CSE102 **HW06**

1 Part 1 40pts

In this homework consider the game you have written in the 5th homework. You wll write the same game with a recursive version of **opencell()** function.

If a **closed** cell is chosen; it can be either opened or flagged.

If a **closed_empty** cell is chosen to **open**; it and its 8 neighbors will be checked for emptiness and **empty neighbors** are also opened; each neighbor's 8 neighbors will also be checked and opened **recursively** until the cell that recursion has been called for is at an illegal location or already opened(an empty cell) or flagged or closed-mined.
(The nature of recursion may cause to open whole grid by only one move; which is acceptable for your homework.)

If a **closed_mined** cell is chosen to open; the game will be terminated by a loose message.

If a **flagged** cell is chosen; it can only be un-flagged.

If **all empty** cells are found; the game will be terminated by a win message.

You will print the grid with 'e' for empty cells; 'f' for flagged cells; '.' for closed cells.

In your code you will use an enumerated data type called "**cell**" as following

```
typedef enum {mined,
              empty,
              flaggedMined,
              flaggedEmpty,
              closedEmpty,
              closedMined,
              }cell;
```

The grid will be a GRIDSIZE x GRIDSIZE multi-dim array of **cell.**

Example : Assume the grid in the 5th homework as below **(Initial Grid)**, each move and its effect on the grid is colored by a different color.

ASSUME THE MATRIX OF THE GRID AS FOLLOWING CREATED BY A PROGRAMMER:

[empty mined mined empty

 empty empty mined mined

 empty empty empty empty

 empty empty empty empty]

.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Assume player made a choice to open the location of **(0,0)(1st Move)**

(Recursive openCell() function checked whole neighbors of neighbors recursively and only one closed-empty cell at the location (0,3) left because it surrounded by closed-mined cells.)

e	.	.	.
e	e	.	.
e	e	e	e
e	e	e	e

Now assume player wants to flag the location of **(0,1)** **(2nd Move)**

e	f	.	.
e	e	.	.
e	e	e	e
e	e	e	e

Now the player wants to open the location of **(0,3)** **(3rd Move)**

e	f	.	e
e	e	.	.
e	e	e	e
e	e	e	e

Every closed-empty cells are open (number of moves : 3)
A message will be printed “Congratulations! You win the game with 3 moves”
(Here three closed-mined and one flagged-mined cells was not opened.)

Signature

```
void printGrid (cell grid[][GRIDSIZE]);
```

```
int openCell(cell grid[][GRIDSIZE], int x, int y); // return value int result is the total number of cells opened.
```

```
void flagCell(cell grid[][GRIDSIZE], int x, int y); // if a cell is wanted to be flagged; check if it is empty or mined: if it is empty, flag as flagged-empty; if it is mined, flag as flagged-mined
```

```
int isCellEmpty(cell grid[][GRIDSIZE], int x, int y); //return value int result=0 if the cell is not an empty cell and result=1 the cell is an empty cell.
```

```
int isLocationLegal(int x, int y); //return value int result=0 if the location is illegal and result=1 if the location is legal(in the grid).
int asMain(); // copy your main function into this function. Use the function to take player's choice, to count their moves, and to call functions according to the player's wish.
void initGrid(cell grid[][GRIDSIZE]);
```

HINT

You can use the functions you have written in Homework 5 for printGrid(), flagCell(), isCellEmpty(), isLocationLegal().

initGrid (); is a function to initialize your matrix as an arbitrary initial game board with closed-empty and closed-mined cells.

2Part 2 60pts

Assume a C program that:

1. Reads a text from the user.
2. Changes all the alphabetical characters to lowercase and removes everything else than letters (punctuation marks, numbers etc.)
3. Exits if the input line contains the word “end”. Otherwise continues to the first step.
4. Sort the words with respect to number of their occurrences.

You will write three functions to achieve the task above :

getAndFilterWord(): Gets a word from input to filter: (task 2); if it returns 1(which means filtered word is not NULL): calls **addWord()** function to add the filtered word to the words array.

addWord(): If a “filtered word” is not in the word array already; adds it into the word array and increment the word counter by one for the same index in the word counter array; If it was in the array only increment the word counter for its index.

sortWords(): After taking every word from the input; sorts them by increasing order with respect to the number of their occurrences.

You will also submit another function called **myTest()** that tests the functions you have written. KADI system will call your myTest() function to test your code.

You can use any functions from “**stdio.h**”, “**stdlib.h**”,”**ctype.h**” and “**string.h**”.

You can assume that there are at most 500 different words that the user can enter and a word cannot be longer than 50 characters.

Signature

int getAndFilterWord(char * w); **//returns 0 if the piece of input does not contain any letter to add into the array; 1 if it does.**

void addWord(char * w, char words[MAX_NUM_WORDS][WORD_LENGTH],int occur[MAX_NUM_WORDS], int * word_count);

void sortWords(char words[MAX_NUM_WORDS][WORD_LENGTH], int occur[MAX_NUM_WORDS],int word_count);

Example:

Input:

This class 192 class!

w=”This”

getAndFilterWord(w); w=”this”

addWord(w,words,occur,word_count): w=”this”; words[0]=”this”; occur[0]=1; word_count=0;

w=”class”

getAndFilterWord(w); w=”class”

addWord(w,words,occur,word_count): w=”class”; words[1]=”class”; occur[1]=1; word_count=1;

w=”192”

getAndFilterWord(w); w=””

w=”class!”

getAndFilterWord(w); w=”class”

addWord(w,words,occur,word_count): w=”class”; occur[1]=2; word_count=2;

sortWords(words,occur,word_count) : words[0]=”class”,occur[0]=2 ; words[1]=”this”,occur[1]=1

Good luck!