Project 4

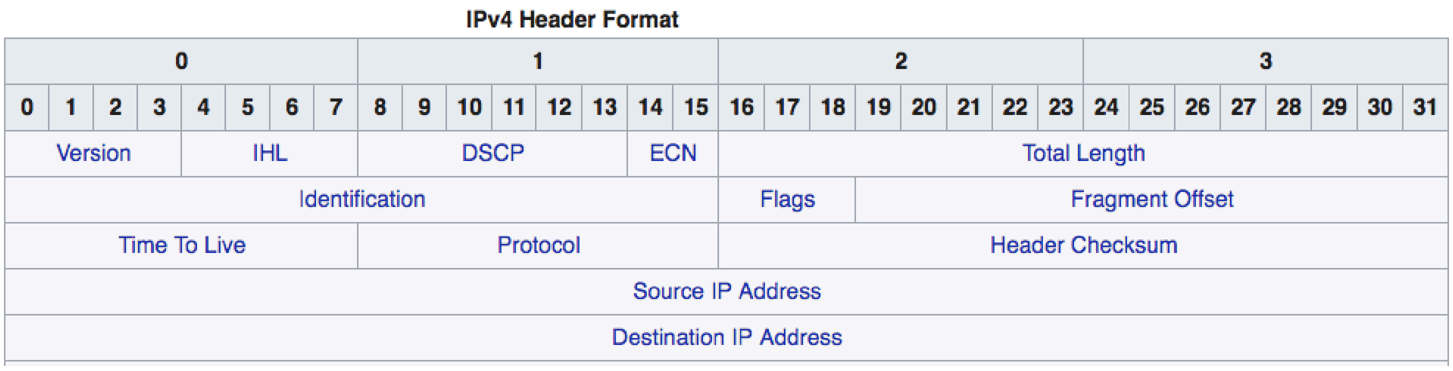
Due: March 27th (Wednesday) before 11:59 pm

1. Learning Goals

The purpose of this assignment is to have practice with low-level programming.

2. Specifications

In this assignment you will write a C program **read-ipheader.c** that extracts various parts of the **IPv4 header** and performs error checking of the header.
This is the format of the header you are expected to work with.



IPv4 Header Format

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version				IHL				DSCP						ECN		Total Length															
Identification																Flags			Fragment Offset												
Time To Live								Protocol								Header Checksum															
Source IP Address																															
Destination IP Address																															

Figure 1: IP header format

For simplicity, we are not considering options field. In the figure the numbers do not indicate contents of the header, but the position of the byte and bits.

2.1. Generate header

To begin with, you are given the program **write-ipheader.c,** which generates an ip header in binary form. The program takes one command line argument <filename> which is the name of the output file.
To use this program to generate the input file for ipheader.c do the following:

1. Create a folder named **p4** in your **private/354** directory.

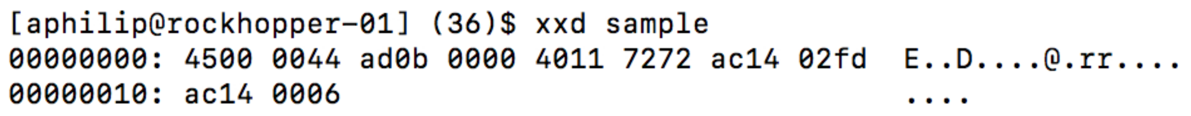
2. Copy the write-ipheader.c file into your p4 directory. You may find this file here: http://pages.cs.wisc.edu/~gerald/cs354/Spring2019/projects/p4/write-ipheader.c

3. Compile the program and store the executable in a file named **write-ipheader**.

4. Run the program using the following command:

```
./write-ipheader sample
```

5. The program should have created a file named **sample**. You can dump the contents of this file in hexadecimal format using ***xxd*** like this:



```
[aphilip@rockhopper-01] (36)$ xxd sample
00000000: 4500 0044 ad0b 0000 4011 7272 ac14 02fd  E..D....@.rr....
00000010: ac14 0006                                ....
```

2.2. Complete the main()

For the following parts of the assignment you have been provided with the file **read_ipheader.c** which contains skeleton code for the functions that you need to complete to make the program to work as expected. Copy the read_ipheader.c file into your p4 directory. You may find this file here: http://pages.cs.wisc.edu/~gerald/cs354/Spring2019/projects/p4/read-ipheader.c
The program takes in a single command line argument <input_file> which is the name of the file you just created in the previous step (sample in our case) that contains an ip header in binary form.

Complete the TODO in the main():

- read the input file into a char *hdr
- store the number of bytes read into n

In main(), we call the function **is_little_endian(),** to decide to exit the program if the machine is not little endian. Complete this function **is_little_endian()** to return 1 if machine is little endian or 0 otherwise.

2.3. Extract fields from header

Next you have to complete the following functions indicated in the code to extract some fields:

a) `unsigned int get_length(char *hdr)`

- This function extracts the **total length field** and returns it. This is the second and third byte as shown in the Figure 1 (Assuming we start at byte 0).
- The input is a pointer to header, whose contents is the ipheader (in binary) that you filled from the input file.
- The output of this function will be the total length that you extracted from the ip header (as an unsigned integer).

b) `unsigned int get_ip(char *hdr, int option)`

- This function is used to extract **source and destination IP address**. This is the third and the fourth word (one word = 32 bits) as shown in the Figure 1.
- The input is a pointer to header, and an option. If the option is 0, you will have to return the source IP. If the option is 1, you will have to return the destination IP.
- The output will be the **IP address** you extracted as an unsigned integer. Note that the main() prints this in decimal and hex format. You can use the hex format to cross verify with the contents of the header to confirm if you did the right thing.

`c) char *format_ip(unsigned int ip_int)`

- This function converts the IP address from a single 32-bit integer into the form A.B.C.D (dotted decimal notation) where A represents the value of the MSB in decimal and D represents the value of the LSB in decimal. For example, if the 32-bit value of an ip address in binary is 11000000 10101000 00000000 00000001 then the dotted decimal form of this ip address is 192.168.0.1.
- The output of the previous function (i.e., get_ip) is passed as input here.
- The output will be the IP address in dotted decimal format (i.e., A.B.C.D) as a char pointer.

You will do the rest of the functions in a similar manner:

d) **`int get_protocol(char *hdr)`**: returns the **8 bit protocol number** as an integer

e) **`int get_version(char *hdr)`**: returns the **4 bit version number** as an integer

f) **`char get_flags(char *hdr, int pos)`**: returns the **1 bit** (corresponding to the pos) from the **flags** field. This has a position argument to indicate which of the three flags is being extracted.

2.4. Validate Checksum

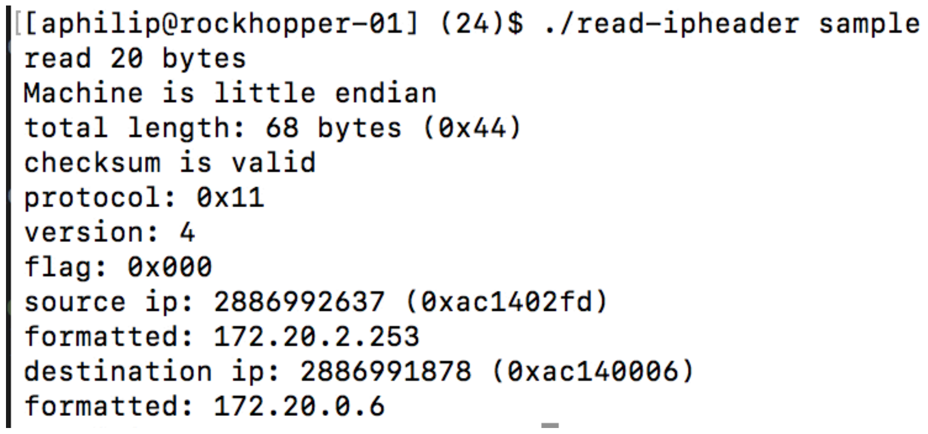
Next you have to complete the following function:

`int is_checksum_valid(char *hdr)`

If the checksum is valid the function has to return 1 else the function will return 0. This return value is used in the main() to print if the checksum is valid or not. To validate an ip header's checksum, you may follow the algorithm described here.

2.5. Sample output

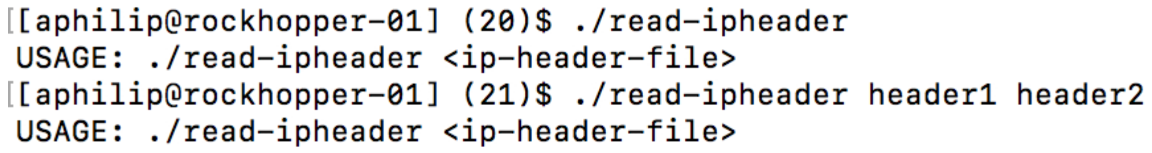
This is a sample output to show you the expected output for the sample header. Note that we will be using different test cases to grade your program.



```
[[aphilip@rockhopper-01] (24)$ ./read-ipheader sample
read 20 bytes
Machine is little endian
total length: 68 bytes (0x44)
checksum is valid
protocol: 0x11
version: 4
flag: 0x000
source ip: 2886992637 (0xac1402fd)
formatted: 172.20.2.253
destination ip: 2886991878 (0xac140006)
formatted: 172.20.0.6
```

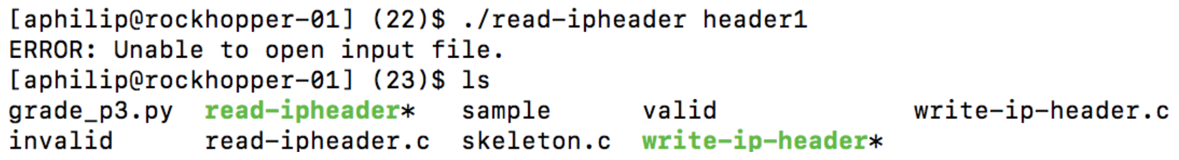
3. Error handling

- If the user invokes the list program incorrectly (for example, without an argument, or with two or more arguments), the program should print an error message and call exit(1)as shown below.



```
[[aphilip@rockhopper-01] (20)$ ./read-ipheader
USAGE: ./read-ipheader <ip-header-file>
[[aphilip@rockhopper-01] (21)$ ./read-ipheader header1 header2
USAGE: ./read-ipheader <ip-header-file>
```

- Be sure to always check the return value of library functions. For example, if a file cannot be opened, then the program should not read input. Instead it should print an error message as shown below. **Points will be deducted for forgetting to check return values from library functions**.



```
[aphilip@rockhopper-01] (22)$ ./read-ipheader header1
ERROR: Unable to open input file.
[aphilip@rockhopper-01] (23)$ ls
grade_p3.py  read-ipheader*   sample      valid            write-ip-header.c
invalid      read-ipheader.c  skeleton.c  write-ip-header*
```

- If the header is not 20 bytes, print the error message **“ERROR: Header not 20 bytes”** and exit the program using exit(1).
- If the pos argument is not valid, print the error message **“ERROR: Invalid input”** and terminate the program using exit (1) (only 0, 1, and 2 are valid arguments for pos in get_flags(); only 0 and 1 are valid arguments for option in get_ip()).
- Use exit(1) whenever your program terminates abnormally due to some error condition.

4. Notes

- Please use the same function signatures that is given in the program. We will be calling these functions to test your program. **Do not modify them**.
- Please do not modify the contents of the main() below the comment indicating so. This is to ensure that you don't have to worry about the printing formats to match our grading scripts.
- **IMPORTANT:** The CSL lab machines follow **little endian** byte ordering system. So please write your program in such a way that it works correctly on the lab machines.

5. Requirements

1. Your program must follow style guidelines as given in Style Guidelines.
2. Include a comment at the top of each source code file with your **name and section**. You must comment every function with a header comment. See the Commenting Guide, where applicable for C.
3. Your programs should operate exactly as the sample outputs shown above.
4. Use a CSL Linux machine for this assignment!
5. We will compile each of your programs with

   ```
   gcc -Wall -m32 -std=gnu99
   ```

   on a CSL Linux machine. So, your programs must compile there, and **without warnings or errors**. It is your responsibility to ensure that your programs compile on the department Linux machines, and **points will be deducted** for any warnings or errors.
6. Remember to do error handling in all your programs. See the instructions on error handling for more details.

6. Handing in the assignment

Copy the file **read-ipheader.c** into your handin directory:

```
/p/course/cs354-gerald/public/spring2019/handin/<your_CS_login>/p4/
```

where <your_CS_login> is the username of your CS account.

Good luck with bits and bytes! :)