HW1

solving a single player EWN

EWN (Einstein würfelt nicht!)

All assignments in this course will be related to EWN.

Rule Description

EWN is a **two player** and **stochastic** (雙人隨機) game.

But in hw1, we’re going to solve a EWN variant that is

single player and **deterministic** (單人無隨機).

Rules of modified EWN

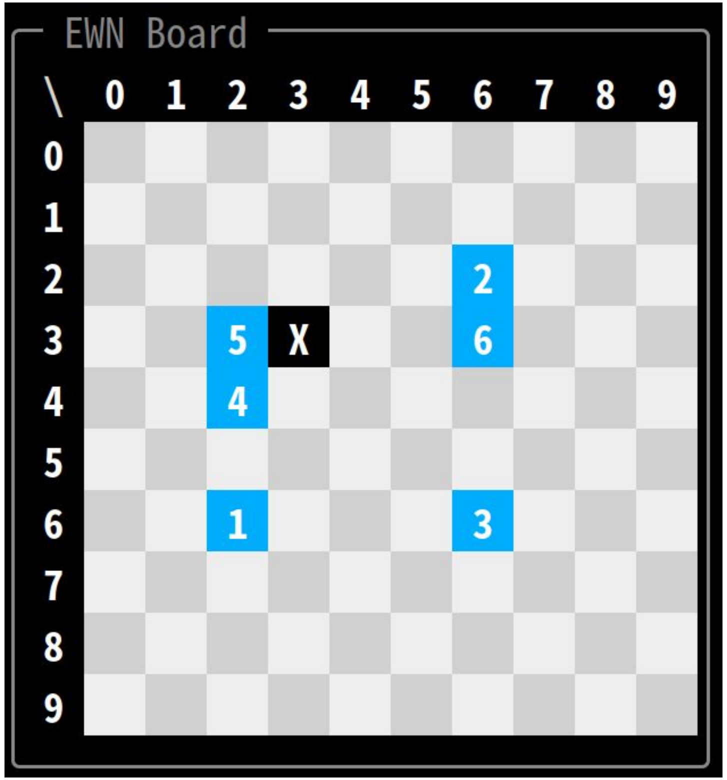
1. board:

   size: 10x10

   position (3,3) is removed, no piece can enter.

   pieces: 1,2,3,4,5,6

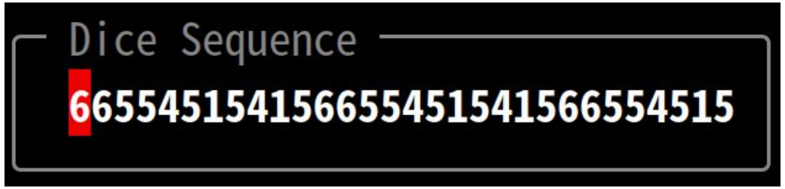
   For all testcases, all the above six chess pieces are guaranteed to exist.



EWN Board

\	0	1	2	3	4	5	6	7	8	9
0										
1										
2							2			
3			5	X			6			
4			4							
5										
6			1				3			
7										
8										
9										

Rules of modified EWN



```
Dice Sequence
665545154156655451541566554515
```

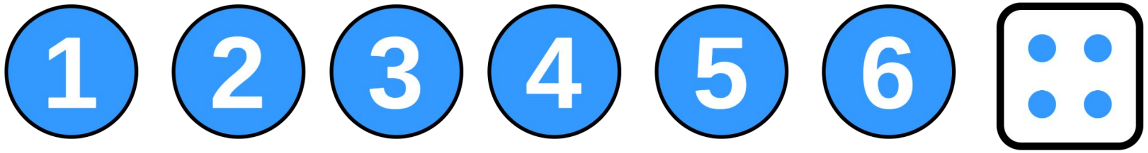
2. moving procedure

a. determine the dice number for the round :

DiceNumber = DiceSequence[round_number]

b. determine the moving piece :

if the dice number matches an existing piece, you can only move that matched piece.



In this example, you can only move piece 4.

Rules of modified EWN

2. moving procedure

   b. determine the moving piece :

if the dice number doesn’t match any existing piece, you can choose to move:

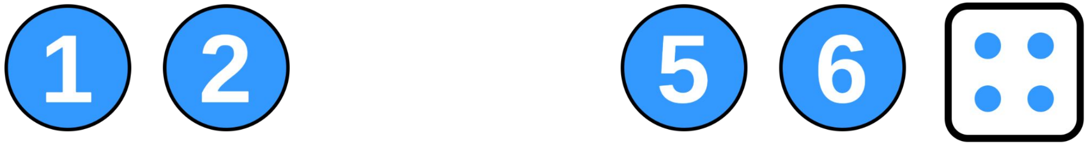
1. The piece that its number is the smallest but bigger than the dice number. (If such piece exists.)
2. The piece that its number is the biggest but smaller than the dice number. (If such piece exists.)

Examples are in the next page.

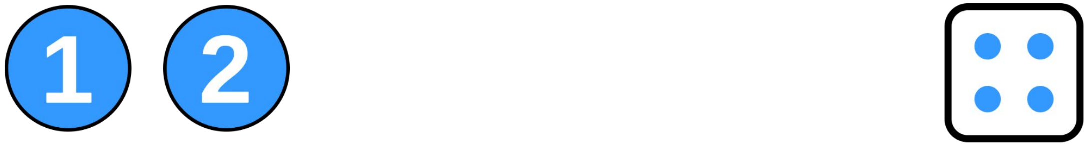
Rules of modified EWN

2. moving procedure

   b. determine the moving piece :



In this example, you can choose to move piece 2 or piece 5.



In this example, you can only move piece 2.

Rules of modified EWN

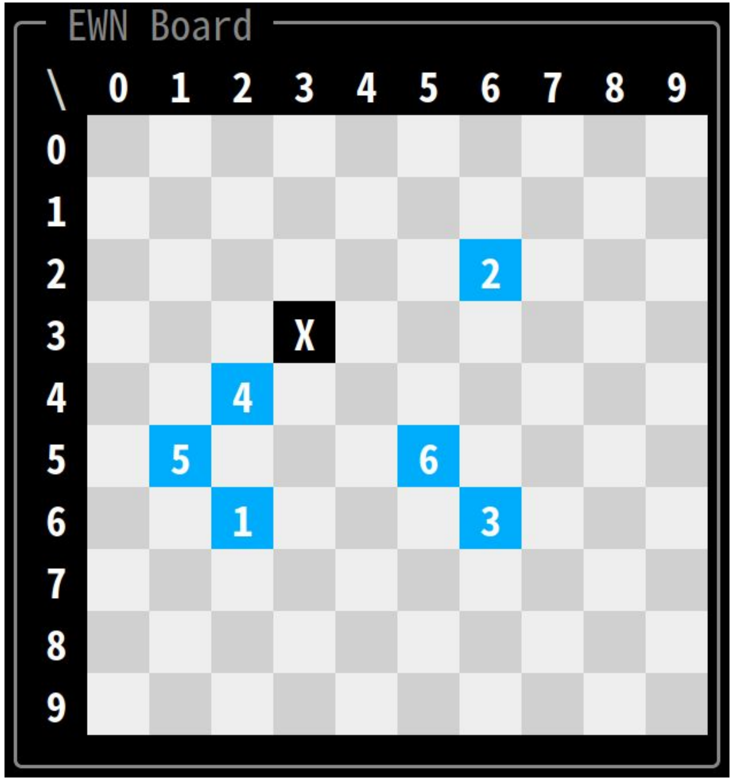
2. moving procedure

 c. piece movement and capture

 After choosing the moving piece, you can move it in every 8 directions.

 If there is a piece at the destination, the piece at the destination will be removed.

 For example, if piece 6 decides to go down-right, then piece 3 will be removed.



Rules of modified EWN

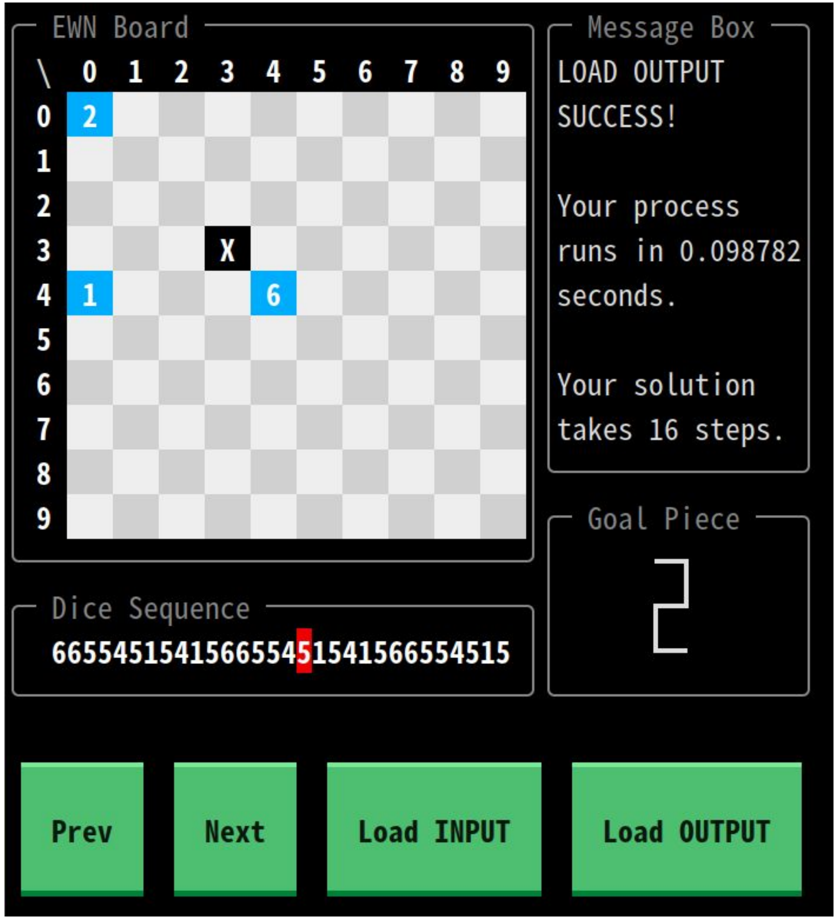
3. end condition:

Every test case will specify a number. The game ends only when the piece with the specified number reaches the position (0,0).

For example, if the test case specifies number 2, the game ends only when piece 2 reaches (0,0).

4. goal:

End the game using the minimum number of moves.



\	0	1	2	3	4	5	6	7	8	9
0	2									
1										
2										
3				X						
4	1				6					
5										
6										
7										
8										
9										

```
LOAD OUTPUT
SUCCESS!

Your process
runs in 0.098782
seconds.

Your solution
takes 16 steps.
```

Goal Piece: 2

Dice Sequence: 6655451541566554**5**1541566554515

Overview of HW1

1. programming 75%
   a. baseline 1: 25%

      2.5%x10

   b. baseline 2: 30%

      3%x10

   c. baseline 3: 20%

      2%x10

2. handwriting 25%
   a. gdb ctf 2.5%
   b. gprof ctf 2.5%
   c. report 20%
3. early bonus 5%
4. hard bonus 10%

   2%x5

total grade won't exceed 100%

Programming part

I/O and grading criteria

Programming Part - Baselines

baseline 1: <=**10 steps** 2.5%x10 3 public, 7 private

suggestion: use brute force search.

baseline 2:<=**15 steps** 3%x10 3 public, 7 private

suggestion: use heuristic search.

baseline 3: <=**20 steps** 2%x10 3 public, 7 private

suggestion: use better heuristics.

The hardest public testcases are harder than every private testcases.

Programming Part - Baselines

bonus: **22~30 steps** 2%x5   5 private, with 3 sample input

suggestion: When you inspect your algorithm in detail, you you might notice it's still doing some repetitive tasks and need further optimization. You can pass this baseline by solving this issue. **Feel free to discuss with TAs.**

These suggestions are not compulsory, you can use any algorithm to solve this homework, as long as it can be compiled and run within time limit.

Programming Part - Input format

This sample input is located in optional-visualizer.

2 =========> **goal piece is 2. range: [1,6]**

62 26 66 42 32 36 =========> **positions for piece 1 to piece 6.**

665545154156655451541566554515

=========> **The dice sequence, the length is guaranteed to be 30.**

You can view the visualized result with the optional-visualizer.

The sample code already parses input into a structure.

Programming Part - Structure format

```
typedef struct problayout{
    int goal_piece;  ⇒ range:[1,6]
    int piece_position[6]; ⇒ *Position of piece 1 is piece_position[0].*
    int dice_sequence[_dice_sequence_len_]; ⇒ each element is range:[1,6]
}prob_layout;
```

Programming Part - Output format

This sample output is located in optional-visualizer.

0.098782 ⇒ **The wall clock time in seconds of your program.**

16 ⇒ **The length of your steps**

36 45 45 55 32 41 41 51 42 51 55 66 62 51 26 15 15 4 51 40 4 3 66 55 55 44 3 2 2 1 1 0 ⇒ **This sequence should be viewed in pairs.**

First pair (36,45) means first step moves the piece at 36 to 45,

second pair (45,55) means second step moves the piece at 45 to 55

... and so on.

You can view the visualized result with the optional-visualizer.

Programming Part - Optional UI

This optional-visualizer is useful for your debugging and knowing the rules.

It is a terminal UI, so you can run it on the csie workstation.

textual is required for the UI. To install, use **pip3 install textual**.

csie workstation may require a **--break-system-packages** flag to install. You can install with the flag and it actually won't break your system packages.

If you want to know more: https://peps.python.org/pep-0668/

To run the UI, cd to the optional-visualizer, and run

./VisualizeUI.py or **python3 VisualizeUI.py**

Programming Part - Grading criteria

1. We will compile your code on csie workstation.

   Make sure your code runs normally on csie workstation.

2. Time limit for a single problem is **10s**.
3. **No multithreading or forking.** You will get zero grade in programming if caught.
4. Memory limit: **4GiB**.
5. **All warnings are treated as errors** except the ignore list shown in the makefile.
6. The judger is open source, you can get details of every grading criteria there.
7. **Don't add any flags to GCC** by putting them in target.mk, using #pragma, or any other way. If you do so, we won't compile your code, and you'll get a zero.

Programming Part - Grading criteria

8. Early Bird bonus (programming part):

If you submit your homework before the early deadline, you'll be considered an Early Bird and enjoy these bonuses:

a. Early Birds will receive their programming grade on the same day as the early deadline.

b. If Early Birds aren't happy with their grade, they can still modify their code and resubmit before the actual deadline.

c. If no new submissions are made after the early deadline, Early Birds will get a **5% bonus** on their HW1 grade, though it won't exceed 100%.

d. If Early Birds resubmit after the early deadline, but the new grade is lower than the early bird grade plus the 5% bonus, the final grade will still be the early bird grade plus 5%.

Programming Part - Grading criteria

8. Late Submission (programming & handwriting part):

You can still submit your homework after deadline.

Your submission time is decided by the server, not client. So don’t submit at the last moment because upload takes time.

Late submissions will be marked red in the submission site.

And the grade will be **deducted by 10% per day**.

Exceeding the deadline by 7 days will result in a zero grade.

Don’t submit at the last moment.

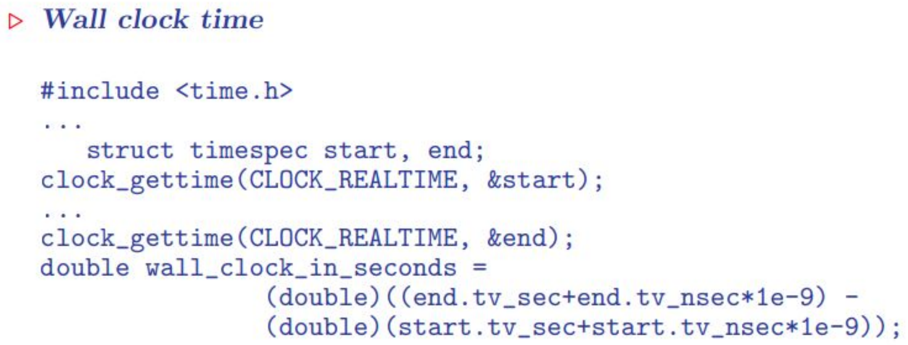
Even if you exceed the deadline by only 1 second, it is still considered late!

Programming Part - Grading criteria

grading criteria :

1. wall clock time

In homework 1, you should output the wall clock time in seconds.



▷ *Wall clock time*

```
#include <time.h>
...
   struct timespec start, end;
clock_gettime(CLOCK_REALTIME, &start);
...
clock_gettime(CLOCK_REALTIME, &end);
double wall_clock_in_seconds =
               (double)((end.tv_sec+end.tv_nsec*1e-9) -
               (double)(start.tv_sec+start.tv_nsec*1e-9));
```

We will run a timer in our judging system. If **|(your time) - (judger time)|>0.1s**, No matter you output the correct steps or not, you get 0 for the problem.

If your program doesn’t output correct steps, but output the correct time, you will still get 0.5% for the problem.

So let your program stops and output the wall clock time before 10s timeout is important!

Programming Part - Grading criteria

2. non optimal solutions

Normally, you should output the shortest possible path to end the game.

If you output a non optimal solution:

your length == optimal+1: full points - 0.5%

your length >= optimal+2: 0 points, but you still can get 0.5% from correct wall clock time.

Programming Part - File structure

```
├── makefile
└── src
    ├── code
    │   ├── solver.c
    │   ├── solver.h
    │   ├── targets.mk
    │   └── [other files]
    ├── lib
    │   ├── input_parser.c
    │   ├── input_parser.h
    │   └── problem_layout.h
    └── main.c
```

You can only modify or create file inside src/code.

Makefile will compile the file provided in **src/code/targets.mk**. If you want to add file to compile, remember to add the filename in targets.mk.

Programming Part - Submission

```
[any name you like].zip
└── code
    ├── solver.c
    ├── solver.h
    ├── targets.mk
    └── [other files]
```

You only need to submit the zip file for your code directory. The submit system will auto rename your zip file, no need to change the name.

submission site:

www.csie.ntu.edu.tw/~tcg/2024/hw1

Small Talk - why C++ is banned on hw1?

1. Do anyone tried to use RBtree to solve HW1 in the past week?
2. **Hw1 doesn’t require you to handcraft any C++ library. Thus no need to handcraft RBtree.**
3. We want to save your time and effort by discouraging you from using RBtree. Last year, many people who used C++ RBtree-related libraries failed to pass HW1 and didn’t understand what went wrong.
4. Please choose your algorithm wisely!
5. If you still insist on using RBtree to solve HW1, you can! But good luck!

Handwritting part

CTF and report

gdb gprof CTF & instructions

gdb and gprof are very useful tools when doing homeworks in this course.

You are highly recommended to complete this part before working on the programming problems!

1. gprof ctf:

   find **gprof.c** in **CTFs/gprof_ctf**

   you should use gprof to determine how many times the **next()** was called before the program terminates.

   You don’t need to modify any code, by correctly following the instructions, you will get full points.

   If your OS is not linux, you can do it on the csie workstation.

gdb gprof CTF & instructions

gprof ctf instructions:

compile with command:

> **gcc -o out gprof.c -pg**

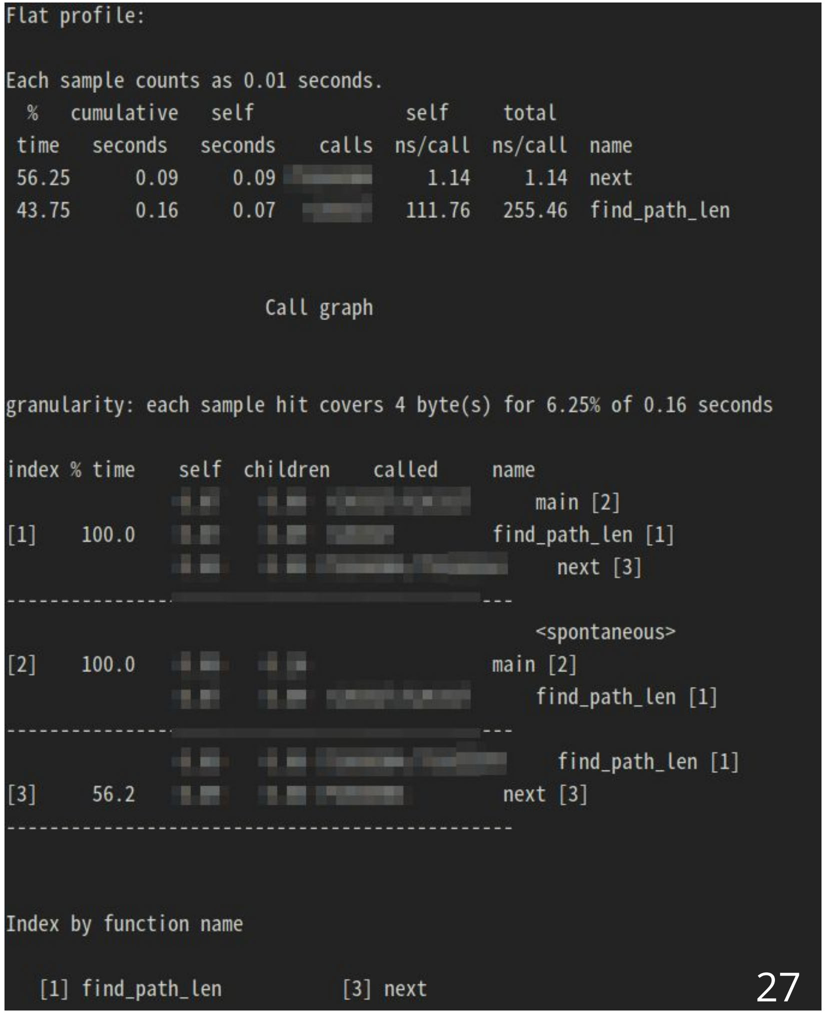
run the program: **./out**

after the program exits, a file named **gmon.out** should appear.

then run:

> **gprof out gmon.out -b**
>
> you will see the answer.

Paste the screenshot like the example on the right to the report, and if the number of calls to **next()** is correct, you will get full points.



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 56.25      0.09     0.09 [redacted]    1.14     1.14  next
 43.75      0.16     0.07 [redacted]  111.76   255.46  find_path_len


                        Call graph


granularity: each sample hit covers 4 byte(s) for 6.25% of 0.16 seconds

index % time    self  children    called     name
               [redacted]                       main [2]
[1]    100.0   [redacted]                   find_path_len [1]
               [redacted]                         next [3]
-----------------------------------------------
                                                <spontaneous>
[2]    100.0   [redacted]                   main [2]
               [redacted]                       find_path_len [1]
-----------------------------------------------
               [redacted]                          find_path_len [1]
[3]     56.2   [redacted]                   next [3]
-----------------------------------------------


Index by function name

   [1] find_path_len           [3] next
```

gdb gprof CTF & instructions

2. **gdb ctf:**

   find **gdb.c** in **CTFs/gdb_ctf**

   you should use gdb to get the value of **len** when the program crashes.

   You don’t need to modify any code, by correctly following the instructions, you will get full points.

   If your OS is not linux, you can do it on the csie workstation.

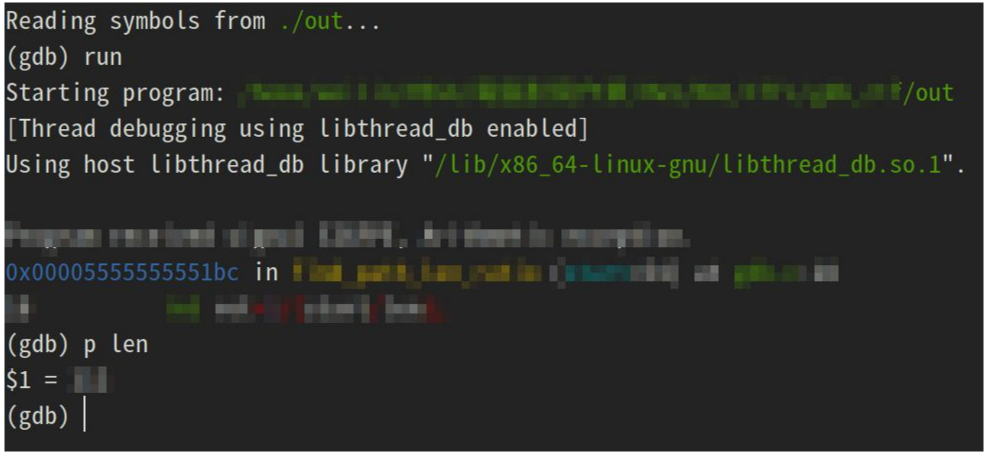
gdb gprof CTF & instructions

gdb ctf instructions:

compile with command:

gcc -o out gdb.c -g

run the program with gdb:
gdb ./out -q



```
Reading symbols from ./out...
(gdb) run
Starting program: [redacted]/out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

[redacted]
0x00005555555551bc in [redacted]
[redacted]
(gdb) p len
$1 = [redacted]
(gdb) |
```

then run the program by the gdb command:

run

after gdb reports an error, you can print the value of len by the gdb command:

p len

Paste the screenshot like the example on the right to the report, and if the value of **len** is correct, you will get full points.

gdb gprof CTF & instructions

Some suggestions of gdb ctf:

The ctf only provides very simple introduction to gdb, it is highly suggested to try the **bt** and **f** command while debugging your hw1!

Report

Your report should include:

1. heuristics & algorithms (10%)
2. experiment results (5%)
   (recommend using gprof to show your experiment result)
3. explain why your algorithm is optimal
   (e.g. why your heuristic is admissible) (5%)

Because **report has different deadline**, it has its own submission site:

www.csie.ntu.edu.tw/~tcg/2024/hw1_report

Only pdf format is accepted.