COP 3223: C Programming
Spring 2009

Arrays In C – Part 1

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

Arrays In C

- Arrays are an elementary data structure found in most modern programming languages. Arrays have a vast number of uses in programming applications.
- Up to this point in the semester, the variables that we have used in all of our programs have been scalar variables. Scalar variables are capable of holding one data item (value) at a time.
- Arrays fall into the category of aggregate variables. Aggregate variables are capable of holding a collection of variable values at one time.
- Arrays are static, meaning that the size of the collection of variable values that they are capable of holding remains fixed throughout program execution.

Arrays In C

- As with normal scalar variables, every array variable must be declared to have some type.
- Every item in an array, must be of the same type. Thus, the same array cannot contain both integers and real numbers.
- The general form for declaring an array variable in C is:

  ```
  type  <var_name> [size];
  ```

  where `type` is any C type, `var_name` is any legal variable name in C, and `size` is any expression that evaluates to an integer.

Arrays In C

- Some example array declarations:

```
//defines an array of 100 integers
int  anArray[100];

//defines an array of 30 doubles
double newValues[30];

//defines an array of 25 integers
int a = 5, b = 5;
int myNumbers[a * b];
```

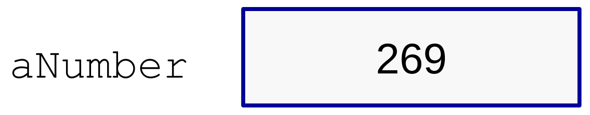
Arrays In C

- Think of an array as a group of memory locations related by the fact that they all have the same name and the same type.
- To refer to a particular location (i.e., variable) or element of the array, you need to specify the name of the array and the position number of the particular element within the array.
- In C, the first element of an array is considered to be in position 0, the second element of the array is considered to be in position 1.
- Thus if we define `int myArray[6];` the array contains six positions numbered 0, 1, 2, 3, 4, and 5.

Arrays In C

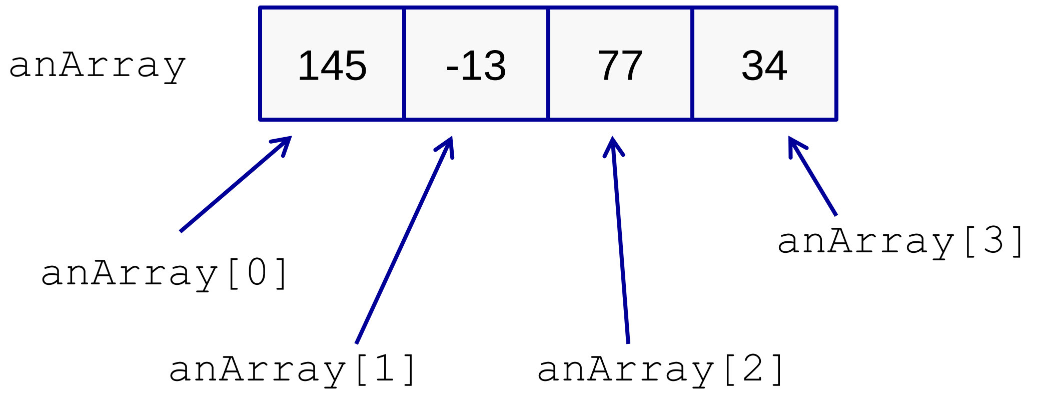
A scalar variable

```
int   aNumber;
```



An aggregate variable

```
int   anArray[4];
```



Arrays In C

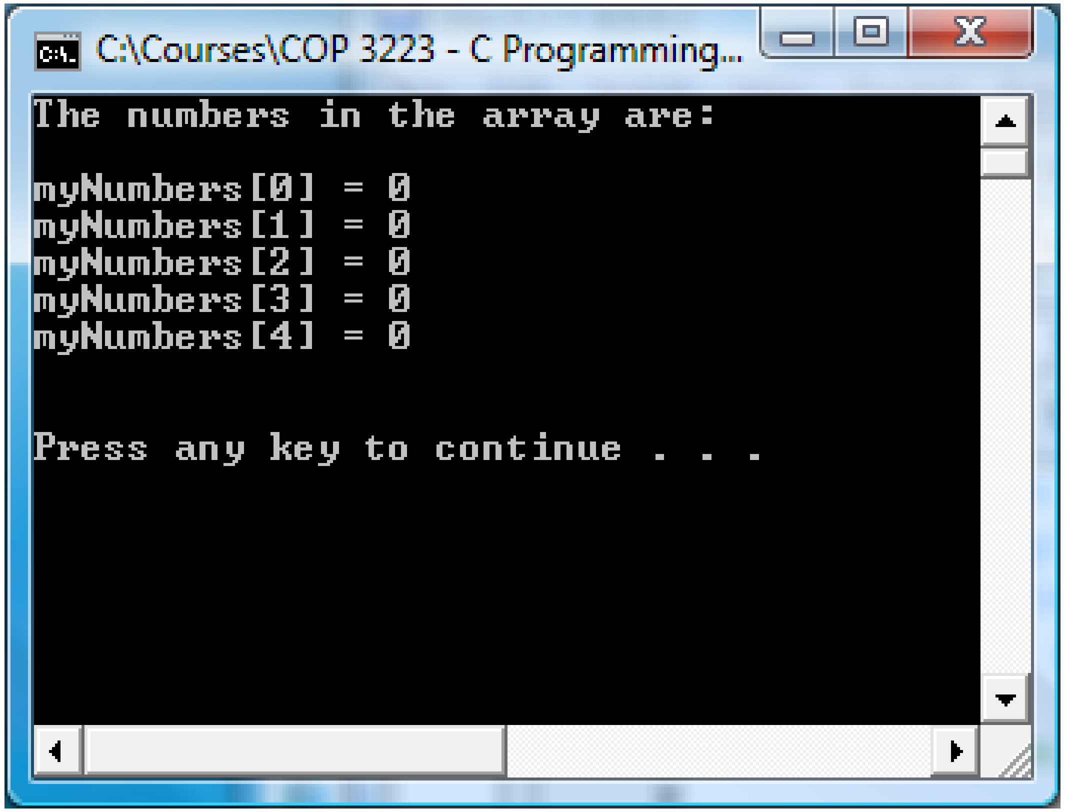
- The position number of an array element is more formally referred to as an array subscript or an array index.
- Thus, to refer to the first element of an array named `myArray` you would use the notation `myArray[0]`.
- In general, the `i`th element of an array is in the position denoted by `[i-1]`.
- In C, an array index must be an integer or any expression that evaluates to an integer.

COMMON PROGRAMMING ERROR

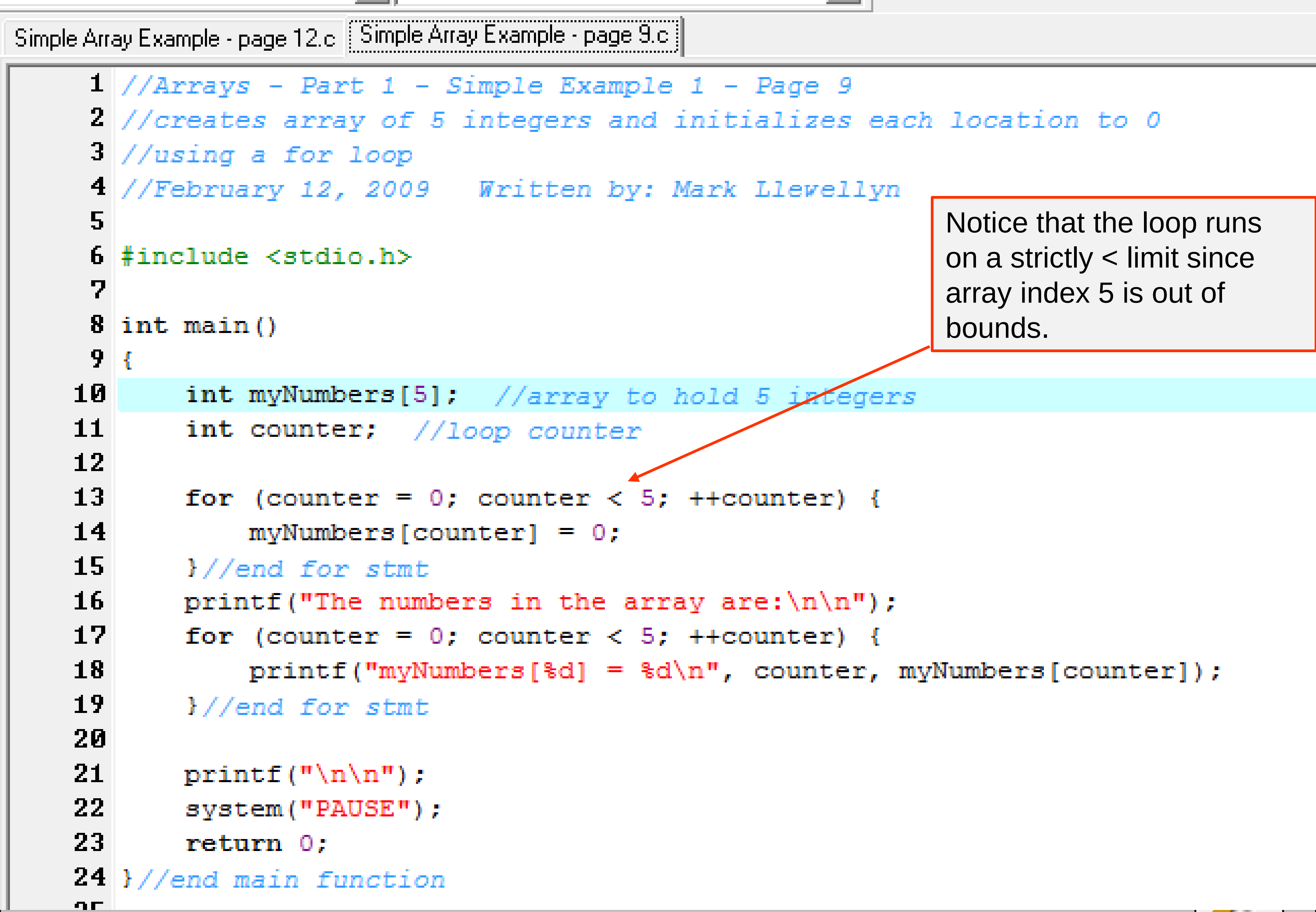
It is a common mistake to make when referring to array elements to say something like, the sixth element of an array named `myArray` is `myArray[6]`, when in fact the sixth element of the array would be `myArray[5]`.

Arrays In C

- Let’s create a simple program that creates and array of five integers and then uses a simple loop to initialize the locations in the array to zero.

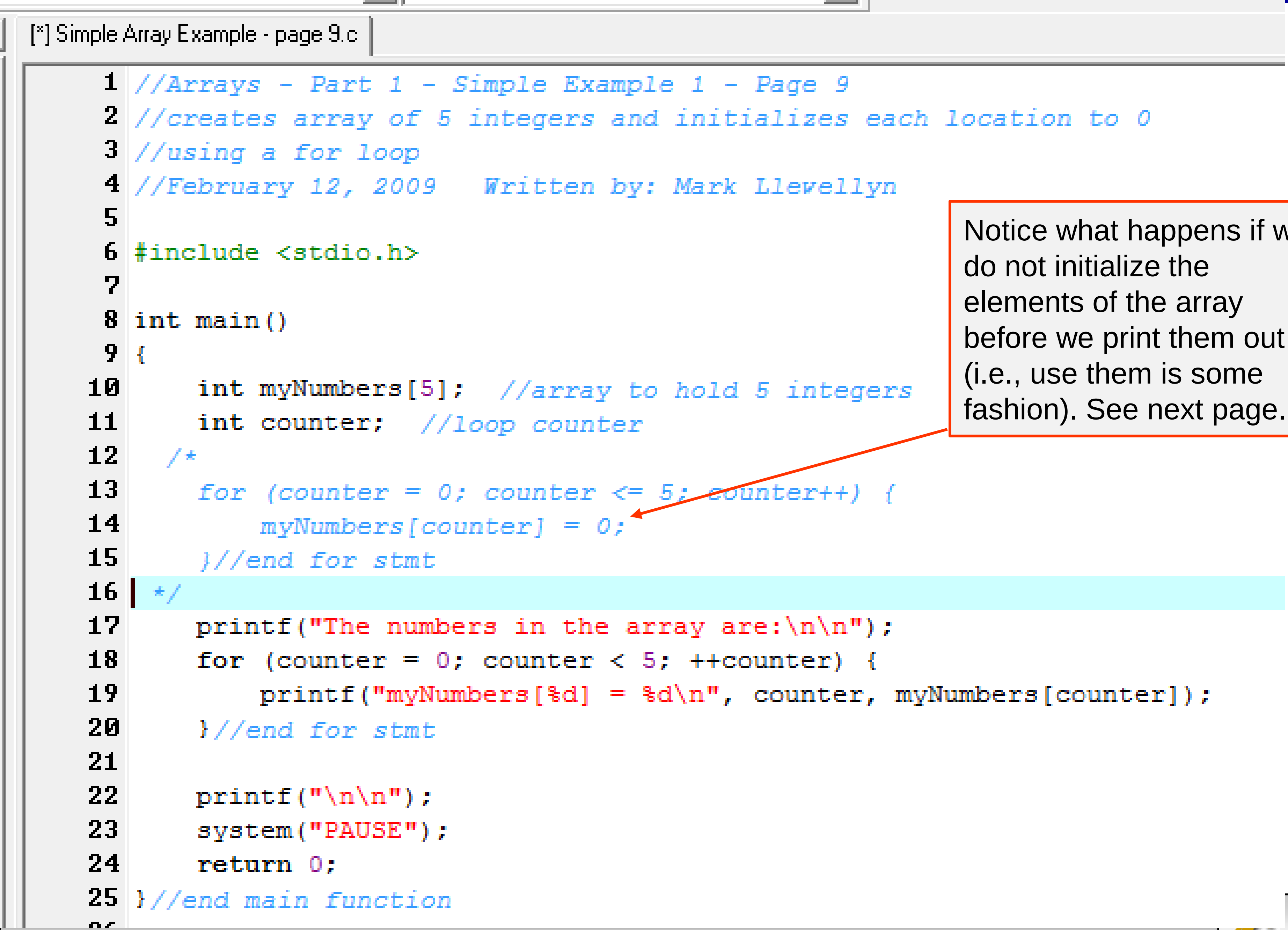


```
The numbers in the array are:

myNumbers[0] = 0
myNumbers[1] = 0
myNumbers[2] = 0
myNumbers[3] = 0
myNumbers[4] = 0


Press any key to continue . . .
```


Simple Array Example - page 12.c
Simple Array Example - page 9.c

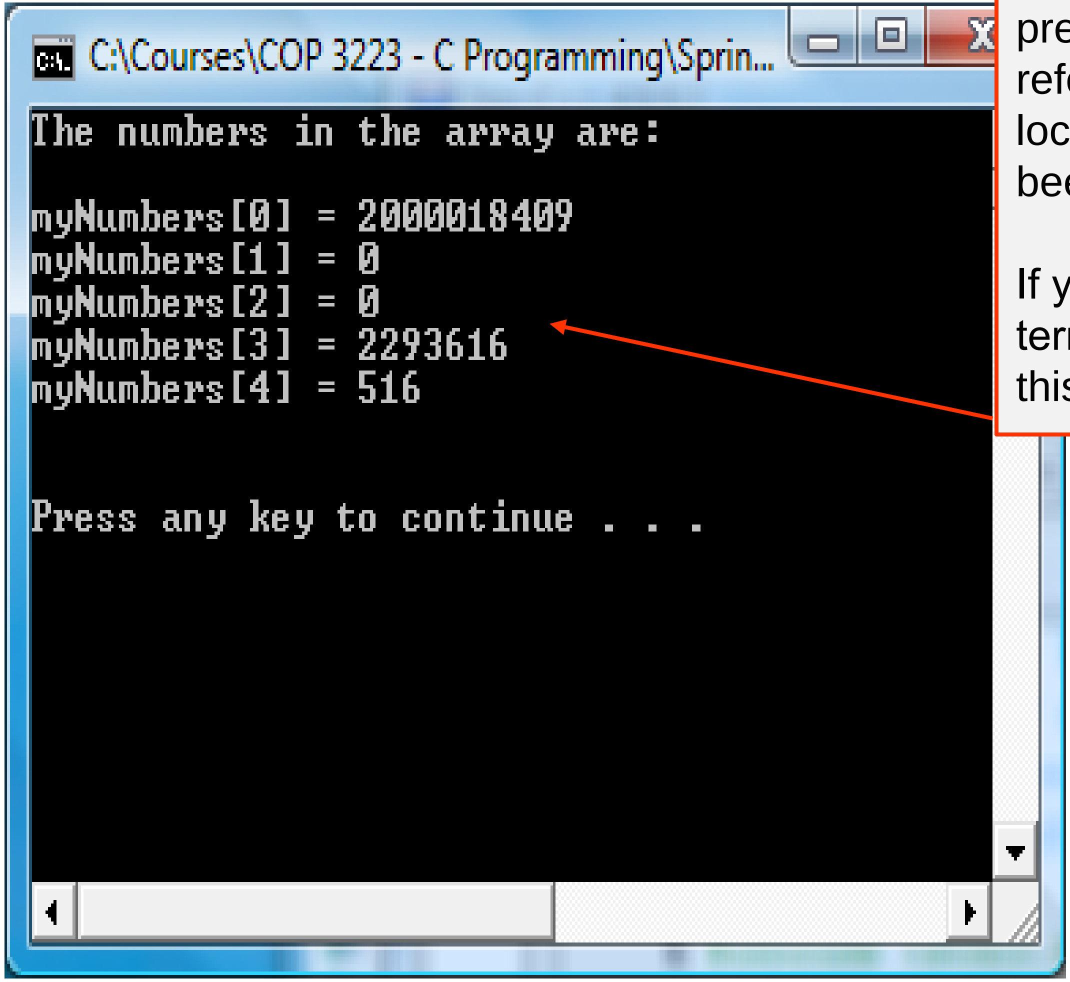
```
//Arrays - Part 1 - Simple Example 1 - Page 9
//creates array of 5 integers and initializes each location to 0
//using a for loop
//February 12, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int myNumbers[5];  //array to hold 5 integers
    int counter;  //loop counter

    for (counter = 0; counter < 5; ++counter) {
        myNumbers[counter] = 0;
    }//end for stmt
    printf("The numbers in the array are:\n\n");
    for (counter = 0; counter < 5; ++counter) {
        printf("myNumbers[%d] = %d\n", counter, myNumbers[counter]);
    }//end for stmt

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

Notice that the loop runs on a strictly < limit since array index 5 is out of bounds.



```c
//Arrays - Part 1 - Simple Example 1 - Page 9
//creates array of 5 integers and initializes each location to 0
//using a for loop
//February 12, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int myNumbers[5];  //array to hold 5 integers
    int counter;  //loop counter
  /*
    for (counter = 0; counter <= 5; counter++) {
        myNumbers[counter] = 0;
    }//end for stmt
 */
    printf("The numbers in the array are:\n\n");
    for (counter = 0; counter < 5; ++counter) {
        printf("myNumbers[%d] = %d\n", counter, myNumbers[counter]);
    }//end for stmt

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

Notice what happens if we do not initialize the elements of the array before we print them out (i.e., use them is some fashion). See next page.



```
The numbers in the array are:

myNumbers[0] = 2000018409
myNumbers[1] = 0
myNumbers[2] = 0
myNumbers[3] = 2293616
myNumbers[4] = 516


Press any key to continue . . .
```

The contents of the array elements are impossible to predict, since they are referring to memory locations that have not been set by our program.

If you've ever heard the term "computer garbage", this is it!

Arrays In C

- There is also a shorthand technique to initialize the elements of an array when the array is declared. The technique is similar in nature to initializing a scalar variable.
- Suppose we want all of an arrays elements to be initially set to 0.

```
int myArray[10] = {0};
```

- The initial value for each element is placed in braces following the assignment operator.
- In the case above all 10 elements of the array will be set to 0, since there are fewer initializers than there are locations in the array.

Arrays In C

```
int myNumbers[4] = {1,2,3,4};
```

For cases where there is more than one initializer, the values are comma separated.

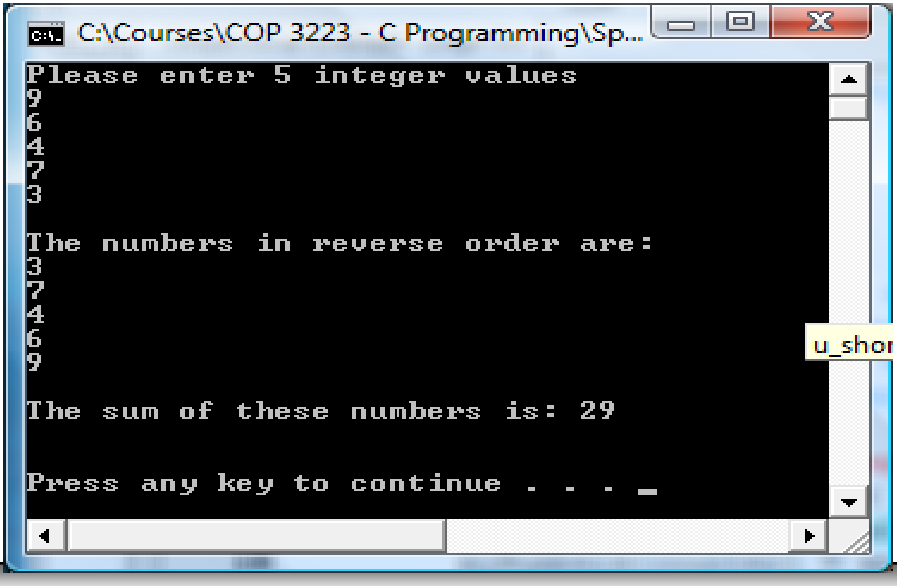
- Sets the initial values in the array as:

```
myNumbers[0] = 1
myNumbers[1] = 2
myNumbers[2] = 3
myNumbers[3] = 4
```

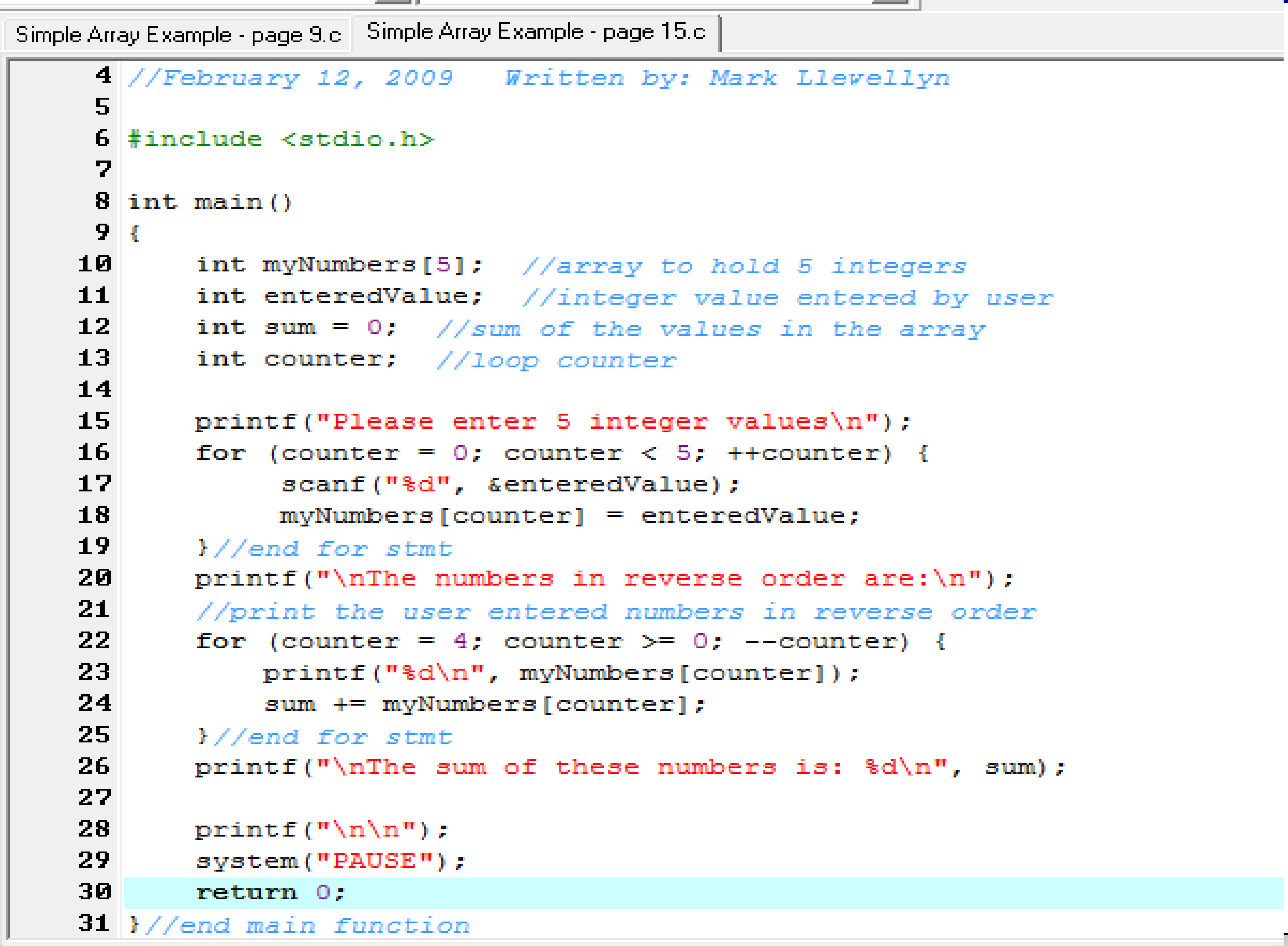
- It is a syntax error to provide more initializers than there are locations in the array. For example, above it we had typed `int myNumbers[4] = {1,2,3,4,5};` the compiler would have issued a syntax error.

Arrays In C

- Let’s create a simple program and creates and array of five integers.
- It will ask the user to enter five integer values that we will store in the array and then print out each integer the user entered in the reverse order that they entered the values, and then will produce the sum of those integers.



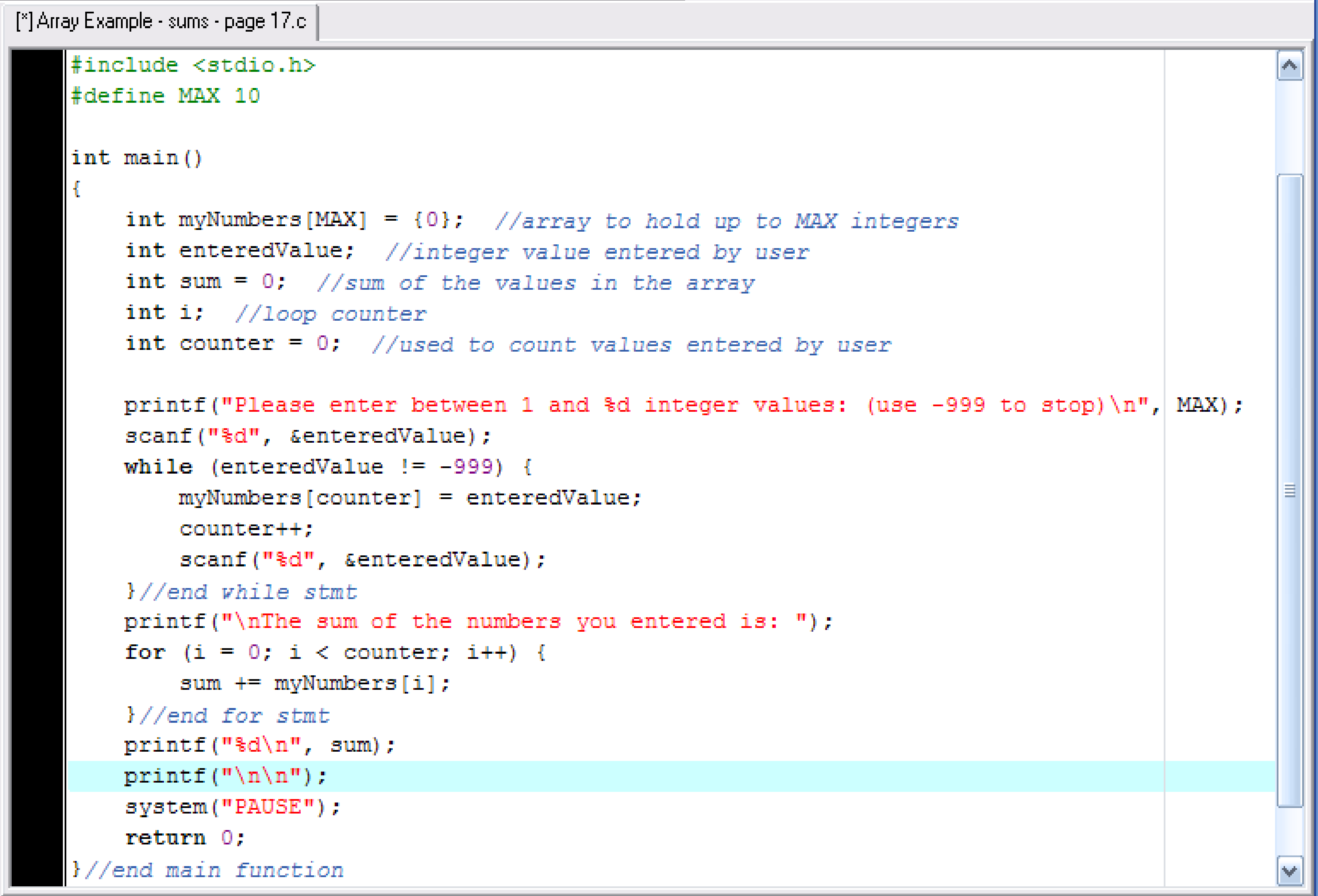
```
Please enter 5 integer values
9
6
4
7
3

The numbers in reverse order are:
3
7
4
6
9

The sum of these numbers is: 29

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


Simple Array Example - page 9.c
Simple Array Example - page 15.c

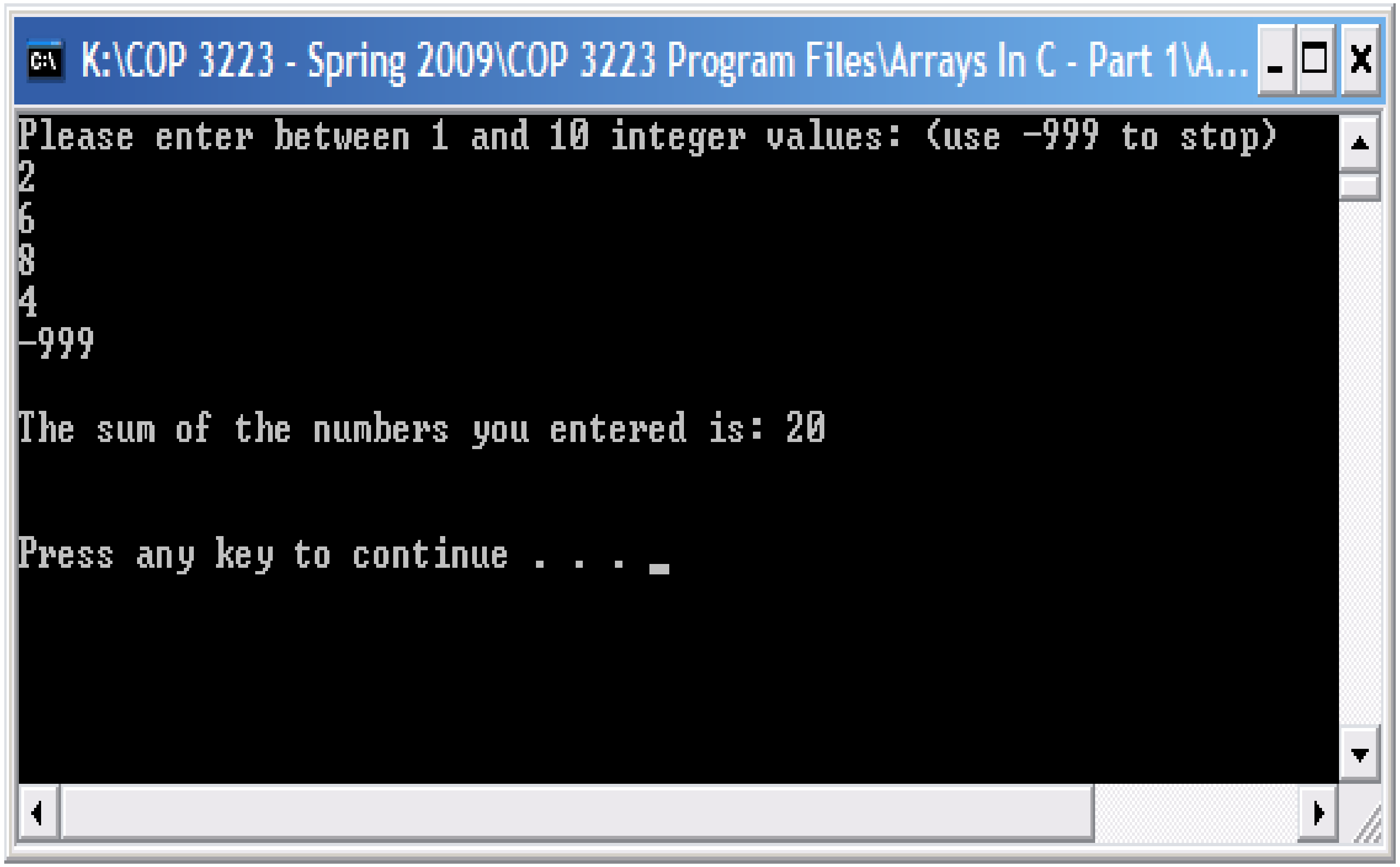
```c
//February 12, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int myNumbers[5];   //array to hold 5 integers
    int enteredValue;   //integer value entered by user
    int sum = 0;  //sum of the values in the array
    int counter;  //loop counter

    printf("Please enter 5 integer values\n");
    for (counter = 0; counter < 5; ++counter) {
         scanf("%d", &enteredValue);
         myNumbers[counter] = enteredValue;
    }//end for stmt
    printf("\nThe numbers in reverse order are:\n");
    //print the user entered numbers in reverse order
    for (counter = 4; counter >= 0; --counter) {
        printf("%d\n", myNumbers[counter]);
        sum += myNumbers[counter];
    }//end for stmt
    printf("\nThe sum of these numbers is: %d\n", sum);

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

Arrays In C

- It is also possible to create an array in C whose size is determined by a symbolic constant.
- For example, if we define `MAX` as `#define MAX 10` and then we define an array as `int myArray[MAX];`, this will define an array of 10 integer elements.
- Using symbolic constants in this fashion makes your program more scalable, since we can now change the size of the array by simply changing the value assigned to the constant `MAX`.
- The program on the next page, creates an array whose size is determined by a constant, then the user is asked to enter values into the array which are later summed.



```c
#include <stdio.h>
#define MAX 10

int main()
{
    int myNumbers[MAX] = {0};  //array to hold up to MAX integers
    int enteredValue;  //integer value entered by user
    int sum = 0;  //sum of the values in the array
    int i;  //loop counter
    int counter = 0;  //used to count values entered by user

    printf("Please enter between 1 and %d integer values: (use -999 to stop)\n", MAX);
    scanf("%d", &enteredValue);
    while (enteredValue != -999) {
        myNumbers[counter] = enteredValue;
        counter++;
        scanf("%d", &enteredValue);
    }//end while stmt
    printf("\nThe sum of the numbers you entered is: ");
    for (i = 0; i < counter; i++) {
        sum += myNumbers[i];
    }//end for stmt
    printf("%d\n", sum);
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```


K:\COP 3223 - Spring 2009\COP 3223 Program Files\Arrays In C - Part 1\A...

```
Please enter between 1 and 10 integer values: (use -999 to stop)
2
6
8
4
-999

The sum of the numbers you entered is: 20


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

Small Case Study – How Arrays Are Useful

- We seen several examples of creating and using arrays so far. We've seen how they are used to represent data and how we can keep related data "together", but can they also save you time and effort when writing code?
- To answer this question we will write a C program to solve the following problem using two different approaches, one without using arrays and one that utilizes arrays; and we'll compare the differences in the code.
- <u>The problem:</u> We want to simulate a user rolling a normal six-sided die 12,000 times. Each value should appear approximately 2,000 times. We want to record the total number of times each of the six possible values is rolled.

Small Case Study – How Arrays Are Useful

- Before we continue with the case study, let’s take a brief aside and look at random number generation in C.
- Most random number generation in C is done with the `rand()` function found in `<math.h>.` This function returns a random number between 0 and `RAND_MAX`, where `RAND_MAX` is a symbolic constant defined to be the value of the maximum integer on the machine (i.e., typically $2^{31} – 1$, or 32,767).
- For simulation purposes programs often need to generate random numbers within a specific range of values. For example to simulate rolling a die, we need only 6 random values between 1 and 6.

Small Case Study – How Arrays Are Useful

- To generate random numbers between 0 and `RAND_MAX` (not inclusive by the way), you would simply call `rand()` as follows:

```
int x = rand();
```

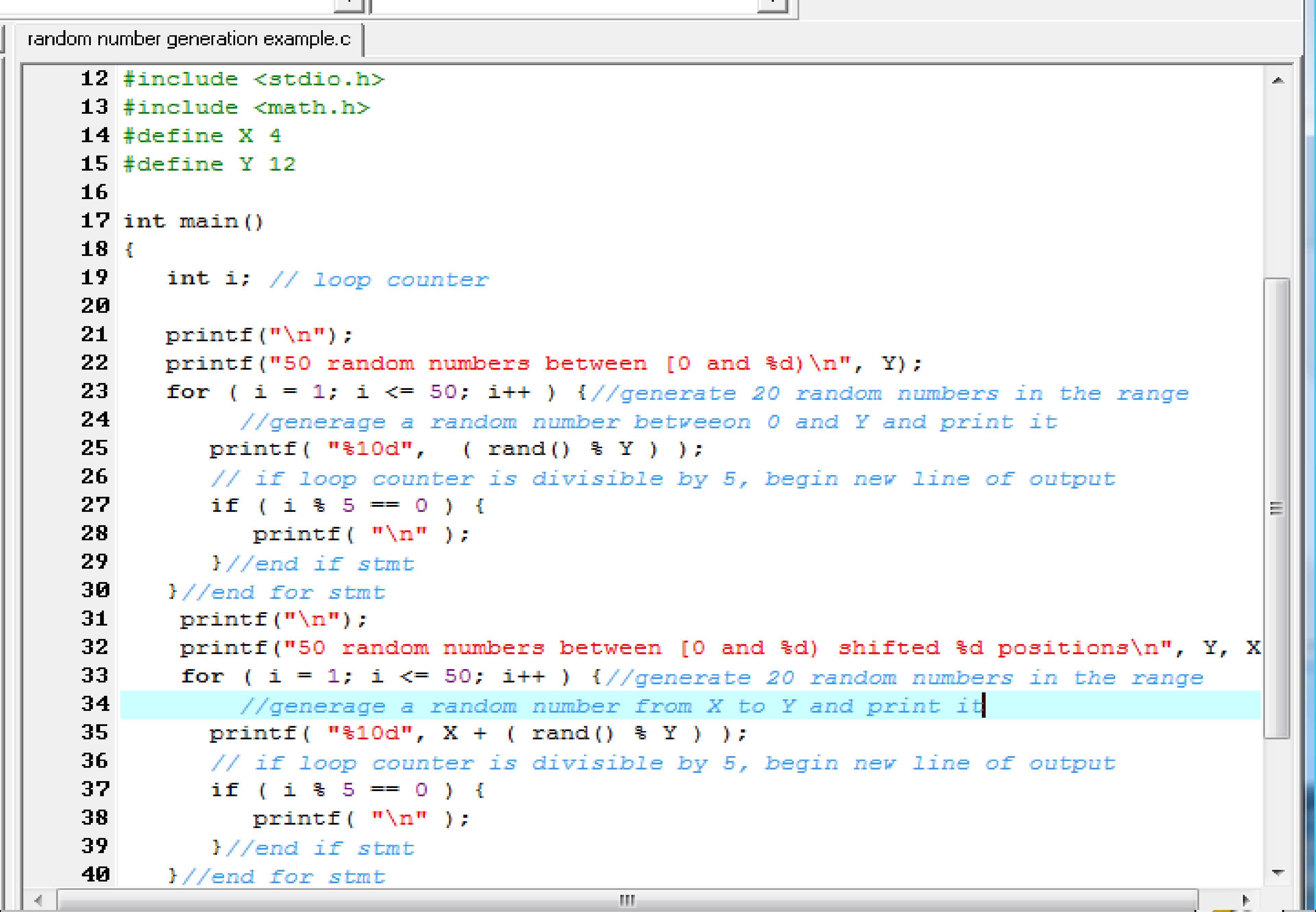
- To generate random numbers between 0 and 50. Where 50 is called the scaling factor, you would simply call `rand()` as follows:

```
int x = rand() % 50;
```

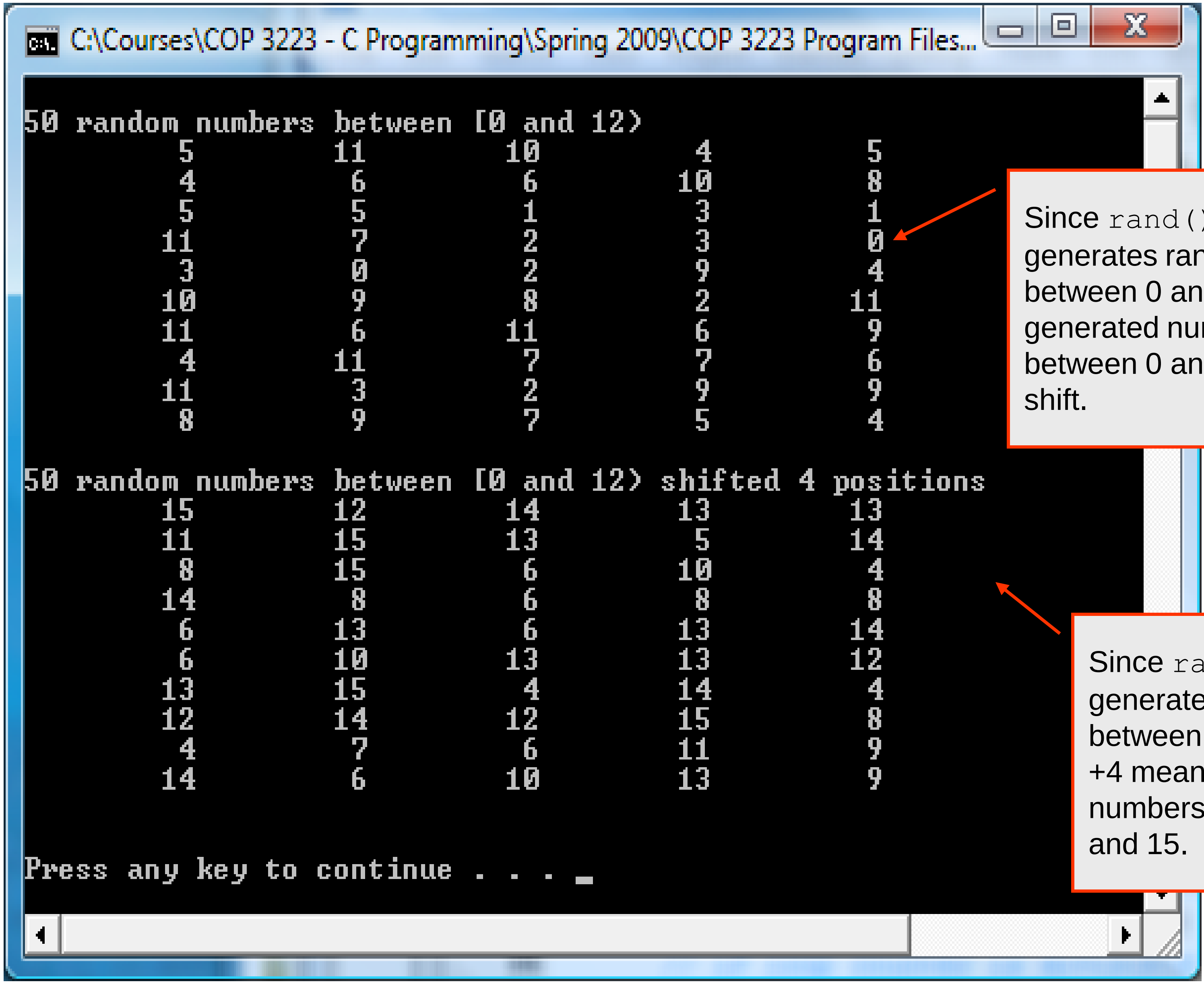
- To generate random numbers between 1 and 50, you need both a scaling factor of 50 and a shift factor of 1. You would then call `rand()` as follows:

```
int x = 1 + rand() % 50;
```

- The program on the next page illustrates calling the `rand()` function.



```
#include <stdio.h>
#include <math.h>
#define X 4
#define Y 12

int main()
{
   int i; // loop counter

   printf("\n");
   printf("50 random numbers between [0 and %d)\n", Y);
   for ( i = 1; i <= 50; i++ ) {//generate 20 random numbers in the range
        //generage a random number betweeon 0 and Y and print it
      printf( "%10d",  ( rand() % Y ) );
      // if loop counter is divisible by 5, begin new line of output
      if ( i % 5 == 0 ) {
         printf( "\n" );
      }//end if stmt
   }//end for stmt
    printf("\n");
    printf("50 random numbers between [0 and %d) shifted %d positions\n", Y, X
    for ( i = 1; i <= 50; i++ ) {//generate 20 random numbers in the range
        //generage a random number from X to Y and print it
      printf( "%10d", X + ( rand() % Y ) );
      // if loop counter is divisible by 5, begin new line of output
      if ( i % 5 == 0 ) {
         printf( "\n" );
      }//end if stmt
   }//end for stmt
```


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files...

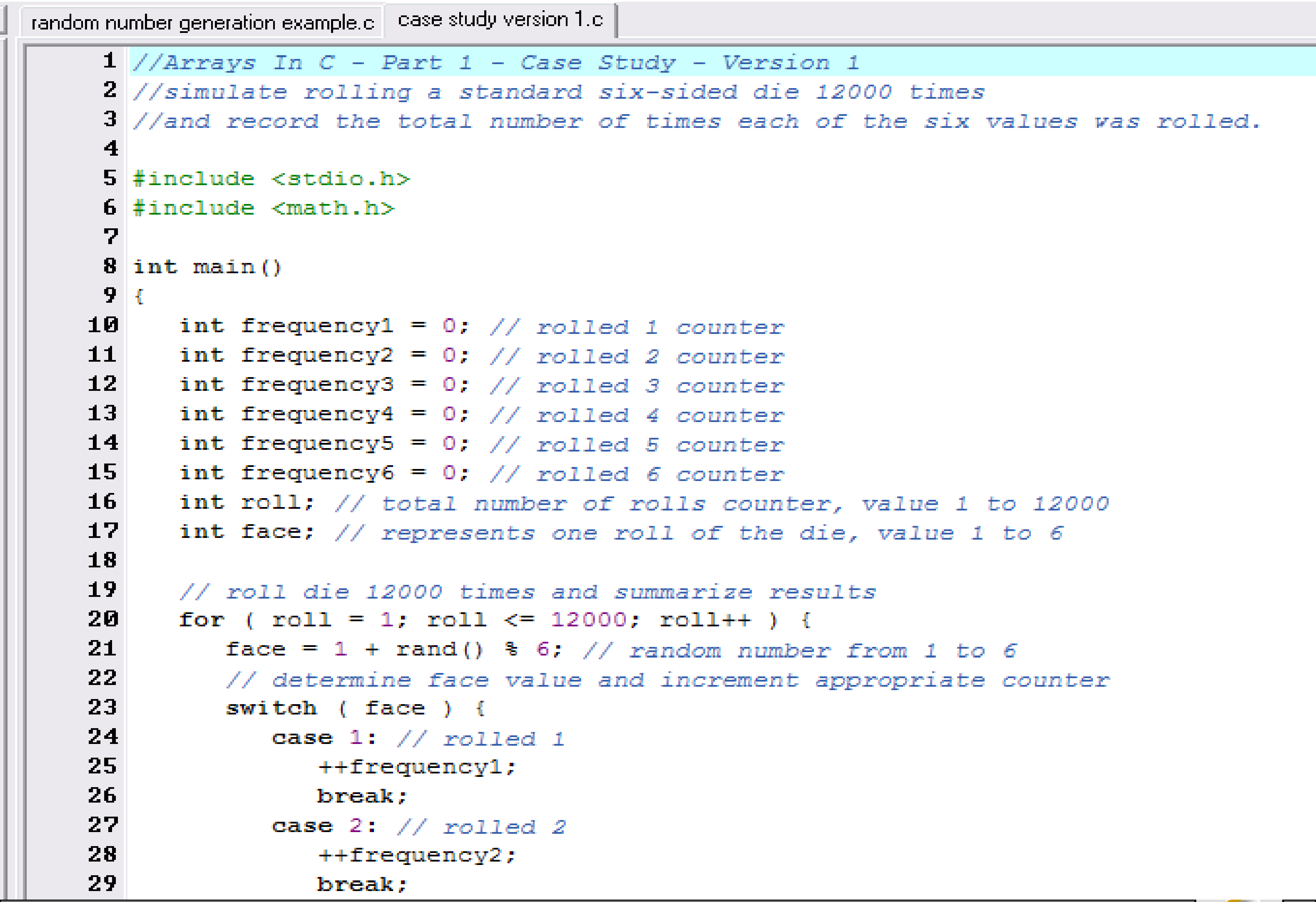
```
50 random numbers between [0 and 12)
         5         11         10          4          5
         4          6          6         10          8
         5          5          1          3          1
        11          7          2          3          0
         3          0          2          9          4
        10          9          8          2         11
        11          6         11          6          9
         4         11          7          7          6
        11          3          2          9          9
         8          9          7          5          4

50 random numbers between [0 and 12) shifted 4 positions
        15         12         14         13         13
        11         15         13          5         14
         8         15          6         10          4
        14          8          6          8          8
         6         13          6         13         14
         6         10         13         13         12
        13         15          4         14          4
        12         14         12         15          8
         4          7          6         11          9
        14          6         10         13          9

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

Since `rand() % 12` generates random numbers between 0 and 11, the generated numbers will be between 0 and 11 with no shift.

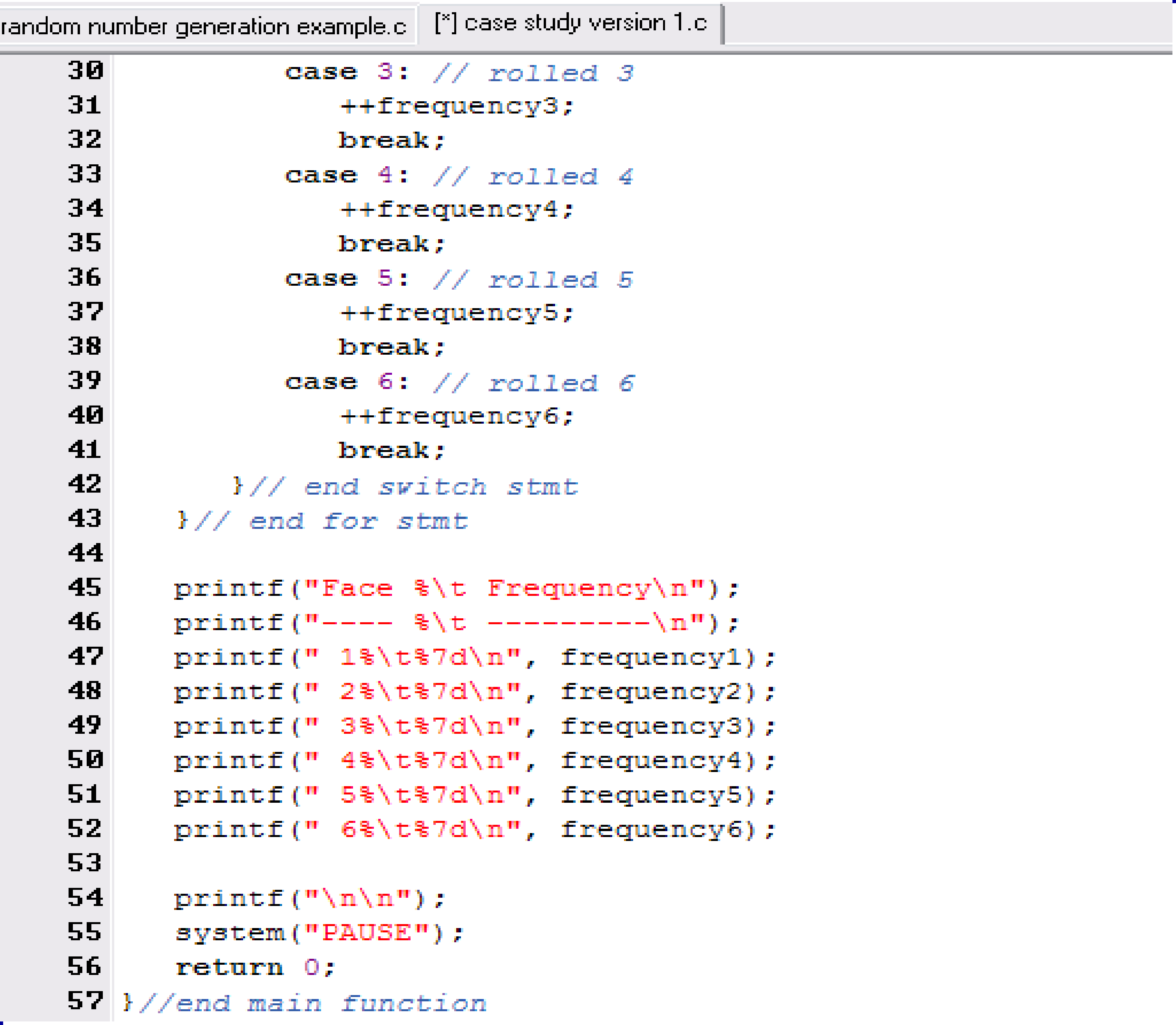
Since `rand() % 12` generates random numbers between 0 and 11, the shift of +4 means the generated numbers will be between 4 and 15.

A Small Case Study - continues

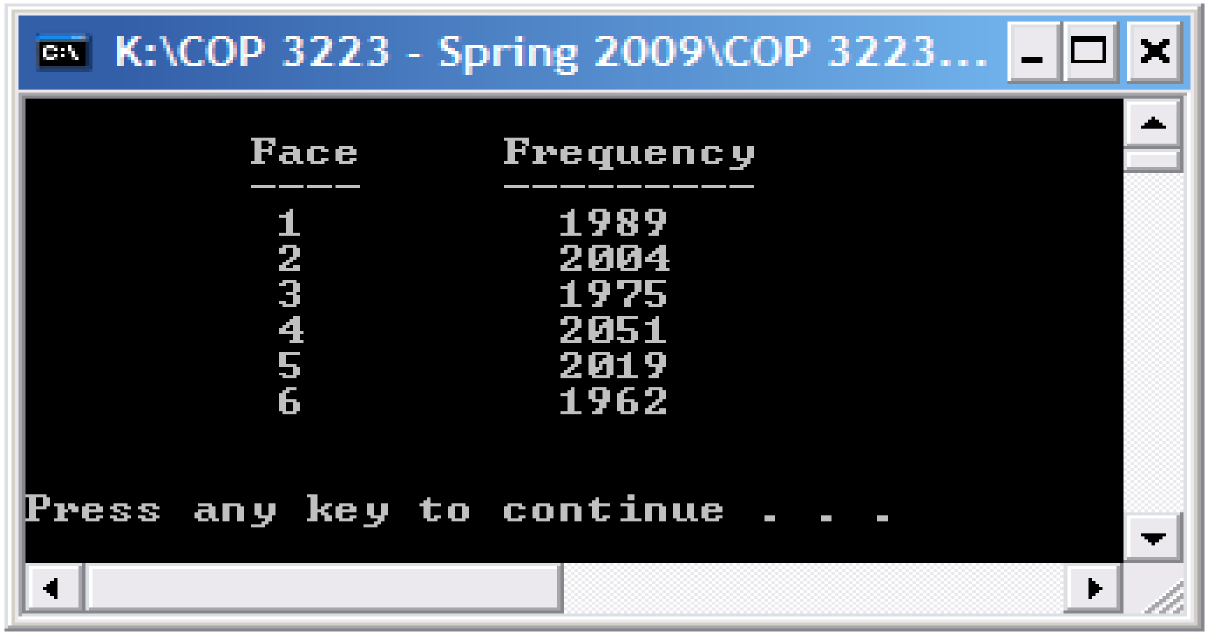
- Now back to the case study… the first version of the code we'll write to solve the problem of rolling a standard die 12,000 times and seeing what the frequency of each value rolled, we'll do without using arrays.
- This will mean that we'll need a separate variable for each of the six possible values that could be rolled.
- The program on the next two pages (it wouldn't fit on one page) illustrates a solution to this problem without the use of arrays


random number generation example.c | case study version 1.c

```c
//Arrays In C - Part 1 - Case Study - Version 1
//simulate rolling a standard six-sided die 12000 times
//and record the total number of times each of the six values was rolled.

#include <stdio.h>
#include <math.h>

int main()
{
   int frequency1 = 0; // rolled 1 counter
   int frequency2 = 0; // rolled 2 counter
   int frequency3 = 0; // rolled 3 counter
   int frequency4 = 0; // rolled 4 counter
   int frequency5 = 0; // rolled 5 counter
   int frequency6 = 0; // rolled 6 counter
   int roll; // total number of rolls counter, value 1 to 12000
   int face; // represents one roll of the die, value 1 to 6

   // roll die 12000 times and summarize results
   for ( roll = 1; roll <= 12000; roll++ ) {
      face = 1 + rand() % 6; // random number from 1 to 6
      // determine face value and increment appropriate counter
      switch ( face ) {
         case 1: // rolled 1
            ++frequency1;
            break;
         case 2: // rolled 2
            ++frequency2;
            break;
```


random number generation example.c
[*] case study version 1.c

```
         case 3: // rolled 3
            ++frequency3;
            break;
         case 4: // rolled 4
            ++frequency4;
            break;
         case 5: // rolled 5
            ++frequency5;
            break;
         case 6: // rolled 6
            ++frequency6;
            break;
      }// end switch stmt
   }// end for stmt

   printf("Face %\t Frequency\n");
   printf("---- %\t ---------\n");
   printf(" 1%\t%7d\n", frequency1);
   printf(" 2%\t%7d\n", frequency2);
   printf(" 3%\t%7d\n", frequency3);
   printf(" 4%\t%7d\n", frequency4);
   printf(" 5%\t%7d\n", frequency5);
   printf(" 6%\t%7d\n", frequency6);

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

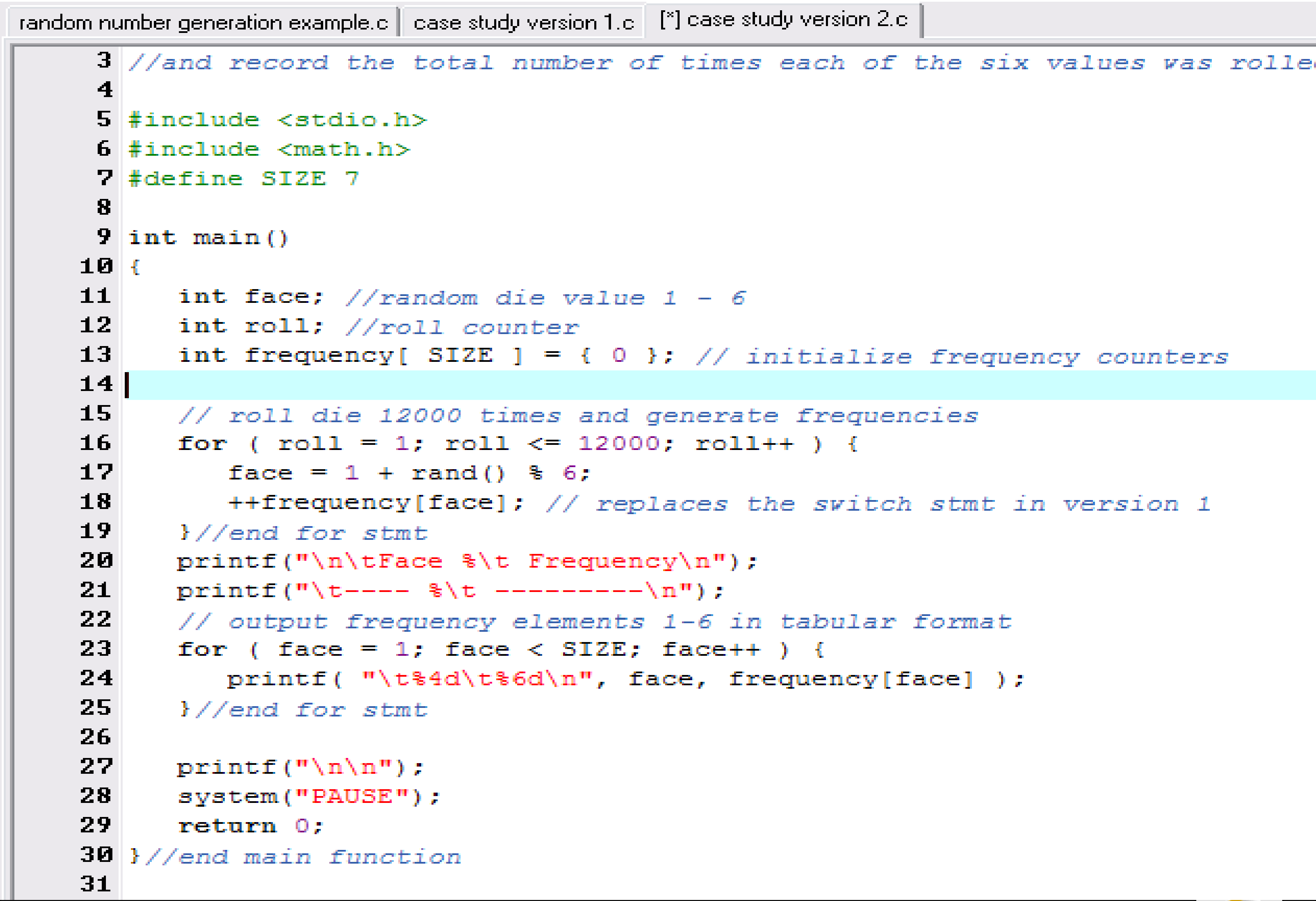


```
        Face        Frequency
        ----        ---------
          1            1989
          2            2004
          3            1975
          4            2051
          5            2019
          6            1962

Press any key to continue . . .
```

Note that as we predicted, with equal probability, each of the numbers was rolled about 2000 times.

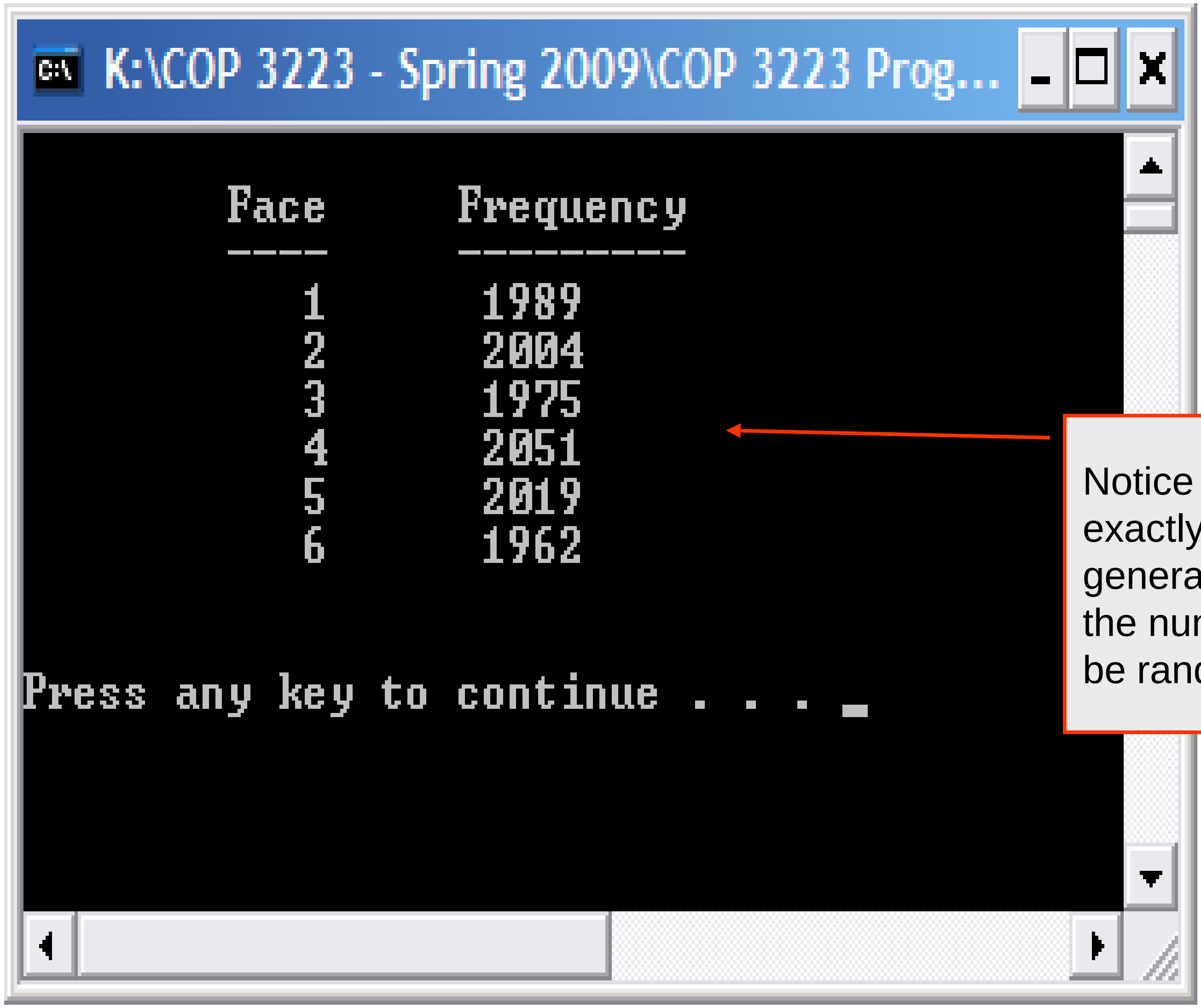
A Small Case Study - continues

- Now let’s rewrite the previous solution, but this time we’ll use an array of integer values to hold the frequency of each face value that is rolled in the corresponding position in the array.
- We’ll call this array back to the case study… the first version of the code we’ll write to solve the problem of rolling a standard die 12,000 times and seeing what the frequency of each value rolled, we’ll do without using arrays.
- This will mean that we’ll need a separate variable for each of the six possible values that could be rolled.
- The program on the next two pages (it wouldn’t fit on one page) illustrates a solution to this problem without the use of arrays.
- Notice how much “smaller” version 2 is compared to version 1.



random number generation example.c | case study version 1.c | [*] case study version 2.c

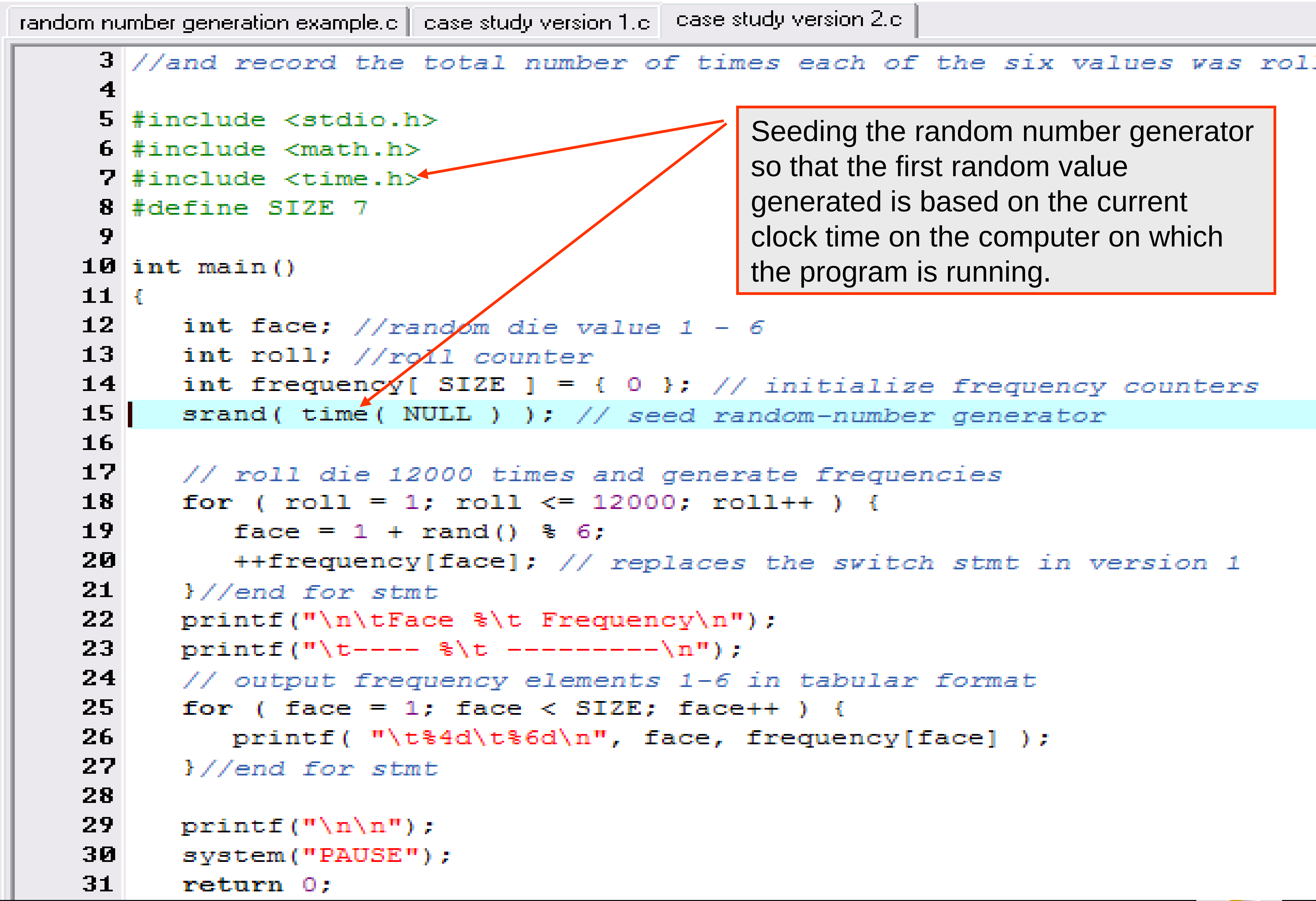
```c
//and record the total number of times each of the six values was rolle

#include <stdio.h>
#include <math.h>
#define SIZE 7

int main()
{
    int face; //random die value 1 - 6
    int roll; //roll counter
    int frequency[ SIZE ] = { 0 }; // initialize frequency counters

    // roll die 12000 times and generate frequencies
    for ( roll = 1; roll <= 12000; roll++ ) {
        face = 1 + rand() % 6;
        ++frequency[face]; // replaces the switch stmt in version 1
    }//end for stmt
    printf("\n\tFace %\t Frequency\n");
    printf("\t---- %\t ---------\n");
    // output frequency elements 1-6 in tabular format
    for ( face = 1; face < SIZE; face++ ) {
        printf( "\t%4d\t%6d\n", face, frequency[face] );
    }//end for stmt

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



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

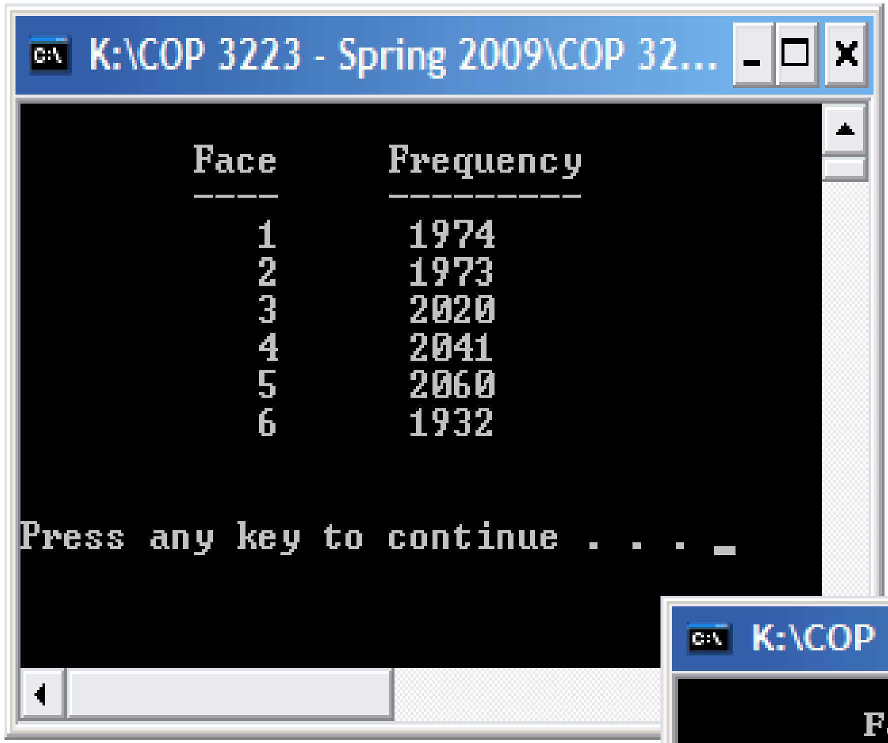
```
        Face        Frequency
        ----        ---------
           1         1989
           2         2004
           3         1975
           4         2051
           5         2019
           6         1962


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

Notice that these values are exactly the same as those generated by version 1! Since the numbers are supposed to be random, why did this occur?

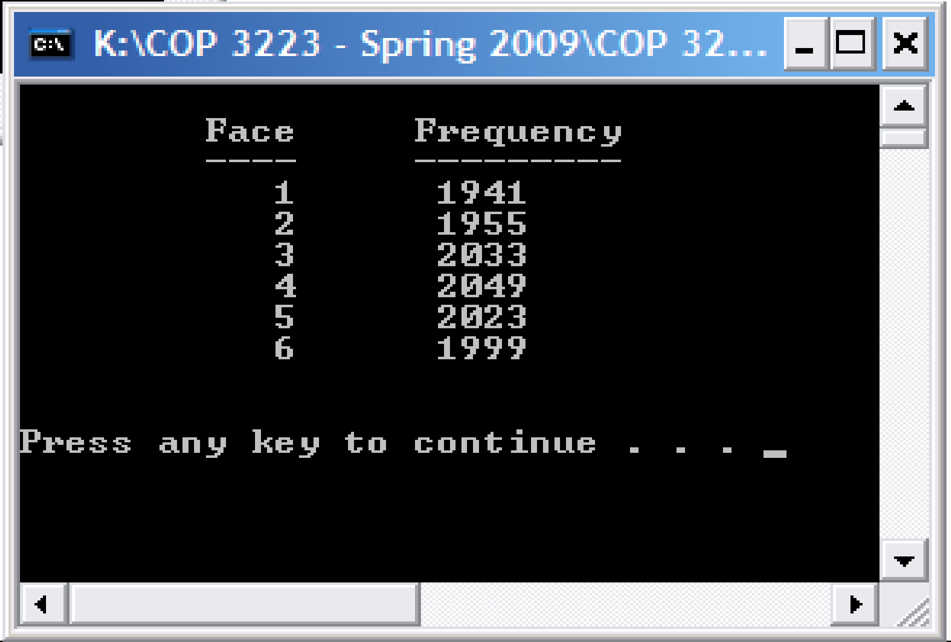


```
//and record the total number of times each of the six values was rol

#include <stdio.h>
#include <math.h>
#include <time.h>
#define SIZE 7

int main()
{
    int face; //random die value 1 - 6
    int roll; //roll counter
    int frequency[ SIZE ] = { 0 }; // initialize frequency counters
    srand( time( NULL ) ); // seed random-number generator

    // roll die 12000 times and generate frequencies
    for ( roll = 1; roll <= 12000; roll++ ) {
        face = 1 + rand() % 6;
        ++frequency[face]; // replaces the switch stmt in version 1
    }//end for stmt
    printf("\n\tFace %\t Frequency\n");
    printf("\t---- %\t ---------\n");
    // output frequency elements 1-6 in tabular format
    for ( face = 1; face < SIZE; face++ ) {
       printf( "\t%4d\t%6d\n", face, frequency[face] );
    }//end for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
```

Seeding the random number generator so that the first random value generated is based on the current clock time on the computer on which the program is running.



```
          Face        Frequency
          ----        ---------

             1         1974
             2         1973
             3         2020
             4         2041
             5         2060
             6         1932


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



```
          Face        Frequency
          ----        ---------
             1         1941
             2         1955
             3         2033
             4         2049
             5         2023
             6         1999

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

Two different runs of case study version 2. Notice that the frequency counters are different, indicating that the sequence of random numbers generated is not based on the same starting (or seed) value, and hence appears more random.

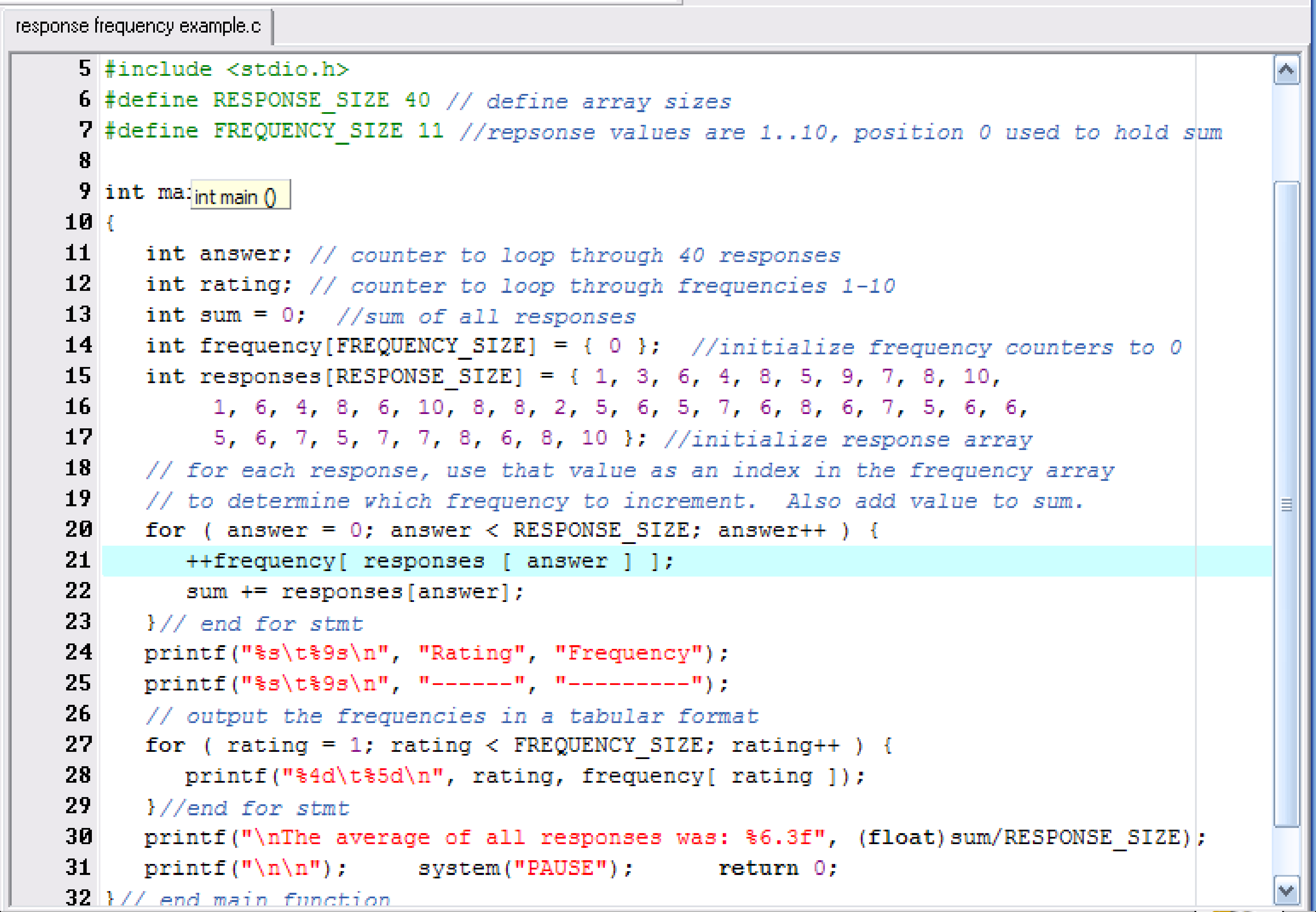
Arrays In C

- This next example, illustrates a fairly common use for arrays. In this case, two arrays are used, `responses[]`, holds user responses to a survey question where the user was asked to rank something on a scale from 1 to 10; `frequency[]`, is used to hold the frequencies of the user responses. Thus, `frequency[i]` holds the number of times the user responded with `i` on the scale from 1 to 10.

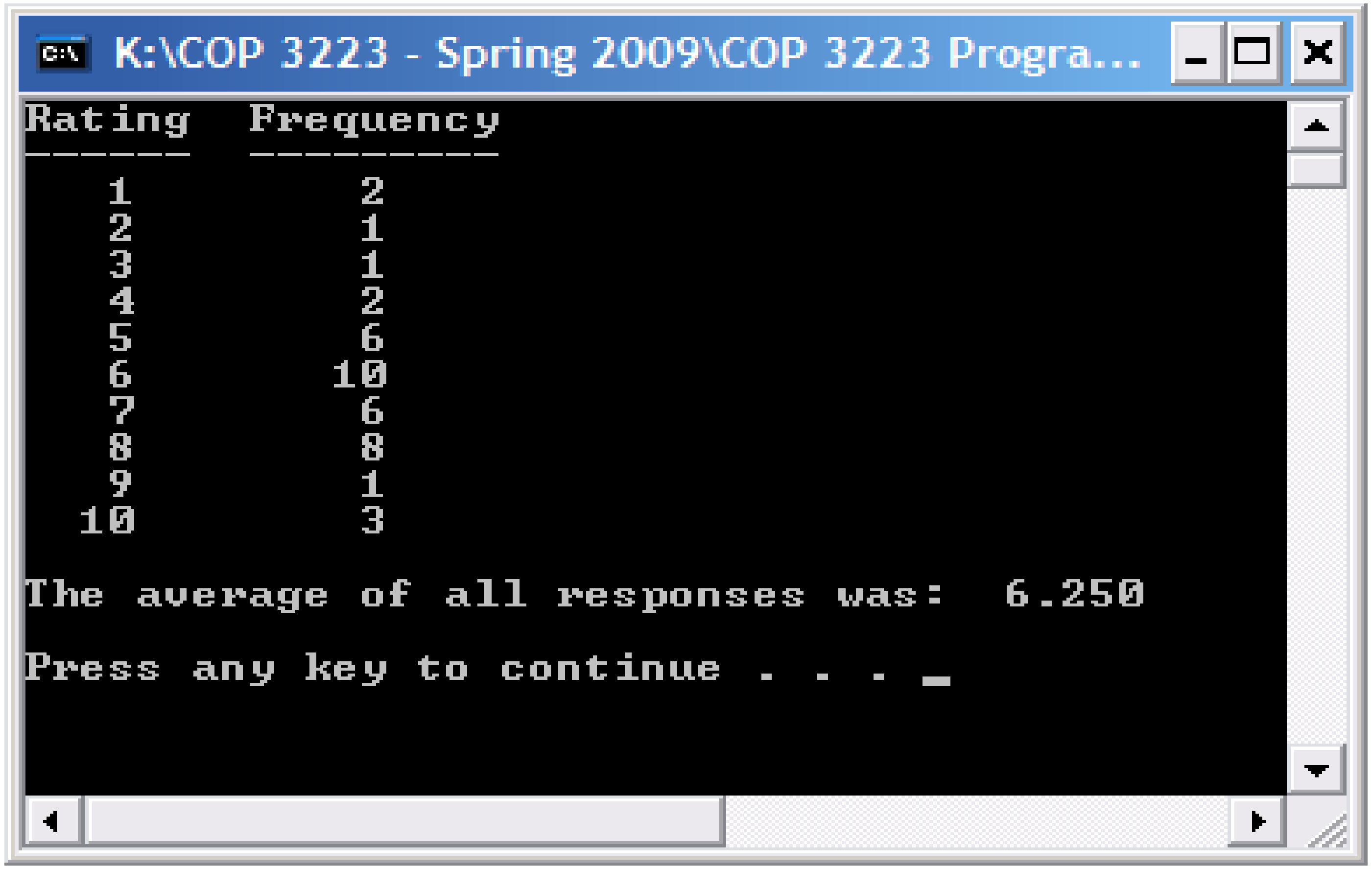
- Notice on line 21 in the program how the array `frequency` is indexed:

```
++frequency[ response[answer] ];
```

`response[answer]` will have a value between 1 and 10, thus is the user's response was 4, then the value in `frequency[4]` will be incremented.



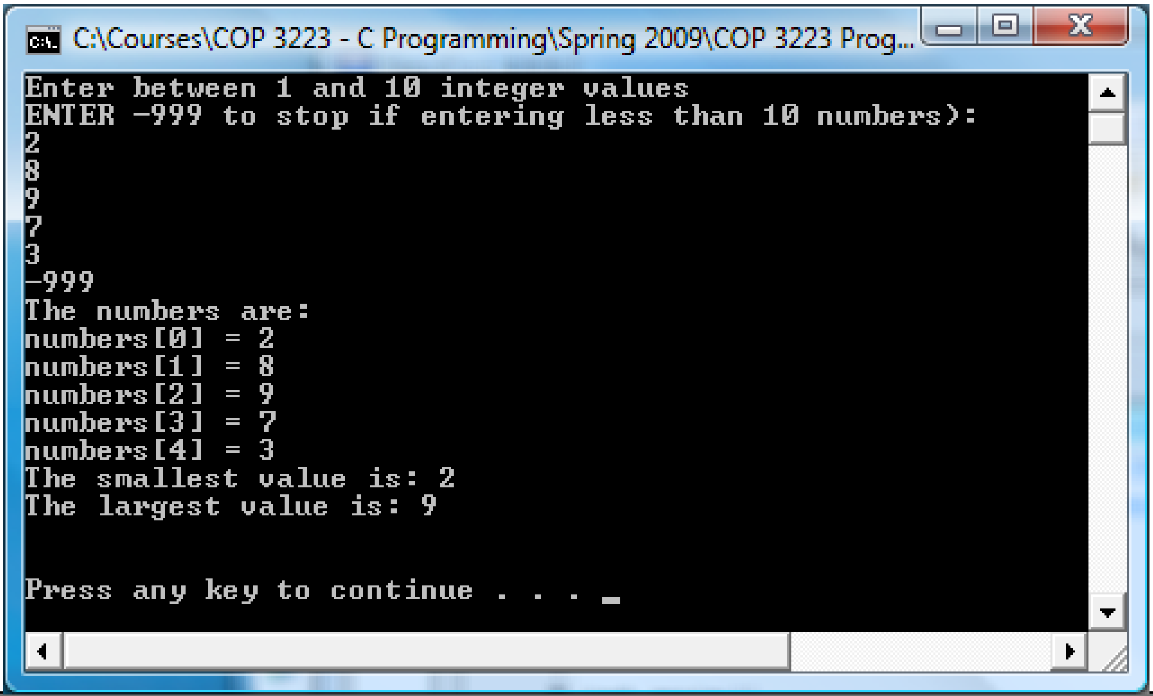
```c
#include <stdio.h>
#define RESPONSE_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11 //repsonse values are 1..10, position 0 used to hold sum

int ma
{
   int answer; // counter to loop through 40 responses
   int rating; // counter to loop through frequencies 1-10
   int sum = 0;  //sum of all responses
   int frequency[FREQUENCY_SIZE] = { 0 };  //initialize frequency counters to 0
   int responses[RESPONSE_SIZE] = { 1, 3, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 4, 8, 6, 10, 8, 8, 2, 5, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 7, 7, 8, 6, 8, 10 }; //initialize response array
   // for each response, use that value as an index in the frequency array
   // to determine which frequency to increment.  Also add value to sum.
   for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
      ++frequency[ responses [ answer ] ];
      sum += responses[answer];
   }// end for stmt
   printf("%s\t%9s\n", "Rating", "Frequency");
   printf("%s\t%9s\n", "------", "---------");
   // output the frequencies in a tabular format
   for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
      printf("%4d\t%5d\n", rating, frequency[ rating ]);
   }//end for stmt
   printf("\nThe average of all responses was: %6.3f", (float)sum/RESPONSE_SIZE);
   printf("\n\n");     system("PAUSE");      return 0;
}// end main function
```



```
Rating   Frequency
------   ---------
    1          2
    2          1
    3          1
    4          2
    5          6
    6         10
    7          6
    8          8
    9          1
   10          3

The average of all responses was:   6.250

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

Practice Problems

1. Write a C program that reads in a maximum of 10 integers from the keyboard and stores the integers in an array. Then using the values in the array finds the smallest and largest number the user entered. Assume that the user will enter the value -999 as a sentinel value it they wish to enter less than 10 integers. Note that the sentinel value is not considered to be one of the numbers stored in the array.


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Prog...

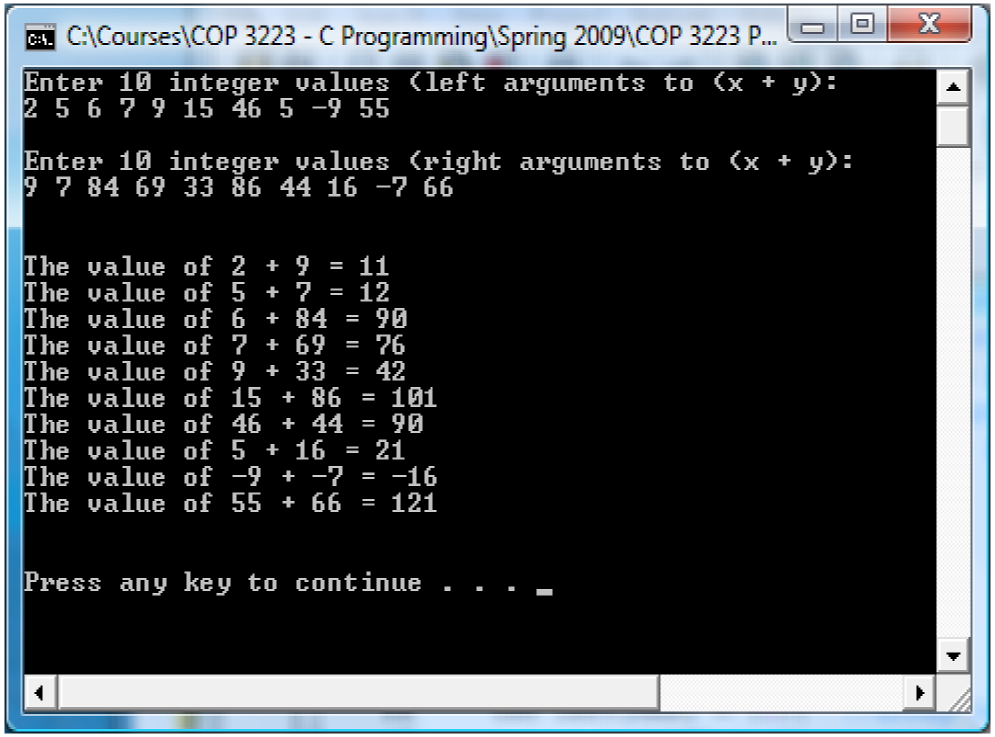
```
Enter between 1 and 10 integer values
ENTER -999 to stop if entering less than 10 numbers):
2
8
9
7
3
-999
The numbers are:
numbers[0] = 2
numbers[1] = 8
numbers[2] = 9
numbers[3] = 7
numbers[4] = 3
The smallest value is: 2
The largest value is: 9


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

Practice Problems

2. Write a C program that creates two integer array each with a `MAX` number of elements (assume `MAX` is defined to be 10), has the user enter `MAX` integer values into each array. Then a third array is used to store the sums of the values in the first two arrays on an element by element basis.

   For example, if the arrays are named `A, B,` and `Sum`, then

   `Sum[0] = A[0] + B[0] and Sum[1] = A[1] + B[1]`… and so on.


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 P...

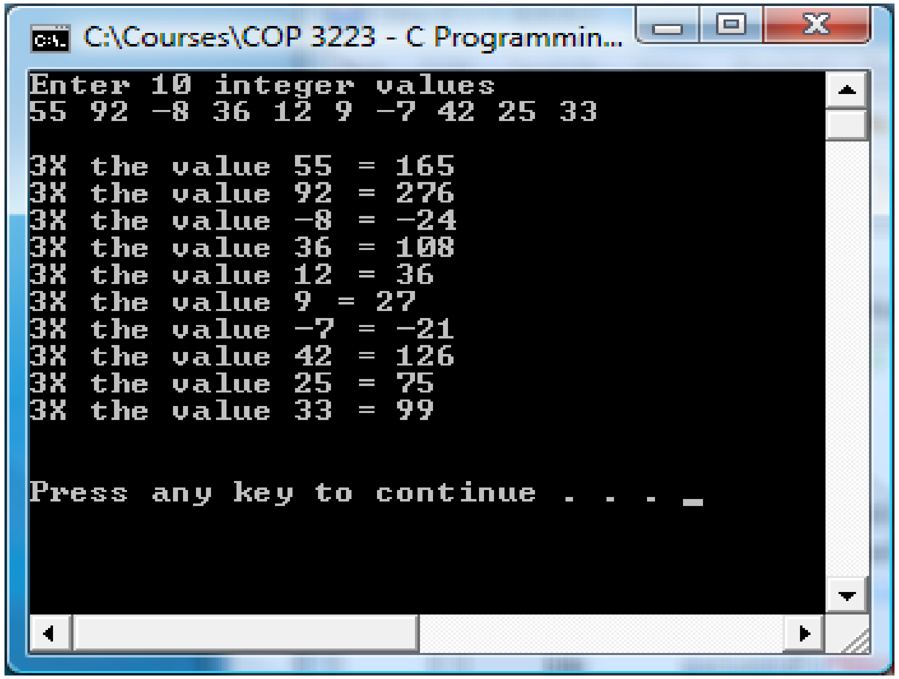
```
Enter 10 integer values (left arguments to (x + y):
2 5 6 7 9 15 46 5 -9 55

Enter 10 integer values (right arguments to (x + y):
9 7 84 69 33 86 44 16 -7 66


The value of 2 + 9 = 11
The value of 5 + 7 = 12
The value of 6 + 84 = 90
The value of 7 + 69 = 76
The value of 9 + 33 = 42
The value of 15 + 86 = 101
The value of 46 + 44 = 90
The value of 5 + 16 = 21
The value of -9 + -7 = -16
The value of 55 + 66 = 121


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

Practice Problems

3. Write a C program that creates and array holding a `MAX` number of elements (assume `MAX` is defined to be 10), has the user enter MAX integer values into the array. Then a second array is used to store three times the value in the first array on an element by element basis.

   For example, if the arrays are named `A` and `B`, then

   `B[0] = A[0] *3 and B[1] = A[1] * 3` … and so on.



C:\Courses\COP 3223 - C Programmin...

```
Enter 10 integer values
55 92 -8 36 12 9 -7 42 25 33

3X the value 55 = 165
3X the value 92 = 276
3X the value -8 = -24
3X the value 36 = 108
3X the value 12 = 36
3X the value 9 = 27
3X the value -7 = -21
3X the value 42 = 126
3X the value 25 = 75
3X the value 33 = 99


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

Practice Problems

4. Modify the frequency response program on page 20, so that the 40 user responses are read from an input file named "`survey responses.dat`". Be sure to create this file before you attempt to run your program since this is an input file (i.e., you are reading from the file so it must pre-exist).

5. Modify the program you just wrote for Practice Problem 4, so that the maximum number of responses is defined to be 100, but any number from 1 up to 100 responses may be in the file, but how many responses are in the file is unknown in advance (i.e., you must detect end-of-file).