For updated version, please click on
http://ocw.ump.edu.my

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

BTE2313

Chapter 8: Arrays

by
Sulastri Abdul Manap
Faculty of Engineering Technology
sulastri@ump.edu.my

Objectives

- In this chapter, you will learn about:
  1. Declaring and Initializing arrays
  2. Accessing/Selecting element of arrays
  3. Working with multi-dimensional arrays
  4. Arrays as parameters

Introduction

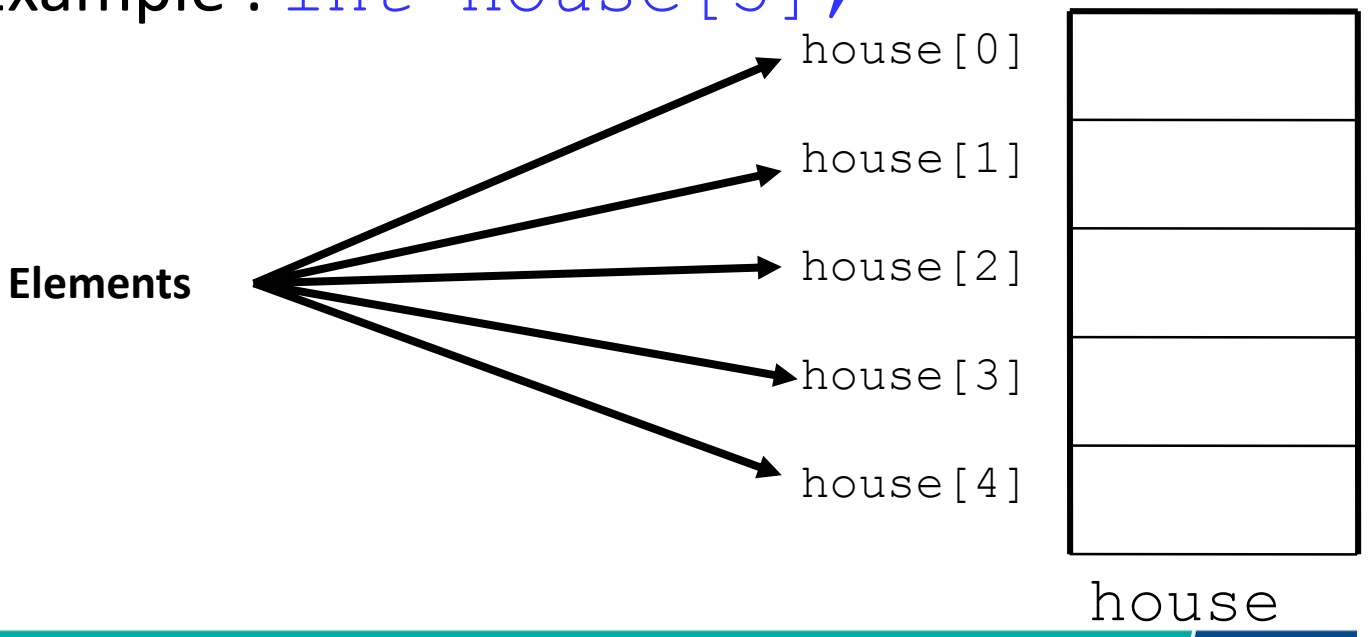
- Array is a list of more than one variable having the same name and data type.
- Array is a fixed-size collection of consecutive memory locations. Each memory location accessed by a relative address called index or subscript.
- Every index starts with 0.
- Two types of Array:

→One-Dimensional Array

→Multi-Dimensional Array

Introduction (cont.)

- Array can be described like a block of building of the same type, containing a number of houses and each house has different number/address as a reference.
- Example : `int house[5];`



Array Declarations

- Format:

  Datatype Array_name [*Array_Size*];

- *Datatype* is the data type (int, float, double, etc.)
- Array_name is an identifier
- *Array_Size* states the size/length of the array in terms of number of elements that must be enclosed in square brackets [ ]
- For example:

```
int numbers[1000];
double rays[200];
string students[25];
char name[25];
```

Array Declarations: Example #1

- Declaring an array, with the name *myNum* , type *double* and has 10 elements

```
double myNum[10];
```

- Array *myNum* can store up 10 floating numbers with the first number occupying location *myNum* [0]; the second number at location *myNum* [1] and last number at location *myNum* [9].

Array Declarations: Example #2

- For the PhoneBills program, declare an array of twelve floating point values:

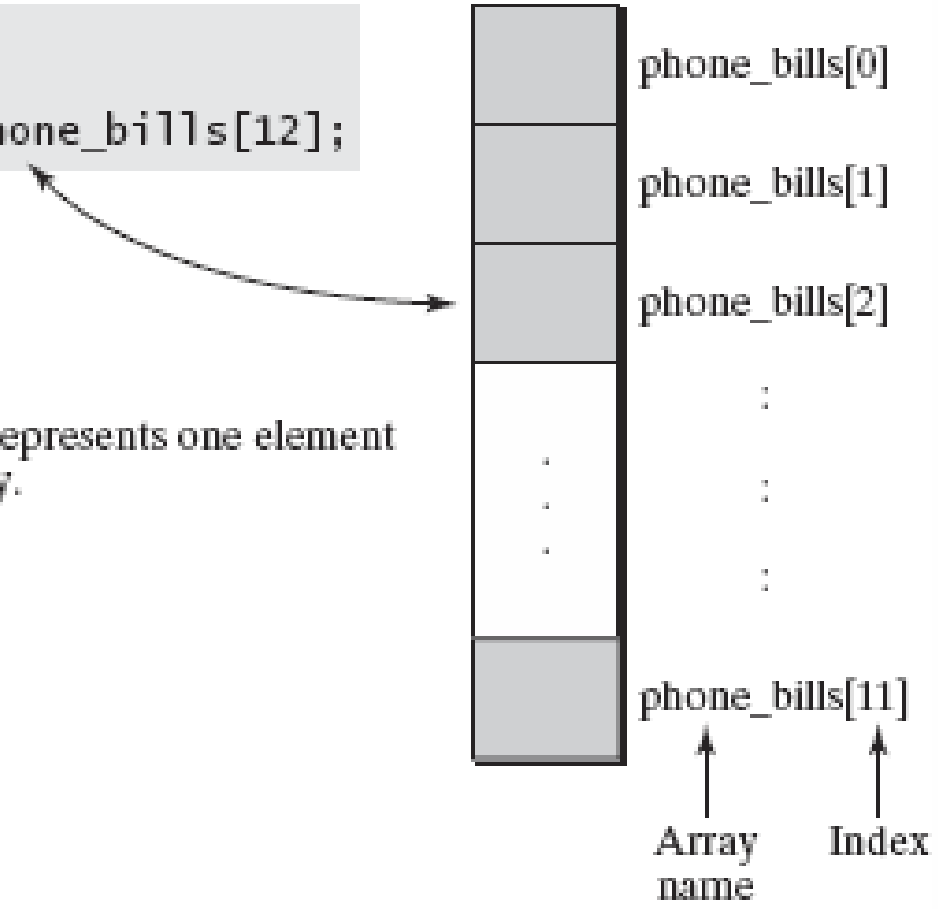
```
float phone_bills[12];
```

- When a programmer creates a list using one dimension (one size) value, it is referred to as a ***single-dimensional array*** and can be thought of as a single list or as a row of values.

- These values are stored contiguously in memory.

Example #2 (cont.)

```
int main()
{
   float phone_bills[12];
```

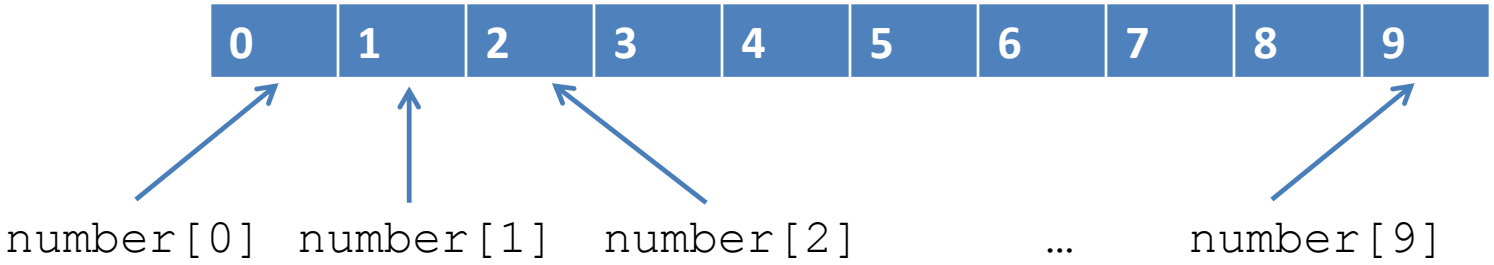


The **`phone_bills`** array. It is useful to visualize an array as a group of boxes.

Array Initializations

- Values must be enclosed with braces { }.
- If there is more than one value, separated by commas.
- MUST provide the number of values not more than the size of array.
- Example 1:

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



Array Initializations (cont.)

Example 2:

```
char vowel[10] = {'a','e','i','o','u'};
```

vowel

Example 3:

```
float mynum[5] = {1.2,1.3,1.5,1.5,1.5};
```

mynum

Example 4:

```
string names[3] = {"Ali","Chong","Ravi"};
```

Array Initializations (cont.)

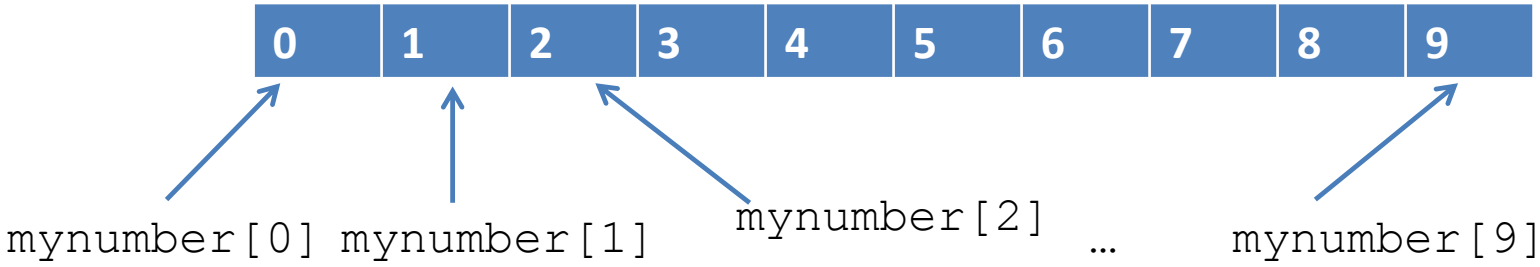
Example 4:

```
int mynumbers[10];
// Declare the array
mynumbers[0] = 3; // 3 is stored as 1st element
mynumbers[9] = 2; // value 2 is stored as last element
mynumbers[-1] = 9; // invalid index (cannot have –ve index)
mynumbers[10] = 2; // invalid index (last valid index is 9)
mynumbers[2.3] = 4; // error! (1.3 is not an integer)
```

`mynumbers[0]` `mynumbers[9]`

Accessing Arrays

- All values in an array can be accessed like the value of a normal variable of the same type.
- The syntax is: *Array_name*[index]

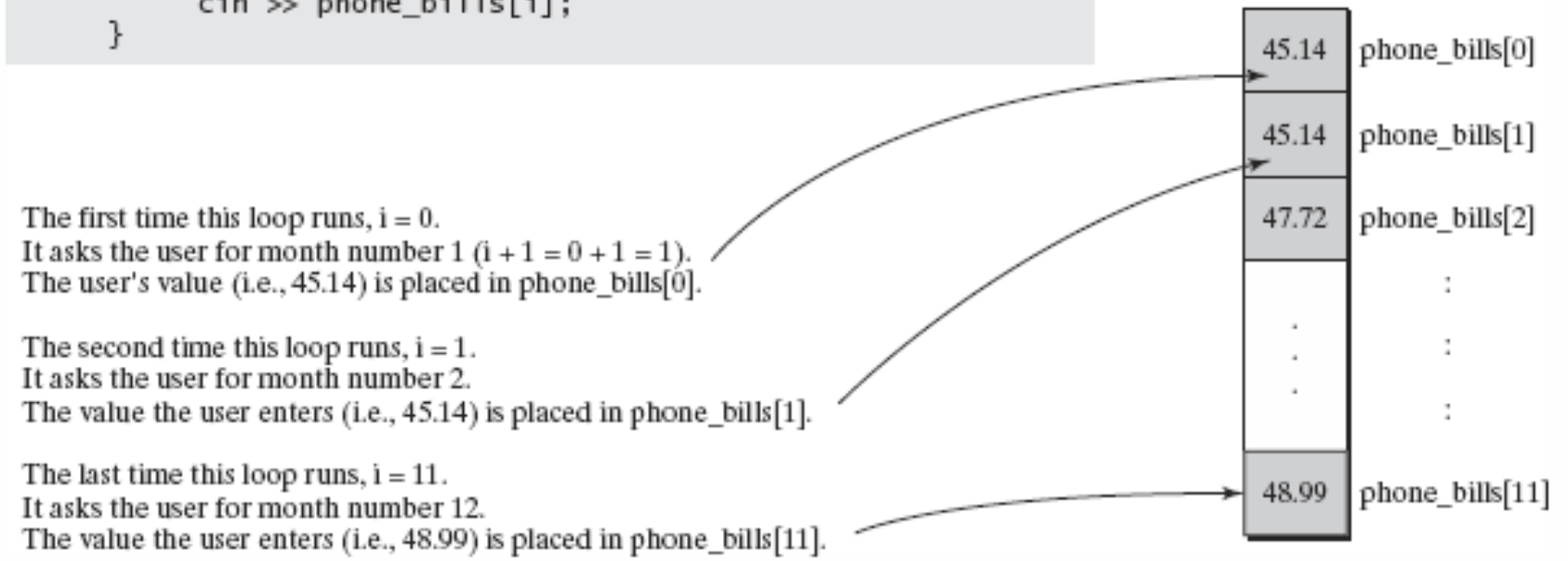


- Brackets [ ] are used to carry out two tasks: to indicate the length/size of an array in declarations; and to indicate the index number for the array elements when they are accessed.

`for` Loops and Arrays

- The **`for`** loop provides an efficient method for going through an array.
- The index of the loop is used not only as a counter for the loop, it can also be used as the index value for the array.

```
// Obtain monthly billing information
     for(i = 0; i < 12; ++i)
     {
          cout << "\n Enter bill for month # " << i + 1<< "$";
          cin >> phone_bills[i];
     }
```



Accessing Arrays: Example #1

```
#include <iostream>
using namespace std;
int mynum[ ] = {15, 3, 43, 32, 1781};
int count, sum=0;

int main ()
{
  for ( count=0 ; count<5 ; ++count )
  {
    cout <<  mynum[count] << endl;
    result += mynum[count];
  }
  cout << sum;
  return 0;
}
```

Array Out of Bounds

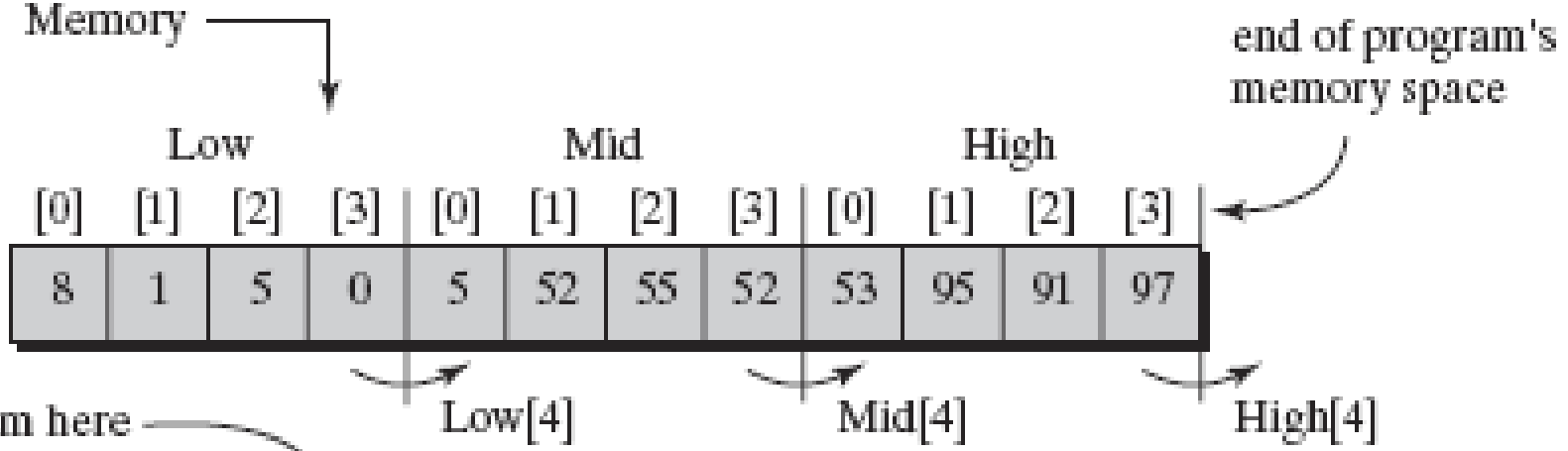
- C++ does not do any type of array boundary checking when a program uses arrays.
- The program does not warn you or stop the program if a statement causes the program to access an array element that is not legally declared.
- This ***out of bounds array*** feature of the C++ language can trash your data or crash your program when the program is executed.

Array Out of Bounds(cont.)

Declaration

```
int High[4], Mid[4], Low[4];
```

Memory



end of program's memory space

Problem here

```
for(i = 0; i <= 4; ++i)
{
  // runs to i = 0 to i = 4
  therefore Low[4] => Mid[0]
            Mid[4] => High[0]
            High[4] => who knows?
}
```

CRASH ! with Access Violation

Arrays as Parameters

- It is impossible to directly pass as argument, the whole chunk of memory as an array into a function.
- Instead of passing the whole array, an array can be passed as argument into a function by providing its address (by reference)
- Example:

```
void procedure (int arg[])
```
//function prototype

```
int myarray [40];
```
//array declaration

```
procedure (myarray);
```
//function call by reference

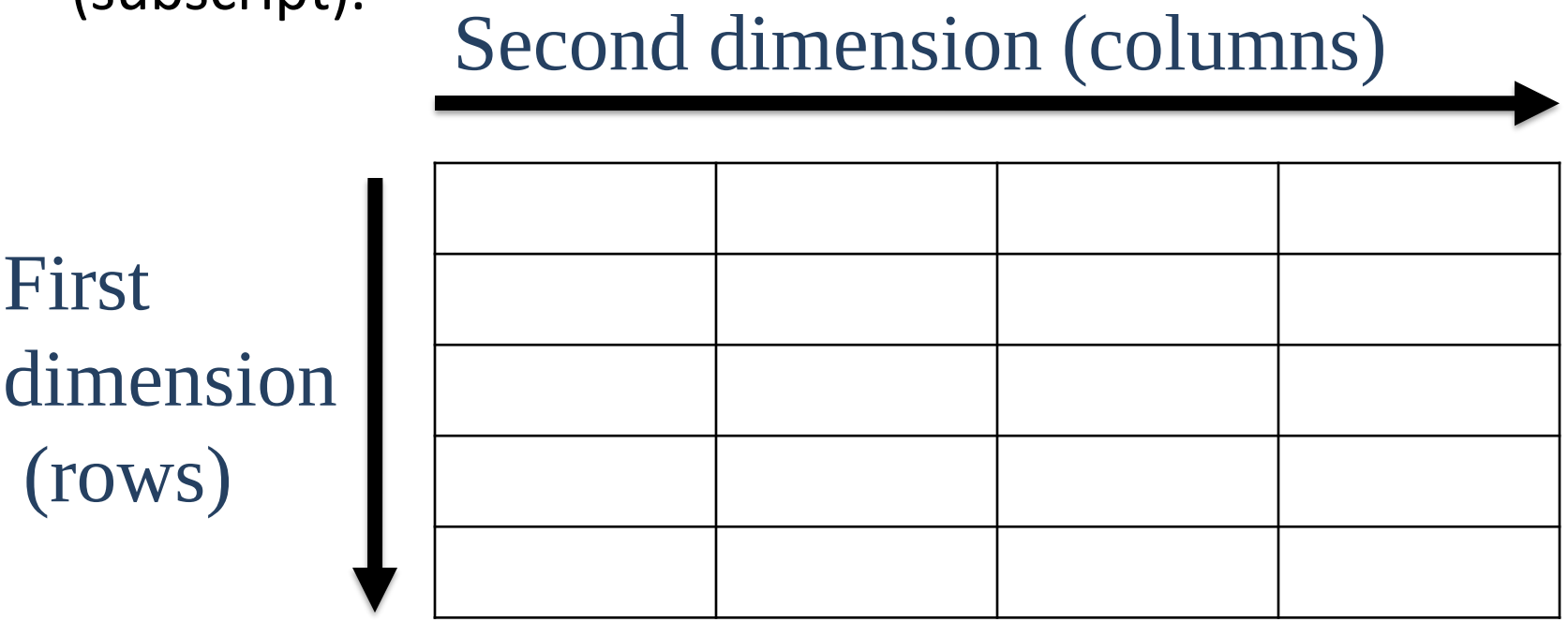
Arrays as Parameters: Example

```
#include <iostream>
using namespace std;
void print_array (int arg[], int len);
int main ()
{
  float array1[] = {2.5, 10.1, 15.3};
  float array2[] = {2.1, 11.4, 16.0, 8.2, 10.1};
 print_array (array1,3);
 print_array (array2,5);
}

void print_array (int arg[], int len){
  for (int i=0; i<len; i++)
    cout << arg[i] << " ";
  cout << "\n";
}
```

Multi-Dimensional Array

- Multidimensional arrays is an array that has more than one dimension, dimensions for rows and columns
- A two-dimensional array (2D) has two indexes (subscript).



2-Dimensional Array

- The declaration of 2D arrays is quite similar with one dimensional array.
- A 2D arrays have two subscripts which the first is for the number of rows and the second is for columns.
- The declaration of two-dimensional array takes the form :

```
datatype Array_name[row_size][column_size];
```

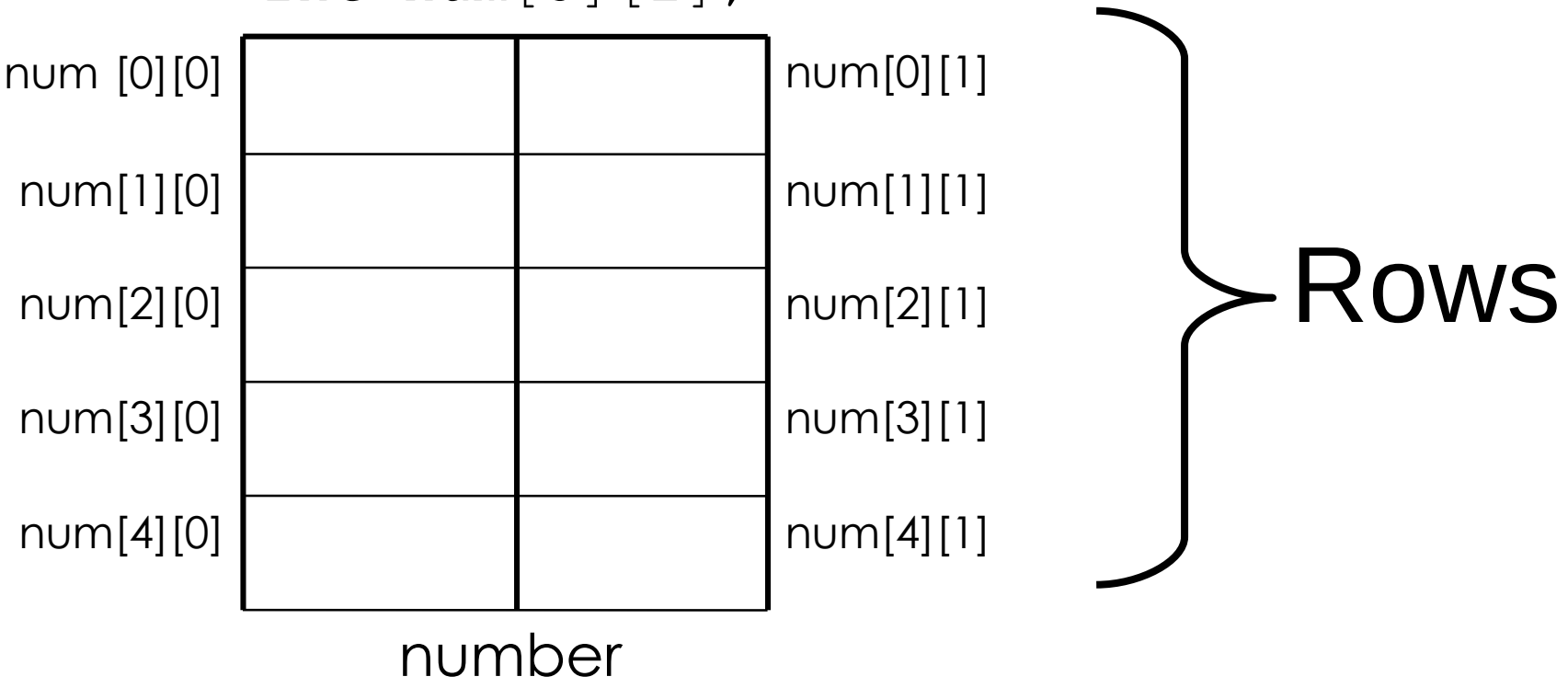
- Example

```
char name[3][10];
int list[rows][cols];
int x[3][4];
```

2-Dimensional Array (cont.)

- Example

```
int num[5][2];
```



2-Dimensional Array Initialization

- Example

```
int number[5][2] =
{ {99,3},{12,30},{40,15},{36,70},{10,2} };
```

or

```
int number[5][2] =
{99,3,12,30,40,15,36,70,10,2};
```

- When compiler encounters the declaration, it allocates memory locations for the elements in a linear fashion