THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 13

Arrays and Final Project Information

Agenda

- **Arrays**
  - What is an array?
  - Array Memory Allocation
  - Array address computation (memory references)
  - Array Declaration
  - Array Characteristics
  - Two Dimensions Arrays
  - Accessing and Processing Arrays
  - Final Project Information

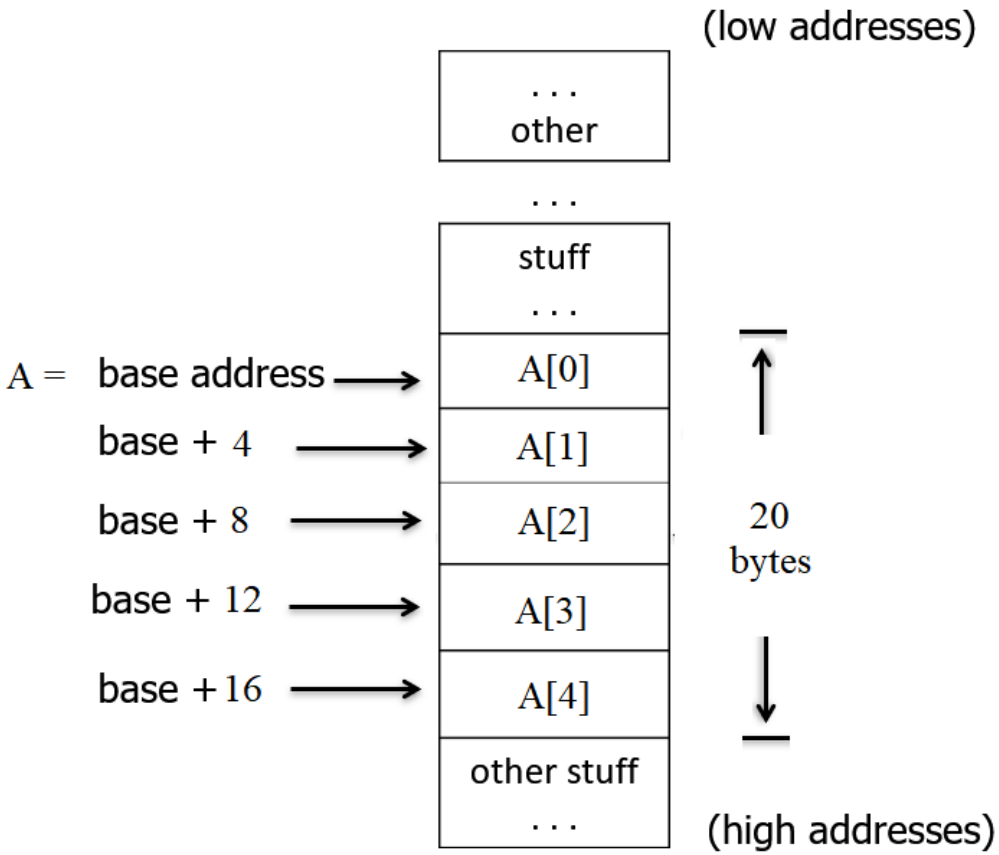
What is an array?

- In programming terminology, an array is a collection of data elements of same type where each element has an 'index' or location associated with it
- The simplest data structure is the one-dimensional array
- The name of the array is a pointer to the array base address that points to the first element of the array
- Example: int A [5] = [12, 34, 100, 2344, 56]
  - An array of five elements
  - Each element is integer type
  - Each element uses 4 bytes of memory (element-size)
  - Total memory storage for the array = 5 * 4 = 20 bytes

- Example: char A [7] = [“A”, “C”, “$”, “C”, „.…]
  - An array of seven elements
  - Each element is a character type
  - Each element uses 1-byte of memory (element-size)
  - Total memory storage for the array = 7 * 1 = 7 bytes

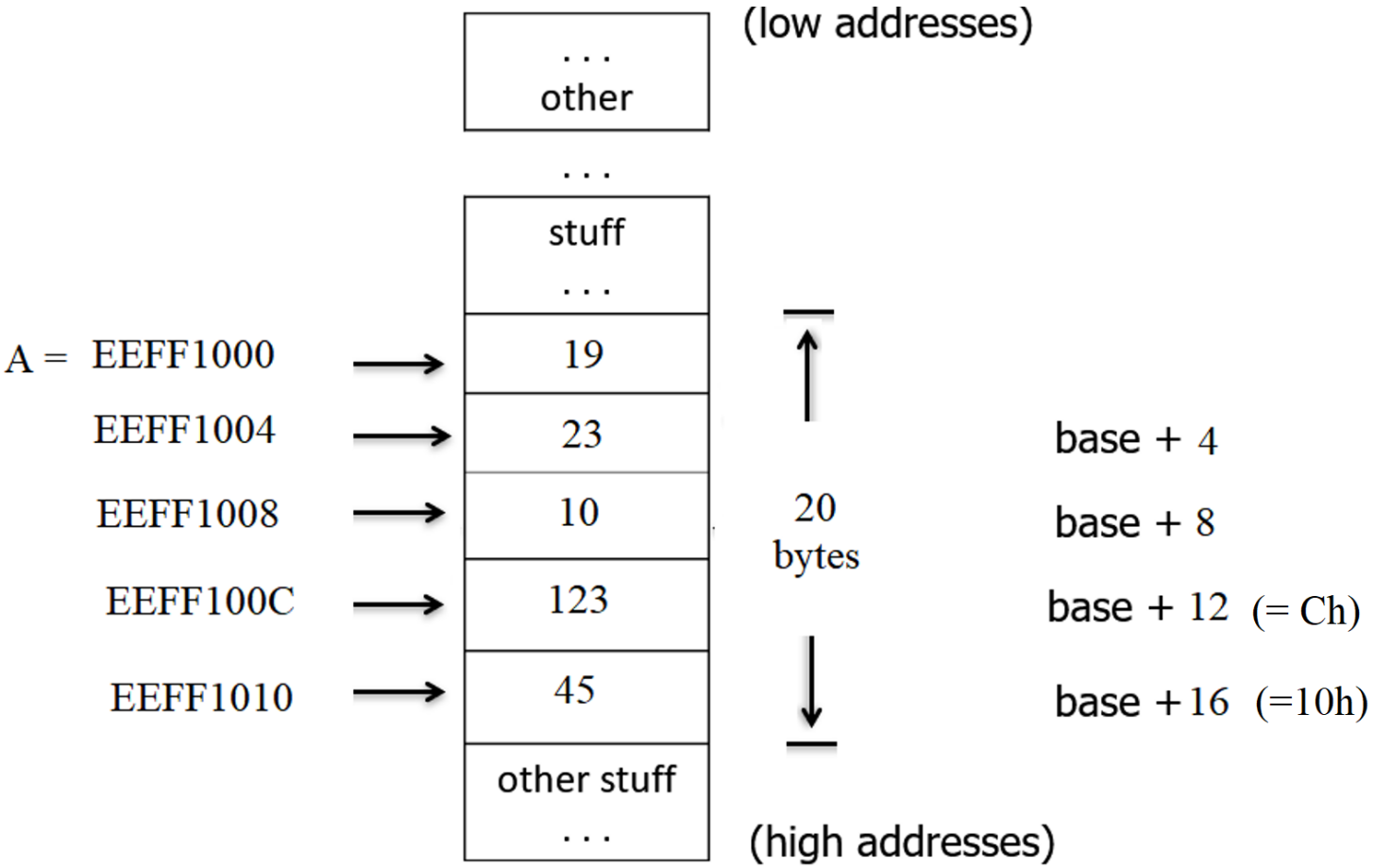
Array Memory Allocation

- Example: int A [5] = [12, 34, 100, 2344, 56]
  - An array of five elements
  - Each element is integer type
  - Each element uses 4 bytes of memory (element-size)
  - Total memory storage for the array = 5 * 4 = 20 bytes



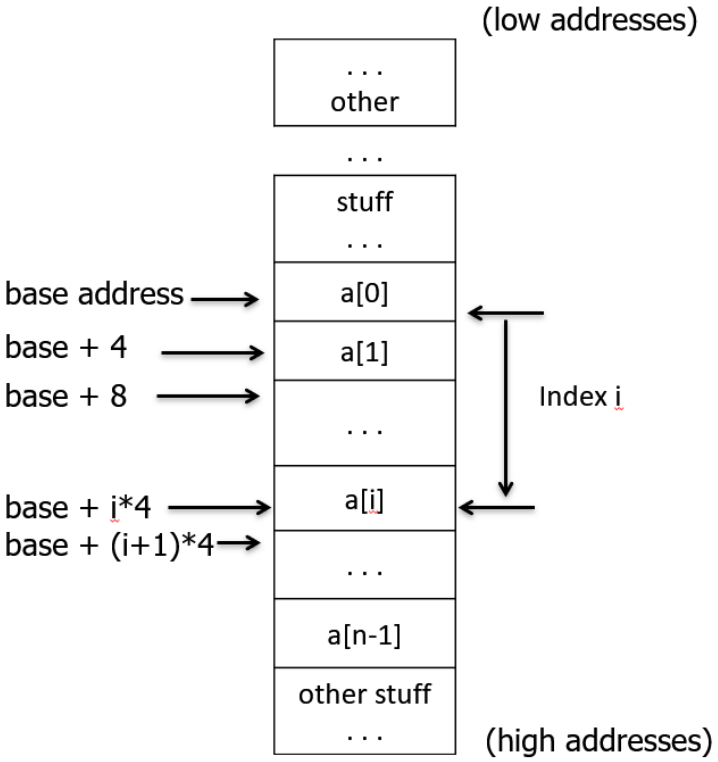
Array Memory Allocation

- Example: int A [5] = { 19, 23, 10, 123, 45}



General Array Memory Allocation

- Example: int A [n]
  - An array of "n" elements
  - Each element is integer type
  - Each element uses 4 bytes of memory (element-size = size of the array type)
  - Total memory storage for the array = (n * 4) bytes
  - Address of the first member of the array (= array name)



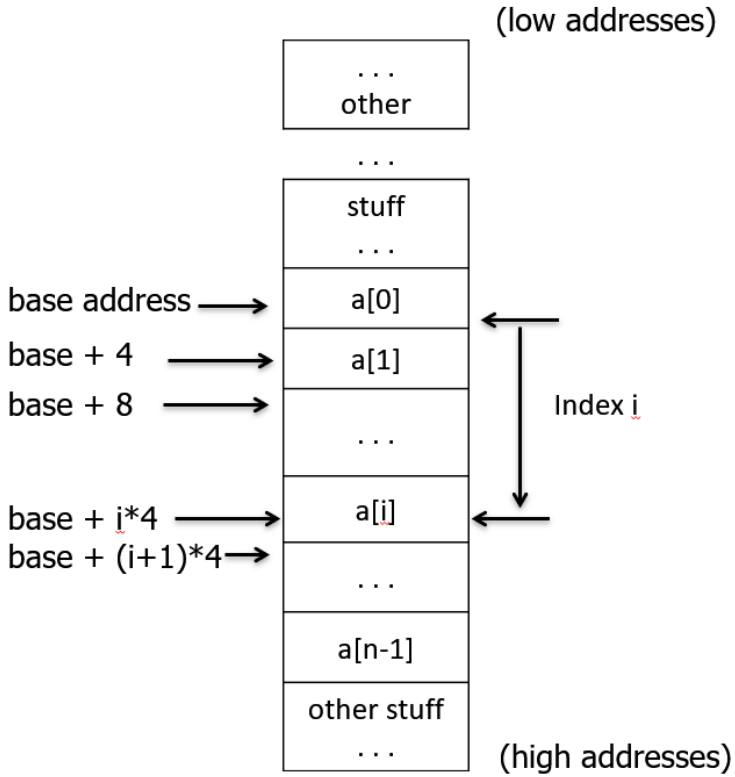
Array Address Computation

- What is the memory address of an element ”i”?

  address of a[i] = base address + (i * element_size)

  address of a[i] = base address + (i * size of the array type)

  Address of the first member of the array (= array name)



Array Characteristics

- An array occupies contiguous memory space. I.e., no memory gaps between elements
- Uniform: Every memory of the array has same memory size
- Calculation of memory location of any element can be calculated by knowing:

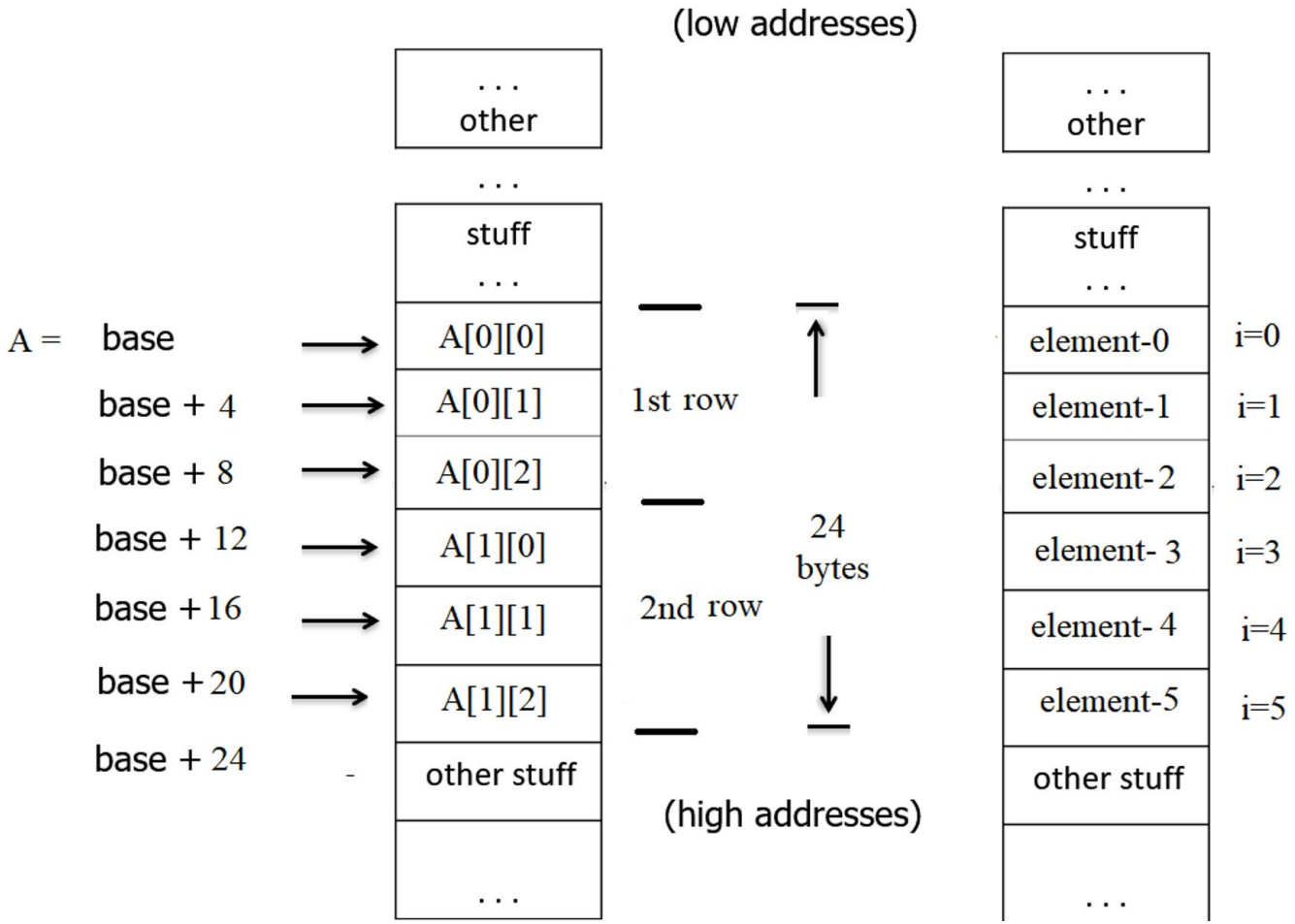
  1- Address of the first member of the array (= array name)

  2- Size of the array element. (int (4), char (1), double (8)

  3- Index (offset) of the element

Two dimensions Arrays

- Memory address of A[i][j]= base-address + [(i * ((# of element per row) + j] * element-size
- "i" and "j" start from 0
- From memory point of view, a two dimensions array is a one dimension array



Two dimensions Array

- Example: int A [2][3]
  - An array of two rows and three columns
  - Each element is integer type
  - Each element uses 4 bytes of memory (element-size)
  - Total number of elements in the array = 2*3 = 6
  - Total memory storage for the array = 6* 4 = 24 bytes

- Memory address of A[i][j]= base-address + [(i * (# of element per row) + j] * element-size

Declaring an Array

- When you declare an array, you actually declare a contiguous block of memory
- Array of five initialized 32-bit integers

```
section .data
        arwd1: dd  5, 10, 20, 12, 25
```

- Array of ten zero initialized 32-bit integers

```
section .data
      arwd2: times 10 dd 0        ; repeat same value by times prefix
```

- Array of twenty elements, initialized as follow

```
section .data
arwd3:   times 5     dd   10        ; 5 elements, each =10
         times  10  dd   30         ; 10 elements, each =30
         dd       40, 50, 60, 70   ; different values of other 5 elements
                                   ; the array has 19 elements
```

- You can declare uninitialized array in the .bss section

Array of ten 32-bit integers

```
section .bss
        arwd4:    resd   10
```

Access and Processing Arrays

- Indirect addressing is often used to access an array element.
- Indirect addressing involves:

  1- Array's base address is stored in a register

  2- Then the register is used as the base for indexing

- Example: Consider the following code to inialize each element to be zero

```
    mov ebx, IntArray       ;  Base address of the IntArray is stored in ebx
    mov  ecx, 6             ;  Size of the array (=6) is stored in ecx.
                            ; ecx register will be used as a loop counter
top:
    mov   DWORD [ebx], 0    ;  initialize first array element with a value = 0
                            ; ebx holds the base address
    add ebx, 4              ; access the following array element.
                            ; move 4 bytes since the element-size is 4 bytes (integer)
    loop  top               ; substract 1 from exc register
                            ; if the result is not zero, jump to the top of the loop
                            ; if ecx is zero, exit the loop
```

Example 1: Access and Print Arrays

- Use loop to print array's elements

```
; Arr1.asm     declare an integer array and  print its member values

SECTION .data              ; Data section
  msg1:    db "Here are the array elment values!",10, 0
  msgL1:  equ $-msg1
  msg2:    db "Here are the array elment values multiplied by 4!",10, 0
  msgL2:  equ $-msg2

  ard1:  dd  1, 3, 5, 7, 9
  ard1N:  equ  ($-ard1) / 4      ;  number of elements = array length / 4
SECTION .text
 global main
 main:
      push ebp
      mov ebp, esp

       mov ecx,msg1             ; print message#1
       mov edx,msgL1
       call PString

       ; save array base address in ebx and save size of the array in ecx
        mov  ebx, ard1
        ;mov  ecx, 5          ; store numebr of elements in ecx
        mov ecx, ard1N
       ; loop to print the arrray element
```

```
PrintArray:
     mov eax, [ebx]     ; move the value to [ebx] to eax
     call printDec
     call  println
     add ebx, 4
     loop PrintArray

     ; exit the program and cleaning
     mov esp, ebp
     pop ebp
     ret
```

PString: Print string. Definded as before in previous lectures

printDec: Print integers using system calls. Defined as before in the previous lectures

println: Print new empty line. Defined as before in the previous lectures

```
Here are the array elment values!
1
3
5
7
9
```

Example 2: Access and Processing Arrays

- Consider the following c-program that

  1- initializes each array element,

  2- calls a function to multiply each array element with 4

```
#include <stdio.h>
 void multiplyfour (int size, int aa[]);
 int main ()
{
 int i;
 int size = 6;
 int array [6]              // declare an integer array of 6 elements
 for (i=0; i < size;  i++) {
 array [i] =  i ;
 {
 multiplyfour (size, array)       // pass the array and its size to the function
                                  // the function will access each array element
                                  // and multiply each array element by 5
}
```

Q: How the assembly code of "multiplyfour" function look like?

Example 2: Implement "Print Array" and "Mby4 "Functions

```
; Arr3.asm      declare an integer array, print its member values, multiply by 4, print again

SECTION .data               ; Data section
   msg1:     db "Here are the array elment values!",10, 0
   msgL1:  equ $-msg1
   msg2:     db "Here are the new array elment values after multiplied by 4!",10, 0
   msgL2:  equ $-msg2
  ard1:  dd  1, 3, 5, 7, 9
  ard1N:  equ  ($-ard1) / 4       ;  number of elements = array length / 4
SECTION .text
 global main
 main:
        push ebp
        mov ebp, esp
        mov ecx,msg1              ; print message#1
        mov edx,msgL1
        call PString

        ; save array base address in ebx and save size of the array in ecx
        mov  ebx, ard1
        mov  ecx, ard1N

       ;call PrintArray to print the arrray element
        call PrintArray

       ; print message2
        mov ecx,msg2
        mov edx,msgL2
        call PString

      ; restore the array addres and its size  to the stack before calling the Mby4 function
       mov  ebx, ard1
       mov  ecx, ard1N
       call Mby4
```

```
root@kali-Test:~/Desktop/Week-13# ./Arr3
Here are the array elment values!
1
3
5
7
9
Here are the new array elment values after multiplied by 4!
4
12
20
28
36
```

```
; restore the array addres and its size  to the stack before calling PrintArry function
 mov  ebx, ard1
 mov  ecx, ard1N
```

```
 call PrintArray     ;call PrintArray to print the new arrray element
 mov esp, ebp        ; exit the program and cleaning
 pop ebp
 ret
```

```
 PrintArray:
     section .text
          push ebp
          mov ebp, esp

 top:
          mov eax, [ebx]      ; move the value of [ebx] to eax
          call printDec
          call println
          add ebx, 4
          loop top

          mov esp, ebp   ; destory the stack
          pop ebp
          ret
```

```
 Mby4:
     section .text
          push ebp
          mov ebp, esp

 top1:
          mov  eax, [ebx]          ; access first array element. Move its value to eax
          shl eax, 2               ; shift left operation. Multiply the value by 4
          mov [ebx], eax           ; store the new value back to the same array elemen
          add ebx, 4               ; move to the next array element
          loop top1                ;  loop back to the top if the register ecx > 0
```

Week 13 Assignments

- **Learning Materials**

  1- Week 13 Presentation

  2- Reading Ch.11: Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux,

- **Assignment**

  1- Complete “Lab 13 assignment” by coming Sunday 11:59 PM.

Final Project Information

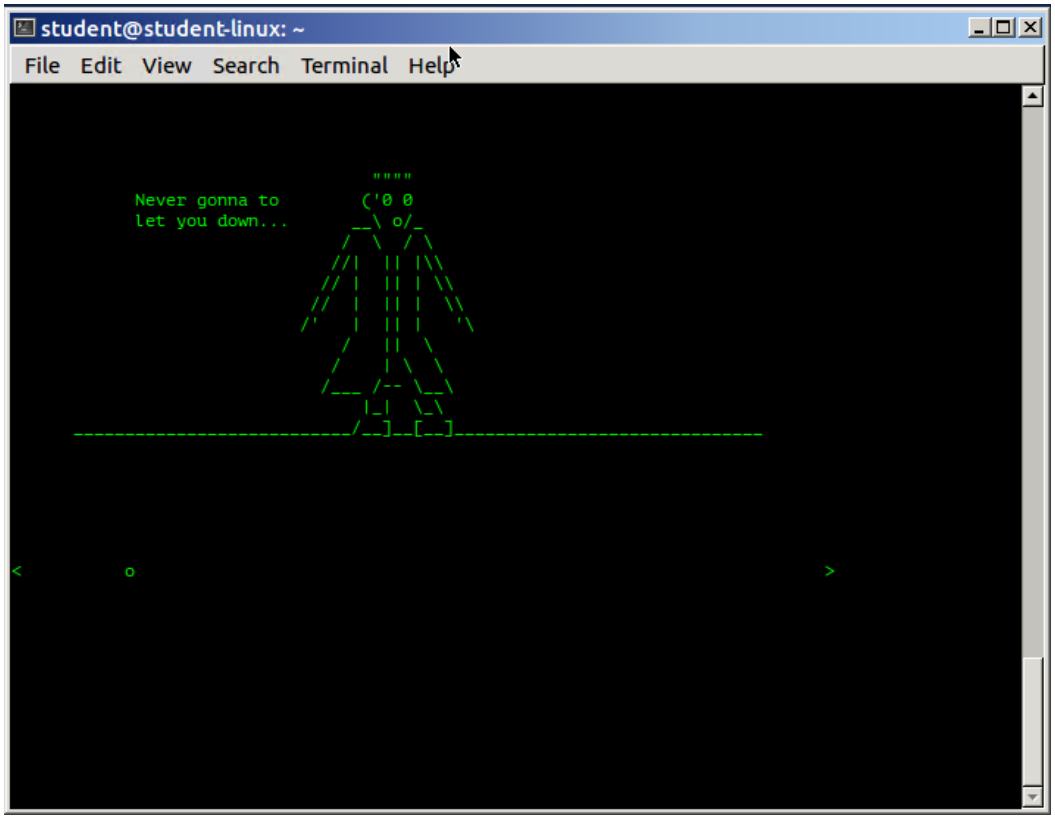
- Please make sure you complete the following tasks before submitting your final project
- In the first page, put your name
- In the following pages, include
- A flow chart that shows the main functions of the “Telent” client functions
- Within the client code, include comments that explain the main task of each function
- Within each function, explain the main task of each line
- Compile the “Telnet” client and include screen shoots for the completion steps
- Test the “Telnet” client and include screen shoots of the test results
- Submit your report in the assignment section by Dec. 6, 2019

Test “Telnet” client

- Test the “Telnet” client using Arizona servers (assume the client name T1.exe)

  ./T1.exe 10.139.201.23 24 (port 24)

  ./T1.exe 10.139.201.23 23 (port 23)



Test “Telnet” client

- Or test “Telnet” client using outside server (assume the client name T1.exe)

  ./T1.exe 167.114.65.195 23 (port 23)

  ./T1.exe 35.160.169.47 23 (port 23)