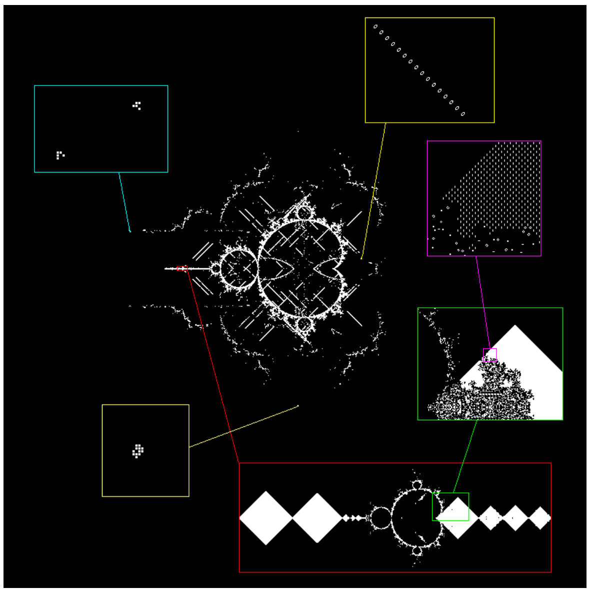
Introduction to Computers and Programming LAB-11 2015/12/16

- ✧ Your input/output must follow our format in examples
- ✧ Please sign in the computer with your own account of CS Computer Center if you have it, and save your codes in your own space in case of something bad happens on your computer.
- ✧ If you cannot finish it in time, you should demo your lab work at next lab time. (The deadline is 21:30).

I. Conway's Game of Life

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves or, for advanced players, by creating patterns with particular properties.



Ref:
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/
http://conwaylife.com/w/index.php?title=Main_Page

We want you to implement a "Conway's Game of Life" program. Don't worry!! We support a sample framework code of "Conway's Game of Life", you just only implement void Cell_life() function. The following are rules of this game:

The Game of Life is an infinite two-dimensional orthogonal grid of cells, each of which is in one of two possible states, alive('X') or dead(' '). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

I. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
II. Any live cell with two or three live neighbours lives on to the next generation.
III. Any live cell with more than three live neighbours dies, as if by over-population.
IV. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The sample data of char map[height][width] (‘X’ , ‘ ’):

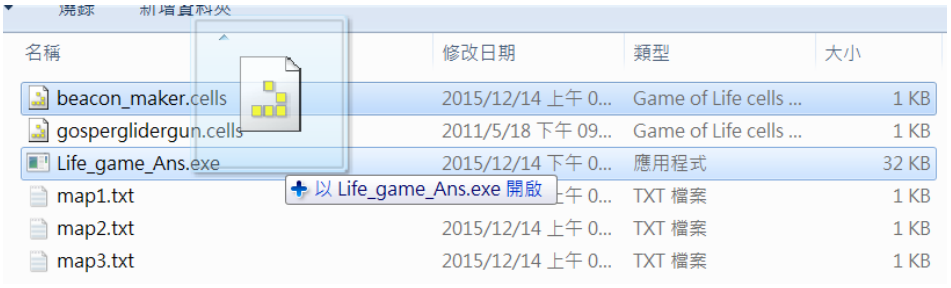
	X	X																	
X	X																		
	X					X					X								
					X		X				X							X	
						X					X				X			X	
															X			X	
		X						X							X	X			
		X	X					X	X							X			
			X													X			

The action of void Cell_life() function:

	X		
	X		
	X		

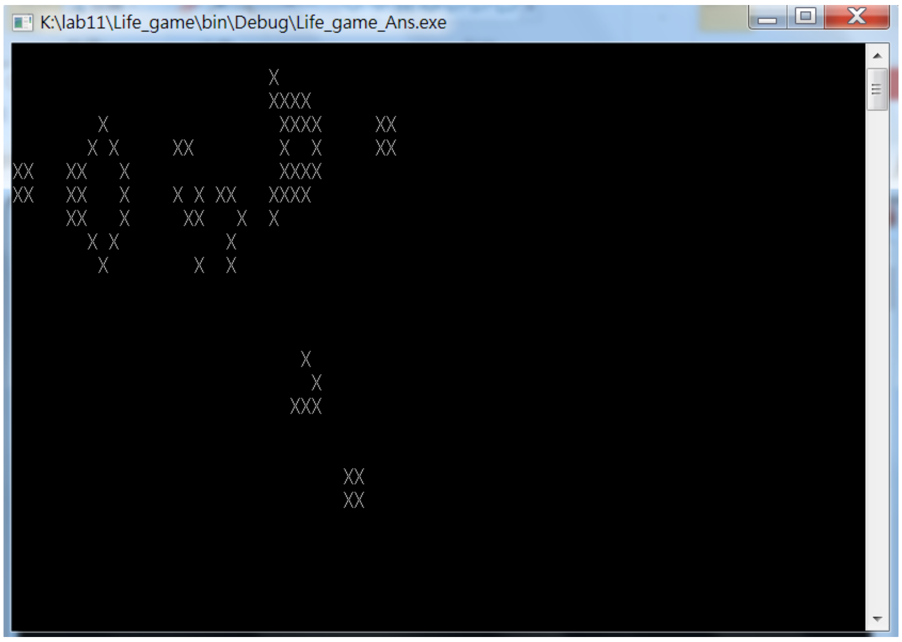
X	X	X	

You can drag drop a “xxx.cells” file on top of “Life_game_Ans.exe” [default map file is "gosperglidergun.cells"] to demo a sample game.



燒錄 新增資料夾

名稱	修改日期	類型	大小
beacon_maker.cells	2015/12/14 上午 0...	Game of Life cells ...	1 KB
gosperglidergun.cells	2011/5/18 下午 09...	Game of Life cells ...	1 KB
Life_game_Ans.exe	2015/12/14 下午 0...	應用程式	32 KB
map1.txt	上午 0...	TXT 檔案	1 KB
map2.txt	2015/12/14 上午 0...	TXT 檔案	1 KB
map3.txt	2015/12/14 上午 0...	TXT 檔案	1 KB



K:\lab11\Life_game\bin\Debug\Life_game_Ans.exe

```
                        X
                        XXXX
        X                XXXX     XX
       X X     XX        X  X     XX
XX   XX   X              XXXX
XX   XX   X    X X XX   XXXX
     XX   X     XX   X  X
       X X          X
        X       X   X



                           X
                            X
                          XXX


                               XX
                               XX
```

2. SOS from TAs

After Quiz 2 finished, TAs are going to arrange your grades in a list. However, TAs forget how to use EXCEL. Therefore, in this program, you are going write a program to help TAs maintain a list of grades, which are able to show the grades of students and the average grades of all. Here are four instructions to maintain the list:

(Let's call a name/grade pair as an element in the following)

(1) Add a student and his/her grades to list

Let user input a student's name and grade. If the input name has already in the list, your program should warn user "*Student xxx has already been added.*" Then ask user to input an instruction again. **If it is a new element, always add it at the correct position in descending order**.

(2) Remove a student from list

Let user input a student's name that user would like to remove. Before user input a student's name, your program should check whether the list is empty. If there is not element in the list, show "*The list has already been empty.*" Then ask user to input an instruction again. **If it is not empty, then remove the element from the list if the input name is in the list; otherwise, show "*No such a student called xxx.*"**

(3) Show the information of the list

If there is not element in the list, show "The list is empty." Otherwise, **show the list from the first element to the last. Moreover, print the average grade after the last element (round to the 3rd place after the decimal point).**

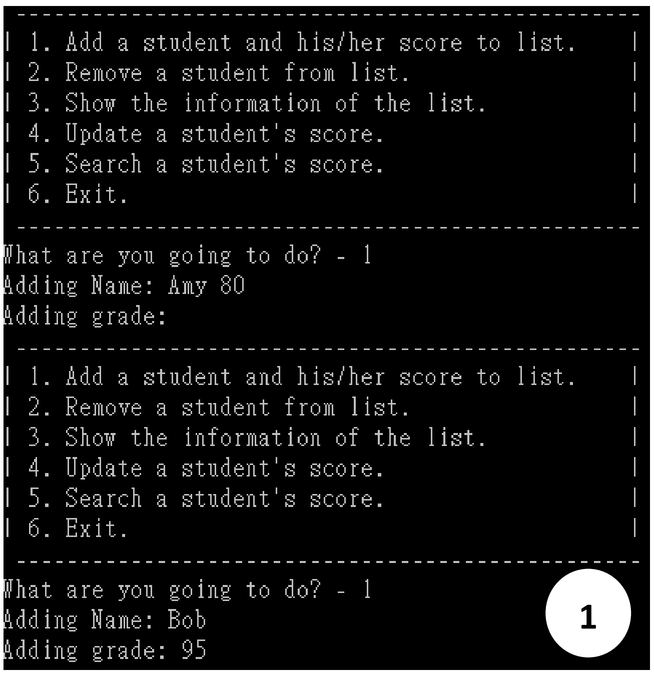
(4) Update a student's score

Use student's name to find a student's score. Print original score when your program find it so that user can update it. Otherwise, print **"*No such a student called xxx.*"**

(5) Search a student's score

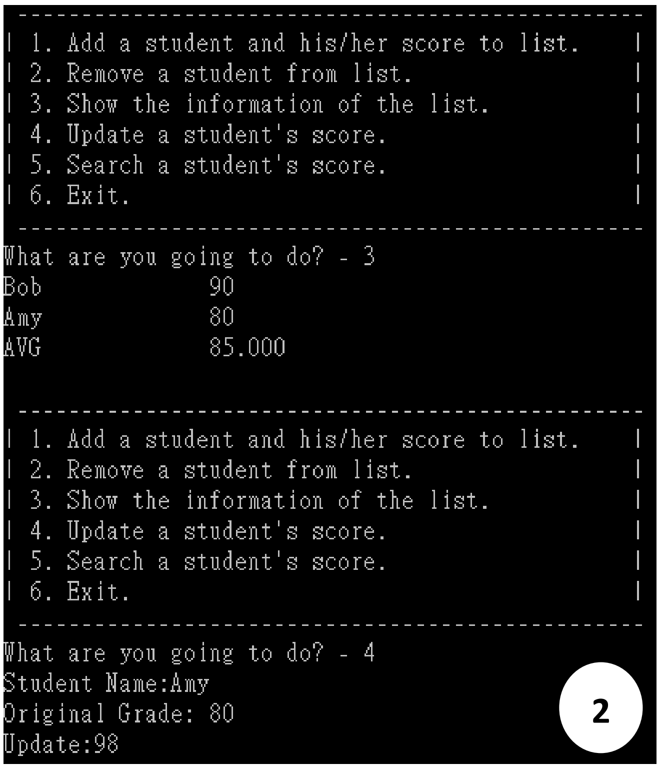
Use student's name to find a student's score. Print name and score when your program find it. Otherwise, print **"*No such a student called xxx.*"**

(6) Exit: Close the program.



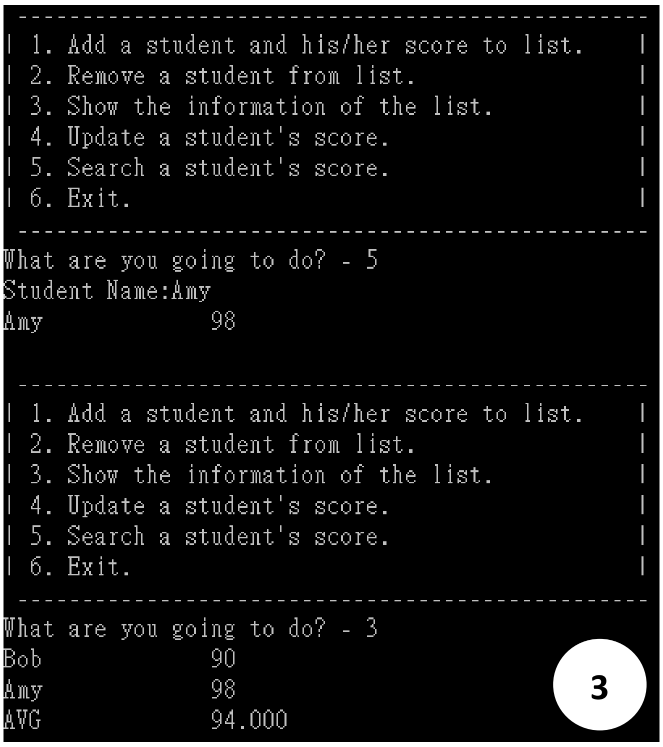
```
 ---------------------------------------------
| 1. Add a student and his/her score to list.   |
| 2. Remove a student from list.                |
| 3. Show the information of the list.          |
| 4. Update a student's score.                  |
| 5. Search a student's score.                  |
| 6. Exit.                                      |
 ---------------------------------------------
What are you going to do? - 1
Adding Name: Amy 80
Adding grade:
 ---------------------------------------------
| 1. Add a student and his/her score to list.   |
| 2. Remove a student from list.                |
| 3. Show the information of the list.          |
| 4. Update a student's score.                  |
| 5. Search a student's score.                  |
| 6. Exit.                                      |
 ---------------------------------------------
What are you going to do? - 1
Adding Name: Bob
Adding grade: 95
```

1

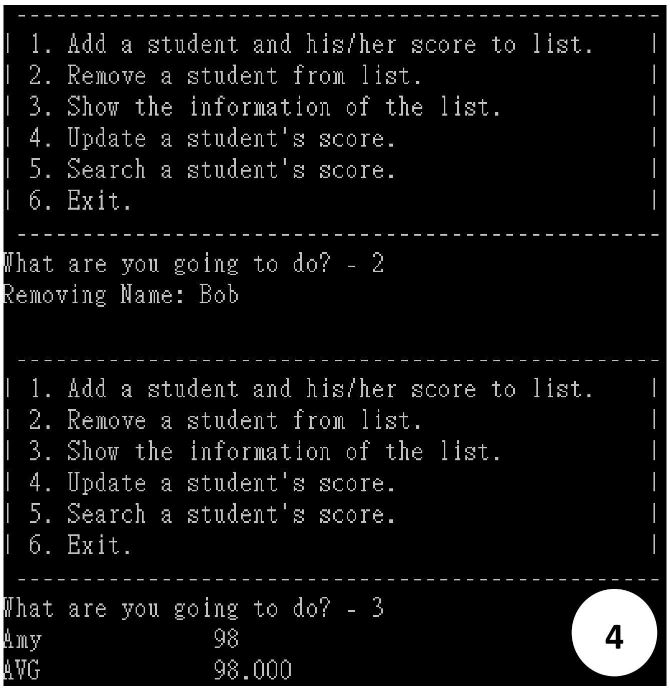


```
 ---------------------------------------------
| 1. Add a student and his/her score to list.   |
| 2. Remove a student from list.                |
| 3. Show the information of the list.          |
| 4. Update a student's score.                  |
| 5. Search a student's score.                  |
| 6. Exit.                                      |
 ---------------------------------------------
What are you going to do? - 3
Bob            90
Amy            80
AVG            85.000

 ---------------------------------------------
| 1. Add a student and his/her score to list.   |
| 2. Remove a student from list.                |
| 3. Show the information of the list.          |
| 4. Update a student's score.                  |
| 5. Search a student's score.                  |
| 6. Exit.                                      |
 ---------------------------------------------
What are you going to do? - 4
Student Name:Amy
Original Grade: 80
Update:98
```

2



```
 ---------------------------------------------
| 1. Add a student and his/her score to list.    |
| 2. Remove a student from list.                 |
| 3. Show the information of the list.           |
| 4. Update a student's score.                   |
| 5. Search a student's score.                   |
| 6. Exit.                                       |
 ---------------------------------------------
What are you going to do? - 5
Student Name:Amy
Amy             98

 ---------------------------------------------
| 1. Add a student and his/her score to list.    |
| 2. Remove a student from list.                 |
| 3. Show the information of the list.           |
| 4. Update a student's score.                   |
| 5. Search a student's score.                   |
| 6. Exit.                                       |
 ---------------------------------------------
What are you going to do? - 3
Bob             90
Amy             98
AVG             94.000
```



```
 ---------------------------------------------
| 1. Add a student and his/her score to list.    |
| 2. Remove a student from list.                 |
| 3. Show the information of the list.           |
| 4. Update a student's score.                   |
| 5. Search a student's score.                   |
| 6. Exit.                                       |
 ---------------------------------------------
What are you going to do? - 2
Removing Name: Bob

 ---------------------------------------------
| 1. Add a student and his/her score to list.    |
| 2. Remove a student from list.                 |
| 3. Show the information of the list.           |
| 4. Update a student's score.                   |
| 5. Search a student's score.                   |
| 6. Exit.                                       |
 ---------------------------------------------
What are you going to do? - 3
Amy             98
AVG             98.000
```

Note:

(1) You have to follow TA's template and finish these five functions to practice how to use struct and how to maintain a well-designed program.

(2) Structure **studentNode** contains a student's name and his/her grade.
Structure **gradeNode** contains a list of **studentNode**, number of students in the list, and average grades.

(3) If there are two students with the same grade, the order of them doesn't matter.

3. Treasure Hunter

You are a treasure hunter in a square maze. The maze is fragile, each time you go through the ground, it collapses into a hole. Your mission is to leave the maze with as more as possible treasures.

We support a sample framework code of “Treasure Hunter”,

you just need to implement the search part.

(Or you can decide not to use it and write your codes from nothing.)

About variables in the framework:

map is a 2D char Array,

'0'	means hole and you can't go through it
'1'	means the ground (after you go through, it collapses into a hole '0')
'#'	means exit (only one, after you go through, it collapses into a hole '0')
'*'	means treasure (may not exist, after you go through, it collapses into a hole '0')
'@'	means your position (only one, after you leave, it collapses into a hole '0')

(The framework also stored your start position in the variable curPos.)

bestPath is a struct Path

- score: how many treasures you found. (If you cannot find exit, you should set it to -1)
- length: how many steps you have to go
- steps: an array of struct Position, the positions you have to go through

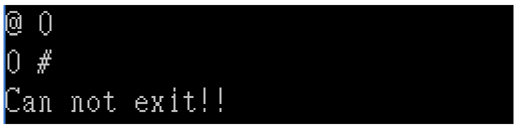
The program will output:

Whether you can find the exit or not.

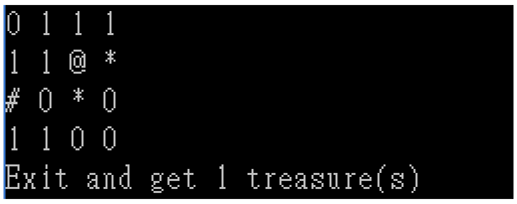
If you can exit, it will also print how many treasures you took.

You can drag drop a “xxx.map” file on top of “Q_3_Ans.exe” [default map file is "map1.map"] to demo a sample map.

Example:



```
@ 0
0 #
Can not exit!!
```



```
0 1 1 1
1 1 @ *
# 0 * 0
1 1 0 0
Exit and get 1 treasure(s)
```