C Programming II
2020 Spring
Homework 05

Instructor: Po-Wen Chi

Due: 2020.06.16 PM 11:59

Policies:

- Zero tolerance for late submission.
- You need to prepare a README file about how to make and run your program. Moreover, you need to provide your name and your student ID in the README file.
- For the writing assignment, I only accept pdf. MS. doc/docx format is not acceptable. Moreover, please use Chinese instead of English.
- Do not forget your Makefile. For your convenience, each assignment needs only one Makefile.
- The executable programs should be hw0501, hw0502 . . ..
- You should pack your homework in one zip file. The file name should be StudentId_hw05.zip.

1 Base64 (25 pts)

In computer science, **Base64** is a group of binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. The term Base64 originates from a specific MIME content transfer encoding. Each Base64 digit represents exactly 6 bits of data. Three 8-bit bytes (i.e., a total of 24 bits) can therefore be represented by four 6-bit Base64 digits.

The Base64 index table can be found in the following link.

`https://en.wikipedia.org/wiki/Base64`

Because Base64 is a six-bit encoding, and because the decoded values are divided into 8-bit octets on a modern computer, every four characters of Base64-encoded text represents three octets of unencoded text or data. This means that when the length of the unencoded input is not a multiple of three, the encoded output must have padding added so that its length is a multiple of four. The padding character is =, which indicates that no further bits are needed to fully encode the input. The padding character is not essential for decoding, since the number of missing bytes can be inferred from the length of the encoded text.

Now I want you to develop the following program.

```
./hw0501 [options]
    -e, --enc        Encode a file to a text file.
    -d, --dec        Decode a text file to a file.
    -o, --output     Output file name.
```

Note that all options' arguments are mandatory.

2 Traffic Monitor (25 pts)

In this class, I have shown you how to use **popen**. This time, I want you to use this to develop a program that can continuously monitor the network traffic throughput. How to calculate the network throughput?

$$\frac{\text{TX/RX Bytes Difference}}{\text{Time Difference}} \text{ Mbps/Kbps/bps.}$$

Note that we use **bps**, which is bit per second, instead of **Bps**, which is Byte per second here. So you need to use popen to get the output of **ifconfig** and calculate the throughput. The program should be with an infinite loop. The user needs to use Ctrl+C to terminate it.

```
./hw0502
1)
enp0s31f6 (192.168.20.176): TX: 53Kbps; RX: 623Kbps
lo (127.0.0.1): TX: 10Kbps; RX: 20Kbps
2)
enp0s31f6 (192.168.20.176): TX: 70Kbps; RX: 43Kbps
lo (127.0.0.1): TX: 19Kbps; RX: 40Kbps
...
```

3 Debug Messages (25 pts)

In this class, I have told you that a professional program developer should has his/her own debug function. I also told you sometimes we do not want to print too many debug messages or the important message will be covered by lots of unimportant messages. Now please develop a header file to support debugging functions.

```
debug_printf( level, fmt, ... )
```

There will be five levels. You need to define them in your header file. If the higher debug level is selected, all messages under this level will be printed. For example, if DEBUG_LEVEL_VVERBOSE is selected, the debug messages labeled with DEBUG_LEVEL_INFO will also be printed. If DEBUG_LEVEL_INFO is selected, the debug messages labeled with DEBUG_LEVEL_VERBOSE will **not** be printed. Of course, developers should not print any debug messages with DEBUG_LEVEL_NONE. This is only a configuration option that can turn off all debug messages.

```
DEBUG_LEVEL_NONE
DEBUG_LEVEL_ERROR
DEBUG_LEVEL_INFO
DEBUG_LEVEL_VERBOSE
DEBUG_LEVEL_VVERBOSE
```

Your debug messages must show the file name, function name and line number. Note that you need to prepare a program called **hw0503** to show how it works.

4 Airport Information (25 pts)

In this class, I have shown you how to use **curl** in C. Now I want you to use this library to get airport information. PTX is a platform which is to effectively satisfy the increasing demand for the information of public transport, including the needs of cross-modes, cross-regions and cross-agencies. It provides lots of application programming interfaces (APIs) for developers to get transportation information. PTX APIs are Web APIs. So you can get information through URLs. This time, I want you to get flight arrival information. Before development, you can use the following command to get the information.

```
curl "https://ptx.transportdata.tw/MOTC/v2/Air/FIDS/Airport/Arrival"
```

So many data, right? I want you to list information in the given format in some different orders.

```
./hw0504 [options]
    -t, --arrival-time         Sorting based on the ScheduleArrivalTime.
    -f, --flight-number        Sorting based on the Flight number.
    -a, --airline-id           Sorting based on the Airline ID.
2020-05-14T23:35 PEK-->TPE CI 518
2020-05-14T23:40 TNA-->TPE SC 103
...
```

You can assume that TA's computer has libcurl library.

5 Bonus: dirent.h (5 pts)

dirent.h is a C POSIX library. That is, it is not a standard header file. Please study it and use its functions to list all files and directories in the given directory.

```
./hw0505 /etc
...
```