C Programming I
2023 Fall
Final

Instructor: Po-Wen Chi

Date: 2023.12.23 10:00 - 16:00

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 Fraction Arithmetic (20 pts)

Please implement the following functions.

```
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.

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

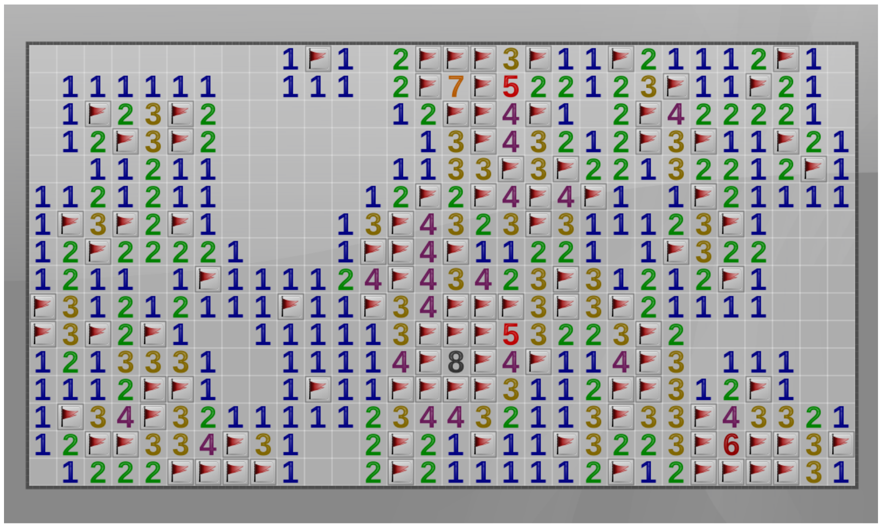

Figure 1: KMines, a free and open-source variant of Minesweeper.

Note that if the input is invalid, return -1; otherwise, return 0. The output must be in the irreducible form.

You should also prepare a header file called frac.h. Our TAs will prepare fin01.c which includes frac.h and uses this function. Do not forget to make fin01.c to fin in your Makefile.

2 Minesweeper(15 pts)

Minesweeper is a logic puzzle video game genre generally played on personal computers and I think that I do not need to teach you how to play this game right? Figure 1 is an example of the game screen.

A player selects a cell to open it. If a player opens a mined cell, the game ends. Otherwise, the opened cell displays either a number, indicating the number of mines diagonally and/or adjacent to it, or a blank tile (or "0"), and all adjacent non-mined cells will automatically be opened. For your simplicity, the board size is fixed to 16×30.

You need to implement a function to print the board result after hitting a given position. The user will input the board, which is a 16×30 two-dimensional array. Each element should be -1 or -2 where -1 implies there is no mine at that position and -2 implies there is a mine at that position. The user also input two int32_t, *row* and *col*, to indicate the hit position, which is **board**[**row**][**col**]. You need to modify the board array to the board result after hitting a position.

```
// board: game board. If the cell is a mine, the value is -2; otherwise, -1.
// row, col: implies hitting board[row][col].
// Note that the row and col should start from the top left corner and start
    from 0.
// Return -1 if the input is invalid and return 1 is hit the mine; otherwise
    return 0.

int32_t hit( int32_t board[16][30], int32_t row, int32_t col );
```

Let's give you an example. However, for my own good, I just give you a small size board example.

```
// Before
int32_t board[4][5] = { {-1,-1,-1,-2,-1},
                        {-1,-2,-1,-1,-1},
                        {-1,-2,-1,-1,-1},
                        {-1,-1,-2,-1,-1} };

// Case 1: hitting (1,2), set the value to the number of surrounding mines.
int32_t board[4][5] = { {-1,-1,-1,-2,-1},
                        {-1,-2, 3,-1,-1},
                        {-1,-2,-1,-1,-1},
                        {-1,-1,-2,-1,-1} };

// Case 2: hitting (1,1), mine is blown up and you do not need to modify the
    array value.
// Just return 1.
int32_t board[4][5] = { {-1,-1,-1,-2,-1},
                        {-1,-2,-1,-1,-1},
                        {-1,-2,-1,-1,-1},
                        {-1,-1,-2,-1,-1} };

// Case 3: hitting (2,4), since there is no surrounding mine, set the value to
     0.
// You should also open surrounding blocks.
int32_t board[4][5] = { {-1,-1,-1,-2,-1},
                        {-1,-2,-1, 1, 1},
                        {-1,-2,-1, 1, 0},
                        {-1,-1,-2, 1, 0} };
```

You should also prepare a header file called mine.h. Our TAs will prepare fin02.c which includes mine.h and uses this function. Do not forget to make fin02.c to fin in your Makefile.

3 Firewall (15 pts)

In computing, a firewall is a network security system that monitors and controls network traffic based on predetermined security rules. This time, I want you to develop a firewall function. Do not worry, I just want you to develop the simulation version.

Network traffic can be treated as a sequence of packets. In this problem, we simply use a byte array as input packets. Each packet has a header, which is composed of two 32 bits integers, which are source identity and destination identity, one 16 bits unsigned integers, which is the packet data size[1], and a sequence of data bytes. Fig. 2 is a schematic diagram. For your convenience, they are encoded in little-endian[2].

Now, you need to implement a firewall function to process incoming packets. Note that your firewall function needs to make users input their own rules, including forwarding packets, dropping packets or modifying packets. For your simplicity, your firewall should support at most **100** pre-defiend rules. Do not worry, it is TAs' duty to implement firewall

[1]The header size is not included.

[2]Actually, network order is big-endian.

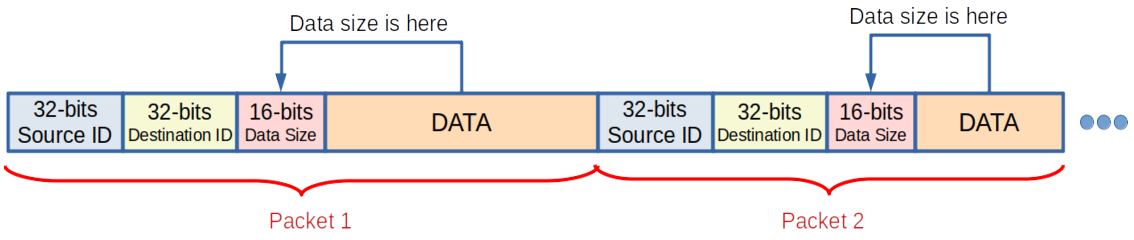

Figure 2: Incoming packets.

rules. Your job is simply recording rules, applying rules on every packets of the input traffic and generating output traffic, which contains many packets. Note that the rule should be applied on packets in order. That is, a packet should be processed by rule 1, if not being dropped, then being processed by rule 2.

```
// First, you should have a rule array, where its size is 100 elements.
// Then set rule[idx] to the input function pointer.
// Rule Function Pointer:
//     int32_t (*rule)( const uint8_t *p_input_packet, const int32_t
    input_size,
//                          uint8_t **pp_output_packet, int32_t *p_output_size )
//     Input: p_input_packet -> input packet (one packet)
//            input_size -> input_packet size
//     Output: pp_output_packet -> output_packet (one packet)
//             p_output_size -> output_packet size
//             return -> 1 if the input packet is dropped, -1 if the input is
    invalid and skip this rule; otherwise, return 0
// If the inputs are invalid, return -1; otherwise, return 0.
int32_t set_rule( int32_t idx, int32_t (*rule)( const uint8_t *p_input_packet,
     const int32_t input_size, uint8_t **pp_output_packet, int32_t *
    p_output_size ) );

// Set rule[idx] to NULL.
// If the inputs are invalid, return -1; otherwise, return 0.
int32_t unset_rule( int32_t idx );

// For every input packet, apply all rules on the packet and store every ouput
     packet on the pp_output_packets.
// If the inputs are invalid, return -1; otherwise, return 0.
int32_t filter( const uint8_t *p_input_packets, const int32_t input_size,
    uint8_t **pp_output_packets, int32_t *p_output_size );
```

I will give you an example. Suppose a user input a byte stream as follows.

```
uint8_t array[] = { 0x01, 0x00, 0x00, 0x00, // Packet 1 -> Source ID: 1
                    0x02, 0x00, 0x00, 0x00, //             Destination ID: 2
                    0x03, 0x00,             //             Size
                    0x01, 0x02, 0x03,       //             Data
                    0x01, 0x00, 0x00, 0x00, // Packet 2 -> Source ID: 1
                    0x03, 0x00, 0x00, 0x00, //             Destination ID: 3
                    0x02, 0x00,             //             Size
                    0xEE, 0xFF,             //             Data
                    0x02, 0x00, 0x00, 0x00, // Packet 3 -> Source ID: 2
                    0x03, 0x00, 0x00, 0x00, //             Destination ID: 3
```

```
                     0x04, 0x00,             //               Size
                     0x00, 0x00, 0x01, 0x02  //               Data
                     };
```

The pre-defined example rules which are implemented by TAs are as follows:

1. If source ID is 1, set destination ID to 5.
2. If source ID is 2, set source ID to 7.
3. If size is 2, duplicate data.
4. If source ID is 7 and destination ID is 3, Drop the packet.

After applying these rules on these packets, the output packets will be

```
uint8_t array[] = { 0x01, 0x00, 0x00, 0x00, // Packet 1 -> Source ID: 1
                     0x05, 0x00, 0x00, 0x00, //               Destination ID: 5
                     0x03, 0x00,             //               Size
                     0x01, 0x02, 0x03,       //               Data
                     0x01, 0x00, 0x00, 0x00, // Packet 2 -> Source ID: 1
                     0x05, 0x00, 0x00, 0x00, //               Destination ID: 5
                     0x04, 0x00,             //               Size
                     0xEE, 0xFF, 0xEE, 0xFF  //               Data
                     };
```

The packet 3 will be dropped so it is not included in the output traffic.

You should also prepare a header file called firewall.h. Our TAs will prepare fin03.c which includes firewall.h and uses this function. Do not forget to make fin03.c to fin in your Makefile.

4 The Tank War Game (50 pts)

First Reminder: In this problem, do as much as you can! The maximum points in this problem is 60 pts. However, you can get at most 50 pts in this problem. Otherwise, TA will ignore some details when judging your program, because the main goal is ”developing a playable game!”.

Second Reminder: Please fill this form: `https://forms.gle/G5WHNu1gh9KGNRrNA`,before the midnight(i.e. 12/23 23:59). TA will **ONLY** check the feature you tick in the form.

You should decide whether to do the Advanced Screen task or not at the beginning since it may affect some major changes and core decision making in the development process. Enjoy the GAME!

4.1 Initial Map (3 pts)

First, you should let player decide the size of the map.

```
Please input the height of the map: 10
Please input the width of the map: 20
```

In the example above, the map is 10×20(include edge). First, you need to setup the game environment like Figure 3.

4.1.1 Players (2 pts)

There's only **ONE** Human and **THREE** Computer players. You should mark the Human Player location with 'P', and the Computer location with 'C'. And the muzzle(炮口) with 'I', and you should initialize the muzzle on the top of 'P' and 'C'.
You should initialize the location randomly.

4.1.2 Obstacle (1 pts)

You should mark about `(height×width)/40` obstacles with 'R'.
You should initialize the location randomly.

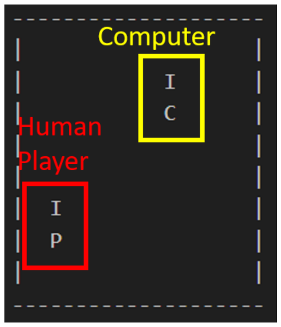

Figure 3: initial map.

4.2 Human Player (25 pts)

4.2.1 Basic requirement (0 pts)

First, you should print the message to player, let player enter the action.

```
(show initial map, do not output this line)
Please input the action:
```

After finish the action, you should print the message and read the input again:

```
(show initial map, do not output this line)
Please input the action: W
(show the map after doing the action 'W', do not output this line)
Please input the action:
...
```

After the player enter the action, you should print the map after doing the action.

4.2.2 Action: Move (2 pts)

The key player will enter: W, A, S, D, R
The player will enter 'W', 'A', 'S' or 'D', which means move up, left, down, right. Move just one block. If the player enter 'R', don't move.

4.2.3 Hit the wall/obstacle (2 pts)

If the tank hit the wall(1 pt) or obstacle(1 pt) when moving, stop at the previous block(i.e. Don't move).

4.2.4 Muzzle direction indicator (3 pts)

The muzzle direction indicator should turn around on each frame(i.e. Each action) with the following order:

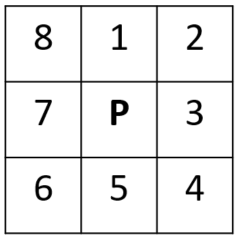

Figure 4: Turn around order.

4.2.5 Action: Shooting (3 pts)

The key player will enter: Q
The player will enter 'Q', which means "Shooting the cannonball".
The cannonball will move 1 block on each frame(i.e. Each action), and the direction should as same as the muzzle, see the Figure 5.

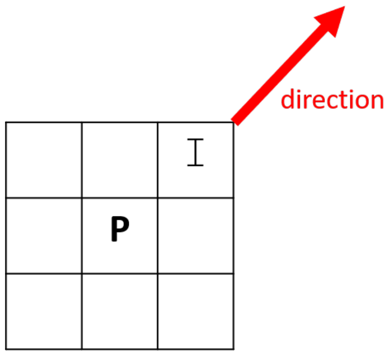

Figure 5: Shoot direction.

And the cannonball should mark with 'o'(small o).

4.2.6 Cannonball: Fly(3 pts)

Your cannonball should move 1 block on each frame(i.e. Each action).

4.2.7 Cannonball: Hit the wall - Straight bounce (3 pts)

When a cannonball strikes the wall at a right angle(直角), you should do straight bounce like the figure 6:

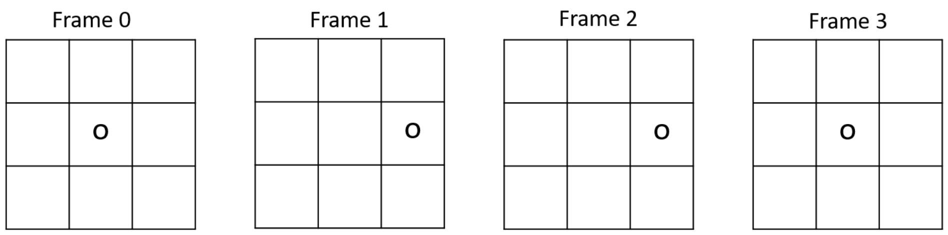

Figure 6: Straight bounce.

4.2.8 Cannonball: Hit the wall - Angled bounce (4 pts)

When a cannonball strikes the wall but NOT at a right angle, you should do angled bounce like the figure 7:

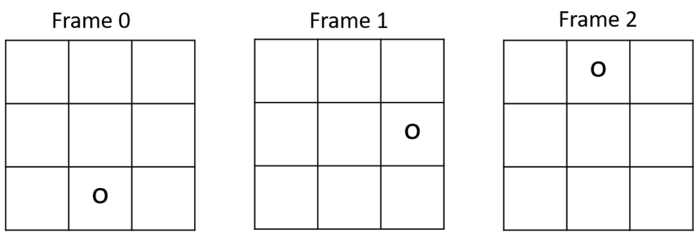

Figure 7: Angled bounce.

4.2.9 Cannonball: Hit the obstacle (2 pts)

When a cannonball strikes the obstacle, you should remove the cannonball and the obstacle it hits.

4.2.10 Cannonball: Time to live (2 pts)

After one cannonball move 10 frames, it will disappear.

4.2.11 Cannonball: Limit of total number (1 pts)

One player can only have ONE cannonball on the board at a time.

Notice: If there are 4 players, it means that there are up to 4 cannonballs on the board!

4.3 Computer Player (14 pts)

4.3.1 Basic requirement (8 pts)

Do the same thing like the human player, but ignore the ”Action: Move” and ”Action: Shooting”.
You can gain 1 pts for each function(except ”Action: Move” and ”Action: Shooting”).

4.3.2 Move (3 pts)

Move randomly or other smart way.

4.3.3 Shooting (3 pts)

Shoot immediately when the cannonball is available.

4.4 Destroy (2 pts)

When the cannonball hit a human or computer player, you should remove that player from the board.

4.5 Advanced Screen (16 pts)

You are requested to implement an advanced viewport tracking system. This system should dynamically adjust the player’s view based on the character’s position on the map, ensuring that the character is always in focus during gameplay.

4.5.1 Character-Focused Viewport Tracking (10 pts)

Dynamic Viewport Adjustment: The viewport should center around the character as they move across the map. This creates a more immersive and responsive gaming experience with vast maps. The viewport’s position must update in real time with the character’s movement, maintaining the character near the center of the screen.

4.5.2 Edge Protection Mechanism (6 pts)

Things become more complex when the player approach to the edge or corner of the map, so we need to implementation of an edge protection mechanism. This mechanism comes into play when the character approaches the edges of the map. Here’s how it should work:

- When the character is near the map’s edge, the viewport should adjust so that it does not display areas outside the map boundaries, even if this means the character is not exactly at the center of the screen.
- The adjustment should be smooth and seamless, avoiding any sudden jumps or shifts in the viewport that can disorient the player.
- The viewport should dynamically realign itself to center the character when they move away from the edge, ensuring a consistent and focused gameplay experience.

Implementation Guidelines

- Calculate the viewport's position based on the character's coordinates, adjusting as necessary when approaching map edges.
- Test the system thoroughly to ensure that it works smoothly across the entire map, including corners and edges.

5 Bonus: Your Comments (5 pts)

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

Reminder

Do not forget to fill your attendance form for bonus!! The form will be closed at 2023/12/26.

Besides, there is another questionnaire about your attendance. Please fill the following form **after the exam.** Do not waste your valuable exam time on it.

`https://forms.gle/KfXkHeWXvUA1k1GX9`

Advertisement: Software Engineering Final Presentation

Welcome to join the final presentation of the class, Software Engineering.

- Date: 2023.12.27 AM 9:00
- Location: Applied Science Building