# CS 410 Project One Proficiency Test Template

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

### “main” function”

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| <main>:  0: 55 push rbp  1: 48 89 e5 mov rbp,rsp  4: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # b <main+0xb>  b: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 12 <main+0x12>  12: e8 00 00 00 00 call 17 <main+0x17>  17: e8 00 00 00 00 call 1c <main+0x1c>  1c: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 22 <main+0x22>  22: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 28 <main+0x28>  28: 83 f8 01 cmp eax,0x1  2b: 74 13 je 40 <main+0x40>  2d: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 34 <main+0x34>  34: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 3b <main+0x3b>  3b: e8 00 00 00 00 call 40 <main+0x40>  40: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 46 <main+0x46>  46: 83 f8 01 cmp eax,0x1  49: 74 02 je 4d <main+0x4d>  4b: eb ca jmp 17 <main+0x17>  4d: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 54 <main+0x54>  54: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 5b <main+0x5b>  5b: e8 00 00 00 00 call 60 <main+0x60>  60: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 67 <main+0x67>  67: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 6e <main+0x6e>  6e: e8 00 00 00 00 call 73 <main+0x73>  73: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 7a <main+0x7a>  7a: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 81 <main+0x81>  81: e8 00 00 00 00 call 86 <main+0x86>  86: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 8d <main+0x8d>  8d: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 94 <main+0x94>  94: e8 00 00 00 00 call 99 <main+0x99>  99: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # a0 <main+0xa0>  a0: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # a7 <main+0xa7>  a7: e8 00 00 00 00 call ac <main+0xac>  ac: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # b3 <main+0xb3>  b3: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # ba <main+0xba>  ba: e8 00 00 00 00 call bf <main+0xbf>  bf: 48 89 c2 mov rdx,rax  c2: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # c8 <main+0xc8>  c8: 89 c6 mov esi,eax  ca: 48 89 d7 mov rdi,rdx  cd: e8 00 00 00 00 call d2 <main+0xd2>  d2: 48 89 c2 mov rdx,rax  d5: 48 8b 05 00 00 00 00 mov rax,QWORD PTR [rip+0x0] # dc <main+0xdc>  dc: 48 89 c6 mov rsi,rax  df: 48 89 d7 mov rdi,rdx  e2: e8 00 00 00 00 call e7 <main+0xe7>  e7: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # ed <main+0xed>  ed: 83 f8 01 cmp eax,0x1  f0: 75 07 jne f9 <main+0xf9>  f2: e8 00 00 00 00 call f7 <main+0xf7>  f7: eb 10 jmp 109 <main+0x109>  f9: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # ff <main+0xff>  ff: 83 f8 02 cmp eax,0x2  102: 75 05 jne 109 <main+0x109>  104: e8 00 00 00 00 call 109 <main+0x109>  109: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 10f <main+0x10f>  10f: 83 f8 03 cmp eax,0x3  112: 74 05 je 119 <main+0x119>  114: e9 34 ff ff ff jmp 4d <main+0x4d>  119: b8 00 00 00 00 mov eax,0x0  11e: 5d pop rbp  11f: c3 ret | * **Function Prologue:**   + push rbp: Save the base pointer of the previous stack frame.   + mov rbp, rsp: Set up the new base pointer for the current stack frame. * **Load Effective Addresses and Function Calls:**   + lea rsi, [rip+0x0]: Load effective address relative to the instruction pointer into the rsi register.   + lea rdi, [rip+0x0]: Load effective address relative to the instruction pointer into the rdi register.   + call 17: Call the function located at address 17.   + call 1c: Call the function located at address 1c. * **Memory Operations and Comparison:**   + mov DWORD PTR [rip+0x0], eax: Move the value in the eax register to the memory location pointed to by rip+0x0.   + mov eax, DWORD PTR [rip+0x0]: Load the value from the memory location pointed to by rip+0x0 into the eax register.   + cmp eax, 0x1: Compare the value in the eax register with 1.   + je 40: Jump to address 40 if the comparison result is equal (eax == 1). * **Additional Function Calls and Jumps:**   + lea rsi, [rip+0x0]: Load effective address relative to the instruction pointer into the rsi register.   + lea rdi, [rip+0x0]: Load effective address relative to the instruction pointer into the rdi register.   + call 40: Call the function located at address 40.   + mov eax, DWORD PTR [rip+0x0]: Load the value from the memory location pointed to by rip+0x0 into the eax register.   + cmp eax, 0x1: Compare the value in the eax register with 1.   + je 4d: Jump to address 4d if the comparison result is equal (eax == 1).   + jmp 17: Unconditionally jump to address 17. * **More Function Calls:**   + lea rsi, [rip+0x0]: Load effective address relative to the instruction pointer into the rsi register.   + lea rdi, [rip+0x0]: Load effective address relative to the instruction pointer into the rdi register.   + call 60: Call the function located at address 60.   + lea rsi, [rip+0x0]: Load effective address relative to the instruction pointer into the rsi register.   + lea rdi, [rip+0x0]: Load effective address relative to the instruction pointer into the rdi register.   + call 73: Call the function located at address 73. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### ChangeCustomerChoice function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| <\_Z20ChangeCustomerChoicev>:  420: 55 push rbp  421: 48 89 e5 mov rbp,rsp  424: 48 83 ec 20 sub rsp,0x20  428: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 42f <\_Z20ChangeCustomerChoicev+0xf>  42f: e8 00 00 00 00 call 434 <\_Z20ChangeCustomerChoicev+0x14>  434: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 43b <\_Z20ChangeCustomerChoicev+0x1b>  43b: e8 00 00 00 00 call 440 <\_Z20ChangeCustomerChoicev+0x20>  440: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 447 <\_Z20ChangeCustomerChoicev+0x27>  447: e8 00 00 00 00 call 44c <\_Z20ChangeCustomerChoicev+0x2c>  44c: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 453 <\_Z20ChangeCustomerChoicev+0x33>  453: e8 00 00 00 00 call 458 <\_Z20ChangeCustomerChoicev+0x38>  458: 48 83 c4 20 add rsp,0x20  45c: 5d pop rbp  45d: c3 ret  45e: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 464 <\_Z20ChangeCustomerChoicev+0x44>  464: 83 f8 01 cmp eax,0x1  467: 75 0e jne 477 <\_Z20ChangeCustomerChoicev+0x57>  469: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 46f <\_Z20ChangeCustomerChoicev+0x4f>  46f: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 475 <\_Z20ChangeCustomerChoicev+0x55>  475: eb 17 jmp 48e <\_Z20ChangeCustomerChoicev+0x6e>  477: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 47d <\_Z20ChangeCustomerChoicev+0x5d>  47d: 83 f8 02 cmp eax,0x2  480: 75 0e jne 490 <\_Z20ChangeCustomerChoicev+0x70>  482: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 488 <\_Z20ChangeCustomerChoicev+0x68>  488: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 48e <\_Z20ChangeCustomerChoicev+0x6e>  48e: eb 17 jmp 4a7 <\_Z20ChangeCustomerChoicev+0x87>  490: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 496 <\_Z20ChangeCustomerChoicev+0x76>  496: 83 f8 03 cmp eax,0x3  499: 75 0e jne 4a9 <\_Z20ChangeCustomerChoicev+0x89>  49b: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4a1 <\_Z20ChangeCustomerChoicev+0x81>  4a1: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 4a7 <\_Z20ChangeCustomerChoicev+0x87>  4a7: eb 49 jmp 4f8 <\_Z20ChangeCustomerChoicev+0xcb>  4a9: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4af <\_Z20ChangeCustomerChoicev+0x88>  4af: 83 f8 04 cmp eax,0x4  4b2: 75 0e jne 4c2 <\_Z20ChangeCustomerChoicev+0x9b>  4b4: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4ba <\_Z20ChangeCustomerChoicev+0x93>  4ba: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 4c0 <\_Z20ChangeCustomerChoicev+0x99>  4c0: eb 30 jmp 4f8 <\_Z20ChangeCustomerChoicev+0xcb>  4c2: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4c8 <\_Z20ChangeCustomerChoicev+0xa1>  4c8: 83 f8 05 cmp eax,0x5  4cb: 75 0e jne 4db <\_Z20ChangeCustomerChoicev+0xb4>  4cd: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4d3 <\_Z20ChangeCustomerChoicev+0xac>  4d3: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 4d9 <\_Z20ChangeCustomerChoicev+0xb2>  4d9: eb 17 jmp 4f8 <\_Z20ChangeCustomerChoicev+0xcb>  4db: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4e1 <\_Z20ChangeCustomerChoicev+0xba>  4e1: 83 f8 05 cmp eax,0x5  4e4: 75 0c jne 4f8 <\_Z20ChangeCustomerChoicev+0xcb>  4e6: 8b 05 00 00 00 00 mov eax,DWORD PTR [rip+0x0] # 4ec <\_Z20ChangeCustomerChoicev+0xc5>  4ec: 89 05 00 00 00 00 mov DWORD PTR [rip+0x0],eax # 4f2 <\_Z20ChangeCustomerChoicev+0xcb>  4f8: 90 nop  4f9: 5d pop rbp  4fa: c3 ret | 1. **Prologue:**    * push rbp: Save the base pointer of the previous stack frame.    * mov rbp, rsp: Set up the new base pointer for the current stack frame.    * sub rsp,0x20: Allocate 0x20 bytes on the stack. 2. **Display and Change Choices:**    * lea rdi,[rip+0x0]: Load the effective address of the string to be displayed into rdi.    * call <address>: Call the function at the specified address to display or change choices.    * This process is repeated for each choice display or change operation. 3. **Decision Making:**    * cmp eax,0x1: Compare the value in eax with 1.    * jne <address>: Jump if not equal to address.    * mov eax,DWORD PTR [rip+0x0]: Load the value from the specified memory location into eax.    * mov DWORD PTR [rip+0x0],eax: Store the value of eax into the specified memory location.    * This process is repeated for multiple choice comparisons and jumps. 4. **Epilogue:**    * nop: No operation (placeholder).    * pop rbp: Restore the previous base pointer.    * ret: Return from the function. |
|  |  |
|  |  |
|  |  |
|  |  |

### CheckUserPermissonAccess Function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| <\_Z25CheckUserPermissionAccessv>:  120: 55 push rbp  121: 48 89 e5 mov rbp,rsp  124: 53 push rbx  125: 48 83 ec 48 sub rsp,0x48  129: 64 48 8b 04 25 28 00 mov rax,QWORD PTR fs:0x28  130: 00 00  132: 48 89 45 e8 mov QWORD PTR [rbp-0x18],rax  136: 31 c0 xor eax,eax  138: 48 8d 45 bb lea rax,[rbp-0x45]  13c: 48 89 c7 mov rdi,rax  13f: e8 00 00 00 00 call 144 <\_Z25CheckUserPermissionAccessv+0x24>  144: 48 8d 55 bb lea rdx,[rbp-0x45]  148: 48 8d 45 c0 lea rax,[rbp-0x40]  14c: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 153 <\_Z25CheckUserPermissionAccessv+0x33>  153: 48 89 c7 mov rdi,rax  156: e8 00 00 00 00 call 15b <\_Z25CheckUserPermissionAccessv+0x3b>  15b: 48 8d 45 bb lea rax,[rbp-0x45]  15f: 48 89 c7 mov rdi,rax  162: e8 00 00 00 00 call 167 <\_Z25CheckUserPermissionAccessv+0x47>  167: c7 45 bc 00 00 00 00 mov DWORD PTR [rbp-0x44],0x0  16e: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 175 <\_Z25CheckUserPermissionAccessv+0x55>  175: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 17c <\_Z25CheckUserPermissionAccessv+0x5c>  17c: e8 00 00 00 00 call 181 <\_Z25CheckUserPermissionAccessv+0x61>  181: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 188 <\_Z25CheckUserPermissionAccessv+0x68>  188: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 18f <\_Z25CheckUserPermissionAccessv+0x6f>  18f: e8 00 00 00 00 call 194 <\_Z25CheckUserPermissionAccessv+0x74>  194: 48 8d 35 00 00 00 00 lea rsi,[rip+0x0] # 19b <\_Z25CheckUserPermissionAccessv+0x7b>  19b: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 1a2 <\_Z25CheckUserPermissionAccessv+0x82>  1a2: e8 00 00 00 00 call 1a7 <\_Z25CheckUserPermissionAccessv+0x87> | 1. **Prologue:**    * push rbp: Save the base pointer of the previous stack frame.    * mov rbp, rsp: Set up the new base pointer for the current stack frame.    * push rbx: Save the rbx register.    * sub rsp,0x48: Allocate 0x48 bytes on the stack. 2. **Setup and Function Calls:**    * mov rax,QWORD PTR fs:0x28: Move the value at fs:0x28 into rax.    * mov QWORD PTR [rbp-0x18],rax: Store rax in the stack frame.    * xor eax,eax: Zero out eax.    * lea rax,[rbp-0x45]: Load effective address of [rbp-0x45] into rax.    * mov rdi,rax: Move rax into rdi (setting up the argument for the next function call).    * call <address>: Call the function at the specified address.    * The subsequent instructions perform similar operations, calling different functions with prepared arguments and storing results in stack locations or registers. |
|  |  |
|  |  |
|  |  |

### DisplayInfo Function

| **Assembly Code Block** | **Explanation of Functionality** |
| --- | --- |
| <\_Z11DisplayInfov>:  210: 55 push rbp  211: 48 89 e5 mov rbp,rsp  214: 48 83 ec 20 sub rsp,0x20  218: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 21f <\_Z11DisplayInfov+0xf>  21f: e8 00 00 00 00 call 224 <\_Z11DisplayInfov+0x14>  224: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 22b <\_Z11DisplayInfov+0x1b>  22b: e8 00 00 00 00 call 230 <\_Z11DisplayInfov+0x20>  230: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 237 <\_Z11DisplayInfov+0x27>  237: e8 00 00 00 00 call 23c <\_Z11DisplayInfov+0x2c>  23c: 48 8d 3d 00 00 00 00 lea rdi,[rip+0x0] # 243 <\_Z11DisplayInfov+0x33>  243: e8 00 00 00 00 call 248 <\_Z11DisplayInfov+0x38>  248: 48 83 c4 20 add rsp,0x20  24c: 5d pop rbp  24d: c3 ret | 1. **Prologue:**    * push rbp: Save the base pointer of the previous stack frame.    * mov rbp, rsp: Set up the new base pointer for the current stack frame.    * sub rsp,0x20: Allocate 0x20 bytes on the stack. 2. **Display Information:**    * lea rdi,[rip+0x0]: Load the effective address of the string to be displayed into rdi.    * call <address>: Call the function at the specified address to display information.    * The process is repeated for each information string to be displayed. 3. **Epilogue:**    * add rsp,0x20: Deallocate the stack space.    * pop rbp: Restore the previous base pointer.    * ret: Return from the function. |
|  |  |
|  |  |
|  |  |
|  |  |