Standard Version of Starting Out with C++, 4th Edition

Chapter 7

Arrays

Topics

7.1 Arrays Hold Multiple Values

7.2 Accessing Array Elements

7.3 No Bounds Checking in C++

7.4 Array Initialization

7.5 Processing Array Contents

7.6 Using Parallel Arrays

Topics

7.7 Arrays as Function Arguments

7.1 Arrays Hold Multiple Values

- Array: variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using `[]` operator:

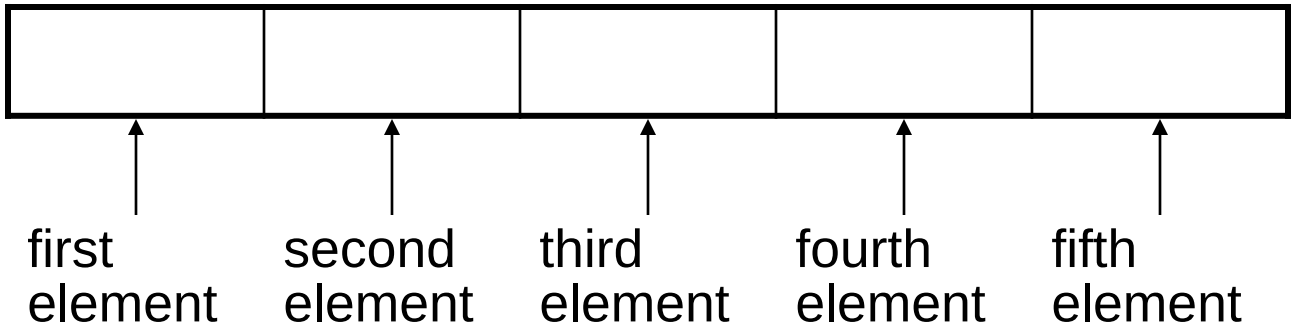
```
int tests[5];
```

Array - Memory Layout

- The definition:

```
int tests[5];
```

allocates the following memory:



Array Terminology

In the definition `int tests[5];`

- `int` is the data type of the array elements
- `tests` is the name of the array
- `5`, in `[5]`, is the size declarator. It shows the number of elements in the array.
- The size of an array is (number of elements) * (size of each element)

Array Terminology

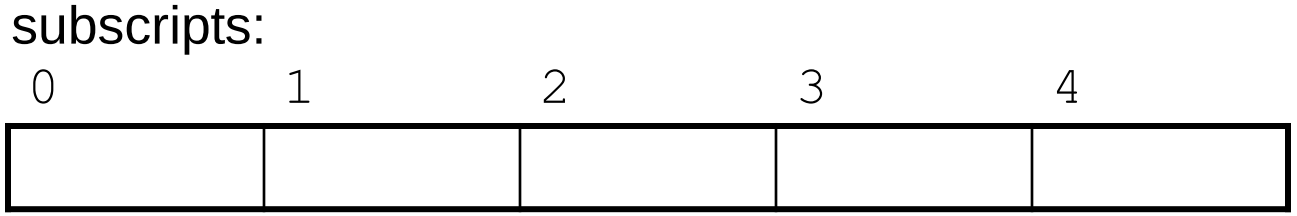
- The size of an array is:
  - the total number of bytes allocated for it
  - (number of elements) * (number of bytes for each element)
- Examples:

  `int tests[5]` is an array of 20 bytes, assuming 4 bytes for an `int`

  `long double measures[10]` is an array of 80 bytes, assuming 8 bytes for a `long double`

7.2 Accessing Array Elements

- Each array element has a subscript, used to access the element.
- Subscripts start at 0



Accessing Array Elements

- Array elements can be used as regular variables:

```
tests[0] = 79;
cout << tests[0];
cin >> tests[1];
tests[4] = tests[0] + tests[1];
```

- Arrays must be accessed via individual elements:

```
cout << tests; // not legal
```

Accessing Array Contents

- Can access element with constant subscript:

```
cout << tests[3] << endl;
```

- Can use integer expression as subscript:

```
for (i = 0; i < 5;i++)
   cout << tests[i] << endl;
```

7.3 No Bounds Checking in C++

- C++ does not check if an array subscript is in the range of values for subscripts of the array
- Can access memory using subscripts that is before or after the memory for an array
- Can corrupt other memory locations, crash program, or lock up computer
- Not recommended

7.4 Array Initialization

- Can be initialized during program execution with assignment statements:

```
tests[0] = 79;
tests[1] = 82; // etc.
```

- Can be initialized at array definition with an initialization list:

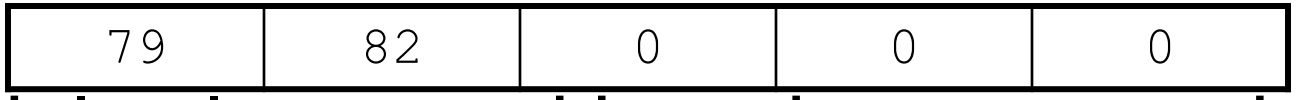
```
int tests[5] = {79,82,91,77,84};
```

- Initialization list cannot exceed array size

Partial Array Initialization

- If array is initialized at definition with fewer initial values than the size declarator of the array, the remaining elements will be set to `0`:

```
int tests[5] = {79, 82};
```

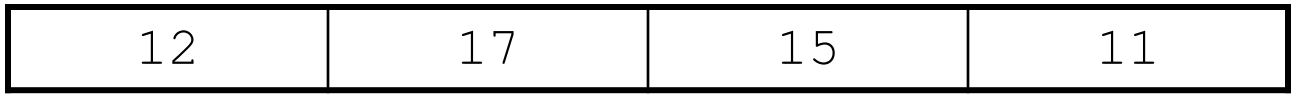


- Initial values used in order; cannot skip over elements to initialize noncontiguous range

Implicit Array Sizing

- Can determine array size by the size of the initialization list:

```
short quizzes[]={12,17,15,11};
```



- Must use either array size declarator or initialization list at array definition

Initializing With a String

- Character array can be initialized by enclosing string in " ":

```
char fName[6] = "Henry";
```

- Must leave room for `\0` at end of array
- If initializing character-by-character, must add in `\0` explicitly:

```
char fName[6] =
{ 'H', 'e', 'n', 'r', 'y', '\0'};
```

7.5 Processing Array Contents

- Array elements can be treated as ordinary variables of the same type as the array
- When using `++`, `--` operators, don't confuse the element with the subscript:

```
tests[i]++; // add 1 to tests[i]
tests[i++]; // increment i, no
            // effect on tests
```

Array Assignment

To copy one array to another,

- don't try to assign one array to the other:

```
newTests = tests;
```

- assign element-by-element:

```
for (i=0; i<5; i++)
   newTests[i] = tests[i];
```

Display the Contents of an Array

- Can display character array by using its name:

```
cout << fName << endl;
```

- For other types of arrays, must go element-by-element:

```
for (i=0; i<5; i++)
    cout << tests[i] << endl;
```

Sum of Array Elements

- Use a simple loop to add together array elements:

```
int tnum;
float average, sum = 0;
for(tnum = 0; tnum < 5; tnum++)
      sum += tests[tnum];
```

- Once summed, can compute average:

```
average = sum/5;
```

7.6 Using Parallel Arrays

- Parallel arrays: two or more arrays that contain related data
- Subscript is used to relate arrays: elements at same subscript are related
- Arrays may be of different types

Parallel Array Example

```
string name[5];  // student name
float average[5];// course average
char grade[5];   // course grade
...
for(int i = 0; i < 5; i++)
    cout << "Student: " << name[i]
         << " average: " << average[i]
         << " grade: " << grade[i]
         << endl;
```

7.7 Arrays as Function Arguments

- To pass an array to a function, just use the array name:

```
showScores(tests);
```

- To define a function that takes an array parameter, use empty `[]` for array argument:

```
void showScores(int []);
                // function prototype
void showScores(int tests[])
                // function header
```

Arrays as Function Arguments

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:

```
showScores(tests, 5);
```

- Array size must also be reflected in prototype, header:

```
void showScores(int [], int);
                // function prototype
void showScores(int tests[], int size)
                // function header
```

Modifying Arrays in Functions

- Array names in functions are similar to reference variables – changes made to array in a function are reflected in actual array in calling function
- Need to exercise caution that array is not inadvertantly changed by a function