KADI: http://138.68.68.37

User: gtu Password: 7516

CSE108 **LW01**

Part 1 30pts

In this part, you will write three functions that read inputs from user and return values. These functions are `readInt`, `readChar` and `readDouble`. As you can understand from function names, you will read and return an integer, a char and a double respectively.

Signature

```
int readInt();
double readDouble();
char readChar();
```

Sample Usage

```
int intValue = readInt();
double doubleValue = readDouble();
char charValue = readChar();
```

Return Value

```
readInt - Integer that you read from user
readDouble - Double that you read from user
readChar - Char that you read from user
```

Part 2 30pts

For the second part, you will write two functions. First one (`calculateBMI`) will calculate BMI (Body Mass Index). It will take two parameters: **height** (as **centimeter**) and **weight** (as **kilogram**).
BMI: "The body mass index (BMI) or Quetelet index is a value derived from the mass (weight) and height of an individual. The BMI is defined as the body mass divided by the square of the body height, and is universally expressed in units of kg/m2, resulting from mass in **kilograms** and height in **metres**." (Ref: Wikipedia). In short, you can calculate BMI as follows:

$$BMI = \frac{weight(kg)}{height(\boldsymbol{m})^2}$$

The second function (`getInfoAndCalculateBMI`) will read an integer for height and a double for weight using functions that you wrote in part 1. Then it will calculate and return BMI for that values.

Signature

```
double calculateBMI(int height, double weight);
double getInfoAndCalculateBMI();
```

Sample Usage

```
calculateBMI(170, 75.5);
getInfoAndCalculateBMI();
```

Return Value

```
calculateBMI - 26.12 (BMI score)
getInfoAndCalculateBMI - BMI for the user input (BMI score)
```

Part 3 40pts

For the last part, you will write two functions about printing information (id, name, height, weight, BMI) about a person. In the first function (`printPersonalInfo`), you will read an integer for id, a char for the initial character of the person's name, an integer for height and a double for weight using functions that you wrote in part 1. Then you will print these values as shown below. Remember that you must print exactly the same text (case sensitive) in the format except bold and italic styling. Bold texts are the values that can change with different input values.

In the second function (`getAndPrintPersonalInfo`), you will read id, name (single character), height and weight values from user using functions that you write in part 1 and print person info with `printPersonalInfo` function.

Signature

```
void printPersonalInfo(int id, char name, int height, double weight);
void getAndPrintPersonalInfo();
```

Sample Usage

```
printPersonalInfo(2, 'A', 170, 75.5);
getAndPrintPersonalInfo();
```

Output

```
printPersonalInfo(2, 'A', 170, 75.5); (Usage)

ID:2
Name:A
Height:170cm
Weight:75.50kg
BMI:26.12

printPersonalInfo - Prints person info for the parameter values
getAndPrintPersonalInfo - Prints person info for the user input
```

Good luck!

3 March 2017

KADI: http://138.68.68.37

User: gtu Password: 7918

CSE108 **LW02**

Part 1 35pts

In this part, you will calculate the mean (average) of positive integers. You are asked to implement a function **mean** which gets double values from user until user inputs a negative value, then returns the average of positive values. If user does not input any positive value, function should return 0.

Signature

```
double mean();
```

Sample Usage

```
mean();
```

Sample User Input and Return Value

```
User Input: 3,4,5,6,-1
Return: 4.5
User Input: -1
Return: 0.0
```

Part 2 35pts

This part consists of two functions. Both functions require user to enter positive values. These functions are responsible for finding the **minimum** and **maximum** values of the user inputs. Like Part 1, functions will stop working after a negative input. If user does not input any positive value, both functions should return 0.

Signature

```
double minimum();
double maximum();
```

Sample Usage

```
minimum();
maximum();
```

Sample User Input and Return Value

```
User Input for minimum: 3,4,5,-1
Return: 3.0
User Input for maximum: 3,4,5,-1
Return: 5.0
User Input: -1 (for both functions)
Return: 0.0
```

Part 3 50pts (😎20pts bonus!)

In the last part, you are asked to implement a function that calculates the sum of maximum increasing sequence (not the maximum sum of increasing sequence). Increasing sequence is a sequence $\{a_n\}$: $a_{n+1} - a_n > 0$. You need to ask for user input until user enters a negative value.

For example, assume that user inputs the following values:

`3.5, 4.1, 5.2, 6.9, 7.8, 8, 9, 2, 99, 100, 101, -1`

There are 2 increasing sequence: `*3.5, 4.1, 5.2, 6.9, 7.8, 8, 9*` and `*99, 100, 101*`. The sums are ***44.5*** and ***300***, respectively. Although the sum of second sequence is bigger than the first's, first sequence has more elements than the second. So that, the return value should be **44.5**.

If user enters either no input or no increasing sequence, you should return 0.

Signature

```
double maxSumOfIncSeq();
```

Sample Usage

```
maxSumOfIncSeq();
```

Sample User Input and Return Value

```
User Input: 3.5, 4.1, 5.2, 6.9, 7.8, 8, 9, 2, 99, 100, 101, -1
Return Value: 44.5
User Input: 99, 98, 97, 96, 95, 94, -1
Return Value: 0.0
User Input: 99, 99 ,99 ,99, -1
Return Value: 0.0
User Input: -1
Return Value: 0.0
```

Good luck!

10 March 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 4554

CSE108 **LW03**

Part 1 30pts

In this part, you will write a function that reads input from user and finds the minimum and maximum values of these inputs. You must return **minimum**, **maximum** and **status** at the same time. status is -1 if there is an error and otherwise 0. Error conditions are listed below.

Error conditions:

- Non-positive input for the first input

Signature

```
int getMinAndMax(int *min, int *max);
```

Sample Usage

```
int min = 0, max = 0, status = 0;
status = getMinAndMax(&min, &max);
2 3 4 5 6 -5
-1
```

Return Value

```
status = 0, min = 2, max = 6
status = -1, min = 0, max = 0
```

Part 2 30pts

For the second part, you will calculate multiple power operations. User inputs the base and the exponent, then the function `getMaxExp` returns the maximum value for the calculated power operations. User will enter two inputs for each operation: one is *base* and the other is *exponent*. Then you will calculate the result for the power operation. You should continue getting user input until user enters a non-positive integer. Error conditions are listed below.

Error conditions:

- Non-positive input for the first input returns -1
- Non-positive input for the exponent input returns -2

Signature

```
int getMaxExp();
```

Sample Usage

```
result = getMaxExp();
2 3 5 2 3 5 9 1 -1 (2^3, 5^2, 3^5, 9^1)
-1
2 -1
```

Return Value

```
result = 243
result = -1
result = -2
```

Part 3 40pts

For the last part, you will write two functions: one is **swap** and the other is **getMaxThreeTotal**. **swap** function takes two parameters and swaps their values. **getMaxThreeTotal** function is a little bit complex. Inputs will be like in part 2. Each iteration user enters two integers – *n1, n2*. Then you will calculate sum of the integers between n1 and n2 (including n1 and n2). This function has multiple return values like part 1. Return values are **first**, **second** and **third**, which are the three biggest sums for the user inputs. **status** is a negative integer (described below) if there is an error and otherwise 0. Error conditions are listed below.

Error conditions:

- Non-positive input for the first input, returns -1
- Non-positive input for the n2 input, returns -2
- n1>n2, returns -3

Signature

```
void swap(int *n1, int *n2);
int getMaxThreeTotal(int *first, int *second, int *third);
```

Sample Usage

```
n1 = 5; n2 = 8
swap(&n1, &n2);
int first = 0, second = 0, third = 0, status = 0;
status = getMaxThreeTotal(&first, &second, &third);
1 9 7 50 3 4 99 100 145 200 -1
1 9 7 50 -1
-1
5 -1
5 8 9 4 -1
```

Output

```
n1 = 8; n2 = 5 (swap)
status = 0, first = 9660, second = 1254, third = 199
status = 0, first = 1254, second = 45, third = 0
status = -1, first = 0, second = 0, third = 0
status = -2, first = 0, second = 0, third = 0
status = -3, first = 0, second = 0, third = 0
```

Good luck!

17 March 2017

KADI: http://bilmuh.gtu.edu.tr/kadi

User: gtu Password: 2314

CSE108 LW04

Part 1 40pts

In this part, you will write a function that makes an operation over a given array. There are 3 predefined functions which are MIN, MAX and SUM operations. You will make the operation and return the result. You are free to write any additional functions.

Error conditions:

- If array size is lower than 1, return -3456
- If given operation is not a valid operation, return -9876
- Otherwise, return the operation result.

Signature

```
typedef enum { MIN, MAX, SUM } operation;
int arrOp(int *arr, int size, operation op);
```

Sample Usage

```
int array[] = {1,2,3,4,5,6,7,-99,200}, size = 9;
arrOp(array,size,MAX);
arrOp(array,size,MIN);
arrOp(array,size,SUM);
```

Return Value

```
200, -99, 129
```

Part 2 60pts

For the second part, you will sort a given character array alphabetically and case insensitively. You may write additional helper functions such as: `swap`, `isLowercase`, `isUppercase`, `isLetter` and `uppercase` (Note that these are just suggestions and will not be graded). You can assume that; all the array elements are letters either uppercase or lowercase.

```
for i = 1 to length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
end for
```

Pseudocode: Insertion Sort

Signature

```
void alphabeticalSort(char* arr, int size);
```

Sample Usage

```
int i;
char arr[] = {'m','e','r','h','a','b','a','A','l','i'};
alphabeticalSort(arr,10);
for (i = 0; i < 10; ++i)
        printf("%c",arr[i]);
printf("\n");
```

Expected Output

```
Aaabehilmr
```

Good luck!

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 3541

CSE108 **LW05**

Part 1 40pts

In this part, you will write a function (**testResults**) that calculates the results of a multiple-choice test. It will take three arrays as parameter, **questionNumbersArr** is for the question numbers, **answersArr** is answers corresponding to these questions and **keysArr** is the correct results. User don't have to answer all the questions. There may be blank answers. If a question is blank, there is no entry for that question in both arrays (**questionNumbersArr** and **answersArr**). For example, if there are 20 questions in a test and user answers only 15 of them, then **questionNumbersArr** and **answersArr** arrays have 15 elements total.

For the result, you will calculate the ratio of correct answers over all questions. For this operation, you will write a function (**calculateResult**) that takes 3 parameters - **trueAns**, **falseAns**, **totalAns**. There is a penalty for the wrong answers. You will reduce total true answers by ¼ of wrong answers and, after that, calculate the ratio.

Error conditions:
- the operation result.

Signature

```
double testResults(int questionNumbersArr[], int qNArrSize, char answersArr[], int
ansArrSize, char keysArr[], int keysArrSize);
double calculateResult(int trueAns, int falseAns, int totalAns);
```

Sample Usage

```
int questionNumbersArr[] = {1,2,4,5,7,9}, qNArrSize = 6, ansArrSize = 6, keysArrSize =
10;
char answersArr[] = {'A','C','D','B','E','B'};
char keysArr[] = {'A','C','E','D','B','A','A','D','B','E'};

testResults(questionNumbersArr, qNArrSize, answersArr,ansArrSize, keysArr, keysArrSize);
calculateResult(5, 1, 10);
calculateResult(7, 2, 10);
```

Return Value

```
0.475
0.475
0.65
```

Part 2 60pts

For the second part, you have a different input format for the multiple-choice test. You have an answer sheet (optik form in Turkish) that represented as a **2d char array**. This array has 5 columns for the options of a question and **numOfQuestions** rows. You must use **'-'** character for blank options and **'*'** character for the marked options. To mark an option for a question, you will change the corresponding cell from '-' to '*'. **If there are two or more marked options for a question, the answer will be considered wrong.**
The result of the operation will be same as in the first part. You will calculate the ratio of correct answers over all questions in `testResults2d` function. You can use all the functions that you wrote in the part 1.

Signature

```
double testResults2d(char questionNumbersArr[][5], int qNArrSize, char keysArr[], int
keysArrSize);
```

Sample Usage

```
char answersArr[][5] = {
{'*', '-', '-', '-', '-',},
{'-', '-', '*', '-', '-',},
{'-', '-', '-', '-', '-',},
{'-', '-', '-', '*', '-',},
{'-', '*', '-', '-', '-',},
{'-', '-', '-', '-', '-',},
{'*', '-', '-', '-', '*',},
{'-', '-', '-', '-', '-',},
{'-', '*', '-', '-', '-',},
{'-', '-', '-', '-', '-',},
};
int numOfQuestions = 10, keysArrSize = 10;
char keysArr[] = {'A','C','E','D','B','A','A','D','B','E'};

testResults2d(answersArr, numOfQuestions, keysArr, keysArrSize);
```

Expected Output

```
0.475
```

Good luck!

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 7865

CSE108 **LW06**

Part 1 40pts

For the first part, you will implement a basic password generator with string interleaving function. You will interleave two given strings to another string, so that users have stronger and easy to remember passwords.

Notes:

- Result string must contain all the characters of both string. The length of the given strings may differ.
- Return value is the pointer to result string.

Signature

```
char* interleave(char str1[], char str2[], char res[]);
```

Sample Usage

```
char str1[255] = "muzaffer", str2[255] = "1234", result[255];
printf("interleave(%s,%s) => %s\n", str1, str2, interleave(str1,str2,result));
```

Output

```
interleave(muzaffer,1234) => m1u2z3a4ffer
```

Part 2 60pts

In the last part, you will convert decimal numbers to roman numerals. Wikipedia explanation for Roman numeric system is below:

> "The numbers 1 to 10 are usually expressed in Roman numerals as follows:
> **I, II, III, IV, V, VI, VII, VIII, IX, X.**
> Numbers are formed by combining symbols and adding the values, so II is two (two ones) and XIII is thirteen (a ten and three ones). Because each numeral has a fixed value rather than representing multiples of ten, one hundred and so on, according to *position*, there is no need for "place keeping" zeros, as in numbers like **207** or **1066**; those numbers are written as **CCVII** (two hundreds, a five and two ones) and **MLXVI** (a thousand, a fifty, a ten, a five and a one).
> Symbols are placed from left to right in order of value, starting with the largest. However, in a few specific cases, to avoid four characters being repeated in succession (such as **IIII** or **XXXX**), subtractive notation is used: as in this table:"

Symbol	**I**	**V**	**X**	**L**	**C**	**D**	**M**
Value	1	5	10	50	100	500	1,000

Number	**4**	**9**	**40**	**90**	**400**	**900**
Notation	IV	IX	XL	XC	CD	CM

In order to complete this part, you need to implement two functions: **append** and **roman**. Append function appends second string argument to first string argument and returns the pointer of first string's initial character. Roman function converts decimal to the given roman string parameter.
Note: Roman numerals are not capable of representing zero or negative numbers.

Signature

```
char* append(char str1[], char str2[]);
int roman(int num, char romanstr[]);
```

Sample Usage

```
char romanstr[255];
int dec = 49;
do {
	printf("Enter a num => ");
	scanf("%d",&dec);
	if(roman(dec,romanstr))
		printf("%d->%s\n", dec, romanstr);
	else
		printf("Unable to convert %d to roman numerals.\n", dec);
} while(dec > 0);
```

Sample Output

```
Enter a num => 10
10->X
Enter a num => 13
13->XIII
Enter a num => 14
14->XIV
Enter a num => 19
19->XIX
```

```
Enter a num => 102
102->CII
Enter a num => 99
99->XCIX
Enter a num => 58
58->LVIII
Enter a num => -5
Unable to convert -5 to roman numerals.
```

Good luck!

8 April 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 2337

CSE108 — Midterm

Part 1 35pts

You will implement two functions:

1. `ifind`: Finds `needle` in the `haystack` then returns its position. If `needle` is not present in `haystack`, return -1. This function is **not** case-sensitive.
2. `count`: Returns the number of occurrence of a `word` in a given `string`. This function does not count overlapping `words`.

Signature

```
int ifind(char haystack[], char needle[]); /* 20 pts */
int count(char haystack[], char needle[]); /* word → needle, haystack → string 15 pts */
```

Sample Usage

```
char haystack[255] = "deHedeheDeHEDE", needle[255] = "hede";
printf("Position: %d, Count: %d. Needle: %s Haystack : %s\n", ifind(haystack, needle), count(haystack, needle), needle, haystack);
```

• • •

```
char haystack[255] = "heHeheheheHEHE", needle[255] = "hehe";
printf("Position: %d, Count: %d. Needle: %s Haystack : %s\n", ifind(haystack, needle), count(haystack, needle), needle, haystack);
```

Output

```
Position: 2, Count: 3. Needle: hede Haystack : deHedeheDeHEDE
```

• • •

```
Position: 0, Count: 3. Needle: hehe Haystack : heHeheheheHEHE
```

Part 2 65pts

In the last part, you will implement several string functions described below.
`reverse`: reverse a given string.
`tostr`: convert a given integer to a string. (all library functions are prohibited.)
`combine`: interleave given 3 strings into result string. All strings must be present in result string and order should be preserved.
`convert`: with a given string and integer, combine the string, integer and the reverse of the string character-wise.

Signature

```
char* reverse(char str[], char rev[]); /* 15 pts */
char* tostr(int num, char str[]); /* 20 pts */
char* combine(char str1[],char str2[],char str3[],char res[]); /* 25 pts */
char* convert(char str[], int num, char res[]); /* 5 pts */
```

Sample Usage

```
char str[255] = "herhalde", str2[255] = "turta", str3[255] = "donmus", res[255];
int dec = 65109;
printf("reverse of %s:  %s\n", str, reverse(str,res));
printf("tostr(%d):%s\n", dec, tostr(dec, res));
printf("combine(%s,%s,%s):%s\n", str,str2,str3,combine(str,str2,str3,res));
printf("convert(%s,%d):%s\n",str, dec, convert(str,dec,res));
```

Sample Output

```
reverse of herhalde:  edlahreh
tostr(65109):65109
combine(herhalde,turta,donmus):htdeuorrnhtmaaulsde
convert(herhalde,65109):h6ee5dr1lh0aa9hlrdeeh
```

Good luck!

14 April 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 2478

CSE108 **LW08**

Part 1 40pts

In the first part, you're asked to implement character histogram of a given string. Your function is responsible counting each character of the string. Any characters other than letter can be considered the same and can be grouped into the last index of the given histogram array.

Signature

```
void hist(char str[], int hist[27]);
```

Sample Usage

```
void printHist(int hist[ALPHABET]) {
	int i;
	for (i = 0; i < ALPHABET-1; ++i)
		printf("%c	=> %d\n", 'A'+i, hist[i]);
	printf("Others => %d\n", hist[i]);
}

int main() {
	char str[] = "The quick brown fox jumps over the lazy dog.";
	int histogram[ALPHABET];
	hist(str,histogram);
	printHist(histogram);
	return 0;
}
```

Output

```
A	=> 1
B	=> 1
C	=> 1
D	=> 1
E	=> 3
F	=> 1
G	=> 1
H	=> 2
I	=> 1
J	=> 1
K	=> 1
L	=> 1
M	=> 1
N	=> 1
O	=> 4
P	=> 1
Q	=> 1
R	=> 2
S	=> 1
T	=> 2
U	=> 2
V	=> 1
W	=> 1
X	=> 1
Y	=> 1
Z	=> 1
Others => 9
```

Part 2 60pts

In the last part, you will implement decimal to binary and binary to decimal converter functions. Binary numbers are represented as integer arrays and only contains 1s and 0s. All binary values are stored in a fixed size integer arrays which can hold exactly 8 ints.
You will handle natural numbers (>= 0) If you're asked to convert a negative integer to binary, you will return -1.

Signature

```
int bin2dec(int bin[]);
int dec2bin(int dec, int bin[]);
```

Sample Usage

```
void printByte(int byte[]) {
	int i;
	for (i = 0; i < LEN; ++i)
		printf("%d", byte[i]);
}

int main() {
	int a[8] = {1,0,1,0,1,0,1,0};
	int b[8];

	printf("Byte:\t");
	printByte(a);
	printf(" => %d\n",bin2dec(a));

	printf("Num %d =>\t",249);
	dec2bin(249,b);
	printByte(b);
	printf("\n");
	return 0;
}
```

Sample Output

```
Byte:  10101010 => 170
Num 249 =>    11111001
```

Samples

$(10001110)_2 = (142)_{10}$

$(00001111)_2 = (15)_{10}$

$(00101110)_2 = (46)_{10}$

$(10000000)_2 = (128)_{10}$

$(11111111)_2 = (255)_{10}$

$(00000000)_2 = (0)_{10}$

Good luck!

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 9889

CSE108

Part 1 100pts

In this part, you will write a **RECURSIVE** function (**exitFromMaze**) that solves the given maze and calculates the number of exits for a start position. There may be multiple exits or no exit depending on the start position and maze structure. The function returns the number of exits for valid start positions and returns -1 for invalid start positions.

Invalid start positions:

- Non-EMPTY tiles for the start position, returns -1.

The maze has four types of tiles – **BORDER, WALL, EMPTY, USED**. These are in TILES enum.

BORDER type is used for the borders of the maze. If you can reach one of the border tiles (one of your neighbors is a border tile), that means current tile is an exit tile. You will write **isExit** function to check whether current tile is an exit tile (**return 1**) or not (**return 0**).

WALL type is used for walls that restricts you to go through that tile.

EMPTY type is for the tiles that you can use for a route to exit.

USED type is used for the tiles that you passed before.

Another helper function that you must write is **isStuck**. If there is no empty typed neighbor for the current tile, it means you stuck at that tile and the function **returns 1**, otherwise **returns 0**.

STARTER CODE: https://goo.gl/nJ3m9k

Signature

```
int exitFromMaze(TILES maze[100][100], int currentTile[2]);
int isExit(TILES maze[100][100], int currentTile[]);
int isStuck(TILES maze[100][100], int currentTile[]);
```

Sample Usage

```
typedef enum _tiles {BORDER, WALL, EMPTY, USED, START} TILES;
int main(){
	int totalExits = 0;
	int startTile[2] = {5,2};
	TILES maze[100][100];
	/* maze will be initialized and filled here */
	totalExits = exitFromMaze(maze, startTile);
	printf("Total Exits: %d\n", totalExits);
	return 0;
}
```

Return Value

```
TotalExits: 4
```

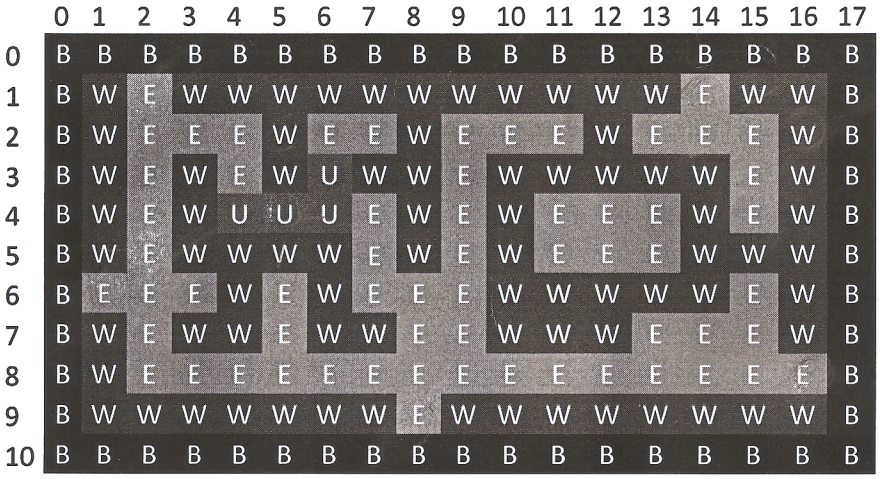

Figure 1: Sample maze

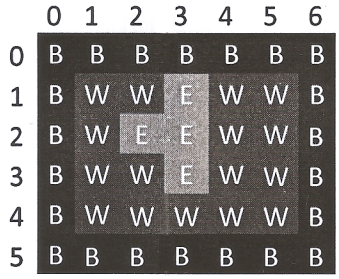

Figure 2: Maze with one exit

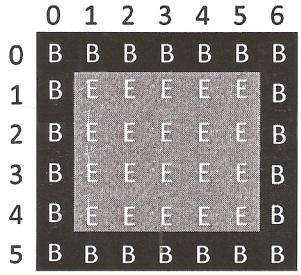

Figure 3: Maze with one exit

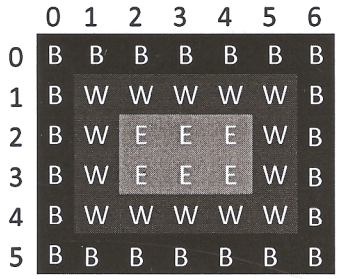

Figure 4: Maze with no exit

28 April 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 8754

CSE108 **LW10**

Part 1 100pts

In this lab, suppose that you have districts' population database. You're asked to implement the empty functions in starter code to have a deep insight about this database. These functions will help you find the cities from districts, city populations, most crowded and most fragmented cities and sort these districts with different sorting methods. All these functions are described as comments in stater code.

You need to place ilceler.txt next to lw10.c. Feel free to use any functions from string, stdio and stdlib libararies.
Database file and macro definitions may change to test your code. Do not modify them and do not use any magic number in your code. Use the predefined macros in your code.

Signature

```
#define ILSAYISI 81
#define ILCESAYISI 970
#define MAXSTRLEN 64

typedef enum order_s {
	ASC, DESC
} order;

typedef enum sortby_s {
	ISIM, IL, NUFUS
} sortby;

typedef struct ilce_s {
	char isim[MAXSTRLEN];
	char il[MAXSTRLEN];
	int nufus;
} ilce;

typedef struct iller_s {
	char isim[ILSAYISI][MAXSTRLEN];
} iller;

typedef struct nufus_s
{
	ilce ilceler[ILCESAYISI];
	int count;
} nufus;


void addIlce(nufus *nuf, char* line);
void constructIller(nufus nuf, iller *sehirler);
int getIlNufus(nufus nuf, char* il);
char* mostCrowdedCity(nufus nuf, iller sehirler);
char* mostFragmentedCity(nufus nuf, iller sehirler);
void printSorted(nufus nuf, sortby sort, order ord);
```

STARTER CODE:
https://goo.gl/EXOXvx

Good luck!

5 May 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 9182

CSE108 LW11

Part 1 100pts

In this lab, you will write a simple hi-score table system. The system includes read, write, insert, and peek operations. Each row in the table has a name field and a score field and the table can have at most ten rows. If you want to insert a row (that has a larger score value from the last row in the table) to a table which has already ten rows, you must delete the last row (or modify values with the new row values).

readTable: Reads all the values in the given file to a table (Score array with 10 elements). If file fails return -1, otherwise 0.
writeTable: Writes all the values in the given table to the given file. If file open fails return -1, otherwise 0.
insertRow: Inserts the given row to the table. Remember the insertion rules.
insertRowAndSave: Reads a table form a file, inserts a row to that table and writes the table to the file that reads before.

Signature

```
typedef struct _score {
        char name[20];
        int score;
}Score;
int readTable(char filename[], Score table[], int *tableSize);
int writeTable(char filename[], Score table[], int tableSize);
void insertRow(Score row, Score table[], int *tableSize);
void insertRowAndSave(char filename[], Score row);
```

Sample Usage

```
Score table[10];
Score row;
int tableSize = 0;

setRowValues(&row, "player1", 60); /* Optional function, no grade value*/
insertRow(row, table, &tableSize);
setRowValues(&row, "player2", 70); /* Optional function, no grade value*/
insertRow(row, table, &tableSize);
writeTable("table.bin", table, tableSize);
readTable("table.bin", table, &tableSize);
setRowValues(&row, "player3", 65); /* Optional function, no grade value*/
insertRowAndSave("table.bin", row);
```

Output

```
player2 70
player3 65
player1 60
```

12 May 2017

KADI: http:// bilmuh.gtu.edu.tr/kadi

User: gtu Password: 2222

CSE108 **LW12**

Part 1 100pts

In this lab, you will implement a basic linked list data structure with the functions described below.

addToLast: Adds the given value to the end of the list.
addToIndex: Adds the given value to the list. The value must be at the given index in the list.
removeLast: Removes the last element from the list.
removeByIndex: Removes the element at the given index.
arrayToLinkedList: Constructs and returns a linked list form an array.

Signature

```
typedef struct _node {
        int val;
        struct _node* next;
}Node;
void addToLast(Node * head, int val);
int addToIndex(Node ** head, int val, int n);
void removeLast(Node * head);
int removeByIndex(Node ** head, int n);
Node * arrayToLinkedList(int arr[], int size);
```

Sample Usage

```
int arr[10] = {0,1,2,3,4,5,6,7,8,9};
Node * list;
list = arrayToLinkedList(arr, 10);
addToLast(list, 10);
removeLast(list);
removeByIndex(&list, 0);
addToIndex(&list, 0, 0);
removeByIndex(&list, 4);
addToIndex(&list, 4, 4);
```

Output

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

Part1

In this part, you are going to find the number of occurrences of a word in a given paragraph. You are expected to find every word for once. Words should be counted **case-insensitively**. Print function is given below. Nothing should be printed other than `printHistogram` function. You are free to use any library functions from string, ctype, stdio libraries.

Signature

```
#define UNIQUEWORDS 100
#define WORDLEN 20
#define MAXSTRLEN 255

/* converts str to lowercase */
void lower(char str[]);

/* finds the num of occurrence of a word in the paragraph */
int numOccurrences(char word[], char paragraph[]);

/* finds the number of occurrences for every word in the paragraph */
void wordHistogram(char paragraph[], char words[UNIQUEWORDS][WORDLEN], int
occurrences[UNIQUEWORDS], int *count);

void printHistogram(char words[UNIQUEWORDS][WORDLEN], int occurrences[UNIQUEWORDS], int count)
{
	int i;
	for(i=0;i<count;++i)
		printf("%-10d%10s\n",occurrences[i], words[i]);
}
```

Sample Usage

```
int main() {
	char paragraph[] = "DAL sarkar Kartal KALKAR kartal kalkar dal sarkar";
	int occurrences[UNIQUEWORDS];
	char words[UNIQUEWORDS][WORDLEN];
	int count;

	wordHistogram(paragraph,words,occurrences,&count);
	printHistogram(words,occurrences,count);
	return 0;
}
```

Expected Output

```
2               dal
2            sarkar
2            kartal
2            kalkar
```

Part2

In this part, you will implement a **vector** (dynamic array) data structure with the functions described below.

init: Sets the capacity of the vector to 10 and allocates the arr for this capacity.
resize: changes vector capacity and reallocates the arr for this capacity.
addToLast: Adds the given value to the end of the vector and increases the size by one. If there is no free space in the vector (size == capacity), you must double the capacity (capacity = 2*capacity).
add: Adds the given value to the vector and increases the size by one. The value must be at the given index in the vector. If there is no free space in the vector (size == capacity), you must double the capacity (capacity = 2*capacity).

Signature

```
typedef struct _vector {
        int *arr;
        int size;
        int capasity;
}Vector;
void init(vector *v);
void resize(Vector *v, int capacity);
void addToLast(vector *v, int item);
void add(Vector *v, int item, int index);
```

Sample Usage

```
int i = 0;
int array[10] = {0,1,2,3,4,5,6,7,8,9};
vector v;
init(&v);
for (i = 0; i < 10; ++i){
    v.arr[i] = array[i];
    v.size++;
}
addToLast(&v, 11);
add(&v, 20, 5);
```

Output

```
0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10 11
0 1 2 3 4 20 5 6 7 8 9 10 11
```

Good luck!