OXFORD BROOKES UNIVERSITY

MSc in Computer Science for Cyber Security

NETW7006: Malware Analysis

Coursework 1 – Security Analysis

Anisa Elezi - 19177366

1. Requirement analysis & system design

The focus of this coursework is to design and implement a simple record system to manage all computers and users of a system by an admin.
Admin should be able to complete these tasks:

a. Add user/computer
b. Delete user/computer
c. Search for user/computer
d. List all user/computer

He should be also able to store the following information:

- Per User (max 100 users)
  1. ***Full name***, which should be not more than 64 characters
  2. ***Department*** name in the specific list (Development, IT Support, Finance, or HR dep)
  3. ***User ID***, which should be unique and, in this format, **pXXXXXXX** p letter accompanied with 7 digits
  4. ***Email address,*** which should be in this format: ***@helpdesk.co.uk,*** that can either be written by the admin when creating a user or attached to the given name or user id.
- Per Computer (max 500 computers)
  1. ***Computer name***, which should be unique and in **cXXXXXXX** format, where is a c letter accompanied with 7 digits
  2. ***IP address***,
  3. ***OS***, should be in a specific list (Windows, Linux or macOS)
  4. Main user ID,
  5. Date of purchase,

Based on the description of a coursework and its requirements, first let's organise a flow chart which will help us then in programming in SASM, an open-source IDE used for assembler language. We will use x86 architecture.

A big picture of this system function would look like below:

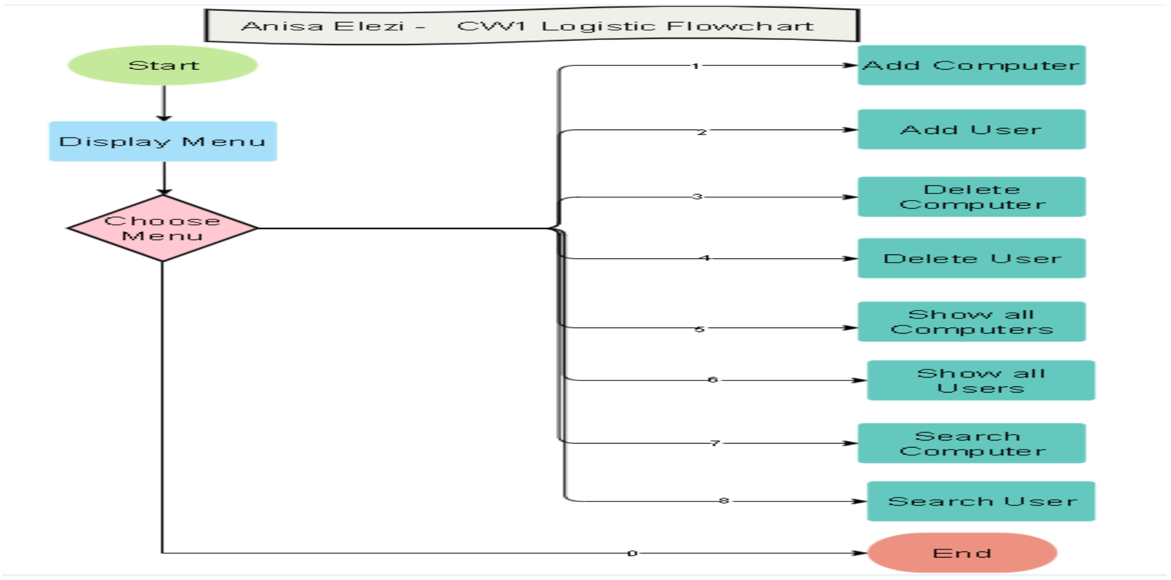
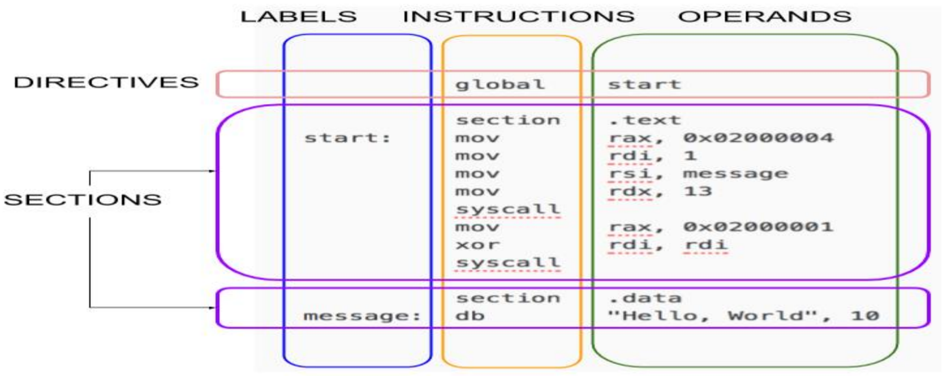


Figure 1. Logistic Flowchart

2. Coding and testing

The programming environment I have used SASM and 64-bit architecture. All is done using assembler language, which the main structure of syntax as follow:

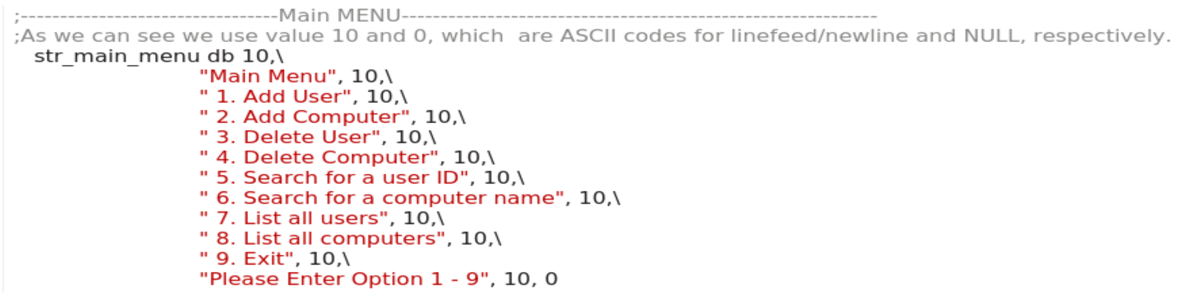


```
          global    start
start:    section   .text
          mov       rax, 0x02000004
          mov       rdi, 1
          mov       rsi, message
          mov       rdx, 13
          syscall
          mov       rax, 0x02000001
          xor       rdi, rdi
          syscall
          section   .data
message:  db        "Hello, World", 10
```

Figure 2. Assembler language syntax

The ***".data***" section is used to define data in memory, strings and other predefined data. This section we can read/write but not execute, because otherwise attacker could modify the data to be malicious executable code and then execute it.

Main menu will be coded like as follow:



```
;---------------------------------Main MENU---------------------------------------------------------------
;As we can see we use value 10 and 0, which  are ASCII codes for linefeed/newline and NULL, respectively.
  str_main_menu db 10,\
                        "Main Menu", 10,\
                        " 1. Add User", 10,\
                        " 2. Add Computer", 10,\
                        " 3. Delete User", 10,\
                        " 4. Delete Computer", 10,\
                        " 5. Search for a user ID", 10,\
                        " 6. Search for a computer name", 10,\
                        " 7. List all users", 10,\
                        " 8. List all computers", 10,\
                        " 9. Exit", 10,\
                        "Please Enter Option 1 - 9", 10, 0
```

Figure 3. Define Main Menu

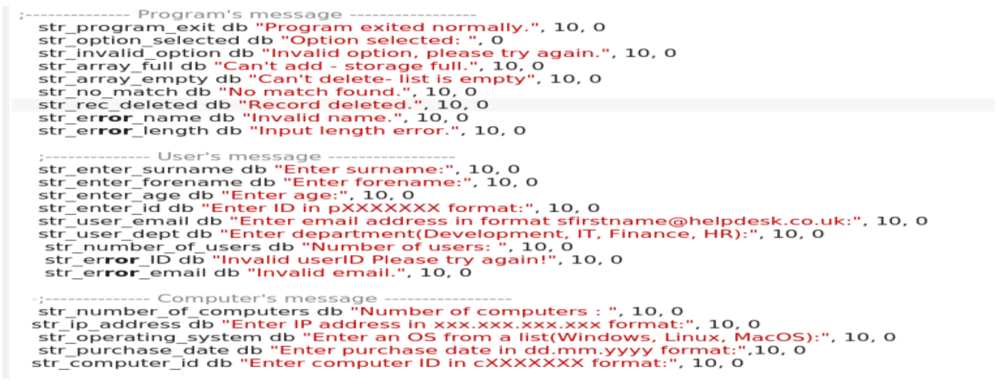
Some terminology we used in these section are:

1. "**db**" = "*Define Byte*", which allocates 1 byte.
2. "**dw**" = "*Define Word*", which allocates 2 bytes
3. "**dd**" = "*Define Doubleword*", which allocates 4 bytes
4. "**dq**" = "*Define Quadword*", which allocates 8 bytes

Above .data section is **global main**, which defines the point in the code from which the final executable starts execution.
As we can see after each string, we add bytes of value 0 and 10 (decimal). These are ASCII codes for NULL and linefeed, respectively.

- The *NULL* is required because we are using null-terminated strings.
- The *linefeed* makes the console drop down a line, which saves us having to call "*print_nl_new*" function separately.

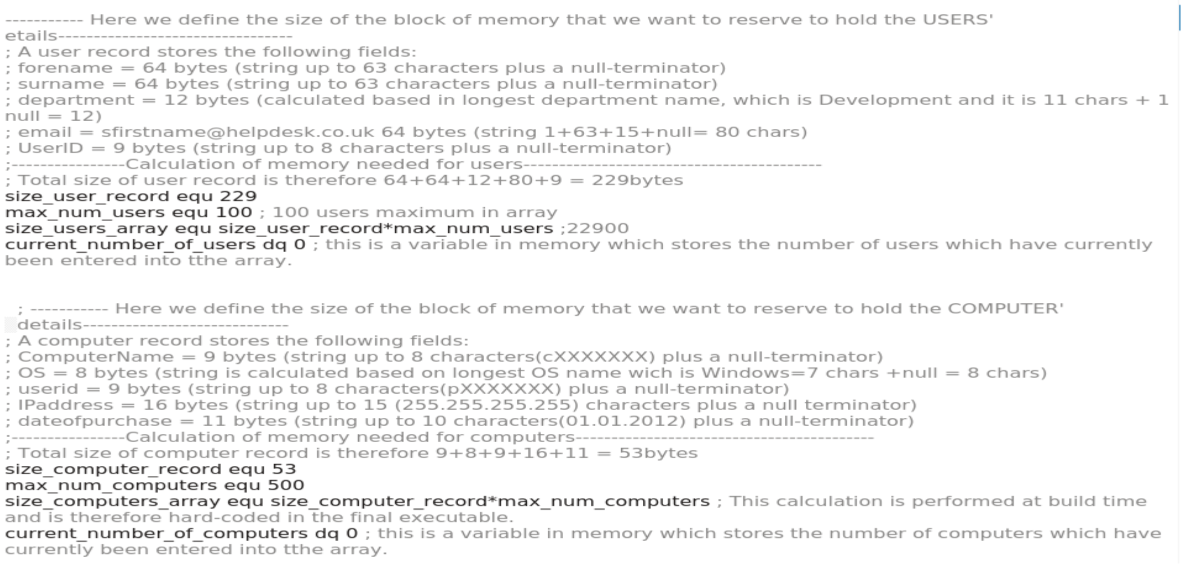
In fact, some strings defined here do not have a linefeed character. These are for occasions when we don't want the console to drop down a line when the program runs.
In this .data section we also add all of messages that we want to show during the execution of program. The complete list of messages would look like:



```
;-------------- Program's message ----------------
  str_program_exit db "Program exited normally.", 10, 0
  str_option_selected db "Option selected: ", 0
  str_invalid_option db "Invalid option, please try again.", 10, 0
  str_array_full db "Can't add - storage full.", 10, 0
  str_array_empty db "Can't delete- list is empty", 10, 0
  str_no_match db "No match found.", 10, 0
  str_rec_deleted db "Record deleted.", 10, 0
  str_error_name db "Invalid name.", 10, 0
  str_error_length db "Input length error.", 10, 0

  ;-------------- User's message ----------------
  str_enter_surname db "Enter surname:", 10, 0
  str_enter_forename db "Enter forename:", 10, 0
  str_enter_age db "Enter age:", 10, 0
  str_enter_id db "Enter ID in pXXXXXXX format:", 10, 0
  str_user_email db "Enter email address in format sfirstname@helpdesk.co.uk:", 10, 0
  str_user_dept db "Enter department(Development, IT, Finance, HR):", 10, 0
   str_number_of_users db "Number of users: ", 10, 0
   str_error_ID db "Invalid userID Please try again!", 10, 0
   str_error_email db "Invalid email.", 10, 0

 ;-------------- Computer's message ----------------
  str_number_of_computers db "Number of computers : ", 10, 0
 str_ip_address db "Enter IP address in xxx.xxx.xxx.xxx format:", 10, 0
  str_operating_system db "Enter an OS from a list(Windows, Linux, MacOS):", 10, 0
  str_purchase_date db "Enter purchase date in dd.mm.yyyy format:",10, 0
 str_computer_id db "Enter computer ID in cXXXXXXX format:", 10, 0
```

Figure 4. Messages used in program

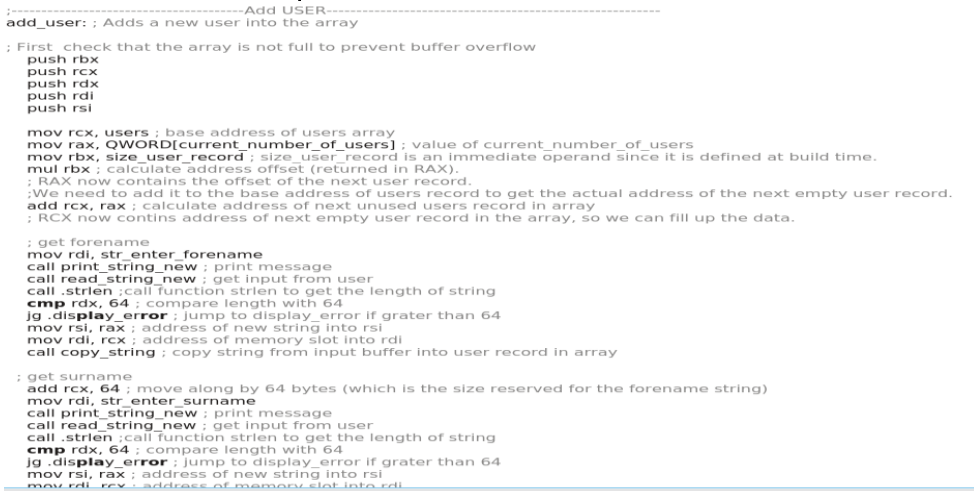
Before going to register, delete, search or list user, we have to define the memory needed for user's data.



```
----------- Here we define the size of the block of memory that we want to reserve to hold the USERS'
etails---------------------------------
; A user record stores the following fields:
; forename = 64 bytes (string up to 63 characters plus a null-terminator)
; surname = 64 bytes (string up to 63 characters plus a null-terminator)
; department = 12 bytes (calculated based in longest department name, which is Development and it is 11 chars + 1
null = 12)
; email = sfirstname@helpdesk.co.uk 64 bytes (string 1+63+15+null= 80 chars)
; UserID = 9 bytes (string up to 8 characters plus a null-terminator)
;----------------Calculation of memory needed for users-----------------------------------------
; Total size of user record is therefore 64+64+12+80+9 = 229bytes
size_user_record equ 229
max_num_users equ 100 ; 100 users maximum in array
size_users_array equ size_user_record*max_num_users ;22900
current_number_of_users dq 0 ; this is a variable in memory which stores the number of users which have currently
been entered into tthe array.


 ; ----------- Here we define the size of the block of memory that we want to reserve to hold the COMPUTER'
 details-----------------------------
; A computer record stores the following fields:
; ComputerName = 9 bytes (string up to 8 characters(cXXXXXXX) plus a null-terminator)
; OS = 8 bytes (string is calculated based on longest OS name wich is Windows=7 chars +null = 8 chars)
; userid = 9 bytes (string up to 8 characters(pXXXXXXX) plus a null-terminator)
; IPaddress = 16 bytes (string up to 15 (255.255.255.255) characters plus a null terminator)
; dateofpurchase = 11 bytes (string up to 10 characters(01.01.2012) plus a null-terminator)
;----------------Calculation of memory needed for computers-----------------------------------------
; Total size of computer record is therefore 9+8+9+16+11 = 53bytes
size_computer_record equ 53
max_num_computers equ 500
size_computers_array equ size_computer_record*max_num_computers ; This calculation is performed at build time
and is therefore hard-coded in the final executable.
current_number_of_computers dq 0 ; this is a variable in memory which stores the number of computers which have
currently been entered into tthe array.
```

Figure 5. User memory calculation

Let's take the user registration. In this section we have to look after input from user for userID, name and surname, Department and email.

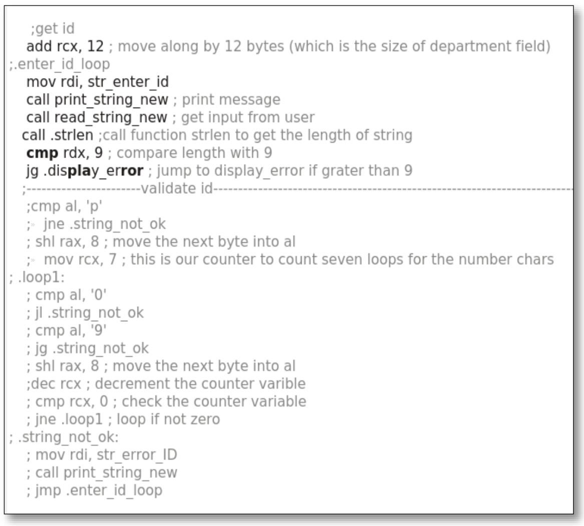


```
;--------------------------------------Add USER-------------------------------------------------------
add_user: ; Adds a new user into the array

; First  check that the array is not full to prevent buffer overflow
   push rbx
   push rcx
   push rdx
   push rdi
   push rsi

   mov rcx, users ; base address of users array
   mov rax, QWORD[current_number_of_users] ; value of current_number_of_users
   mov rbx, size_user_record ; size_user_record is an immediate operand since it is defined at build time.
   mul rbx ; calculate address offset (returned in RAX).
   ; RAX now contains the offset of the next user record.
   ;We need to add it to the base address of users record to get the actual address of the next empty user record.
   add rcx, rax ; calculate address of next unused users record in array
   ; RCX now contins address of next empty user record in the array, so we can fill up the data.

   ; get forename
   mov rdi, str_enter_forename
   call print_string_new ; print message
   call read_string_new ; get input from user
   call .strlen ;call function strlen to get the length of string
   cmp rdx, 64 ; compare length with 64
   jg .display_error ; jump to display_error if grater than 64
   mov rsi, rax ; address of new string into rsi
   mov rdi, rcx ; address of memory slot into rdi
   call copy_string ; copy string from input buffer into user record in array

  ; get surname
   add rcx, 64 ; move along by 64 bytes (which is the size reserved for the forename string)
   mov rdi, str_enter_surname
   call print_string_new ; print message
   call read_string_new ; get input from user
   call .strlen ;call function strlen to get the length of string
   cmp rdx, 64 ; compare length with 64
   jg .display_error ; jump to display_error if grater than 64
   mov rsi, rax ; address of new string into rsi
   mov rdi, rcx ; address of memory slot into rdi
```

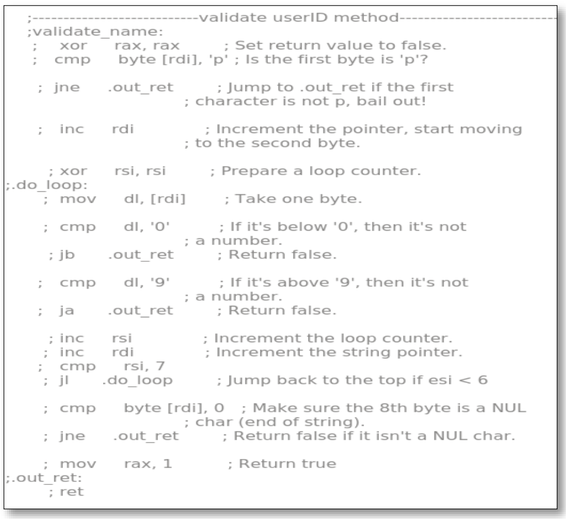
Figure 6. Enter forename and surname of user

As we can noticed here, there is an instruction ***add rcx,65***, which reserve memory for forename string.

The most difficult task was to validate inputs, for example UserID to be in a proper format pXXXXXXX. This is how I thought to solve the problem but it still needs to work on it.



```
   ;get id
  add rcx, 12 ; move along by 12 bytes (which is the size of department field)
;.enter_id_loop
  mov rdi, str_enter_id
  call print_string_new ; print message
  call read_string_new ; get input from user
 call .strlen ;call function strlen to get the length of string
  cmp rdx, 9 ; compare length with 9
  jg .display_error ; jump to display_error if grater than 9
 ;----------------------validate id------------------------------------------------------------------
  ;cmp al, 'p'
  ;   jne .string_not_ok
  ; shl rax, 8 ; move the next byte into al
  ;   mov rcx, 7 ; this is our counter to count seven loops for the number chars
; .loop1:
  ; cmp al, '0'
  ; jl .string_not_ok
  ; cmp al, '9'
  ; jg .string_not_ok
  ; shl rax, 8 ; move the next byte into al
  ;dec rcx ; decrement the counter varible
  ; cmp rcx, 0 ; check the counter variable
  ; jne .loop1 ; loop if not zero
; .string_not_ok:
  ; mov rdi, str_error_ID
  ; call print_string_new
  ; jmp .enter_id_loop
```

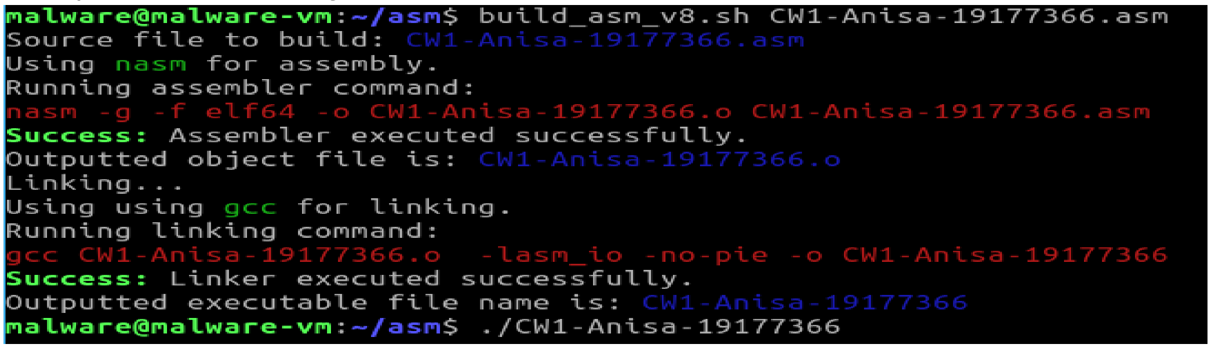


```
 ;--------------------------validate userID method--------------------------
 ;validate_name:
  ;    xor     rax, rax        ; Set return value to false.
  ;   cmp     byte [rdi], 'p' ; Is the first byte is 'p'?

   ;  jne     .out_ret        ; Jump to .out_ret if the first
                              ; character is not p, bail out!

   ;   inc     rdi              ; Increment the pointer, start moving
                              ; to the second byte.

     ; xor     rsi, rsi        ; Prepare a loop counter.
;.do_loop:
    ;  mov     dl, [rdi]       ; Take one byte.

    ;  cmp     dl, '0'         ; If it's below '0', then it's not
                              ; a number.
     ; jb      .out_ret        ; Return false.

   ;   cmp     dl, '9'         ; If it's above '9', then it's not
                              ; a number.
   ;   ja      .out_ret        ; Return false.

     ; inc     rsi             ; Increment the loop counter.
    ;  inc     rdi              ; Increment the string pointer.
   ;   cmp     rsi, 7
    ;  jl     .do_loop        ; Jump back to the top if esi < 6

    ;  cmp     byte [rdi], 0   ; Make sure the 8th byte is a NUL
                              ; char (end of string).
    ;  jne     .out_ret        ; Return false if it isn't a NUL char.

    ;  mov     rax, 1          ; Return true
;.out_ret:
     ; ret
```

Figure 7. Validating userID 2 different methods

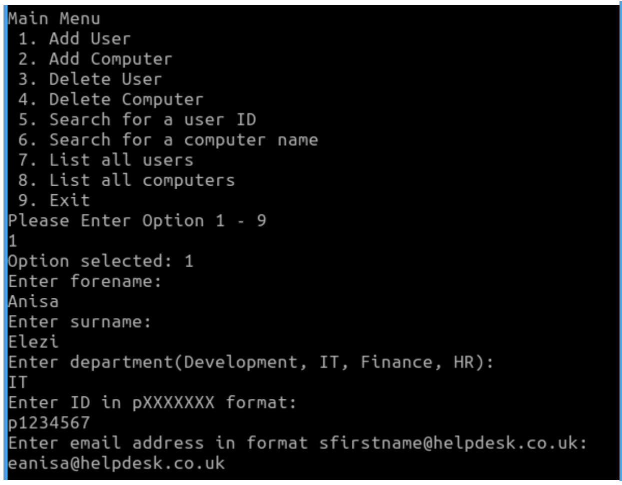
There is another method that I worked on validating but it still didn't work, although the logic behind is ok. The same logic would be followed on validating computer name

To test the program, it was mor comfortable to work from terminal because in SASM needed the input before executing.

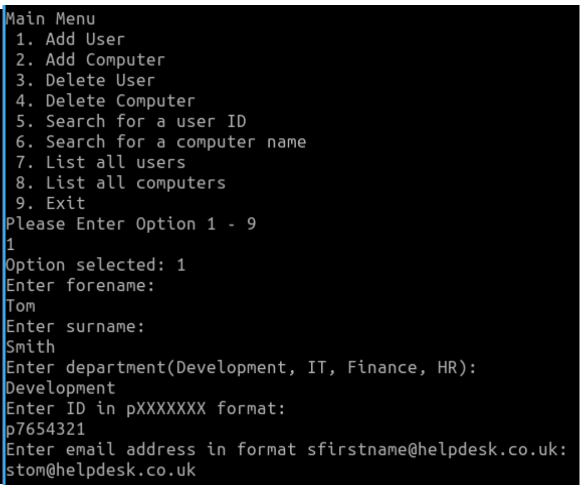


```
malware@malware-vm:~/asm$ build_asm_v8.sh CW1-Anisa-19177366.asm
Source file to build: CW1-Anisa-19177366.asm
Using nasm for assembly.
Running assembler command:
nasm -g -f elf64 -o CW1-Anisa-19177366.o CW1-Anisa-19177366.asm
Success: Assembler executed successfully.
Outputted object file is: CW1-Anisa-19177366.o
Linking...
Using using gcc for linking.
Running linking command:
gcc CW1-Anisa-19177366.o  -lasm_io -no-pie -o CW1-Anisa-19177366
Success: Linker executed successfully.
Outputted executable file name is: CW1-Anisa-19177366
malware@malware-vm:~/asm$ ./CW1-Anisa-19177366
```

Figure 8. Build and run

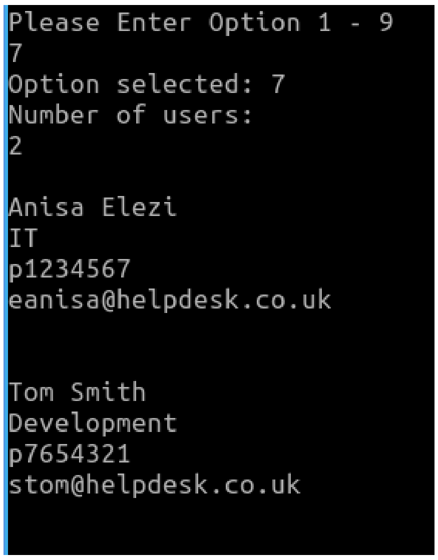


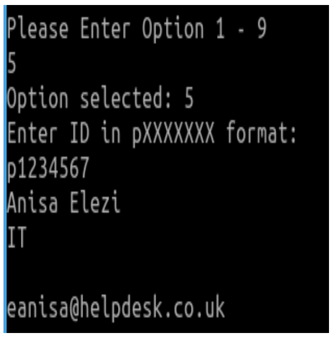
```
Main Menu
 1. Add User
 2. Add Computer
 3. Delete User
 4. Delete Computer
 5. Search for a user ID
 6. Search for a computer name
 7. List all users
 8. List all computers
 9. Exit
Please Enter Option 1 - 9
1
Option selected: 1
Enter forename:
Anisa
Enter surname:
Elezi
Enter department(Development, IT, Finance, HR):
IT
Enter ID in pXXXXXXX format:
p1234567
Enter email address in format sfirstname@helpdesk.co.uk:
eanisa@helpdesk.co.uk
```



```
Main Menu
 1. Add User
 2. Add Computer
 3. Delete User
 4. Delete Computer
 5. Search for a user ID
 6. Search for a computer name
 7. List all users
 8. List all computers
 9. Exit
Please Enter Option 1 - 9
1
Option selected: 1
Enter forename:
Tom
Enter surname:
Smith
Enter department(Development, IT, Finance, HR):
Development
Enter ID in pXXXXXXX format:
p7654321
Enter email address in format sfirstname@helpdesk.co.uk:
stom@helpdesk.co.uk
```

Figure 9. Adding 2 users


```
Please Enter Option 1 - 9
7
Option selected: 7
Number of users:
2

Anisa Elezi
IT
p1234567
eanisa@helpdesk.co.uk


Tom Smith
Development
p7654321
stom@helpdesk.co.uk
```


```
Please Enter Option 1 - 9
5
Option selected: 5
Enter ID in pXXXXXXX format:
p1234567
Anisa Elezi
IT

eanisa@helpdesk.co.uk
```

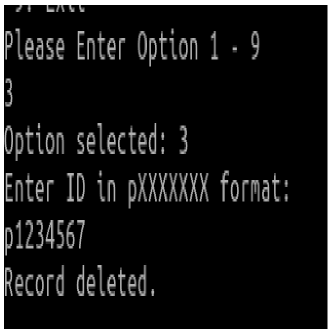
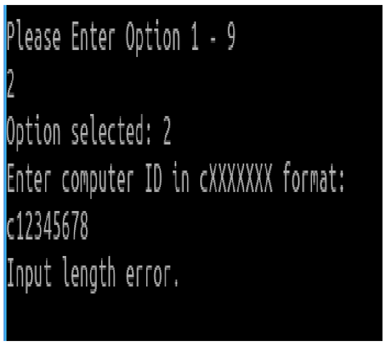

```
Please Enter Option 1 - 9
3
Option selected: 3
Enter ID in pXXXXXXX format:
p1234567
Record deleted.
```

Figure 10. ***List all users*** ***search*** *user by id* ***delete_user*** *by id*

Adding user with longer computer name is as follow


```
Please Enter Option 1 - 9
2
Option selected: 2
Enter computer ID in cXXXXXXX format:
c12345678
Input length error.
```

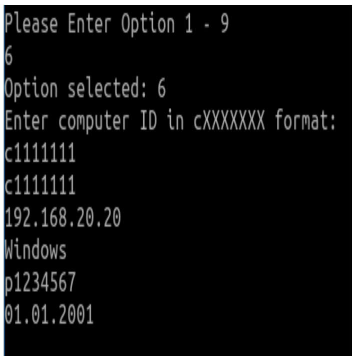
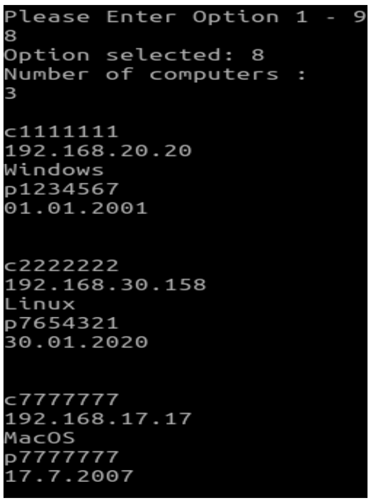
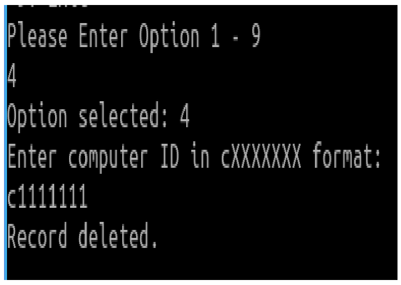

```
Please Enter Option 1 - 9
6
Option selected: 6
Enter computer ID in cXXXXXXX format:
c1111111
c1111111
192.168.20.20
Windows
p1234567
01.01.2001
```

Figure 11. Invalid computer name and search by computer name and delete computer


```
Please Enter Option 1 - 9
8
Option selected: 8
Number of computers :
3

c1111111
192.168.20.20
Windows
p1234567
01.01.2001


c2222222
192.168.30.158
Linux
p7654321
30.01.2020


c7777777
192.168.17.17
MacOS
p7777777
17.7.2007
```


```
Please Enter Option 1 - 9
4
Option selected: 4
Enter computer ID in cXXXXXXX format:
c1111111
Record deleted.
```

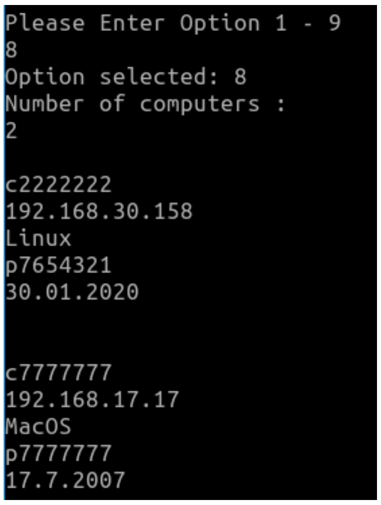

```
Please Enter Option 1 - 9
8
Option selected: 8
Number of computers :
2

c2222222
192.168.30.158
Linux
p7654321
30.01.2020


c7777777
192.168.17.17
MacOS
p7777777
17.7.2007
```

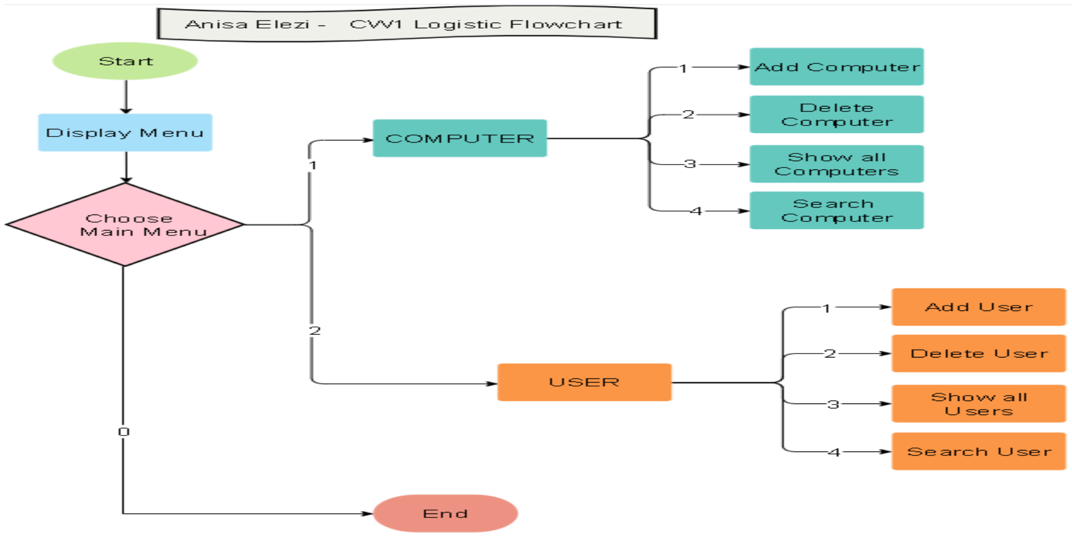
Figure 12. List of registered computer deletion by computer name list after deletion

3. Ethical and legal consideration

Regard ethical and legal issue, this program is real-time with cache memory, that ensure secure data, which will be destroyed as soon as we close the program.

4. Conclusion and future works

As it was demonstrated above, for simplicity the main menu is a single level menu. If I have time, I will try another model, which is with ***two layers menu*** as follow, and all the data to store in secure with multi-level of authentication and secure database.


Anisa Elezi - CW1 Logistic Flowchart
Start
Display Menu
Choose Main Menu
1
COMPUTER
1
Add Computer
2
Delete Computer
3
Show all Computers
4
Search Computer
2
USER
1
Add User
2
Delete User
3
Show all Users
4
Search User
0
End