4 Array- Multi Dimensional

Nur Nabilah Abu Mangshor

Objectives

- Learn about parallel arrays
- Discover how to manipulate data in two-dimensional array
- Learn about multidimensional array

Parallel Arrays

- Two (or more) arrays are called parallel if their corresponding components hold related information.
- For example:

```
    int studentId [50];
    char courseGrade [50];
```

Two-Dimensional Arrays

- Two-dimensional Array: A collection of a fixed number of components arranged in two dimensions.
- All components are of the same type.
- The syntax declaring a two-dimensional array is:

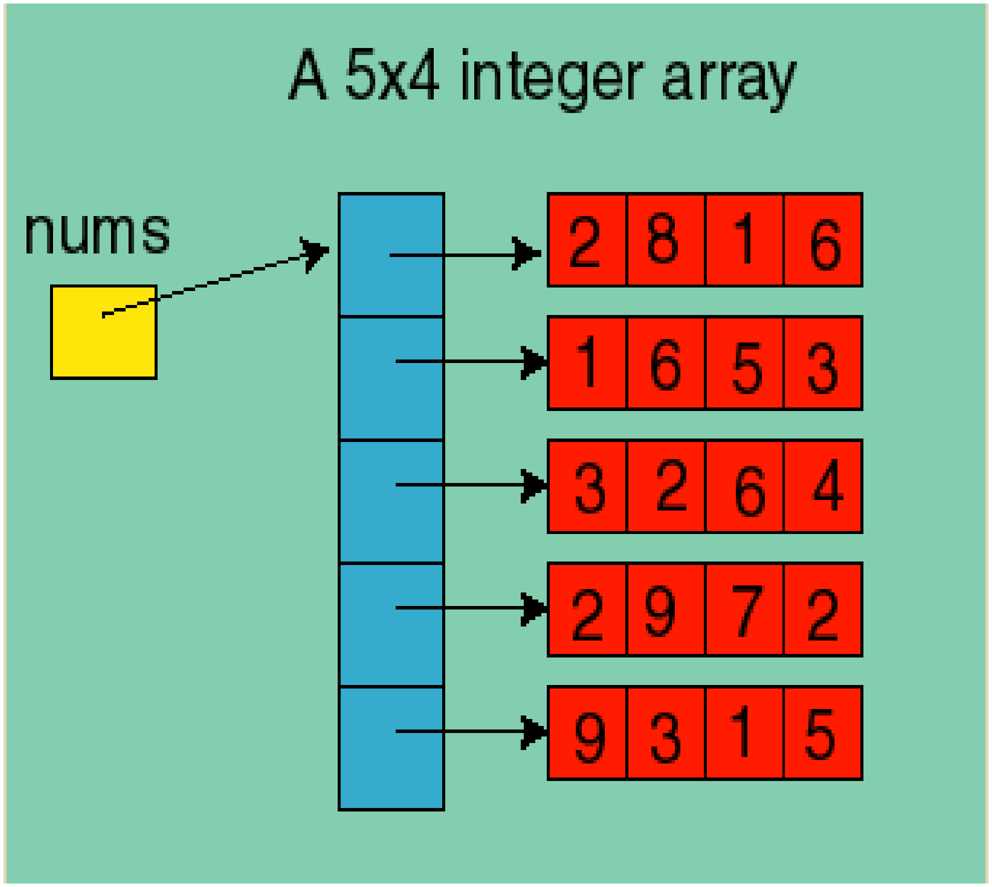
```
dataType arrayName[intexp1][intexp2];
```

where `intexp1` and `intexp2` are expressions yielding positive integer values.

Two-Dimensional Arrays (cont)

- The two expression `intexp1` and `intexp2` specify the <u>number of rows</u> and the <u>number of columns</u>, respectively in the array.
- Two-dimensional arrays are sometimes called matrices or tables.

Two-Dimensional Arrays (cont)



Declaration :

`int nums [5][4];`

Two-dimensional array `nums`

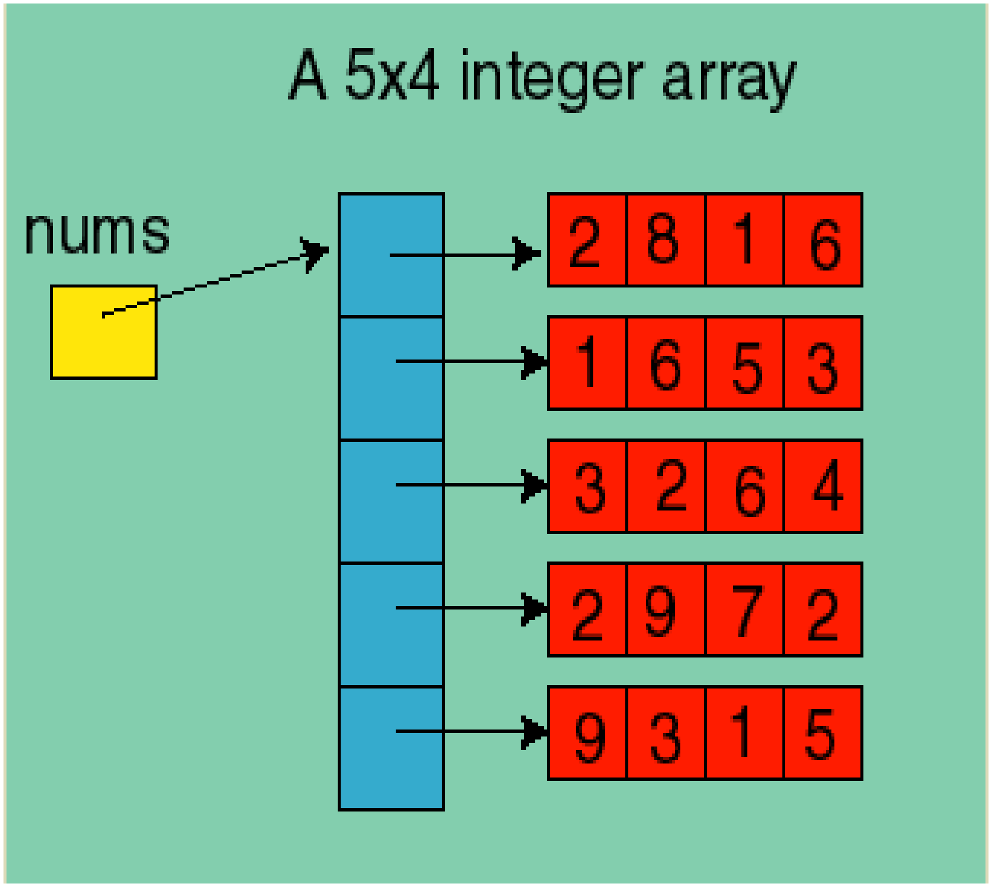
Accessing Array Components

- The syntax to access a component of a two-dimensional array is:

  `arrayName[indexexp1][indexexp2]`

  where `indexexp1` and `indexexp2` are expressions yielding nonnegative integer values.
- `indexexp1` specifies the <u>row position</u> and `indexexp2` specifies the <u>column position</u>.

Accessing Array Components



```
nums[2][1]  =  2
nums[3][2]  =  7
nums[4][3]  =  5
```

Initialization

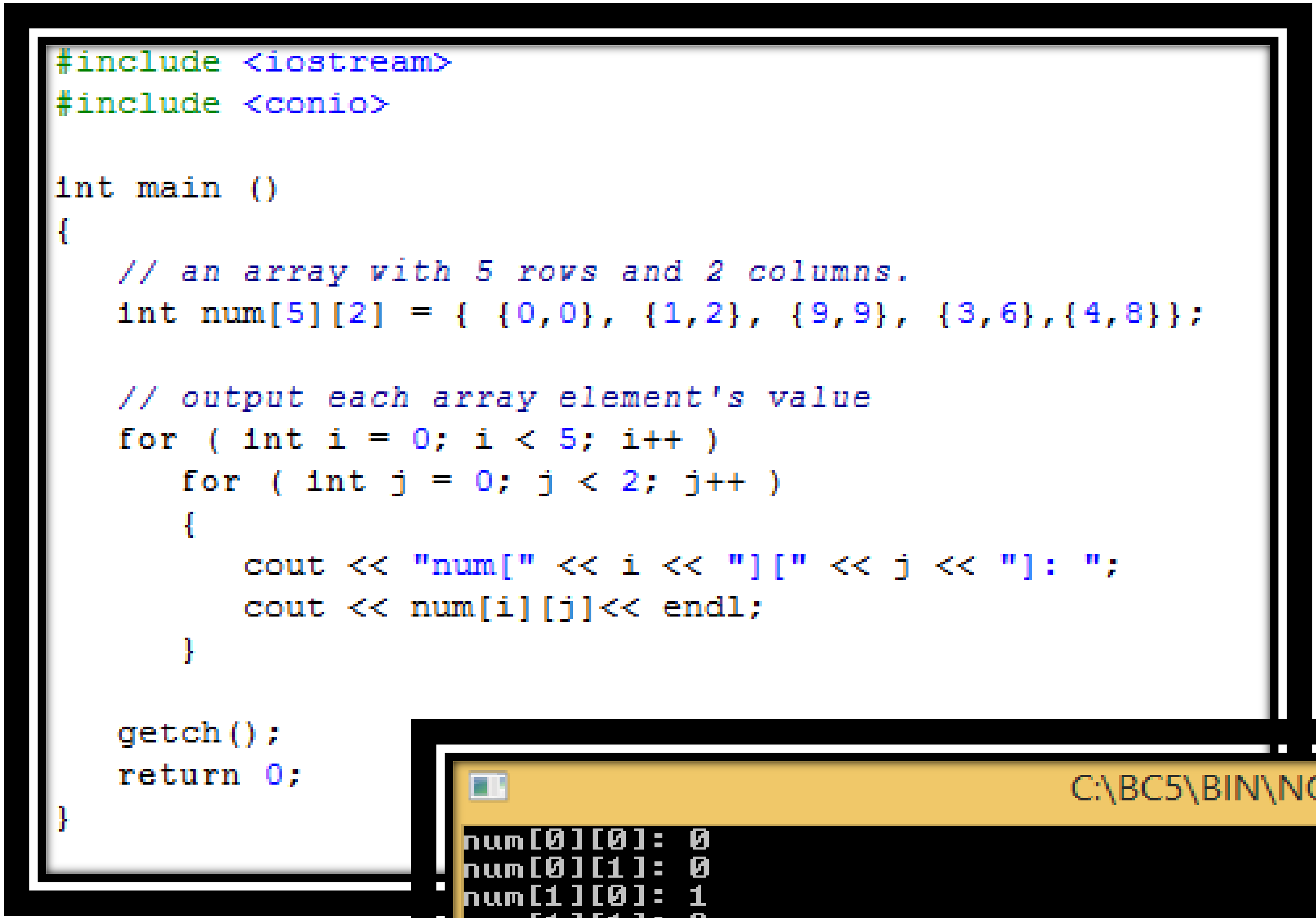
- Like one-dimensional arrays, two-dimensional arrays can be initialized when they are declared.
- To initialize a two-dimensional array when it is declared:
  1. Elements of each row are enclosed within braces and separated by commas.
  2. All rows are enclosed within braces.
  3. For number arrays, if all components of a row are not specified, the unspecified components are initialized to zero.

Initialization (cont)

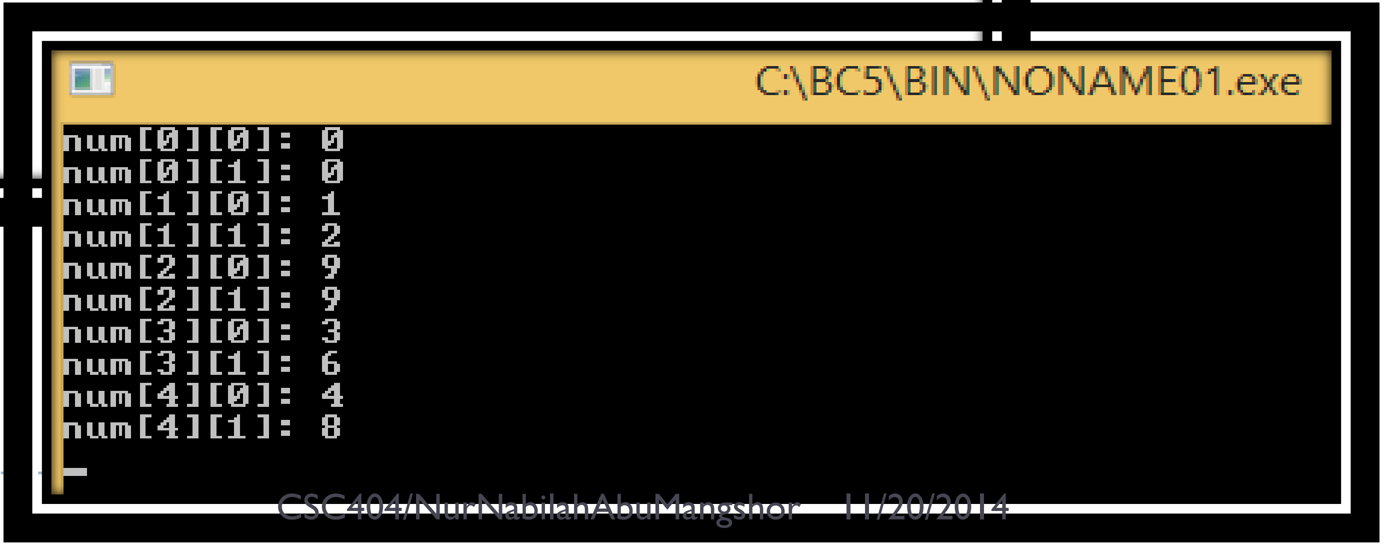
- Two-dimensional array may be initialized by specifying bracketed values for each row. Example:

```
int num [3][4] = {
    {0, 1, 2, 3},    //Row indexed by 0
    {4, 5, 6, 7} ,   //Row indexed by 1
    {8, 9, 10, 11}   //Row indexed by 2
};
```

- The nested braces which indicate the intended row, are optional.



```
#include <iostream>
#include <conio>

int main ()
{
   // an array with 5 rows and 2 columns.
   int num[5][2] = { {0,0}, {1,2}, {9,9}, {3,6},{4,8}};

   // output each array element's value
   for ( int i = 0; i < 5; i++ )
      for ( int j = 0; j < 2; j++ )
      {
         cout << "num[" << i << "][" << j << "]: ";
         cout << num[i][j]<< endl;
      }

   getch();
   return 0;
}
```



C:\BC5\BIN\NONAME01.exe

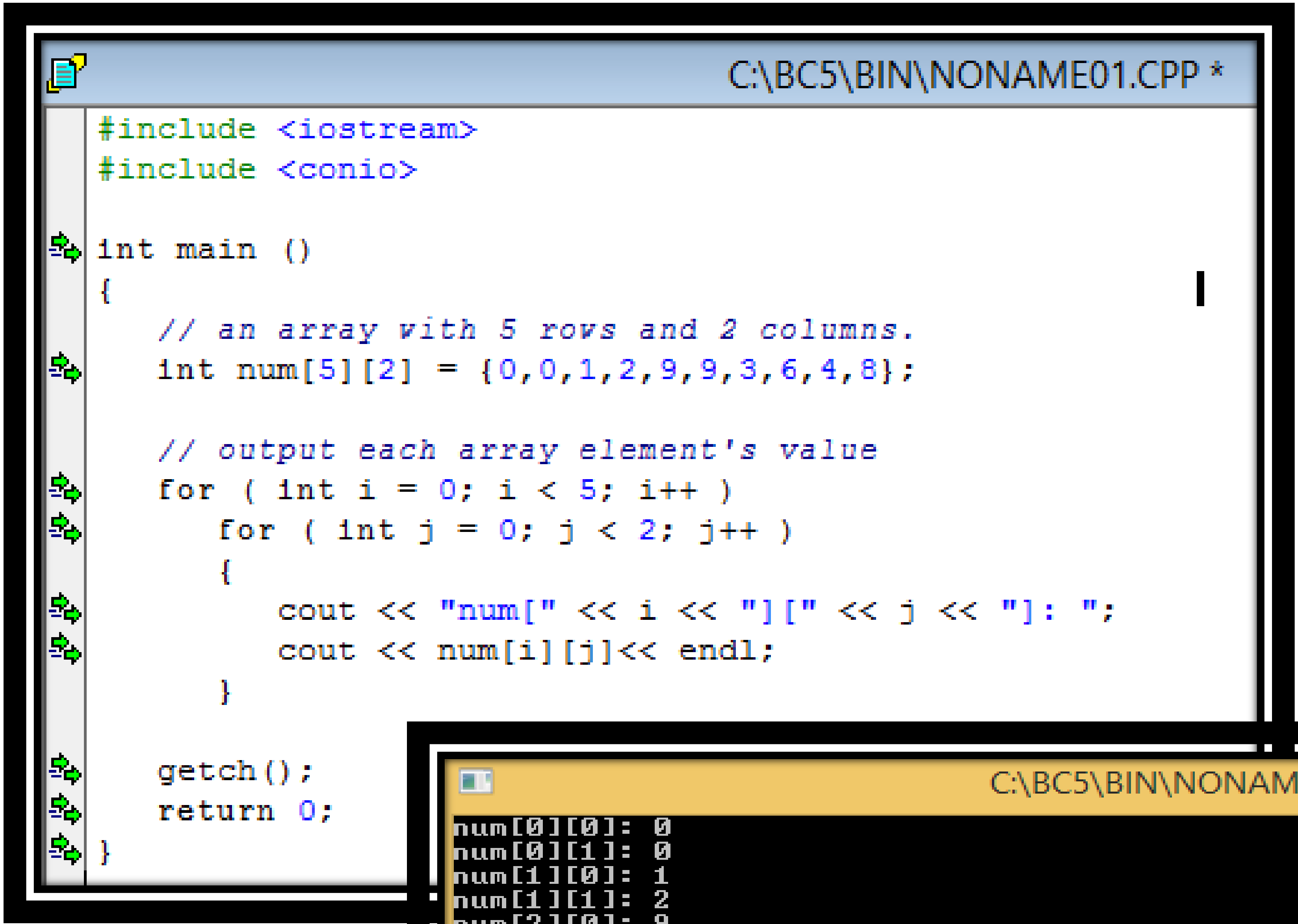
```
num[0][0]: 0
num[0][1]: 0
num[1][0]: 1
num[1][1]: 2
num[2][0]: 9
num[2][1]: 9
num[3][0]: 3
num[3][1]: 6
num[4][0]: 4
num[4][1]: 8
_
```

Initialization (cont)

- The following initialization is equivalent to the previous example:

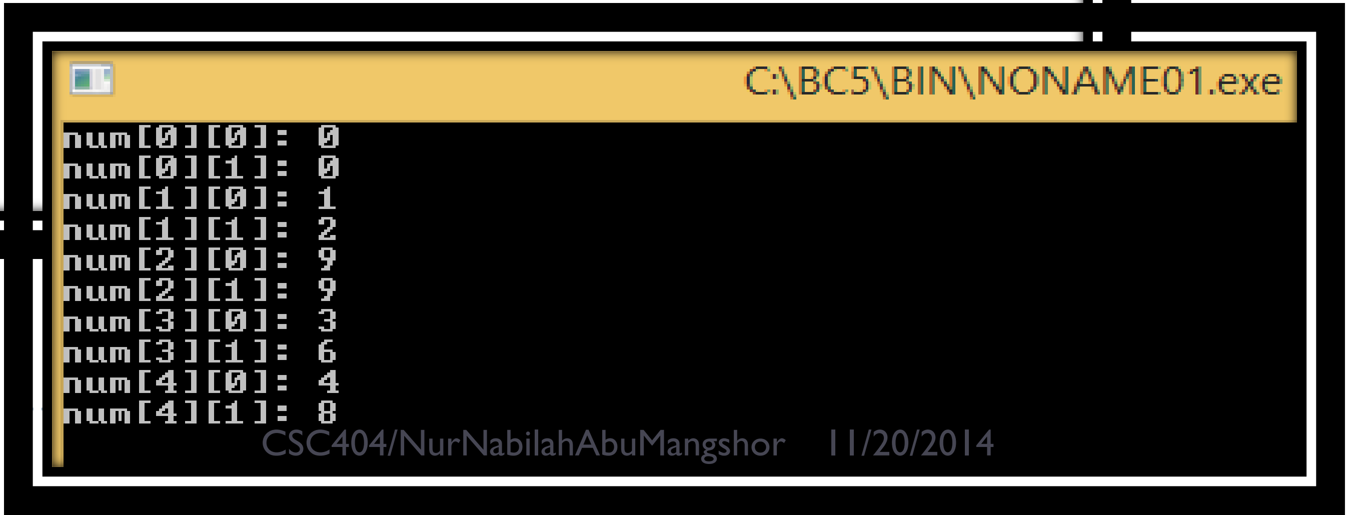
```
int nums [3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

- It will yields to same output.



C:\BC5\BIN\NONAME01.CPP *

```
#include <iostream>
#include <conio>

int main ()
{
   // an array with 5 rows and 2 columns.
   int num[5][2] = {0,0,1,2,9,9,3,6,4,8};

   // output each array element's value
   for ( int i = 0; i < 5; i++ )
      for ( int j = 0; j < 2; j++ )
      {
         cout << "num[" << i << "][" << j << "]: ";
         cout << num[i][j]<< endl;
      }

   getch();
   return 0;
}
```



C:\BC5\BIN\NONAME01.exe

```
num[0][0]: 0
num[0][1]: 0
num[1][0]: 1
num[1][1]: 2
num[2][0]: 9
num[2][1]: 9
num[3][0]: 3
num[3][1]: 6
num[4][0]: 4
num[4][1]: 8
```

Referring to Array Elements

- A pair of indices is required in for accessing the elements of a two-dimensional array which are row position and column position.
- Examples:

```
cout<<nums[3][0];   //print an array element
nums[3][0] = 12     //assigning a value
cin>>nums[3][0];    //input element
```

Processing Two-Dimensional Arrays

- A two-dimensional array can be processed in three different ways:
  I. Process the entire array.
  II. Process a particular row of the array, called row processing.
  III. Process a particular column of the array, called column processing.

Processing Two-Dimensional Arrays (cont)

- Each row and each column of a two-dimensional array is a one-dimensional array.
- When processing a particular row or column of a two-dimensional array:
  - We use algorithms similar to processing one-dimensional arrays.

Initialization

```
for (row=0; row < NUMBER_OF_ROWS; row++)
     for (col=0; col<NUMBER_OF_COLUMNS; col++)
          matrix[row][col] = 0;
```

Print

```
for (row=0; row < NUMBER_OF_ROWS; row++)
{
    for (col=0; col<NUMBER_OF_COLUMNS; col++)
          cout<<setw(5) << matrix[row][col]
          << " ";
          cout<<endl;
}
```

Input

```
for (row=0; row < NUMBER_OF_ROWS; row++)
     for (col=0; col<NUMBER_OF_COLUMNS; col++)
          cin >> matrix[row][col];
```

Sum by Row

```
//Sum of each individual row
for (row=0; row < NUMBER_OF_ROWS; row++)
{
    sum =  0;
    for (col=0; col<NUMBER_OF_COLUMNS; col++)
      sum = sum + matrix[row][col];

    cout<< "Sum of row" << row+1 <<" = "<<
sum << endl;
}
```

Sum by Column

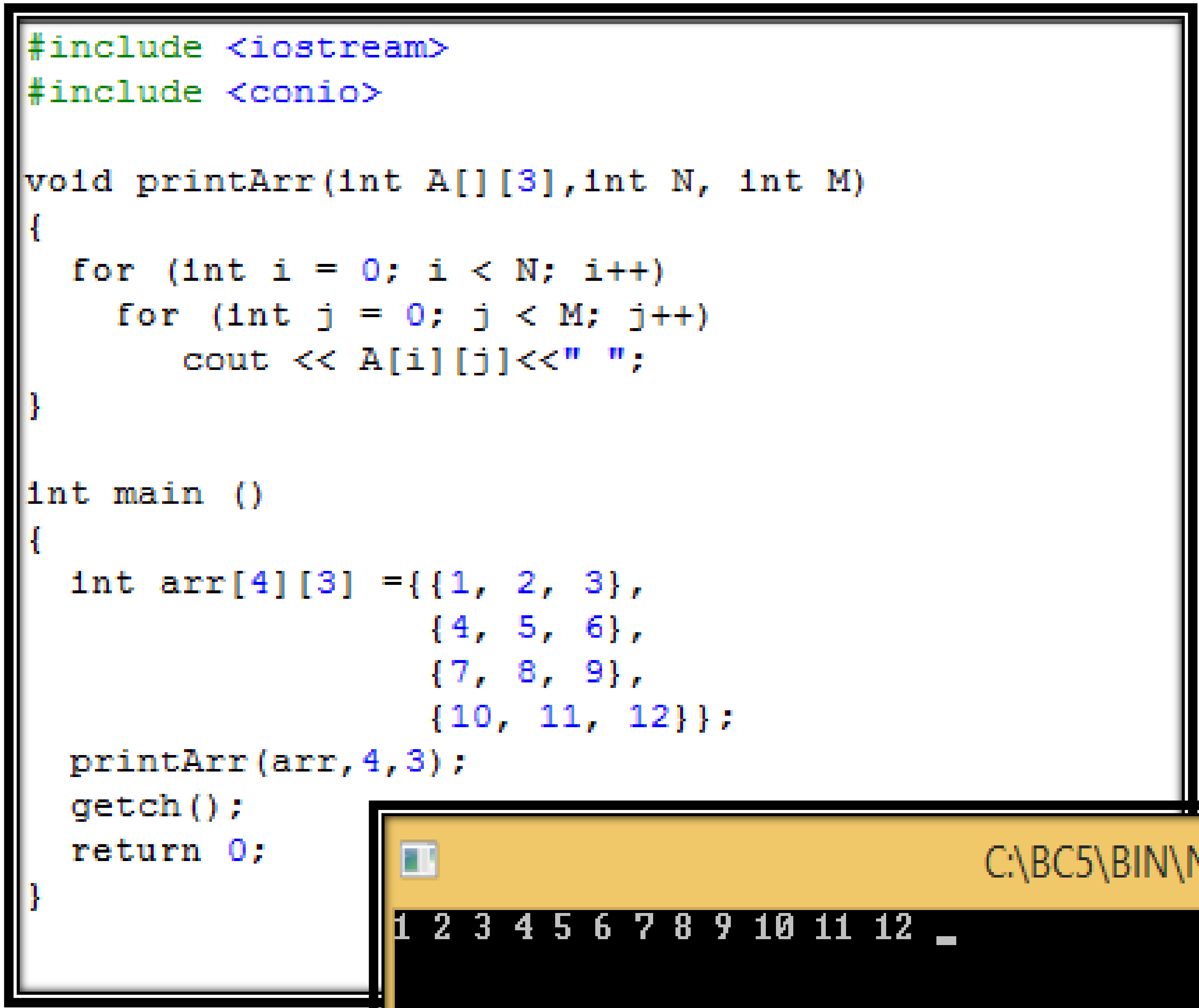
```
//Sum of each individual column
for (col=0; col < NUMBER_OF_COLUMNS; col++)
{
    sum =  0;
    for (row=0; row<NUMBER_OF_ROWS; row++)
      sum = sum + matrix[row][col];

    cout<< “Sum of column” << col+1 <<“ = “<<
sum << endl;
}
```

Passing Two-Dimensional Arrays as Parameters to Functions

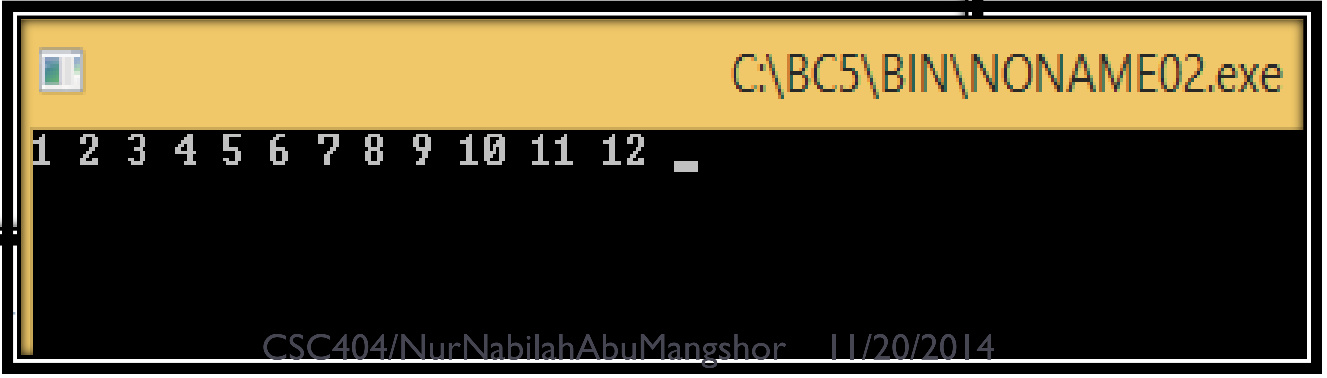
- Two-dimensional arrays can be passed as parameters to a function.
- By default, arrays are passed by reference.

Two-Dimensional Array in Functions (EXAMPLE)

```
const int numRow = 6;
const int numCol = 5;

void printMatrix(int matrix[][numCol], int
  noOfRows)
{
  int row, col;
  for(row=0; row<noOfRows; row++)
  {
       for(col=0; col<numCol; col++)
       cout<<setw(5)<<matrix[row][col]<<" ";
       cout<<endl;
  }
}
```



```
#include <iostream>
#include <conio>

void printArr(int A[][3],int N, int M)
{
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
       cout << A[i][j]<<" ";
}

int main ()
{
  int arr[4][3] ={{1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9},
                  {10, 11, 12}};
  printArr(arr,4,3);
  getch();
  return 0;
}
```



C:\BC5\BIN\NONAME02.exe

```
1 2 3 4 5 6 7 8 9 10 11 12 _
```

Multidimensional Arrays

- **Array** : A collection of a fixed number of elements (called components) of the same type.
- **1D-Array** : An array in which elements are arranged in list form.
- **2D-Array** : An array in which elements are arranged in a table form.
- **Multidimensional Array** : A collection of a fixed number of elements in *n* dimensions (*n*>=*1*) arrangement.

Multidimensional Arrays (cont)

- Multidimensional array is also called an n-dimensional array.
- General syntax of declaring an n-dimensional array is:

```
dataType arrayName[intExp1][intExp2]…[intExpn];
```

- Where `intExp1`, `intExp2`,… are constant expressions yielding positive integer values.

Multidimensional Arrays (cont)

- The syntax for accessing a component of an `n`-dimensional array is:

  `arrayName[intExp1][intExp2]…[intExpn];`

  where `intExp1`, `intExp2`,... are expressions yielding nonnegetive integer values.
- `indexExpi` gives the position of the array component in the `i`th dimension.

Multidimensional Arrays (cont)

- When declaring a multi-dimensional array as a formal parameter in a function.
  - Can omit size of the first dimension but not other dimensions
- As parameters, multi-dimensional arrays are passed by reference only.
- A function cannot return a value of the type array.
- There is no check if the array indices are within bounds.

Review Questions

- What is array 2D?
- What is the difference between array 1D and array 2D?