



Lecture 6

MAC 125

Advanced C++
Programming

Arrays



Learning Objectives

- Introduction to Arrays
 - Declaring and referencing arrays
 - For-loops and arrays
 - Arrays in memory
- Arrays in Functions
 - Arrays as function arguments, return values
- Programming with Arrays
 - Partially Filled Arrays, searching, sorting
- Multidimensional Arrays



Introduction to Arrays

- Array definition:
 - A collection of data of same type
- First "aggregate" data type
 - Means "grouping"
 - int, float, double, char are simple data types
- Used for lists of like items
 - Test scores, temperatures, names, etc.
 - Avoids declaring multiple simple variables
 - Can manipulate "list" as one entity

Declaring Arrays

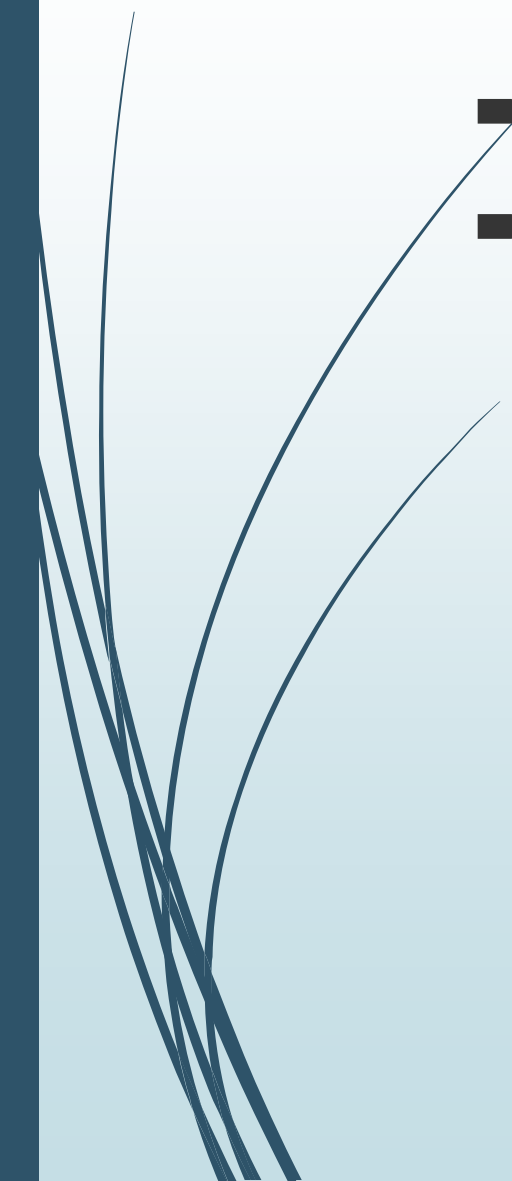
- Declare the array → allocates memory
`int score[5];`
 - Declares array of 5 integers named "score"
 - Similar to declaring five variables:
`int score[0], score[1], score[2], score[3], score[4]`
- Individual parts called many things:
 - Indexed or subscripted variables
 - "Elements" of the array
 - Value in brackets called index or subscript
 - Numbered from 0 to size - 1

Accessing Arrays

- Access using index/subscript
 - `cout << score[3];`
- Note two uses of brackets:
 - In declaration, specifies SIZE of array
 - Anywhere else, specifies a subscript
- Size, subscript need not be literal
 - `int score[MAX_SCORES];`
 - `score[n+1] = 99;`
 - If n is 2, identical to: `score[3]`



Array Usage

- Powerful storage mechanism
 - Can issue command like:
 - "Do this to i^{th} indexed variable"
where i is computed by program
 - "Display all elements of array score"
 - "Fill elements of array score from user input"
 - "Find highest value in array score"
 - "Find lowest value in array score"
- 

Array Program Example: Program Using an Array (1 of 2)

Display 5.1 Program Using an Array

```
1  //Reads in five scores and shows how much each
2  //score differs from the highest score.
3  #include <iostream>
4  using namespace std;
5  int main( )
6  {
7      int i, score[5], max;
8      cout << "Enter 5 scores:\n";
9      cin >> score[0];
10     max = score[0];
11     for (i = 1; i < 5; i++)
12     {
13         cin >> score[i];
14         if (score[i] > max)
15             max = score[i];
16         //max is the largest of the values score[0],..., score[i].
17     }
```

Array Program Example: Program Using an Array (2 of 2)

```
18     cout << "The highest score is " << max << endl
19         << "The scores and their\n"
20         << "differences from the highest are:\n";
21     for (i = 0; i < 5; i++)
22         cout << score[i] << " off by "
23             << (max - score[i]) << endl;
24     return 0;
25 }
```

SAMPLE DIALOGUE

Enter 5 scores:

5 9 2 10 6

The highest score is 10

The scores and their
differences from the highest are:

5 off by 5

9 off by 1

2 off by 8

10 off by 0

6 off by 4

for-loops with Arrays

- Natural counting loop
 - Naturally works well "counting through" elements of an array
- Example:

```
for (idx = 0; idx<5; idx++)  
{  
    cout << score[idx] << "off by "  
        << max - score[idx] << endl;  
}
```

 - Loop control variable (idx) counts from 0 – 5



Major Array Pitfall

- Array indexes always start with zero!
- Zero is "first" number to computer scientists
- C++ will "let" you go beyond range
 - Unpredictable results
 - Compiler will not detect these errors!
- Up to programmer to "stay in range"

Major Array Pitfall Example

- ▶ Indexes range from 0 to (array_size – 1)
 - ▶ Example:
double temperature[24]; // 24 is array size
// Declares array of 24 double values called temperature
 - ▶ They are indexed as:
temperature[0], temperature[1] ... temperature[23]
 - ▶ Common mistake:
temperature[24] = 5;
 - ▶ Index 24 is "out of range"!
 - ▶ No warning, possibly disastrous results

Defined Constant as Array Size

- Always use defined/named constant for array size
- Example:

```
const int NUMBER_OF_STUDENTS = 5;  
int score[NUMBER_OF_STUDENTS];
```
- Improves readability
- Improves versatility
- Improves maintainability

Uses of Defined Constant

- Use everywhere size of array is needed
 - In for-loop for traversal:

```
for (idx = 0; idx < NUMBER_OF_STUDENTS; idx++)  
{  
    // Manipulate array  
}
```
 - In calculations involving size:

```
lastIndex = (NUMBER_OF_STUDENTS - 1);
```
 - When passing array to functions (later)
- If size changes → requires only ONE change in program!

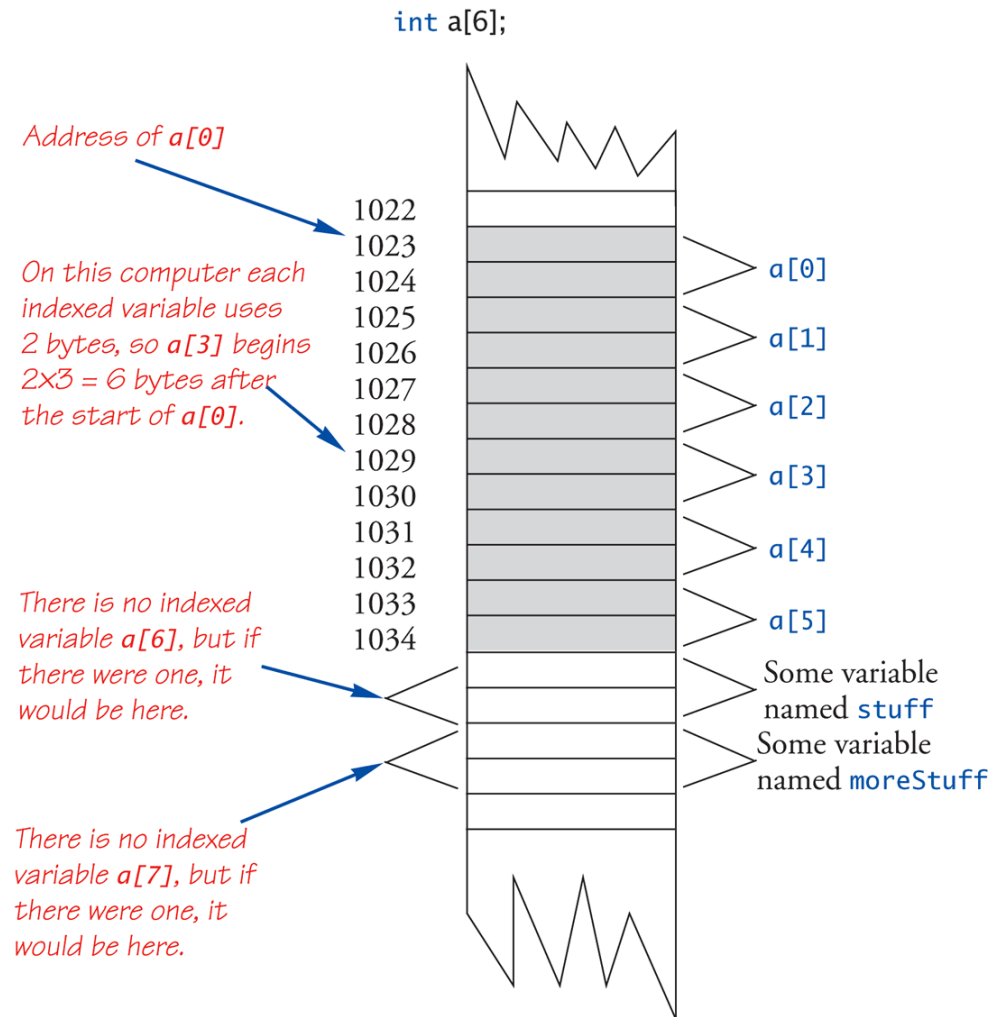


Arrays in Memory

- Recall simple variables:
 - Allocated memory in an "address"
- Array declarations allocate memory for entire array
- Sequentially-allocated
 - Means addresses allocated "back-to-back"
 - Allows indexing calculations
 - Simple "addition" from array beginning (index 0)

An Array in Memory

Display 5.2 An Array in Memory



Initializing Arrays

- As simple variables can be initialized at declaration:
`int price = 0; // 0 is initial value`
- Arrays can as well:
`int children[3] = {2, 12, 1};`
 - Equivalent to following:
`int children[3];`
`children[0] = 2;`
`children[1] = 12;`
`children[2] = 1;`

Auto-Initializing Arrays

- If fewer values than size supplied:
 - Fills from beginning
 - Fills "rest" with zero of array base type
- If array-size is left out
 - Declares array with size required based on number of initialization values
 - Example:
`int b[] = {5, 12, 11};`
 - Allocates array b to size 3



Arrays in Functions

- As arguments to functions
 - Indexed variables
 - An individual "element" of an array can be function parameter
 - Entire arrays
 - All array elements can be passed as "one entity"

Indexed Variables as Arguments

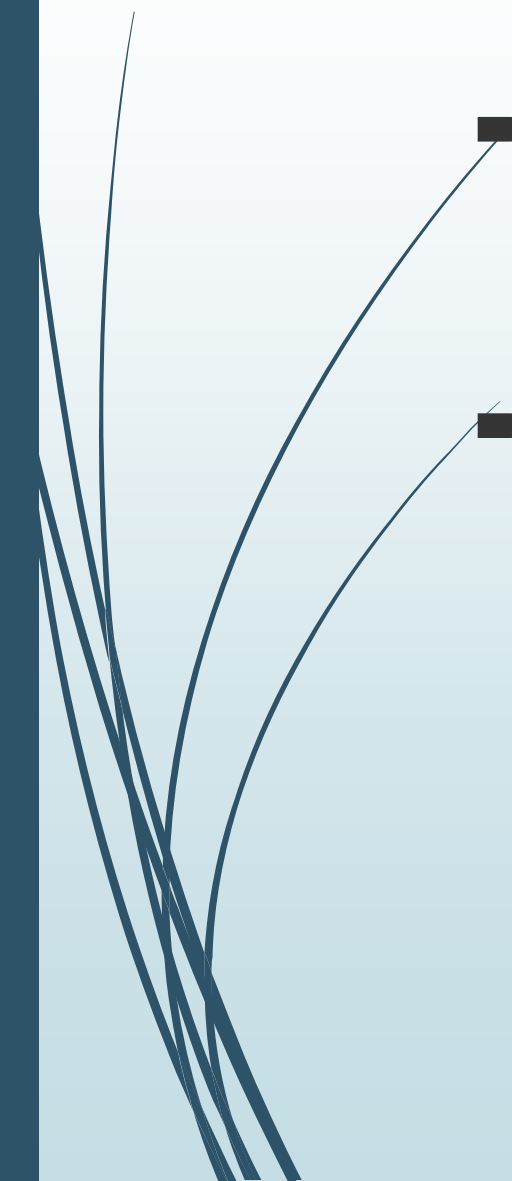
- ▶ Indexed variable handled same as simple variable of array base type
- ▶ Given this function declaration:
`void myFunction(double par1);`
- ▶ And these declarations:
`int i; double n, a[10];`
- ▶ Can make these function calls:
`myFunction(i); // i is converted to double`
`myFunction(a[3]); // a[3] is double`
`myFunction(n); // n is double`

Subtlety of Indexing

- Consider:
`myFunction(a[i]);`
 - Value of `i` is determined first
 - It determines which indexed variable is sent
- `myFunction(a[i*5]);`
- Perfectly legal, from compiler's view
- Programmer responsible for staying "in-bounds" of array



Entire Arrays as Arguments

- Formal parameter can be entire array
 - Argument then passed in function call is array name
 - Called "array parameter"
 - Send size of array as well
 - Typically done as second parameter
 - Simple int type formal parameter
- 

Entire Array as Argument Example: Function with an Array Parameter

Display 5.3 Function with an Array Parameter

SAMPLE DIALOGUEFUNCTION DECLARATION

```
void fillUp(int a[], int size);  
//Precondition: size is the declared size of the array a.  
//The user will type in size integers.  
//Postcondition: The array a is filled with size integers  
//from the keyboard.
```

SAMPLE DIALOGUEFUNCTION DEFINITION

```
void fillUp(int a[], int size)  
{  
    cout << "Enter " << size << " numbers:\n";  
    for (int i = 0; i < size; i++)  
        cin >> a[i];  
    cout << "The last array index used is " << (size - 1) << endl;  
}
```

Entire Array as Argument Example

- Given previous example:
- In some main() function definition, consider this calls:
int score[5], numberOfScores = 5;
fillup(score, numberOfScores);
 - 1st argument is entire array
 - 2nd argument is integer value
- Note no brackets in array argument!



Array as Argument: How?

- What's really passed?
- Think of array as 3 "pieces"
 - Address of first indexed variable (`arrName[0]`)
 - Array base type
 - Size of array
- Only 1st piece is passed!
 - Just the beginning address of array
 - Very similar to "pass-by-reference"



Array Parameters

- May seem strange
 - No brackets in array argument
 - Must send size separately
- One nice property:
 - Can use SAME function to fill any size array!
 - Exemplifies "re-use" properties of functions
 - Example:

```
int score[5], time[10];  
fillUp(score, 5);  
fillUp(time, 10);
```

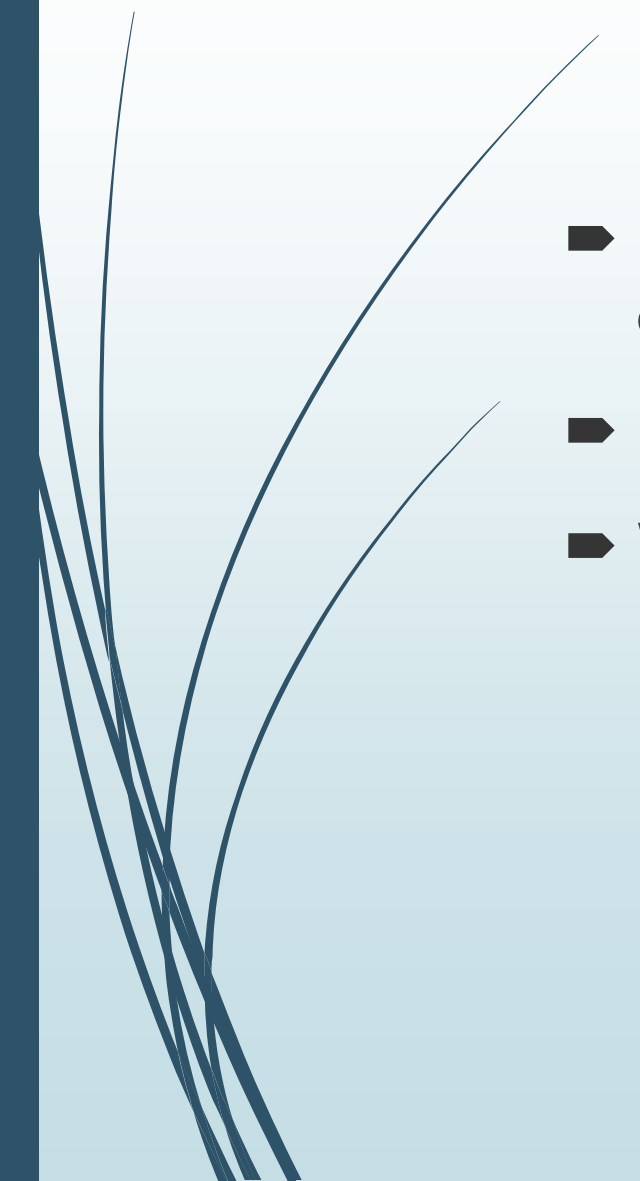


The const Parameter Modifier

- Recall: array parameter actually passes address of 1st element
 - Similar to pass-by-reference
- Function can then modify array!
 - Often desirable, sometimes not!
- Protect array contents from modification
 - Use "const" modifier before array parameter
 - Called "constant array parameter"
 - Tells compiler to "not allow" modifications

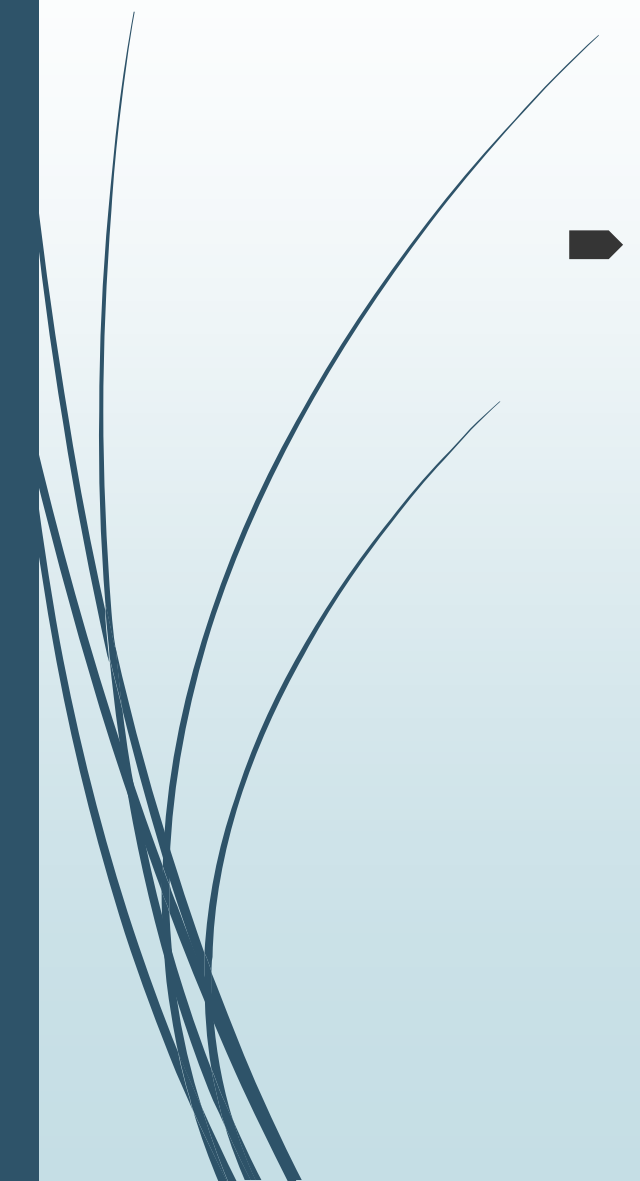


Functions that Return an Array

- Functions cannot return arrays same way simple types are returned
 - Requires use of a "pointer"
 - Will be discussed later...
- 



Programming with Arrays

- Plenty of uses
 - Sorting
 - Searching
- 



Global Constants vs. Parameters

- Constants typically made "global"
 - Declared above main()
- Functions then have scope to array size constant
 - No need to send as parameter then?
 - Technically yes
 - Why should we anyway?
 - Function definition might be in separate file
 - Function might be used by other programs!

Searching an Array (1 of 4)

Display 5.6 Searching an Array

```
1 //Searches a partially filled array of nonnegative integers.
2 #include <iostream>
3 using namespace std;
4 const int DECLARED_SIZE = 20;

5 void fillArray(int a[], int size, int& numberUsed);
6 //Precondition: size is the declared size of the array a.
7 //Postcondition: numberUsed is the number of values stored in a.
8 //a[0] through a[numberUsed-1] have been filled with
9 //nonnegative integers read from the keyboard.

10 int search(const int a[], int numberUsed, int target);
11 //Precondition: numberUsed is <= the declared size of a.
12 //Also, a[0] through a[numberUsed -1] have values.
13 //Returns the first index such that a[index] == target,
14 //provided there is such an index; otherwise, returns -1.
```

Searching an Array (2 of 4)

```
15  int main( )
16  {
17      int arr[DECLARED_SIZE], listSize, target;
18
19      fillArray(arr, DECLARED_SIZE, listSize);
20
21      char ans;
22      int result;
23      do
24      {
25          cout << "Enter a number to search for: ";
26          cin >> target;
27
28          result = search(arr, listSize, target);
29          if (result == -1)
30              cout << target << " is not on the list.\n";
31          else
32              cout << target << " is stored in array position "
33                  << result << endl
34                  << "(Remember: The first position is 0.)\n";
```

Searching an Array (3 of 4)

Display 5.6 Searching an Array

```
32         cout << "Search again?(y/n followed by Return): ";
33         cin >> ans;
34     } while ((ans != 'n') && (ans != 'N'));
35     cout << "End of program.\n";
36     return 0;
37 }

38 void fillArray(int a[], int size, int& numberUsed)
39 <The rest of the definition of fillArray is given in Display 5.5>
40 int search(const int a[], int numberUsed, int target)
41 {
42     int index = 0;
43     bool found = false;
44     while ((!found) && (index < numberUsed))
45     if (target == a[index])
46         found = true;
47     else
48         index++;
```


Searching an Array (4 of 4)

```
49     if (found)
50         return index;
51     else
52         return -1;
53 }
```

SAMPLE DIALOGUE

Enter up to 20 nonnegative whole numbers.

Mark the end of the list with a negative number.

10 20 30 40 50 60 70 80 -1

Enter a number to search for: **10**

10 is stored in array position 0

(Remember: The first position is 0.)

Search again?(y/n followed by Return): **y**

Enter a number to search for: **40**

40 is stored in array position 3

(Remember: The first position is 0.)

Search again?(y/n followed by Return): **y**

Enter a number to search for: **42**

42 is not on the list.

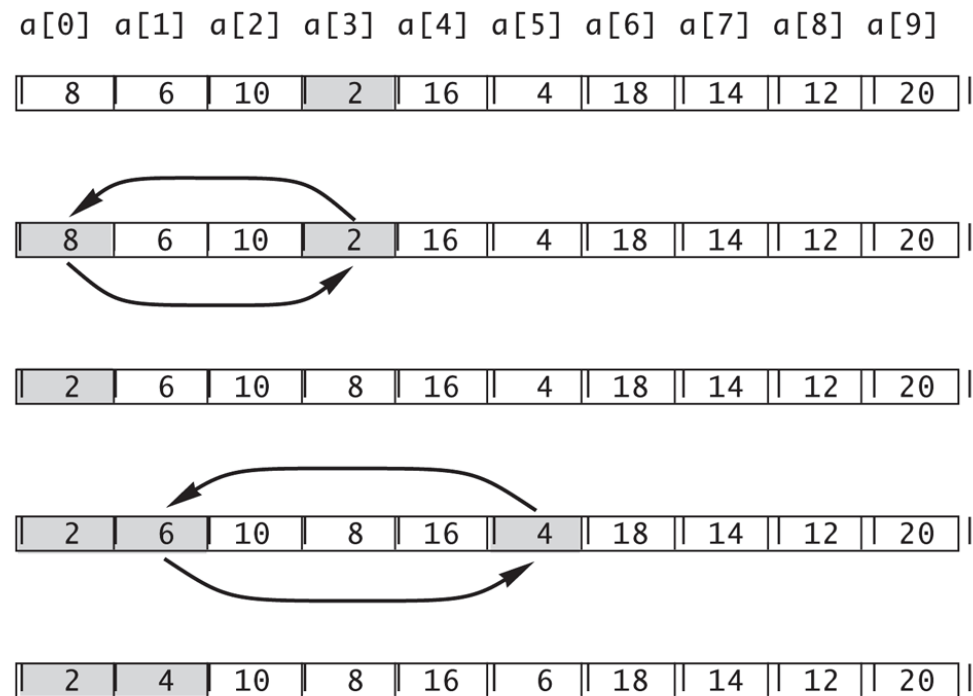
Search again?(y/n followed by Return): **n**

End of program.

Sorting an Array: Selection Sort

➤ Selection Sort Algorithm

Display 5.7 Selection Sort



Sorting an Array Example:

Sorting an Array (1 of 4)

Display 5.8 **Sorting an Array**

```
1  //Tests the procedure sort.
2  #include <iostream>
3  using namespace std;

4  void fillArray(int a[], int size, int& numberUsed);
5  //Precondition: size is the declared size of the array a.
6  //Postcondition: numberUsed is the number of values stored in a.
7  //a[0] through a[numberUsed - 1] have been filled with
8  //nonnegative integers read from the keyboard.
9  void sort(int a[], int numberUsed);
10 //Precondition: numberUsed <= declared size of the array a.
```

(continued)

Sorting an Array Example:

Sorting an Array (2 of 4)

Display 5.8 Sorting an Array

```
11 //The array elements a[0] through a[numberUsed - 1] have values.
12 //Postcondition: The values of a[0] through a[numberUsed - 1] have
13 //been rearranged so that a[0] <= a[1] <= ... <= a[numberUsed - 1].

14 void swapValues(int& v1, int& v2);
15 //Interchanges the values of v1 and v2.

16 int indexOfSmallest(const int a[], int startIndex, int numberUsed);
17 //Precondition: 0 <= startIndex < numberUsed. Reference array elements
18 //have values. Returns the index i such that a[i] is the smallest of the
19 //values a[startIndex], a[startIndex + 1], ..., a[numberUsed - 1].

20 int main( )
21 {
22     cout << "This program sorts numbers from lowest to highest.\n";

23     int sampleArray[10], numberUsed;
24     fillArray(sampleArray, 10, numberUsed);
25     sort(sampleArray, numberUsed);

26     cout << "In sorted order the numbers are:\n";
27     for (int index = 0; index < numberUsed; index++)
28         cout << sampleArray[index] << " ";
29     cout << endl;

30     return 0;
31 }
```

Sorting an Array Example:

Sorting an Array (3 of 4)

```
32 void fillArray(int a[], int size, int& numberUsed)
33     <The rest of the definition of fillArray is given in Display 5.5.>

34 void sort(int a[], int numberUsed)
35 {
36     int indexOfNextSmallest;
37     for (int index = 0; index < numberUsed - 1; index++)
38         { //Place the correct value in a[index]:
39             indexOfNextSmallest =
40                 indexOfSmallest(a, index, numberUsed);
41             swapValues(a[index], a[indexOfNextSmallest]);
42             //a[0] <= a[1] <= ... <= a[index] are the smallest of the original array
43             //elements. The rest of the elements are in the remaining positions.
44         }
45 }

46 void swapValues(int& v1, int& v2)
47 {
48     int temp;
49     temp = v1;
50     v1 = v2;
```

Sorting an Array Example:

Sorting an Array (4 of 4)

Display 5.8 Sorting an Array

```
51     v2 = temp;
52 }
53
54 int indexOfSmallest(const int a[], int startIndex, int numberUsed)
55 {
56     int min = a[startIndex],
57     indexOfMin = startIndex;
58     for (int index = startIndex + 1; index < numberUsed; index++)
59         if (a[index] < min)
60         {
61             min = a[index];
62             indexOfMin = index;
63             //min is the smallest of a[startIndex] through a[index]
64         }
65
66     return indexOfMin;
67 }
```

SAMPLE DIALOGUE

This program sorts numbers from lowest to highest.

Enter up to 10 nonnegative whole numbers.

Mark the end of the list with a negative number.

80 30 50 70 60 90 20 30 40 -1

In sorted order the numbers are:

20 30 30 40 50 60 70 80 90

Multidimensional Arrays

- Arrays with more than one index

- `char page[30][100];`

- Two indexes: An "array of arrays"

- Visualize as:

- `page[0][0], page[0][1], ..., page[0][99]`

- `page[1][0], page[1][1], ..., page[1][99]`

- ...

- `page[29][0], page[29][1], ..., page[29][99]`

- C++ allows any number of indexes

- Typically no more than two

Multidimensional Array Parameters

- Similar to one-dimensional array

- 1st dimension size not given

- Provided as second parameter

- 2nd dimension size IS given

- Example:

```
void DisplayPage(const char p[][100], int sizeDimension1)
{
    for (int index1=0; index1<sizeDimension1; index1++)
    {
        for (int index2=0; index2 < 100; index2++)
            cout << p[index1][index2];
        cout << endl;
    }
}
```




Summary 1

- Array is collection of "same type" data
- Indexed variables of array used just like any other simple variables
- for-loop "natural" way to traverse arrays
- Programmer responsible for staying "in bounds" of array
- Array parameter is "new" kind
 - Similar to call-by-reference



Summary 2

- Array elements stored sequentially
 - "Contiguous" portion of memory
 - Only address of 1st element is passed to functions
- Partially-filled arrays → more tracking
- Constant array parameters
 - Prevent modification of array contents
- Multidimensional arrays
 - Create "array of arrays"