Conditional operator

A very compact if-else.

variable = (condition) ? expression2 : expression3

means

if (condition)

      expression2

else

      expression3

Array Pointers
Chapter 7

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn how to declare and use arrays for storing collections of values of the same type

- To understand how to use a subscript to reference the individual values in an array

- To learn how to process the elements of an array in sequential order using loops

Chapter Objectives

- To understand how to pass individual array elements and entire arrays through function arguments
- To learn a method for searching an array
- To learn a method for sorting an array
- To learn how to use multidimensional arrays for storing tables of data
- To understand the concept of parallel arrays
- To learn how to declare and use your own data types

Basic Terminology

- data structure
  - a composite of related data items stored under the same name

- array
  - a collection of data items of the same type

Declaring and Referencing Arrays

- array element
  - a data item that is part of an array
- subscripted variable
  - a variable followed by a subscript in brackets, designating an array element
- array subscript
  - a value or expression enclosed in brackets after the array name, specifying which array element to access

```
double x[8];
```

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

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`.

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	28.0	26.0	2.5	12.0	14.0	−54.5

Array Initialization

int prime_lt_100 = {2, 3, 5, 7, 11, 13, 17, 19,

23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97}

char vowels = {'a', 'e', 'i', 'o', u'}

Array Subscripts

- Syntax:

  aname [subscript]

- Examples:

  x[3]

  x[i + 1]

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

Using for Loops for Sequential Access

for (i = 0; i < SIZE; ++i)

square[i] = i * i;

Array square

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	1	4	9	16	25	36	49	64	81	100

Statistical Computations

- Arrays are commonly used to store a collection of relation data values.

- Once the values are stored, you can perform simple statistical computations.

- The following program prints a table of differences.

FIGURE 7.2 Program to Print a Table of Differences

```
1.  /*
2.   * Computes the mean and standard deviation of an array of data and displays
3.   * the difference between each value and the mean.
4.   */
5.
6.  #include <stdio.h>
7.  #include <math.h>
8.
9.  #define MAX_ITEM 8    /* maximum number of items in list of data        */
10.
11. int
12. main(void)
13. {
14.     double x[MAX_ITEM],     /* data list                               */
15.            mean,            /* mean (average) of the data              */
16.            st_dev,          /* standard deviation of the data          */
17.            sum,             /* sum of the data                         */
18.            sum_sqr;         /* sum of the squares of the data          */
19.     int    i;
20.
21.       /* Gets the data                                                 */
22.       printf("Enter %d numbers separated by blanks or <return>s\n> ",
23.              MAX_ITEM);
24.       for   (i = 0; i < MAX_ITEM; ++i)
25.           scanf("%lf", &x[i]);
26.
27.       /* Computes the sum and the sum of the squares of all data       */
28.       sum = 0;
29.       sum_sqr = 0;
30.       for   (i = 0;   i < MAX_ITEM; ++i) {
31.           sum += x[i];
32.           sum_sqr += x[i] * x[i];
33.       }
```

(continued)

FIGURE 7.2   (continued)

```
34.         /* Computes and prints the mean and standard deviation          */
35.         mean = sum / MAX_ITEM;
36.         st_dev = sqrt(sum_sqr / MAX_ITEM - mean * mean);
37.         printf("The mean is %.2f.\n", mean);
38.         printf("The standard deviation is %.2f.\n", st_dev);
39.
40.         /* Displays the difference between each item and the mean         */
41.         printf("\nTable of differences between data values and mean\n");
42.         printf("Index        Item        Difference\n");
43.         for  (i = 0; i < MAX_ITEM; ++i)
44.             printf("%3d%4c%9.2f%5c%9.2f\n", i, ' ', x[i], ' ', x[i] - mean);
45.
46.         return (0);
47. }
```

```
Enter 8 numbers separated by blanks or <return>s
> 16  12  6  8  2.5  12  14  -54.5
The mean is 2.00.
The standard deviation is 21.75.

Table of differences between data values and mean
Index        Item        Difference
  0           16.00          14.00
  1           12.00          10.00
  2            6.00           4.00
  3            8.00           6.00
  4            2.50           0.50
  5           12.00          10.00
  6           14.00          12.00
  7          -54.50         -56.50
```

$$sum = \texttt{x[0]} + \texttt{x[1]} + \cdots + \texttt{x[6]} + \texttt{x[7]} = \sum_{\texttt{i=0}}^{\texttt{MAX\_ITEM} - 1} \texttt{x[i]}$$

$$sum\_sqr = \texttt{x[0]}^2 + \texttt{x[1]}^2 + \cdots + \texttt{x[6]}^2 + \texttt{x[7]}^2 = \sum_{\texttt{i=0}}^{\texttt{MAX\_ITEM} - 1} \texttt{x[i]}^2$$

$$standard\ deviation = \sqrt{\dfrac{\displaystyle\sum_{\texttt{i=0}}^{\texttt{MAX\_ITEM-1}} \texttt{x[i]}^2}{\texttt{MAX\_ITEM}} - mean^2}$$

TABLE 7.3 Partial Trace of Computing for Loop

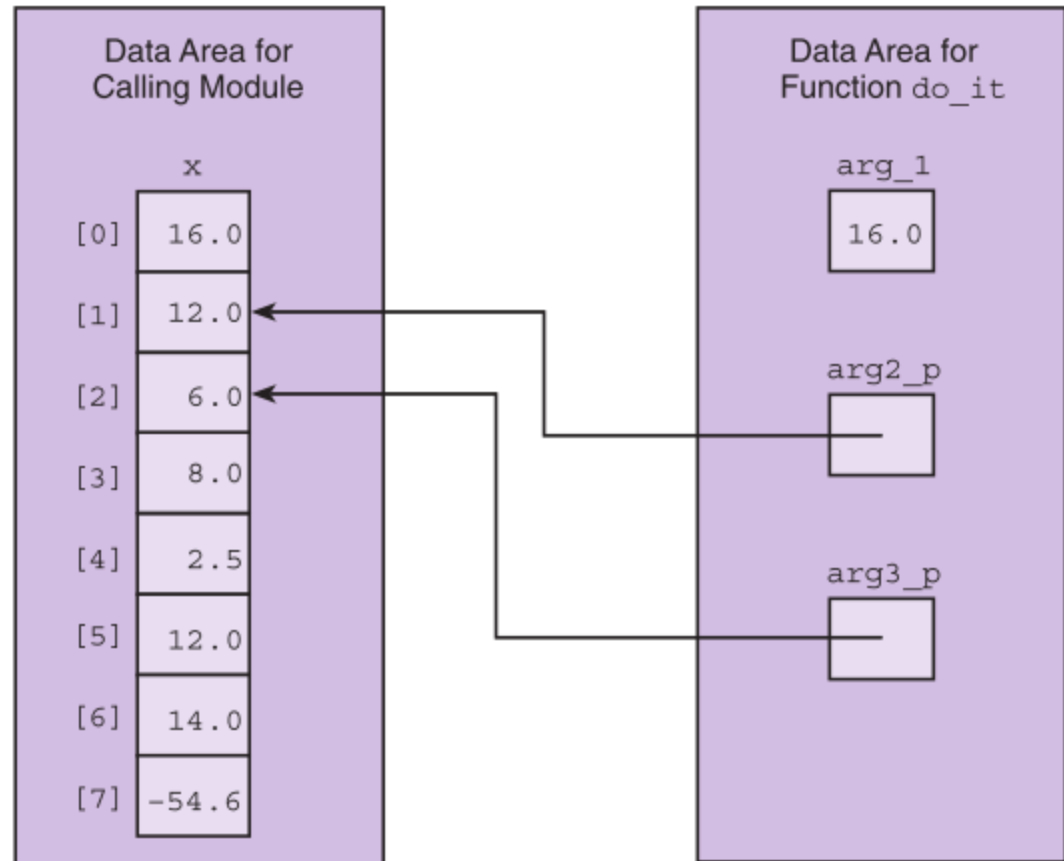| Statement	i	x[i]	sum	sum_sqr	Effect
`sum = 0;` `sum_sqr = 0;`			0.0	0.0	Initializes sum Initializes sum_sqr

`for (i = 0;`
`    i < MAX_ITEM;`
`    ++i)`
	0	16.0			Initializes i to 0
					which is less than 8

`sum += x[i];`
`sum_sqr +=`
`   x[i] * x[i];`
			16.0	256.0	Adds x[0] to sum
					
					Adds 256.0 to sum_sqr

increment and test i `sum += x[i];`	1	12.0	28.0		1 < 8 is true Adds x[1] to sum
`sum_sqr +=` `   x[i] * x[i];`				400.0	Adds 144.0 to sum_sqr
increment and test i `sum += x[i];`	2	6.0	34.0		2 < 8 is true Adds x[2] to sum
`sum_sqr +=` `   x[i] * x[i];`				436.0	Adds 36.0 to sum_sqr

Using Array Elements as Function Arguments

scanf("%lf", &x[i]);

do_it(x[0], &x[1], &x[2]);

FIGURE 7.3

Data Area for Calling Module and Function do_it

Data Area for Calling Module

x

[0] 16.0
[1] 12.0
[2] 6.0
[3] 8.0
[4] 2.5
[5] 12.0
[6] 14.0
[7] −54.6

Data Area for Function do_it

arg_1
16.0

arg2_p

arg3_p

Array Arguments

- We can write functions that have arrays as arguments.

- Such functions can manipulate some, or all, of the elements corresponding to an actual array argument.

FIGURE 7.4 Function fill_array

```
1.  /*
2.   * Sets all elements of its array parameter to in_value.
3.   * Pre: n and in_value are defined.
4.   * Post: list[i] = in_value, for 0 <= i < n.
5.   */
6.  void
7.  fill_array (int list[],     /* output - list of n integers              */
8.              int n,          /* input - number of list elements          */
9.              int in_value)   /* input - initial value                    */
10. {
11.
12.      int i;                 /* array subscript and loop control         */
13.
14.      for (i = 0; i < n; ++i)
15.          list[i] = in_value;
16. }
```

FIGURE 7.5

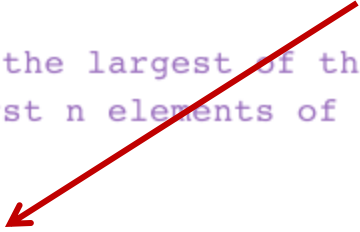Data Areas Before Return from `fill_array (x, 5, 1);`
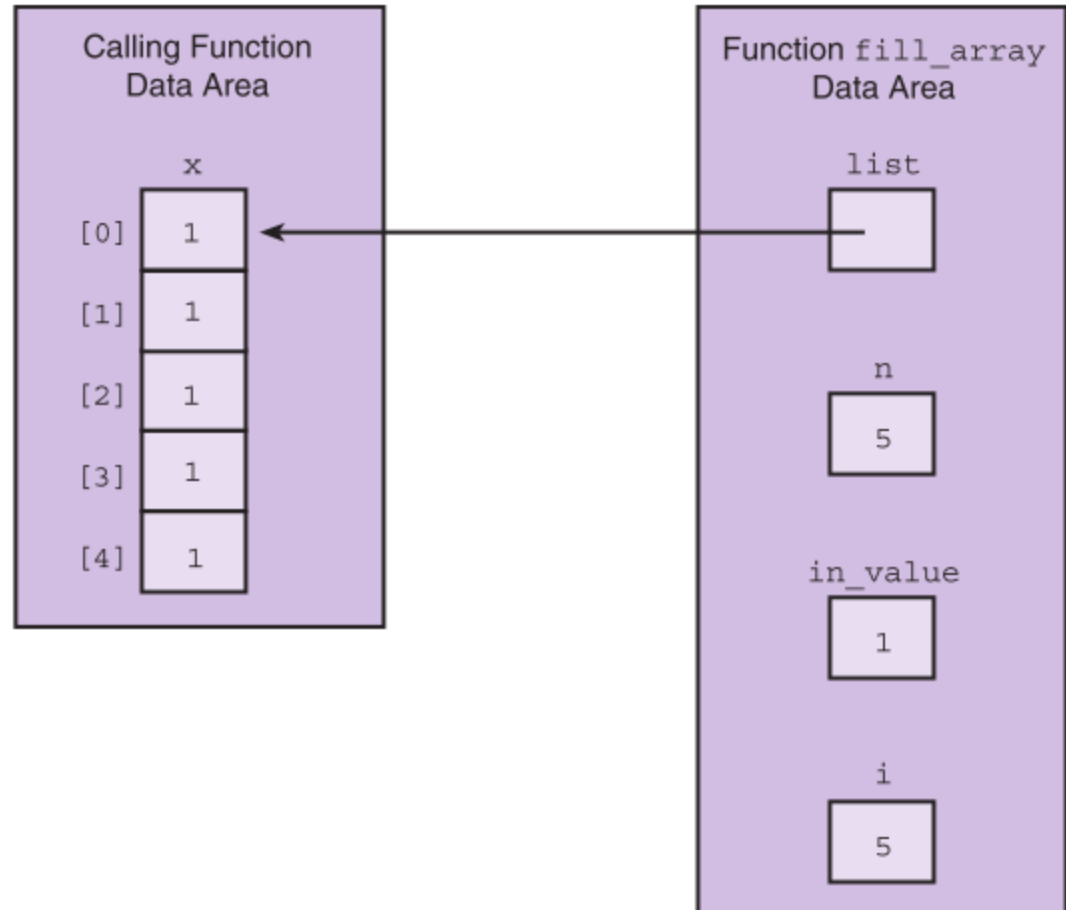
Arrays as Input Arguments

- ANSI C provides a qualifier that we can include in the declaration of the array formal parameter in order to notify the C compiler that the array is only an input to the function and the function does not intend to modify the array.

- The qualifier const allows the compiler to mark as an error any attempt to change an array element within the function.

FIGURE 7.6   Function to Find the Largest Element in an Array

```
1.  /*
2.   * Returns the largest of the first n values in array list
3.   * Pre: First n elements of array list are defined and n > 0
4.   */
5.  int
6.  get_max(const int list[], /* input - list of n integers              */
7.          int        n)     /* input - number of list elements to examine  */
8.  {
9.      int i,
10.         cur_large;       /* largest value so far                     */
11.
12.     /* Initial array element is largest so far.                     */
13.     cur_large = list[0];
14.
15.     /* Compare each remaining list element to the largest so far;
16.        save the larger                                              */
17.     for  (i = 1; i < n; ++i)
18.         if (list[i] > cur_large)
19.             cur_large = list[i];
20.
21.     return (cur_large);
22. }
```

Returning an Array Result

- In C, it is not legal for a function's return type to be an array.

- You need to use an output parameter to send your array back to the calling module.

FIGURE 7.7

Diagram of a Function That Computes an Array Result

input parameters → function → array result (output parameter)
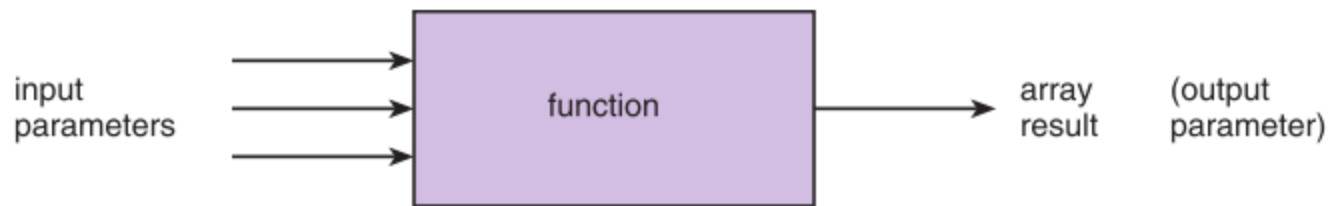
FIGURE 7.8 Function to Add Two Arrays

```
1.  /*
2.   * Adds corresponding elements of arrays ar1 and ar2, storing the result in
3.   * arsum. Processes first n elements only.
4.   * Pre: First n elements of ar1 and ar2 are defined. arsum's corresponding
5.   *      actual argument has a declared size >= n (n >= 0)
6.   */
7.  void
8.  add_arrays(const double ar1[],    /* input -                            */
9.             const double ar2[],    /* arrays being added                 */
10.            double       arsum[],  /* output - sum of corresponding
11.                                        elements of ar1 and ar2           */
12.            int          n)        /* input - number of element
13.                                        pairs summed                      */
14. {
15.     int i;
16.
17.     /* Adds corresponding elements of ar1 and ar2                        */
18.     for  (i = 0; i < n; ++i)
19.         arsum[i] = ar1[i] + ar2[i];
20. }
```

FIGURE 7.9

Function Data Areas for `add_arrays(x, y, x_plus_y, 5);`
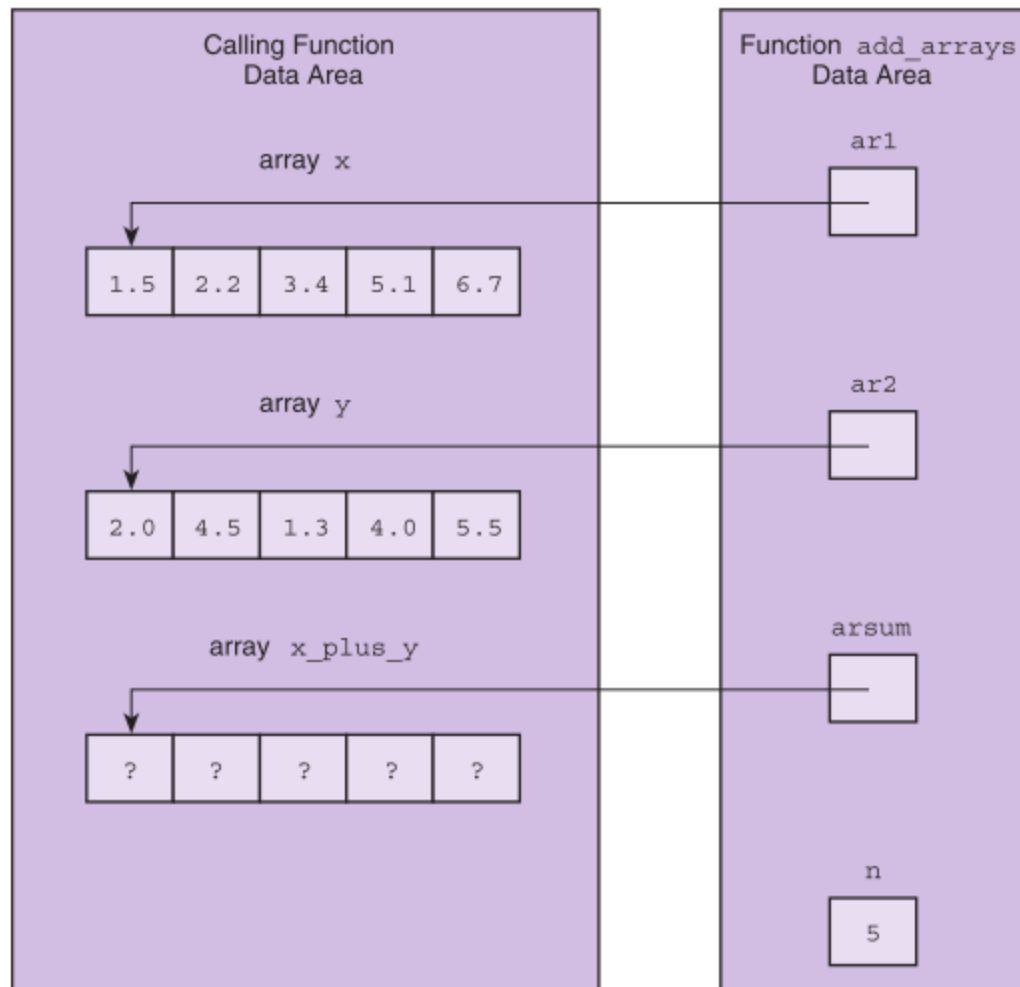
Partially Filled Arrays

- A program may need to process many lists of similar data but the lists may not all be the same length.

- In order to reuse an array for processing more than one data set, you ca declare an array large enough to hold the largest data set anticipated.

- Then your program should keep track of how many array elements are actually in use.

FIGURE 7.10

Diagram of
Function
fill_to_sentinel

dbl_max → fill_to_sentinel → dbl_arr

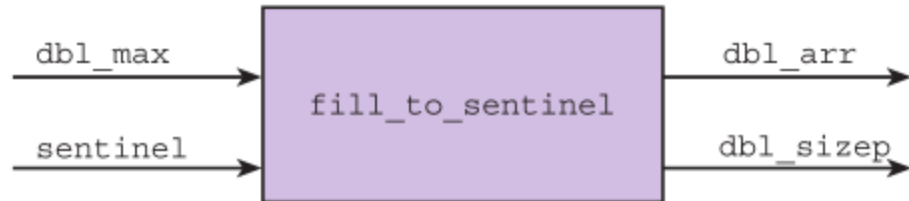sentinel → fill_to_sentinel → dbl_sizep

28

FIGURE 7.11 Function Using a Sentinel-Controlled Loop to Store Input Data in an Array

```
1.  /*
2.   * Gets data to place in dbl_arr until value of sentinel is encountered in
3.   * the input.
4.   * Returns number of values stored through dbl_sizep.
5.   * Stops input prematurely if there are more than dbl_max data values before
6.   * the sentinel or if invalid data is encountered.
7.   * Pre:  sentinel and dbl_max are defined and dbl_max is the declared size
8.   *          of dbl_arr
9.   */
10. void
11. fill_to_sentinel(int      dbl_max,    /* input - declared size of dbl_arr      */
12.                     double sentinel,  /* input - end of data value in
13.                                          input list                            */
14.                     double dbl_arr[], /* output - array of data                */
15.                     int    *dbl_sizep) /* output - number of data values
16.                                          stored in dbl_arr                     */
17. {
18.       double data;
19.       int    i, status;
20.
21.       /* Sentinel input loop                                                   */
22.       i = 0;
23.       status = scanf("%lf", &data);
24.       while (status == 1 && data != sentinel && i < dbl_max) {
25.           dbl_arr[i] = data;
26.           ++i;
27.           status = scanf("%lf", &data);
28.       }
29.
30.       /* Issues error message on premature exit                                */
31.       if (status != 1) {
32.             printf("\n*** Error in data format ***\n");
33.             printf("*** Using first %d data values ***\n", i);
```

(continued)

FIGURE 7.11 (continued)

```
34.        } else if (data != sentinel) {
35.              printf("\n*** Error: too much data before sentinel ***\n");
36.              printf("*** Using first %d data values ***\n", i);
37.        }
38.
39.        /* Sends back size of used portion of array                    */
40.        *dbl_sizep = i;
41. }
```

FIGURE 7.12 Driver for Testing fill_to_sentinel

```
1.  /* Driver to test fill_to_sentinel function */
2.
3.  #define A_SIZE 20
4.  #define SENT -1.0
5.
6.  int
7.  main(void)
8.  {
9.        double arr[A_SIZE];
10.       int    in_use,     /* number of elements of arr in use */
11.              i;
12.
13.       fill_to_sentinel(A_SIZE, SENT, arr, &in_use);
14.
15.       printf("List of data values\n");
16.       for  (i = 0; i < in_use; ++i)
17.           printf("%13.3f\n", arr[i]);
18.
19.       return (0);
20. }
```

Stacks

- A stack is a data structure in which only the top element can be accessed.

- pop
  - remove the top element of a stack

- push
  - insert a new element at the top of the stack

FIGURE 7.13 Functions push and pop

```
1.  void
2.  push(char stack[],      /* input/output - the stack */
3.        char item,        /* input - data being pushed onto the stack */
4.        int  *top,        /* input/output - pointer to top of stack */
5.        int  max_size)    /* input - maximum size of stack */
6.  {
7.      if (*top < max_size-1) {
8.          ++(*top);
9.          stack[*top] = item;
10.     }
11. }
12.
13. char
14. pop(char stack[],       /* input/output - the stack */
15.     int *top)           /* input/output - pointer to top of stack */
16. {
17.     char item;          /* value popped off the stack */
18.
19.     if (*top >= 0) {
20.         item = stack[*top];
21.         --(*top);
22.     } else {
23.         item = STACK_EMPTY;
24.     }
25.
26.     return (item);
27. }
```

Array Search

1. Assume the target has not been found.
2. Start with the initial array element.
3. repeat while the target is not found and there are more array elements
   4. if the current element matches the target
      5. Set a flag to indicate that the target has been found
   else
      6. Advance to the next array element.
7. if the target was found
   8. Return the target index as the search result
   else
   9. Return -1 as the search result.

FIGURE 7.14 Function That Searches for a Target Value in an Array

```
1.  #define NOT_FOUND -1      /* Value returned by search function if target not
2.                                found                                              */
3.
4.  /*
5.   * Searches for target item in first n elements of array arr
6.   * Returns index of target or NOT_FOUND
7.   * Pre:  target and first n elements of array arr are defined and n>=0
8.   */
9.  int
10. search(const int arr[],   /* input - array to search                            */
11.        int       target,  /* input - value searched for                         */
12.        int       n)        /* input - number of elements to search               */
13. {
14.     int i,
15.         found = 0,         /* whether or not target has been found               */
16.         where;             /* index where target found or NOT_FOUND              */
17.
18.     /* Compares each element to target                                            */
19.     i = 0;
20.     while (!found && i < n) {
21.         if (arr[i] == target)
22.             found = 1;
23.         else
24.             ++i;
25.     }
26.
27.     /* Returns index of element matching target or NOT_FOUND                       */
28.     if (found)
29.         where = i;
30.     else
31.         where = NOT_FOUND;
32.
33.     return (where);
34. }
```

Selection Sort

1. for each value of fill from 0 to n-2

    2. Find index_of_min, the index of the smallest element in the unsorted subarray list[fill] through list[n-1]

    3. if fill is not the position of the smallest element (index_of_min)

        4. Exchange the smallest element with the one at position fill.

FIGURE 7.15

Trace of Selection Sort

```
         [0]    [1]    [2]    [3]
        ┌─────┬─────┬─────┬─────┐
        │ 74  │ 45  │ 83  │ 16  │
        └─────┴─────┴─────┴─────┘
```

fill is 0. Find the smallest element in subarray list[1] through list[3] and swap it with list[0].

```
         [0]    [1]    [2]    [3]
        ┌─────┬─────┬─────┬─────┐
        │ 16  │ 45  │ 83  │ 74  │
        └─────┴─────┴─────┴─────┘
```

fill is 1. Find the smallest element in subarray list[1] through list[3]—no exchange needed.

```
         [0]    [1]    [2]    [3]
        ┌─────┬─────┬─────┬─────┐
        │ 16  │ 45  │ 83  │ 74  │
        └─────┴─────┴─────┴─────┘
```

fill is 2. Find the smallest element in subarray list[2] through list[3] and swap it with list[2].

```
         [0]    [1]    [2]    [3]
        ┌─────┬─────┬─────┬─────┐
        │ 16  │ 45  │ 74  │ 83  │
        └─────┴─────┴─────┴─────┘
```

FIGURE 7.16 Function select_sort

```
1.  /*
2.   * Finds the position of the smallest element in the subarray
3.   * list[first] through list[last].
4.   * Pre: first < last and elements 0 through last of array list are defined.
5.   * Post: Returns the subscript k of the smallest element in the subarray;
6.   *       i.e., list[k] <= list[i] for all i in the subarray
7.   */
8.  int get_min_range(int list[], int first, int last);
9.
10.
11. /*
12.  * Sorts the data in array list
13.  * Pre: first n elements of list are defined and n >= 0
14.  */
15. void
16. select_sort(int list[],   /* input/output - array being sorted       */
17.             int n)        /* input - number of elements to sort      */
18. {
19.     int fill,             /* index of first element in unsorted subarray  */
20.         temp,             /* temporary storage                            */
21.         index_of_min;     /* subscript of next smallest element           */
22.
23.     for (fill = 0; fill < n-1; ++fill) {
24.         /* Find position of smallest element in unsorted subarray */
25.         index_of_min = get_min_range(list, fill, n-1);
26.
27.         /* Exchange elements at fill and index_of_min */
28.         if (fill != index_of_min) {
29.             temp = list[index_of_min];
30.             list[index_of_min] = list[fill];
31.             list[fill] = temp;
32.         }
33.     }
34. }
```

Parallel Arrays

- two or more arrays with the same number of elements used for storing related information about a collection of data objects

id[0]	5503
id[1]	4556
id[2]	5691
. . .	
id[49]	9146

gpa[0]	2.71
gpa[1]	3.09
gpa[2]	2.98
. . .	
gpa[49]	1.92

FIGURE 7.17 Student Data in Parallel Arrays

```
1.   /* Read data for parallel arrays and echo stored data.                   */
2.
3.   #include <stdio.h>
4.   #define NUM_STUDENTS 50
5.
6.   int
7.   main(void)
8.   {
9.      int id[NUM_STUDENTS];
10.     double gpa[NUM_STUDENTS];
11.     int i;
12.
13.     for (i = 0; i < NUM_STUDENTS; ++i) {
14.        printf("Enter the id and gpa for student %d: ", i);
15.        scanf("%d%lf", &id[i], &gpa[i]);
16.        printf("%d   %4.2f\n", id[i], gpa[i]);
17.     }
```

(continued)

FIGURE 7.17 (continued)

```
18.
19.     return (0);
20.  }
```

```
Enter the id and gpa for student 0: 5503 2.71
5503    2.71
Enter the id and gpa for student 1: 4556 3.09
4556    3.09
```

Enumerated Types

- enumerated type
  - a data type whose list of values is specified by the programmer in a type declaration
- enumeration constant
  - an identifier that is one of the values of an enumerated type

```
typedef enum
        {Monday, Tuesday, Wednesday, Thursday,
         Friday, Saturday, Sunday}
day_t;
```

FIGURE 7.18 Enumerated Type for Budget Expenses

```
1.  /* Program demonstrating the use of an enumerated type */
2.
3.  #include <stdio.h>
4.
5.  typedef enum
6.         {entertainment, rent, utilities, food, clothing,
7.          automobile, insurance, miscellaneous}
8.  expense_t;
9.
10. void print_expense(expense_t expense_kind);
11.
12. int
13. main(void)
14. {
15.      expense_t expense_kind;
16.
17.      printf("Enter an expense code between 0 and 7>>");
18.      scanf("%d", &expense_kind);
19.      printf("Expense code represents ");
20.      print_expense(expense_kind);
21.      printf(".\n");
22.
23.      return (0);
24. }
25.
```

(continued)

```
26.  /*
27.   * Display string corresponding to a value of type expense_t
28.   */
29.  void
30.  print_expense(expense_t expense_kind)
31.  {
32.        switch (expense_kind) {
33.        case entertainment:
34.             printf("entertainment");
35.             break;
36.
37.        case rent:
38.             printf("rent");
39.             break;
40.
```

(continued)

FIGURE 7.18 (continued)

```
40.          case utilities:
41.                  printf("utilities");
42.                  break;
43.
44.          case food:
45.                  printf("food");
46.                  break;
47.
48.          case clothing:
49.                  printf("clothing");
50.                  break;
51.
52.          case automobile:
53.                  printf("automobile");
54.                  break;
55.
56.          case insurance:
57.                  printf("insurance");
58.                  break;
59.
60.          case miscellaneous:
61.                  printf("miscellaneous");
62.                  break;
63.
64.          default:
65.                  printf("\n*** INVALID CODE ***\n");
66.          }
67.  }
```

FIGURE 7.19

Arrays answer and score

answer[0]	T
answer[1]	F
answer[2]	F
	· · ·
answer[9]	T

score[monday]	9
score[tuesday]	7
score[wednesday]	5
score[thursday]	3
score[friday]	1

```
ascore = 9;
for  (today = monday; today <= friday; ++today) {
    score[today] = ascore;
    ascore -= 2;
}
```

Multidimensional Arrays

- **multidimensional array**
  - an array with two or more dimensions
- **multidimensional array parameter declaration**
  - only the first dimension can be omitted
    - char tictac[ ][3];

FIGURE 7.20

A Tic-tac-toe Board
Stored as Array
tictac

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

```
1.  /* Checks whether a tic-tac-toe board is completely filled.            */
2.  int
3.  filled(char ttt_brd[3][3])  /* input - tic-tac-toe board             */
4.  {
5.      int r, c, /* row and column subscripts   */
6.          ans;  /* whether or not board filled */
7.
8.      /* Assumes board is filled until blank is found                  */
9.      ans = 1;
10.
11.     /* Resets ans to zero if a blank is found                        */
12.     for (r = 0; r < 3; ++r)
13.        for  (c = 0; c < 3; ++c)
14.           if (ttt_brd[r][c] == ' ')
15.                ans = 0;
16.
17.     return (ans);
18. }
```

FIGURE 7.22

Three-Dimensional
Array enroll

Number of seniors (year 3)
taking course 0 at campus 2
enroll[0][2][3]
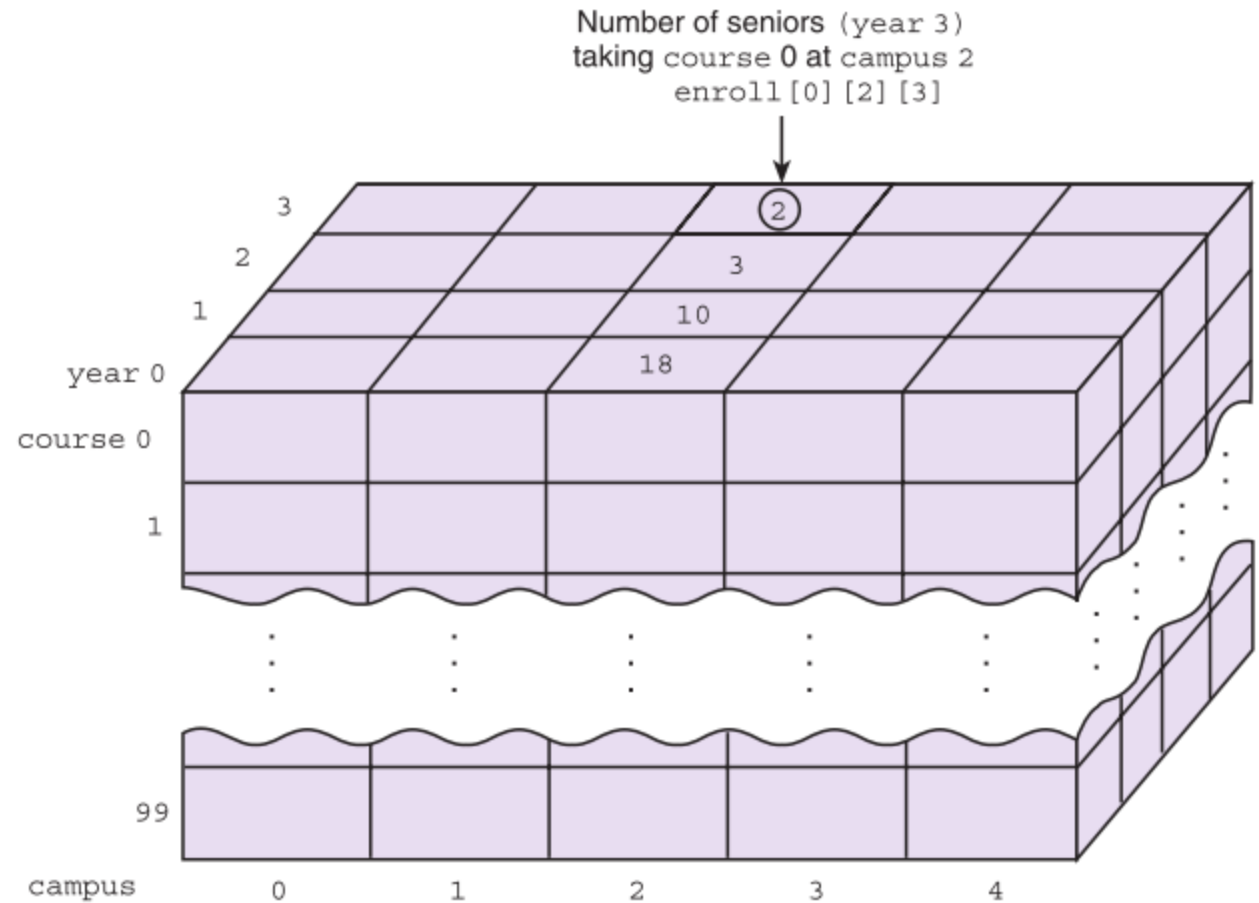
Graphics Programs with Arrays

- Drawing a Polygon
  - drawpoly
  - fillpoly

```
/*                    (x,    y)                                  */
int poly[12] = { 100, 200,     /* top-left corner        */
                 300, 100,     /* roof peak              */
                 500, 200,     /* top-right corner       */
                 500, 400,     /* bottom-right corner    */
                 100, 400,     /* bottom-left corner     */
                 100, 200      /* top-left corner        */
              };
drawpoly(6, poly);
```

FIGURE 7.27 Drawing and Filling a Polygon

```
1.   /* Draw and fill a polygon that is shaped like a house.
2.    * The color and fill pattern are selected randomly.
3.    */
4.   #include <graphics.h>
5.   #include <stdlib.h>
6.   #include <time.h>
7.
8.   int main()
9.   {
10.      /*                    (x ,   y)                      */
11.      int poly[12] = { 100, 200, /* top-left corner     */
12.                       300, 100, /* roof peak           */
13.                       500, 200, /* top-right corner     */
14.                       500, 400, /* bottom-right corner */
15.                       100, 400, /* bottom-left corner   */
16.                       100, 200, /* top-left corner     */
17.                     };
18.      char ch;
19.      int color;    /* color of house */
20.      int fill;     /* fill pattern of house */
21.      char message[] =
22.              "Press a key to repaint house - press Q to quit";
23.      initwindow(640, 500, "House");
24.      outtextxy(100, 450, message);
```

(continued)

```
26.      srand(time(NULL));
27.      ch = '*';
28.      while (!(ch == 'q' || ch == 'Q')) {
29.         color = rand() % 16;
30.         fill = rand() % 12;
31.         setfillstyle(fill, color);
32.         fillpoly(6, poly);
33.         ch = getch();
34.      }
35.
36.      closegraph();      /* close the window */
37. }
```

FIGURE 7.28

A Painted House

Press a key to repaint house - press Q to quit
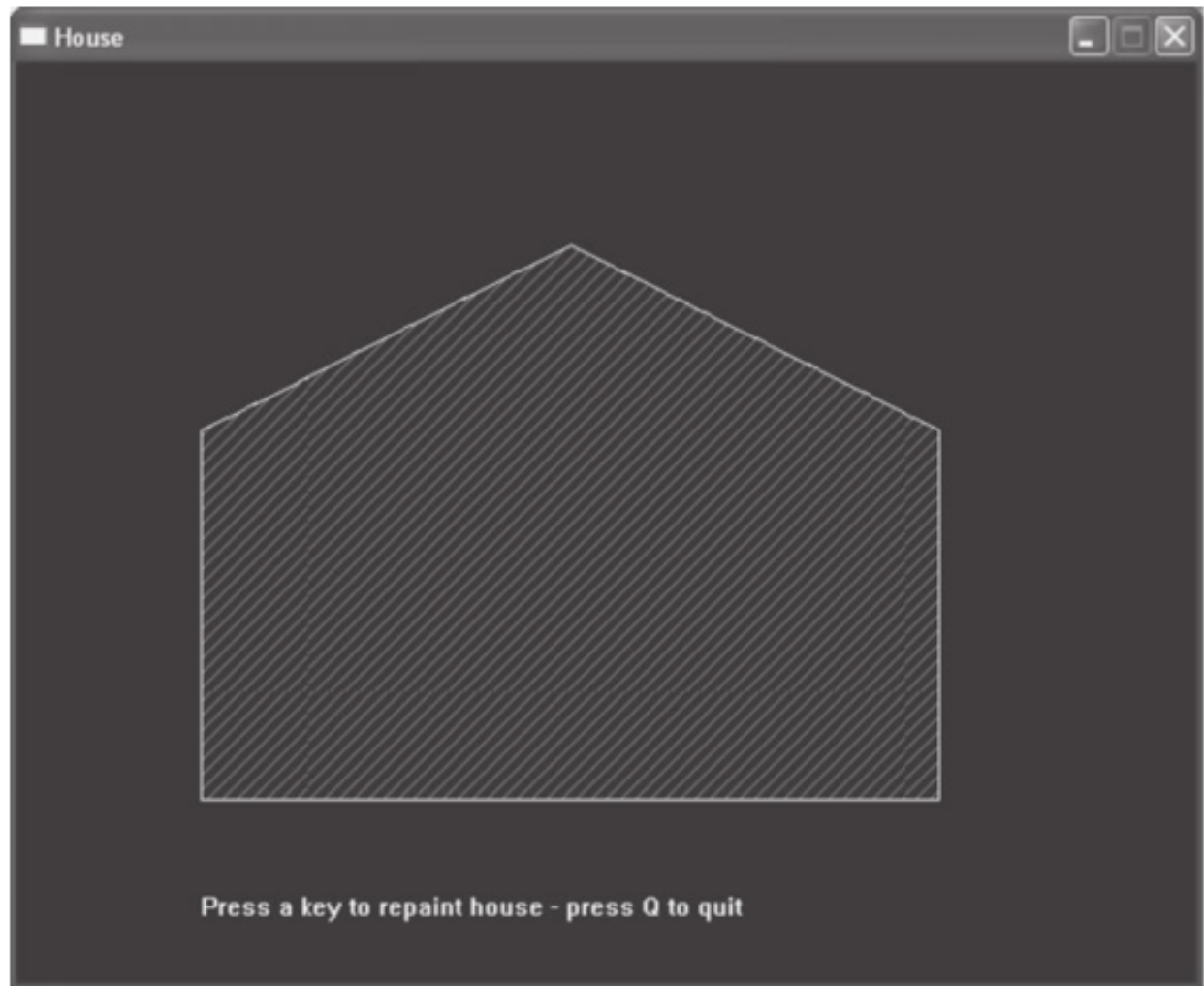
Generating Random Numbers

- Library stdlib provides a function rand that generates a pseudo-random integer.

- seed
  - see for a random number generator
  - an initial value used in the computation of the first random number
  - srand(time(NULL));

FIGURE 7.29   Program to Draw a Two-Dimensional Grid

```
1.  /* Draws a grid and displays the row, column
2.   * of a selected cell
3.   */
4.  #include <graphics.h>
5.  #include <stdio.h>
6.
7.  #define NUM_ROWS  5
8.  #define NUM_COLS  6
9.  #define CELL_SIZE 50 /* dimensions of a square cell */
10.
11. void drawGrid(int gridArray[][NUM_COLS]);
12. void reportResult(int *row, int *column, int *color);
13. void recolor(int r, int c, int color, int pattern);
14.
15. int main(void)
16. {
17.     int gridArray[NUM_ROWS][NUM_COLS] =
18.                           { {0, 0, 0, 0, 0, 0},
19.                             {2, 4, 2, 4, 2, 4},
20.                             {1, 2, 3, 4, 5, 6},
21.                             {2, 3, 4, 15, 6, 7},
22.                             {5, 5, 5, 5, 5, 5}
23.                           };
24.
25.     int width = CELL_SIZE * NUM_COLS;
26.     int height = CELL_SIZE * NUM_ROWS + 100;
27.     int row, column;   /* row, column of cell clicked */
28.     int color;               /* color of cell clicked */
29.     char message[100] = "Select a cell by clicking it";
30.
31.     initwindow(width, height, "Grid");
32.     outtextxy(100, height - 50, message);
33.
34.     drawGrid(gridArray);    /* Draw and color the grid */
35.
36.     /* Get position and color of cell clicked */
37.     reportResult(&row, &column, &color);
38.     printf("You clicked the cell in row %d, column %d\n",
39.             (row + 1), (column + 1));
40.
```

(continued)

FIGURE 7.29   (continued)

```
41.      /* Draw hatch pattern in cell clicked */
42.      recolor(row, column, color, HATCH_FILL);
43.
44.      getch();
45.      closegraph();
46.  }
47.
48.
49.  /* Draw the grid corresponding to the color values in gridArray
50.   * Pre:    Array gridArray is defined
51.   * Post:   A two-dimensional grid is drawn with the same number
52.   *         rows and columns as gridArray. The cell at row r,
53.   *         column c has the color value of gridArray[r][c].
54.   */
55.  void drawGrid(int gridArray[][NUM_COLS])
56.  {
57.      int color;                              /* color of cell */
58.      int r, c;                 /* row and column subscripts */
59.      /* Draw a grid cell for each array element */
60.      for (r = 0; r < NUM_ROWS; ++r) {
61.         for (c = 0; c < NUM_COLS; ++c) {
62.            /* Fill the cell */
63.            color = gridArray[r][c] % 16;    /* cell color */
64.            setfillstyle(SOLID_FILL, color);
65.            bar(c*CELL_SIZE, r*CELL_SIZE, /* top-left corner */
66.                c*CELL_SIZE + CELL_SIZE,      /* bottom-right */
67.                r*CELL_SIZE + CELL_SIZE);     /*     corner    */
68.
69.            /* Draw cell border */
70.            setcolor(WHITE);                        /* border color */
71.            rectangle(c*CELL_SIZE, r*CELL_SIZE, /* top-left corner */
72.                      c*CELL_SIZE + CELL_SIZE,      /* bottom-right */
73.                      r*CELL_SIZE + CELL_SIZE);     /*     corner    */
74.         }
75.      }
76.  }
77.
78.
79.  /* Report the results of a mouse click by the user.
80.   * Pre:    The grid is displayed
```

(continued)

FIGURE 7.29 (continued)

```
81.   * Post:    The row, column, and color of the cell
82.   *          clicked are returned through row and column.
83.   */
84.  void reportResult(int *row, int *column, int *color)
85.  {
86.     int x, y;    /* (x, y) position of mouse click */
87.
88.     clearmouseclick(WM_LBUTTONDOWN);
89.     while (!ismouseclick(WM_LBUTTONDOWN)) {
90.        delay(100);
91.     }
92.     getmouseclick(WM_LBUTTONDOWN, x, y);
93.
94.     /* Convert x (y) pixel position to row (column) */
95.     *row = y / CELL_SIZE;
96.     *column = x / CELL_SIZE;
97.     *color = getpixel(x, y);      /* get pixel color */
98.  }
99.
100. /* Recolors a selected grid cell with a new color and pattern.
101.  * Pre:    The cell position, color,
102.  *         and pattern are defined.
103.  * Post:   The selected cell is filled with pattern
104.  */
105. void recolor(int r, int c, int color, int pattern)
106. {
107.    setfillstyle(pattern, color);
108.    bar(c*CELL_SIZE, r*CELL_SIZE,    /* top-left corner */
109.        c*CELL_SIZE + CELL_SIZE,        /* bottom-right */
110.        r*CELL_SIZE + CELL_SIZE);       /*    corner    */
111. }
```

FIGURE 7.30

Two-Dimensional Grid
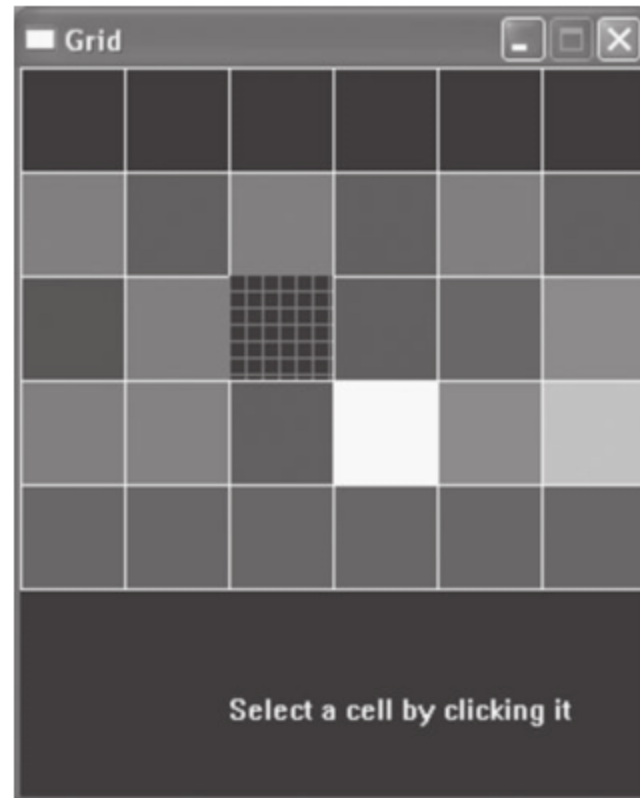
Wrap Up

- A data structure is a grouping of related data items in memory.

- An array is a data structure used to store a collection of data items of the same type.