National Taiwan Normal University
CSIE Computer Programming I

Instructor: Po-Wen Chi
Due Date: 2022.12.27 PM 11:59

Assignment 5

Policies:

- Zero tolerance for late submission.
- 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.
- Please provide a README.

5.1 Rotation (20 pts)

Given a 2-d coordinate and θ, please rotate this point with θ **counter-clockwise** where the circle center is O, as shown in figure 5.1.

Note that you need to implement the following function. The coordinate after rotation should be returned in the input arguments. θ is in degree, like 185°. For example, given $x = 1, y = 0, \theta = 90$, after calling this function, $x = 0, y = 1$.

```
// theta: in degree, not radian.
void rotate( double *x, double *y, double theta );
```

You have to prepare **rotate.h** and **rotate.c**. Our TA will provide **hw0501.c** which will include **rotate.h**. **Do Not Forget to Build hw0501.c in your Makefile**.

Again, precision is not the concern in this problem.

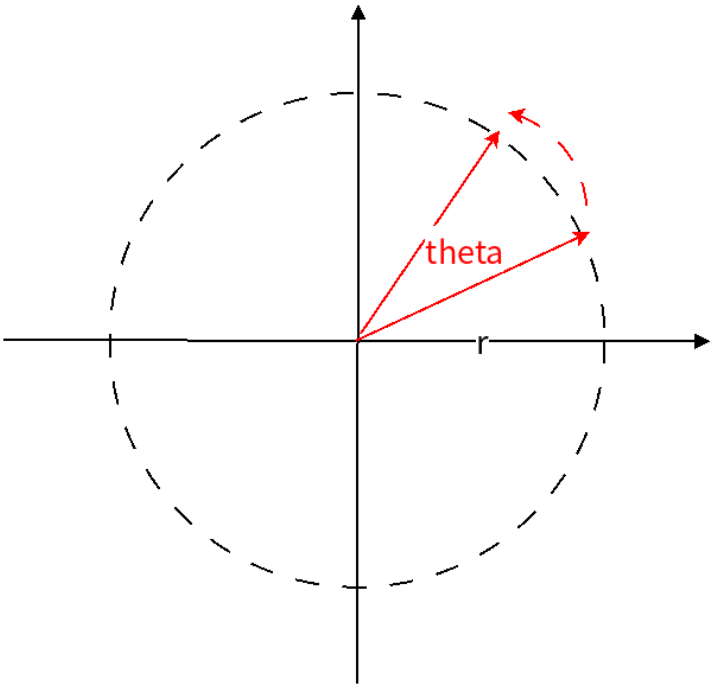

FIGURE 5.1: Rotation.

5.2 memcmp (20 pts)

Please read the manual of **memcmp**. This is a C standard function. This time, I want you to develop this function, **mymemcmp**, yourself. The behavior should be the same with the manual.

You have to prepare **mymemcmp.h** and **mymemcmp.c**. Our TA will provide **hw0502.c** which will include **mymemcmp.h**. **Do Not Forget to Build hw0502.c in your Makefile**.

5.3 Finite State Machine (20 pts)

A finite state machine (FSM), is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition. Please implement a finite state machine according to figure 5.2. The user can input integers until the final state. For your simplicity, I promise all inputs are integers. However, there is a limitation in this problem: You cannot use **if-else** and **switch** in the **state transition** in this problem!!

```
$ ./hw0503
```

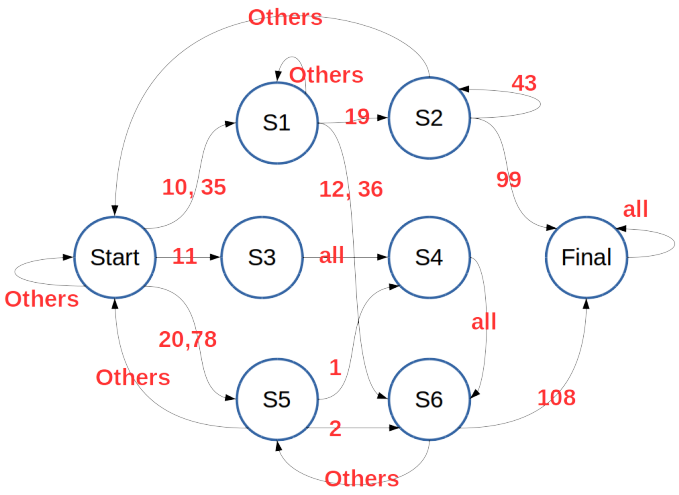

FIGURE 5.2: Deterministic Finite Automata.

```
Start
Please enter an integer: 35
S1
Please enter an integer: 19
S2
Please enter an integer: 43
S2
Please enter an integer: 99
Final
```

Hint: array + function pointer.

5.4 Big Integer Division (20 pts)

Let's implement big integer division. We use an **uint8_t** array to present a big number. For example, 12345 can be presented as figure 5.3. Note that each element in the array will only be 0-9. For your convenience, I promise:

- The first element will not be zero.
- All elements are in range 0-9.
- The given integer is definitely positive.

Now, please implement the following function. The quotient and the remainder should be placed on **ppQuotient** and **ppRemainder** respectively. The encoding rule is the same with the above. Here I use double pointer because you need to allocate memory space in this function. Note that the allocated memory space should be just enough.

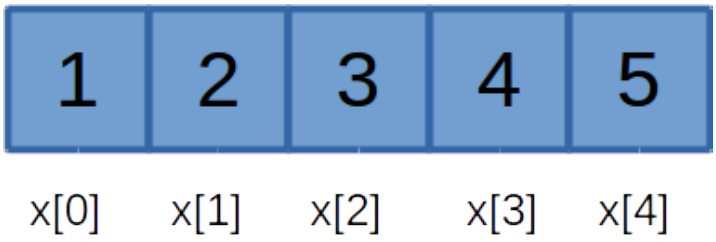

FIGURE 5.3: Big Number.

```
// X/Y
void division( uint8_t **ppQuotient, uint8_t *pQuotientSize,
               uint8_t **ppRemainder, uint8_t *pRemainderSize,
               uint8_t *pX, uint8_t xSize,
               uint8_t *pY, uint8_t ySize );
```

You have to prepare **division.h** and **division.c**. Our TA will provide **hw0504.c** which will include **division.h**. **Do Not Forget to Build hw0504.c in your Makefile**.

5.5 CSIE Simply not a Instruction Encoder

CSIE is a language that has only 8 commands. Commands are represented as Table 5.1 shown. The commands are stored in an array of **uint8_t**. The commands are executed in sequence, that is, there will be a command pointer points to the current running command, and after the command execution, the command pointer will increase one (move to the next command).

The memory model of this language is an array of 8-bit unsigned integer, and an internal pointer points to one of the cells. The pointer's initial position is at the left most cell.

The array of this language is originally be infinite length, but TA's computer has limited memory, thus the memory limit will be pass to your function and will not exceed 10,000,000.

Now, your task is to write a function that executes CSIE, that is, given an array of CSIE, the array size, and the memory limit, the function should perform the CSIE program.

The prototype of this function is:

```
void csie_interpreter(const uint8_t code[], size_t size, size_t
    mem_limit);

```

Tell you some fun facts, one is that CSIE is very similar to a Turing Machine, that is, an abstract machine that can model every program or action in computer.

Command	Meaning
0	Increase(right move) the data pointer (to point to the next cell to the left).
1	Decrease(left move) the data pointer (to point to the next cell to the right).
2	Increase the number by one at the data pointer.
3	Decrease the number by one at the data pointer.
4	Output the `uint8_t` number at the data pointer.
5	Read a `uint8_t` number from stdin (keyboard input).
6	If the number at the data pointer is zero, then instead of moving the command pointer forward to the next command, and move the command pointer to the matching `7` command.
7	If the number at the data pointer is nonzero, then instead of moving the command pointer forward to the next command, and move the command pointer to the matching `6` command.

TABLE 5.1: CSIE command list and its meaning

Another one is that this language seems to be familiar with Brainf*ck language, the wikipedia page link is provided for your reference.

5.5.1 Exception Handling

- You cannot left move the pointer over the original position or move over the the memory limit, in this case, you should terminate the whole program and report this runtime error. (e.g. `data_array[-1]` or `data_array[mem_limit]` is not allowed, the `data_array` refers to the `uint8_t` array of CSIE language.)
- When encounter unsigned integer underflow or overflow, follow the behaviour of the C programming language.
- Do not worry about invalid command. Command input are guarantee to be valid.

5.5.2 Requirement

Compile your code and link it dynamically into `libhw0505.so` that contains the designated function implementation. You do not need to prepare other file except `libhw0505.so`. TA will dynamically link your shared object (dynamically linked library) and call the function in it. You do not need to

prepare main program for TA, that is, the `libhw0505.so` should not include `main` function.

5.5.3 Example of CSIE code

5.5.3.1 Loop

Code of CSIE:

```
uint8_t code[] = {2, 2, 2, 2, 2, 2, 2, 2, 6, 3, 7, 4};
csie_interpreter(code, sizeof(code), 100);

```

Output:

```
0

```

This program will add the first cell 8 times, then use a the command `6` and `7` to create a loop, inside the loop, it will continuously decrease the first cell until zero and exit the loop. The next instruction of the loop is output the value of current cell, wihch is zero.

5.5.3.2 Input and output

Code of CSIE:

```
uint8_t code[] = {5, 4};
csie_interpreter(code, sizeof(code), 100);

```

This code will accept a character input and output that character, then terminate. The sample inputs are from stdin, e.g. keyboard input.
Sample Input 1:

```
10

```

Sample Output 1:

```
10

```

Sample Input 2:

```
20

```

Sample Output 2:

```
20

```

5.5.4 Appendix - How to generate and link a shared object

To generate a shared object, you need to use the following command to compile your source code file into shared object(.so file).

```
gcc -shared -fPIC -o <output file name> <source file>

```

To dynamically link the shared object, use the following command to link your main program to your shared object.

```
gcc -o <ouput file name> -L. -l<shared object name> <main
    program source file>

```

Be aware that the shared object name do not should not include `lib` prefix and `.so` suffix. For example, if I have a shared object called `libhw0505.so` and the main program source file is called `main.c`, then the command will be:

```
gcc -o main -L. -lhw0505 main.c

```

5.6 Bonus: Array Size (5 pts)

Please compare the following two codes. Both can be compiled without errors. However, if you run the first program, you will get **segmentation fault** while the second one is fine. Could you please explain this situation?

```
int32_t array[999999999] = {0};
```

```
int32_t *ptr = malloc( sizeof( int32_t ) * 999999999 );
for( int32_t i = 0 ; i < 999999999 ; i++ )
{
    ptr[i] = 0;
}
```