05: ARRAY

Programming Technique I
(SCSJ1013)

Array

Contents:

- Introduction
- Array Declaration
- Memory Layout
- Terminology
- Accessing Array Elements
- Array Initialization
- Processing Array Contents
- Array Assignment
- Arrays as Function Arguments
- Two-Dimensional Arrays
- Array of Strings

Introduction

- Array: variable that can store a collection of data of the **same** type
  - Examples: A list of names, A list of temperatures
- Why do we need arrays?
  - Imagine keeping track of 5 test scores, or 100, or 1000 in memory
    - How would you name all the variables?
    - How would you process each of the variables?

Declaring an Array

- An array, named **`test`**, containing five variables of type **`int`** can be declared as

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

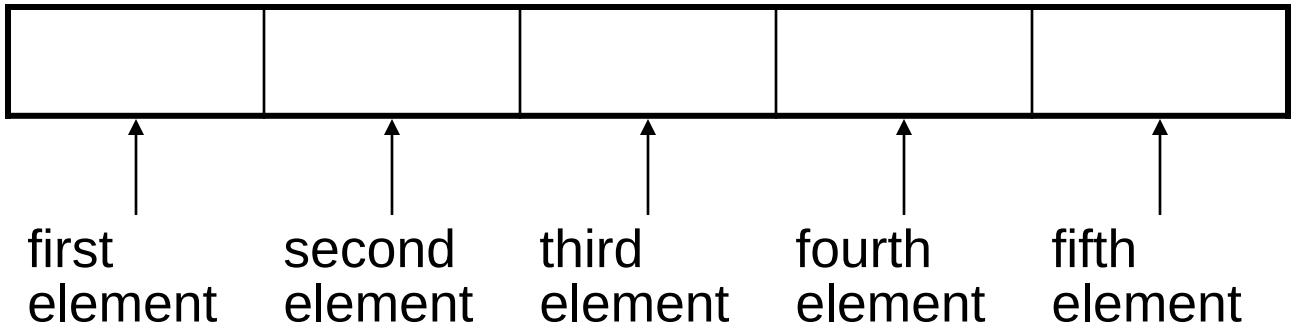
- The value in brackets is called
  - A subscript
  - An index

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`

Size Declarators

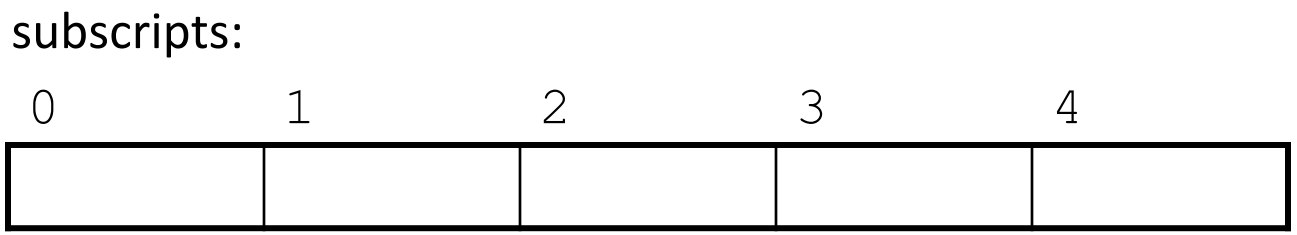
- Named constants are commonly used as size declarators.

```
const int SIZE = 5;
int tests[SIZE];
```

- This eases program maintenance when the size of the array needs to be changed.

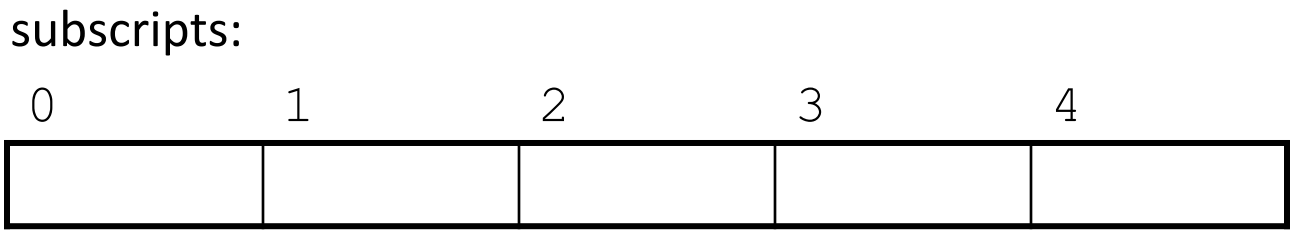
Accessing Array Elements

- Each element in an array is assigned a unique *subscript*.
- Subscripts start at 0



Accessing Array Elements

- The last element's subscript is *n*-1 where *n* is the number of elements in the array.



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 Elements - example

Program 7-1

```
1  // This program asks for the number of hours worked
2  // by six employees. It stores the values in an array.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     const int NUM_EMPLOYEES = 6;
9     int hours[NUM_EMPLOYEES];
10
11    // Get the hours worked by six employees.
12    cout << "Enter the hours worked by six employees: ";
13    cin >> hours[0];
14    cin >> hours[1];
15    cin >> hours[2];
16    cin >> hours[3];
17    cin >> hours[4];
18    cin >> hours[5];
19
```

(Program Continues)

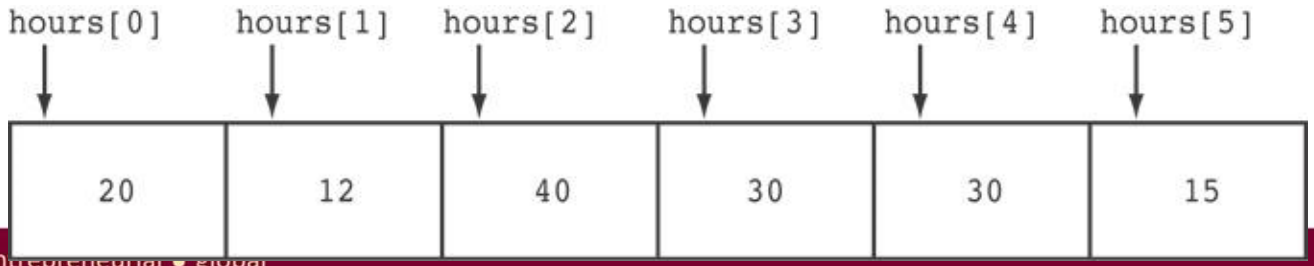
Accessing Array Elements - example

```
20      // Display the values in the array.
21      cout << "The hours you entered are:";
22      cout << " " << hours[0];
23      cout << " " << hours[1];
24      cout << " " << hours[2];
25      cout << " " << hours[3];
26      cout << " " << hours[4];
27      cout << " " << hours[5] << endl;
28      return 0;
29  }
```

Program Output with Example Input

```
Enter the hours worked by six employees: 20 12 40 30 30 15 [Enter]
The hours you entered are: 20 12 40 30 30 15
```

Here are the contents of the `hours` array, with the values entered by the user in the example output:



Accessing Array Contents

- Can access element with a constant or literal subscript:

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

- Can use integer expression as subscript:

```
int i = 5;
cout << tests[i] << endl;
```

Using a Loop to Step Through an Array

Example – The following code defines an array, `numbers`, and assigns 99 to each element:

```
const int ARRAY_SIZE = 5;
int numbers[ARRAY_SIZE];

for (int count = 0; count < ARRAY_SIZE; count++)
     numbers[count] = 99;
```

A Closer Look At the Loop

The variable `count` starts at 0, which is the first valid subscript value.

The loop ends when the variable `count` reaches 5, which is the first invalid subscript value.

```
for (count = 0; count < ARRAY_SIZE; count++)
   numbers[count] = 99;
```

The variable `count` is incremented after each iteration.

Default Initialization

- Global array → all elements initialized to 0 by default
- Local array → all elements *uninitialized* by default

No Bounds Checking in C++

- When you use a value as an array subscript, C++ does not check it to make sure it is a *valid* subscript.

- In other words, you can use subscripts that are beyond the bounds of the array.

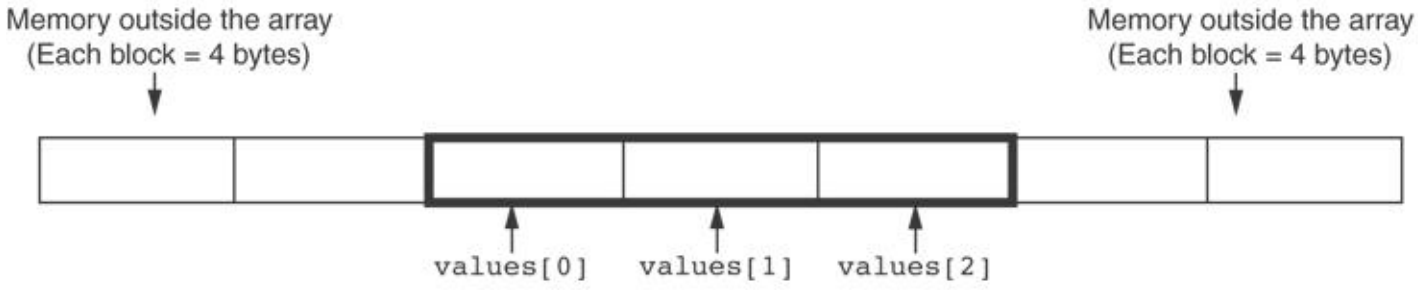
Example

- The following code defines a three-element array, and then writes five values to it!

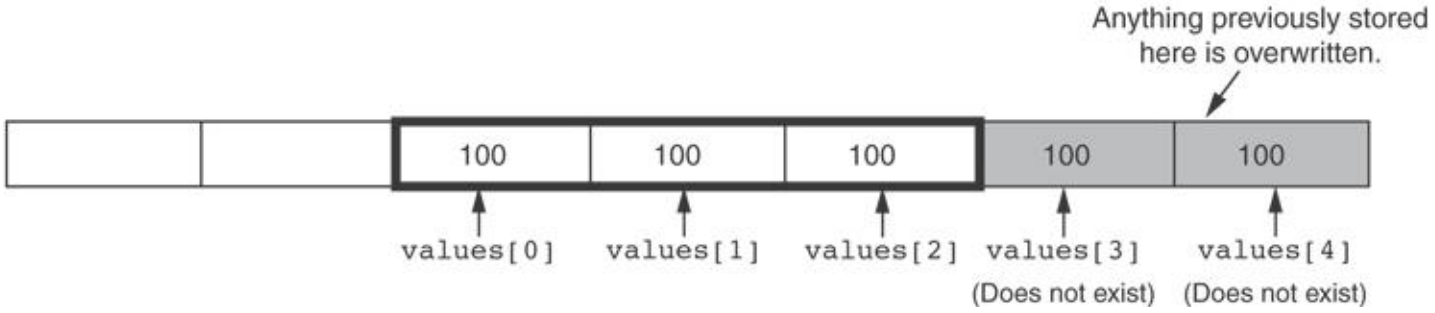
```
 9     const int SIZE = 3;  // Constant for the array size
10     int values[SIZE];    // An array of 3 integers
11     int count;           // Loop counter variable
12
13     // Attempt to store five numbers in the three-element array.
14     cout << "I will store 5 numbers in a 3 element array!\n";
15     for (count = 0; count < 5; count++)
16        values[count] = 100;
```

What the Code Does

The way the `values` array is set up in memory.
The outlined area represents the array.



How the numbers assigned to the array overflow the array's boundaries.
The shaded area is the section of memory illegally written to.



No Bounds Checking in C++

- Be careful not to use invalid subscripts.
- Doing so can corrupt other memory locations, crash program, or lock up computer, and cause elusive bugs.

Array Initialization

- Arrays can be initialized with an initialization list:

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

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.

Example

```
 7       const int MONTHS = 12;
 8       int days[MONTHS] = { 31, 28, 31, 30,
 9                            31, 30, 31, 31,
10                            30, 31, 30, 31};
11
12       for (int count = 0; count < MONTHS; count++)
13       {
14          cout << "Month " << (count + 1) << " has ";
15          cout << days[count] << " days.\n";
16       }
```

Program Output

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

Array Initialization

- Valid

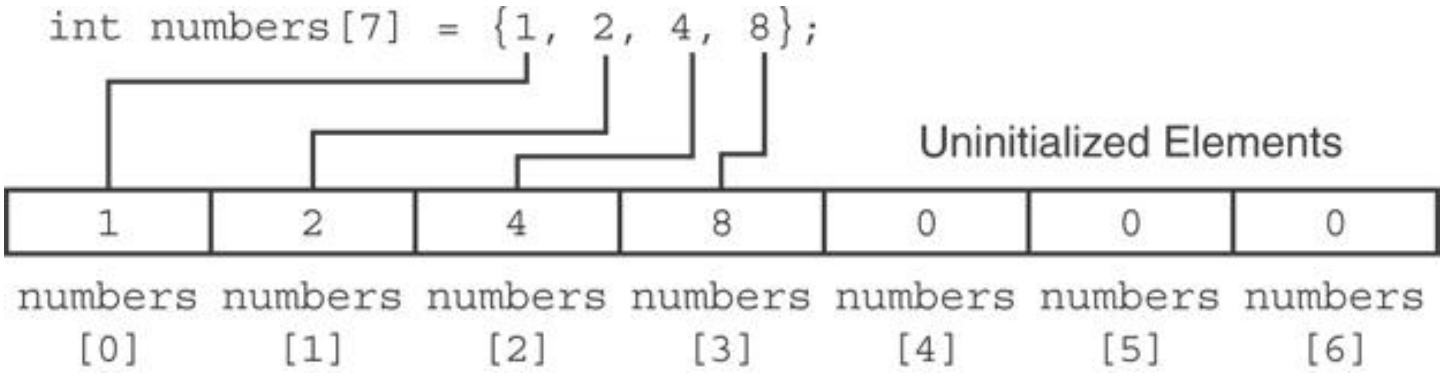
```
int tests[3] = { 3, 5, 11 };
```

- Invalid

```
int tests[3];
tests= { 3, 5, 11 };
```

Partial Array Initialization

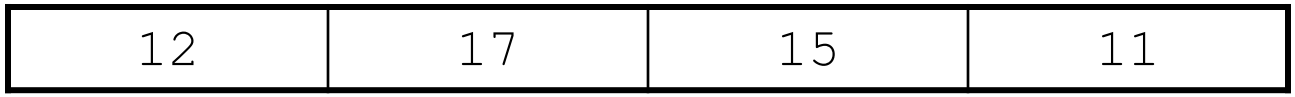
- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to `0` :



Implicit Array Sizing

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

```
int 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 " ":

```
const int SIZE = 6;
char fName[SIZE] = "Henry";
```

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

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

In-Class Exercise

- Are each of the following valid or invalid array definitions? (If a definition is invalid, explain why)

```
int numbers[10] = {0, 0, 1, 0, 0, 1, 0, 0, 1, 1};
int matrix[5] = {1, 2, 3, 4, 5, 6, 7};
double radix[10] = {3.2, 4.7};
int table[7] = {2, , , 27, , 45, 39};
char codes [] = {'A', 'X', '1', '2', 's'};
int blanks[];
char name[6] = "Joanne";
```

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;  // Won't work
```

- Instead, assign element-by-element:

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

Printing the Contents of an Array

- You can display the contents of a *character* array by sending its name to cout:

```
char fName[] = "Henry";
cout << fName << endl;
```

But, this ONLY works with character arrays!

Printing the Contents of an Array

- For other types of arrays, you must print element-by-element:

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

Summing and Averaging Array Elements

- Use a simple loop to add together array elements:

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

- Once summed, can compute average:

```
average = sum / SIZE;
```

Finding the Highest Value in an Array

```
int count;
int highest;
highest = numbers[0];
for (count = 1; count < SIZE; count++)
{
   if (numbers[count] > highest)
      highest = numbers[count];
}
```

When this code is finished, the `highest` variable will contain the highest value in the `numbers` array.

Finding the Lowest Value in an Array

```
int count;
int lowest;
lowest = numbers[0];
for (count = 1; count < SIZE; count++)
{
   if (numbers[count] < lowest)
      lowest = numbers[count];
}
```

When this code is finished, the `lowest` variable will contain the lowest value in the `numbers` array.

Partially-Filled Arrays

- If it is unknown how much data an array will be holding:
  - Make the array large enough to hold the largest expected number of elements.
  - Use a counter variable to keep track of the number of items stored in the array.

Comparing Arrays

- To compare two arrays, you must compare element-by-element:

```
const int SIZE = 5;
int firstArray[SIZE] = { 5, 10, 15, 20, 25 };
int secondArray[SIZE] = { 5, 10, 15, 20, 25 };
bool arraysEqual = true; // Flag variable
int count = 0;           // Loop counter variable
// Compare the two arrays.
while (arraysEqual && count < SIZE)
{
   if (firstArray[count] != secondArray[count])
      arraysEqual = false;
   count++;
}
if (arraysEqual)
   cout << "The arrays are equal.\n";
else
   cout << "The arrays are not equal.\n";
```

In-Class Exercise

Given the following array definition:

```
int values[] = {2,6,10,14};
```

What does each of the following display?

a)
```
cout<<values[2];
```

b)
```
cout<<++values[0];
```

c)
```
cout<< values[1]++;
```

d)
```
x = 2;
cout<<values[++x];
```

In-Class Exercise

- Declare an integer array named `names` with 20 elements. Write a loop that prints each element of the array.

In-Class Exercise

- Write a program that lets the user enter 10 values into an array. The program should then display the largest and smallest values stored in the array.
- Write a program that lets the user enter the total rainfall for each of 12 months into an array of doubles. The program should then calculate and display the total rainfall for the year, the average monthly rainfall, and the months with the highest and lowest amounts.

  Input Validation: Do not accept negative numbers for monthly rainfall figures.

Using Parallel Arrays

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

Parallel Array Example

```
const int SIZE = 5;    // Array size
int id[SIZE];          // student ID
double average[SIZE]; // course average
char grade[SIZE];      // course grade
...
for(int i = 0; i < SIZE; i++)
{
    cout << "Student ID: " << id[i]
         << " average: " << average[i]
         << " grade: " << grade[i]
         << endl;
}
```

Parallel Array Example

Program 7-12

```
1  // This program stores, in an array, the hours worked by 5
2  // employees who all make the same hourly wage.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     const int NUM_EMPLOYEES = 5;
10    int hours[NUM_EMPLOYEES];         // Holds hours worked
11    double payRate[NUM_EMPLOYEES];    // Holds pay rates
12
13    // Input the hours worked.
14    cout << "Enter the hours worked by " << NUM_EMPLOYEES;
15    cout << " employees and their\n";
16    cout << "hourly pay rates.\n";
17    for (int index = 0; index < NUM_EMPLOYEES; index++)
18    {
19       cout << "Hours worked by employee #" << (index+1) << ": ";
20       cin >> hours[index];
21       cout << "Hourly pay rate for employee #" << (index+1) << ": ";
22       cin >> payRate[index];
23    }
24
```

(Program Continues)

Parallel Array Example

Program 7-12 *(Continued)*

```
25      // Display each employee's gross pay.
26      cout << "Here is the gross pay for each employee:\n";
27      cout << fixed << showpoint << setprecision(2);
28      for (index = 0; index < NUM_EMPLOYEES; index++)
29      {
30         double grossPay = hours[index] * payRate[index];
31         cout << "Employee #" << (index + 1);
32         cout << ": $" << grossPay << endl;
33      }
34      return 0;
35   }
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 5 employees and their
hourly pay rates.
Hours worked by employee #1: 10 [Enter]
Hourly pay rate for employee #1: 9.75 [Enter]
Hours worked by employee #2: 15 [Enter]
Hourly pay rate for employee #2: 8.62 [Enter]
Hours worked by employee #3: 20 [Enter]
Hourly pay rate for employee #3: 10.50 [Enter]
Hours worked by employee #4: 40 [Enter]
Hourly pay rate for employee #4: 18.75 [Enter]
Hours worked by employee #5: 40 [Enter]
Hourly pay rate for employee #5: 15.65 [Enter]
```

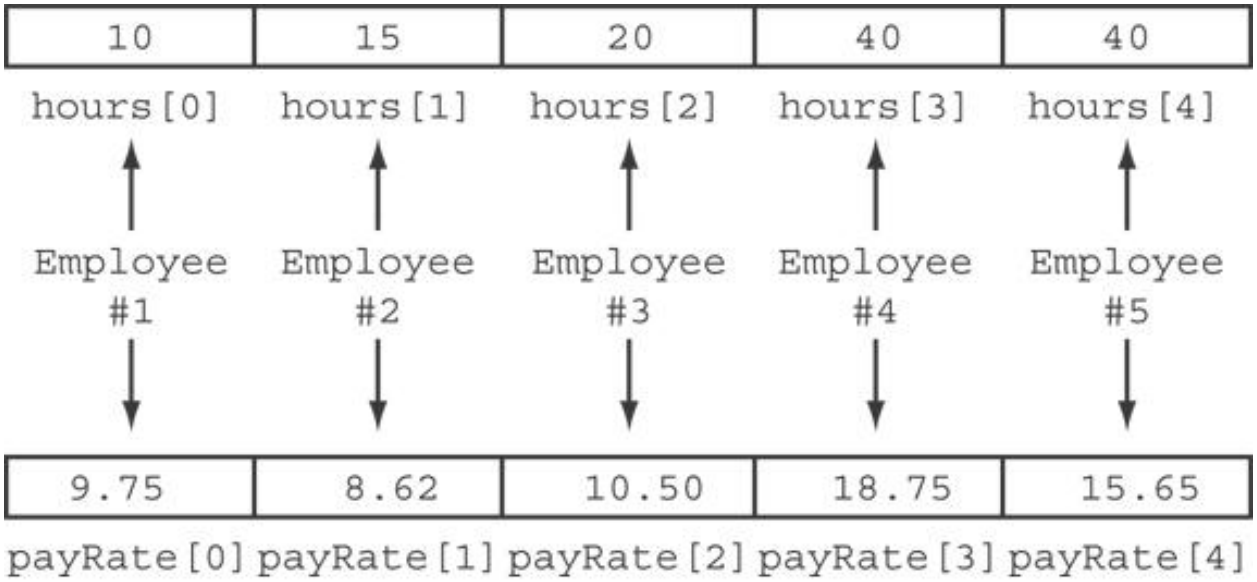
(program output continues)

Parallel Array Example

Program 7-12 *(continued)*

```
Here is the gross pay for each employee:
Employee #1: $97.50
Employee #2: $129.30
Employee #3: $210.00
Employee #4: $750.00
Employee #5: $626.00
```

The `hours` and `payRate` arrays are related through their subscripts:



In-Class Exercise

- What is the output of the following code?

  (You may need to use a calculator.) .

```
const int SIZE = 5;
int time[SIZE] = {1, 2, 3, 4, 5},
speed[SIZE] = {18, 4, 27, 52, 100},
dist[SIZE];

for (int count = 0; count < SIZE; count++)
      dist[count] = time[count] * speed[count];
for (int count = 0; count < SIZE; count++) {
      cout << time[count] << " ";
      cout << speed[count] << " ";
      cout « dist[count] << endl;
}
```

In-Class Exercise

- Write a program that store the populations of 12 countries. Define 2 arrays that may be used in parallel to store the names of the countries and their populations. Write a loop that uses these arrays to print each country's name and its population.

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, ARRAY_SIZE);
```

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

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

Arrays as Function Arguments - example

Program 7-14

```
1  // This program demonstrates an array being passed to a function.
2  #include <iostream>
3  using namespace std;
4
5  void showValues(int [], int); // Function prototype
6
7  int main()
8  {
9     const int ARRAY_SIZE = 8;
10    int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
11
12    showValues(numbers, ARRAY_SIZE);
13    return 0;
14 }
15
```

(Program Continues)

Arrays as Function Arguments - example

Program 7-14 *(Continued)*

```
16  //*********************************************
17  // Definition of function showValue.          *
18  // This function accepts an array of integers and  *
19  // the array's size as its arguments. The contents *
20  // of the array are displayed.                *
21  //*********************************************
22
23  void showValues(int nums[], int size)
24  {
25     for (int index = 0; index < size; index++)
26        cout << nums[index] << " ";
27     cout << endl;
28  }
```

Program Output

```
5 10 15 20 25 30 35 40
```

Modifying Arrays in Functions

- Array names in functions are like 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 inadvertently changed by a function

In-Class Exercise

- The following program skeleton, when completed, will ask the user to enter 10 integers which are stored in an array. The function `avgArray`, which you must write, is to calculate and return the average of the numbers entered.

```
#include <iostream>
//Write your function prototype here
int main() {
        const int SIZE = 10;
        int userNums[SIZE];
        cout << "Enter 10 numbers: ";
        for (int count = 0; count < SIZE; count++){
                cout << "#" « (count + 1) << " ";
                cin >> userNums[count];
        }
        cout <<  "The average of those numbers is ";
        cout << avgArray(userNUms, SIZE) << endl;
        return 0;
}
//Write the function avgArray here.
```

In-Class Exercise

```
#include <iostream>
using namespace std;
void Test(int []);
int main()
{
int myArr [4]={3,4,5,6};
   for(int i=0;i<4;i++)
 cout<<myArr[i]<<" ";
   cout<<endl;
   Test(myArr);
   cout<<endl;
 for(int i=0;i<4;i++)
   cout<<myArr[i]<<" ";
       system("pause");
       return 0;}
```

```
void Test(int z[])
{
   int temp=z[3];
    z[3]=z[0];
    z[0]=temp;

    for(int
   j=0;j<4;j++)
       cout<<z[j]<<"   ";
}
```

In-Class Exercise

```
#include <iostream>
using namespace std;
void Test(int , int,int[]);
int main()
{  int x = 1;
   int y[3];
   y[0]=1;
   Test(x,y[0],y);
   cout<<"x is: " << x<< endl;
   cout<<"y[0] is: " <<y[0] <<
   endl;
   for(int i=0;i<3;i++)
           cout<<y[i]<<endl;
       system("pause");
       return 0;}
```

```
void Test(int num, int num1,
   int z[])
{
   num=1001;
   num1=290;
   z[1]=34;
   z[2]=35;
}
```

In-Class Exercise

Each of the following definitions and program segments has errors. Locate as many as you can and correct the errors.

a)
```
void showValues(int nums)
{
        for(int i = 0; i<8; i++)
                cout<<nums[i];
}
```

b)
```
void showValues(int nums [4])
{
  for(int i = 0; i<8; i++)
          cout<<nums[i];
}
```

In-Class Exercise

- Consider the following function prototypes:

```
void funcOne(int [], int);
int findSum(int, int);
```

And the declarations:

```
int list[50];
int num;
```

Write a C++ statements that:

a) Call the function `funcOne` with the actual parameters, `list` and 50 respectively.

b) Print the value returned by the function `funcSum` with the actual parameters, 50, and the fourth element of `list` respectively.

c) Print the value returned by the function `funcSum` with the actual parameters, the thirtieth and tenth elements of `list`, respectively.

In-Class Exercise

- Write a program that has two overloaded functions that return the average of an array with the following headers:

```
int average(int array[], int size)
double average(int array[], int size
```

Use `{1,2,3,4,5,6}` and `{6.0,4.4,1.9,2.9,3.4,3.5}` to test the functions.

In-Class Exercise

- Write a program that has a function that returns the index of the smallest element in an array of integers. If there are more than one such elements, return the smallest index. Use `{1,2,4,5,10,100,2,-22}` to test the function.

Two-Dimensional Arrays

- Can define one array for multiple sets of data
- Like a table in a spreadsheet
- Use two size declarators in definition:

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

- First declarator is number of rows; second is number of columns

Two-Dimensional Array Representation

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

columns

rows

exams[0][0]	exams[0][1]	exams[0][2]
exams[1][0]	exams[1][1]	exams[1][2]
exams[2][0]	exams[2][1]	exams[2][2]
exams[3][0]	exams[3][1]	exams[3][2]

- Use two subscripts to access element:

```
exams[2][2] = 86;
```

Two-Dimensional Array Representation - Example

Program 7-18

```
1  // This program demonstrates a two-dimensional array.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  int main()
7  {
8     const int NUM_DIVS = 3;           // Number of divisions
9     const int NUM_QTRS = 4;           // Number of quarters
10    double sales[NUM_DIVS][NUM_QTRS]; // Array with 3 rows and 4 columns.
11    double totalSales = 0;            // To hold the total sales.
12    int div, qtr;                     // Loop counters.
13
14    cout << "This program will calculate the total sales of\n";
15    cout << "all the company's divisions.\n";
16    cout << "Enter the following sales information:\n\n";
17
```

(program continues)

Two-Dimensional Array Representation - Example

Program 7-18 *(continued)*

```
18      // Nested loops to fill the array with quarterly
19      // sales figures for each division.
20      for (div = 0; div < NUM_DIVS; div++)
21      {
22         for (qtr = 0; qtr < NUM_QTRS; qtr++)
23         {
24            cout << "Division " << (div + 1);
25            cout << ", Quarter " << (qtr + 1) << ": $";
26            cin >> sales[div][qtr];
27         }
28         cout << endl; // Print blank line.
29      }
30
31      // Nested loops used to add all the elements.
32      for (div = 0; div < NUM_DIVS; div++)
33      {
34         for (qtr = 0; qtr < NUM_QTRS; qtr++)
35            totalSales += sales[div][qtr];
36      }
37
38      cout << fixed << showpoint << setprecision(2);
39      cout << "The total sales for the company are: $";
40      cout << totalSales << endl;
41      return 0;
42   }
```

Two-Dimensional Array Representation - Example

Program Output with Example Input Shown in Bold

```
This program will calculate the total sales of
all the company's divisions.
Enter the following sales data:

Division 1, Quarter 1: $31569.45 [Enter]
Division 1, Quarter 2: $29654.23 [Enter]
Division 1, Quarter 3: $32982.54 [Enter]
Division 1, Quarter 4: $39651.21 [Enter]

Division 2, Quarter 1: $56321.02 [Enter]
Division 2, Quarter 2: $54128.63 [Enter]
Division 2, Quarter 3: $41235.85 [Enter]
Division 2, Quarter 4: $54652.33 [Enter]

Division 3, Quarter 1: $29654.35 [Enter]
Division 3, Quarter 2: $28963.32 [Enter]
Division 3, Quarter 3: $25353.55 [Enter]
Division 3, Quarter 4: $32615.88 [Enter]

The total sales for the company are: $456782.34
```

2D Array Initialization

- Two-dimensional arrays are initialized row-by-row:

```
const int ROWS = 2, COLS = 2;
int exams[ROWS][COLS] = { {84, 78},
                          {92, 97} };
```

84	78
92	97

- Can omit inner `{ }`, some initial values in a row – array elements without initial values will be set to `0` or `NULL`

Two-Dimensional Array as Parameter, Argument

- Use array name as argument in function call:

```
getExams(exams, 2);
```

- Use empty `[]` for row, size declarator for column in prototype, header:

```
const int COLS = 2;
// Prototype
void getExams(int [][COLS], int);

// Header
void getExams(int exams[][COLS], int rows)
```

Example – The `showArray` Function from Program 7-19

```
30  //*********************************************************
31  // Function Definition for showArray                      *
32  // The first argument is a two-dimensional int array with COLS    *
33  // columns. The second argument, rows, specifies the number of    *
34  // rows in the array. The function displays the array's contents. *
35  //*********************************************************
36
37  void showArray(int array[][COLS], int rows)
38  {
39     for (int x = 0; x < rows; x++)
40     {
41        for (int y = 0; y < COLS; y++)
42        {
43           cout << setw(4) << array[x][y] << " ";
44        }
45        cout << endl;
46     }
47  }
```

How `showArray` is Called

```
15      int table1[TBL1_ROWS][COLS] = {{1, 2, 3, 4},
16                                     {5, 6, 7, 8},
17                                     {9, 10, 11, 12}};
18      int table2[TBL2_ROWS][COLS] = {{10, 20, 30, 40},
19                                     {50, 60, 70, 80},
20                                     {90, 100, 110, 120},
21                                     {130, 140, 150, 160}};
22
23      cout << "The contents of table1 are:\n";
24      showArray(table1, TBL1_ROWS);
25      cout << "The contents of table2 are:\n";
26      showArray(table2, TBL2_ROWS);
```

Summing All the Elements in a Two-Dimensional Array

Given the following definitions:

```
const int NUM_ROWS = 5; // Number of rows
const int NUM_COLS = 5; // Number of columns
int total = 0;          // Accumulator
int numbers[NUM_ROWS][NUM_COLS] =
   {{2, 7, 9, 6, 4},
    {6, 1, 8, 9, 4},
    {4, 3, 7, 2, 9},
    {9, 9, 0, 3, 1},
    {6, 2, 7, 4, 1}};
```

Summing All the Elements in a Two-Dimensional Array

```
// Sum the array elements.
for (int row = 0; row < NUM_ROWS; row++)
{
   for (int col = 0; col < NUM_COLS; col++)
      total += numbers[row][col];
}

// Display the sum.
cout << "The total is " << total << endl;
```

Summing the Rows of a Two-Dimensional Array

Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;   // Accumulator
double average; // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
     {{88, 97, 79, 86, 94},
      {86, 91, 78, 79, 84},
      {82, 73, 77, 82, 89}};
```

Summing the Rows of a Two-Dimensional Array

```
// Get each student's average score.
for (int row = 0; row < NUM_STUDENTS; row++)
{
   // Set the accumulator.
   total = 0;
   // Sum a row.
   for (int col = 0; col < NUM_SCORES; col++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_SCORES;
   // Display the average.
   cout << "Score average for student "
        << (row + 1) << " is " << average <<endl;
}
```

Summing the Columns of a Two-Dimensional Array

Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;   // Accumulator
double average; // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
     {{88, 97, 79, 86, 94},
      {86, 91, 78, 79, 84},
      {82, 73, 77, 82, 89}};
```

Summing the Columns of a Two-Dimensional Array

```
// Get the class average for each score.
for (int col = 0; col < NUM_SCORES; col++)
{
   // Reset the accumulator.
   total = 0;
   // Sum a column
   for (int row = 0; row < NUM_STUDENTS; row++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_STUDENTS;
   // Display the class average.
   cout << "Class average for test " << (col + 1)
        << " is " << average << endl;
}
```

Array of Strings

- Use a two-dimensional array of characters as an array of strings:

```
const int NAMES = 3, SIZE = 10;
char students[NAMES][SIZE] =
 { "Ann", "Bill", "Cindy" };
```

- Each row contains one string

- Can use row subscript to reference the string in a particular row:

```
cout << students[i];
```

Array of Strings - example

Program 7-20

```
1  // This program displays the number of days in each month.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     const int NUM_MONTHS = 12;  // The number of months
8     const int STRING_SIZE = 10; // Maximum size of each string
9
10    // Array with the names of the months
11    char months[NUM_MONTHS][STRING_SIZE] =
12                  { "January", "February", "March",
13                    "April", "May", "June",
14                    "July", "August", "September",
15                    "October", "November", "December" };
16
17    // Array with the number of days in each month
18    int days[NUM_MONTHS] = {31, 28, 31, 30,
19                            31, 30, 31, 31,
20                            30, 31, 30, 31};
21
22    // Display the months and their numbers of days.
23    for (int count = 0; count < NUM_MONTHS; count++)
24    {
25       cout << months[count] << " has ";
26       cout << days[count] << " days.\n";
27    }
28    return 0;
29 }
```

Array of Strings

Program 7-20 *(continued)*

Program Output

```
January has 31 days.
February has 28 days.
March has 31 days.
April has 30 days.
May has 31 days.
June has 30 days.
July has 31 days.
August has 31 days.
September has 30 days.
October has 31 days.
November has 30 days.
December has 31 days.
```

Arrays with Three or More Dimensions

- Can define arrays with any number of dimensions:

```
short rectSolid[2][3][5];
double timeGrid[3][4][3][4];
```

- When used as parameter, specify all except 1st dimension in prototype, heading:

```
void getRectSolid(short [][3][5]);
```

In-Class Exercise

- Define a two-dimensional array of `int` named `grades`. It should have 30 rows and 10 columns.
- How many elements are in the following array?

```
double sales[6][5];
```

In-Class Exercise

- Define an array of strings to store the name of your friends in this class.
- Initialize the array with 5 names.
- Print the names.
- Write a function to change the names in the array.

```
void changeName(char [][25], int size);
```

In-Class Exercise

- Consider the following declarations:

```
const int CAR_TYPES = 5;
const int COLOR_TYPES = 6;
double sales[CAR_TYPES][COLOR_TYPES];
```

a) How many elements does the array sales have?
b) What is the number of rows in the array sales?
c) What is the number of columns in the array sales?
d) Write a complete code to sum the sales by `CAR_TYPES`.
e) Write a complete code to sum the sales by `COLOR_TYPES`.

- Write a complete program that stores the following number of medal collection for 5 countries into the 2-D array called medals.

	Gold	Silver	Bronze
Country 1	129	257	590
Country 2	120	279	394
Country 3	115	290	123
Country 4	98	209	112

- Your program must have the following functions that do the following:
  - Read the number of medal for each country from a keyboard and store them inside the medals array.
  - Return total number of medals won by country 3.
  - Return the largest number of medals won.
  - Return the smallest number of medals won.
  - Return the highest number of gold medal won.
  - Return the total number of bronze medal won.