C++ Programming:
Arrays

Some material taken from: C++ Programming: Program Design Including Data Structures

Motivation

- Consider the following problems:
  - Read a large list of numbers and print it in reverse order
  - Sort a large list of numbers
- Need to store a large list of number in memory and manipulate it
- Arrays

Objectives

- Learn about arrays
- Explore how to declare and manipulate data into arrays
- Array reverse problem
- Sorting problem: Selection Sort

Data Types

- A data type is called simple if variables of that type can store only one value at a time
- A structured data type is one in which each data item is a collection of other data items

Arrays

- <u>Array</u> - a collection of a fixed number of components wherein all of the components have the same data type
- One-dimensional array - an array in which the components are arranged in a list form
- The general form of declaring a one-dimensional array is:

```
dataType arrayName[intExp];
```

where intExp is any expression that evaluates to a constant positive integer

Declaring an array

- The statement

  int num[5];

  declares an array num of 5 components of the type int
- The components are num[0], num[1], num[2], num[3], and num[4]

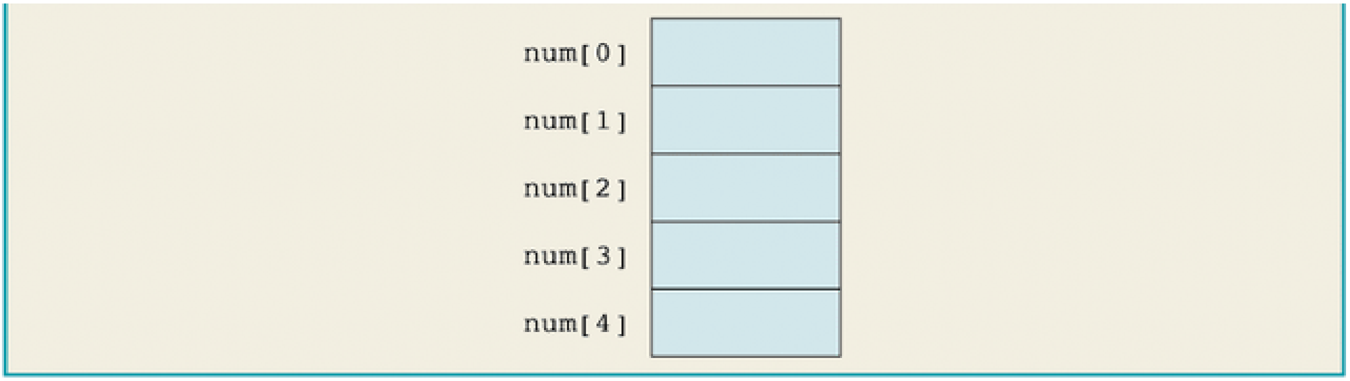

Figure 9-1 Array num

Accessing Array Components

- The general form (syntax) of accessing an array component is:

  arrayName[indexExp]

  where indexExp, called index, is any expression whose value is a nonnegative integer
- Index value specifies the position of the component in the array
- The [] operator is called the array subscripting operator
- The array index always starts at 0

Processing One-Dimensional Arrays

- Some basic operations performed on a one-dimensional array are:
  - Initialize
  - Input data
  - Output data stored in an array
  - Find the largest and/or smallest element
- Each operation requires ability to step through the elements of the array
- Easily accomplished by a loop

Accessing Array Components (continued)

- Consider the declaration

```
int list[100];          //list is an array of the size 100
int i;
```

- This for loop steps-through each element of the array list starting at the first element

```
for(i = 0; i < 100; i++)        //Line 1
    process list[i]             //Line 2
```

Accessing Array Components (continued)

- If processing list requires inputting data into list
  - the statement in Line 2 takes the from of an input statement, such as the cin statement

```
for(i = 0; i < 100; i++)        //Line 1
    cin>>list[i];
```

Array Index Out of Bounds

- If we have the statements:

  double num[10];

  int i;
- The component num[i] is a valid index if i = 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9
- The index of an array is in bounds if the index >=0 and the index <= arraySize-1
- Otherwise, it is out of bounds
- There is no automatic guard against indices that are out of bounds

Array Initialization

- As with simple variables
  - Arrays can be initialized while they are being declared
- When initializing arrays while declaring them
  - Not necessary to specify the size of the array
- Size of array is determined by the number of initial values in the braces
- For example:

```
double sales[] = {12.25, 32.50, 16.90, 23, 45.68};
```

or

```
double sales[5] = {12.25, 32.50, 16.90, 23, 45.68};
```

Restrictions on Array Processing

- Assignment does not work with arrays
  - If x and y are two arrays of the same type and size then the following statement is illegal:

```
int x[100], y[100];
y = x;  // C++ illegal
```

- We will see later that matlab allows it
- In order to copy one array into another array we must copy component-wise:

```
for(j = 0; j < 25; j++)
    y[j] = x[j];
```

Restrictions on Array Processing (continued)

- Comparison of arrays, reading data into an array and printing the contents of an array must be done component-wise

```
cin >>x;   //not supported
cout <<y; //not supported
```

- C++ does not allow aggregate operations on an array
- An aggregate operation on an array is any operation that manipulates the entire array as a single unit

Summary

- An array is a structured data type with a fixed number of components
  - Every component is of the same type
  - Components are accessed using their relative positions in the array
- Elements of a one-dimensional array are arranged in the form of a list
- An array index can be any expression that evaluates to a non-negative integer
- The value of the index must always be less than the size of the array

Examples

- Reverse input
- Array Max
- Selection-Sort

Example I: print in reverse

- Read a large list of numbers and print it in reverse order

Example I: print in reverse (Continued)

```
int A[1000];  // declare an array
int n;
cout<<"Enter number of items (at most 1000):";
cin>>n;
cout << "Enter " << n<< " numbers." << endl;
int i = 0;
for (i=0;  i < n; i++)
       cin >> A[i];
cout << "The numbers in reverse order are: ";
for (i = n-1; i >= 0; i--)
      cout << A[i] << " ";
```

Example II: find max in array

```
int A[10] = {14,2,97,10,3,5,-19,56,89,-43};
```

Find and print largest element in A

Example II: find max in array (continued)

```
int A[10] = {14,2,97,10,3,5,-19,56,89,-43};
int max = A[0];
for (int i = 1; i <10; i ++)
            if(max < A[i])
                  max = A[i];
cout << "The largest is "<<max<<endl;
```

Example III: Sorting

- Sorting problem
- Selection Sort

Sorting problem

- ***Input:*** sequence of *n* numbers

  A[0],   A[1],· · ·, A[*n-1*] stored in a size-*n* array A
- ***Output:*** permutation of the elements of A such A[0] ≤ A[1] ≤ · · · ≤A[*n*-1]
- **Example:**
  - ***Input:*** 8 2 4 9 3 6
  - ***Output:*** 2 3 4 6 8 9
- There are many sorting algorithms
- Will study: Selection-Sort

Idea of selection-Sort

To sort an array A[0….n]:

1. find the index minIndex of smallest element in A[0…n]

Idea of Selection-Sort (continued)

To sort an array A[0….n]:

1. find the index minIndex of smallest element in A[0…n]
2. exchange A[minIndex] (swap it) with A[0], hence now the number stored in A[0] is in its correct position in the desired sorted order
3. find the index minIndex of the smallest element in A[1… n]
4. exchange A[minIndex] with A[1] , hence the number stored in A[1] is in its correct position in the desired sorted order.
5. find the index minIndex of the smallest element in A[2 … n]
6. exchange A[minIndex] with A[2]
7. and so on until A[0… n] is sorted

Try it on an example

5 2 4 6 1 3

Pseudocode of Selection-Sort

- Pseudocode: Syntax-independent description of the algorithm
- To sort A[0...n-1]:

```
for i=0...n-1,
  1. find the  index minIndex of the smallest element of
     A[i...n-1] as follows:
       minIndex = i;
       for j=i+1…n-1,
         if A[minIndex] > A[j]
               minIndex = j;
  2.swap A[i] and A[minIndex]
```

- Nested loops

Swapping

- Say that we want to exchange (i.e., swap) A[7] and A[2]:

```
int temp= A[7]; // temporary variable
A[7]  =  A[2];
A[2]  =  temp;
```

Selection-Sort Code

```
for(i=0;i<n;i++) {
   // find the the index minIndex of the smallest element of A[i...n-1]
     int minIndex = i;
     for(int j=i+1; j<n; j++)
          if (A[minIndex] > A[j])
                    minIndex = j;
   // swap A[i] and A[minIndex]
     int temp = A[i];
     A[i] = A[minIndex];
     A[minIndex] = temp;
}
```