Course: EEE2020 Data Structure

Homework 2 **Due date: 2014.4.10 before class starts**

Things you must do for this homework.

(1) **Upload *Report File* on YSCEC** (Report file should contain 'Flowchart', 'source code' and 'results')
(2) **Upload all *source codes* on YSCEC**.
(3) ***Print your Report File***, and ***hand it in*** before class

Late submissions will NOT be accepted.
This homework is not a group assignment. Group submissions will NOT be accepted. You need to submit all text files which are used as inputs or outputs of your program.

0. Prime number

Write down a program to list all the prime numbers smaller than a given number *n*. First, print out the number of prime numbers less than an input integer N. The input integer must be greater than 10000. After printing out the number of prime numbers, you need to calculate the sum of the prime numbers, and print it.

You need to store the list of prime numbers one by one into an array. You can divide a number by the list of all the prime numbers smaller than or equal to *sqrt(n)* to check if the number is a prime number. For example, to check if 101 is a prime number, you can divide 101 by the prime numbers 2, 3, 5, 7 (A[0]=2, A[1]=3, A[2]=5, A[3]=7, … automatically saved in the array) smaller than or equal to sqrt(101). It is not necessary to divide 101 by all the numbers 2,3,…, 100. This method is one of efficient algorithms to find the list of prime numbers.

Theorem) For natural number n larger than 1, if all the prime numbers in the range of 1~$\sqrt{n}$ cannot divide n, n is a prime number.

ex) input : 15212
of the prime numbers : 1230
sum of the prime numbers: 5736396

```
A[0]=2; count=1;
flag=0;  // assume n is a prime number;
for (k=2; k < n; k++)
   if (n % k == 0) { flag++; break; }

if (!flag) A[count++]=n;  // save it into an array
```

Here, you should modify the above code checking if *n* is a prime number. You need to use an array of prime numbers saved so far for division, and incrementally *n* is stored into an array when *n* is a prime number. In the next stage, the new array element can be used to check the next prime number. This procedure is continued to collect all the prime numbers in an array. Initially A[0]=2; should be given.

1. Snail matrix

Using iteration loops, write down a program to print out the following results, a snail matrix. The size of the matrix can be determined by an input. The following examples are the output of programs with a given input.

(1). N = 5

```
1   2   3   4   5
16  17  18  19  6
15  24  25  20  7
14  23  22  21  8
13  12  11  10  9
```

(2). N = 7

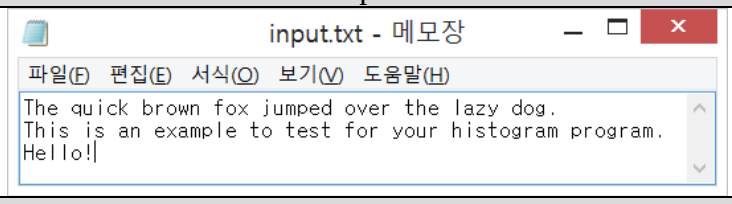
```
1   2   3   4   5   6   7
24  25  26  27  28  29  8
23  40  41  42  43  30  9
22  39  48  49  44  31  10
21  38  47  46  45  32  11
20  37  36  35  34  33  12
19  18  17  16  15  14  13
```

2. Verbal histogram

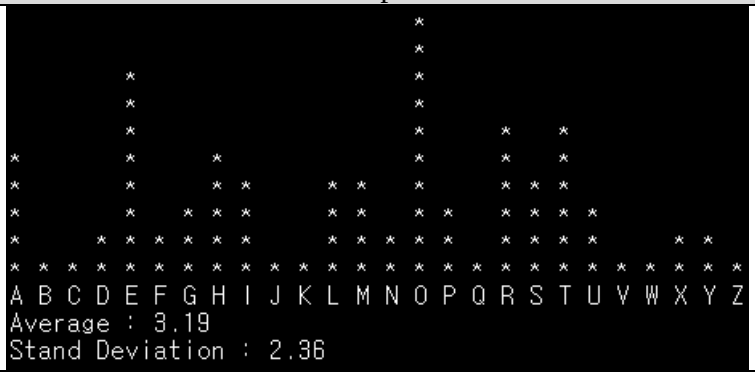
Write down a program to read text from an input file and print a vertical histogram (NOT horizontal histrogram) to show how many times each letter (but not blanks, digits, or punctuation) appears in the text, and upper case & lower case alphabets should not be discriminated, for example, ‘a’ or ‘A’ should be counted together. Additionally, calculate the average and the standard deviation of histogram. You need to prepare for your own text file. Format your output file exactly as shown below:

ex)

Input



Output



3. IP address

IP address is made up of four decimal numbers. Indeed, routers and computers identify themselves using IP address but actually it is made up of total 32 bit binary code (8 bits X 4). For this reason, we have to make parser to recognize IP address easily.

(1) Write down a program to get binary IP address from the four decimal number IP address (four digit numbers are only separated by dots(.), not blanks)
ex) input : 3.128.255.255 (Decimal code)
output : 00000011.10000000.11111111.11111111 (Binary code)

(2) Write down a program to get four decimal number IP address from binary IP address
ex) input : 11001011.10000100.11100101.00000001 (Binary code)
output : 203.132.229.1 (Decimal code)

4. The traffic jam

131	673	234	103	18
201	**96**	**342**	965	150
630	803	**746**	**422**	111
537	699	497	**121**	956
805	732	524	**37**	**331**

Figure 1

James is trying to go back to his home in rush hour. Unfortunately, a handle of his car is suddenly broken so that his car can move only down and right direction. Now, his car is located in the first element of array (0,0) and he wishes to go to the last element of array (4,4) as fast as he can. Each element in **Figure 1** represents the passage time for each element. The shortest path sum in **Figure 1** is 2427, that is, the shortest passage time.
Our goal is to write a program to find out the shortest passage time between the starting point (the first element of array) and his destination (the last element of array). Note that we are not interested in the path of how to get to the goal position from the starting position, but only in the shortest path sum.

Find the smallest path sum in a given array, and how many paths (Manhattan path) are available. Also show the distribution of the path sums (for example, range 100-120: 10, range 121-140: 15, range 141-160: 18, …, you can choose properly the range and show the corresponding frequency). You can use the code shown below to solve the problem. The code shows exploring all the possible zigzag path. You need to complete the code.

Attachment

```
#include <stdio.h>
#define N 5

int count=0;

path (A,i,j,sum,n)  // A: array, (i,j) location
int A[10][10];      // sum: cost sum
int i,j, sum, n;    // n : parameter for indentation
{
 int k; int cost;

for(k=0; k <n; k++) printf("   ");   // indentation depending on parameter n

 cost = sum + A[i][j];   // adding the cost in the path
 printf("(%d,%d)=%d",i,j,A[i][j]);  // print the element location and cost

 if( i==N-1 && j == N-1)   // reached the goal position
   printf(" sum of cost %d for the route %d \n", cost, ++count);
else
   printf("\n");

 if (i+1< N)
   path(A,i+1,j, cost ,n+1);  // check the path to A[i+1][j], depth n+1

 if (j+1< N)
   path(A,i,j+1, cost, n+1);  // check the path to A[i][j+1], depth n+1

}


main()
{
  int A[10][10]={0};
  int i,j;

  for(i=0; i < 10; i++)
    for(j=0; j < 10; j++)
       A[i][j]=rand()%100;

  path(A,0,0,0,0);  // starting from A[0][0] with the cost 0, depth 0

}
```

Output:

smallest path sum: 105
number of paths: 70
Histogram for path sum:
Range 100-120: 10
Range 121-140: 15
Range 141-160: 18
Range 161-180: 8
Range 181-200: 5

5. Mine Field

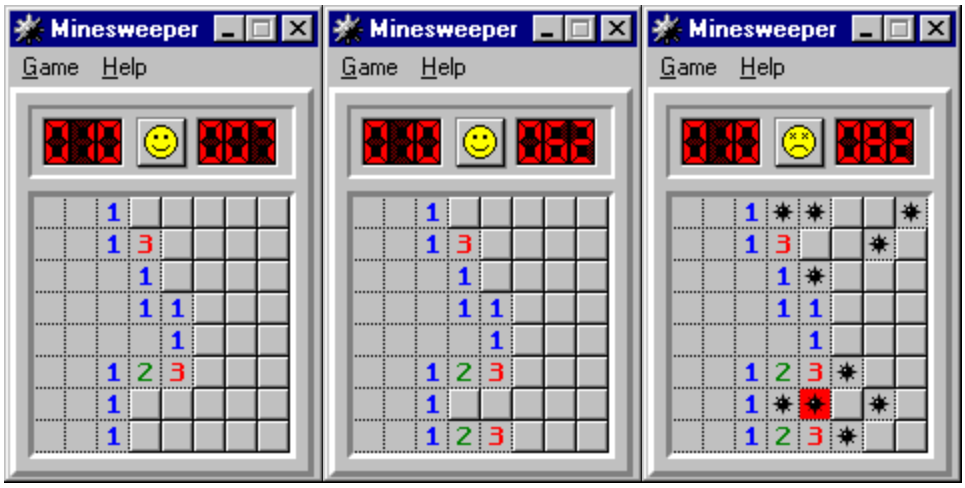


Figure 2

We are supposed to make a 'Minefield'. Unttaged cells in **Figure 2** represent there are no mines nearby it. The number in the cell indicates the number of mines in the eight neighbor cells (north, northeast, east, southeast, south, southwest, west, northwest).

First, read the number of mines as an input. Then distribute mines into random positions in the field (the number of mines should be the same as the user input). To indicate mine, we will use *(asterisk). For each cell, you need to record the number of mines around the eight neighbor cells. Here, you should print out the 8 x 8 map with the mine information.

Condition

Map size : 8 x 8
Number of mine (input) : 5~35
MineField(output)

Ex) Input : 8
Output:

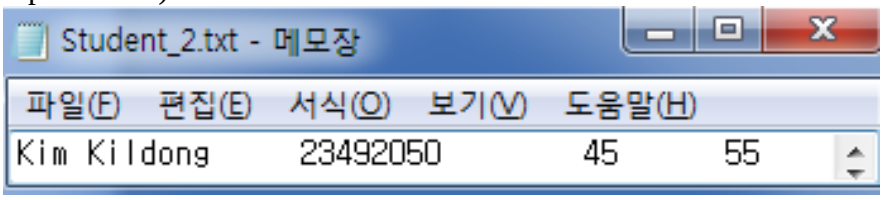
00000000
00111111
01*3*11*
123*3221
1*322*10
12*11110
01221000
001*1000

6. Check List

Write down a program. First make up input files whose names are *Student_1.txt* ~ *Student_5.txt* respectively. Each file has student name, student number and score (1~100). It follows the table as below:

File name	Student name	Student number	Score 1	Score 2
Student_1.txt	(your name)	(your number)	35	65
Student_2.txt	Kim Kildong	23492050	45	55
Student_3.txt	Son Ohgong	2583942052	23	12
Student_4.txt	Gwang huimun	69239204	43	70
Student_5.txt	Park Jiseong	345323	33	55

Input file ex)



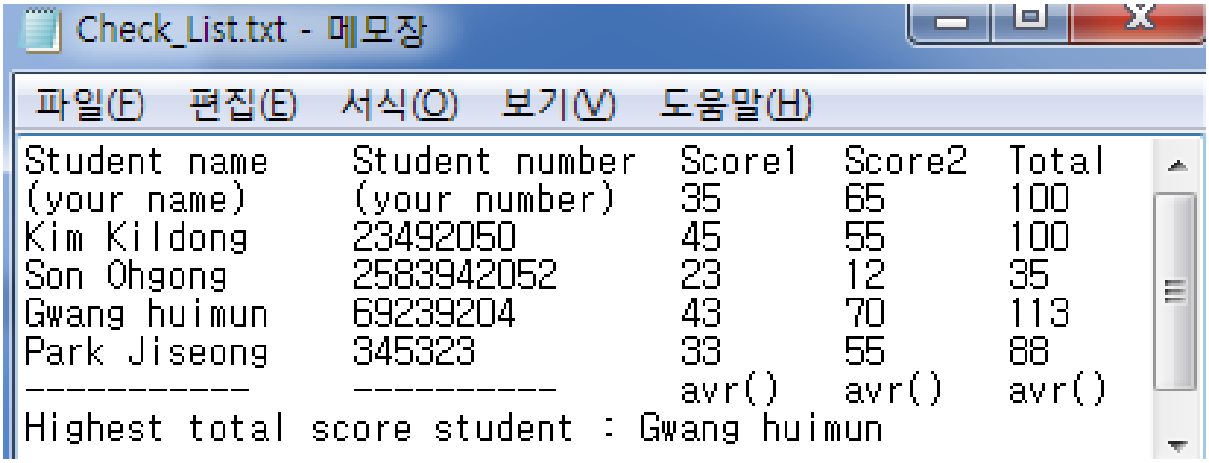
Read the student information from each file and save the information for five students in the structured array. Then calculate the total score with Score1 and Score2 for each student and save it into the structured array.

- Example of structure.

```
struct Student{
	char name[100];
	long number;
	int score1;
	int score2;
	int total;
};
```

Calculate the average score for Score1, Score2 and Total score over all the students, using *avr()* function and show the result in the output. You need to design the function in your own way. Then find a student who has the highest total score and print out all the information of that student and the saved information to *output.txt*. The format of *output.txt* should follow the screeshot as below.

- Output file format


Check_List.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Student name (your name)
Student number (your number)
Score1
Score2
Total
Kim Kildong 23492050 45 55 100
Son Ohgong 2583942052 23 12 35
Gwang huimun 69239204 43 70 113
Park Jiseong 345323 33 55 88
avr() avr() avr()
Highest total score student : Gwang huimun