C Programming II
2021 Spring
Final

Instructor: Po-Wen Chi

Date: 2022.01.07 PM 14:00-18:00

Policies:

- Online test.
- Do not forget to include your Makefile. TA will only use the command make to build your program. If make fails, you will get zero points and no room for bargaining. So if you do not know how to solve a problem, please, do not include it in your Makefile.
- I do not care your source code file names, but the executive binary names should be **fin01, fin02, fin03, fin04**.
- You can ask TA any questions on the moodle forum if you do not understand the problems. But remember, TA QA time is only 14:00-16:00.

1 Pentagon Area (20 pts)

Given five points in a 2-dimensional plane, as shown in figure 1. Please calculate the area of this pentagon formed by these five points.

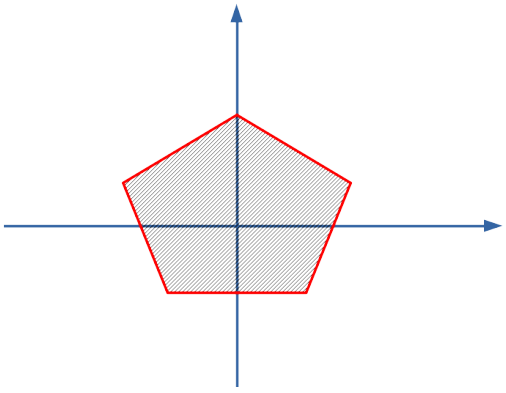

Figure 1: The area of this pentagon.

For your convenience, all inputs are 32-bits integers.

```
$ ./fin01
Please enter P1: 2,0
Please enter P2: 0,1
Please enter P3: -2,0
Please enter P4: -1,-1
Please enter P5: 1,-1
5
```

If this is not a pentagon, your need to print "**This is not a pentagon**". For your simplicity, I guarantee that the five input points are in order, which means the five edges are $\overline{P_1P_2}, \overline{P_2P_3}, \overline{P_3P_4}, \overline{P_4P_5}, \overline{P_5P_1}$. BTW, you should use **double** and the precision is not a concern in this problem.

2 Go (40 pts)

Have you ever played Go? Go is an abstract strategy board game for two players in which the aim is to surround more territory than the opponent. The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played to the present day. Figure 2 is a picture of a board with stones.

Figure 2: Go is played on a grid of black lines (usually 19×19). Source: wikipedia.

Now, I will teach you the basic rule: liberty. We use Fig. 3 as an example. In Fig. 3 A, the black stone has four liberties. Once a white stone is put beside the black stone, the number of the black stone liberties will be reduced to three, as shown in Fig. 3 B. Fig. 3 C and D are similar. If a stone has no liberty, it will be removed. Note that only the intersections on the board are counted. If a stone is put on the corner, it will has only two liberties, as shown in Fig. 4.

Liberties are shared among all stones of a chain (group) and can be counted. For example, in Fig. 5, there are one black chain and two white chains. The black group has 5 liberties, while the two white chains have 4 liberties each.

Calm down! I will not make you develop a Go game in the final exam. So I skip other rules here. The above rules are enough for this problem. In this problem, I want you to

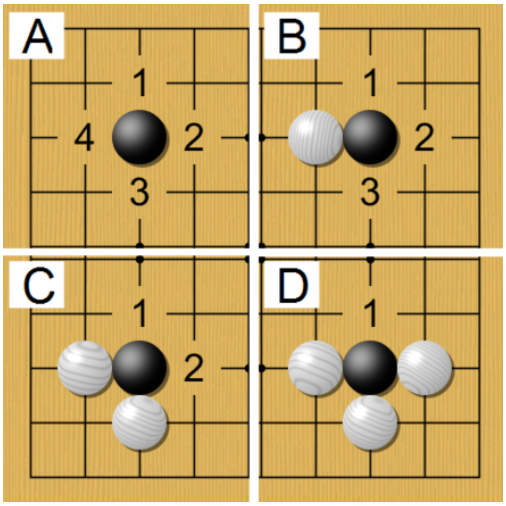

Figure 3: Introduction to liberties.

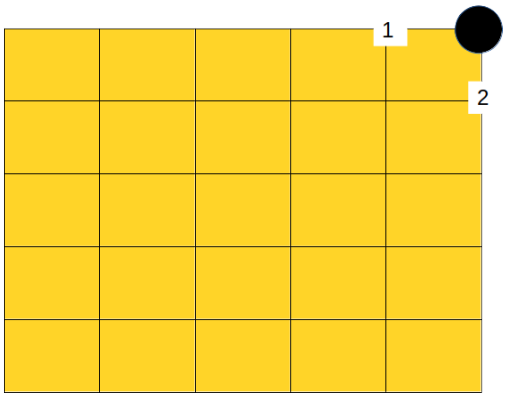

Figure 4: The corner stone.

implement a function to **find the black chain which has most liberties and return its liberty number.** You need to implement the following function:

```
int32_t max_black_chain( int32_t board[][19] );
```

The input is a 2-dimensional integer array. 0 means that there is no stone, 1 means black and 2 means white. The return value is the liberty number of the black chain that has most most liberties.

For your simplicity, I promise that

1. The input is always 19×19 since the Go board is 19×19.
2. You do not to care about the input is invalid:
   - All integers are in $\{0, 1, 2\}$.
   - There is no stone chain that has no liberty.

Again, you need to implement this function in **go.c** with a header file **go.h**. The TA will prepare **fin02.c** for you. Do not forget to build **fin02.c** in your makefile.

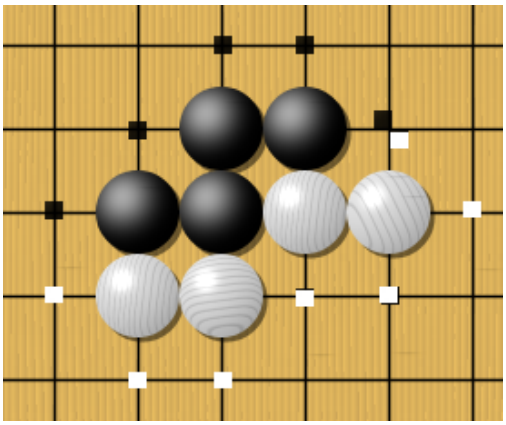

Figure 5: Liberties for chains.

For your own good, the TA will prepare the following test cases:

1. 5 pts: only one black chain on the board without any white stones.
2. 5 pts: only one black chain on the board.
3. 5 pts: multiple black chains on the board without any white stones.
4. 25 pts: a real game board.

BTW, if you are interested in Go, please contact Prof. Shun-Shii Lin.

3 Fraction Arithmetic (20 pts)

Please implement the following functions in **frac.c** with a header file **frac.h**. The TA will prepare **fin03.c** for you. Do not forget to build **fin03.c** in your makefile.

```
int32_t frac_add( int32_t *x, int32_t *y,
                  int32_t a, int32_t b, int32_t c, int32_t d );
int32_t frac_del( int32_t *x, int32_t *y,
                  int32_t a, int32_t b, int32_t c, int32_t d );
int32_t frac_mul( int32_t *x, int32_t *y,
                  int32_t a, int32_t b, int32_t c, int32_t d );
int32_t frac_div( int32_t *x, int32_t *y,
                  int32_t a, int32_t b, int32_t c, int32_t d );
```

The above functions are for the following arithmetic operations respectively.

$$\frac{x}{y} = \frac{a}{b} + \frac{c}{d}, \frac{x}{y} = \frac{a}{b} - \frac{c}{d},$$
$$\frac{x}{y} = \frac{a}{b} \times \frac{c}{d}, \frac{x}{y} = \frac{a}{b} \div \frac{c}{d},$$

Again, you need to implement this function in **mem.c** with a header file **mem.h**. The TA will prepare **fin04.c** for you. Do not forget to build **fin04.c** in your makefile.

Note that if the input is invalid, return -1; otherwise, return 0.

4 Your Own Memory Allocation Functions (20 pts)

In this class, I have taught you how to use memory related functions. When you use **malloc**, the computer will allocate a block of memory for you. But sometimes we need a customized memory allocation function like Fig. 6. The reason is that sometime you may want to attach data in front of an existing memory block.

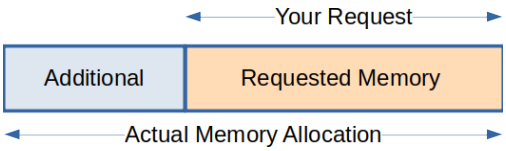

Figure 6: Customized memory allocation.

In this problem, I want you to implement the following functions:

```
/*
Input:
  size: requested memory size.
  front_size: additional memory size which is in front of the requested memory
    block.
  *available_front_size: used to record the available space of the front
   additional size. The user should setup a valid available_front_size first.
    If the available_front_size is NULL, you should treat this as zero, which
    means there is no additional memory space.
Output:
  The allocate memory address which does not include the front additional
   memory.
*/
void * my_malloc( size_t size, size_t front_size, size_t *available_front_size
    );
/*
Input:
  **ptr: the memory address that will be changed.
  size: use the front memory, which implies moving the address forward.
  *available_front_size: used to record the available space of the front
   additional size. If the size is larger than *available_front_size, the
   function will only use all available space. If available_front_size is
   NULL, then the address will not be changed.
Output:
  None
*/
void my_push( void **ptr, size_t size, size_t *available_front_size );
/*
Input:
  *ptr: the memory address that will be freed.
  available_front_size: the available memory space in front of the memory
   address. You can assume that available_front_size is valid.
Output:
  None
*/
void my_free( void *ptr, size_t available_front_size );
```

This problem comes from the real case. The linux network related memory allocation function uses the similar way for adding headers.

5 Bonus: Your Comments (5 pts)

Again, any comments are welcomed. However, you will get nothing if you leave this question blank.