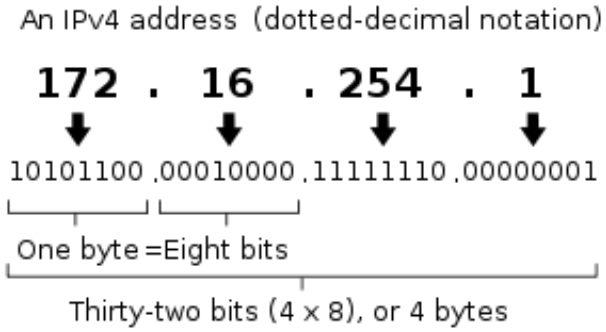
CS 531: Fundamentals of Systems Programming Homework # 2 Rubric

Most computers on the internet have a 32 bit **Internet Protocol Version 4** (IPv4) address. As reading these addresses would be difficult using binary or hexadecimal notation, IPv4 addresses are usually represented in **dotted decimal notation**.

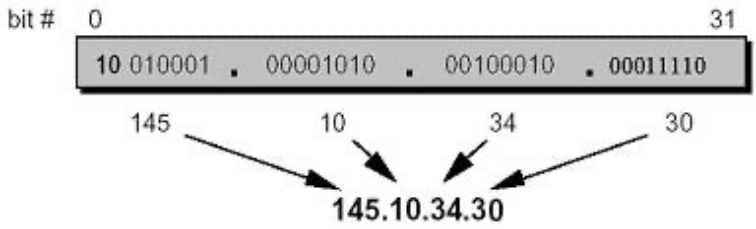
For purposes of representation, the 32 bits composing the address may be divided into four octets (bytes) written as decimal numbers, each ranging from 0 to 255, and concatenated as a character string with a full stop (ASCII 46) between each number.

For example:



The first two components of the address indicate the computer's *locality* on the network. In the above example, the locality is specified by the ordered pair: **172** and **16**.

Another example:



Locally, computers are often known by an **alias** (nickname) as well. You will design and write a program to process a list of Internet addresses from file "CS531_Inet.txt". Each line in "CS531_Inet.txt" will contain an addresses/alias pair as shown below:

Sample CS531_Inet.txt:

```
111.22.3.44 platte
131.250.95.21 jet
172.66.7.88 wabash
111.22.5.66 green
131.250.47.63 baker
```

For this assignment, you will create your own test data file using the sample format. Grading will involve using different data files with the same format.

Each address and each alias within the file is unique. Alias names are NOT case sensitive.

Program structure and design:

Create a structure type called **address_t** with components for the four integers of an IPv4 address along with a fifth component in which to store an associated alias (key field) of up to 10 characters. You will then create a singly linked list of address_t structures which will contain all of the address/alias pairs read in from the "CS531_Inet.txt" file. For this exercise, the linked list may be unsorted.
Once the linked list has been created, the user will receive the following menu options:

```
1)  Add address
2)  Look up address
3)  Update address
4)  Delete address
5)  Display list
6)  Display aliases for location
7)  Save to file
8)  Quit
```

1) **Add address**: Prompt user for an IPv4 address/alias pair. Both data elements are read in from the keyboard as character strings. The address string will be parsed and the four component integers will be stored separately (hint: use sscanf()). Each of the four component integers must be within the range: [0-255]. If any component integers are out of range, display an error message and re-prompt for a new address. If the alias has a strlen() value > 10, display an error message, and re-prompt. If either the address or alias already exists within the list, display an error message and redisplay the menu.

2) **Look up Address**: Prompt user for an alias, and display the corresponding address. If the alias does not exist within the list, display an error message and redisplay the menu.

3) **Update address**: Prompt user for an alias and display the corresponding address. Allow user to update the address and have the newly entered address replace the old address within the linked list ***assuming*** the new address does not already exist within the list. Each of the four component integers must be within the range: [0-255]. If any component integers are out of range, display an error message and re-prompt for a new address. If the alias does not exist within the list, or the newly entered address is a duplicate, display an error message and redisplay the menu.

4) **Delete address**: Prompt user for an alias, and display the corresponding address. Confirm that user wishes to delete this node from the linked list. If the alias does not exist within the list, display an error message and call **displayList()** in order to show user the current alias/address pairs within the list. Then redisplay the menu.

5) **Display list**: Display all address/alias pairs within the list. For this exercise, the list is unsorted. If the list is empty, print a message to that effect and redisplay the menu. Also display the number of nodes within the list.

6) **Display aliases for location**: Prompt user for an address location. (i.e. two separately entered values between 0 – 255). If either value is out of range, re-prompt. List all aliases that map to this location. If the location does not exist within the list, display an error message and redisplay the menu.

7) **Save to file**: Prompt user for a file name, and save the revised list to the specified file keeping file format consistent with that of the input file (i.e. one address/alias pair per line.)

8) **Quit**: Exit program

You may have one global variable as follows:

```
struct address_t
{
   int octet[4]; char alias[11];
   struct address_t *next;
}
struct address_t *head = NULL;
```

Rubric 10 points:

- Is the source code well documented and formatted using clearly readable indentation and white space (while viewed within **vi**)? **1 point**
- Is the singly linked list of address_t structures properly implemented? **1 point**
- Each menu option shall be implemented as a separate UDF. Are each of the 8 UDFs properly implemented? **8 points** (1 point each)

Submit via Blackboard by Sunday March 12, 11:59 PM.