Lab Exercise - Week 9

Experiment 1

Program name: read_matrix.c

A matrix will be stored in a text file in the following format:

The first line of the text file will be a positive integer denoting the number of rows (m) in the matrix. The second line of the text file will be a positive integer denoting the number of columns (n) in the matrix. Then there will be mn more lines in the text file with the matrix data (real numbers) written by scanning the matrix row by row from left to right and top to bottom.

For example the matrix

will be stored in a file as

```
2
3
1.0
3.3
-5.5
2.3
4.5
6.4
```

Write a C program to **read a file name from the user** and then read the matrix from that file and store it in a 2D array. To set the size of the array during declaration, you can assume that the maximum size of the matrix is 100 x 100. Also print the matrix in its transposed form. You can use matrix1.txt, matrix2.txt, matrix3.txt and matrix4.txt as test inputs.

Your can start building from the program below.

```
#include <stdio.h>


int main()
{
  FILE *fp;
  char filename[100];


  float matrix[100][100];
  int m, n, i, j;
```

```
 /* Ask the user to give the name of the file which contains the matrix */
 printf("Enter the file name: ");
 scanf("%s", filename);

 /* Open the file for reading and check if it was opened successfully */
 fp = fopen(filename, "r");
 if (fp == 0)
 {
  printf("Error in opening the file %s.\n", filename);
  return(1);
 }
 fscanf(fp, "%d", &m);
 fscanf(fp, "%d", &n);
 for(i = 0; i < m; i++)
 {
   for(j = 0; j < n; j++)
   {
     fscanf(fp, "%f", &matrix[i][j]); /* Read one entry */
   }
 }


 /* Insert your code to print the matrix in transpose form here */



 fclose(fp); /* Close the file */
 return(0);
}
```

Experiment 2
Program name: matrix_symmetries.c

Write a C program to read a file name from the user, then read the matrix from that file, and decide whether the matrix is (i) square, (ii) symmetric and (iii) skew-symmetric. You can use matrix1.txt to matrix4.txt as test inputs.

Experiment 3:
primes.c

The following C program reads a positive integer n and print the first n prime numbers. The portion of the program which checks whether a number is prime is implemented as a function called **is_prime()**. For an integer k, is_prime(k) will return 1 if k is prime and return 0 if k is not prime. It is used as a subroutine in primes.c.

```
/*
        Program to print the first n prime numbers

*/


#include <stdio.h>


/*
        The function is_prime() accepts an integer and returns
        1 if it is prime and 0 otherwise.
*/
int is_prime(int a)
{
        int i;
        if(a < 2)
                return(0);      /* a is not prime */
        for(i = 2; i < a; i++)
                if(a % i == 0)
                        return(0);  /* a is not prime because i divides a */


        /* If the program reaches here,
        it means that no number from 2 to a-1  divides a */
        return(1);      /*a is prime */
}


int main()
{
        int n, i, count;


        printf("How many prime numbers do you need? ");
        scanf(" %d", &n);
```

```
        count = 0;      /* Counts the number of primes printed so far */
        i = 2;          /* 2 is the smallest prime */
        while(count < n)
        {
                if(is_prime(i)) /* Calls the function is_prime() */
                {
                        printf("%d, ", i);
                        count++;
                }
        i++;
        }
        printf("...\n\n");
        return 0;
}
```

Experiment 4
prime_factors.c

Write a C program to read an input n and print all the prime factors of n. If the input is 12, the output should be 2,3. This was a homework exercise in Week-6. But now, you have to **modify it by using the is_prime() function** from the previous experiment. Reuse (copy-paste) the is_prime() function from the previous experiment and use it as a subroutine here.

Experiment 5
nearly_prime.c

Write a C program to read an input n and print "Prime" if n is prime, "Nearly prime" if n is a product of exactly two primes and "Far from prime" otherwise. For example 2,3,5 etc are Prime, 4, 6, 10 etc are Nearly prime and 8, 12, 30 etc are Far from prime.

Reuse (copy-paste) the is_prime() function from the first experiment and use it as a subroutine here.