Arrays

(CS 1002)

Dr. Muhammad Aleem,

Department of Computer Science,

National University of Computer & Emerging Sciences,

Islamabad Campus

Arrays

- **Collection data items**
- **Collection** of the **same types** of **data**.
- **Static entity** – **Same size** throughout **program**

Arrays

- **Simple data type** => a **single value**

15   84.35   ‘A’

- **Structured data type** => a **collection of data values**

- **Array** is a **structured data-type** (collection of values)

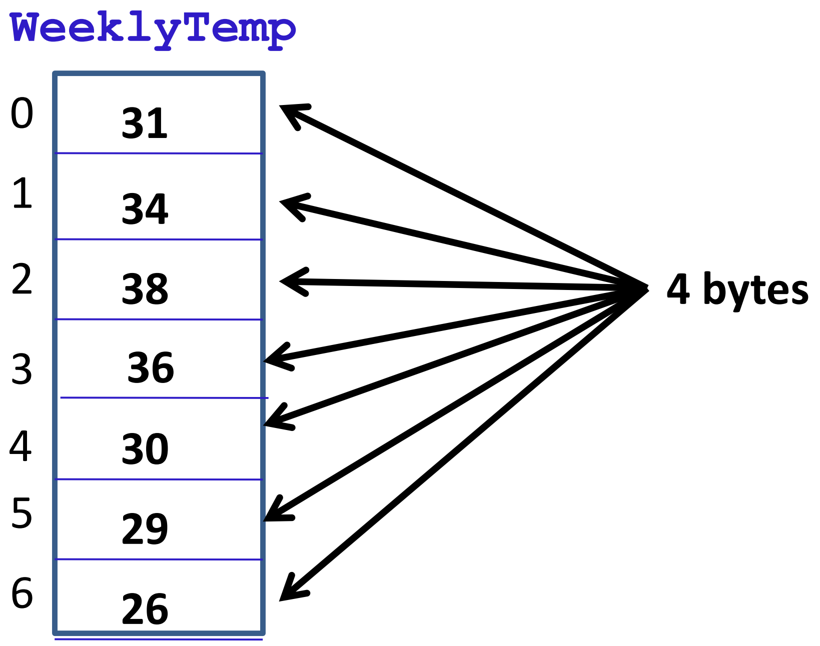
85	79	92	57	68	80

One Dimensional Array

- **Collection of components**
  - **All of the same type**
- Structure **given** a **single name**
- Individual **elements accessed** by **index** indicating ***relative position in collection***
- **Index of an array must be an integer**

One Dimensional Array

```
int WeeklyTemp[7];
```



```
cout<<WeeklyTemp[0];
cout<<WeeklyTemp[2];
cout<<WeeklyTemp[4];
```

Declaring Array Variables

```
datatype arrayName[arraySize];
```

```
Example: double myList[10];
```

Array Size: must be constant (i.e., constant literal, constant identifier)

```
int size = 4;
double myList[size];  // Wrong
const int size = 4;
double myList[size]; // Correct
double myList[20]; // Correct
```

Input/Output of Array elements

```
int marks[3];
marks[0] = 76;
marks[1] = 65;
marks[2] = 27;

cout<<marks[2]<<marks[0]<<marks[1];
```

Input/Output of Array elements – Using Loops

```
int marks[5];
for(int i=0;i<5;i++)
    cin>>marks[i];

for(int j=0;j<5;j++)
    cout<<marks[j];
```

Indexed Variables

- **Array elements** are **accessed through** the **index**.
- Array indices are **0-based**; that is, ***they start from 0 to arraySize-1,*** example:

```
            int marks[5];

→Five int values:  marks[0]
                    marks[1]
                    marks[2]
                    marks[3]
                    marks[4]

→Using array values:

            marks[2] = marks[1] + marks[0];
```

No Bound Checking

- **C++ does not check array's boundary.**

- **Subscripts (index variable)** beyond the boundary <u>*does not does not cause syntax errors*</u>,

- Operating system may report a **memory access violation** (***Compiler*** *or* ***System*** *may crash!*)

- Example: …

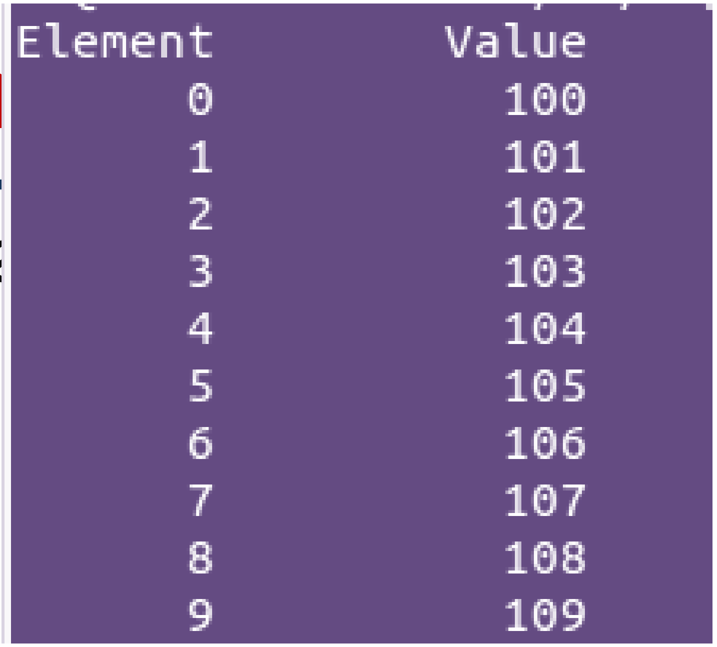
Arbitrary Initial Values

- When an array is created, its **elements** are **assigned** with **arbitrary values.**

```
int marks[5];
for(int i=0;i<5;i++)
     cout<<marks[i];
```

1D Array Exan

```
int n[10]; //n is an array of 1(

// initialize elements of array
for (int i = 0; i<10; i++) {
        n[i] = i + 100;
}
cout<<"Element"<<setw(13)<<"Value"<<endl;

// output each array element's value
for (int j = 0; j<10; j++) {
        cout<<setw(7)<<j<<setw(13)<<n[j]<<endl;
}
```



```
Element        Value
      0          100
      1          101
      2          102
      3          103
      4          104
      5          105
      6          106
      7          107
      8          108
      9          109
```

Initializing an Array

- Declaring, creating, initializing in one step:

```
dataType arrayName[Size] = {value0, value1, ..., valuek};
```

- Example:

```
int myList[4] = {32, 11, -6, 65};
```

```
What about:
  int List2[4];
  List2= {32, 11, -6, 65};
```

Implicit Size

- **C++ allows** you to **omit the array size, example:**

```
int myList[ ]={63,9,3,13};
```

C++ automatically figures out how many elements are in the array.

```
int  myList [ ];  //WRONG
```

Partial Initialization

- **Initializing a part of the array:**

double myList[4] = {1.9, 4.65}; //correct

The above example assigns values **1.9**, **4.65** to the **first two elements** of the **array**.

- The **other two elements** will be **set to zero.**

Initializing arrays with random values

- Following **loop initializes** the array **myList** with random values between **0** and **99**:

```
int myList[10];

for (int i = 0; i < 10; i++) {
myList[i] = rand( ) % 100;
cout<<“\nArray Element”<<i<<“ has val:”<<myList[i];
}
```

Copying Arrays

- Can you copy array using a syntax like this?

```
int list[3];int myList[3];
list = myList;  //Wrong
```

•Copy **individual elements** from **one array to the other** as follows:

```
for (int i = 0; i < 3; i++) {
        list[i] = myList[i];
}
```

C-Strings or Character Arrays

- The **elements** of an **array** can be just about **anything** (**any-datatype**)

- Consider an array whose elements are **all characters** (**char type**)
  - Called a **C-String**
  - **Treated differently** for **I/O** than **other types of arrays**

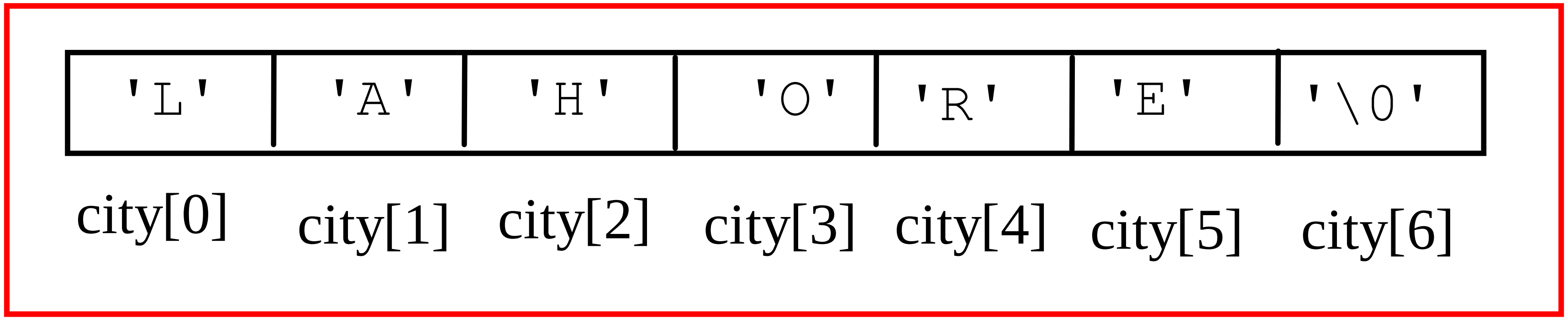
Declaration of C-Strings

- Similar to declaration of any array:

```
char  name[30]; // no initialization

char  title[20] = “Hello World";
//initialized at declaration with a string

char chList[6] = {‘H', ‘e', ‘l', ‘l‘, ’o’};
//initialized with list of char values
```

Initializing Character Arrays

```
char city[ ] =  “LAHORE";
```



Printing Character Array

- For a **character array**, it can be **printed using one print statement**.

- **Character arrays** are **handled differently** than **other types of arrays**

For example:

```
char city[ ] = “Lahore";
cout << city;  //Correct

int marks [ ] = {20,65,30};
cout << marks;  //Wrong
```

Character Array (string) Input

- Declare strings **1 element bigger** than **planned size** to allow for **‘\0’ (null character)**

```
    char city[10];
    cin>>city;  //User enters  Islamabad (9 chars)
```

- When input takes place, **C++ automatically places the ‘\0’** in memory at the **end of the characters** typed in

Example-1: Summing All Elements

- Write a program to create an array of 100 elements, assign each element with the same value (its index uses). Sum all the array values and print the Sum.

Example-2: Reversing an Array

Write a program to create an array of 10 elements, assign each element a random value (1 to 50). Print the array values. Then, Reverse the values stored in array. Output the final array values.

Example-3: Searching in Array

- Write a program that creates an integer array having 50 elements. Then, ask the user to input values in the array. After that, find the largest number, smallest number in the and calculate the average of the values in the array.

Example-4: Searching in Array

-Write a program that creates an integer array having 100 elements. Then, randomly assign values (0—99) to the arrays elements. After that the program should ask the user to enter a number and print the total number of occurrences (how many time the number appeared) in the array.

-Example:

```
Enter the number: 29
The number 29 appeared 7 times in the array
```

Example-5: Finding Element (Searching)

Write a program to create an array of 50 elements, initialize each element random value (1 to 100). Find the location (index) of the value entered by user. In the end, print both the index and largest value.

Example output:

```
Enter a number to search: 44
44 is at location 6
```

Sorting

Sorting An Array

Sorting: Arranging values of an array in *Ascending* or Descending order.

E.g., 65, 34, 12, 7, 5, 2, 1 *{Descending order}*

3, 13, 23, 37, 49, 87 *{Ascending order}*

Bubble Sort

- Repeatedly stepping through the array to be sorted, comparing each pair of adjacent items and swapping them *if they are in the wrong order*. The pass through the array is repeated until no swaps are needed (which indicates that the list is sorted)

Sorting An Array(Ascending) (bubble sort)

6 5 3 1 8 7 2 4

Sorting An Array(Ascending)

```
int a[10]={33,10,1,87,6,44,23,3,11,82};
int i,j,temp;
int N=10;

for (i=0; i<N; i++)
{
     for (int j=0; j<N-1-i; j++)
          if (a[j] > a[j+1])
          {    temp = a[j];
               a[j] = a[j+1];
               a[j+1] = temp;
          }
}
```

Sorting An Array(Descending)

```
int a[10]={33,10,1,87,6,44,23,3,11,82};
int i,j,temp;
int N=10;

for (i=0; i<N; i++)
{
     for (int j=0; j<N-1-i; j++)
          if (a[j] < a[j+1])
          {    temp = a[j];
               a[j] = a[j+1];
               a[j+1] = temp;
          }
}
```

Two Dimensional Arrays

Two Dimensional Arrays

- A **two dimensional array** **stores data** as a logical **collection** of **rows** and **columns**
- Each **element** of a **two-dimensional array** has a **row position** and a **column position** (indicated by ***two indexes***)
- **To access** an element in a **two-dimensional array**, you must specify the **name of the array** followed by:
  - a **row index**
  - a **column index**

Declaration and Initialization

- **Declaration** of a **two-dimensional array** requires a **row size** and a **column size**

- A **consecutive** block of **(row size)*****(column size) memory locations** are **allocated**.

- All **array elements** must be of the **same type.**

- **Elements accessed** by two offsets: a ***row offset*** and a ***column offset.***

2D Array - Example

```
//Declaration
int data[2][3];
```

	col 0	col 1	col 2
row 0	?	?	?
row 1	?	?	?

Declaring 2D Arrays

```
datatype arrayName[rowSize][coulmnSize];
```

```
Example:
     double myList[2][4];
```

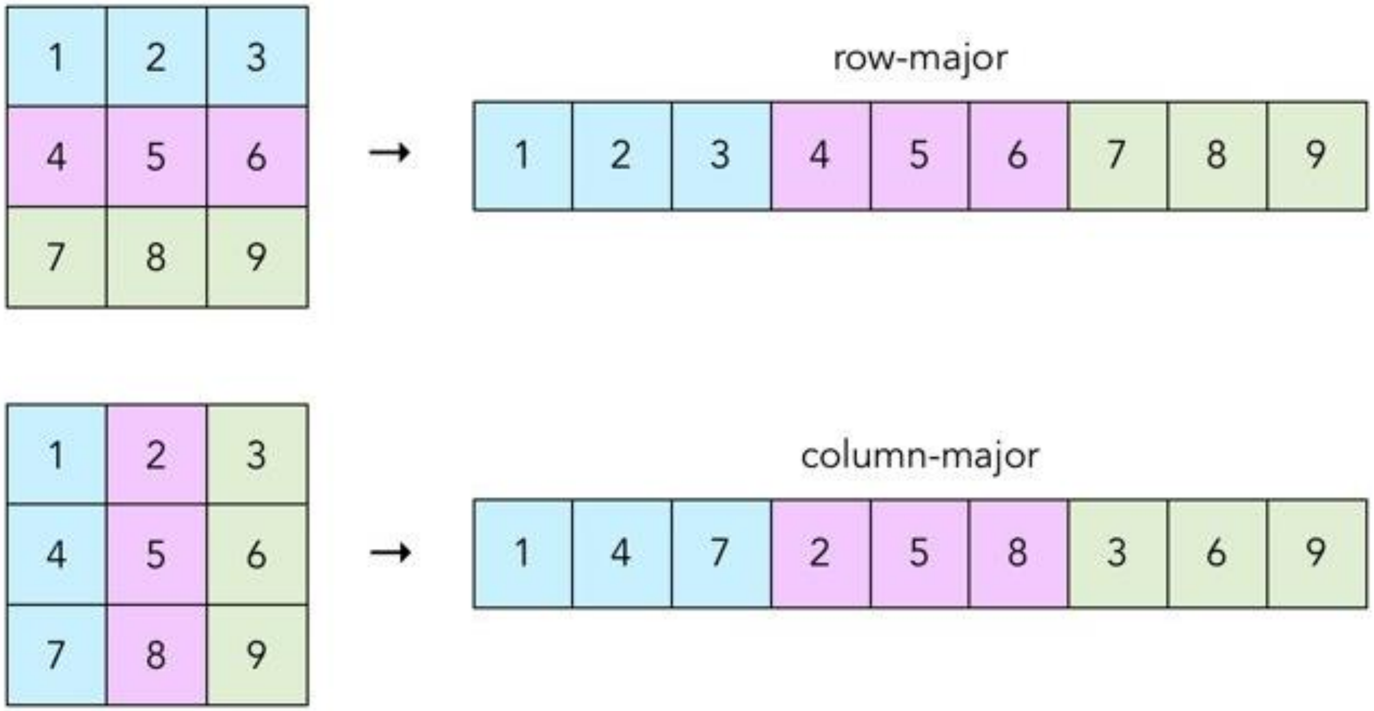
rowSize, and coulmnSize: MUST BE constant

- constant literal
- constant identifier

2D Arrays in Memory

- **Two Possibilities:**
  1. **Row-major order**
  2. **Column-major order**

C/C++ follows row-major order

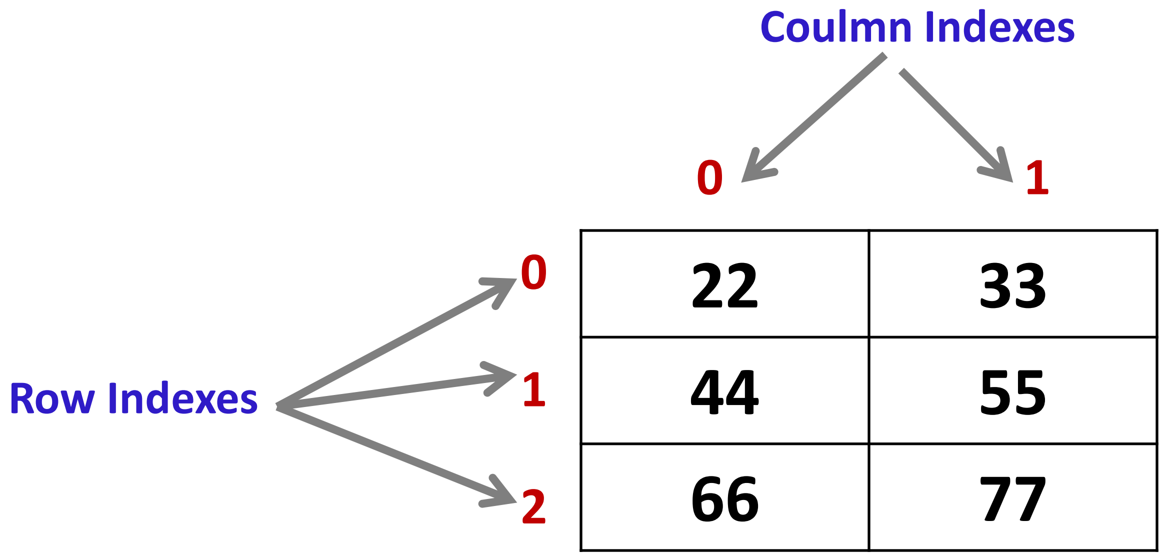


https://craftofcoding.files.wordpress.com/2017/02/rowcolumnarrays.jpg

Declaring and Initializing Arrays

```
int myList[3][2]={{22,33}, {44,55}, {66,77}};
```

→ `myList has 3 Rows and 2 coulmns in each row:`



	0	1
0	22	33
1	44	55
2	66	77

Initialization Examples

```
int temp[4][3] = {{50, 70, 60}, {48, 75, 62},
                  {51, 69, 60}, {52, 78, 63}};


int t2[7][4]   =  {{50, 70, 60}, {48, 75, 62},
                   {51, 69, 60}, {52, 78, 63}};


int temp[][3]  = {{50, 70, 60}, {48, 75, 62},
                  {51, 69, 60}, {52, 78, 63}};
```

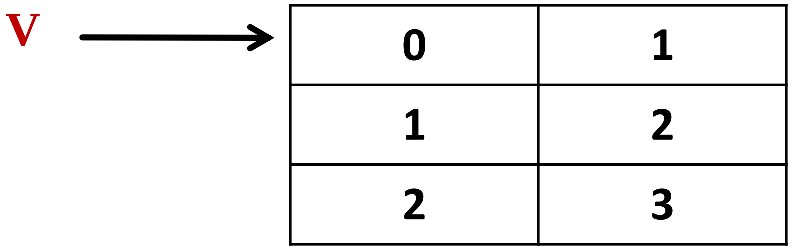
Example: Input Using *cin*

- **Nested for loops** are often used when **inputting** and **assigning values** to a **two-dimensional array**

```
//Declaration
double table[10][10];

for (int i=0; i<10; i++)      //every row
   for (int j=0; j<10; j++) //every col
        cin >> table[i][j];
```

Example: *Assignment*

```
//Declaration
const int RSIZE=3;
Const int CSIZE=2;
double v[RSIZE][CSIZE];


for (int i=0; i<RSIZE; i++)    //every row
  for (int j=0; j<CSIZE; j++ )//every col
          v[i][j] = i+j;
```



0	1
1	2
2	3

Example: Computations

- Compute the **average value** of an **matrix** with ***n*** **rows** and ***m*** **columns**.

```
double sum=0;
double average;

for (int i=0; i<n; i++)   //every row
   for (int j=0; j<m; j++ )//every col
        sum += array[i][j];

average = sum / (n*m);
```

2D Array Example-1

Compute the average value of the 3rd row of a 2D array with r rows and c columns.

```
double sum=0;
double rowAverage;

for (int j=0; j<c; j++) //every col
    sum += array[2][j];

average = sum / c;
```

Outputting 2D Arrays

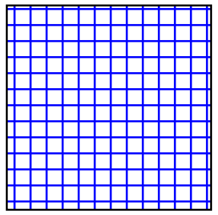
- Two dimensional arrays are often printed in a row by row format, using nested for statements.

```
for (int i=0; i<n; i++)
{
   //every row
   for (int j=0; j<m; j++ )//every col
        cout << array[i][j] << ‘ ‘;

   cout << endl; //add end-of-line each row
}
```

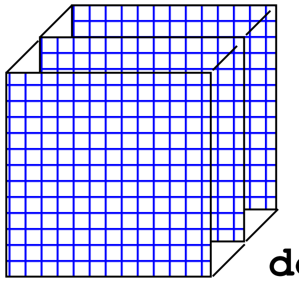
Higher-Dimensional Arrays

- An array can be declared with multiple dimensions.

2 Dimensional

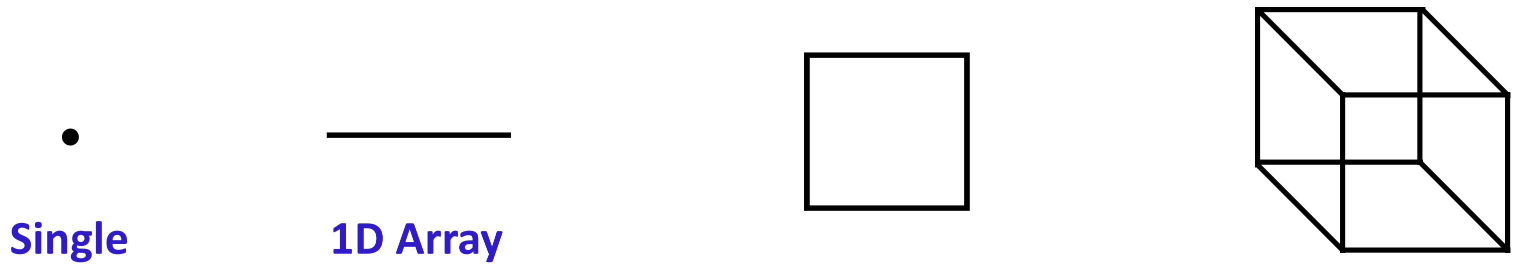


3 Dimensional



```
double Coord[100][100][100];
```

- **Multiple dimensions get difficult to visualize graphically.**



Larger-Dimension Arrays

- **Arrays** with **more than two dimensions** **allowed** in C**++** but **not commonly used**

Example:

```
int ThreeD[4][10][6];
```

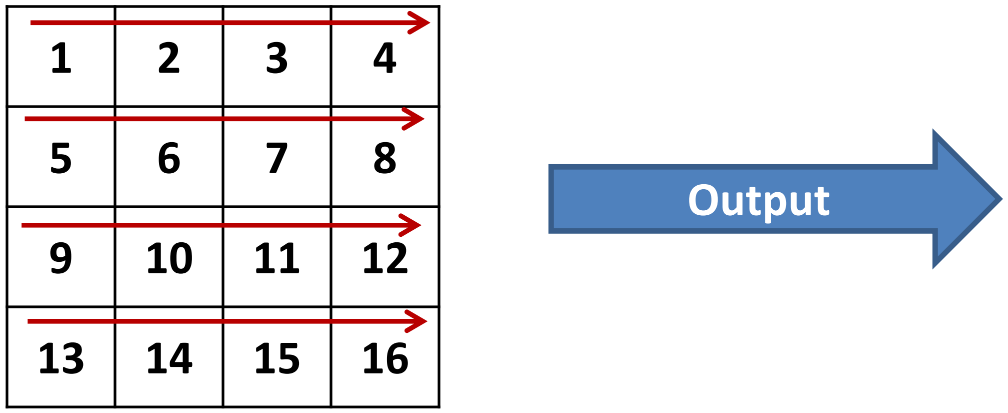
» First element: ThreeD[0][0][0]

» Last element: ThreeD[3][9][5]

(Nested Loops) – Example Program-1

- Write a program to that creates a matrix of size 5 by 5 (5 Columns, and 5 Rows). The program should ask the user to enter values in each matrix element. Then the program should display the matrix Row-wise.

Example:



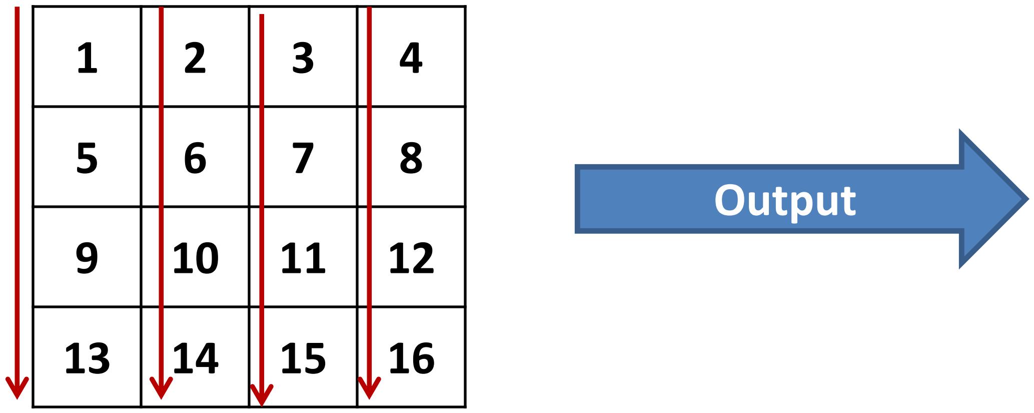
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
 1,   2,   3,   4,

 5,   6,   7,   8,

 9,  10,  11,  12,

13,  14,  15,  16,
```

(Nested Loops) – Example Program-2

- Write a program to that creates a matrix of size 5 by 5 (5 Columns, and 5 Rows). The program should ask the user to enter values in each matrix element. Then the program should display the matrix Coulmn-wise.

Example:



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
1,    5,    9,    13,

2,    6,    10,   14,

3,    7,    11,   15,

4,    8,    12,   16,
```

(Nested Loops) – Example Program-3

- Write a program to that creates a matrix of size 10 by 10 (10 Columns, and 10 Rows). The program should ask the user to enter values in each matrix element. Then the program should display the left-diagonal elements of the matrix.

Example (5 by 5 matrix):

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Output

1, 6, 11, 16,

Example-4: Zero Matrix

- Write a program that creates a matrix of 3 by 3 (3 rows, and 3 coulmns). Get input values from the user for the complete matrix. Then, the program should determine whether the matrix is a “Zero” matrix (all elements are zero) or not.

Example-5: Coulmn sum

- Write a program that creates a matrix of 4x4 (4 rows, and 4 coulmns). Get input values from the user for the complete matrix. The program should calculate and print the sums of each individual coulmn.

Example-6: Matrix Vector

- Write a program that creates a matrix of 7 by 7 (7 rows and 7 coulmns). Create a 1D array having 6 elements. The program should multiply the matrix with the given vector and print the resultant matrix in proper format.

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} AP + BQ + CR \\ DP + EQ + FR \\ GP + HQ + IR \end{bmatrix}$$

Example-6: Matrix Vector-code

```
int A[4][4];
int B[4]; int C[4]; int r;
// Get input in the “A” matrix and “B” array
for(int i=0; i<4; i++)
{
      r = 0;
      for(int j=0;j<4; j++)
            r = r + A[i][j] * B[j];

      C[i] = r;
}
```

Passing Arrays to Functions

Passing Arguments to Functions

- **Three ways** to pass arguments to function:
  1. **Pass-by-value**
  2. **Pass-by-reference** with **reference arguments**
  3. **Pass-by-reference** with **pointer arguments**

1. Pass by value – Example

```
void AddGraceMarks(int marks) {
     cout<<“\nActual marks:"<<marks<<endl;
     marks = marks + 10;
     cout<<“\nMarks Updated to:"<<marks<<endl;
}


int main() {
   int marks = 75;
   AddGraceMarks(marks);
   cout<<“You marks in PF are: ”<<marks<<endl;
   return 0;
}
```

Using Reference Variables with Functions

- To create a **second name** (**Alias**) for a **variable** in a program
- A **variable** that acts as an **alias** for another **variable** is called a **reference variable**, or simply a **reference**
- **Arguments** passed to **function using reference arguments**:
  - ***Modify original values of arguments***

1. Pass by Reference– Example

```
void AddGraceMarks(int &marks) {
     cout<<“\nActual marks:"<<marks<<endl;
     marks = marks + 10;
     cout<<“\nMarks Updated to:"<<marks<<endl;
}


int main() {
   int marks = 75;
   AddGraceMarks(marks);
   cout<<“You marks in PF are: ”<<marks<<endl;
   return 0;
}
```

Passing an Array to a Function

- We **need to tell the compiler** what the **type of the array,** and give it a **variable name** (a **reference**)

```
float a[ ]
```

- We **don't want to specify the size** so function can work with **different sized arrays**
- **Size** may be provided as **second parameter**
- **Arrays** are **automatically passed by reference**.
- **Do not use &** symbol

Passing an Array to a Function

An int array

The size of the array

```
int Display(int data[], int N) {

  cout<<“Array contains”<<endl;
  for (int k=0; k<N; k++)
      cout<<data[k]<<“ “;
  cout<<endl;
}


int main( )
{
   int a[4] = {11, 33, 55, 77};
   Display(a, 4);
   return 0;
}
```

The array argument,
no [ ] or & symbol

Passing 2D Arrays to Functions

- **One important difference only:**
- Include **empty brackets** for the **leftmost index**,
- Use **specific dimensions** for all **other indices** (along with the type)

```
int Max(int[][10], int);
```

- **Function definition:**

```
int Max(int Arr[][10], int sz)
      { . . .  }
```

- **Function call:**

```
Display(TDArr, Size);
```

```
int FindSum(int Arr[][4],int rows, int cols) {

    int sum=0;
    cout << “\n Calculating sum…" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
                 sum+= Arr[i][j];
    return sum;
}



int main() {

    // initialize 2d array
    int TDArr[3][4] = {{3,4,1,3},{9,5,2,1},{7,0,2,1}};

    // call the function, pass a 2d array as an argument
    cout<<“\n Sum is:”<<FindSum(TDArr,3,4);

    return 0;
}
```

Any Questions!