National University of computer and emerging sciences

Foundation for Advancement of Science and Technology

Programming Fundamentals

Arrays in C++

BS (SE) Fall-2025

Arrays in C++

- Definition:
  - An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.

  Or:

  - Variables to hold multiple values of same data-type
  - Values are stored in adjacent memory locations

Arrays declaration

- Array declaration requires a data-type, a name and a “constant” size of the array
  - data_type array_name [array_size]
- Data type: can be any valid data type
- Name: The rules of variable naming convention apply to array names
- Size: defines how many values can be stored in the array
- The size of the array tells how many elements are there in the array
  - The size of the array should be a precise number
- Example
  - Declared using `[]` operator:

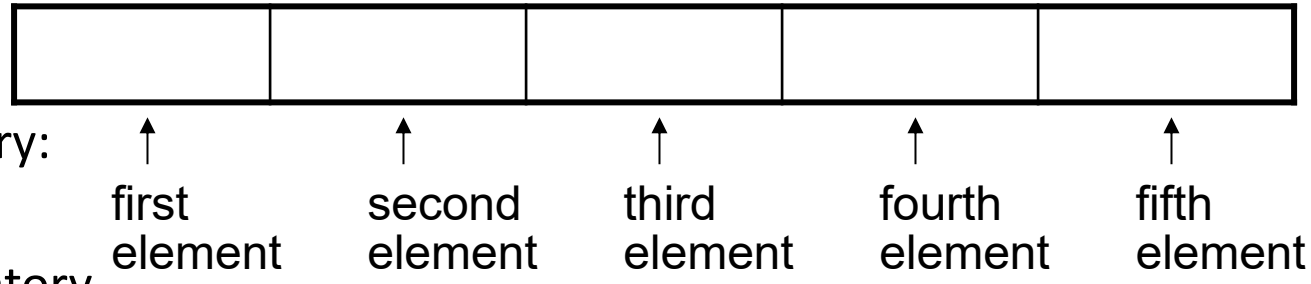
```
int tests[5];
```

Arrays memory layout

- The arrays occupy the memory depending upon their size and have contiguous area of memory
- The definition:

  `int tests[5];`

  allocates the following memory:



- `5`, in `[5]`, is the size declaratory
- 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`

In the definition `int tests[5];`

- `int` is the data type of the array elements
- `tests` is the name of the array

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.

Array Initialization

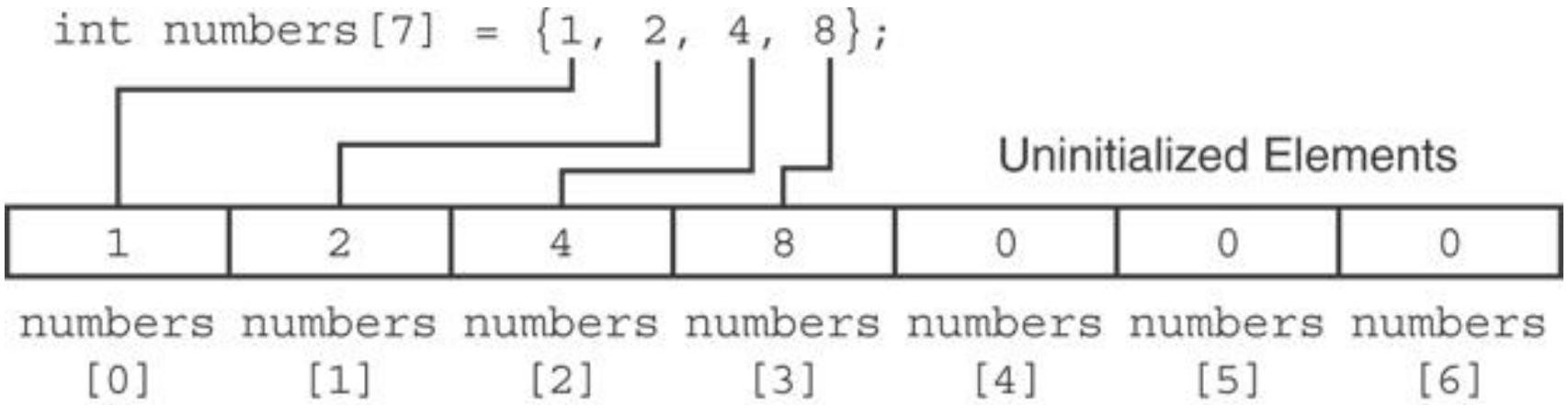
- Arrays can be initialized with an initialization list:

```
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.

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

- We can access the arrays using the array index (index also called subscript)

Arrays in C++

- Variables used so far are designed to hold only one value at a time
  - Examples: int count = 12345; OR float price = 52.5; OR char letter = ‘A’;
- Arrays (Definition): An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier. Or variable that hold multiple values of same type
- Application:
  - Suppose a class has 50 students, and you need to store the grades of all of them
  - Instead of creating 50 separate variables, you can simply create an array

    For example: double grade[50];
  - grade is an array that can hold a maximum of 50 elements of double type
  - In C++, the size and type of arrays cannot be changed after its declaration

Example Program (variable vs Arrays)

```
int main() {
        //Declaring int variables for class of PF to store the Quiz3 marks of students
        int std1Marks = 50;
        int std2Marks = 45;
        int std3Marks = 65;
        int std4Marks = 80;
        int std5Marks = 95;
        cout<<"Printing students mark using variables"<<endl;
        cout<<"student 1 Marks "<<std1Marks<<endl;
        cout<<"student 2 Marks "<<std2Marks<<endl;
        cout<<"student 3 Marks "<<std3Marks<<endl;
        cout<<"student 4 Marks "<<std4Marks<<endl;
        cout<<"student 5 Marks "<<std5Marks<<endl;
        cout<<"Printing students mark using Arrays"<<endl;
        //Declare an int array for class of PF to store the Quiz3 marks of students
        int stdMarks[5] = {50, 45, 65, 80, 95};
        for(int i=0; i<=4; i++)
        cout<<"student "<<i+1<<" Marks "<<stdMarks[i]<<endl;
        return 0;
}
```

Output:

```
Printing students mark using variables
student 1 Marks 50
student 2 Marks 45
student 3 Marks 65
student 4 Marks 80
student 5 Marks 95
Printing students mark using Arrays
student 1 Marks 50
student 2 Marks 45
student 3 Marks 65
student 4 Marks 80
student 5 Marks 95
```

Array’s terminologies in C++

- Array Index:
  - The index of an element in an array identifies the element’s location. The array index starts at 0
- Array element:
  - Elements are the items contained in an array. The elements can be found using the index
- Array size:
  - The size of an array is determined by the number of elements it can hold
- Array dimension:
  - Array dimension is a direction in which you can vary the specification of an array's elements.

Example of Array

- Let A is an integer array of size 8 and having elements of 11, 23, 65, 88, 39, 49, 83, 22, 84
- Array representation

int A[9] = {11, 23, 65, 88, 39, 49, 83, 22, 84}

For accessing Array Elements/members

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

Array indices/ subscripts

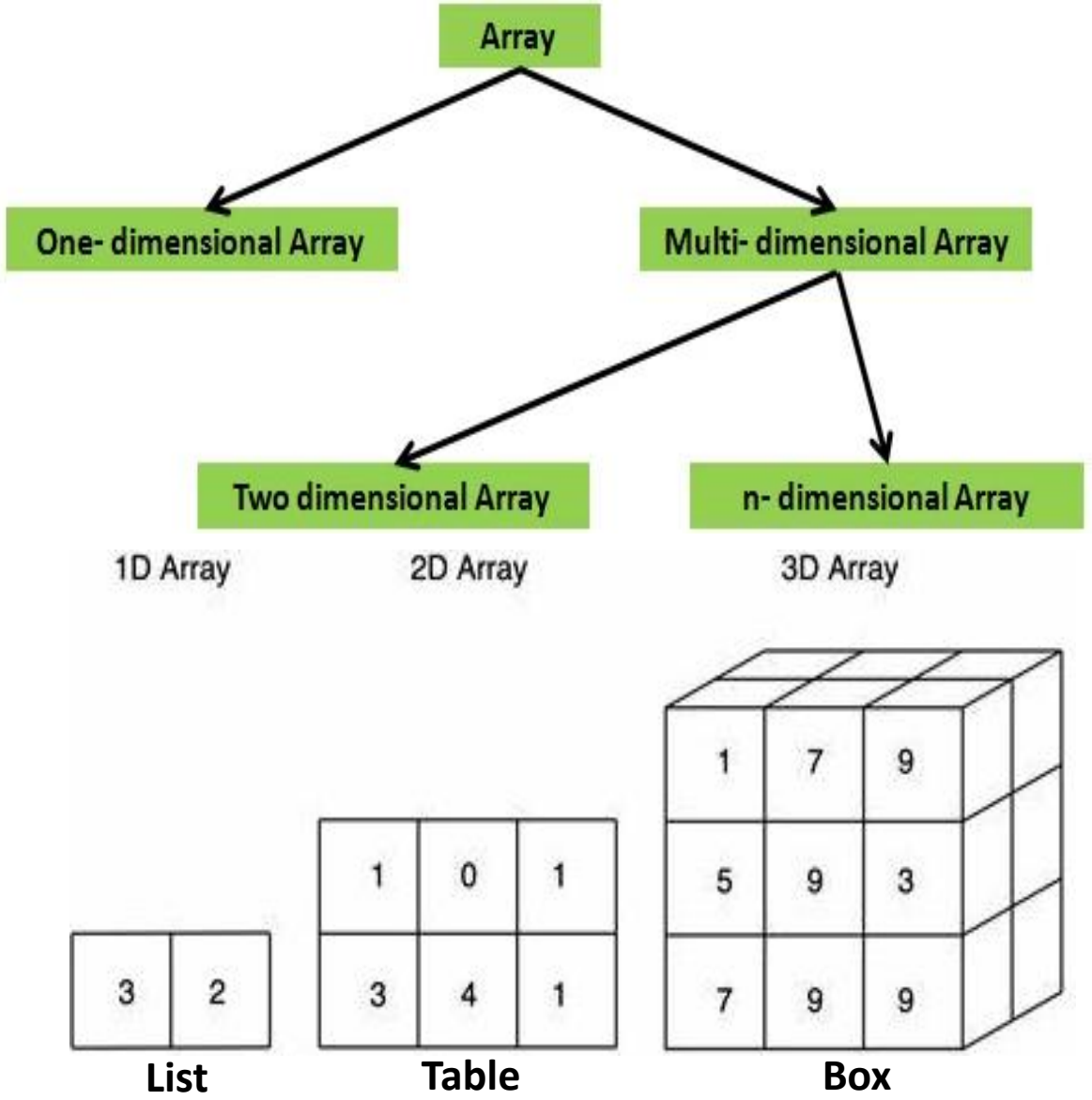
0	1	2	3	4	5	6	7	8

Values stored in Array

11	23	65	88	39	49	83	22	84

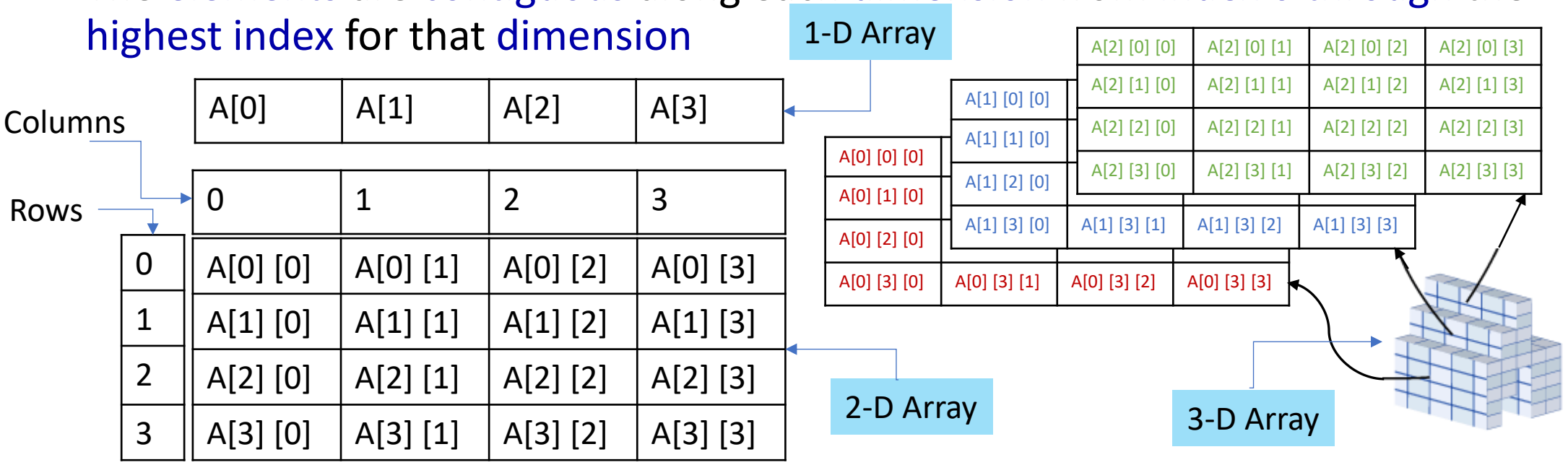
Types of arrays

- One dimensional array (1D array): The one-dimensional arrays stores values in the form of a list or Vector
- Multi-dimensional array: (arrays of arrays) stores the value in the form c a matrix
  - Two dimensional array or 2D array (example: Table)
  - Three dimensional arrays or 3D array (example: Box)



Arrays Dimensions

- A dimension is a direction in which you can vary the specification of an array's elements
- And index or subscript is used to specify an element of an array for each of its dimensions
- The elements are contiguous along each dimension from index 0 through the highest index for that dimension



Size of the Array

- Let the declaration of an array tests: `int tests[5];`
  - `5`, in `[5]`, is the size declaration
  - It shows the number of elements in the array.
- The arrays occupy the memory depending upon their size and have contiguous area of memory
- 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`

Array Definition	Number of Elements	Size of Each Element	Size of the Array
`char letters[25];`	25	1 byte	25 bytes
`short rings[100];`	100	2 bytes	200 bytes
`int miles[84];`	84	4 bytes	336 bytes
`float temp[12];`	12	4 bytes	48 bytes
`double distance[1000];`	1000	8 bytes	8000 bytes

Example program

```
#include <iostream>
 using namespace std;

int main() {
        int iarray[] = {11, 10, 17, 29, 20,
31, 34};
        cout <<sizeof(iarray);
        return 0;
}
```

Output: 28

```
#include <iostream>
using namespace std;
int main() {

        //Declare and assign int array type
        int iarray[] = {5, 10, 15, 20, 25, 30, 35};
        //Getting the array length
        int array_size = sizeof(iarray) / sizeof(int);
        //Display the array length
        cout <<"Length of array: " << array_size;
        return 0;
      }
```

Length of array: 7

Copying arrays

- For being copy able, both arrays need to be of same data type and same size
- Examples:

```
int a [10];
int b [10];
```

- Value can be assigned to an element of array using the index
- Don't try to assign one array to the other:

```
a = b;  // Won't work
```

- Instead, assign element-by-element:

```
for (int i = 0; i < 10; i++)
        a[i] = b[i];
```

- We can write assignment statements to copy these arrays as:

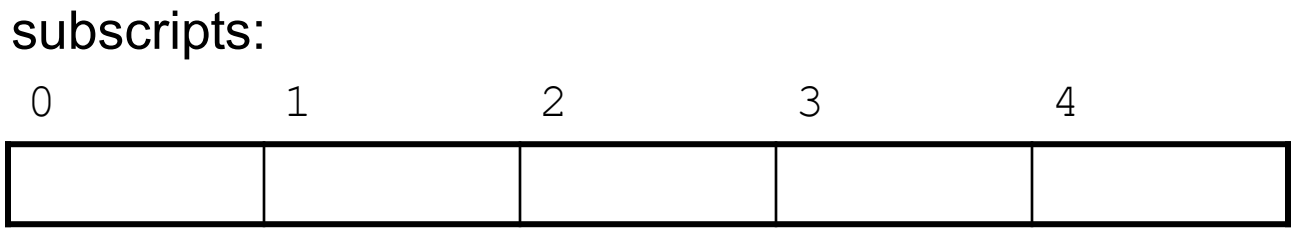
```
b[0] = a[0] ;
b[1] = a[1] ;
b[2] = a[2] ;
……
b[9] = a[9] ;
```

We can use the loop to deal with it easily

```
for (i = 0; i < 10 ; i ++)
  {
        b[i] = a[i];
  }
```

Accessing Array Elements

- Each element in an array is assigned a unique index or subscript.
  - We can access the arrays using the array index (index also called subscript)
  - Subscripts start at 0
  - The last element's subscript is n-1 where n is the number of elements in the array.



Accessing Array Elements

- To access array, we can not use the whole array at a time. However, arrays can be accessed element by element
  - Subscript numbering in C++ always starts at zero
  - To access first element we write like tests[0]
  - To access the 5th element, we will write tests[4] and so on
  - Array elements can be use as regular variables using the index mechanism
- Examples:

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

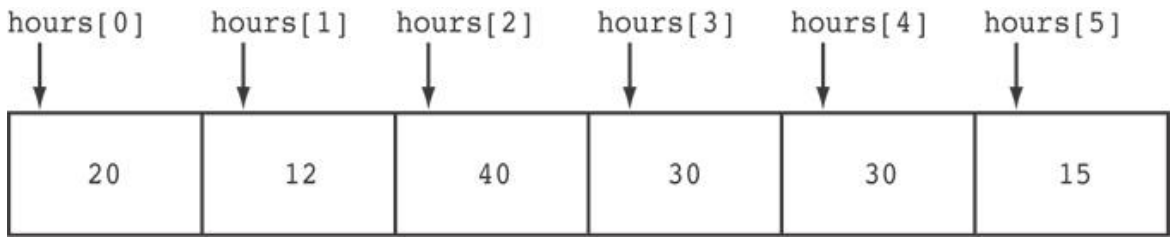
Note: Arrays must be accessed via individual elements: cout << tests; // not legal

Example Program

```
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 each employee.
12    cout << "Enter the hours worked by "
13         << NUM_EMPLOYEES << " employees: ";
14    cin >> hours[0];
15    cin >> hours[1];
16    cin >> hours[2];
17    cin >> hours[3];
18    cin >> hours[4];
19    cin >> hours[5];
20
21    // Display the values in the array.
22    cout << "The hours you entered are:";
23    cout << " " << hours[0];
24    cout << " " << hours[1];
25    cout << " " << hours[2];
26    cout << " " << hours[3];
27    cout << " " << hours[4];
28    cout << " " << hours[5] << endl;
29    return 0;
30 }
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 6 employees: 20 12 40 30 30 15 [Enter]
The hours you entered are: 20 12 40 30 30 15
```

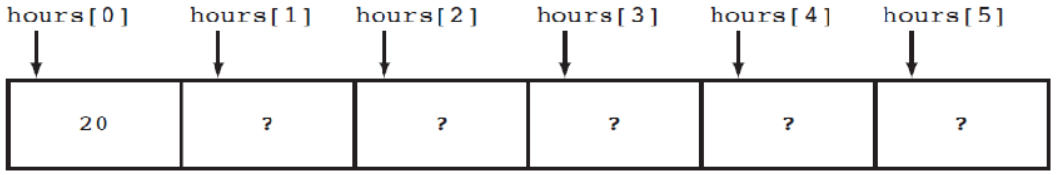


Here are the contents of the `hours` array, with the values entered by the user in the example output:

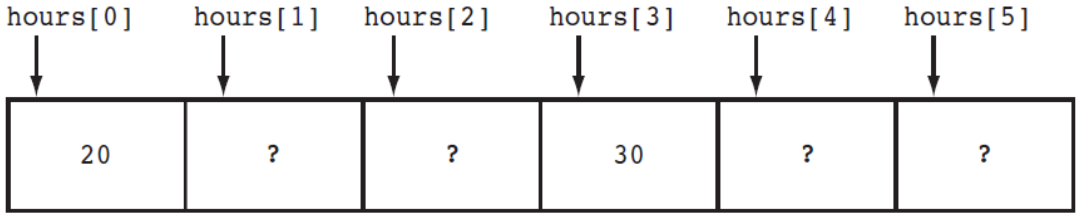
Accessing Array Elements (Example)

- Let an array: short hours[6];
  - Elements of array hours starts from hours[0] to hours[5] and the element hours[6] does not exist
  - Note: If an array is defined globally, all of its elements are initialized to zero by default
  - The Local arrays, however, have no default initialization value or uninitialized by default

- Example: hours[0] = 20;
  - No values have assigned to the other elements now and those are unknown



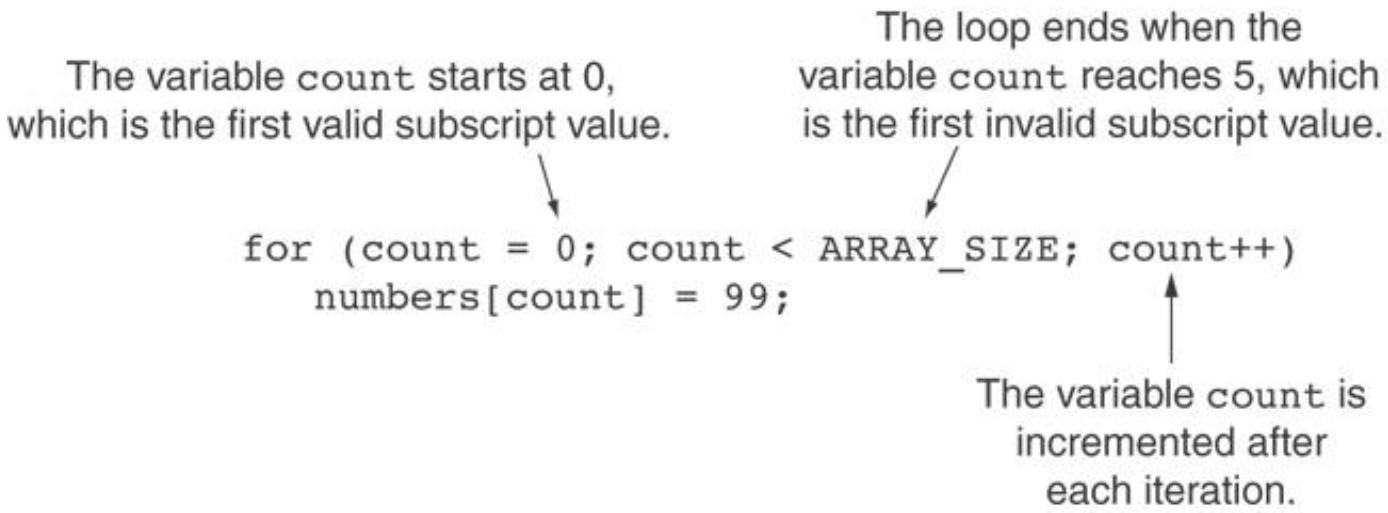
- Example: Also assign a value to hours [3] e.g., hours[3] = 30;



Using a loop to Step through an Array

- Example: A Closer Look At the Loop:
  - 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;
```



Using a loop to Step through an Array

```
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  int main ()
6  {
7     const int NUM_EMPLOYEES = 6; // Number of employees
8     int hours[NUM_EMPLOYEES]; // Each employee's hours
9     int count; // Loop counter
10 // Input the hours worked.
11    for (count = 0; count < NUM_EMPLOYEES; count++)
12    {
13       cout << "Enter the hours worked by employee" << (count + 1) << ": ";
14       cin >> hours[count];
15    }
16 // Display the contents of the array.
17    cout << "The hours you entered are:";
18    for (count = 0; count < NUM_EMPLOYEES; count++)
19       cout << " " << hours[count];
20    cout << endl;
21    return 0;
22 }
```

Output:

```
Enter the hours worked by employee1: 20
Enter the hours worked by employee2: 12
Enter the hours worked by employee3: 40
Enter the hours worked by employee4: 30
Enter the hours worked by employee5: 30
Enter the hours worked by employee6: 15
The hours you entered are: 20 12 40 30 30 15
```

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
- Note: Be careful not to use invalid subscripts.
  - Doing so can corrupt other memory locations, crash the program, or lock up the computer, and cause elusive bugs.
- Example:
  - The following code in the example program defines a three-element array, and then writes five values to it!

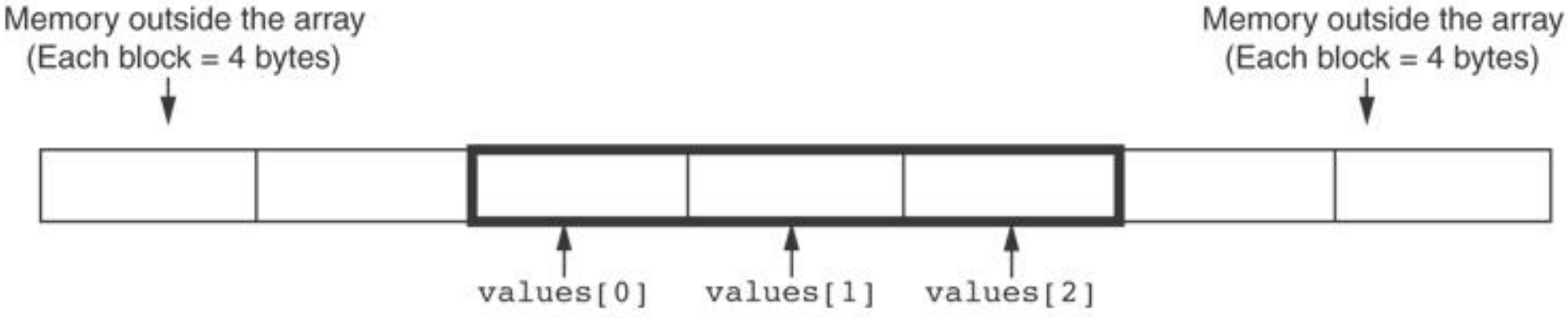
Example program

```
1  // This program unsafely accesses an area of memory by writing
2  // values beyond an array's boundary.
3  // WARNING: If you compile and run this program, it could crash.
4  #include <iostream>
5  using namespace std;
6  int main ()
7  {
8     const int SIZE = 3; // Constant for the array size
9     int values[SIZE]; // An array of 3 integers
10    int count; // Loop counter variable
11 // Attempt to store five numbers in the three-element array.
12    cout << "I will store 5 numbers in a 3-element array!\n";
13    for  ( count = 0; count < 5; count++)
14       values[count] = 100;
15 // If the program is still running, display the numbers.
16    cout << "If you see this message, it means the program\n";
17    cout << "has not crashed! Here are the numbers:\n";
18    for  ( count = 0; count < 5; count++)
19       cout << values[count] << endl;
20    return 0;
21 }
```

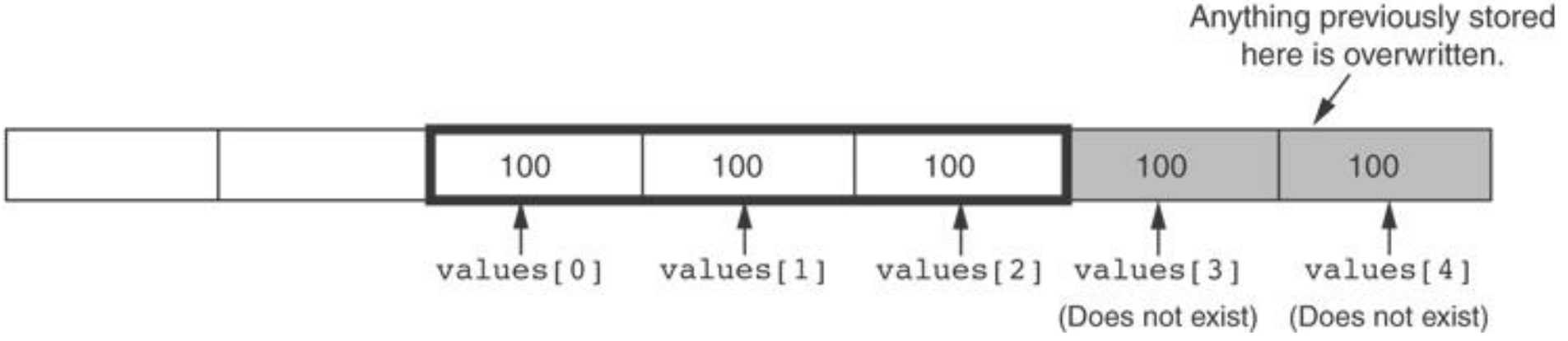
Output:
I will store 5 numbers in a 3-element array!
If you see this message, it means the program has not crashed! Here are the numbers:
100
100
100
100
4

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.



Invalid subscripts and off-by-one error

- 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:

```
// 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;
```

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]

ement i, and no effect on tests

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5     int i=2;
6     int tests[i]= {5, 2, 9} ;
7     tests[0]++; // add 1 to tests[i]
8     cout<<"tests[i]++ = "<<tests[0]<<endl;
9     cout<<"tests[i+1] ="<<tests[i++]<<endl;
10    cout<<" i = "<<i;
11    return 0;
12 }
```

Output:

tests[i]++ = 6

tests[i+1] =9

i = 3

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];
```

Processing array contents

- 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!
- 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;
```

Example Program

```
1  // This program calculates the sum of squares of numbers stored in an array.
2  #include <iostream>
3  using namespace std;
4  main ()
5  {
6    int a[10];
7    int sumOfSquares = 0 ;
8    int i =0;
9    cout << "Please enter the ten numbers one by one " << endl;
10 // Getting the input from the user.
11   for  (i = 0 ; i < 10 ; i++ )
12    {
13      cin >> a [i];
14   }
15 // Calculating the sum of squares.
16   for  ( i = 0 ; i < 10 ; i ++ )
17    {
18      sumOfSquares = sumOfSquares + a[ i ] * a[ i ] ;
19   }
20   cout <<"The sum of squares is" << sumOfSquares << endl;
21 }
```

Output:

Please enter the ten numbers one by one
8
9
4
5
10
5
7
5
8
9
The sum of squares is 530

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.

Example program

```
1  #include<iostream>
2  using namespace std;
3  int main ()
4  {
5     const int SIZE = 5;
6     int firstArray[SIZE] = { 5, 10, 15, 20, 25} ;
7     int secondArray[SIZE] = { 5, 10, 15, 20, 25} ;
8     bool arraysEqual = true; // Flag variable
9     int count = 0;          // Loop counter variable
10 // Compare the two arrays.
11    while (arraysEqual && count < SIZE)
12    {
13       if (firstArray[count] != secondArray[count])
14          arraysEqual = false;
15       count++;
16    }
17    if (arraysEqual)
18       cout << "The arrays are equal.\n";
19    else
20       cout << "The arrays are not equal.\n";
21    return 0;
22 }
```

Output:
The arrays are equal.

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
- 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;
}
```

Example program

```
1  // This program uses two parallel arrays: one for hours
2  // worked and one for pay rate.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     const int NUM_EMPLOYEES = 5;    // Number of employees
10    int hours[NUM_EMPLOYEES];       // Holds hours worked
11    double payRate[NUM_EMPLOYEES]; // Holds pay rates
12
13    // Input the hours worked and the hourly pay rate.
14    cout << "Enter the hours worked by " << NUM_EMPLOYEES
15         << " employees and their\n"
16         << "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

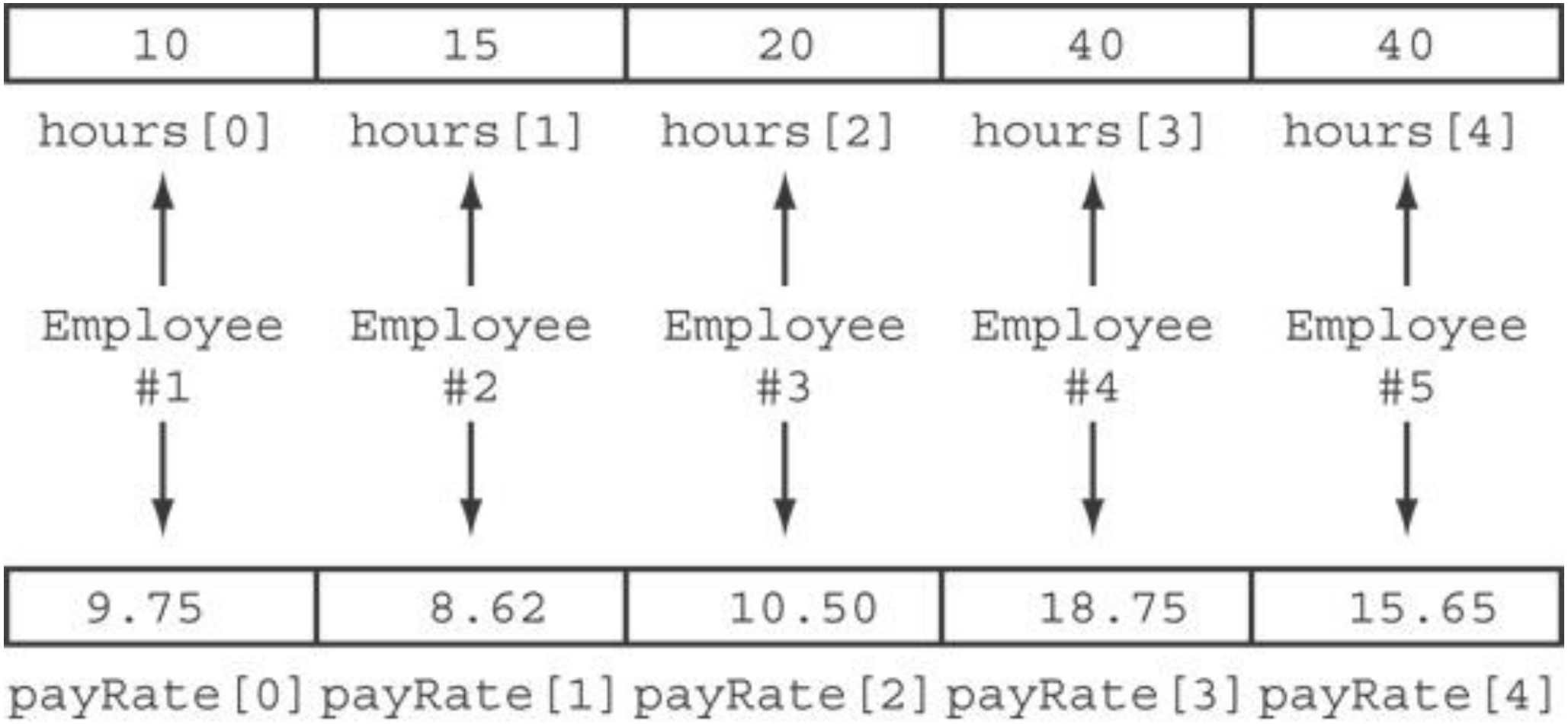
```
25    // Display each employee's gross pay.
26    cout << "Here is the gross pay for each employee:\n";
27    cout << fixed << showpoint << setprecision(2);
28    for (int index = 0; index < NUM_EMPLOYEES; index++)
29    {
30       double grossPay = hours[index] * payRate[index];
31       cout << "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]
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
```

Parallel arrays

- The hours and payRate arrays are related through their subscripts



Task

- At a local bookstore, the manager wants to keep track of the stock of different book titles. You need to write a C++ program that stores the names, prices, and available quantities of each book using arrays. Ask the manager for the number of records they want to keep. The program should display all books with their total stock value (price × quantity)

- The program should also identify which book is the most and least expensive.

Two-Dimensional Arrays

Two-Dimensional Arrays

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

  Syntax: Data_Type Array_Name [x][y];

```
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;
```

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} };
```

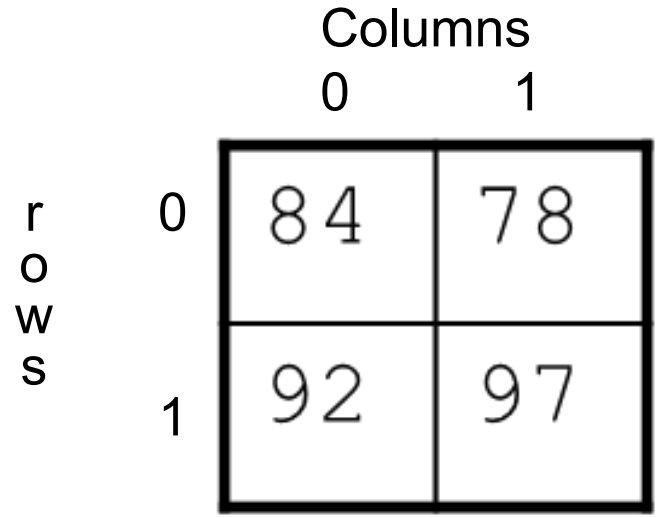
84	78
92	97

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

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

Accessing elements of 2-D array

- Using row index and column index



- Use two subscripts to access element:

```
exams[1][1] = 97;
cout<<[1][1]; //will print 97
```

Example program

```cpp
// C++ Program to display all elements
// of an initialised two dimensional array

#include <iostream>
using namespace std;

int main() {
    int test[3][2] = {{2, -5},
                      {4, 0},
                      {9, 1}};

    // use of nested for loop
    // access rows of the array
    for (int i = 0; i < 3; ++i) {

        // access columns of the array
        for (int j = 0; j < 2; ++j) {
            cout << "test[" << i << "][" << j << "] = " << test[i][j] << endl;
        }
    }

    return 0;
}
```

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```

Size of 2-D array

- The size of two dimensional array is equal to the total number of elements in the 2-D array
- Example: int test [2][3];

  size of 2-D array = number of rows x number of columns= 2x3 = 6

  size of 2-D array in bytes = 6 x 4 = 24 bytes

Example Program

```
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
```

Example Program

```
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   }
```

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
```

Summing All the Elements in a 2-D Array

- Given the following definitions:

```
const int NUM_ROWS = 5; // Number of rowsconst 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 2-D 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 2-D 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 2-D 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 2-D 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 2-D 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;
}
```

Char array

```
char str[3][5] = {'a','b','c','d', 'e','f','g','h','i'
                  'j','k','l','m','n','o'};
```

	0	1	2	3	4
0	a	b	c	d	e
1	f	g	h	i	j
2	k	l	m	n	o

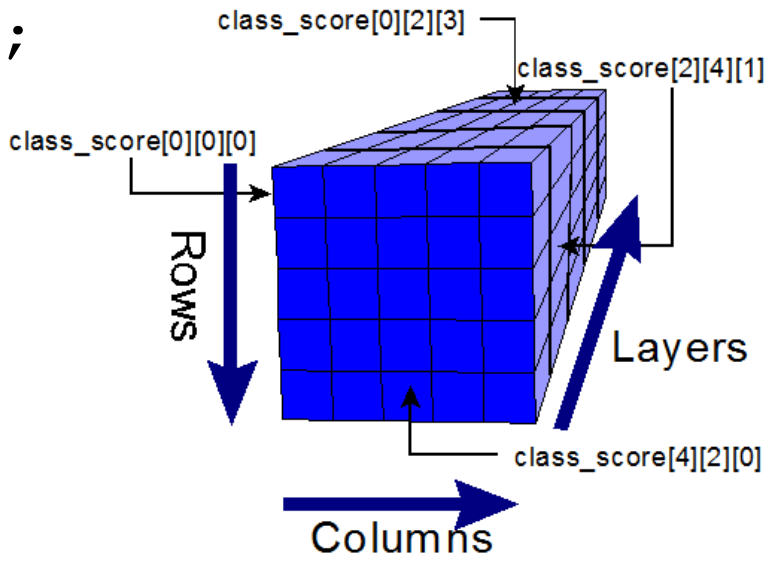
Arrays with Three or More Dimensions

- 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]);
```



3D Array Example

- Imagine you have 2 schools, each with 3 classes, and each class has 4 students. We'll use a 3D array to store the marks of every student.

```
int marks[2][3][4] = {
      {   // School 1
         {85, 90, 78, 88}, // Class 1
         {75, 80, 72, 70}, // Class 2
         {92, 88, 95, 91}  // Class 3
      },
      {   // School 2
         {65, 70, 68, 72}, // Class 1
         {80, 82, 78, 84}, // Class 2
         {90, 85, 88, 92}  // Class 3
      }
  };
```

3D Array Example

```
for (int school = 0; school < 2; school++) {
        cout << "\nSchool " << school + 1 << ":\n";
        for (int cls = 0; cls < 3; cls++) {
            cout << "  Class " << cls + 1 << ": ";
            for (int student = 0; student < 4; student++) {
                cout << marks[school][cls][student] << " ";
            }
            cout << endl;
        }
    }
```

Sample Programs

- **Reverse Array**
- program takes an array of integers and prints it in reverse order.
- **Count Even and Odd Numbers**
- program counts how many even and odd numbers are in the array.
- **Identify Duplicates from an Array**
- Program identifies elements from an array and prints the unique elements.
- **Transpose of a Matrix**
- program calculates the transpose of a given 2D matrix.

Sample Programs

- **Matrix Multiplication**
- program multiplies two matrices, which are represented as 2D arrays.
- **Sort**
- program sorts an array and counts the number of swaps.
- **Find Longest Increasing Subsequence**
- program finds the length of the longest increasing subsequence in an array.