CS1002 – Programming Fundamentals

Lecture # 25
Monday, November 28, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Rizwan Ul Haq

Array as a parameter to Functions

- Arrays are passed by reference only
- The symbol **&** is not used when declaring an array as a formal parameter
- The size of the array is usually omitted
- If provided, it is ignored by the compiler

E.g.

```
void foo(double firstList[], int secondList[])
{
    .
    .
    .
}
```

Arrays as Parameters to Functions

```
void initialize(int list[], int listSize)
{
   int count;
   for (count = 0; count < listSize; count++)
      list[count] = 0;
}
```

- The first parameter of the function initialize is an **int** array of any size
- When the function initialize is called, the size of the actual array is passed as the second parameter of the function initialize

Passing an Entire Array

- Use the array name, without any brackets, as the argument
- Can also pass the array size so the function knows how many elements to process

```
void printGrades(int[], int);       // prototype

void printGrades(int A[], int size) // header

printGrades(tests, 5);              // call
```

Constant Arrays as Formal Parameters

EXAMPLE 9-6

```
    //Function to initialize an int array to 0.
    //The array to be initialized and its size are passed
    //as parameters. The parameter listSize specifies the
    //number of elements to be initialized.
void initializeArray(int list[], int listSize)
{
    int index;

    for (index = 0; index < listSize; index++)
        list[index] = 0;
}
    //Function to print the elements of an int array.
    //The array to be printed and the number of elements
    //are passed as parameters. The parameter listSize
    //specifies the number of elements to be printed.
void printArray(const int list[], int listSize)
{
    int index;

    for (index = 0; index < listSize; index++)
        cout << list[index] << " ";
}
```

Base Address of an Array and Array in Computer Memory

- The base address of an array is the address, or memory location of the first array component
- If **list** is a one-dimensional array, its base address is the address of **list[0]**
- When we pass an array as a parameter, the base address of the actual array is passed to the formal parameter

Base Address of an Array and Array in Computer Memory (

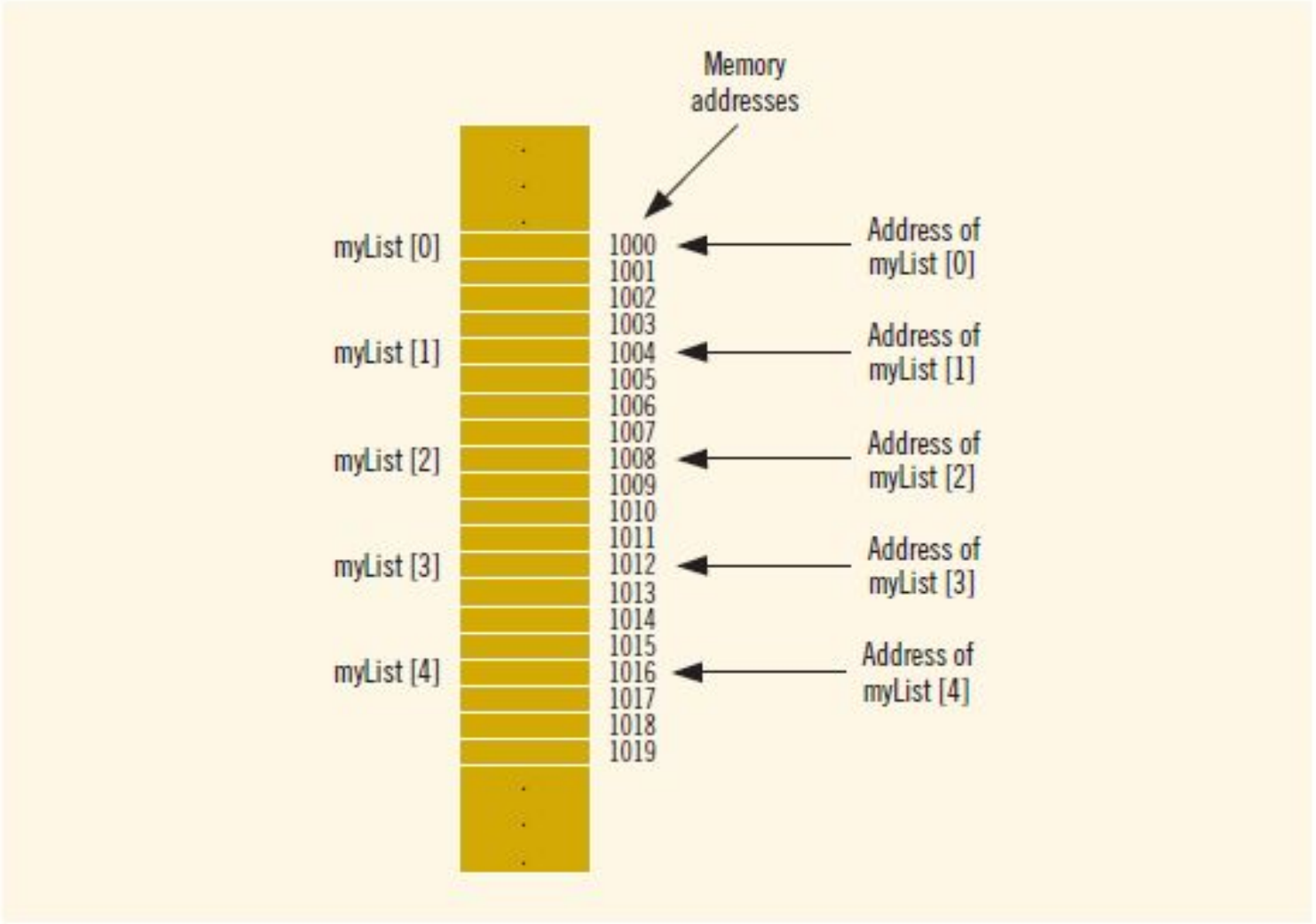

FIGURE 9-7 Array `myList` and the addresses of its components

Base Address of an Array and Array in Computer Memory (cont'd.)

- Consider the following statement:
  - cout << myList << endl; //Line 2
- This statement will not output the values of the components of **myList**
- In fact, the statement outputs the **value of myList**, which is the base address of the array
- This is why the statement will not generate a syntax error

Base Address of an Array and Array in Computer Memory (cont'd.)

- Suppose that you also have the following statement:

```
int yourList[5];
```

- Then, in the statement:

```
if (myList <= yourList)
```

- It does not determine whether the elements of **myList** are less than or equal to the corresponding elements of **yourList**

Base Address of an Array and Array in Computer Memory (cont'd.)

- when you declare an array, the only things about the array that the computer remembers are:
  - Name of the array
  - Its base address
  - The data type of each component
  - and (possibly) the number of components

Example

- Suppose you want to access the value of **myList[3]**
- Now, the base address of **myList** is 1000
  - Each component of **myList** is of type **int**, so it uses four bytes to store a value, and the index is 3
- To access the value of myList[3], the computer calculates the address

  1000 + 4 * 3 = 1000 + 12 = 1012
- That is, this is the starting address of myList[3]
  - So, starting at 1012, the computer accesses the next four bytes

Example

```
void arrayAsParameter(int list[], int size)
{
    ...
    list[2] = 28; //Line 4
    ...
}
```

- Suppose that you have the following call to this function:
- arrayAsParameter(myList, 5);
- list[2] = 28; This statement stores 28 into list[2]. To access list[2], the computer calculates the address as follows: 1000 + 4 * 2 = 1008

Functions Cannot Return a Value of the Type Array

- C++ does not allow functions to return a value of the type array

Searching an Array for a Specific Item

- Sequential search or linear search
  - Searching a list for a given item
  - Starting from the first array element
  - Compare **searchItem** with the elements in the array
  - Continue the search until either you find the item or no more data is left in the **list** to compare with **searchItem**

Searching an Array for a Specific Item (cont'd.)

```
int seqSearch(const int list[], int listLength, int searchItem)
{
    int loc;
    bool found = false;

    loc = 0;

    while (loc < listLength && !found)
        if (list[loc] == searchItem)
            found = true;
        else
                loc++;

    if (found)
        return loc;
    else
        return -1;

}
```

```
int seqSearch(const int list[], int listLength,
              int searchItem);                                  //Line 4

int main()                                                      //Line 5
{                                                               //Line 6
    int intList[ARRAY_SIZE];                                    //Line 7
    int number;                                                 //Line 8

    cout << "Line 9: Enter " << ARRAY_SIZE
         << " integers." << endl;                               //Line 9

    for (int index  = 0; index < ARRAY_SIZE; index++)           //Line 10
        cin >> intList[index];                                  //Line 11

    cout << endl;                                               //Line 12

    cout << "Line 13: Enter the number to be "
         << "searched: ";                                       //Line 13
    cin >> number;                                              //Line 14
    cout << endl;                                               //Line 15

    int pos = seqSearch(intList, ARRAY_SIZE, number);           //Line 16

    if (pos!= -1)                                               //Line 17
        cout <<"Line 18: " << number
             << " is found at position " << pos
             << endl;                                           //Line 18
    else                                                        //Line 19
        cout << "Line 20: " << number
             << " is not in the list." << endl;                 //Line 20

    return 0;                                                   //Line 21
}                                                               //Line 22

//Place the definition of the function seqSearch
```

Passing Two-Dimensional Arrays as Parameters to Functions

- Two-dimensional arrays can be passed as parameters to a function
  - Pass by reference
    - Base address (address of first component of the actual parameter) is passed to formal parameter
- Two-dimensional arrays are stored in row order
- When declaring a two-dimensional array as a formal parameter, can omit size of first dimension, but not the second

Example

Suppose we have following declaration:

```
const int NUMBER_OF_ROWS = 6;
const int NUMBER_OF_COLUMNS = 5;
```

Consider the following definition of function printMatrix :

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

Function outputs the sum of the elements of each row

```
void sumRows(int matrix[][NUMBER_OF_COLUMNS], int noOfRows)
{
    int row, col, sum = 0;
    for (row = 0; row < noOfRows; row++)
    {
        sum = 0;
        for (col = 0; col < NUMBER_OF_COLUMNS; col++)
            sum = sum + matrix[row][col] << " ";
        cout << "Sum of row " << row + 1
            << " = " << sum << endl;
    }
}
```

Function determines the largest element in each row:

```
void largestInRows(int matrix[][NUMBER_OF_COLUMNS],
int noOfRows)
{
    int row, col, sum = 0;
    //Largest element in each row
    for (row = 0; row < noOfRows; row++)
    {
        largest = matrix[row][0];
        for (col = 0; col < NUMBER_OF_COLUMNS; col++)
            if (matrix[row][col] > largest)
                largest = matrix[row][col];

        cout << “The Largest element of row ”
            << row + 1 << “ = ” << largest << endl;
    }
}
```

Multidimensional Arrays (cont'd.)

- When declaring a multidimensional array as a formal parameter in a function
  - Can omit size of first dimension but not other dimensions
- As parameters, multidimensional 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

Summary

- Functions (modules) are miniature programs
  - Divide a program into manageable tasks
- C++ provides the standard functions
- Two types of user-defined functions: value-returning functions and void functions
- Variables defined in a function heading are called formal parameters
- Expressions, variables, or constant values in a function call are called actual parameters

Summary (cont'd.)

- In a function call, the number of actual parameters and their types must match with the formal parameters in the order given
- To call a function, use its name together with the actual parameter list
- Function heading and the body of the function are called the definition of the function
- A value-returning function returns its value via the return statement

Summary (cont'd.)

- A prototype is the function heading without the body of the function; prototypes end with the semicolon
- Prototypes are placed before every function definition, including main
- User-defined functions execute only when they are called
- In a call statement, specify only the actual parameters, not their data types

Questions