Array problem I

Problem 1:

A **binary (digital) picture** is a two-dimensional array, each of whose elements is 0 or 1. The image is interpreted as light (1) on a dark (0) background. For example, here is a binary picture of a football:

Input:

```
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
```

In analyzing the picture, it is often necessary to identify the edges. A **pixel (p**icture **el**ement) is an edge pixel if it is 1 and at least one of pixels immediately above, below, left or right is 0. We can show the edges to 1 and all other pixels to 0. After identifying the edges in the preceding picture, we obtain

Output:

```
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
```

Please write the program with following requirements:

(1) Include the head file “1.h” which contains a static char array for the input.

```
#define H 5
#define W 7
char image[H][W] = {
   '0','0','0','0','0','0','0',
   '0','0','1','1','1','0','0',
   '0','1','1','1','1','1','0',
   '0','0','1','1','1','0','0',
   '0','0','0','0','0','0','0'};
```

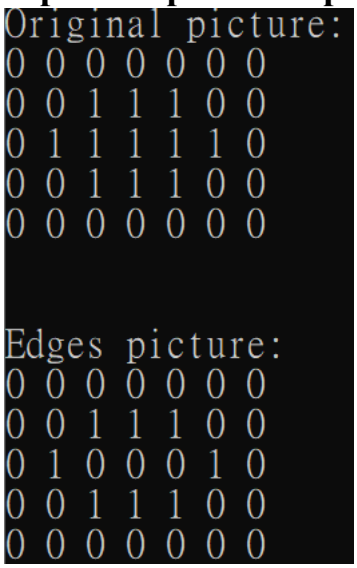
(2) Write a function

char** findEdges(char image[H][W]); which accept one original char array with dimension H*W and return one found edges **dynamic array**.

(3) Print the original array and the return processed array in main function.

(4) Do not use global array, global variable, static array in this program.

Input/Output Example:



Problem 2:

Design the matrix operations of the following:

Case I: Design a matrix multiplication function. Using dynamic array only.

Case II: Create one Toeplitz matrix: Using dynamic array only.

In linear algebra, a matrix is called a **Toeplitz matrix** when the elements of each diagonal to the main diagonal are equal between each other.
For example the following 5x5 matrix demonstrates the generic form of a 5x5 **Toeplitz matrix** :

$$t = \begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}$$

Please write the program with the following requirements.

(1) Write 5 functions as follows:

(a) int **generateMat(int, int);

(b) void inputMat(int**,int, int);

(c) void printMat(int **Mat, int M, int N);

which prints the matrix data with dimension MxN

(d) int **multiplyMat(int **Mat1, int **Mat2, int M, int N, int K);

which multiplies Mat1(MxN)by Mat2 (NxK) and returns the result.

(e) int **toeplitzMat(int **Mat, int M, int N); which use the first row and the first column from Mat to create a Toeplitz matrix.

(2) Write a main program to let user continuously input data and stop when inputting CTRL+D. **Create Toeplitz matrix** from **the answer of multiplication.**
(3) Do not use global array, global variable, static array in this program.
(4) Yu only can use printMat(…) to print 2D array.
(5) Please execute the following main function to test your program.

```
int main(){
  int M,N,K;
  int **Mat1,**Mat2,**ans,**toep;
  printf("input row,column of 1st matrix and column of 2nd matrix (m n k)): ");
  while(scanf("%d %d %d",&M,&N,&K)){

      Mat1=generateMat(M,N);
      printf("input data of a matrix: ");
      inputMat(Mat1,M,N);

      Mat2=generateMat(N,K);
      printf("input data of a matrix: ");
      inputMat(Mat2,N,K);

      ans=multiplyMat(Mat1,Mat2,M,N,K);

      printf("\nMatrix1:\n");
      printMat(Mat1,M,N);
      printf("\nMatrix2:\n");
      printMat(Mat2,N,K);
      printf("\nAnswer:\n");
      printMat(ans,M,K);

      toep = toeplitzMat(ans,M,K);
      printf("\nToeplitzMatrix:\n");
      printMat(toep,M,K);


      printf("\ninput row,column of 1st matrix and column of 2nd matrix (m n k)): ");
  }

  return 0;
}
```

Input/Output Example:

```
input row,col of 1st matrix and col of 2nd matrix (m n k): 4 3 4
input data of a matrix: 1 2 3 4 5 6 7 8 9 10 11 12
input data of a matrix: 1 2 3 4 5 6 7 8 9 10 11 12

Matrix1:
   1   2   3
   4   5   6
   7   8   9
  10  11  12

Matrix2:
   1   2   3   4
   5   6   7   8
   9  10  11  12

Answer:
  38  44  50  56
  83  98 113 128
 128 152 176 200
 173 206 239 272

ToeplitzMatrix:
  38  44  50  56
  83  38  44  50
 128  83  38  44
 173 128  83  38

input row,col of 1st matrix and col of 2nd matrix (m n k): ^Z
```

Problem 3 :

Write a program that creates an array of N random integers in the range from 1 to M and then, using the sequential search and binary search, searches the array M times using randomly generated targets in the same range. Under N<=M constraint.

(產生 N 個範圍從 1 到 M 的隨機整數，再產生 M 個範圍從 1 到 M 的隨機整數當作 target 搜尋)

Please write a program with following requirements.

(1) At least there are 3 functions in this program:

int seqsearch(int* data, int N, int target, int &c);
---Using the sequential search to find the target in the array.
---The parameter &c returns the number of the search to find the target
---The function returns 1 if the target number is in the array, otherwise returns 0.

int binsearch(int* data, int N, int target, int &c);
---Using the binary search to find the target in the array.
---The parameter &c returns the number of the search to find the target.
---The function returns 1 if the target number is in the array, otherwise returns 0.
--- Remind that you should sort the array before searching.

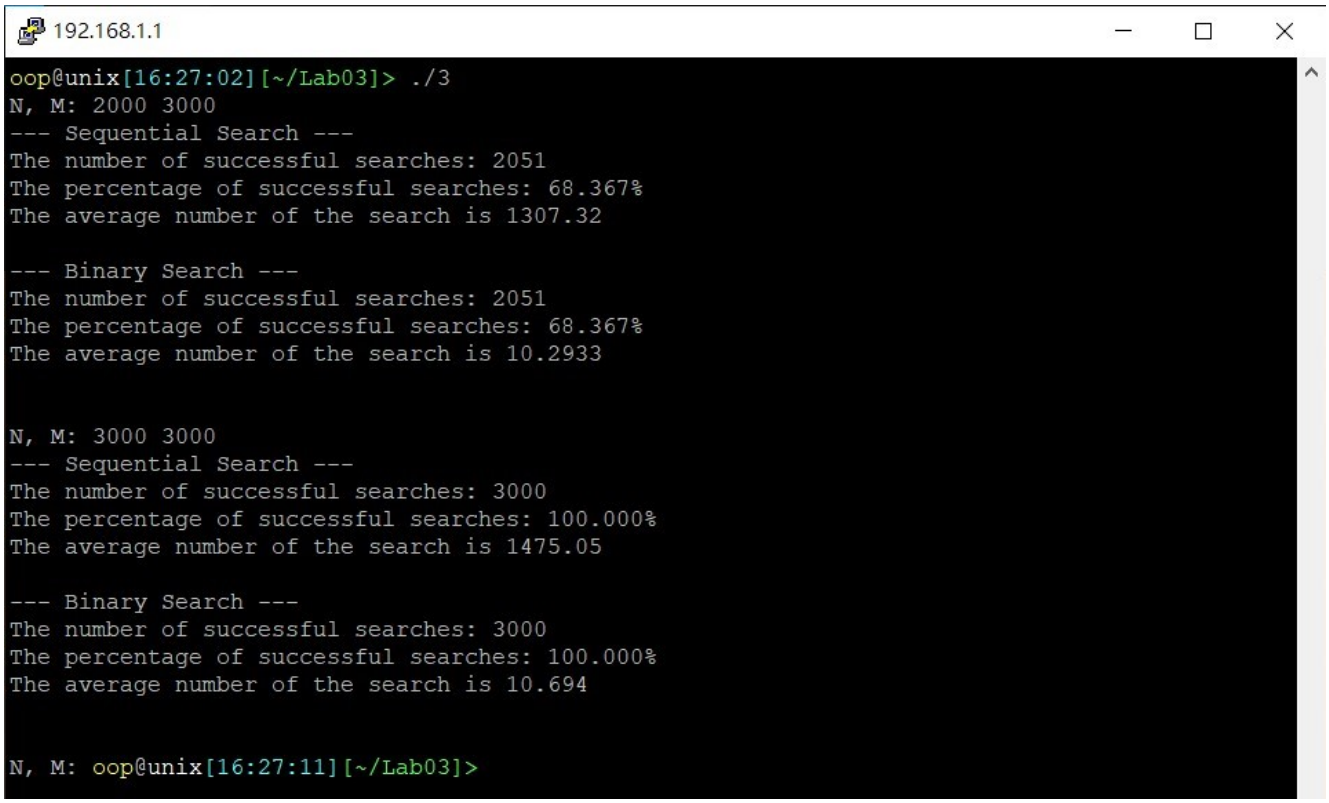
void sort(int *data, int N); which sort the data in ascendent order.

(2) The array of N random integers in the range from 1 to M cannot be duplicated.
(3) Write a main program to let user continuously input N and M and stop when inputting CTRL+Z.
(4) Do not use global array, global variable, static array in this program.
(5) At the end of the program, display the following statistics:
    1. The number of successful searches.
    2. The percentage of successful searches.
    3. The average number of tests per search.
(6) Let seed be 12345 in srand(seed).
(7) Please execute the following main function to test your program.

```
int seqsearch(int*,int, int, int&);
int binsearch(int*,int, int, int&);
void sort(int*,int);

int main(){
    int N, M;
    srand(SEED);

    while( printf("N, M: "),scanf("%d %d",&N,&M)!=EOF){
        int *data=new int[N];
        int *datatmp=new int[M];
        int *sortdata=new int[N];

     //The following three for loop to randomly create N 個  "NOT duplicated" data
      //range from 1 to M
        for (int i=1; i<=M; i++)
              datatmp[i-1] = i;
        for (int i=0; i<M; i++) {
              int t = rand()%M;
              int d = datatmp[i];
              datatmp[i] = datatmp[t];
              datatmp[t] = d;
          }
        for (int i=0; i<N; i++)
             data[i] = datatmp[i];

       for(int i=0; i<N; i++)
            sortdata[i]=data[i];

        sort( sortdata, N);
        int seqfind=0,binfind=0,sc=0,bc=0, tsc=0,tbc=0;
        for(int i=0; i<M; i++){
           int tar = rand()%M+1;
           …
           if( seqsearch(…) )    seqfind++;
           if( binsearch(…) )    binfind++;
```

```
            …
             }

        printf("--- Sequential Search ---\n");
        printf("The number of successful searches: %d\n", seqfind);
        printf("The percentage of successful searches: %.3lf%%\n", 100.*seqfind/M );
        printf("The average number of the search is %lg\n", 1.0*tsc/M );
        printf("\n--- Binary Search ---\n");
        printf("The number of successful searches: %d\n", binfind );
        printf("The percentage of successful searches: %.3lf%%\n", 100.*binfind/M );
        printf("The average number of the search is %lg\n", 1.0*tbc/M );
        printf("\n\n");
        }
    return 0;
}
```

Input/Output Example:



Problem 4 (*The Sieve of Eratosthenes*)

A prime integer is any integer greater than 1 that can be divided evenly only by itself and 1. The most efficient way to find all of the small primes (say all those less than 10,000,000) is by using a sieve such as **the Sieve of Eratosthenes**

Make a list of all the integers less than or equal to *n* (and greater than one). Strike out the multiples of all primes less than or equal to the square root of *n*, then the numbers that are left are the primes.

For example, to find all the primes less than or equal to 30, first list the numbers from 2 to 30. 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The first number 2 is prime, so keep it (we will color it green) and cross out its multiples (we will color them red), so the red numbers are not prime.
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The first number left (still black) is 3, so it is the first odd prime. Keep it and cross out all of its multiples. We know that all multiples less than 9 (i.e. 6) will already have been crossed out, so we can start crossing out at 3^2=9.
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Now the first number left (still black) is 5, the second odd prime. So keep it also and cross out all of its multiples (all multiples less than 5^2=25 have already been crossed out, and in fact 25 is the only multiple not yet crossed out).
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The next number left, 7, is larger than the square root of 30, so there are no multiples of 7 to cross off that haven't already been crossed off (14 and 28 by 2, and 21 by 3), and therefore the sieve is complete.

Therefore all of the numbers left are primes: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}.
Notice we just found these primes without dividing.

Please write a program with following requirements.
1) Write sieve function as follows
   int* sieve(int n); which receives the array size n and returns the array “prime” with size n. The array “prime” record whether the index number of the array is prime or not. For example, prime[2]=1 since 2 is a prime number, and prime[4]=0 since 4 is not a prime number.
2) Print the prime numbers in the main function.
3) Write a main program to let user continuously input data and stop when inputting CTRL+D.
4) Do not use global array, global variable, static array in this program.
5) Please follow the required sample input/output as follows.

Input/Output Example:

```
input n: 50

prime numbers:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

input n: 100

prime numbers:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

input n: ^Z
```