CENG 230
Introduction to C Programming

Week 12 – Arrays

Sinan Kalkan

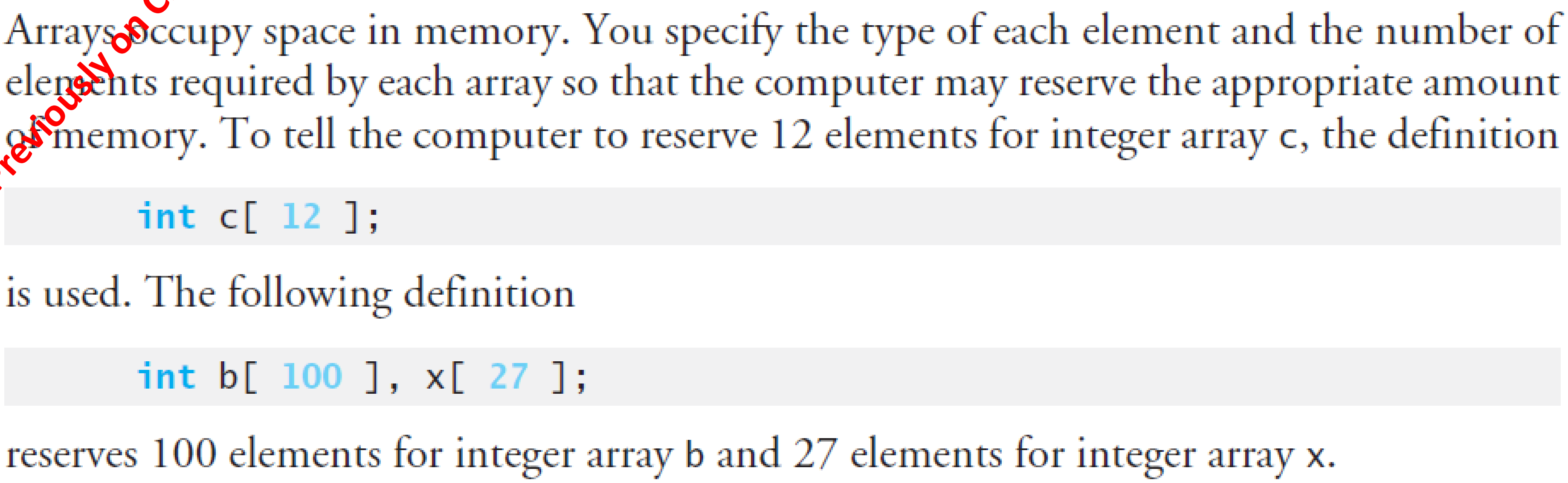
Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli, and the lecture notes of the textbook by Hanly and Koffman.

Last week

Previously on CEng 230!

- Arrays:
  - Storing and working collections of data
  - Declaration & use
  - Initialization
  - Passing arrays to functions
  - Multi-dimensional arrays

Previously on CEng 230!



Arrays occupy space in memory. You specify the type of each element and the number of elements required by each array so that the computer may reserve the appropriate amount of memory. To tell the computer to reserve 12 elements for integer array c, the definition

```
      int c[ 12 ];
```

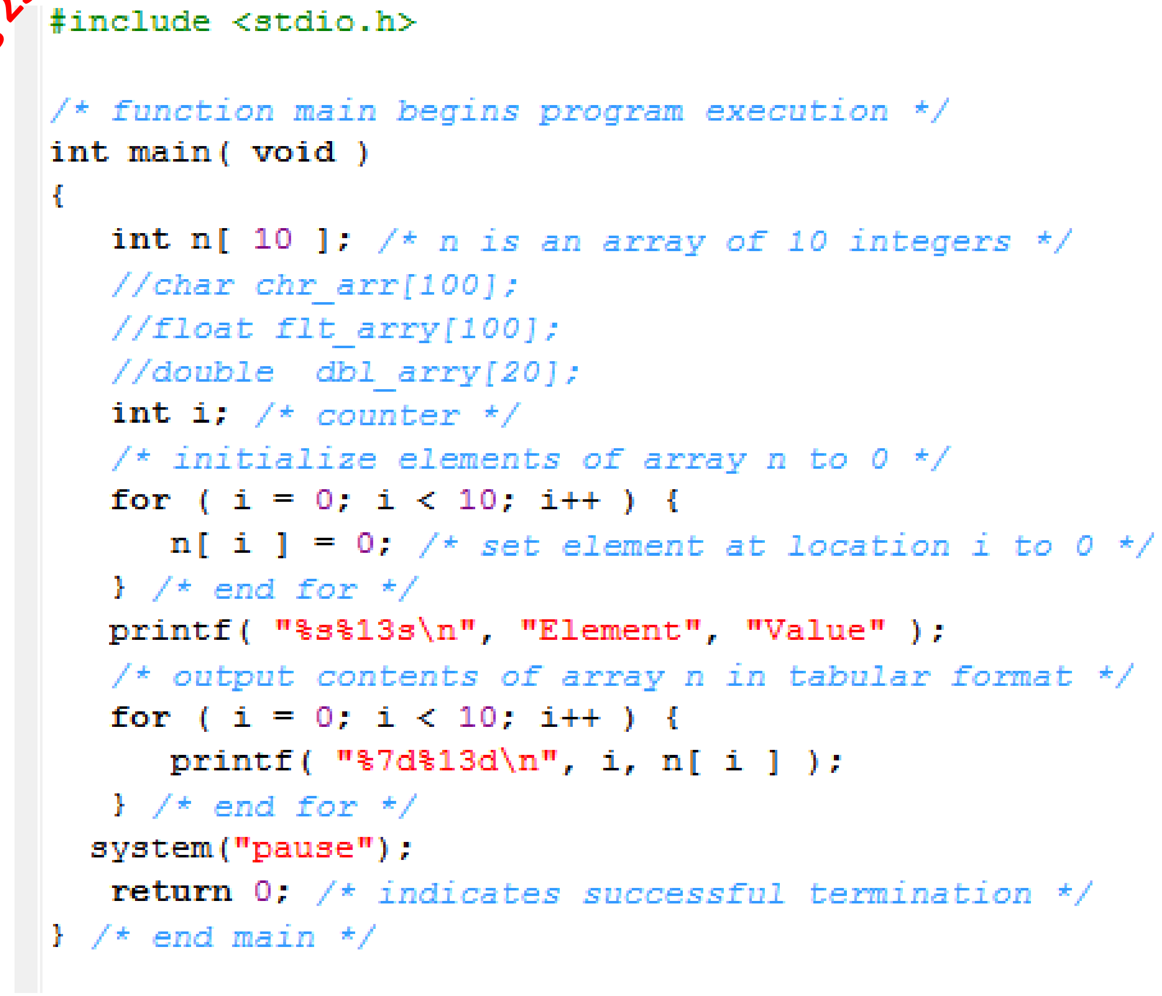
is used. The following definition

```
      int b[ 100 ], x[ 27 ];
```

reserves 100 elements for integer array b and 27 elements for integer array x.

```
char    chr_arr[100];
float   flt_arry[100];
double    dbl_arry[20];
```

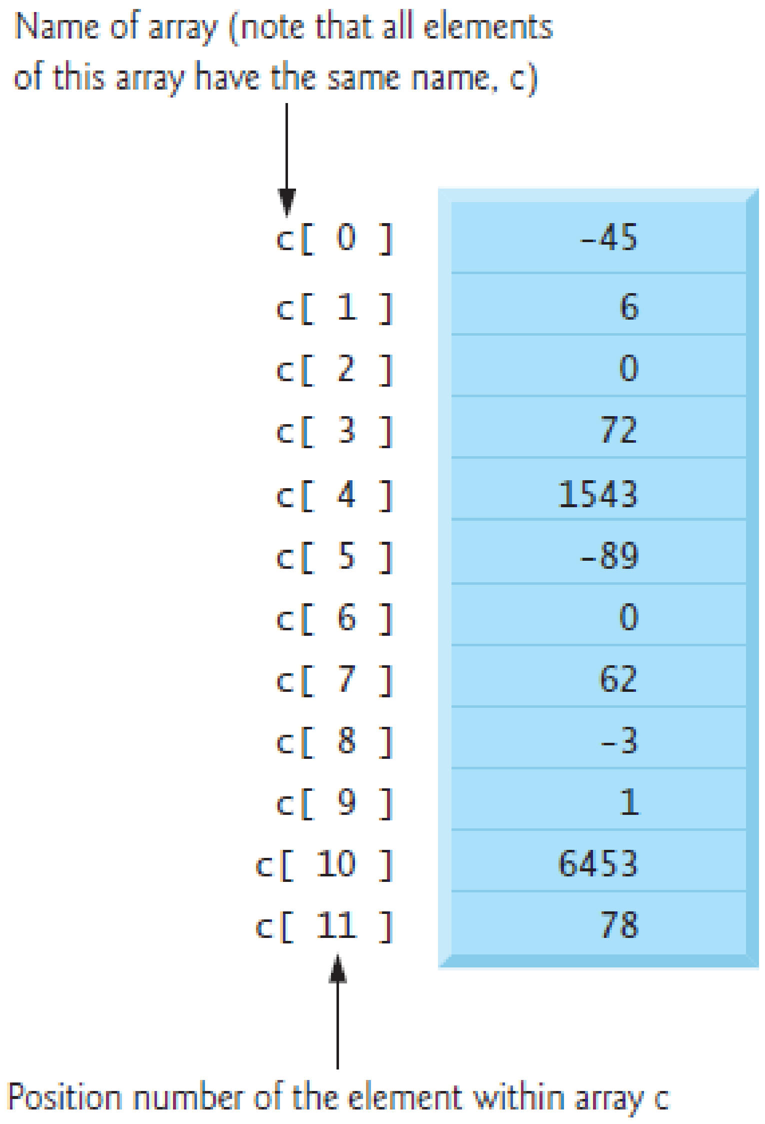
Previously on CEng 230!



```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int n[ 10 ]; /* n is an array of 10 integers */
   //char chr_arr[100];
   //float flt_arry[100];
   //double  dbl_arry[20];
   int i; /* counter */
   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = 0; /* set element at location i to 0 */
   } /* end for */
   printf( "%s%13s\n", "Element", "Value" );
   /* output contents of array n in tabular format */
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   } /* end for */
  system("pause");
   return 0; /* indicates successful termination */
} /* end main */
```

An array is 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 or element in the array, we specify the name of the array and the **position number** of the particular element in the array.

Previously on CEng 230!



Name of array (note that all elements of this array have the same name, c)
c[0] -45
c[1] 6
c[2] 0
c[3] 72
c[4] 1543
c[5] -89
c[6] 0
c[7] 62
c[8] -3
c[9] 1
c[10] 6453
c[11] 78
Position number of the element within array c

Previously on CEng 230!

TABLE 7.1 Statements That Manipulate Array x

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

Previously on CEng 230!

For example, if a = 5 and b = 6, then the statement

```
    c[ a + b ] += 2;
```

adds 2 to array element c[11]. A subscripted array name is an *lvalue*—it can be used on the left side of an assignment.

```
    printf( "%d", c[ 0 ] + c[ 1 ] + c[ 2 ] );
```

```
    x = c[ 6 ] / 2;
```

Previously on CEng230!

7.2 Array Subscripts

We use a subscript to differentiate between the individual array elements and to specify which array element is to be manipulated. We can use any expression of type `int` as an array subscript. However, to create a valid reference, the value of this subscript must lie between 0 and one less than the declared size of the array.

EXAMPLE 7.3

Understanding the distinction between an array subscript value and an array element value is essential. The original array `x` from Fig. 7.1 follows. The subscripted variable `x[i]` references a particular element of this array. If `i` has the value `0`, the subscript value is `0`, and `x[0]` is referenced. The value of `x[0]` in this case is `16.0`. If `i` has the value `2`, the subscript value is `2`, and the value of `x[i]` is `6.0`. If `i` has the value `8`, the subscript value is `8`, and we cannot predict the value of `x[i]` because the subscript value is out of the allowable range.

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Previously on CEng 230!

TABLE 7.2 Code Fragment That Manipulates Array x

Statement	Explanation
`i = 5;`	
`printf("%d %.1f", 4, x[4]);`	Displays 4 and 2.5 (value of `x[4]`)
`printf("%d %.1f", i, x[i]);`	Displays 5 and 12.0 (value of `x[5]`)
`printf("%.1f", x[i] + 1);`	Displays 13.0 (value of `x[5]` plus 1)
`printf("%.1f", x[i] + i);`	Displays 17.0 (value of `x[5]` plus 5)
`printf("%.1f", x[i + 1]);`	Displays 14.0 (value of `x[6]`)
`printf("%.1f", x[i + i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f", x[2 * i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f", x[2 * i - 3]);`	Displays -54.5 (value of `x[7]`)
`printf("%.1f", x[(int)x[4]]);`	Displays 6.0 (value of `x[2]`)
`printf("%.1f", x[i++]);`	Displays 12.0 (value of `x[5]`); then assigns 6 to `i`
`printf("%.1f", x[--i]);`	Assigns 5 (6 - 1) to `i` and then displays 12.0 (value of `x[5]`)
`x[i - 1] = x[i];`	Assigns 12.0 (value of `x[5]`) to `x[4]`
`x[i] = x[i + 1];`	Assigns 14.0 (value of `x[6]`) to `x[5]`
`x[i] - 1 = x[i];`	Illegal assignment statement

Previously on CEng 230!

Array initialization

Previously on CEng 230!

If there are fewer initializers than elements in the array, the remaining elements are initialized to zero. For example, the elements of the array n in Fig. 6.3 could have been initialized to zero as follows:

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

The array definition

```
int n[ 5 ] = { 32, 27, 64, 18, 95, 14 };
```

causes a syntax error because there are six initializers and only five array elements.

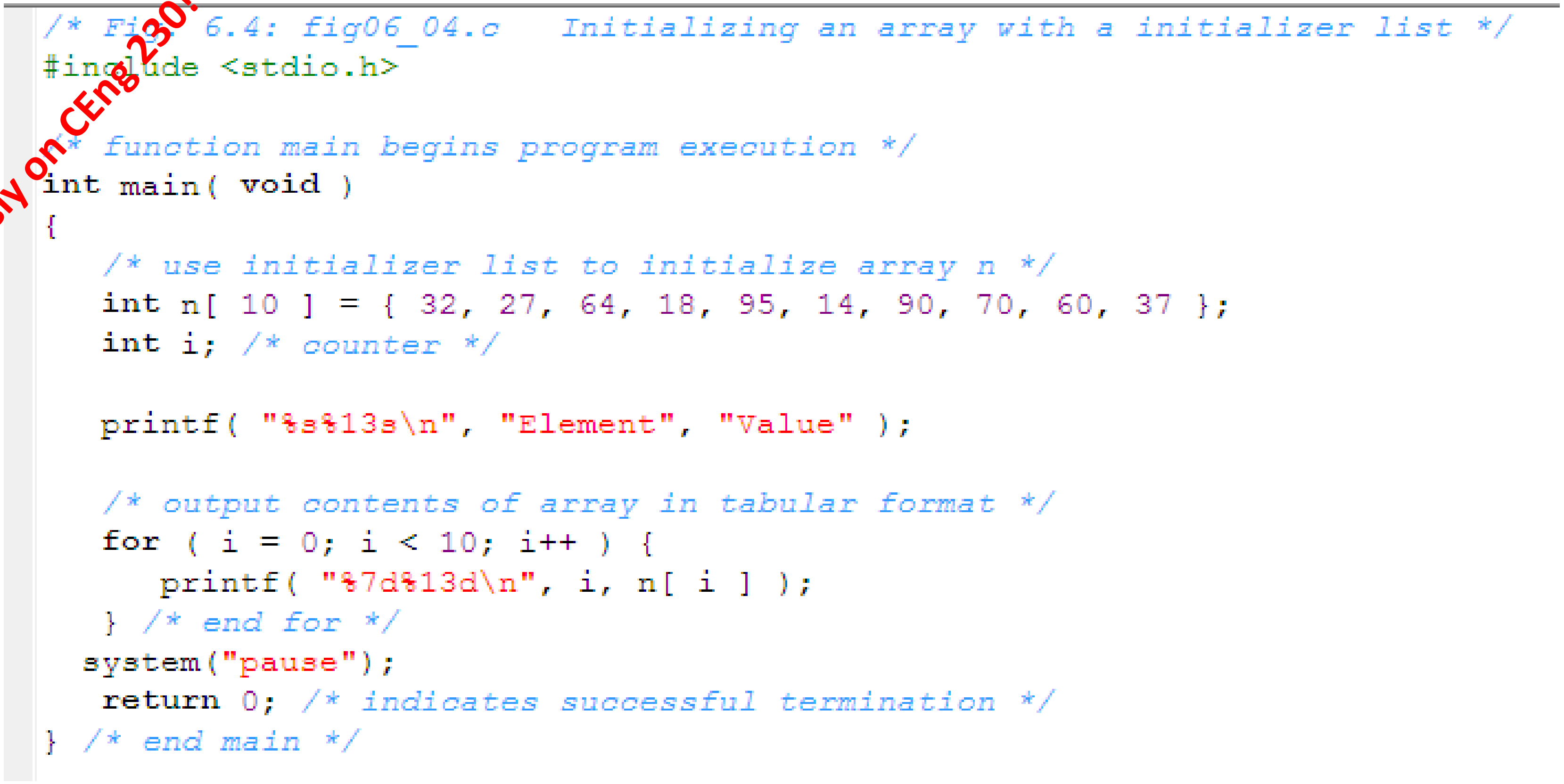
If the array size is omitted from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list. For example,

```
int n[] = { 1, 2, 3, 4, 5 };
```

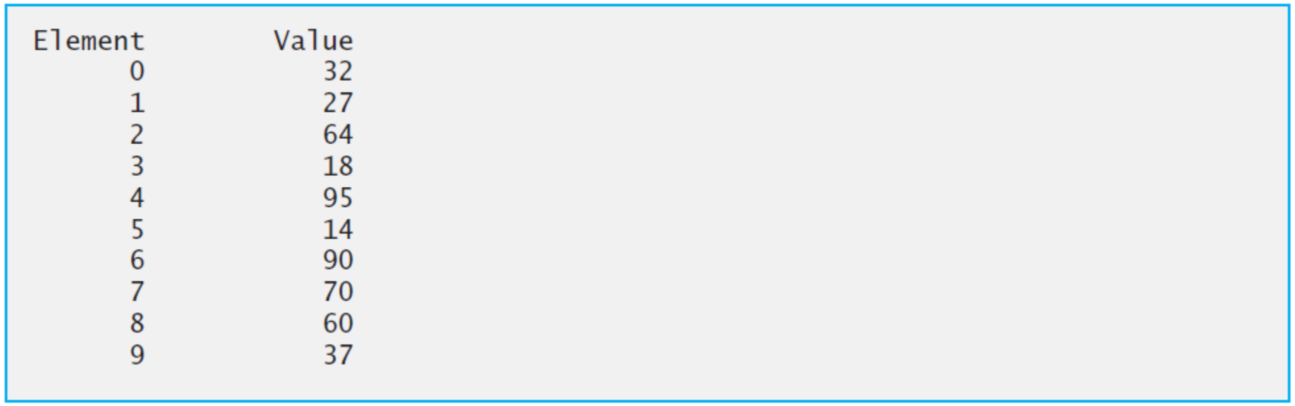
would create a five-element array.

Initializing an Array in a Definition with an initializer List

Previously on CEng 230!



```c
/* Fig. 6.4: fig06_04.c   Initializing an array with a initializer list */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   /* use initializer list to initialize array n */
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
   int i; /* counter */

   printf( "%s%13s\n", "Element", "Value" );

   /* output contents of array in tabular format */
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   } /* end for */
  system("pause");
   return 0; /* indicates successful termination */
} /* end main */
```



```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

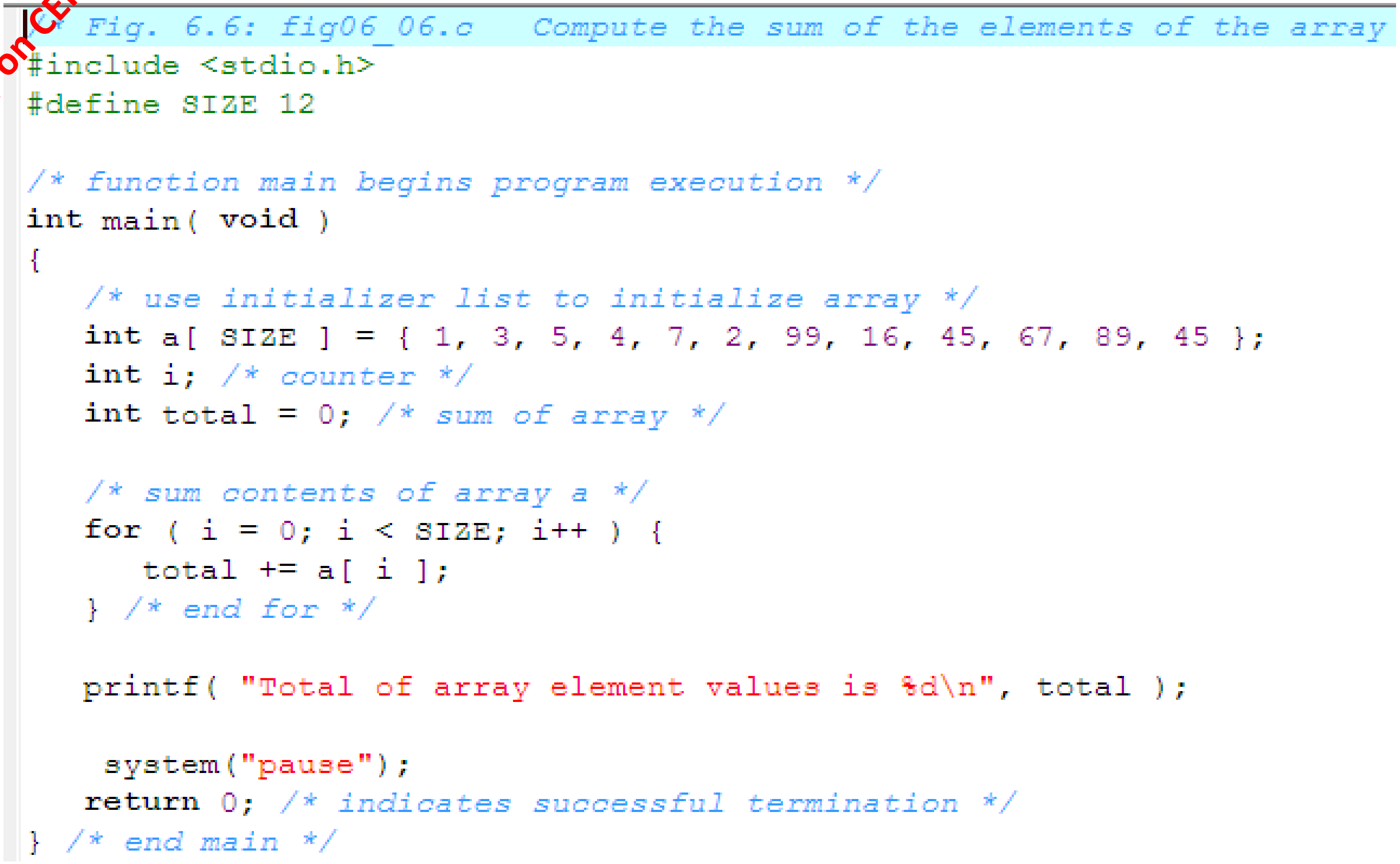
Specifying an Array’s Size with a Symbolic Constant and Initializing Array Elements with Calculations

Previously on CEng 230!

```
/* Fig. 6.5: fig06_05.c
   Initialize the elements of array s to the even integers from 2 to 20 */
#include <stdio.h>
#define SIZE 10 /* maximum size of array */

/* function main begins program execution */
int main( void )
{
   /* symbolic constant SIZE can be used to specify array size */
   int s[ SIZE ]; /* array s has SIZE elements */
   int j; /* counter */

   for ( j = 0; j < SIZE; j++ ) { /* set the values */
      s[ j ] = 2 + 2 * j;
   } /* end for */

   printf( "%s%13s\n", "Element", "Value" );

   /* output contents of array s in tabular format */
   for ( j = 0; j < SIZE; j++ ) {
      printf( "%7d%13d\n", j, s[ j ] );
   } /* end for */

   return 0; /* indicates successful termination */
} /* end main */
```

Summing the Elements of an Array

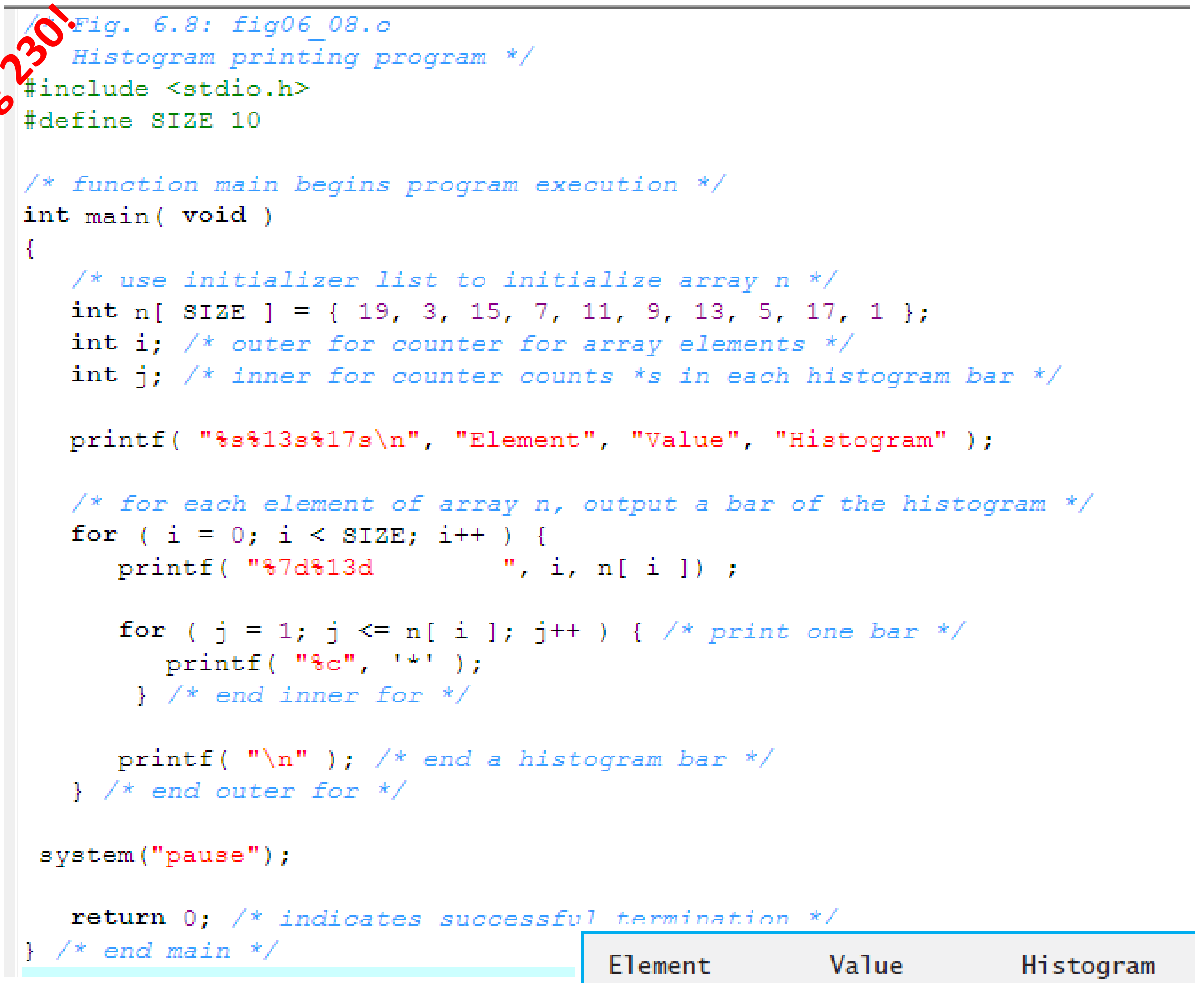
Previously on CEng 230!



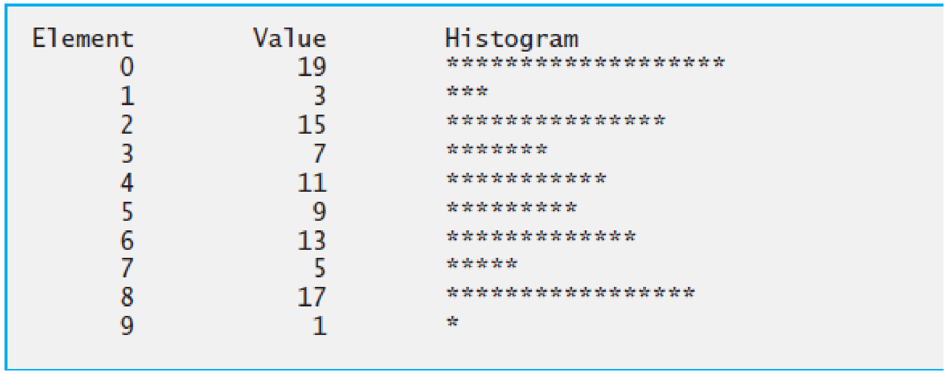
```
/* Fig. 6.6: fig06_06.c   Compute the sum of the elements of the array
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main( void )
{
   /* use initializer list to initialize array */
   int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   int i; /* counter */
   int total = 0; /* sum of array */

   /* sum contents of array a */
   for ( i = 0; i < SIZE; i++ ) {
      total += a[ i ];
   } /* end for */

   printf( "Total of array element values is %d\n", total );

    system("pause");
   return 0; /* indicates successful termination */
} /* end main */
```

Graphing Array Element Values with Histograms

Previously on CEng 230!



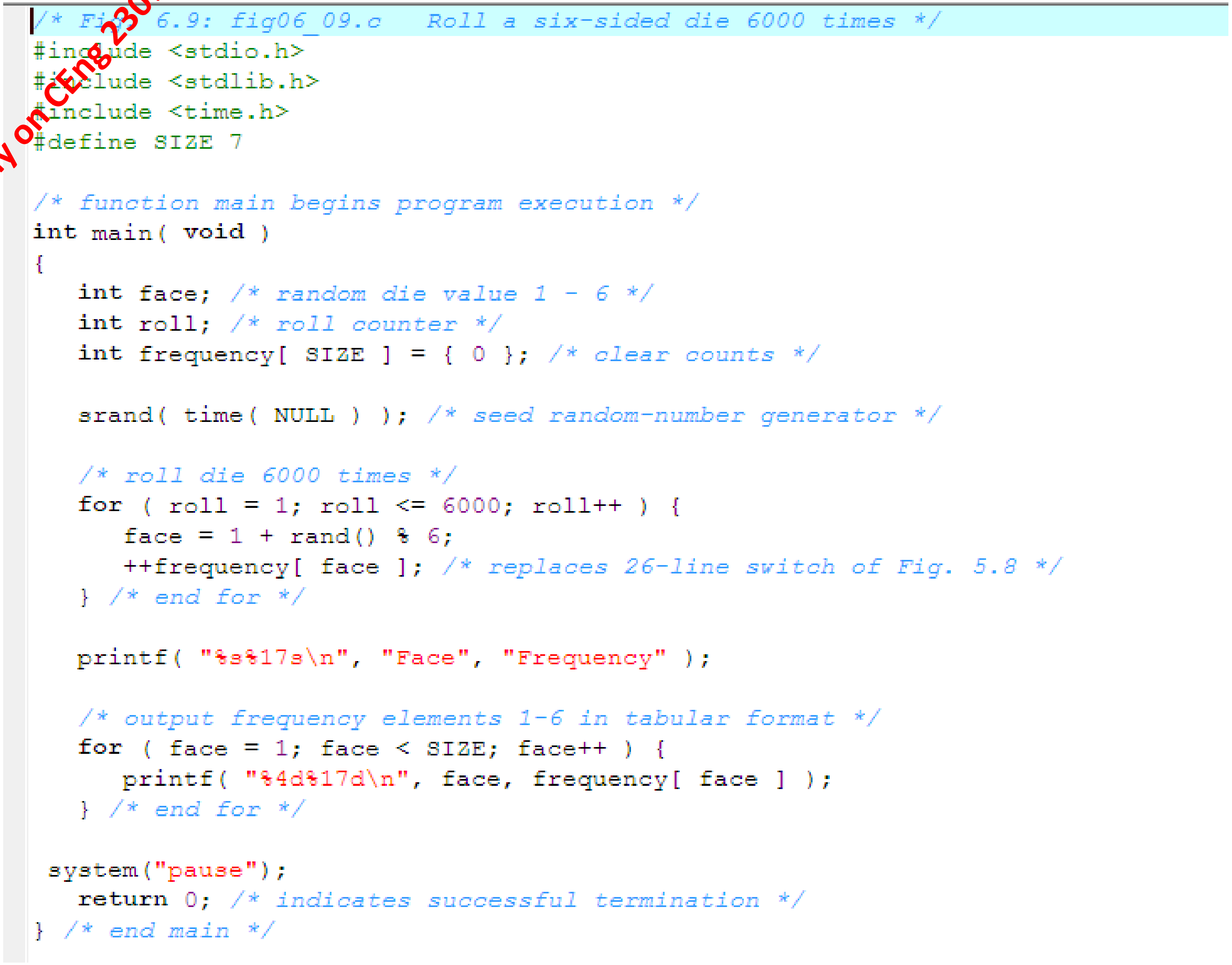
```
/* Fig. 6.8: fig06_08.c
   Histogram printing program */
#include <stdio.h>
#define SIZE 10

/* function main begins program execution */
int main( void )
{
   /* use initializer list to initialize array n */
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   int i; /* outer for counter for array elements */
   int j; /* inner for counter counts *s in each histogram bar */

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   /* for each element of array n, output a bar of the histogram */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%7d%13d        ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
         printf( "%c", '*' );
       } /* end inner for */

      printf( "\n" ); /* end a histogram bar */
   } /* end outer for */

 system("pause");

   return 0; /* indicates successful termination */
} /* end main */
```



```
Element        Value        Histogram
      0           19        *******************
      1            3        ***
      2           15        ***************
      3            7        *******
      4           11        ***********
      5            9        *********
      6           13        *************
      7            5        *****
      8           17        *****************
      9            1        *
```

Rolling a Die 6000 Times and Summarizing the Results in an Array

Previously on CEng 230!



```
/* Fig. 6.9: fig06_09.c   Roll a six-sided die 6000 times */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 7

/* function main begins program execution */
int main( void )
{
   int face; /* random die value 1 - 6 */
   int roll; /* roll counter */
   int frequency[ SIZE ] = { 0 }; /* clear counts */

   srand( time( NULL ) ); /* seed random-number generator */

   /* roll die 6000 times */
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = 1 + rand() % 6;
      ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
   } /* end for */

   printf( "%s%17s\n", "Face", "Frequency" );

   /* output frequency elements 1-6 in tabular format */
   for ( face = 1; face < SIZE; face++ ) {
      printf( "%4d%17d\n", face, frequency[ face ] );
   } /* end for */

 system("pause");
   return 0; /* indicates successful termination */
} /* end main */
```

Previously on CEng 230!

6.5 Passing Arrays to Functions

To pass an array argument to a function, specify the name of the array without any brackets. For example, if array `hourlyTemperatures` has been defined as

```
    int hourlyTemperatures[ 24 ];
```

the function call

```
    modifyArray( hourlyTemperatures, 24 )
```

passes array `hourlyTemperatures` and its size to function `modifyArray`. Unlike `char` arrays that contain strings, other array types do not have a special terminator. For this reason, the size of an array is passed to the function, so that the function can process the proper number of elements.

C automatically passes arrays to functions by reference—the called functions can modify the element values in the callers' original arrays. The name of the array evaluates to the address of the first element of the array. Because the starting address of the array is passed, the called function knows precisely where the array is stored. Therefore, when the called function modifies array elements in its function body, it's modifying the actual elements of the array in their original memory locations.

Previously on CEng 230!

For a function to receive an array through a function call, the function's parameter list must specify that an array will be received. For example, the function header for function `modifyArray` (that we called earlier in this section) might be written as

```
void modifyArray( int b[], int size )
```

Fig. 6.13:

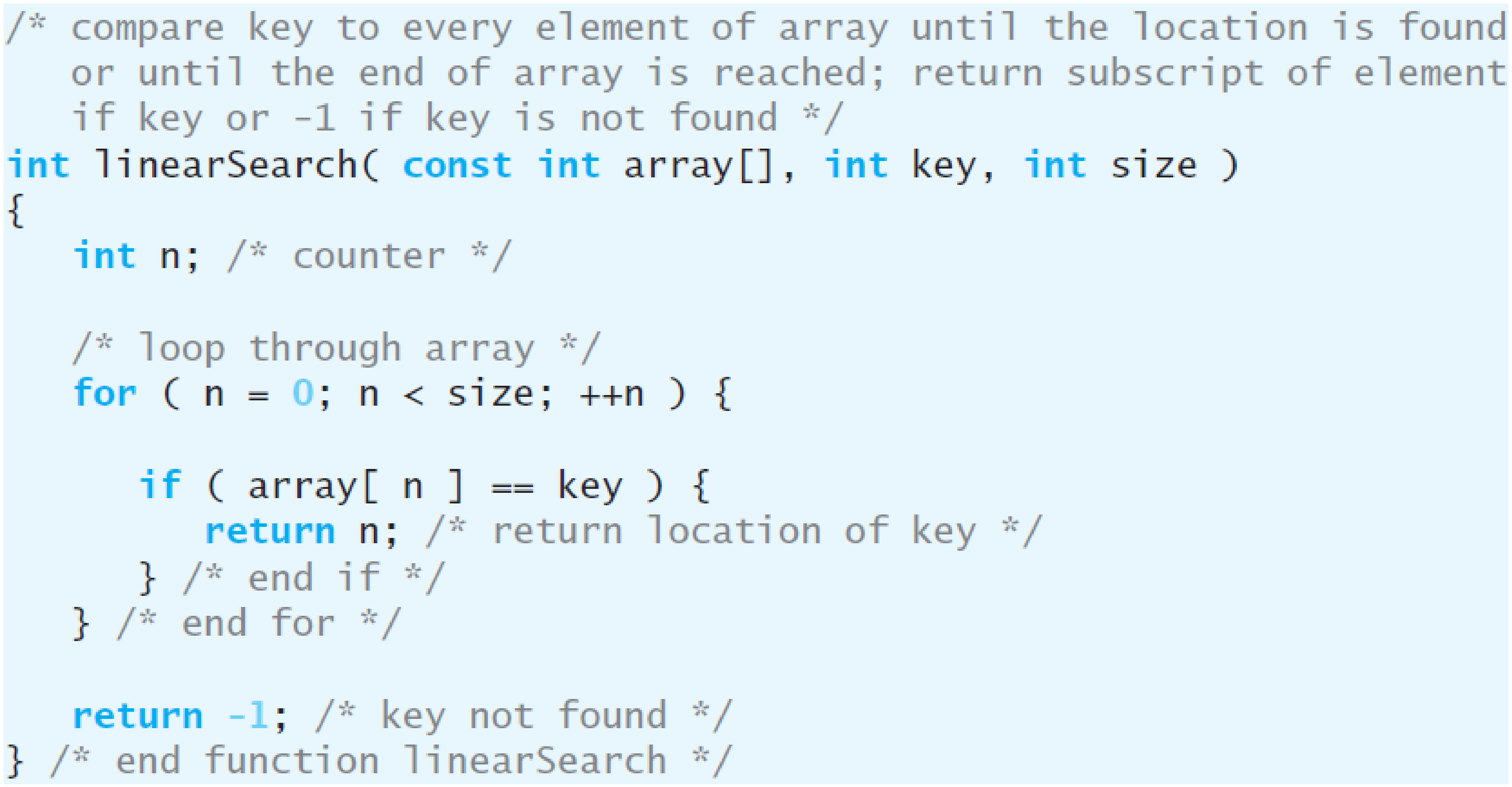
```
void modifyArray( int b[], int size )
{
   int j; /* counter */
   /* multiply each array element by 2 */
   for ( j = 0; j < size; j++ ) {
      b[ j ] *= 2;
   } /* end for */
} /* end function modifyArray */

/* in function modifyElement, "e" is a local copy of array element
   a[ 3 ] passed from main */
void modifyElement( int e )
{
   /* multiply parameter by 2 */
   printf( "Value in modifyElement is %d\n", e *= 2 );
} /* end function modifyElement */
```

```
modifyArray( a, SIZE );

modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
```

Today

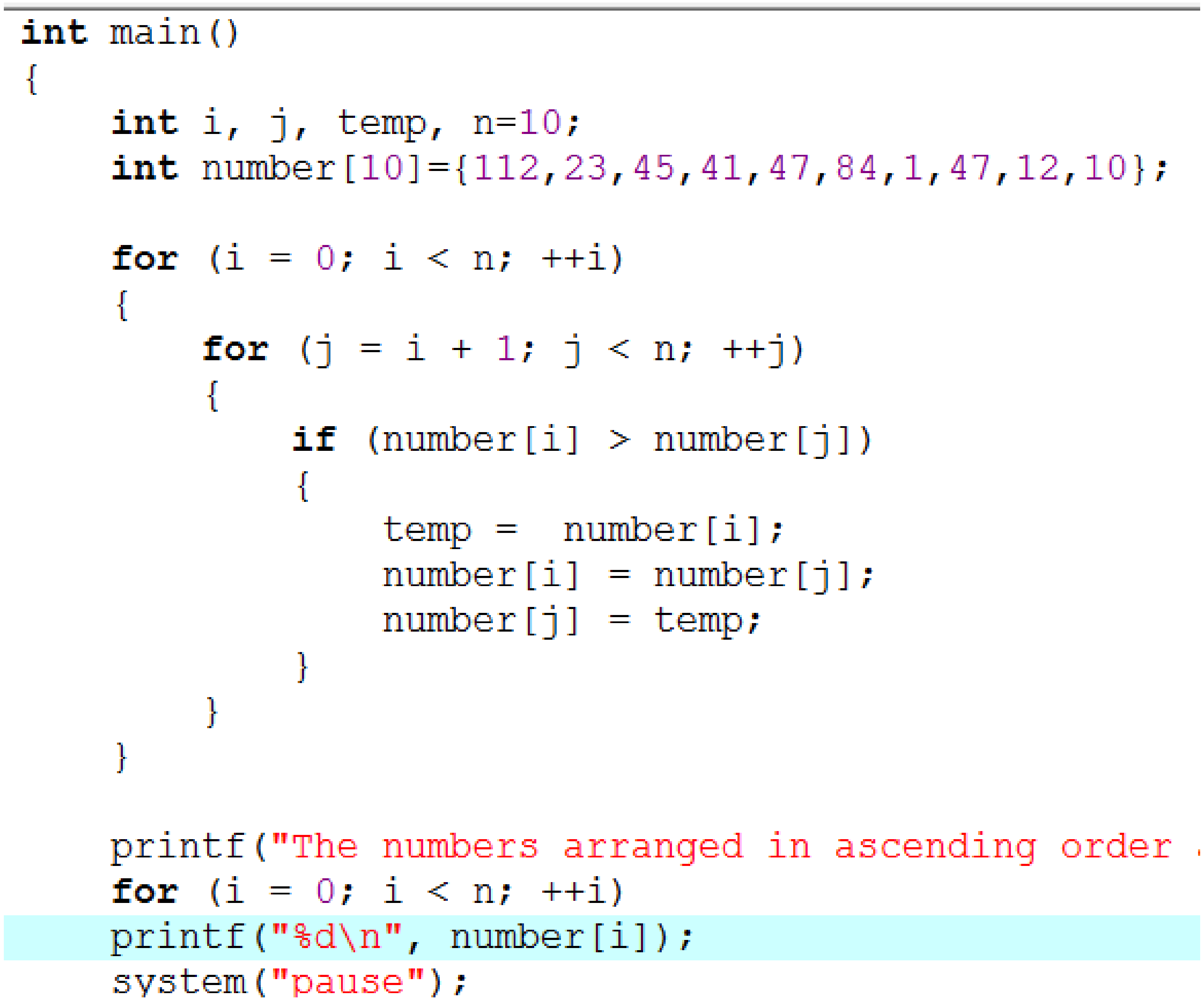
- Continue with arrays
  - Some examples
  - Multi-dimensional arrays

Searching Arrays (linear)



```c
/* compare key to every element of array until the location is found
   or until the end of array is reached; return subscript of element
   if key or -1 if key is not found */
int linearSearch( const int array[], int key, int size )
{
   int n; /* counter */

   /* loop through array */
   for ( n = 0; n < size; ++n ) {

      if ( array[ n ] == key ) {
         return n; /* return location of key */
      } /* end if */
   } /* end for */

   return -1; /* key not found */
} /* end function linearSearch */
```

Example: Sorting Arrays

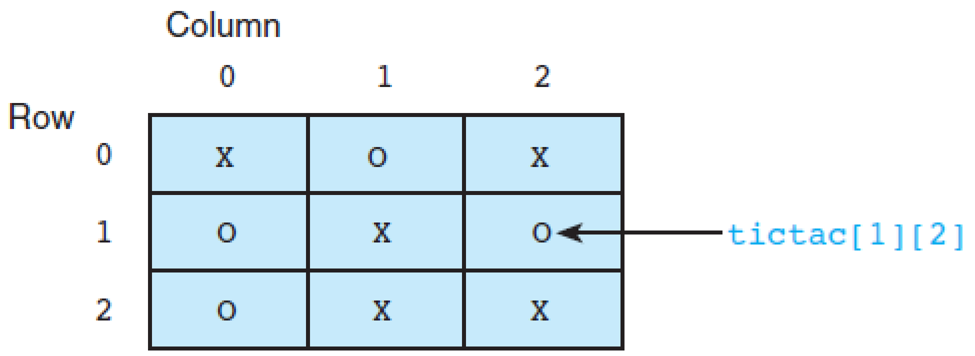


```
int main()
{
    int i, j, temp, n=10;
    int number[10]={112,23,45,41,47,84,1,47,12,10};

    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (number[i] > number[j])
            {
                temp =  number[i];
                number[i] = number[j];
                number[j] = temp;
            }
        }
    }

    printf("The numbers arranged in ascending order
    for (i = 0; i < n; ++i)
    printf("%d\n", number[i]);
    system("pause");
```

Multi-dimensional Arrays

Multidimensional Arrays

```
char tictac[3][3];
```



Initialization:

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

FIGURE 7.21 Function to Check Whether Tic-tac-toe Board Is Filled

```
/* Checks whether a tic-tac-toe board is completely filled.                      */
int
filled(char ttt_brd[3][3])  /* input - tic-tac-toe board                         */
{
      int r, c, /* row and column subscripts   */
          ans;  /* whether or not board filled */

      /* Assumes board is filled until blank is found                            */
      ans = 1;

      /* Resets ans to zero if a blank is found                                  */
      for (r = 0; r < 3; ++r)
         for  (c = 0; c < 3; ++c)
            if (ttt_brd[r][c] == ' ')
                ans = 0;

      return (ans);
}
```

Reading into an array

```
int a[10][20];
for (row=0; row < 10; row = row+1){
   for(col=0; col < 20; col = col+1) {
      printf(“Enter a number: ");
      scanf("%d",&a[row][col]);
   }
}
```

```c
/* Fig. 6.21: fig06_21.c
   Initializing multidimensional arrays */
#include <stdio.h>

void printArray( const int a[][ 3 ] ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   /* initialize array1, array2, array3 */
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );
   return 0; /* indicates successful termination */
} /* end main */

```

Homework

- Read two matrices (with integer elements) from the input. Let their sizes be KxL and MxN.

- Write a function to multiply these to matrices, and print the resultant matrix (size KxN) to the screen with another function.

35) What is the output?

```
 int i,a[]={1,2,3,4,5},b[]={10,20,30,40,50};
for(i=1; i<5; i++)
   b[i]=a[i]+b[i-1];
for(i=0; i<5; i++)
   printf("%d ",b[i]);
```

a) 10 12 15 19 24 **b)** 1 12 23 34 45 **c)** 11 22 33 44 55
d) 10 32 53 74 95 **e)** 21 32 43 54 50

36) What is the output?

```
int i,a3[4];
  for(i=0; i<4; i++)
     a3[i]=i*2+1;
  for(i=0; i<3; i++)
     a3[i]=a3[i+1];
  a3[3]=a3[0];
  for(i=0; i<4; i++)
     printf("%d",a3[i]);
```

a) 1111 **b)** 3571 **c)** 3573 **d)** 5793 **e)** 7777

37) What is the output?

```
 int ar1[]={1,2,3,4,5};
  int ar2[3]={9};
  printf("%d,%d",ar1[1],ar2[1]);
```

a) 0,9 **b)** 1,9 **c)** 1,0 **d)** 2,0 **e)** 2,9

38) What is the output?

```
      int a[5]={11,1,16,-1,13};
      printf("%d",a[a[2]-a[4]]);
```

a) -1 **b)** 1 **c)** 2 **d)** 11 **e)** 16

39) What is the output?

```
      int j,c,x[]={5,-3,-1,7,8,-2,0,9,-6,8};
      for(c=0,j=0;j<10;j++)
      if (x[j]<0) c=c+j;
      printf("%d",c);
```

a) 0 **b)** 16 **c)** 20 **d)** 8 **e)** 4

40) Which one of the following declarations is wrong and causes a compile error?

a) `char a[]={'a',61,'9'};` **b)** `int b[5]={0,1};`

c) `float c[]={5.5,3};` **d)** `int d[]={};`

e) `double e[3]={7.8,1.0,3.5};`

43) What is the output?

```
void main() {
  int A[10][10]={{1,2,3},{3,4,5}};
  int B[10][10]={{2,2,2},{5,4,3}},C[10][10];
  int i,j, N=2, M=3;
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      if (A[i][j]>B[i][j])
       C[i][j]=A[i][j];
      else
      C[i][j]=B[i][j];
  for (i=0;i<N;i++)
  {
    for (j=0;j<M;j++)
    printf("%d ",C[i][j]);
    printf("\n");
  }
}
```

a)
```
1 2 3
3 4 5
```
b)
```
2 5
2 4
2 3
```
c)
```
1 2 3
5 4 3
```
d)
```
2 5
2 4
3 5
```
e)
```
2 2 3
5 4 5
```

44) What is the output?

```
 void main() {
  int A[10][10]={{1,2,3},{3,4,5}};
  int B[10][10]={{2,2,2},{5,4,3}},C[10][10];
  int i,j, N=2, M=3;
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      if (A[i][j]>B[i][j])
       C[j][i]=A[i][j];
      else
      C[j][i]=B[i][j];
  for (i=0;i<M;i++)
  {
    for (j=0;j<N;j++)
    printf("%d ",C[i][j]);
    printf("\n");
  }
}
```

a)
```
1 2 3
3 4 5
```

b)
```
2 5
2 4
2 3
```

c)
```
1 2 3
5 4 3
```

[✓ **d)**
```
2 5
2 4
3 5
```
]

e)
```
2 2 3
5 4 5
```

45) What is the output?

```
#include <stdio.h>
 void f3(int n, int a[]) {
   int i;
    for (i=0;i<n;i++)
      a[i]++;
    for (i=0;i<n;i++)
      printf("%d ",a[i]);
    printf("\n"); }
void main() {
  int A[]={1,2,3,3,4,5},B[10];
  int i, N=6;
  for (i=0;i<N;i++)
    B[i]=A[i];
  f3(N,A);
  for (i=0;i<N;i++)
     printf("%d ",A[i]);
  printf("\n");
  for (i=0;i<N;i++)
     printf("%d ",B[i]);
  printf("\n");  }
```

a) 1 2 3 3 4 5	[✓ **b)** 2 3 4 4 5 6	**c)** 1 2 3 3 4 5	**d)** 2 3 4 4 5 6	**e)** 2 3 4 4 5 6
1 2 3 3 4 5	2 3 4 4 5 6	2 3 4 4 5 6	1 2 3 3 4 5	2 3 4 4 5 6
1 2 3 3 4 5	1 2 3 3 4 5]	1 2 3 3 4 5	1 2 3 3 4 5	2 3 4 4 5 6

46) What is the output?

```
#include <stdio.h>
void f4(int n, int a[])
{
   int i; int b[10];
   for (i=0;i<n;i++)
      b[i]=a[i];
   for (i=0;i<n;i++)
    b[i]++;
   for (i=0;i<n;i++)
     printf("%d ",a[i]);
   printf("\n");
   for (i=0;i<n;i++)
     printf("%d ",b[i]);
  printf("\n");
}
void main() {
  int A[]={1,2,3,3,4,5};
  int i, N=6;
  f4(N,A);
  for (i=0;i<N;i++)
     printf("%d ",A[i]);
  printf("\n");
}
```

a) 1 2 3 3 4 5	**b)** 2 3 4 4 5 6	[✓ **c)** 1 2 3 3 4 5]	**d)** 2 3 4 4 5 6	**e)** 2 3 4 4 5 6
1 2 3 3 4 5	2 3 4 4 5 6	2 3 4 4 5 6	1 2 3 3 4 5	2 3 4 4 5 6
1 2 3 3 4 5	1 2 3 3 4 5	1 2 3 3 4 5	1 2 3 3 4 5	2 3 4 4 5 6

47) What is the output?

```
#include <stdio.h>
void main () {
 int A[10][10]={{1,2,3},{3,4,5},{5,6,7}};
 int i,j,N=3,t;
 for (i=0;i<N;i++)
 {
  t=A[i][i];
  A[i][i]=A[i][N-i-1];
  A[i][N-i-1]=t;
 }
 for (i=0;i<N;i++)
 {
  for (j=0;j<N;j++)
  printf("%d ",A[i][j]);
  printf("\n");
 }  }
```

```
a) 1 2 3     b) 5 2 7     c) 1 2 3     d) 1 6 3     e) 3 2 1
   3 4 5        3 4 5        5 4 3        3 4 5        3 4 5
   5 6 7        1 6 3        5 6 7        5 2 7        7 6 5
```

30) What is the output of the code below?

```
#include <stdio.h>
int main (void) {
        int b[3] [2]={1,2,3,4,5,6},i,j;
        for(i=0;i<2;++i) {
                printf("\n");
                for(j=0;j<3;j+=2)
                    printf ("%d", b[j][i]);
                }
        return 0; }
```

a) 12	**b)** 12	**c)** 135	**d)** 15	**e)** 246
56	34	246	26	135
	56			

31) What is the output of the code below?

```
#include <stdio.h>
int main (void) {
        int b[2][2]={{1},{2}},i,j;
        for(i=0;i<2;++i) {
                printf("\n");
                for(j=0;j<2;j++)
                  printf ("%d", b[i][j]);  }
         return 0; }
```

Note: (Assume, uninitialized elements are supposed to be zero)

a) 10	**b)** 00	**c)** 12	**d)** 11	**e)** 01
20	12	00	22	02

33) int x[10];

is given. Which of the array referencing is illegal?

a) x[(10%6)*4]=7; **b)** x[(10/3)-1]=7; **c)** x[(10/3)+1]=7; **d)** x[(10%3)-1]=7; **e)** x[(100%30)-1]=7.5;

38) What will be output if you will compile and execute the following c code?

```
#include <stdio.h>
        main (){
          int array[3][2][2]={0,1,2,3,4,5,6,7,8,9,10,11};
          printf("%d",array[1][1][1]);   }
```

a) 7 **b)** 8 **c)** 9 **d)** 10 **e)** 11

39) What is the output of the following program?

```
#include <stdio.h>
int compute(int [], int);
void main( ){
int a[]={6,7,8,9},i;
printf("%d", compute(a, 4));    }
int compute(int array[], int arraySize){
   int i, sum =0;
   for(i=0; i<arraySize; i++)
          if (array[i]%2==0) sum += array[i];
   return sum;  }
```

a) 6 **b)** 8 **c)** 14 **d)** 13 **e)** 16

41) Determine the output of the below program?

```
#include <stdio.h>
main ( ){
  int i, j, k, array[3][2][2]={1, 3, 5, 2, 4, 6, 7};
  for (i=0;i<2;++i)
    for (j=0;j<2;++j)
      for (k=0;k<2;++k)
   printf("%d  ", array[i][j][k]);
  return 0; }
```

a) 1 3 5
2 4 6
6 7

b) 1 3 5
2 4 6
6 7 0

c) 1 3
2 4

d) 1 3 5 2 4 6 7 0

e) 1 3 5 2 4 6

42) Determine the output of the below program?

```
#include <stdio.h>
int a=0, b=5, index=0, array[5] = {8,3,7,2,9};
calculate_a_b(){
  for( index=0; index<4; ++index )
    a = array[index];
  return 0;  }
main(){
  if (!a)
   if (a=++b)
     calculate_a_b();
   else b=8;
  else a++;
  printf("a=%d b=%d", a, b);
  return 0; }
```

a) a=2 b=6 **b)** a=6 b=2 **c)** a=0 b=5 **d)** a=0 b=8 **e)** a=0 b=0

44) Determine the output of the below program?

```
#include <stdio.h>
    main ( ){
      int i, j;
      int array[3][5]={0, 1, 2, 3, 4,
                              1, 2, 3, 4, 5,
                              2, 3, 4, 5, 6};
      for (i=0;i<=2;++i){
        for (j=0;j<5;++j)
          if (i==0||i==2)
                printf("%d", array[i][j]);
          else if (j==0) printf("%d", array[i][j]);
          else if (j==4) printf("%d", array[i][j]);
          else printf("  ");
      printf("\n");    }
     return 0; }
```

a)
```
01234
12345
23456
```

b)
```
01234
1      5
23456
```

c)
```
0, 1, 2, 3, 4
1, 2, 3, 4, 5
2, 3, 4, 5, 6
```

d)
```
0, 1, 2, 3, 4
1,          5
2, 3, 4, 5, 6
```

e) 012341   523456

40) Determine the output of the below program?

```
#include <stdio.h>
int compute(int, int);
int array[5][5];
int main (){
  compute(5, 5);
  return 0; }
int compute(int row, int column){
int  i, j, sum;
  for (i=0;i<row;++i)
     for (j=0;j<column;++j)
       if (i==j) array[i][j]=0;
       else if (i>j)  array[i][j]=3;
       else array[i][j]=2;
  array[2][2]=5;
  for (i=0;i<5;++i)
    for (j=0;j<5;++j)
      if (i==j) sum=array[i][j];
 printf("%d", sum);
return 0; }
```

a) 0 **b)**2 **c)** 3 **d)**14 **e)**15

Strings

Basics
Initialization
strcpy, strncpy,
strcat, strncat,
strcmp, strncmp functions

Declaring and Initializing String Variables

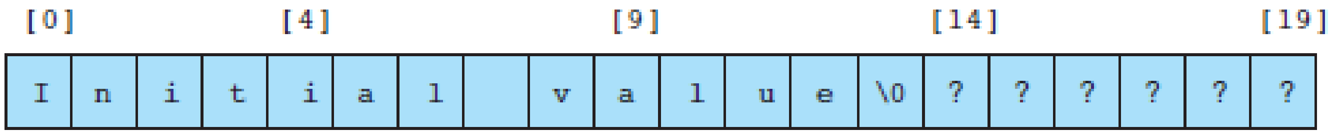
As we mentioned earlier, a string in C is implemented as an array, so declaring a string variable is the same as declaring an array of type `char`. In

```
char string_var[30];
```

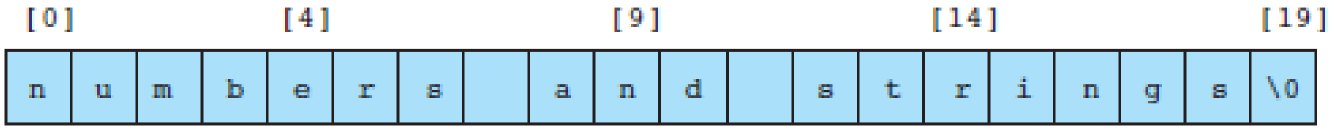
the variable `string_var` will hold strings from 0 to 29 characters long. It is C's handling of this varying length characteristic that distinguishes the string data structure from other arrays. C permits initialization of string variables using a string constant as shown in the following declaration of `str`.

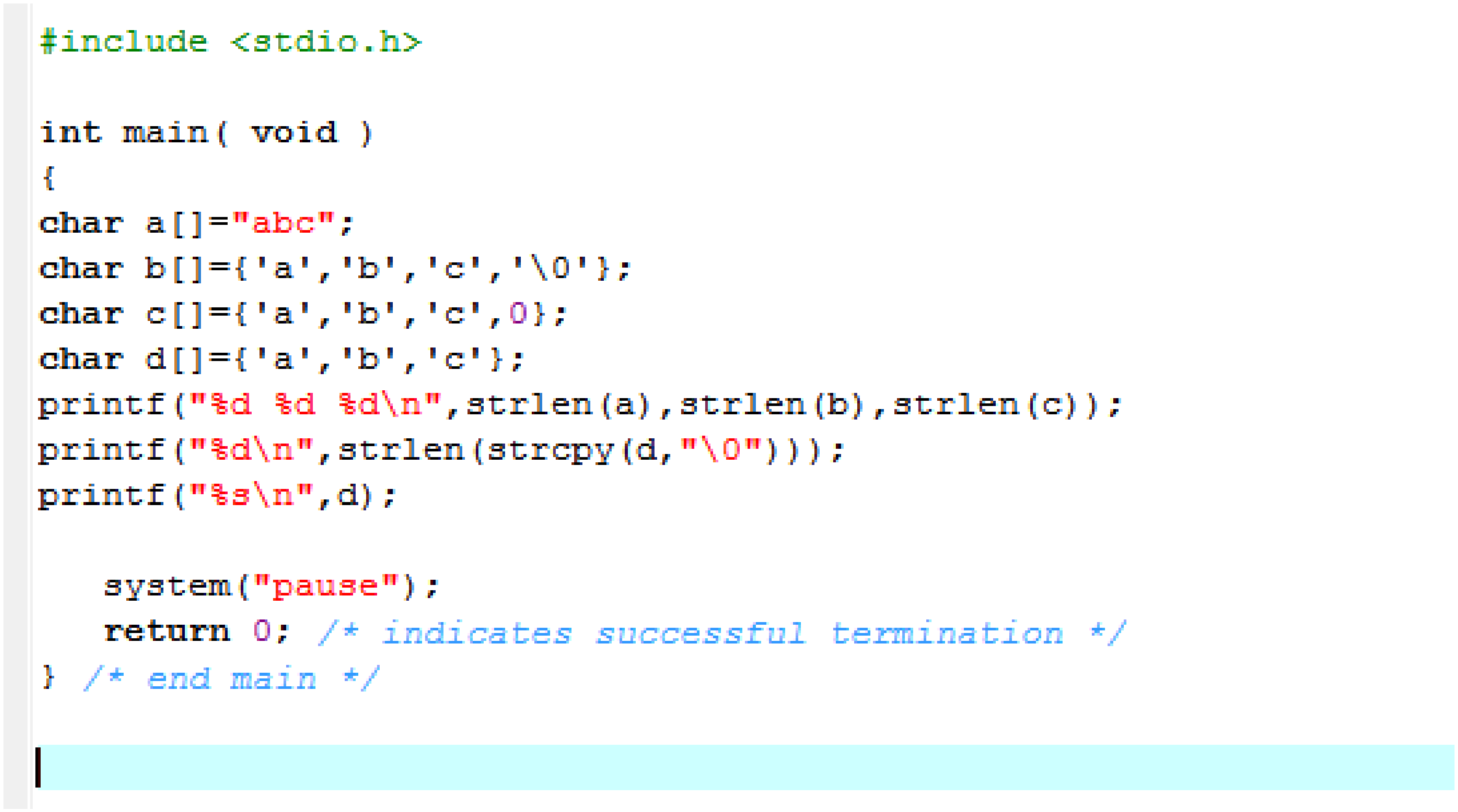
```
char str[20] = "Initial value";
```

Let's look at `str` in memory after this declaration with initialization.



Notice that `str[13]` contains the character `'\0'`, the **null character** that marks the end of a string. Using this marker allows the string's length within the character array to vary from 0 to one less than the array's declared size. All of C's string-handling functions simply ignore whatever is stored in the cells following the null character. The following diagram shows `str` holding a string that is the longest it can represent—19 characters plus the null character.





```c
#include <stdio.h>

int main( void )
{
char a[]="abc";
char b[]={'a','b','c','\0'};
char c[]={'a','b','c',0};
char d[]={'a','b','c'};
printf("%d %d %d\n",strlen(a),strlen(b),strlen(c));
printf("%d\n",strlen(strcpy(d,"\0")));
printf("%s\n",d);

   system("pause");
   return 0; /* indicates successful termination */
} /* end main */

```

Arrays of Strings

Because one string is an array of characters, an array of strings is a two-dimensional array of characters in which each row is one string. The following are statements to declare an array to store up to 30 names, each of which is less than 25 characters long.

```
#define NUM_PEOPLE 30
#define NAME_LEN 25
  . . .
char names[NUM_PEOPLE][NAME_LEN];
```

We can initialize an array of strings at declaration in the following manner:

```
char month[12][10] = {"January", "February", "March", "April",
                      "May", "June", "July", "August",
                      "September", "October", "November",
                      "December"};
```

String Assignment

Function `strcpy` copies the string that is its second argument into its first argument. To carry out the desired assignment shown in our faulty code above, we would write

```
strcpy(one_str, "Test String");
```

Like a call to `scanf` with a `%s` placeholder, a call to `strcpy` can easily overflow the space allocated for the destination variable (`one_str` in the example given). Variable `one_str` has room for up to 19 characters plus the null character. This call to `strcpy`

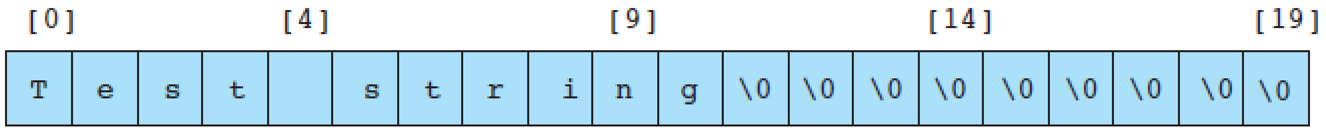
```
strcpy(one_str, "A very long test string");
```

would overflow `one_str`, storing the final characters `'i'`, `'n'`, `'g'`, and `'\0'` in memory allocated for other variables. The values of these other variables would seem to change spontaneously. On rare occasions, such overflow would generate a run-time error message.

The string library provides another string-copying function named `strncpy` that takes an argument specifying the number of characters to copy (call this number *n*). If the string to be copied (the source string) is shorter than *n* characters, the remaining characters stored are null. For example,

```
strncpy(one_str, "Test string", 20);
```

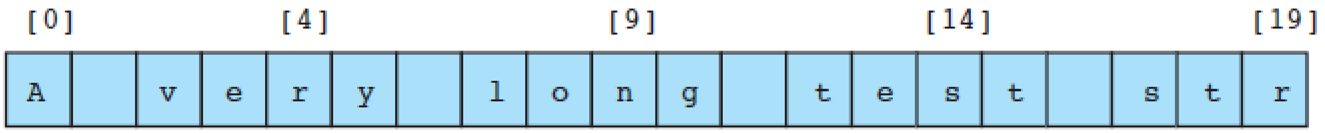
would give `one_str` the value:



The net effect is the same as the call

```
strcpy(one_str, "Test string");
```

```
strncpy(one_str, “A very long test string”, 20);
```



Notice that although this call to `strncpy` has prevented overflow of destination string `one_str`, it has not stored a valid string in `one_str`: There is no terminating `'\0'`. In general, one can assign as much as will fit of a source string (`source`) to a destination (`dest`) of length `dest_len` by using these two statements:

```
strncpy(dest, source, dest_len - 1);
dest[dest_len - 1] = '\0';
```

```
char one_str[20];
one_str = "Test string";      /* Does not work */
```

TABLE 8.1 Some String Library Functions from string.h

Function	Purpose: Example	Parameters	Result Type

`strcpy`	Makes a copy of `source`, a string, in the character array accessed by `dest`:
	`strcpy(s1, "hello");`
		`char *dest`
		`const char *source`
			`char *` | h | e | l | l | o | \0 | ? | ? | ... |

`strncpy`	Makes a copy of up to `n` characters from `source` in `dest`: `strncpy(s2, "inevitable", 5)` stores the first five characters of the source in `s1` and does NOT add a null character.	`char *dest`
		`const char *source`
		`size_t† n`
			`char *` | i | n | e | v | i | ? | ? | ... |

`strcat`	Appends `source` to the end of `dest`:
	`strcat(s1, "and more");`
		`char *dest`
		`const char *source`
			`char *` | h | e | l | l | o | a | n | d | | m | o | r | e | \0 |

`strncat`	Appends up to `n` characters of `source` to the end of `dest`, adding the null character if necessary:
	`strncat(s1, "and more", 5);`
		`char *dest`
		`const char *source`
		`size_t† n`
			`char *` | h | e | l | l | o | a | n | d | | m | \0 | ? |

`strcmp`	Compares `s1` and `s2` alphabetically; returns a negative value if `s1` should precede `s2`, a zero if the strings are equal, and a positive value if `s2` should precede `s1` in an alphabetized list:
	`if (strcmp(name1, name2) == 0)...`
		`const char *s1`
		`const char *s2`
			`int`

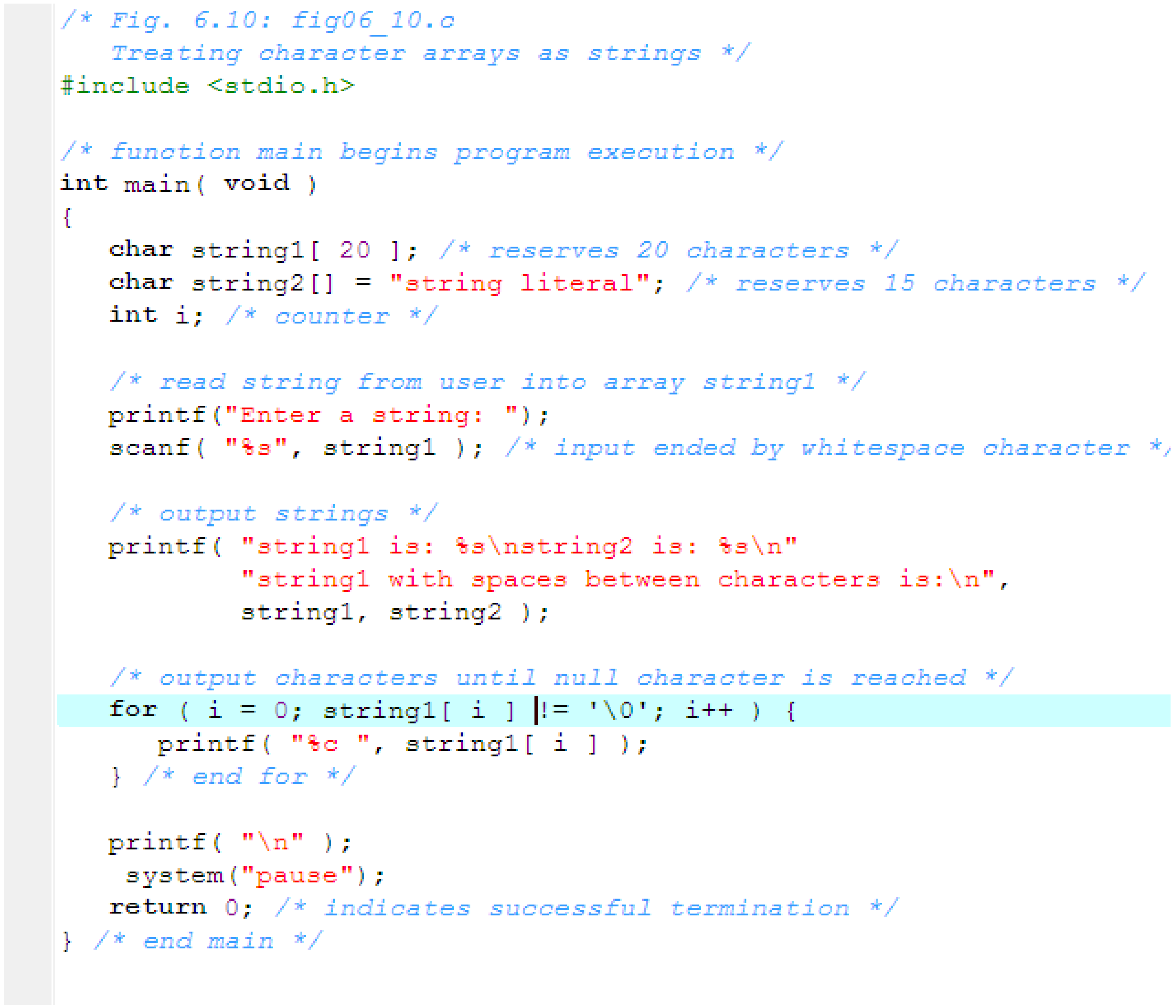
`strncmp`	Compares the first `n` characters of `s1` and `s2` returning positive, zero, and negative values as does `strcmp`:
	`if (strncmp(n1, n2, 12) == 0) ...`
		`const char *s1`
		`const char *s2`
		`size_t† n`
			`int`

`strlen`	Returns the number of characters in `s`, not counting the terminating null: `strlen("What")` returns `4`.	`const char *s`	`size_t`

Using Character Arrays to Store and Manipulate Strings

```c
char string1[] = "first";
```

```c
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```



```c
/* Fig. 6.10: fig06_10.c
   Treating character arrays as strings */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   char string1[ 20 ]; /* reserves 20 characters */
   char string2[] = "string literal"; /* reserves 15 characters */
   int i; /* counter */

   /* read string from user into array string1 */
   printf("Enter a string: ");
   scanf( "%s", string1 ); /* input ended by whitespace character */

   /* output strings */
   printf( "string1 is: %s\nstring2 is: %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   /* output characters until null character is reached */
   for ( i = 0; string1[ i ] != '\0'; i++ ) {
      printf( "%c ", string1[ i ] );
   } /* end for */

   printf( "\n" );
    system("pause");
   return 0; /* indicates successful termination */
} /* end main */
```

35. What would be the output of the following code segment?

```
char myStr[100];
printf(“Please enter a string:\n”);
scanf(“%s”, myStr);
printf(“%d”, strlen(myStr));
```

if the user has entered:

H	e	l	l	o		W	o	r	l	d	!

as input (each character entered is displayed in a box).

a) 5 b) 6 c) 11 [✓ d) 12] e) 13

q1

48) What is the output?

```
#include <stdio.h>
void main() {
   char a[]="abc";
   char b[]={'a','b','c','\0'};
   char c[]={'a','b','c',0};
   char d[]={'a','b','c'};
printf("%d %d %d\n",strlen(a),strlen(b),strlen(c));
   printf("%d\n",strlen(strcpy(d,"\0")));
   printf("%s\n",d);
}
```

a) 3 3 3
0

b) 3 4 4
4
abc

c) 4 4 4
4
abc

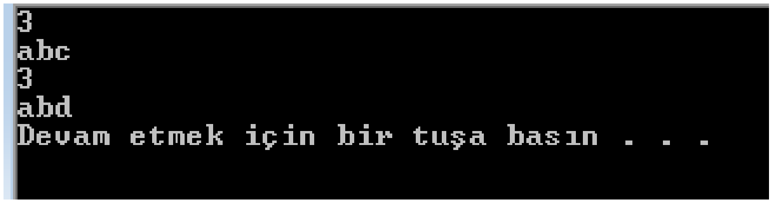
d) 3 3 4
3
abc

e) 3 4 3
4
abc

q2

49) What is the output?

```
 #include <stdio.h>
 void main () {
   char e[10],f[10];
   e[0]='a';e[1]='b';e[2]='\0';
   strcat(e,"c");
   printf("%d\n",strlen(e));
   printf("%s\n",e);
   strcpy(f,e);
   f[2]='d';
   printf("%d\n",strlen(f));
   printf("%s\n",f);
}
```



```
3
abc
3
abd
Devam etmek için bir tuşa basın . . .
```

q3

47) What is the output of the below code segment?

```
char str1[20] = "Hello";
char str2[20] = "World!";
strcat(str1, str2);
printf("%d", strlen(str1));
```

a) 1 **b)** 6 **c)** 5 **d)** 10 [✓ **e)** 11]

q5

50) What is the output of the following code segment?

```
char d[]= "need a hero";
printf("%d",strchr(d,'a')-d);
```

a) -4 **b)** 11 **c)** 6 **d)** 4 [✓ **e)** 5]

q6

42. What will be the output of the following code segment?

```
char st1[ ]="can you hear the voice?";
char st2[ ]="we must call the police!";
strcpy(st2+8,st1+8);
if(strncmp(st1+7,st2+7,8)==0) printf("%s",st2);
else printf("%s",st1);
```

a) "can you call the police!"
b) "can you hear the police!"
c) "we must call the police!"
d) "can you hear the voice?"
[✓ e) "we must hear the voice?"]

q7

50) What is the output?

```
   #include <stdio.h>
void main () {
 char e[10],f[10];
 strcpy(e,"abc");
 strcpy(f,"abd");
 if (strcmp(e,f))
 printf("%s\n",e);
else
printf("%s\n",f);
 f[3]='x';f[4]='\0';
 printf("%s\n",strchr(f,'d'));
 printf("%s\n",strstr(f,"d"));
}
```

a) abc	**b)** abd	[✓ **c)** abc	**d)** c	**e)** c
d	dx	dx	d	x
d	dx	dx]	d	x

43) What would be the output after implementation of the code?

```
char c[ ]= "a long string";
char s[20]= "It is my";
strncat(s,c+1,12);
 printf("%s",s);
```

a) "It is my long strin" **b)** "It is my long string"

c) Wrong output will be produced (missing endpoint of s)

d) Error (out of boundry) **e)** None of these.