COP 3223: C Programming
Spring 2009

Arrays In C – Part 3

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

More Examples Using Arrays In C

- In this section of notes, we'll just look at several more examples of using arrays in various applications.
- The first application will be to be write a program that reads the values into a 2-d array from a file, and then computes all of the row and columns sums of the table. The size of the matrix will be the first two values in the file.

<u>Example:</u>

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \end{bmatrix}$$

The sum of each row and column in this matrix equals 15.

[*] matrix row and column sums.c

```c
#include <stdio.h>
#define MAX_ROW_SIZE 12      //maximum number of rows in the matrix
#define MAX_COLUMN_SIZE 12  //maximum number of columns in the matrix

int main()
{
    int i, j, k;  //loop control
    int numberOfRows, numberOfColumns;  //matrix dimensions
    int rowSum; //sum of a row
    int columnSum;  //sum of a column
    int matrix[MAX_ROW_SIZE+1][MAX_COLUMN_SIZE+1];  //table holding values
    //will use row = numberOfRows+1 to hold row sums
    //and column = numberOfColumns +1 to hold column sums
    FILE *inFilePtr;  //input file holds input data

    if ( (inFilePtr = fopen("matrix input.dat", "r")) == NULL ) {
         printf("Sorry, could'nt open input file\n");
    }
    else {
         //assume first two values in file are row and column size
         fscanf(inFilePtr, "%d%d", &numberOfRows, &numberOfColumns);
         //initialize the row and column sum counter positions to 0
         for (i = 0; i < numberOfRows; ++i) {
             matrix[i][numberOfColumns] = 0;
         }//end for stmt to initialize row sum counters
         for (j = 0; j < numberOfColumns; ++j) {
             matrix[numberOfRows][j]= 0;
         }//end for stmt to initialize column sum counters

```

[*] matrix row and column sums.c

```c
    for (i = 0; i < numberOfRows; ++i) { //row loop
        for (j = 0; j < numberOfColumns; ++j) { //column loop
            fscanf(inFilePtr, "%d", &matrix[i][j]);
        }//end column loop for stmt
    }//end row loop for stmt
    //print matrix
    printf("\n");
    for (i = 0; i < numberOfRows; ++i) {
       for (j = 0; j < numberOfColumns; ++j) {
            printf("%3d", matrix[i][j]);
       }//end column loop
       printf("\n");//move to next row
    }//end row loop
    printf("\n\n");
    //compute row sums
    for (i = 0; i < numberOfRows; ++i) {//go down all rows computing sums
       for (j = 0; j < numberOfColumns; ++j) { //go across all columns
          matrix[i][numberOfColumns] += matrix[i][j];
       }//end column loop
    }//end row loop
    //compute column sums
    for (j = 0; j < numberOfColumns; ++j) {//go across all columns
       for (i = 0; i < numberOfRows; ++i) {//go down all rows
          matrix[numberOfRows][j] += matrix[i][j];
       }//end row loop
    }//end column loop

    printf("\n");
    printf("   The row sums are: ");
```

```
        printf("\n");
        printf("   The row sums are: ");
        for (i = 0; i < numberOfRows; ++i){
            printf("%4d", matrix[i][numberOfColumns]);
        } //end printing row sums
        printf("\n");
        printf("The column sums are: ");
        for ( j = 0; j < numberOfColumns; ++j) {
            printf("%4d", matrix[numberOfRows][j]);
        }
        printf("\n");
  }//end fopen else stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

Resources | Compile Log | Debug | Find Results

Insert | Ready.

K:\COP 3223 - Spring 2009\COP 3223 Progr...

```
  1   2   3   4   5
  2   3   4   5   1
  3   4   5   1   2
  4   5   1   2   3
  5   1   2   3   4



   The row sums are:    15   15   15   15   15
The column sums are:    15   15   15   15   15


Press any key to continue . . .
```

K:\COP 3223 - Spring 2009\COP 3223 Program...

```
  8   3   9   0  10
  3   5  17   1   1
  2   8   6  23   1
 15   7   3   2   9
  6  14   2   6   0



   The row sums are:    30   27   40   36   28
The column sums are:    34   37   37   32   21


Press any key to continue . . . _
```

More Examples Using Arrays In C

- For our second application we'll look at creating a magic square. A magic square is a square matrix that contains each of the numbers 1, 2, 3, ..., n^2 exactly once and has the sum of the numbers in each of its rows, columns and main diagonals equal to the same value. The sum of each row, each columns and the main diagonals is called the magic constant and is equal to $(n^3+n)/2$. So, for example, a 5 X 5 magic square should include each of the numbers 1, 2, 3,…, 25 exactly once and the row, column, and diagonal sums should be $(5^3 + 5)/2 = 65$.
- I want you to go thru this code and figure out the technique that I used to generate the magic square. How did I know where to place each value in the matrix?

More Examples Using Arrays In C

- I used a technique known as the Siamese method which is:
  - Starting from the central column of the first row with the number 1, the fundamental movement for filling the squares is diagonally up and right, one step at a time. If a filled square is encountered, one moves vertically down one square instead, then continuing as before. When a move would leave the square, it is wrapped around to the last row or first column, respectively.
  - For magic squares of odd size `n` constructed using the Siamese method, several factors apply:

1. Smallest number = 1
2. Largest number = n^2
3. Middle number = $(n^2 /2) + 0.5$ (real number division here)
4. The middle number is always on the diagonal bottom left to top right
5. The largest number is always opposite 1 in an outside column or row.

magic square.c

```c
//Arrays In C - Part 3 - Magic Square program
//This program will generate magix squares of any size from 1 to 15
//User enters the size magic square they want to create.
//February 22, 2009    Written by:  Mark Llewellyn

#include <stdio.h>
#define MAX 15

int main()
{
  int magicSquare[MAX][MAX] = { {0} };  //the magic square 2-d array, initialized to (
  int size; //user determined size of magic square
  int i, row, col, oldRow, oldCol;  //loop controls

  printf("This program creates a magic square of a specified size.\n");
  printf("The size must be an odd number between 1 and 15.\n\n");
  printf("Please enter the size of the magic square you want to create: ");
  scanf("%d", &size);
  //build magic sqaure
  row = 0;
  col = size / 2; //note integer division, if size = 5, col equals 2
  for (i = 1; i <= size * size; i++) {
     if (magicSquare[row][col] != 0) {
        row = (oldRow + 1) % size;
        col = oldCol;
     }//end for stmt
```

Resources | Compile Log | Debug | Find Results

Insert | 46 Lines in file

magic square.c

```c
      magicSquare[row][col] = i;
      oldRow = row;
      oldCol = col;
      if (row > 0)
         row = row - 1;
      else row = size - 1;
      col = (col + 1) % size;
   }//end for stmt
   //print out the magic square
   for (row = 0; row < size; row++) {
     for (col = 0; col < size; col++)
       printf("%5d", magicSquare[row][col]);
     printf("\n");
   }//end for stmt

   printf("\n\n");
   system("PAUSE");
   return 0;
}//end main function
```

K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - Part 3\magi...

```
This program creates a magic square of a specified size.
The size must be an odd number between 1 and 15.

Please enter the size of the magic square you want to create: 5

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


Press any key to continue . . .
```

More Examples Using Arrays In C

- For our second example of using arrays and loops, we'll generate the first 30 Fibonacci numbers. Fibonacci numbers are a sequence of integers where the each number is the sum of the two preceding numbers. The first few Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on.
- We'll write a program that uses a 1-d array named `fibonacci`, initialize the first two values in the array and then uses a loop to generate the remaining Fibonacci numbers.
- We'll have our program write output to both the screen and to a file named "`fibonacci.dat`". What we'll do with this output file, is slightly modify the program you wrote for Assignment #3 and determine which of the first 30 Fibonacci numbers are prime.

An aside on Fibonacci numbers:

On many plants, the number of petals is a Fibonacci number: buttercups have 5 petals (F(5)) lilies and iris have 3 petals (F(4)); some delphiniums have 8 (F(6)); corn marigolds have 13 petals (F(7)); some asters have 21 (F(8)) whereas daisies can be found with 34 (F(9)), 55 (F(10)) or even 89 (F(11)) petals.

Look at your own hands: You have **2** hands each of which has **5** fingers, each of which has **3** parts separated by **2** knuckles; all Fibonacci numbers!

Some weird/interesting facts about the Fibonacci sequence.

Every third number in the Fibonacci sequence is even.

Every k^{th} number in the sequence is a multiple of F(k). For example F(4) = 3, and F(8) = 31 which is 7X F(4), F(12) = 144 which is 48X F(4).

In the movie "The DaVinci Code" Fibonacci numbers were used as a combination to unlock a safe.

If you take any 3 consecutive Fibonacci numbers, sum them and divide the sum by two, you will always get the third number! Example: 13 + 21 + 34 = 68 and 68/2 = 34!

fibonacci.c

```c
//This program uses an array and a loop to generate the first 30
//Fibonacci numbers and prints them out.
//February 22, 2009     Written by: Mark Llewellyn

#include <stdio.h>
#define MAX 30

int main()
{
    int i; //loop control
    int fibonacci[MAX]; //1-d array of Fibonacci numbers

    //initialize first two fibonacci numbers
    fibonacci[0] = 0;
    fibonacci[1] = 1;
    for (i = 2; i < MAX; ++i) {
        fibonacci[i] = fibonacci[i-2] + fibonacci[i-1];
    }//end fibonacci number generation loop
    printf("\nThe First %2d Fibonacci Numbers\n", MAX);
    printf("------------------------------\n");
    for (i = 0; i < MAX; ++i){
        printf("Fibonacci(%2d): %8d\n", i, fibonacci[i]);
    }

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - Part 3\fibonacci.exe

```
The First 30 Fibonacci Numbers
------------------------------
Fibonacci( 0):          0
Fibonacci( 1):          1
Fibonacci( 2):          1
Fibonacci( 3):          2
Fibonacci( 4):          3
Fibonacci( 5):          5
Fibonacci( 6):          8
Fibonacci( 7):         13
Fibonacci( 8):         21
Fibonacci( 9):         34
Fibonacci(10):         55
Fibonacci(11):         89
Fibonacci(12):        144
Fibonacci(13):        233
Fibonacci(14):        377
Fibonacci(15):        610
Fibonacci(16):        987
Fibonacci(17):       1597
Fibonacci(18):       2584
Fibonacci(19):       4181
Fibonacci(20):       6765
Fibonacci(21):      10946
Fibonacci(22):      17711
Fibonacci(23):      28657
Fibonacci(24):      46368
Fibonacci(25):      75025
Fibonacci(26):     121393
Fibonacci(27):     196418
Fibonacci(28):     317811
Fibonacci(29):     514229


Press any key to continue . . . _
```

I've put two versions of this program on the web site for you. This version uses an `int` type array and prints only the first 30 Fibonacci numbers. The second version uses a `double` type array and prints the first 80 Fibonacci numbers.

fibonacci.c | [*] prime fibonacci numbers.c | matrix row and column sums.c

```c
// Arrays In C - Part 3 - Finding prime Fibonacci numbers
//This program is a slight modification of the program you wrote for
//assignment #3
//February 22, 2009      Written by: Mark Llewellyn

#include <stdio.h>

int main(void)
{
    int number; //a fibonacci number
    int i, index; //loop control
    int numfactors = 0;//number of factors of a number
    FILE *inFilePtr;  //input file holds fibonacci numbers

    if ( (inFilePtr = fopen("fibonacci.dat", "r")) == NULL ) {
         printf("Sorry, could'nt open the input file\n");
    }
    else{
        printf("\n");
        index = 0;
        fscanf(inFilePtr, "%8d", &number);
        while (!feof(inFilePtr)) {
           if (number < 2){
             printf("Fibonacci(%d) = %6d: is prime\n", index, number);
           }
```

fibonacci.c | prime fibonacci numbers.c | matrix row and column sums.c

```c
            }
            else {
              numfactors = 0;
              for(i = 1; i <= number; ++i) {
                  if (number % i == 0) {
                     numfactors++;
                  }//end if stmt
              }//end for stmt
              if (numfactors > 2){ //number is not prime
                printf("Fibonacci(%d) = %6d: is not prime\n", index, number);
              }//end if
              else {
                 printf("Fibonacci(%d) = %6d: is prime\n", index, number);
              }//end else stmt
            }//end else stmt
            index++;
            fscanf(inFilePtr, "%8d", &number);
        }//end while stmt
    }//end else
    fclose(inFilePtr);
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - ...

```
Fibonacci(0) =      0: is prime
Fibonacci(1) =      1: is prime
Fibonacci(2) =      1: is prime
Fibonacci(3) =      2: is prime
Fibonacci(4) =      3: is prime
Fibonacci(5) =      5: is prime
Fibonacci(6) =      8: is not prime
Fibonacci(7) =     13: is prime
Fibonacci(8) =     21: is not prime
Fibonacci(9) =     34: is not prime
Fibonacci(10) =     55: is not prime
Fibonacci(11) =     89: is prime
Fibonacci(12) =    144: is not prime
Fibonacci(13) =    233: is prime
Fibonacci(14) =    377: is not prime
Fibonacci(15) =    610: is not prime
Fibonacci(16) =    987: is not prime
Fibonacci(17) =   1597: is prime
Fibonacci(18) =   2584: is not prime
Fibonacci(19) =   4181: is not prime
Fibonacci(20) =   6765: is not prime
Fibonacci(21) =  10946: is not prime
Fibonacci(22) =  17711: is not prime
Fibonacci(23) =  28657: is prime
Fibonacci(24) =  46368: is not prime
Fibonacci(25) =  75025: is not prime
Fibonacci(26) = 121393: is not prime
Fibonacci(27) = 196418: is not prime
Fibonacci(28) = 317811: is not prime
Fibonacci(29) = 514229: is prime


Press any key to continue . . . _
```

Case Study – Caesar Cipher

- In cryptography, a Caesar cipher, also known as a Caesar's cipher, a shift cipher, or Caesar code, is one of the simplest and most widely known encryption techniques and falls in the general category of cyclic substitution ciphers.
- Basically, a Caesar cipher is a substitution cipher where each letter in plaintext (the uncoded message) is replaced by a letter some fixed number of positions down the alphabet, typically referred to as the shift factor, the letter in this shifted position becomes the letter in the ciphertext (the coded message).
- This encryption technique was named after Julius Caesar, who used it to communicate with his generals. This encryption technique is often incorporated as part of more complex encryption techniques such as the Vigenère cipher and the ROT13 systems.

Case Study – Caesar Cipher

Example of a Caesar cipher with shift of 4

```
Encoder
Original alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Shifted alphabet:  W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
Plaintext message:  I LOVE C PROGRAMMING
Encoded message:    E HKRA Y LNKCNWIIEJC
Since: I = E with shift of 4, L = H with shift of 4, . . .


Decoder
Encoded message:  E HKRA Y LNKCNWIIEJC
E = I with shift -4, H = L with shift -4, . . .
Decoded message:  I LOVE C PROGRAMMING
```

Case Study – Caesar Cipher

- What we'll do in this case study is develop two programs, one that encodes text messages using a Caesar cipher and a second program that will decode a message that was encrypted using a Caesar cipher.
- Both programs will be rather similar in nature, in that both will use a 2-d array to hold an original alphabet and a number of shifted versions of that alphabet. The encoder program will read a message from the terminal and ask the user to enter the shift amount for the encryption. The encryption program will also read the original alphabet from a file named "`alphabet.dat`". It will encrypt the message and write the shift amount and the encoded message to a file named "`encoded message.dat`".

Case Study – Caesar Cipher

- The decoder program will read the file named `“encoded message.dat”` and then using the shift amount that was stored in this file; decode the message back into plaintext.
- Both of the programs are a bit larger than most of the programs we’ve seen so far in the course. We will return to this case study later in the semester when we covered functions and we’ll rewrite these two programs and make both cosmetic and functional improvements to the code.
- I have not shown all the code for either of these two program in this set of notes. I’ve only printed out some of the code to highlight the approach that was taken to solve the problem. You will need to download the code from our code page in order to see the whole thing! I encourage you to play with this program a bit.

encoder.c

```c
//Case Study - Caesar Ciphers - Encoder
//Several shifts are stored in a 2d array.
//user enters shift to encode/decode a message
//February 22, 2009    Written by: Mark Llewellyn

#include <stdio.h>
#define SIZE 26     //number of characters in alphabet
#define SHIFTS 26  //maximum number of ciphers stored in arrays + original
#define LENGTH 40

int main()
{
    int i, j;  //loop control
    int found;  //boolean for finding search character
    char searchChar;  //character to be found
    int clearPosition, encodePosition;  //index positions of characters
    int charCounter; //count characters in the clearMessage
    int shiftAmount; //integer describing the code shift amount
    int positionShift;  //index to new position in table after shift
    char codes[SHIFTS][SIZE];  //table holding codes
    char clearMessage[LENGTH]; //user entered message
    char codedMessage[LENGTH]; //encoded message
    FILE *inFilePtr;  //input file holds initial alphabet
    FILE *outFilePtr; //output file holds encoded message and shift amount

    if ( (inFilePtr = fopen("alphabet.dat", "r")) == NULL ) {
         printf("Sorry, could'nt open input file\n");
```

[*] encoder.c

```c
        }
        else {

             for (i = 0; i < SIZE; ++i) {
                 fscanf(inFilePtr, "%c\n", &codes[0][i]);
             }//end reading initial alphabet for stmt

            //generate caesar codes for several shifts
            shiftAmount = 1;
            for (i = 1; i < SHIFTS; ++i) { //row loop
               for (j = 0; j < SIZE; ++j) { //column loop
                  codes[i][j] = codes[0][(j + shiftAmount) % 26];
              }//end column loop for stmt
              shiftAmount++;
            }//end row loop for stmt
      //TESTING CODE ** REMOVE AT END
            printf("\n");
            for (i = 0; i < SHIFTS; ++i) {
               for (j = 0; j < SIZE; ++j) {
                  printf("%c  ", codes[i][j]);
               }//end
               printf("\n");
            }


```

Once the SHIFTS number of shifted alphabets has been generated, print them out to see if they look ok. See next page. First row is unshifted alphabet, 2nd row is a shift of 1, 3rd row is a shift of 2 and so on.

K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - Part 3\encoder.exe

```
a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a
c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b
d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c
e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d
f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e
g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f
h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g
i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h
j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i
k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j
l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k
m  n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l
n  o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m
o  p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n
p  q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
q  r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p
r  s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q
s  t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r
t  u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s
u  v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
v  w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u
w  x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v
x  y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w
y  z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x
z  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y


Press any key to continue . . .
```

[*] encoder.c

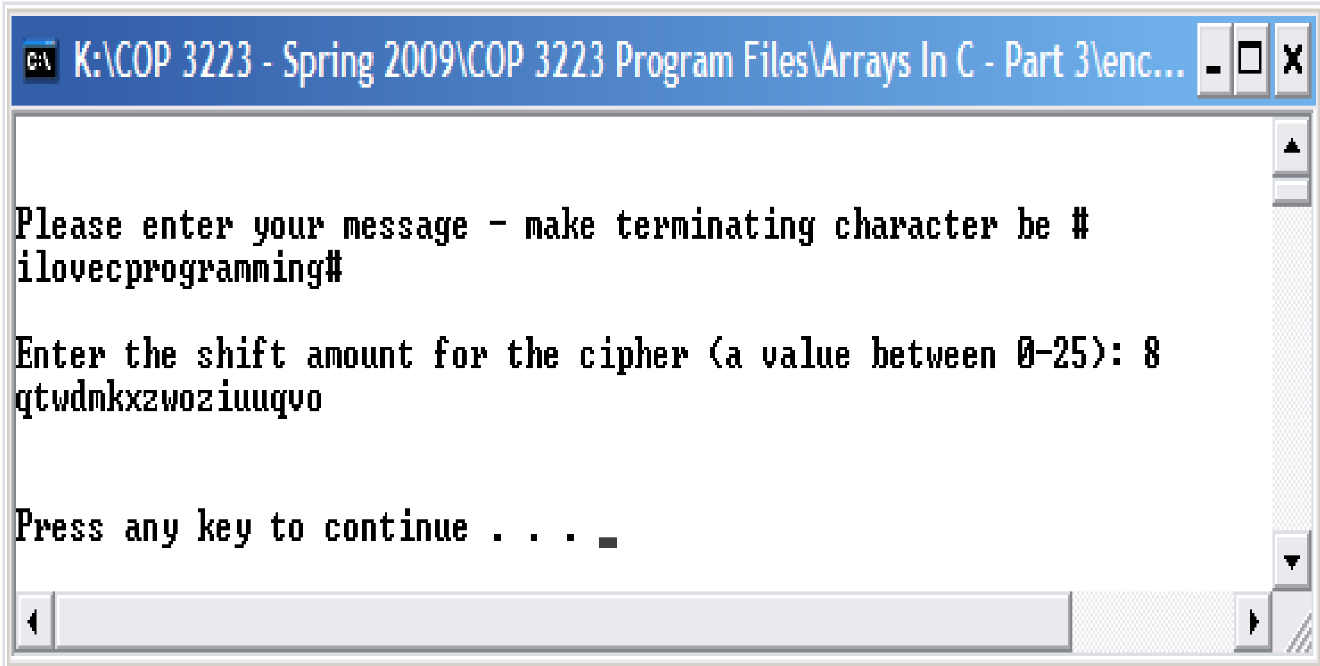
```c
        //END TESTING CODE
        //Get user message and encode it - write it to a file named "coded message"
        printf("Please enter your message - make terminating character be #\n");
        charCounter = 0;
        do {
           scanf("%c", &clearMessage[charCounter]);
           charCounter++;
        } while ( clearMessage[charCounter-1] != '#');
        charCounter--;// decrement counter - its 1 too large
        printf("\nEnter the shift amount for the cipher (a value between 0-25): ");
        scanf("%d", &shiftAmount);
        //encode message
        for ( i = 0; i < charCounter; ++i) {
            searchChar = clearMessage[i];
            found = 0; //haven't found the character yet
            clearPosition = 0;
            while (!found) {
                if (searchChar == codes[0][clearPosition]) {
                    found = 1; //found the character
                    codedMessage[i] = codes[shiftAmount][clearPosition];
                }//end if stmt
                clearPosition++;
            }//end while stmt
        }//end for stmt
        for (i = 0; i < charCounter; ++i) {
            printf("%c", codedMessage[i]);
        }//end for stmt
```

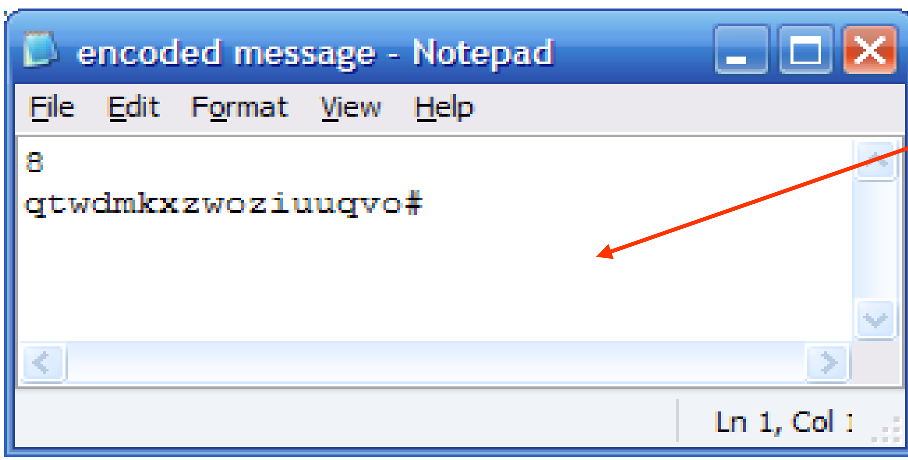
[*] encoder.c

```c



        //generate output file for decoder program
        if ( (outFilePtr = fopen("encoded message.dat", "w")) == NULL ) {
         printf("Sorry, could'nt create output file\n");
        }
        else {
            fprintf(outFilePtr, "%d\n", shiftAmount);
            for (i = 0; i < charCounter; ++i) {
                fprintf(outFilePtr, "%c", codedMessage[i]);
            }//end for stmt
            fprintf(outFilePtr, "#");
            fclose(outFilePtr);
        }//end else stmt
        printf("\n");
        fclose(inFilePtr);
    }//end else fopen block
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```



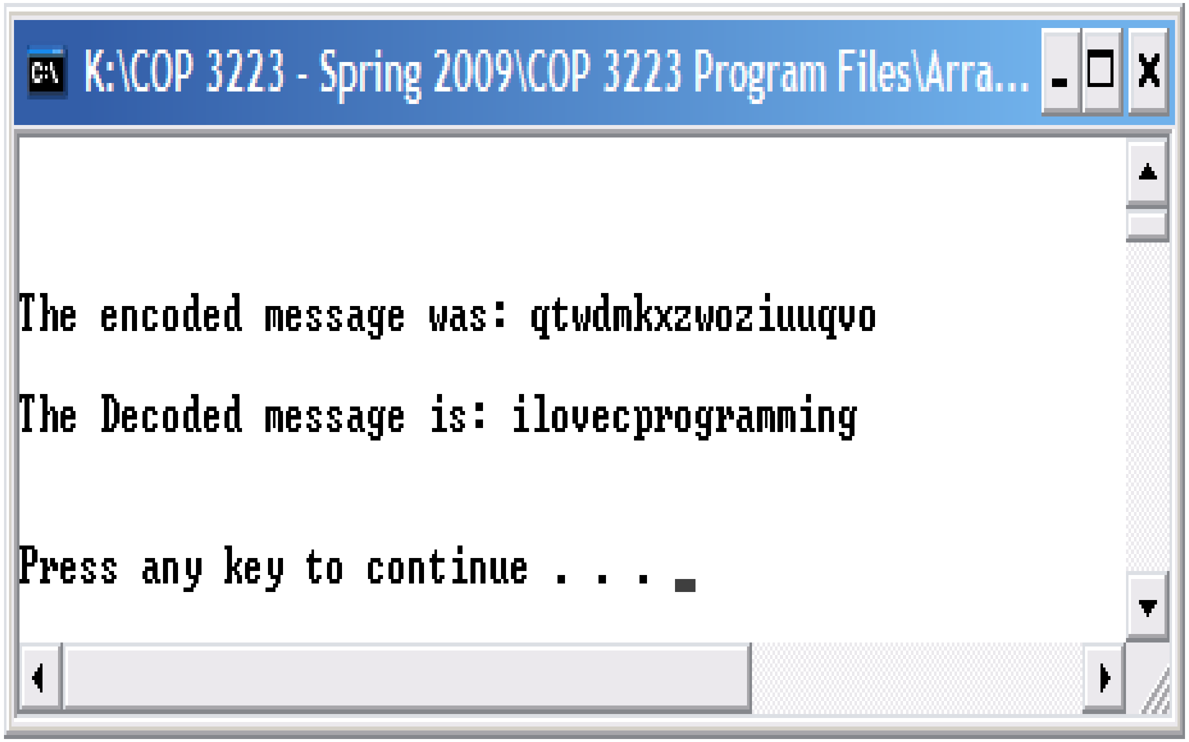


The output file "`encoded message.dat`"

encoder.c [*] decoder.c

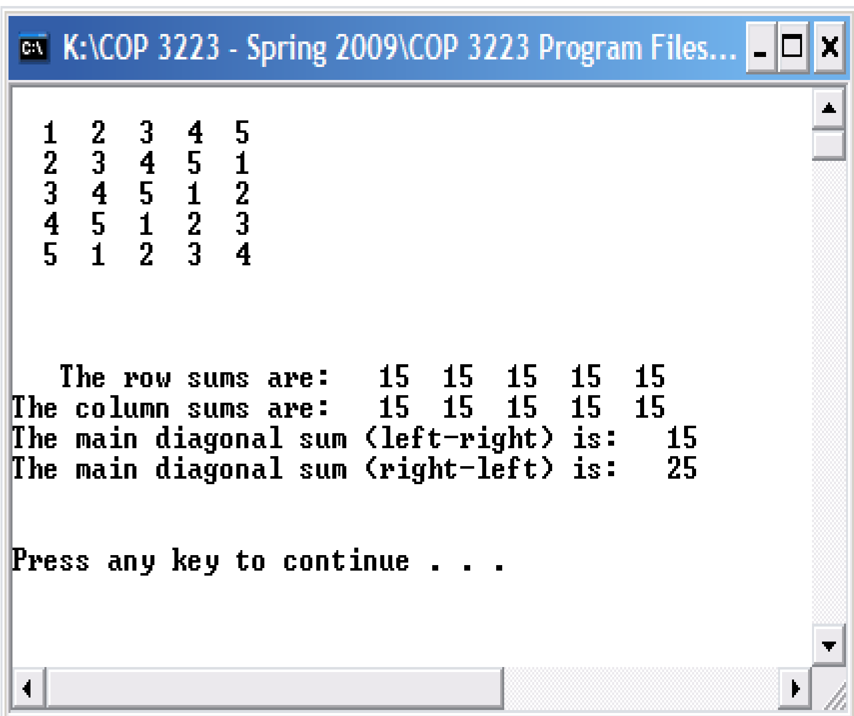
```
        //decode message
        if ( (inFilePtr2 = fopen("encoded message.dat", "r")) == NULL ) {
         printf("Sorry, could'nt open the input file\n");
        }
        else {
             fscanf(inFilePtr2, "%d\n", &shiftAmount);
             charCounter = 0;
             do {
                fscanf(inFilePtr2, "%c", &codedMessage[charCounter]);
                charCounter++;
              } while ( codedMessage[charCounter-1] != '#');
              charCounter--;// decrement counter - its 1 too large for
              printf("\nThe encoded message was: ");
              for (i = 0; i < charCounter; ++i) {
                   printf("%c", codedMessage[i]);
              }//end for stmt
              printf("\n\nThe Decoded message is: ");
              for ( i = 0; i < charCounter; ++i) {
                 searchChar = codedMessage[i];
                 found = 0; //haven't found the character yet
                 codedPosition = 0;
                 while (!found) {
                    if (searchChar == codes[shiftAmount][codedPosition]) {
                        found = 1; //found the character
                        decodedMessage[i] = codes[0][codedPosition];
                    }//end if stmt
                    codedPosition++;
```

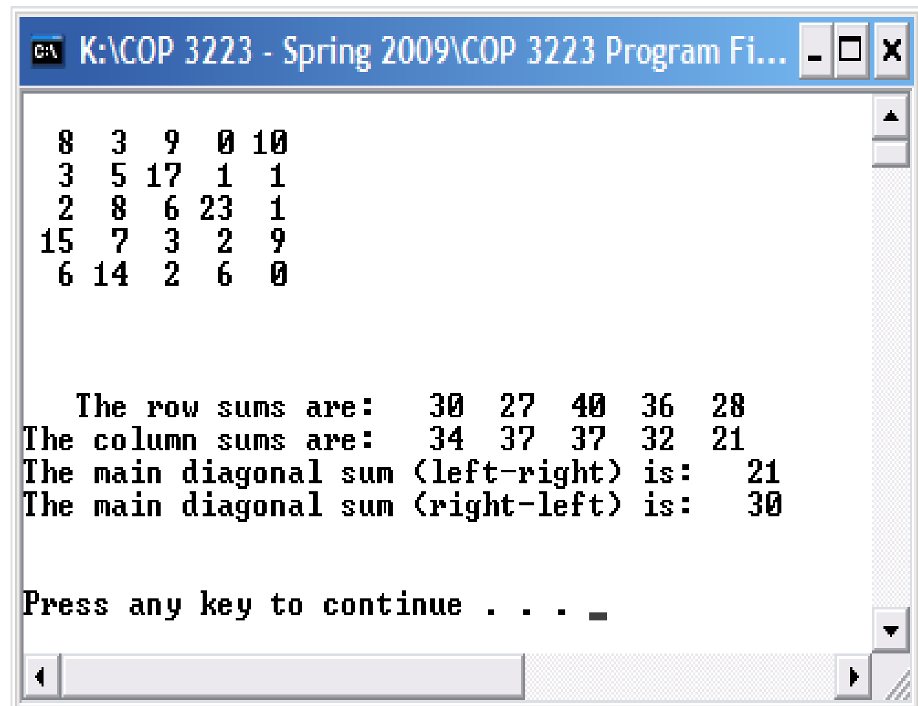
Output from the decoder program



Practice Problems

1. Modify the example that begins on page 3 that computes the row and column sums for any matrix so that in addition to row and column sums, it will also produce the sum along the two main diagonals if the array represents a square matrix. If the matrix is not square nothing different happens.





Practice Problems

2. Modify the magic square example from pages 8 & 9 so that writes to an output file named "`magic square.dat`", the row and column dimensions of the matrix as well as the matrix itself. Then, using this file as the input file to the program you just wrote to solve Practice Problem 1, show that every row, column, and main diagonal sum is the same.

   Note that all you should need to do to your solution to Practice Problem 1 is to change the file name that the program reads.

K:\COP 3223 - Spring 2009\COP 3223 Program...

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



   The row sums are:    65   65   65   65   65
The column sums are:    65   65   65   65   65
The main diagonal sum (left-right) is:   65
The main diagonal sum (right-left) is:   65


Press any key to continue . . .
```

Practice Problems

3. Modify the matrix addition example found in Arrays In C – part 2 lecture notes, to multiply two compatible matrices together.

K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - Part 3\matrix multiplicat...

```
Enter the number of rows and columns for matrix A [max value for each is 10]:
2 3
Enter the number of rows and columns for matrix B [max value for each is 10]:
3 2
Enter Matrix A[0][0] value:
4
Enter Matrix A[0][1] value:
3
Enter Matrix A[0][2] value:
-2

Enter Matrix A[1][0] value:
1
Enter Matrix A[1][1] value:
-3
Enter Matrix A[1][2] value:
2


  4  3 -2
  1 -3  2

Enter Matrix B[0][0] value:
1
Enter Matrix B[0][1] value:
-1
Enter Matrix B[1][0] value:
2
Enter Matrix B[1][1] value:
4
Enter Matrix B[2][0] value:
3
Enter Matrix B[2][1] value:
2


  1 -1
  2  4
  3  2
The summation matrix is:
  4  4
  1 -9
```