**LEcTURE 07**

**ARRAYS**

IMRAN IHSAN
ASSISTANT PROFESSOR
AIR UNIVERSITY, ISLAMABAD

**ARRAYS**

HOLD MULTIPLE VALUES AND MEMORY LAYOUT

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

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

- Allocates the following memory:

  ![Memory Layout Diagram]

  - First element
  - Second element
  - Third element
  - Fourth element
  - Fifth element
Arrays

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)

- 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.
ARRAYS

ACCESSING ELEMENTS

- Each element in an array is assigned a unique subscript.
- Subscripts start at 0
- The last element's subscript is n-1 where n is the number of elements in the array.

```
subscripts:

0   1   2   3   4
```

- 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
```

ARRAYS

IN PROGRAM

Program 7-1

```cpp
// This program asks for the number of hours worked
// by six employees. It stores the values in an array.
#include <iostream>
using namespace std;

int main()
{
    const int NUM_EMPLOYEES = 6;
    int hours[NUM_EMPLOYEES];

    // Get the hours worked by each employee.
    cout << "Enter the hours worked by "
        << NUM_EMPLOYEES << " employees: ";
    cin >> hours[0];
    cin >> hours[1];
    cin >> hours[2];
    cin >> hours[3];
    cin >> hours[4];
    cin >> hours[5];
```

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>12</td>
<td>40</td>
<td>30</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

Arrays

**ACCESSING CONTENT**

- 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;
  ```

**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;
```
CLOSER LOOK AT LOOPS

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

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

The variable count is incremented after each iteration.

GLOBAL INITIALIZATION

- **Global array**
  - all elements initialized to 0 by default

- **Local array**
  - all elements uninitialized by default
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.

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

```
const int SIZE = 3; // Constant for the array size
int values[SIZE]; // An array of 3 integers
int count; // Loop counter variable

// Attempt to store five numbers in the three-element array.
cout << "I will store 5 numbers in a 3 element array!\n";
for (count = 0; count < 5; count++)
    values[count] = 100;
```
ARRAYS
OFF-BY-ONE ERRORS

- An off-by-one error happens when you use array subscripts that are off by one.
- This can happen when you start subscripts at 1 rather than 0:

```c
// This code has an off-by-one error.
const int SIZE = 100;
int numbers[SIZE];
for (int count = 1; count <= SIZE; count++)
    numbers[count] = 0;
```

ARRAY INITIALIZATION

- Arrays can be initialized with an initialization list:

```c
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.

```
const int MONTHS = 12;
for (int count = 0; count < MONTHS; count++)
    {
    cout << "Month " << (count + 1) << " has ";
    cout << days[count] << " days.\n";
    }
```

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

PARTIAL ARRAY INITIALIZATION

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

\[
\text{int numbers[7] = \{1, 2, 4, 8\};}
\]

Uninitialized Elements

\[
\begin{array}{ccccccc}
1 & 2 & 4 & 8 & 0 & 0 & 0 \\
\end{array}
\]

numbers numbers numbers numbers numbers numbers numbers

\[
\begin{array}{ccccccc}
\end{array}
\]

ARRAY INITIALIZATION

IMPLICIT ARRAY SIZING

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

\[
\text{int quizzes[] = \{12, 17, 15, 11\};}
\]

12 17 15 11

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

• 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:

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

PROCESSING ARRAY CONTENT

ARRAY ASSIGNMENT

• To copy one array to another,
• Don't try to assign one array to the other:

```c
newTests = tests;  // Won't work
```

• Instead, assign element-by-element:

```c
for (i = 0; i < ARRAY_SIZE; i++)
    newTests[i] = tests[i];
```
PROCESSING ARRAY CONTENT

PRINTING THE CONTENT

- You can display the contents of a character array by sending its name to `cout`:
  ```cpp
  char fName[] = "Henry";
  cout << fName << endl;
  ```
- But, this ONLY works with character arrays!
- For other types of arrays, you must print element-by-element:
  ```cpp
  for (i = 0; i < ARRAY_SIZE; i++)
  cout << tests[i] << endl;
  ```

PROCESSING ARRAY CONTENT

SUMMING AND AVERAGING

- Use a simple loop to add together array elements:
  ```cpp
  int tnum;
  double average, sum = 0;
  for(tnum = 0; tnum < SIZE; tnum++)
  sum += tests[tnum];
  ```
- Once summed, can compute average:
  ```cpp
  average = sum / SIZE;
  ```
**PROCESSING ARRAY CONTENT**

**FINDING THE HIGHEST VALUE**

```c
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.

---

**PROCESSING ARRAY CONTENT**

**FINDING THE LOWEST VALUE**

```c
int count;
int lowest;
highest = 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 highest value in the `numbers` array.
PROCESSING ARRAY CONTENT
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.

PROCESSING ARRAY CONTENT
COMPARING ARRAYS

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

```cpp
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";
```
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
- Example

```cpp
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 ARRAYS
IN PROGRAM

Program 7.12

```cpp
// This program uses two parallel arrays: one for hours
// and one for pay rate.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    const int NUM_EMPLOYEES = 5; // Number of employees
    int hours[NUM_EMPLOYEES]; // Holds hours worked
    double payRate[NUM_EMPLOYEES]; // Holds pay rates
    // Input the hours worked and the hourly pay rate.
    cout << "Enter the hours worked by " << NUM_EMPLOYEES << " employees and their\n" << "hourly pay rates.\n";
    for (int index = 0; index < NUM_EMPLOYEES; index++)
    {
        cout << "Hours worked by employee #" << (index+1) << ": ";
        cin >> hours[index];
        cout << "Hourly pay rate for employee #" << (index+1) << ": ";
        cin >> payRate[index];
    }
}
```
PARALLEL ARRAYS

CONTINUED

```java
38 // Display each employee's gross pay.
39 cout << "Here is the gross pay for each employee:\n";
40 cont << fixed << showpoint << setprecision(2);
41 for (int index = 0; index < NUM_EMPLOYEES; index++)
42 {
43     double grossPay = hours[index] * payRate[index];
44     cout << "Employee #" << (index + 1) << " $" << grossPay << endl;
45 }
46 return 0;
47 }
```

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]

(paragraph output continues)

PARALLEL ARRAYS

CONTINUED

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: $624.00

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

```
+-------------------+-------------------+-------------------+-------------------+-------------------+
+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+
| Employee #1      |                  |                  |                  |                  |                  |
| Employee #2      |                  |                  |                  |                  |                  |
| Employee #3      |                  |                  |                  |                  |                  |
| Employee #4      |                  |                  |                  |                  |                  |
| Employee #5      |                  |                  |                  |                  |                  |
```

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>40</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>payRate[0]</td>
<td>9.75</td>
<td>8.62</td>
<td>10.50</td>
<td>18.75</td>
<td>15.65</td>
</tr>
<tr>
<td>payRate[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>payRate[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>payRate[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>payRate[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ARRAYS AS FUNCTION ARGUMENTS

- To pass an array to a function, just use the array name:
  ```c
  showScores(tests);
  ```

- To define a function that takes an array parameter, use empty [] for array argument:
  ```c
  void showScores(int []);
  ```
  ```c
  // function prototype
  void showScores(int tests[]);
  ```
  ```c
  // function header
  ```

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:
  ```c
  showScores(tests, ARRAY_SIZE);
  ```

- Array size must also be reflected in prototype, header:
  ```c
  void showScores(int [], int);
  ```
  ```c
  // function prototype
  void showScores(int tests[], int size);// function header
  ```

**Program 7-14**

```c
// This program demonstrates an array being passed to a function.
#include <iostream>
using namespace std;

void showValues(int [], int); // Function prototype

int main()
{
    const int ARRAY_SIZE = 8;
    int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
    showValues(numbers, ARRAY_SIZE);
    return 0;
}
```
Arrays as Function Arguments

CONTINUED

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

Program Output

5 10 15 20 25 30 35 40

Arrays as Function Arguments

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
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

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

- Use two subscripts to access element: 
  ```
  exams[2][2] = 86;
  ```

TWO-DIMENSIONAL ARRAYS

IN PROGRAM

*Program 7-18*

```cpp
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";
(program continues)
```
TWO-DIMENSIONAL ARRAYS

Program 7.18 (continued)

    // Nested loops to fill the array with quarterly sales figures for each division.
    for (div = 0; div < NUM_DIVS; div++)
    {
        for (qtr = 0; qtr < NUM_QTRS; qtr++)
        {
            cout << "Division " << (div + 1) << ", Quarter " << (qtr + 1) << ": ", $; "
            cin >> sales[div][qtr];
        }
        cout << endl; // Print blank line.
    }

    // Nested loops used to add all the elements.
    for (div = 0; div < NUM_DIVS; div++)
    {
        for (qtr = 0; qtr < NUM_QTRS; qtr++)
            totalSales += sales[div][qtr];
    }

    cout << fixed << setprecision(2);
    cout << "The total sales for the company are: ";
    cout << totalSales << endl;
    return 0;

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: $20654.23 [Enter]
Division 1, Quarter 3: $29682.54 [Enter]
Division 1, Quarter 4: $39651.21 [Enter]
Division 2, Quarter 1: $56321.02 [Enter]
Division 2, Quarter 2: $41286.63 [Enter]
Division 2, Quarter 3: $41283.85 [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: $456762.34
TWO-DIMENSIONAL ARRAYS

INITIALIZATION

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

```c
const int ROWS = 2, COLS = 2;
int exams[ROWS][COLS] = { {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 ARRAYS

AS PARAMETER, ARGUMENT

- Use array name as argument in function call:

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

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

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

// Header
void getExams(int exams[][COLS], int rows)
```
TWO-DIMENSIONAL ARRAYS

SHOWARRAY FUNCTION

```c
void showArray(int array[][COLS], int rows)
{
    for (int x = 0; x < rows; x++)
    {
        for (int y = 0; y < COLS; y++)
        {
            cout << setw(4) << array[x][y] << " ";
        }
        cout << endl;
    }
}
```

TWO-DIMENSIONAL ARRAYS

HOW SHOWARRAY IS CALLED

```c
int table1[TBL1_ROWS][COLS] = {{1, 2, 3, 4},
                               {5, 6, 7, 8},
                               {9, 10, 11, 12}};
int table2[TBL2_ROWS][COLS] = {{10, 20, 30, 40},
                               {50, 60, 70, 80},
                               {90, 100, 110, 120},
                               {130, 140, 150, 160}};

cout << "The contents of table1 are:\n";
showArray(table1, TBL1_ROWS);
cout << "The contents of table2 are:\n";
showArray(table2, TBL2_ROWS);
```
TWO-DIMENSIONAL ARRAYS
SUMMING ALL THE ELEMENTS

• Given the following definitions

```cpp
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}};
```

// 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;

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}};

// 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;
}
TWO-DIMENSIONAL ARRAYS
SUMMING THE COLUMNS

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}};

// 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;
}
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 but 1st dimension in prototype, heading:
  
  ```
  void getRectSolid(short [][3][5]);
  ```

STL VECTOR INTRODUCTION

• A data type defined in the Standard Template Library (covered more later)
• Can hold values of any type:
  
  ```
  vector<int> scores;
  ```

• Automatically adds space as more is needed - no need to determine size at definition
• Can use [] to access elements
STL VECTOR

DECLARING VECTORS

• You must `#include<vector>`

• Declare a vector to hold `int` element:
  ```cpp
  vector<int> scores;
  ```

• Declare a vector with initial size 30:
  ```cpp
  vector<int> scores(30);
  ```

• Declare a vector and initialize all elements to 0:
  ```cpp
  vector<int> scores(30, 0);
  ```

• Declare a vector initialized to size and contents of another vector:
  ```cpp
  vector<int> finals(scores);
  ```

STL VECTOR

ADDING ELEMENTS TO A VECTOR

• Use `push_back` member function to add element to a full array or to an array that had no defined size:
  ```cpp
  scores.push_back(75);
  ```

• Use `size` member function to determine size of a vector:
  ```cpp
  howbig = scores.size();
  ```
STL VECTOR

REMOVING VECTOR ELEMENTS

- Use the `pop_back` member function to remove the last element from the vector:
  
  ```cpp
  scores.pop_back();
  ```

- To remove all contents of the vector, use the `clear` member function:
  
  ```cpp
  scores.clear();
  ```

- To determine if the vector is empty, use the `empty` member function:
  
  ```cpp
  while (!scores.empty()) ... 
  ```

STL VECTOR

OTHER USEFUL MEMBER FUNCTIONS

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>at(elt)</code></td>
<td>Returns the value of the element at position <code>elt</code> in the vector</td>
<td><code>cout &lt;&lt; vec1.at(i);</code></td>
</tr>
<tr>
<td><code>capacity()</code></td>
<td>Returns the maximum number of elements a vector can store without allocating more memory</td>
<td><code>maxelts = vec1.capacity();</code></td>
</tr>
<tr>
<td><code>reverse()</code></td>
<td>Reverse the order of the elements in a vector</td>
<td><code>vec1.reverse();</code></td>
</tr>
<tr>
<td><code>resize(els, val)</code></td>
<td>Add elements to a vector, optionally initializes them</td>
<td><code>vec1.resize(5, 0);</code></td>
</tr>
<tr>
<td><code>swap(vec2)</code></td>
<td>Exchange the contents of two vectors</td>
<td><code>vec1.swap(vec2);</code></td>
</tr>
</tbody>
</table>