdx,%rdi** 0x0000000000000373 <+306>: **callq 0x378 <_Z11DisplayInfov+311>** 0x0000000000000378 <+311>: **lea 0x0(%rip),%rsi # 0x37f <_Z11DisplayInfov+318>** 0x000000000000037f <+318>: **lea 0x0(%rip),%rdi # 0x386 <_Z11DisplayInfov+325>** 0x0000000000000386 <+325>: **callq 0x38b <_Z11DisplayInfov+330>** 0x000000000000038b <+330>: **lea 0x0(%rip),%rsi # 0x392 <_Z11DisplayInfov+337>** 0x0000000000000392 <+337>: **mov %rax,%rdi** 0x0000000000000395 <+340>: **callq 0x39a <_Z11DisplayInfov+345>** 0x000000000000039a <+345>: **lea 0x0(%rip),%rsi # 0x3a1 <_Z11DisplayInfov+352>** 0x00000000000003a1 <+352>: **mov %rax,%rdi** 0x00000000000003a4 <+355>: **callq 0x3a9 <_Z11DisplayInfov+360>** 0x00000000000003a9 <+360>: **mov %rax,%rdx**	1.) 0(%rip) is moved into %eax 2.) %eax is then moved into %esi 3.) %rdx is moved into %rdi. 4.) A call is made to run line <+290> which moves %rax into %rdx. 5.) Then **mov** moves 0(%rip) into %rax. 6.) %rax is moved into %rsi. 7.) %rdx is moved into %rdi. 8.) A call is made to run line <+311> which executes a **lea** command that points 0(%rip) into %rsi. Another **lea** command then points 0(%rip) into %rdi. 9.) A **callq** command is then called to run line <+330> which runs another **lea** which points 0(%rip) into %rsi. 10.)%rax is moved into %rdi. 11.)A call is made to run line <+345> which uses a **lea** command to move 0(%rip) into %rsi. 12.)%rax is moved into %rdi 13.)Finally, another call is used to execute line <+360> which uses **mov** to %rax into %rdx. Again, the use of **%esi** and **%rsi** with the various **mov** and **lea** commands let us know that some sort of print statement is being used to output strings.

Assembly Code Block	Explanation of Functionality
0x00000000000003ac <+363>: **mov 0x0(%rip),%eax # 0x3b2 <_Z11DisplayInfov+369>** 0x00000000000003b2 <+369>: **mov %eax,%esi** 0x00000000000003b4 <+371>: **mov %rdx,%rdi** 0x00000000000003b7 <+374>: **callq 0x3bc <_Z11DisplayInfov+379>** 0x00000000000003bc <+379>: **mov %rax,%rdx** 0x00000000000003bf <+382>: **mov 0x0(%rip),%rax # 0x3c6 <_Z11DisplayInfov+389>** 0x00000000000003c6 <+389>: **mov %rax,%rsi** 0x00000000000003c9 <+392>: **mov %rdx,%rdi** 0x00000000000003cc <+395>: **callq 0x3d1 <_Z11DisplayInfov+400>** 0x00000000000003d1 <+400>: **lea 0x0(%rip),%rsi # 0x3d8 <_Z11DisplayInfov+407>** 0x00000000000003d8 <+407>: **lea 0x0(%rip),%rdi # 0x3df <_Z11DisplayInfov+414>** 0x00000000000003df <+414>: **callq 0x3e4 <_Z11DisplayInfov+419>** 0x00000000000003e4 <+419>: **lea 0x0(%rip),%rsi # 0x3eb <_Z11DisplayInfov+426>** 0x00000000000003eb <+426>: **mov %rax,%rdi** 0x00000000000003ee <+429>: **callq 0x3f3 <_Z11DisplayInfov+434>** 0x00000000000003f3 <+434>: **lea 0x0(%rip),%rsi # 0x3fa <_Z11DisplayInfov+441>** 0x00000000000003fa <+441>: **mov %rax,%rdi** 0x00000000000003fd <+444>: **callq 0x402 <_Z11DisplayInfov+449>** 0x0000000000000402 <+449>: **mov %rax,%rdx**	1.) 0(%rip) is moved into %eax 2.) %eax is then moved into %esi 3.) %rdx is moved into %rdi. 4.) A call is made to run line <+379> which moves %rax into %rdx. 5.) next 0(%rip) moves into %rax. 6.) %rax is moved into %rsi. 7.) %rdx is moved into %rdi. 8.) A call is made to run line <+400> which executes a **lea** command that points 0(%rip) into %rsi. Another **lea** command then points 0(%rip) into %rdi. 9.) A **callq** command is then called to run line <+419> which runs another **lea** which points 0(%rip) into %rsi. 10.)%rax is moved into %rdi. 11.)A call is made to run line <+434> which uses a **lea** command to move 0(%rip) into %rsi. 12.)%rax is moved into %rdi 13.) Another call is used to execute line <+449> which uses a **mov** command to move %rax into %rdx. We again see that this assembly block is using the combination of **mov**, **lea** and **callq** commands with, specifically, the %rsi, %eax, and %esi registers that can point to the program outputting strings.
0x0000000000000405 <+452>: **mov 0x0(%rip),%eax # 0x40b <_Z11DisplayInfov+458>** 0x000000000000040b <+458>: **mov %eax,%esi** 0x000000000000040d <+460>: **mov %rdx,%rdi** 0x0000000000000410 <+463>: **callq 0x415 <_Z11DisplayInfov+468>** 0x0000000000000415 <+468>: **mov %rax,%rdx** 0x0000000000000418 <+471>: **mov 0x0(%rip),%rax # 0x41f <_Z11DisplayInfov+478>** 0x000000000000041f <+478>: **mov %rax,%rsi** 0x0000000000000422 <+481>: **mov %rdx,%rdi**	1.) 0(%rip) is moved into %eax. 2.) %eax is then moved into %esi. 3.) %rdx is moved to %rdi. 4.) A **callq** command is used here to run line <+468> which moves %rax to %rdx. 5.) 0(%rip) is moved to %rax. 6.) %rax is moved to %rsi 7.) Then %rdx is moved to %rdi.

Assembly Code Block	Explanation of Functionality
0x0000000000000425 <+484>: **callq 0x42a <_Z11DisplayInfov+489>** 0x000000000000042a <+489>: **nop** 0x000000000000042b <+490>: **pop %rbp** 0x000000000000042c <+491>: **retq**	1.) A **callq** is made here to run line <+489> which runs a **nop** command which is used here as a null check. This is to make sure that the function is not empty. 2.) %rbp is popped from the stack. 3.) The return statement is called and the function is terminated.

DisplayInfo () function

```
void DisplayInfo()
{
        cout << "You chose 1" << endl;

        cout << "   Client's Name   Service Selected (1 = Brokerage, 2 = Retirement)" << endl;
        cout << "1. Bob Jones selected option " << bobOption << endl;
        cout << "2. Sarah Davis selected option " << sarahOption << endl;
        cout << "3. Amy Friendly selected option " << amyOption << endl;
        cout << "4. Johnny Smith selected option " << johnnyOption << endl;
        cout << "5. Carol Spears selected option " << carolOption << endl;

        return;

}
```

Code and Assembly

All assembly functionality has been discussed above. The purpose of the below table is to group the c++ program and its assembly together.

Block of Assembly Code	C++ Code	Explanation
0x0000000000000241 <+0>: **push %rbp** 0x0000000000000242 <+1>: **mov %rsp,%rbp**	**void DisplayInfo**() {	This is the basic set up for all of our functions.
0x0000000000000245 <+4>: **lea 0x0(%rip),%rsi # 0x24c <_Z11DisplayInfov+11>** 0x000000000000024c <+11>: **lea 0x0(%rip),%rdi # 0x253 <_Z11DisplayInfov+18>** 0x0000000000000253 <+18>: **callq 0x258 <_Z11DisplayInfov+23>** 0x0000000000000258 <+23>: **mov %rax,%rdx** 0x000000000000025b <+26>: **mov 0x0(%rip),%rax # 0x262 <_Z11DisplayInfov+33>** 0x0000000000000262 <+33>: **mov %rax,%rsi**	cout << "You chose 1" << **endl**; cout << " Client's Name Service Selected (1 = Brokerage, 2 = Retirement)" << **endl**; cout << "1. Bob Jones selected option " << bobOption << **endl**; cout << "2. Sarah Davis selected option " << sarahOption << **endl**; cout << "3. Amy Friendly selected option " << amyOption << **endl**;	This huge block of assembly code uses a similar pattern for several lines. It mainly uses the **lea, mov** and **callq** commands because the entire purpose of this function is to display information to the user. We can also see the **mov** commands being used here to change the global variable values when called in the ChangeCustomerChoice() function.

0x0000000000000265 <+36>: **mov %rdx,%rdi** 0x0000000000000268 <+39>: **callq 0x26d <_Z11DisplayInfov+44>** 0x000000000000026d <+44>: **lea 0x0(%rip),%rsi # 0x274 <_Z11DisplayInfov+51>** 0x0000000000000274 <+51>: **lea 0x0(%rip),%rdi # 0x27b <_Z11DisplayInfov+58>** 0x000000000000027b <+58>: **callq 0x280 <_Z11DisplayInfov+63>** 0x0000000000000280 <+63>: **lea 0x0(%rip),%rsi # 0x287 <_Z11DisplayInfov+70>** 0x0000000000000287 <+70>: **mov %rax,%rdi** 0x000000000000028a <+73>: **callq 0x28f <_Z11DisplayInfov+78>** 0x000000000000028f <+78>: **lea 0x0(%rip),%rsi # 0x296 <_Z11DisplayInfov+85>** 0x0000000000000296 <+85>: **mov %rax,%rdi** 0x0000000000000299 <+88>: **callq 0x29e <_Z11DisplayInfov+93>** 0x000000000000029e <+93>: **mov %rax,%rdx** 0x00000000000002a1 <+96>: **mov 0x0(%rip),%eax # 0x2a7 <_Z11DisplayInfov+102>** 0x00000000000002a7 <+102>: **mov %eax,%esi** 0x00000000000002a9 <+104>: **mov %rdx,%rdi** 0x00000000000002ac <+107>: **callq 0x2b1 <_Z11DisplayInfov+112>** 0x00000000000002b1 <+112>: **mov %rax,%rdx** 0x00000000000002b4 <+115>: **mov 0x0(%rip),%rax # 0x2bb <_Z11DisplayInfov+122>** 0x00000000000002bb <+122>: **mov %rax,%rsi** 0x00000000000002be <+125>: **mov %rdx,%rdi** 0x00000000000002c1 <+128>: **callq 0x2c6 <_Z11DisplayInfov+133>** 0x00000000000002c6 <+133>: **lea 0x0(%rip),%rsi # 0x2cd <_Z11DisplayInfov+140>** 0x00000000000002cd <+140>: **lea 0x0(%rip),%rdi # 0x2d4 <_Z11DisplayInfov+147>** 0x00000000000002d4 <+147>: **callq 0x2d9 <_Z11DisplayInfov+152>** 0x00000000000002d9 <+152>: **lea 0x0(%rip),%rsi # 0x2e0 <_Z11DisplayInfov+159>** 0x00000000000002e0 <+159>: **mov %rax,%rdi** 0x00000000000002e3 <+162>: **callq 0x2e8 <_Z11DisplayInfov+167>** 0x00000000000002e8 <+167>: **lea 0x0(%rip),%rsi # 0x2ef <_Z11DisplayInfov+174>** 0x00000000000002ef <+174>: **mov %rax,%rdi**	cout << "4. Johnny Smith selected option " << johnnyOption << **endl**; cout << "5. Carol Spears selected option " << carolOption << **endl**;	

0x00000000000002f2 <+177>: **callq 0x2f7 <_Z11DisplayInfov+182>** 0x00000000000002f7 <+182>: **mov %rax,%rdx** 0x00000000000002fa <+185>: **mov 0x0(%rip),%eax # 0x300 <_Z11DisplayInfov+191>** 0x0000000000000300 <+191>: **mov %eax,%esi** 0x0000000000000302 <+193>: **mov %rdx,%rdi** 0x0000000000000305 <+196>: **callq 0x30a <_Z11DisplayInfov+201>** 0x000000000000030a <+201>: **mov %rax,%rdx** 0x000000000000030d <+204>: **mov 0x0(%rip),%rax # 0x314 <_Z11DisplayInfov+211>** 0x0000000000000314 <+211>: **mov %rax,%rsi** 0x0000000000000317 <+214>: **mov %rdx,%rdi** 0x000000000000031a <+217>: **callq 0x31f <_Z11DisplayInfov+222>** 0x000000000000031f <+222>: **lea 0x0(%rip),%rsi # 0x326 <_Z11DisplayInfov+229>** 0x0000000000000326 <+229>: **lea 0x0(%rip),%rdi # 0x32d <_Z11DisplayInfov+236>** 0x000000000000032d <+236>: **callq 0x332 <_Z11DisplayInfov+241>** 0x0000000000000332 <+241>: **lea 0x0(%rip),%rsi # 0x339 <_Z11DisplayInfov+248>** 0x0000000000000339 <+248>: **mov %rax,%rdi** 0x000000000000033c <+251>: **callq 0x341 <_Z11DisplayInfov+256>** 0x0000000000000341 <+256>: **lea 0x0(%rip),%rsi # 0x348 <_Z11DisplayInfov+263>** 0x0000000000000348 <+263>: **mov %rax,%rdi** 0x000000000000034b <+266>: **callq 0x350 <_Z11DisplayInfov+271>** 0x0000000000000350 <+271>: **mov %rax,%rdx** 0x0000000000000353 <+274>: **mov 0x0(%rip),%eax # 0x359 <_Z11DisplayInfov+280>** 0x0000000000000359 <+280>: **mov %eax,%esi** 0x000000000000035b <+282>: **mov %rdx,%rdi** 0x000000000000035e <+285>: **callq 0x363 <_Z11DisplayInfov+290>** 0x0000000000000363 <+290>: **mov %rax,%rdx** 0x0000000000000366 <+293>: **mov 0x0(%rip),%rax # 0x36d <_Z11DisplayInfov+300>** 0x000000000000036d <+300>: **mov %rax,%rsi** 0x0000000000000370 <+303>: **mov %rdx,%rdi** 0x0000000000000373 <+306>: **callq 0x378 <_Z11DisplayInfov+311>** 0x0000000000000378 <+311>: **lea 0x0(%rip),%rsi # 0x37f <_Z11DisplayInfov+318>**		

0x000000000000037f <+318>: **lea 0x0(%rip),%rdi # 0x386 <_Z11DisplayInfov+325>** 0x0000000000000386 <+325>: **callq 0x38b <_Z11DisplayInfov+330>** 0x000000000000038b <+330>: **lea 0x0(%rip),%rsi # 0x392 <_Z11DisplayInfov+337>** 0x0000000000000392 <+337>: **mov %rax,%rdi** 0x0000000000000395 <+340>: **callq 0x39a <_Z11DisplayInfov+345>** 0x000000000000039a <+345>: **lea 0x0(%rip),%rsi # 0x3a1 <_Z11DisplayInfov+352>** 0x00000000000003a1 <+352>: **mov %rax,%rdi** 0x00000000000003a4 <+355>: **callq 0x3a9 <_Z11DisplayInfov+360>** 0x00000000000003a9 <+360>: **mov %rax,%rdx** 0x00000000000003ac <+363>: **mov 0x0(%rip),%eax # 0x3b2 <_Z11DisplayInfov+369>** 0x00000000000003b2 <+369>: **mov %eax,%esi** 0x00000000000003b4 <+371>: **mov %rdx,%rdi** 0x00000000000003b7 <+374>: **callq 0x3bc <_Z11DisplayInfov+379>** 0x00000000000003bc <+379>: **mov %rax,%rdx** 0x00000000000003bf <+382>: **mov 0x0(%rip),%rax # 0x3c6 <_Z11DisplayInfov+389>** 0x00000000000003c6 <+389>: **mov %rax,%rsi** 0x00000000000003c9 <+392>: **mov %rdx,%rdi** 0x00000000000003cc <+395>: **callq 0x3d1 <_Z11DisplayInfov+400>** 0x00000000000003d1 <+400>: **lea 0x0(%rip),%rsi # 0x3d8 <_Z11DisplayInfov+407>** 0x00000000000003d8 <+407>: **lea 0x0(%rip),%rdi # 0x3df <_Z11DisplayInfov+414>** 0x00000000000003df <+414>: **callq 0x3e4 <_Z11DisplayInfov+419>** 0x00000000000003e4 <+419>: **lea 0x0(%rip),%rsi # 0x3eb <_Z11DisplayInfov+426>** 0x00000000000003eb <+426>: **mov %rax,%rdi** 0x00000000000003ee <+429>: **callq 0x3f3 <_Z11DisplayInfov+434>** 0x00000000000003f3 <+434>: **lea 0x0(%rip),%rsi # 0x3fa <_Z11DisplayInfov+441>** 0x00000000000003fa <+441>: **mov %rax,%rdi** 0x00000000000003fd <+444>: **callq 0x402 <_Z11DisplayInfov+449>** 0x0000000000000402 <+449>: **mov %rax,%rdx** 0x0000000000000405 <+452>: **mov 0x0(%rip),%eax # 0x40b <_Z11DisplayInfov+458>** 0x000000000000040b <+458>: **mov %eax,%esi**		

0x000000000000040d <+460>: **mov %rdx,%rdi** 0x0000000000000410 <+463>: **callq 0x415 <_Z11DisplayInfov+468>** 0x0000000000000415 <+468>: **mov %rax,%rdx** 0x0000000000000418 <+471>: **mov 0x0(%rip),%rax # 0x41f <_Z11DisplayInfov+478>** 0x000000000000041f <+478>: **mov %rax,%rsi** 0x0000000000000422 <+481>: **mov %rdx,%rdi**		
0x0000000000000425 <+484>: **callq 0x42a <_Z11DisplayInfov+489>** 0x000000000000042a <+489>: **nop** 0x000000000000042b <+490>: **pop %rbp** 0x000000000000042c <+491>: **retq**	**return**; }	This calls the return statement, does a null check and then terminates the function.

Project 1 Program (all functions)

```
//============================================================================
// Name        : Practice.cpp
// Author      : Zane Brown
// Version     :
// Description : Project 1
//============================================================================

#include <iostream>
#include <string>
using namespace std;

int bobOption = 1;
int sarahOption = 2;
int amyOption = 1;
int johnnyOption = 1;
int carolOption = 2;


void CheckUserPermissionAccess()
{

	int userPassInput;
	string usernameInput;
	int Pass = 123;

	cout << "Enter your username: " << endl;
	cin >> usernameInput;
	cout << "Enter your password: " << endl;
	cin >> userPassInput;

	if (userPassInput != Pass)
	{
		while (userPassInput != Pass) {
			cout << "Invalid Password. Please try again" << endl;
			cout << "Enter your username: " << endl;
			cin >> usernameInput;
			cout << "Enter your password: " << endl;
			cin >> userPassInput;
		}

	}
	else
	{
		cout << "password accepted" << endl;
	}

	return;

}
```

```
void ChangeCustomerChoice()
{
	int clientNumber;
	int clientNewService;

	cout << "You chose 2" << endl;
	cout << "Enter the number of the client that you wish to change" << endl;
	cin >> clientNumber;
	cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;
	cin >> clientNewService;

	if (clientNumber == 1)
	{
		bobOption = clientNewService;
	}
	else if (clientNumber == 2)
	{
		sarahOption = clientNewService;
	}
	else if (clientNumber == 3)
	{
		amyOption = clientNewService;
	}
	else if (clientNumber == 4)
	{
		johnnyOption = clientNewService;
	}
	else if (clientNumber == 5)
	{
		carolOption = clientNewService;
	}

	return;
}

void DisplayInfo()
{
	cout << "You chose 1" << endl;

	cout << "   Client's Name   Service Selected (1 = Brokerage, 2 = Retirement)" << endl;
	cout << "1. Bob Jones selected option " << bobOption << endl;
	cout << "2. Sarah Davis selected option " << sarahOption << endl;
	cout << "3. Amy Friendly selected option " << amyOption << endl;
	cout << "4. Johnny Smith selected option " << johnnyOption << endl;
	cout << "5. Carol Spears selected option " << carolOption << endl;

	return;

}


int main() {
```

```
	int userSelection;

	cout << "Hello! Welcome to our Investment Company" << endl;

	CheckUserPermissionAccess();

	while (userSelection != 3)
	{
		cout << "What would you like to do?" << endl;
		cout << "DISPLAY the client list (enter 1)" << endl;
		cout << "CHANGE a client's choice (enter 2)" << endl;
		cout << "Print Creators Name (enter 4)" << endl;
		cout << "Exit the program.. (enter 3)" << endl;
		cin >> userSelection;

		if (userSelection == 1)
		{
			DisplayInfo();
		}
		else if (userSelection == 2)
		{
			ChangeCustomerChoice();
		}
		else if (userSelection == 4)
		{
			cout << "Zane Brown whom belongs to the Slytherin house made this." << endl;
		}
	}

	return 0;
}
```