Program 2

Use the Kali VM and netdump files as you did in Program 1. You will be adding code to netdump.c.

Please note the following:

- You will add code to the function/routine `raw_print()`
  This is where you will parse the packets – it is executed every time a new packet is captured. For example, to print out the number of IP packets you would declare a global variable that increments the number of IP packets seen and prints this number in the `raw_print()` routine.
  - the packet is contained in the array `p` where `p[0]` is the first byte of the HW destination address
  - `caplen` is the length of the packet
- You will add code to the function/routine `program_ending()`
  This is where you will put your final counts – it is executed only when you stop the program.
- You do not add code to any function/routine other than `raw_print()` and `program_ending()`
- The function/routine `default_print()` prints the packet out in hex
  You can use this function to decode a packet by hand to see if your print statements are accurate.
- You may create global variables, if necessary
- You may create helper functions *within* netdump.c
- Please do not create files or make changes in files outside of netdump.c

Steps to complete:

0. Reminders: To start the program, type `sudo ./netdump` and to end the program, press Ctrl+C. If you make any changes in netdump.c, you must call `make netdump` to compile the changes.

   It might be useful to pipe your output to a text file so you can run the code for a while and then use the find function to locate specific packets (`sudo ./netdump > yourfile.txt`). You can always open the text file from the GUI file folder system – you don't need to do this in the terminal. **Make sure you use Ctrl+C to stop netdump or you risk filling up your disk space and crashing your VM.**

   If, for some reason, you are starting netdump from scratch and not from Program 1, make sure you have made the changes noted in Program 1 for `e_type` and `bmf_dump`. You also need to have the pcap library installed.

1. In raw_print(), decode and print the Ethernet header. Print the destination *and* source addresses using colons (e.g. Destination Address= 00:16:22:F3:33:45). [Decode & print means look at pages 98-100 to see the fields are in an Ethernet packet, what information the fields contain, and how long the fields are.]

   To help you get started, this line of code will print the HW destination address out:
   ```
   printf("DEST Address = %02X:%02X:%02X:%02X:%02X:%02X\n", p[0], p[1],p[2],p[3],p[4],p[5]);
   ```

2. Print the Ethernet type/length field as Type = (hex value) or Len = (in decimal).

   To help you get started, this line of code will print the type/length field
   ```
   uint16_t e_type; //this goes after u_int caplen = h->caplen;
   e_type = p[12] * 256 + p[13];
   printf("E_Type = 0x%04X ", e_type);
   ```
   <- Looking at `0x%04X`, the 0x signifies the output is in hex (just printed ASCII characters); the %04X is a format specifier that prints the e_type

3. Print the Ethernet protocol being used. E.g. If the type is 0x0800, then print Payload = IPv4. If the type is 0x0806, then print Payload = ARP.

   To help you get started, this line of code will print if the payload is IP:
   ```
   if (e_type == 0x0800) printf(" -> IP\n");
   ```

4. Add a counter to keep track of the number of ARP and IP packets seen while running the code. Print these total numbers of packets seen in `program_ending()`.

5. Decode and print the ARP header and ARP request and reply headers using the information about ARP packets in your textbook. Print the IP addresses in standard notation (e.g. 129.186.215.40). Print all other values in the data (i.e. any value not already decoded should be printed as the payload).

6. Decode and print the IP header using the information about IP packets in your textbook. Print the IP addresses in standard notation. Print Flag bits individually and state their meaning (see examples). Print all other values in the data.

7. Decode and print the ICMP header using the information about ICMP packets in your textbook. Print the IP addresses in standard notation. Use the tables in Chapter 6 to help you determine what goes in the Parameter and Information fields – be specific in your output. Print all other values in the data.

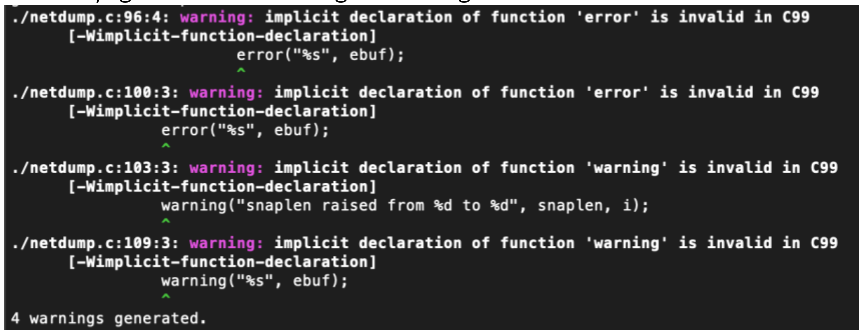
8. Decode and print the TCP header. Print Flag bits individually and state their meanings (see examples). Print any options as hex values; if there are no options, print "No options". Print all other values in the data, if data exists.

9. Add counters to keep track of the number of ICMP packets and TCP packets seen while running the code. Print the total number of ICMP and TCP packets seen in `program_ending()`.

10. Determine how to tell if a packet is a DNS packet from Chapter 7. Add a counter to keep track of the number of DNS packets seen while running the code and print the total in `program_ending()`.

11. Format your output. Please work on making your printouts readable by printing with newlines (\n) and indentations/tabs (\t). Some people like to put lines (i.e. -------------) between packets to separate them.

12. Resolve any warnings or errors and make sure your code compiles.
    You may ignore the following 4 warnings:



```
./netdump.c:96:4: warning: implicit declaration of function 'error' is invalid in C99
      [-Wimplicit-function-declaration]
                        error("%s", ebuf);
                        ^
./netdump.c:100:3: warning: implicit declaration of function 'error' is invalid in C99
      [-Wimplicit-function-declaration]
                error("%s", ebuf);
                ^
./netdump.c:103:3: warning: implicit declaration of function 'warning' is invalid in C99
      [-Wimplicit-function-declaration]
                warning("snaplen raised from %d to %d", snaplen, i);
                ^
./netdump.c:109:3: warning: implicit declaration of function 'warning' is invalid in C99
      [-Wimplicit-function-declaration]
                warning("%s", ebuf);
                ^
4 warnings generated.
```

    Submit netdump.c on Canvas.

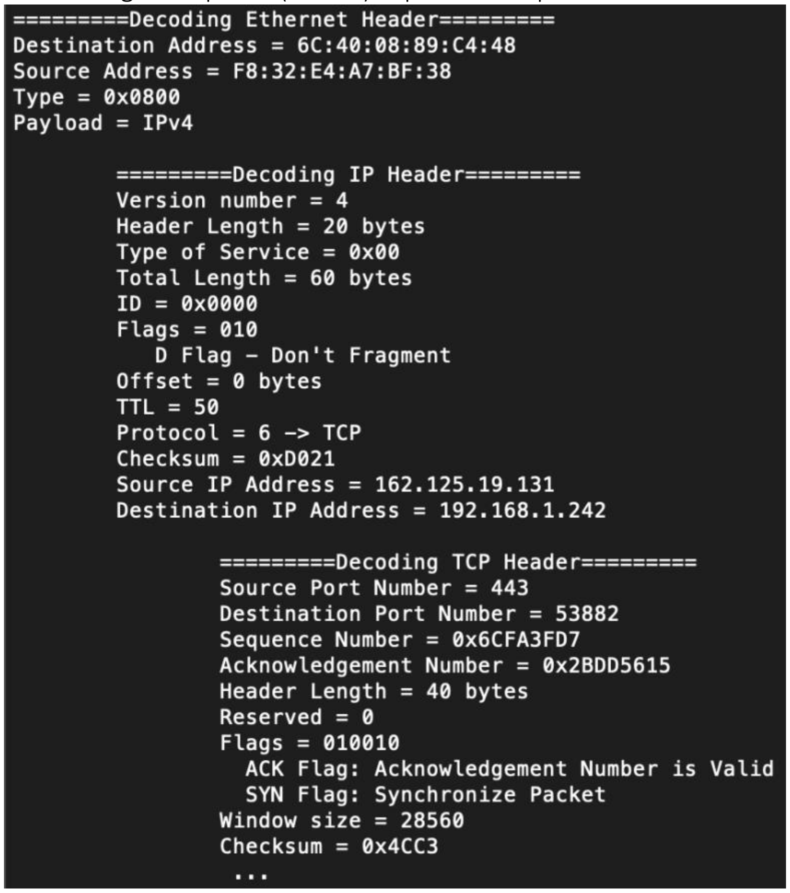
Example expected output can be found on the next page.

Example 1

Packet received:

```
6c40 0889 c448 f832 e4a7 bf38 0800 4500
003c 0000 4000 3206 d021 a27d 1383 c0a8
01f2 01bb d27a 6cfa 3fd7 2bdd 5615 a012
6f90 4cc3 0000 0204 05a0 0402 080a 0f61
dc56 4a31 da2f 0103 030a
```

Formatting Example of (Partial) Expected Output:



```
=========Decoding Ethernet Header=========
Destination Address = 6C:40:08:89:C4:48
Source Address = F8:32:E4:A7:BF:38
Type = 0x0800
Payload = IPv4

        =========Decoding IP Header=========
        Version number = 4
        Header Length = 20 bytes
        Type of Service = 0x00
        Total Length = 60 bytes
        ID = 0x0000
        Flags = 010
           D Flag - Don't Fragment
        Offset = 0 bytes
        TTL = 50
        Protocol = 6 -> TCP
        Checksum = 0xD021
        Source IP Address = 162.125.19.131
        Destination IP Address = 192.168.1.242

                =========Decoding TCP Header=========
                Source Port Number = 443
                Destination Port Number = 53882
                Sequence Number = 0x6CFA3FD7
                Acknowledgement Number = 0x2BDD5615
                Header Length = 40 bytes
                Reserved = 0
                Flags = 010010
                  ACK Flag: Acknowledgement Number is Valid
                  SYN Flag: Synchronize Packet
                Window size = 28560
                Checksum = 0x4CC3
                 ...
```

(continue decoding – this example is not complete)

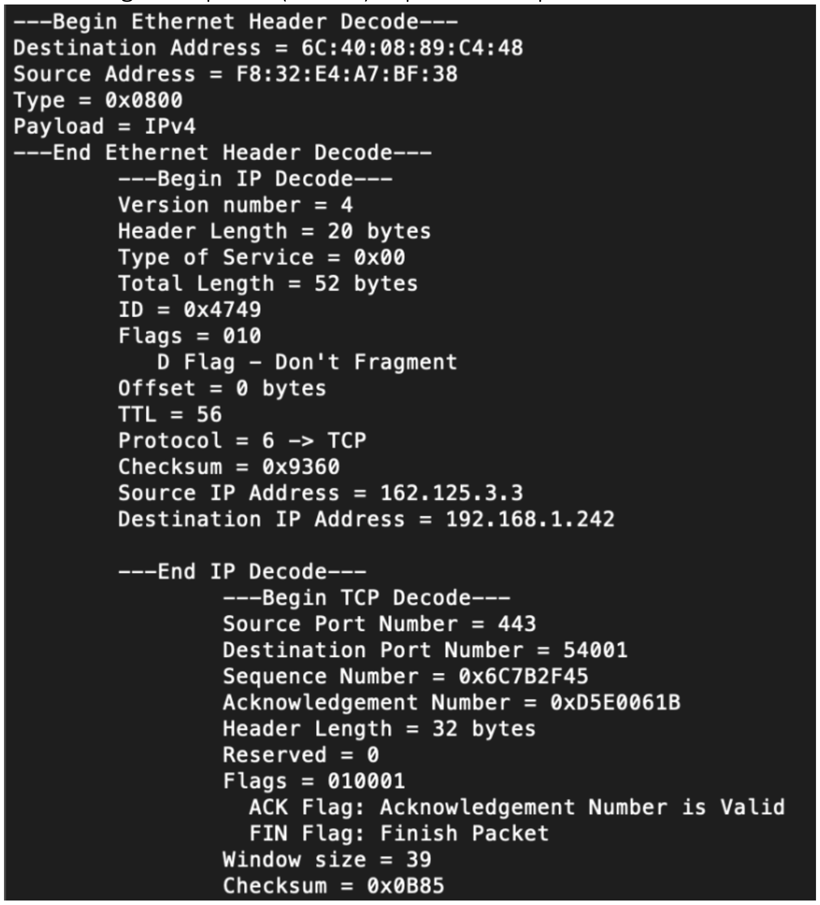
There is another example on the next page.

Example 2

Packet received:

```
6c40 0889 c448 f832 e4a7 bf38 0800 4500
0034 4749 4000 3806 9360 a27d 0303 c0a8
01f2 01bb d2f1 6c7b 2f45 d5e0 061b 8011
0027 0b85 0000 0101 080a 4a1f 61e4 4a46
c042
```

Formatting Example of (Partial) Expected Output:



```
---Begin Ethernet Header Decode---
Destination Address = 6C:40:08:89:C4:48
Source Address = F8:32:E4:A7:BF:38
Type = 0x0800
Payload = IPv4
---End Ethernet Header Decode---
        ---Begin IP Decode---
        Version number = 4
        Header Length = 20 bytes
        Type of Service = 0x00
        Total Length = 52 bytes
        ID = 0x4749
        Flags = 010
           D Flag - Don't Fragment
        Offset = 0 bytes
        TTL = 56
        Protocol = 6 -> TCP
        Checksum = 0x9360
        Source IP Address = 162.125.3.3
        Destination IP Address = 192.168.1.242

        ---End IP Decode---
                ---Begin TCP Decode---
                Source Port Number = 443
                Destination Port Number = 54001
                Sequence Number = 0x6C7B2F45
                Acknowledgement Number = 0xD5E0061B
                Header Length = 32 bytes
                Reserved = 0
                Flags = 010001
                  ACK Flag: Acknowledgement Number is Valid
                  FIN Flag: Finish Packet
                Window size = 39
                Checksum = 0x0B85
```

(continue decoding – this example is not complete)