Programming Fundamentals

Aamina Batool

```
int main()
{int test0, test1, test2, test3, test4;
double average;
cin >> test0 >> test1 >> test2 >> test3 >> test4;
average = (test0 + test1 + test2 + test3 + test4) / 5.0;
cout << "The average test score = " << average << endl;
if (test0 < average)
   cout << test0 << " is less than the average “<<endl;
if (test1 < average)
   cout << test1 << " is less than the average "<< endl;
if (test2 < average)
   cout << test2 << " is less than the average “ << endl;
if (test3 < average)
   cout << test3 << " is less than the average " << endl;
if (test4 < average)
   cout << test4 << " is less than the average " << endl;
return 0;}
```

Data Types

- A data type is called simple if variables of that type can store only one value at a time
- A structured data type is one in which each data item is a collection of other data items

Arrays

- Array - a collection of a fixed number of components wherein all of the components have the same data type
- One-dimensional array - an array in which the components are arranged in a list form
- The general form of declaring a one-dimensional array is:

```
dataType arrayName[intExp];
```

where `intExp` is any expression that evaluates to a positive integer

Declaring an array

- The statement

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

  declares an array `num` of `5` components of the type `int`
- The components are `num[0]`, `num[1]`, `num[2]`, `num[3]`, and `num[4]`

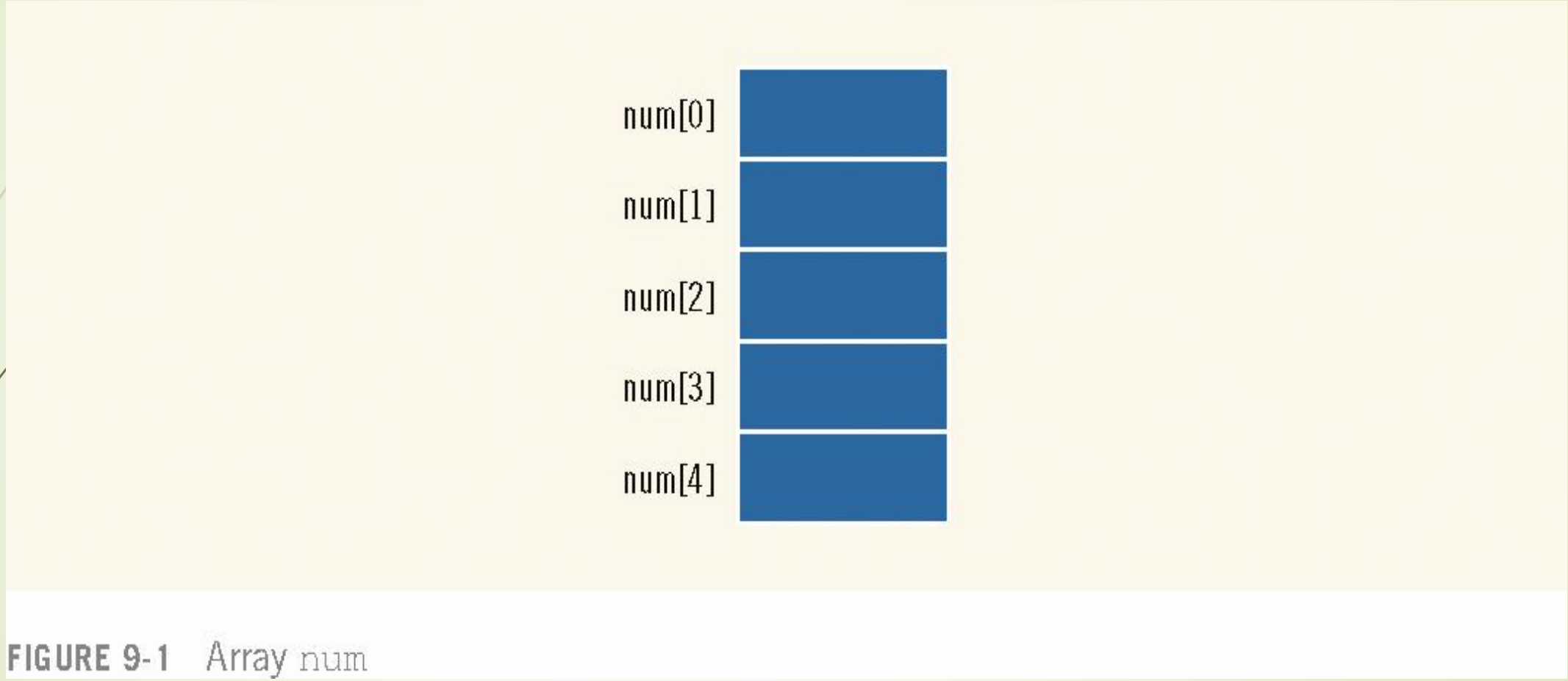


FIGURE 9-1 Array num

Accessing Array Components

- The general form (syntax) of accessing an array component is:

  ```
  arrayName[indexExp]
  ```

  where `indexExp`, called **index**, is any expression whose value is a nonnegative integer
- Index value specifies the position of the component in the array
- The `[]` operator is called the **array subscripting operator**
- The array index always starts at `0`

```
int list[10];
```

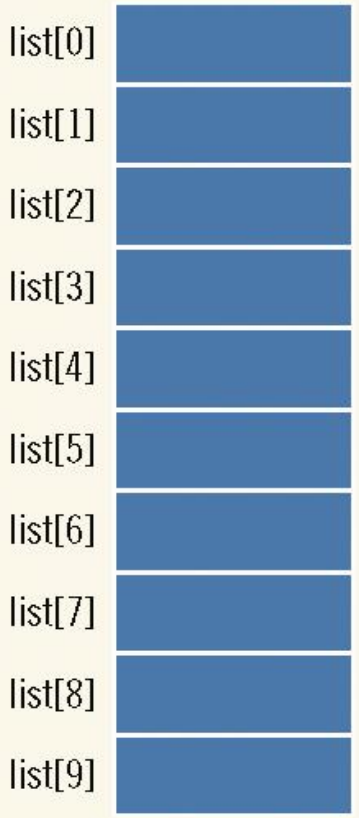


FIGURE 9-2 Array `list`

```
list[5] = 34;
```

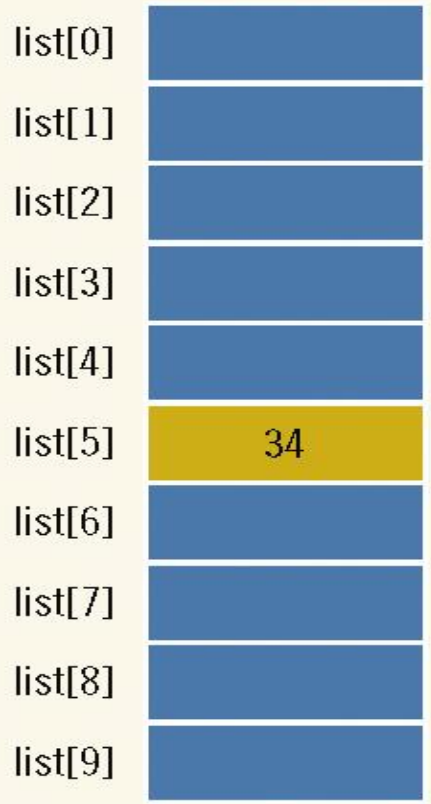


FIGURE 9-3 Array `list` after execution of the statement `list[5]= 34;`

```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```

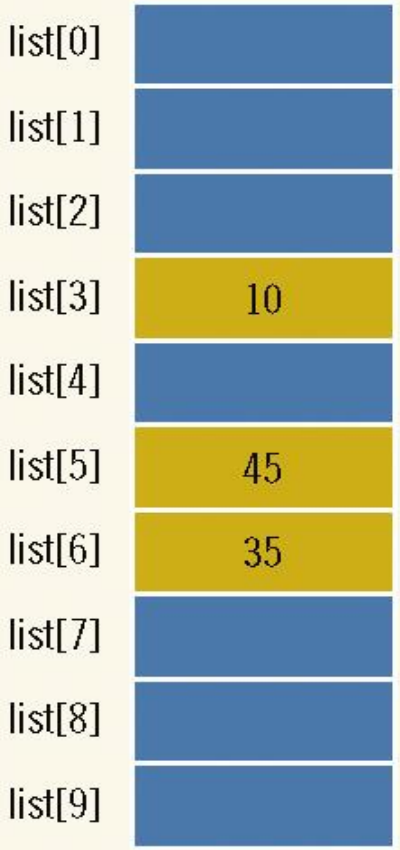


FIGURE 9-4 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];`

Accessing/updating an element

- int list [10];
- list[2 * i – 1] = 5

Array size unknown at compile time?

When you declare an array, its size must be specified. For example, you cannot do the following:

```
int arraySize;                                  //Line 1

cout << "Enter the size of the array: "; //Line 2
cin >> arraySize;                               //Line 3
cout << endl;                                   //Line 4

int list[arraySize];                            //Line 5; not allowed
```

EXAMPLE 9-2

You can also declare arrays as follows:

```
const int ARRAY_SIZE = 10;
int list[ARRAY_SIZE];
```

That is, you can first declare a named constant and then use the value of the named constant to declare an array and specify its size.

Processing One-Dimensional Arrays

- Some basic operations performed on a one-dimensional array are:
  - Initialize
  - Input data
  - Output data stored in an array
  - Find the largest and/or smallest element
- Each operation requires ability to step through the elements of the array
- Easily accomplished by a loop

Accessing Array Components

- Consider the declaration

```
int list[100];  //list is an array
                //of the size 100
int i;
```

- This `for` loop steps-through each element of the array list starting at the first element

```
for (i = 0; i < 100; i++) //Line 1
        //process list[i]  //Line 2
```

Accessing Array Components (continued)

- If processing list requires inputting data into list
  - the statement in Line 2 takes the from of an input statement, such as the `cin` statement

```
for (i = 0; i < 100; i++)  //Line 1
    cin >> list[i];
```

EXAMPLE 9-3

```
double sales[10];
int index;
double largestSale, sum, average;
```

Initializing an array:

```
for (index = 0; index < 10; index++)
    sales[index] = 0.0;
```

Reading data into an array:

```
for (index = 0; index < 10; index++)
    cin >> sales[index];
```

Printing an array:

```
for (index = 0; index < 10; index++)
    cout << sales[index] << " ";
```

Finding the sum and average of an array:

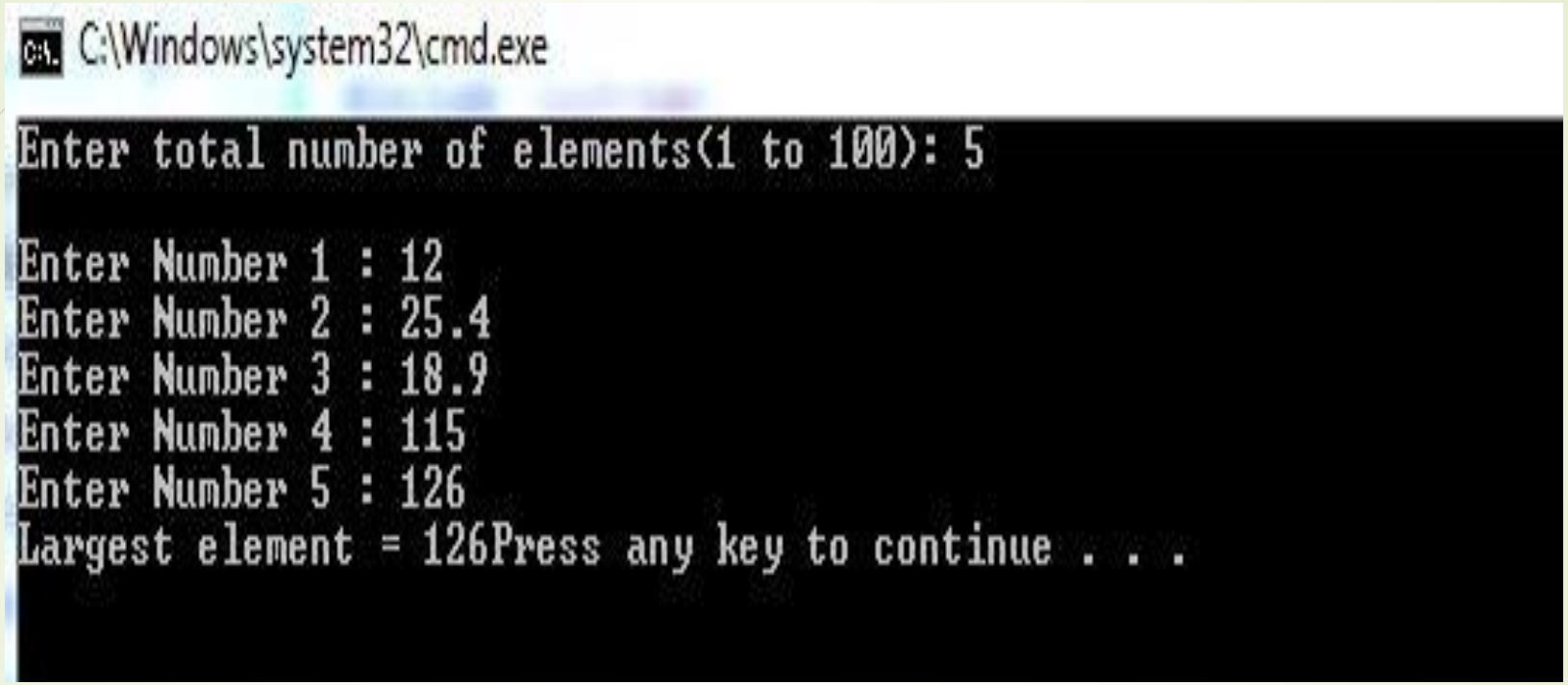
```
sum = 0;
for (index = 0; index < 10; index++)
     sum = sum + sales[index];

average = sum / 10;
```

Largest element in the array:

```
maxIndex = 0;
for (index = 1; index < 10; index++)
     if (sales[maxIndex] < sales[index])
          maxIndex = index;
largestSale = sales[maxIndex];
```

Display Largest Element of an array

```
int main()
{   int i, n;
    float arr[100];
    cout << "Enter total number of elements(1 to 100):
";
    cin >> n;
    cout << endl;
    for(i = 0; i < n; ++i)
    {  cout << "Enter Number " << i + 1 << " : ";
       cin >> arr[i];}
   for(i = 1;i < n; ++i)
    { // Change < to > if you want to find the smallest
element
       if(arr[0] < arr[i])
           arr[0] = arr[i];}
    cout << "Largest element = " << arr[0];
    return 0;}
```


C:\Windows\system32\cmd.exe
Enter total number of elements(1 to 100): 5
Enter Number 1 : 12
Enter Number 2 : 25.4
Enter Number 3 : 18.9
Enter Number 4 : 115
Enter Number 5 : 126
Largest element = 126Press any key to continue . . .

Array Index Out of Bounds

- If we have the statements:

```
double num[10];

int i;
```

- The component `num[i]` is a valid index if `i = 0, 1, 2, 3, 4, 5, 6, 7, 8,` or `9`
- The index of an array is in bounds if the `index >=0` and the `index <= ARRAY_SIZE-1`

Array Index Out of Bounds (continued)

- If either the `index < 0` or the `index > ARRAY_SIZE-1`
  - then we say that the `index` is out of bounds
- There is no guard against indices that are out of bounds
  - C++ does not check if the index value is within range

Array Initialization

- As with simple variables
  - Arrays can be initialized while they are being declared
- When initializing arrays while declaring them
  - Not necessary to specify the size of the array
- Size of array is determined by the number of initial values in the braces
- For example:

```
double sales[] = {12.25, 32.50, 16.90, 23,
                  45.68};
```

Partial Initialization

- The statement

```
int list[10] = {0};
```

declares `list` to be an array of `10` components and initializes all components to zero

- The statement

```
int list[10] = {8, 5, 12};
```

declares `list` to be an array of `10` components, initializes `list[0]` to `8`, `list[1]` to `5`, `list[2]` to `12` and all other components are initialized to `0`

Partial Initialization (continued)

- The statement

```
int list[] = {5, 6, 3};
```

declares `list` to be an array of `3` components and initializes `list[0]` to `5`, `list[1]` to `6`, and `list[2]` to `3`

- The statement

```
int list[25]= {4, 7};
```

declares `list` to be an array of `25` components

  - The first two components are initialized to `4` and `7` respectively
  - All other components are initialized to `0`

Restrictions on Array Processing

```
int myList[5] = {0, 4, 8, 12, 16};  //Line 1
int yourList[5];  //Line 2
```

Assignment does not work with arrays

```
yourList = myList;  //illegal
```

In order to copy one array into another array we must copy component-wise

```
for (int index = 0; index < 5; index ++)
    yourList[index] = myList[index];
```

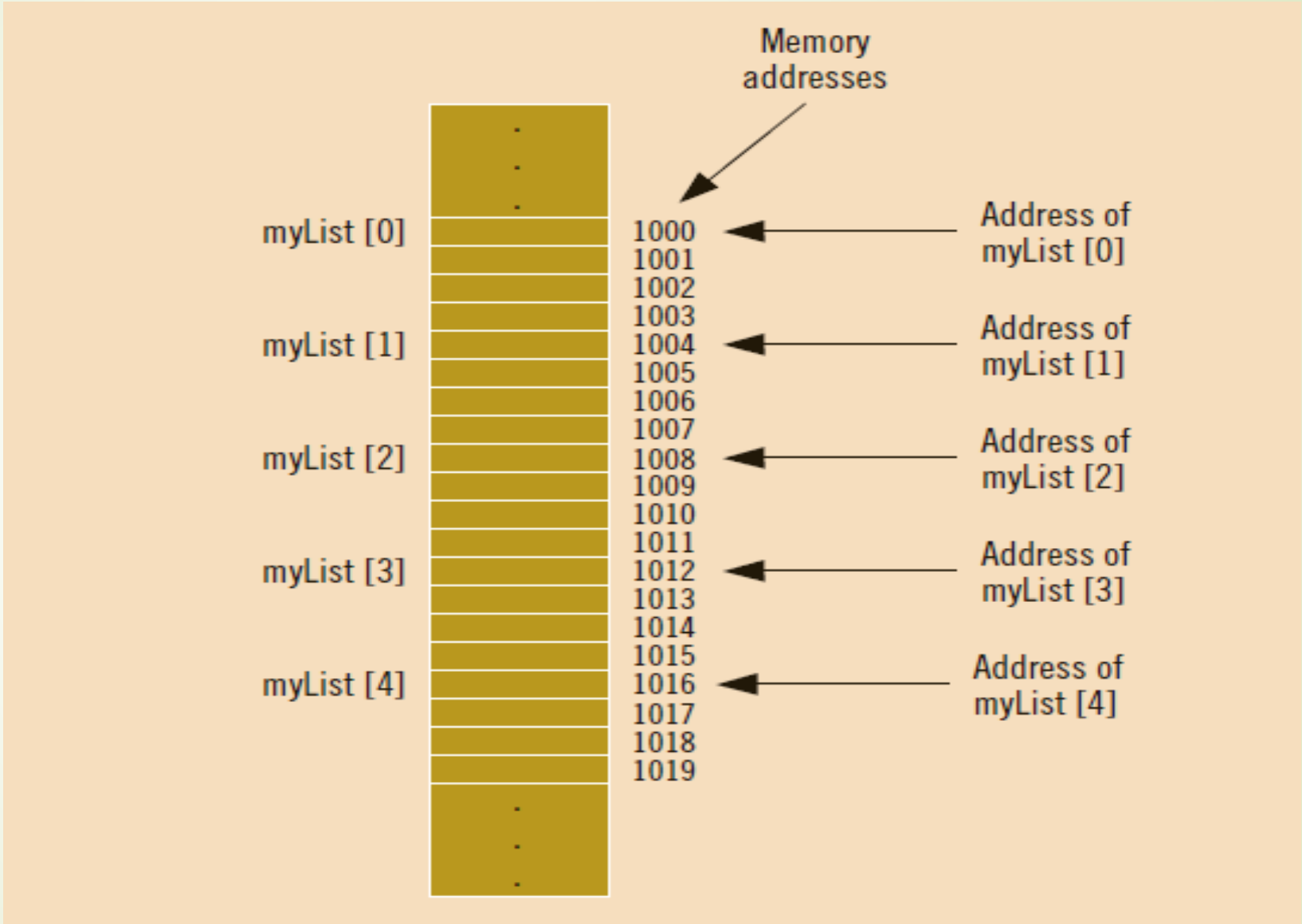
Restrictions on Array Processing (continued)

```
cin >> yourList; //illegal
```

```
for (int index = 0; index < 5; index ++)
     cin >> yourList[index];
```

Base Address of an Array

- The base address of an array is the address, or memory location of the first array component
- If `list` is a one-dimensional array
  - base address of list is the address of the component `list[0]`
- When we pass an array as a parameter
  - base address of the actual array is passed to the formal parameter
- Functions cannot return a value of the type array


Memory addresses
myList [0]
myList [1]
myList [2]
myList [3]
myList [4]
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
Address of myList [0]
Address of myList [1]
Address of myList [2]
Address of myList [3]
Address of myList [4]

Address of a specific element

- 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: **`1012`**, **`1013`**, **`1014`**, and **`1015`**.

Where is the base address stored?

- Now **`myList`** is the name of an array. There is also a memory space associated with the identifier **`myList`**, and the base address of the array is stored in that memory space. Consider the following statement:
- **`cout << myList << endl;`**
- **`int yourList[5];`**
- Then, in the statement:
- **`if (myList <= yourList)`**
- the expression **`myList <= yourList`** evaluates to **`true`** if the base address of the array **`myList`** is less than the base address of the array **`yourList`**; and evaluates to **`false`** otherwise.
- It *does not* determine whether the elements of **`myList`** are less than or equal to the corresponding elements of **`yourList`**

References

1. C++ Programming: From Problem Analysis to Program Design, Third Edition
2. https://www.just.edu.jo/~yahya-t/cs115/