C Programming II
2022 Spring
Homework 01

Instructor: Po-Wen Chi

Due: 2022.05.24 PM 11:59

Policies:

- **Zero tolerance** for late submission.
- **Plagiarism is not allowed.** Both source and copycat will be **zero**.
- 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.
  - Your Name and Your ID.
  - The functional description for each code.
  - Anything special.
- Please pack all your submissions in one zip file. **RAR is not allowed!!**
- For convenience, your executable programs must be named following the rule hwXXYY, where the red part is the homework number and the blue part is the problem number. For example, hw0102 is the executable program for homework #1 problem 2.
- I only accept **PDF**. MS Word is not allowed.
- Do not forget your Makefile. For convenience, each assignment needs only one Makefile.

1 Memory Monitor (30+5 pts)

The proc filesystem is a **pseudo-filesystem** which provides an interface to kernel data structures. It is commonly mounted at /proc. Most of the files in the proc filesystem are read-only, but some files are writable, allowing kernel variables to be changed.

This time, I want you to implement a memory monitor, which you can monitor the memory usage status periodically. The memory usage information can be derived from **/proc/meminfo**. You can simply treat this as a text file, though actually it is not.

```
$ ./hw0501 --help
Usage:
    free [options]
Options:
 -b, --bytes          show output in bytes
     --kilo           show output in kilobytes (default)
     --mega           show output in megabytes
 -s N, --seconds N    repeat printing every N seconds
 -c N, --count N      repeat printing N times, then exit
     --help     display this help and exit
$ ./hw0501
Available: 10047712 KB (61.74%)
```

When printing the memory information, do not forget the unit and round off to the 2nd decimal place.

Additional Bonus (5 pts): Why *MemFree* is not equal to *MemAvailable*? What is different between these two values?

2 ID3 Editor(30 pts)

ID3 is a metadata container most often used in conjunction with the MP3 audio file format. It allows information such as the title, artist, album, track number, and other information about the file to be stored in the file itself. You can see the wiki for more details.

`https://en.wikipedia.org/wiki/ID3`

This time, I want you to develop an ID3v1[1] editor for MP3 files.

For simplicity, I promise that the input file is a valid MP3 file. However, the given file may has no ID3v1 tags.

```
$ ./hw0502 -h
usage: hw0502 -[tTaAycg] "value" file1 [file2...]
       hw0502 -d file1 [file2...]
       hw0502 -l file1 [file2...]
       hw0502 -h
 -t   Modifies a Title tag
 -T   Modifies a Track tag
 -a   Modifies an Artist tag
 -A   Modifies an Album tag
 -y   Modifies a Year tag
 -c   Modifies a Comment tag
 -g   Modifies a Genre tag
 -l   Lists an ID3 tag
 -d   Deletes an ID3 tag
 -h   Displays this help info
$ ./hw0502 -t "Four Seasons" four_seasons.mp3
$ ./hw0502 -A "Vivaldi Collection" four_seasons.mp3
$ ./hw0502 -a "Neokent" four_seasons.mp3
$ ./hw0502 -c "Good" four_seasons.mp3
$ ./hw0502 -y "2022" four_seasons.mp3
$ ./hw0502 -g "Classical" four_seasons.mp3
```

[1]ID3v2 is more complicated. I am really a good guy.

```
$ ./hw0502 -l four_seasons.mp3
four_seasons.mp3:
  Title  : Four Seasons
  Artist: Neokent
  Album  : Vivaldi Collection
  Year: 2022
  Genre: Classical (32)
  Comment: Good
  Track: 0
```

3 Base64 (20 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.
$ ./hw0501 -e maldives.bmp -o maldives.txt
$ ./hw0501 -d maldives.txt -o maldives.bmp
```

Note that all options' arguments are mandatory. By the way, do not use **popen** or **system** to call **base64** command directly.

4 Taiwanese Learner (20 pts)

For some typing reason, please check the problem from the course official site.

You can assume that libcul has been successfully installed on the TA's computer.

If you have any questions, please contact 盧昭華。

5 Bonus: What's Wrong (5 pts)

Alice writes the following code.

```c
#include <stdio.h>

void init( int n )
{
    char str[10000000];
    for( int i = 0 ; i < n ; i++ )
    {
        str[i] = i;
    }
    return;
}

int main()
{
    init( 10 );
    return 0;
}
```

When she builds this code and runs the process, she gets

```
$ ./a.out
Segmentation Fault (Core Dump)
```

Alice is curious what happens. There is no compiler error! So she asks Bob and Bob says that you can modify your code like this:

```c
#include <stdio.h>

void init( int n )
{
    static char str[10000000]; // <---------
    for( int i = 0 ; i < n ; i++ )
    {
        str[i] = i;
    }
    return;
}

int main()
{
    init( 10 );
    return 0;
}
```

It works! However, Alice has no idea why it works. So she asks Catherine and Catherine use another way.

```c
#include <stdio.h>

char str[10000000]; // <---------

void init( int n )
{
    for( int i = 0 ; i < n ; i++ )
    {
        str[i] = i;
    }
    return;
}

int main()
{
    init( 10 );
    return 0;
}
```

It works! Would you please explain what happens to Alice?