Chapter 7: Single-Dimensional Arrays and C-Strings

Sections 7.1)7.7, 7.11

Textbooks: Y. Daniel Liang, Introduction to Programming with C++, 3rd Edition

These slides were adapted by Prof. Gheith Abandah from the Computer Engineering Department of the University of Jordan for the Course: Computer Skills for Engineers (0907101)
Updated by Dr. Ashraf Suyyagh (Summer 2021)

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

Introduction

- How to read one hundred numbers and compute their average?
-
- Use **A1**, **A2**, …, **A100**?
- Or use a single array that stores all the numbers?

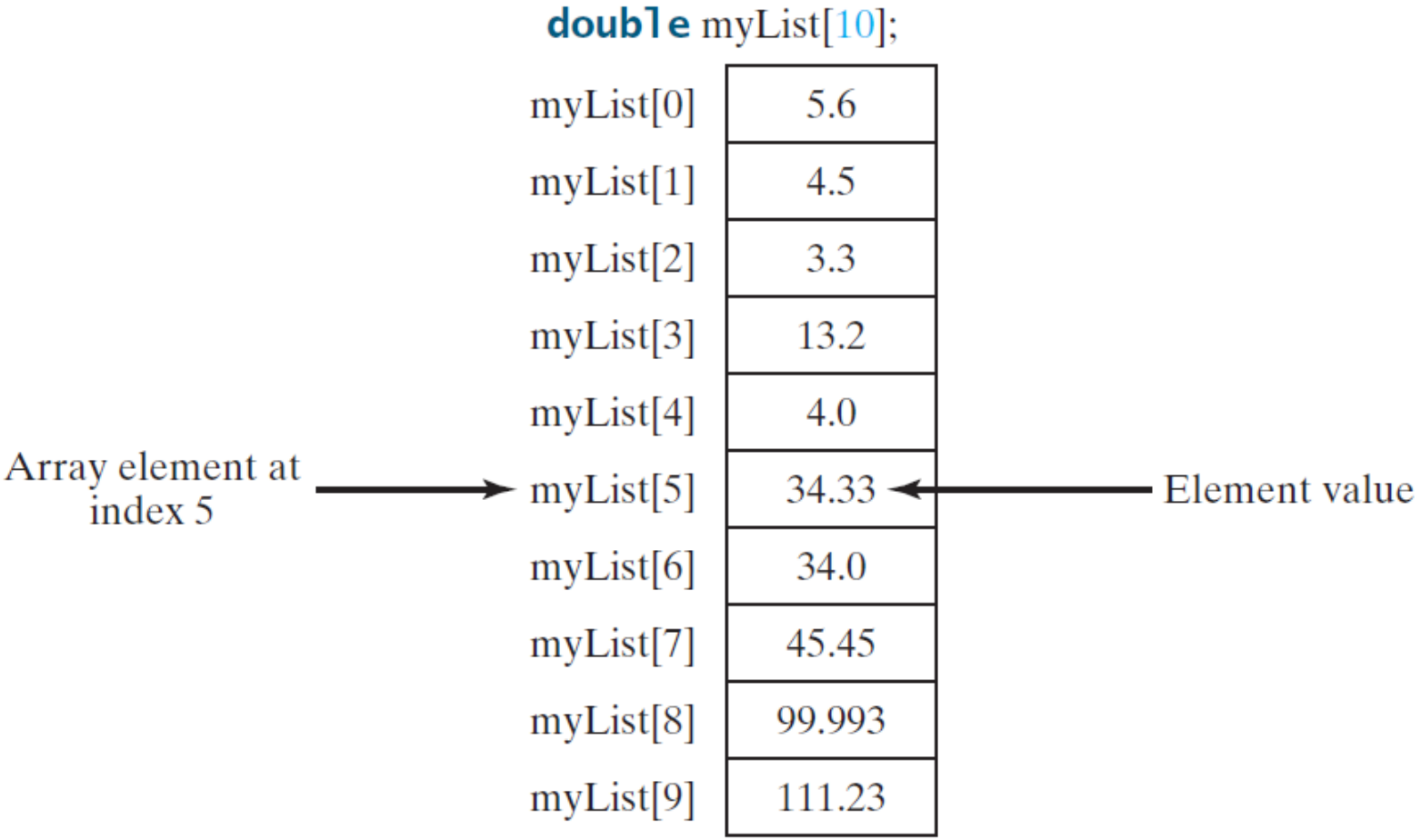
```
#include <iostream>
using namespace std;

int main()
{
    double numbers[100];
    double sum = 0;

    for (int i = 0; i < 100; i++)
    {
        cout << "Enter a number:
";
        cin >> numbers[i];
        sum += numbers[i];
    }
    double average = sum / 100;
    cout << "Average is " <<
average
         << endl;
    return 0;
}
```

Introduction

Array is a data structure that represents a collection of the same types of data.



Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

Declaring Array Variables

```
datatype arrayRefVar[arraySize];
```

Example:

```
double myList[10];
```

C++ requires that the array size used to declare an array must be a constant expression. For example, the following code is illegal:

```
int size = 10;
double myList[size]; // Wrong
```

But it would be OK, if **size** is a constant as follow:

```
const int size = 10;
double myList[size], list2[5]; // Correct
```

Arbitrary Initial Values

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

They are not initialized.

Accessing Array Elements

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

- Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayName[index];
```

- For example, **myList[9]** represents the last element in the array **myList**.

Using Indexed Variables

- After an array is created, an indexed variable can be used in the same way as a regular variable.
- Examples:

```
myList[2] = myList[0] + myList[1];
myList[3]++;
cout << max(myList[0], myList[1]) << endl;
```

- C++ does not check array's boundary. So, accessing array elements using subscripts beyond the boundary (e.g., `myList[-1]` and `myList[11]`) does not cause syntax errors, but the operating system might report a memory access violation.

Array Initializers

Declaring, creating, initializing in one step:

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

Examples:

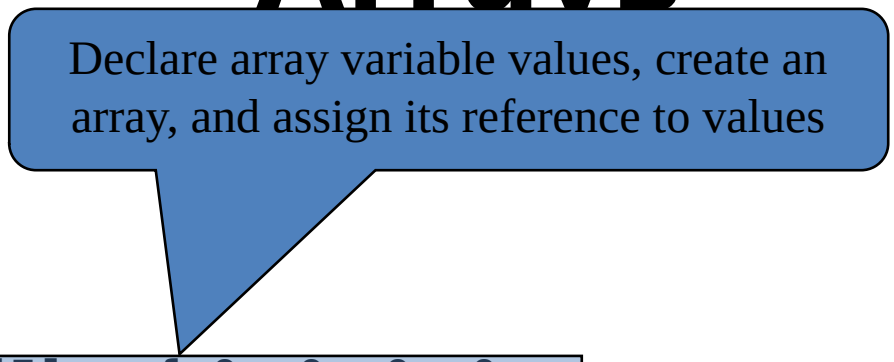
```
double myList[4] = {1.9, 2.9, 3.4, 3.5};

double myList[] = {1.9, 2.9, 3.4, 3.5};

double myList[4] = {1.9, 2.9};
```

animation

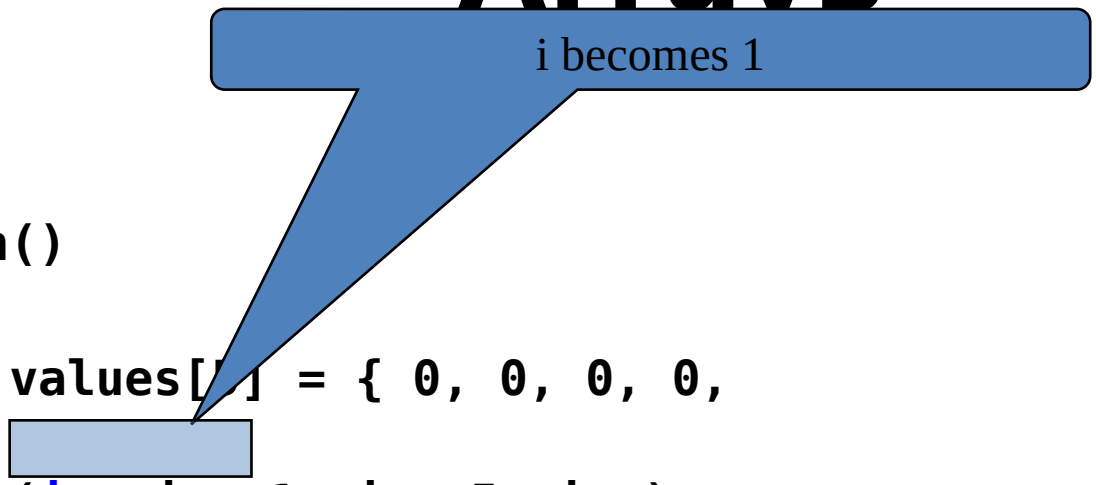
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

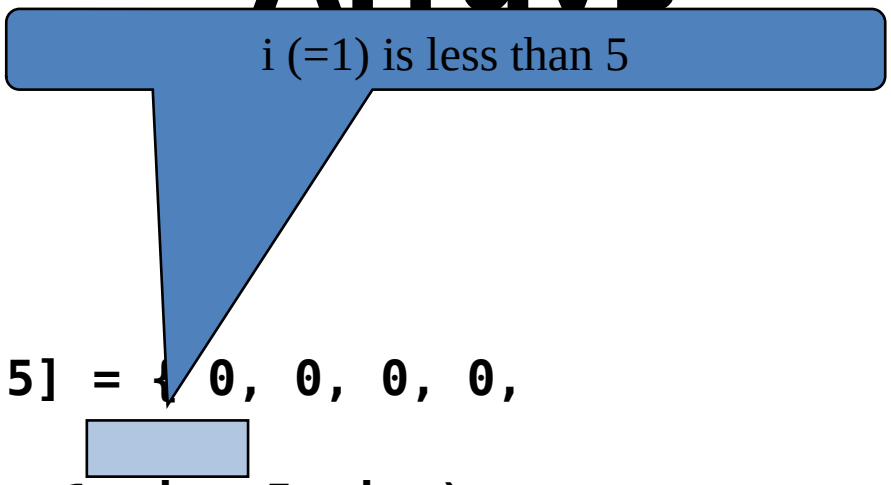
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

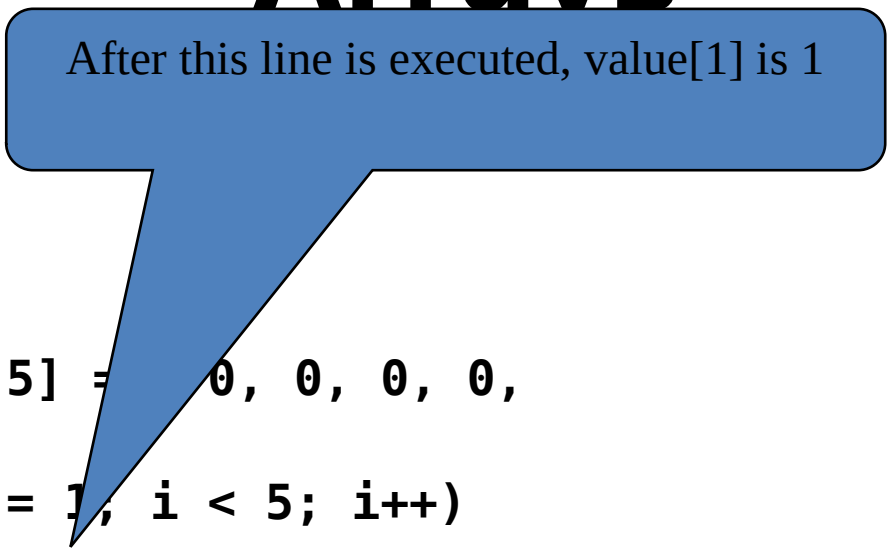
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

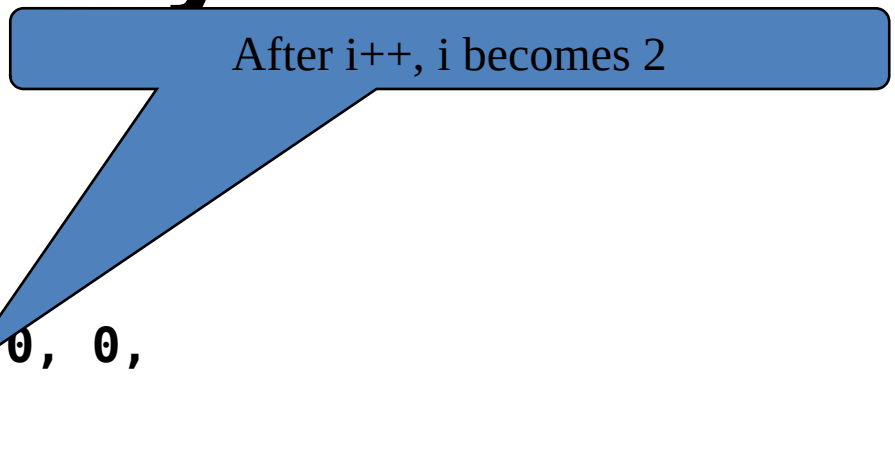
Trace Program with Arrays




```
int main()
{
    int values[5] =    0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

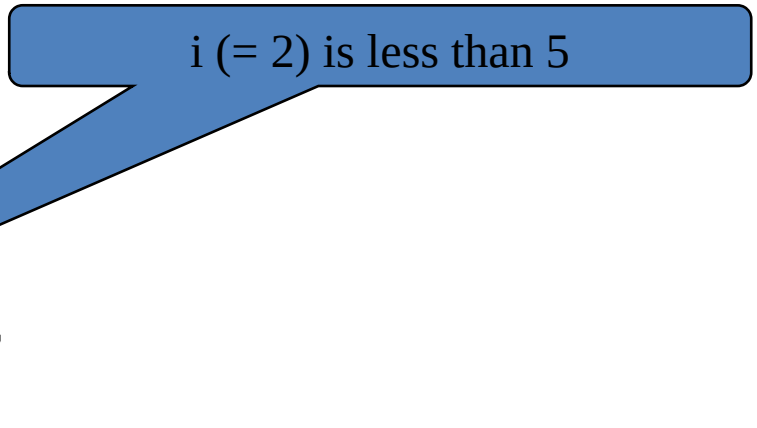
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

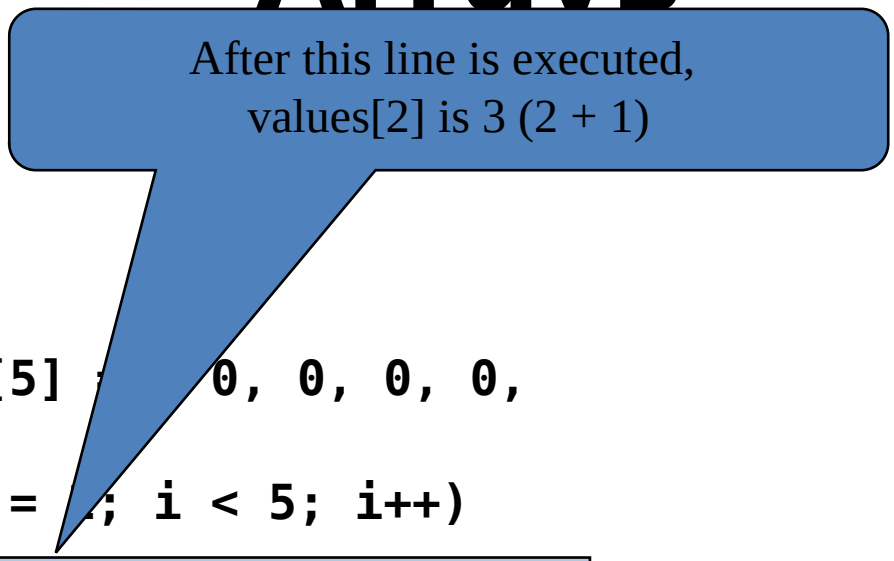
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

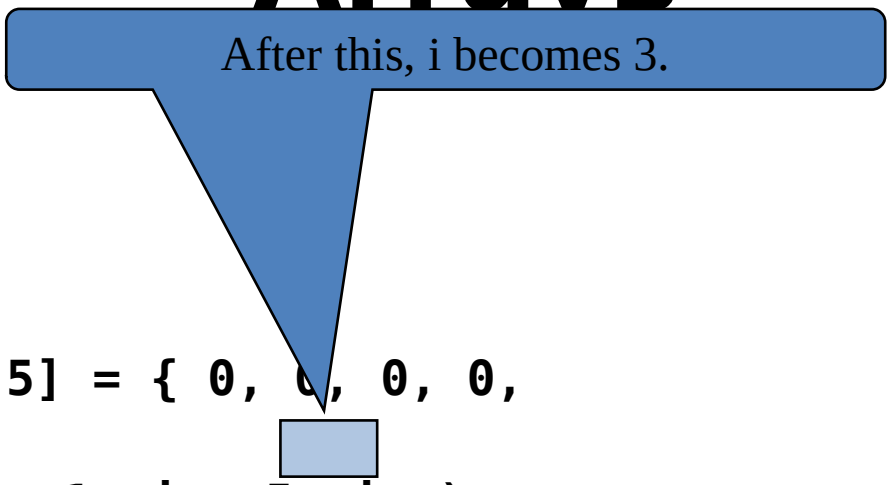
Trace Program with Arrays




```
int main()
{
    int values[5] =    0, 0, 0, 0,
0 };
    for (int i =  ; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

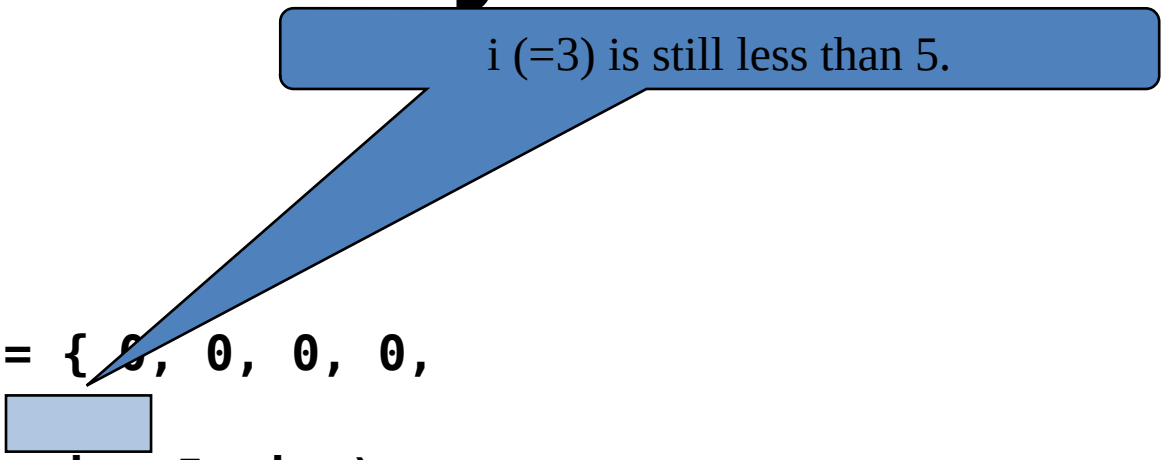
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

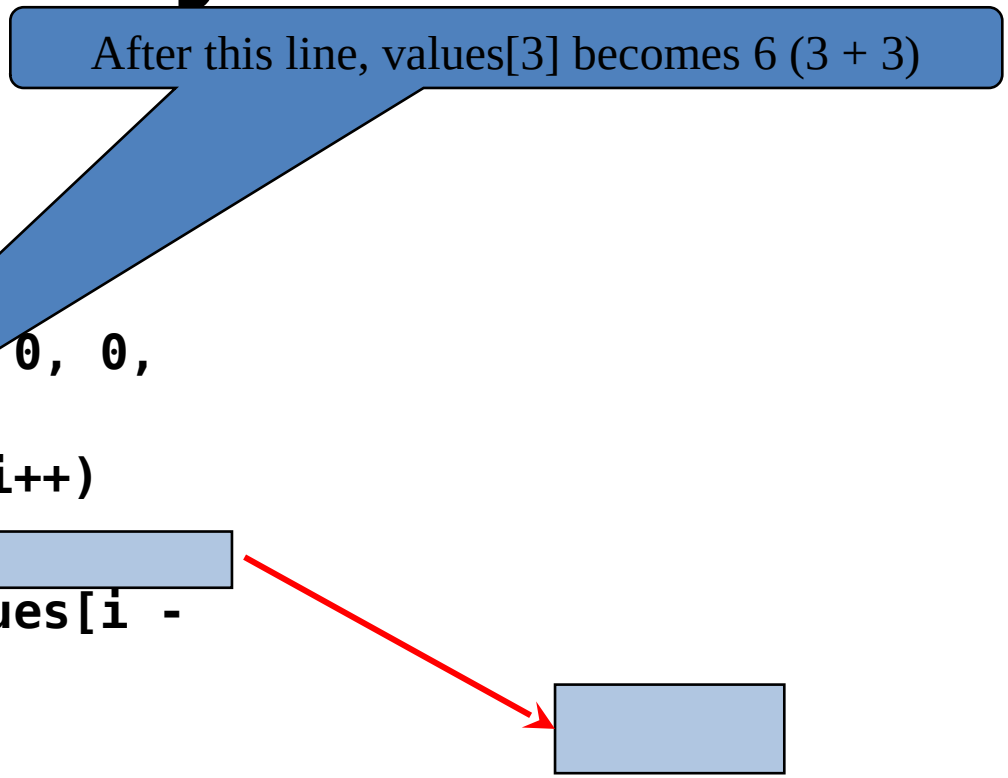
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

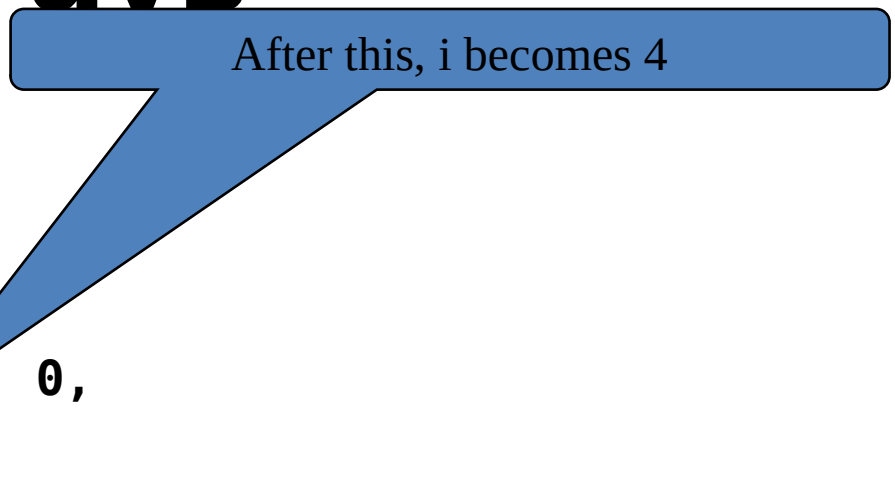
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

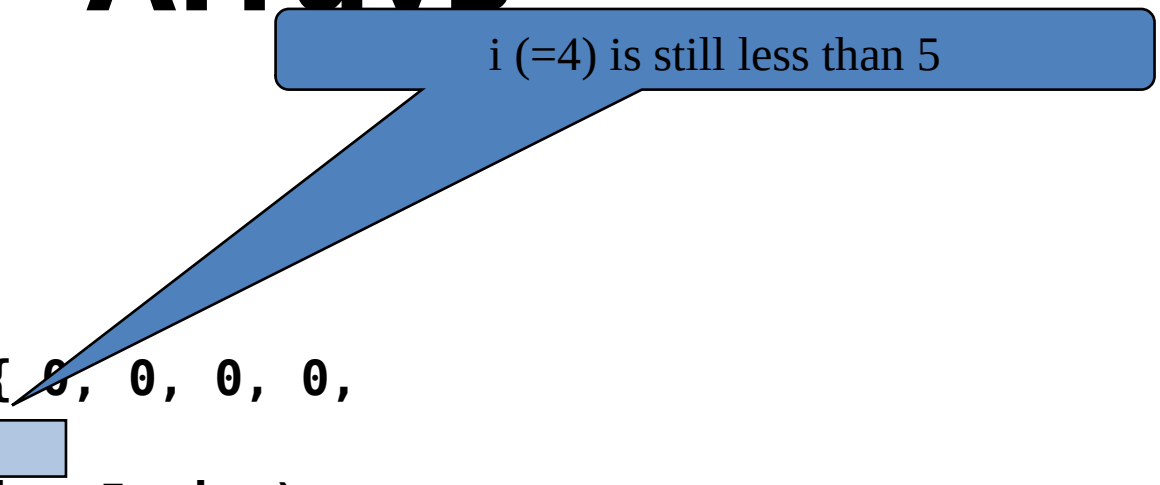
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

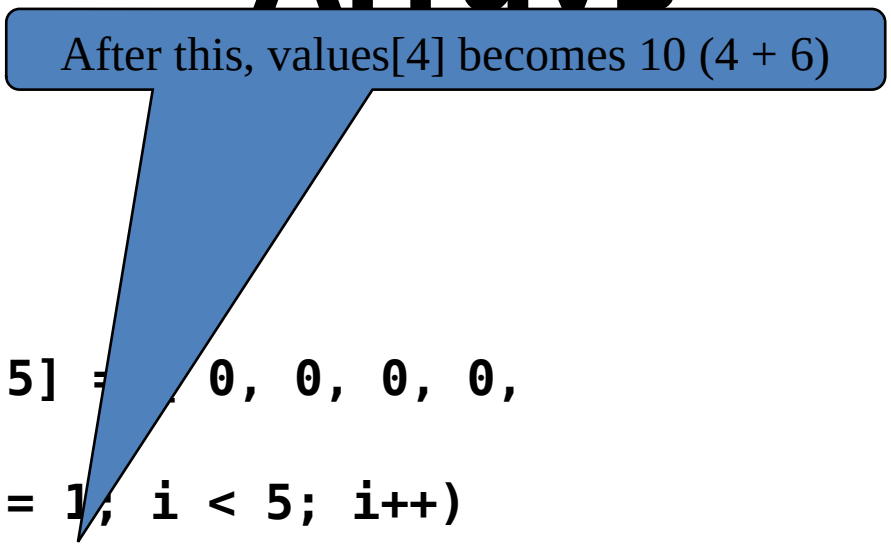
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

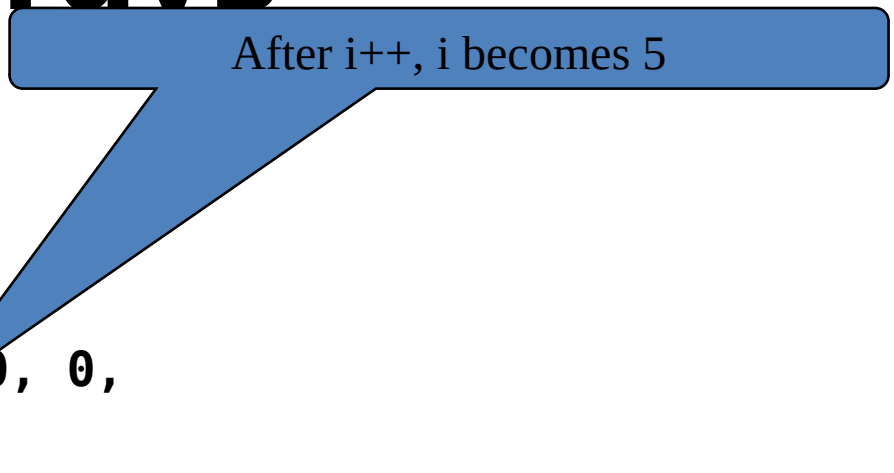
Trace Program with Arrays




```
int main()
{
    int values[5] =  0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

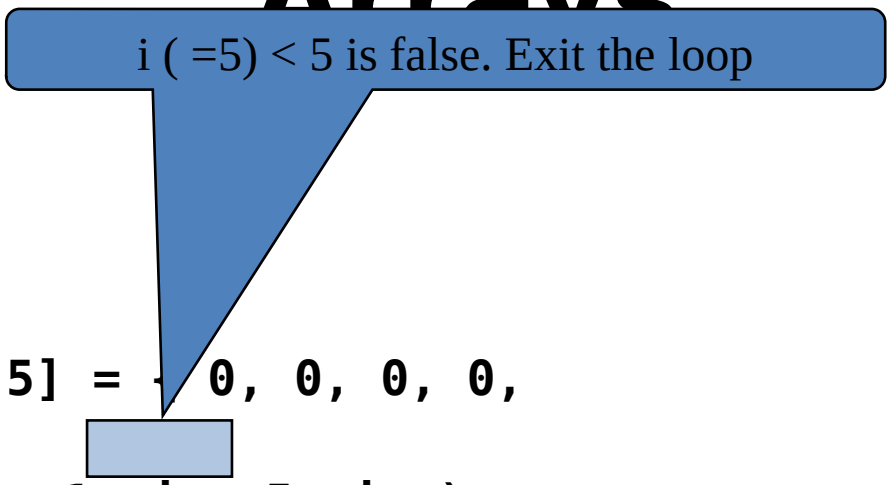
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

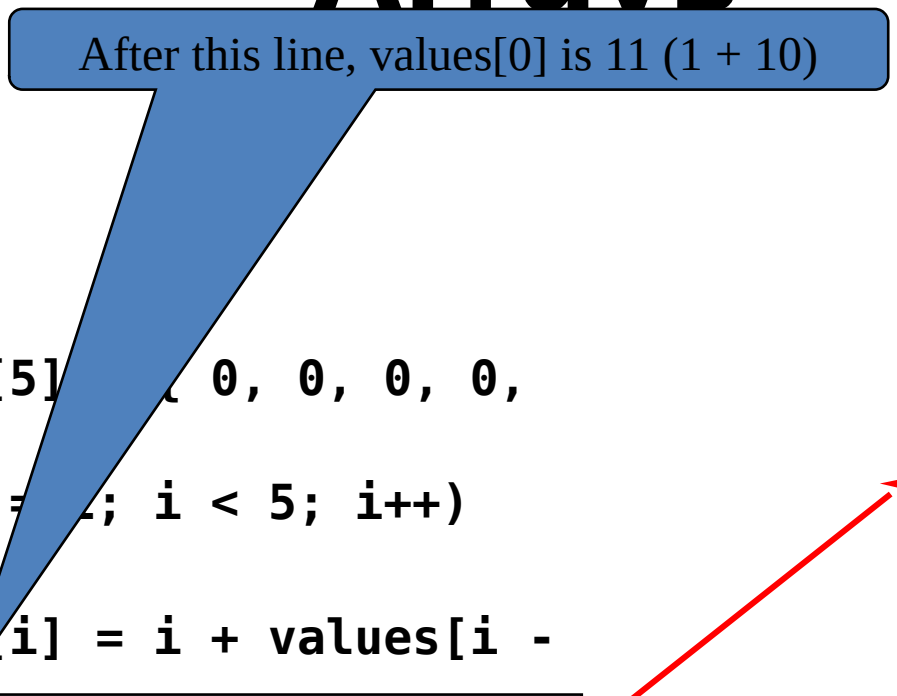
Trace Program with Arrays




```
int main()
{
    int values[5] = { 0, 0, 0, 0,
0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

animation

Trace Program with Arrays




```
int main()
{
    int values[5]    { 0, 0, 0, 0,
0 };
    for (int i =  ; i < 5; i++)
    {
        values[i] = i + values[i -
1];
    }
    values[0] = values[1] +
values[4];
}
```

Processing Arrays

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

```
const int ARRAY_SIZE = 10;
double myList[ARRAY_SIZE];
for (int i = 0; i < ARRAY_SIZE; i++)
{
    myList[i] = rand() % 100;
}
```

- Summing all elements:

```
double total = 0;
for (int i = 0; i < ARRAY_SIZE; i++)
{
    total += myList[i];
}
```

Printing Arrays

To print an array, you have to print each element in the array using a loop like the following:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
    cout << myList[i] << " ";
}
```

Copying Arrays

Can you copy array using a syntax like this?

```
list = myList;  // Does not work
```

This is not allowed in C++. You have to copy individual elements from one array to the other as follows:

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

Finding the Largest Element

- Use a variable named **max** to store the largest element. Initially **max** is **myList[0]**.
- To find the largest element in the array **myList**, compare each element in **myList** with **max**, update **max** if the element is greater than **max**.

```
double max = myList[0];
for (int i = 1; i < ARRAY_SIZE; i++)
{
    if (myList[i] > max)
        max = myList[i];
}
```

Finding the Smallest Index of the Largest Element

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < ARRAY_SIZE; i++)
{
    if (myList[i] > max)
    {
        max = myList[i];
        indexOfMax = i;
    }
}
```

Shifting/Rotating Elements

```
double temp = myList[0]; // Save the first
// Shift elements up
for (int i = 1; i < ARRAY_SIZE; i++)
{
    myList[i - 1] = myList[i];
}
// First element to last position
myList[ARRAY_SIZE - 1] = temp;
```

C++11: Foreach loops are defined in C++11

Foreach Loops

```
double myList[] = { 0, 1.5, 2.1 };
for (double e : myList) {
    cout << e << endl;
}
```

```
0
1.5
2.1
```

Outline

SELF-STUDY Examples are not explained in class because they are long examples. Students are advised to watch the video recorded and provided by the professor

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

supplement

Self-Study Example: Lotto Numbers

The problem is to write a program that checks if all the input numbers cover 1 to 99

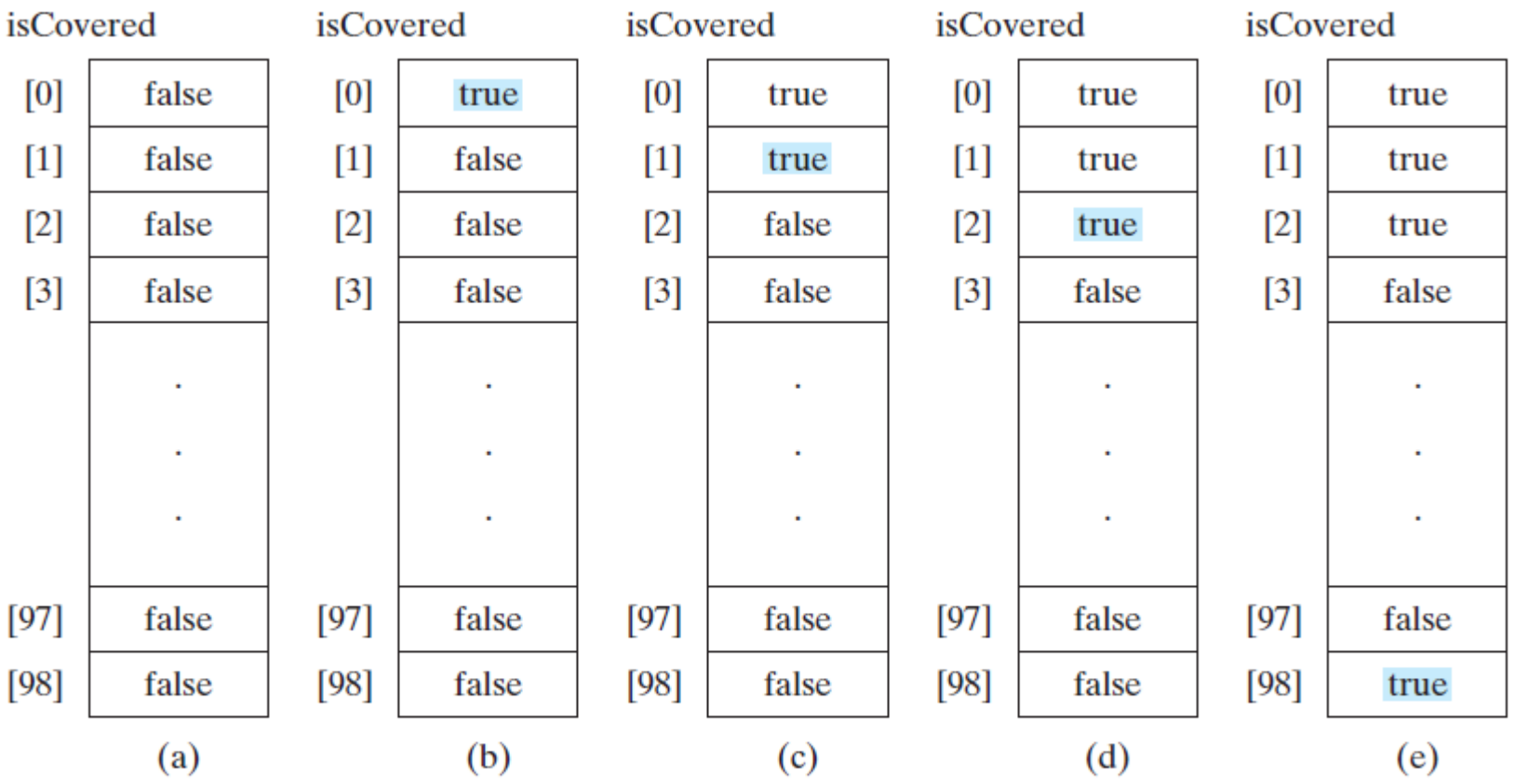


FIGURE 7.2 If number `i` appears in a lotto ticket, `isCovered[i-1]` is set to `true`.

Run

Self-Study Example:
LottoNumbers.cpp 1/2

```
#include <iostream>
using namespace std;

int main()
{
    bool isCovered[99];
    int number; // number read from a file

    // Initialize the array
    for (int i = 0; i < 99; i++)
        isCovered[i] = false;

    // Read each number and mark its corresponding element
    cin >> number;
    while (number != 0)
    {
        isCovered[number - 1] = true;
        cin >> number;
    }
```

Self-Study Example:
LottoNumbers.cpp 2/2

```
    // Check if all covered
    bool allCovered = true; // Assume all covered
initially
    for (int i = 0; i < 99; i++)
        if (!isCovered[i])
        {
            allCovered = false; // Find one number not
covered
            break;
        }

    // Display result
    if (allCovered)
        cout << "The tickets cover all numbers" << endl;
    else
        cout << "The tickets don't cover all numbers" <<
endl;

    return 0;
```

Outline

SELF-STUDY Examples are not explained in class because they are long examples. Students are advised to watch the video recorded and provided by the professor

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

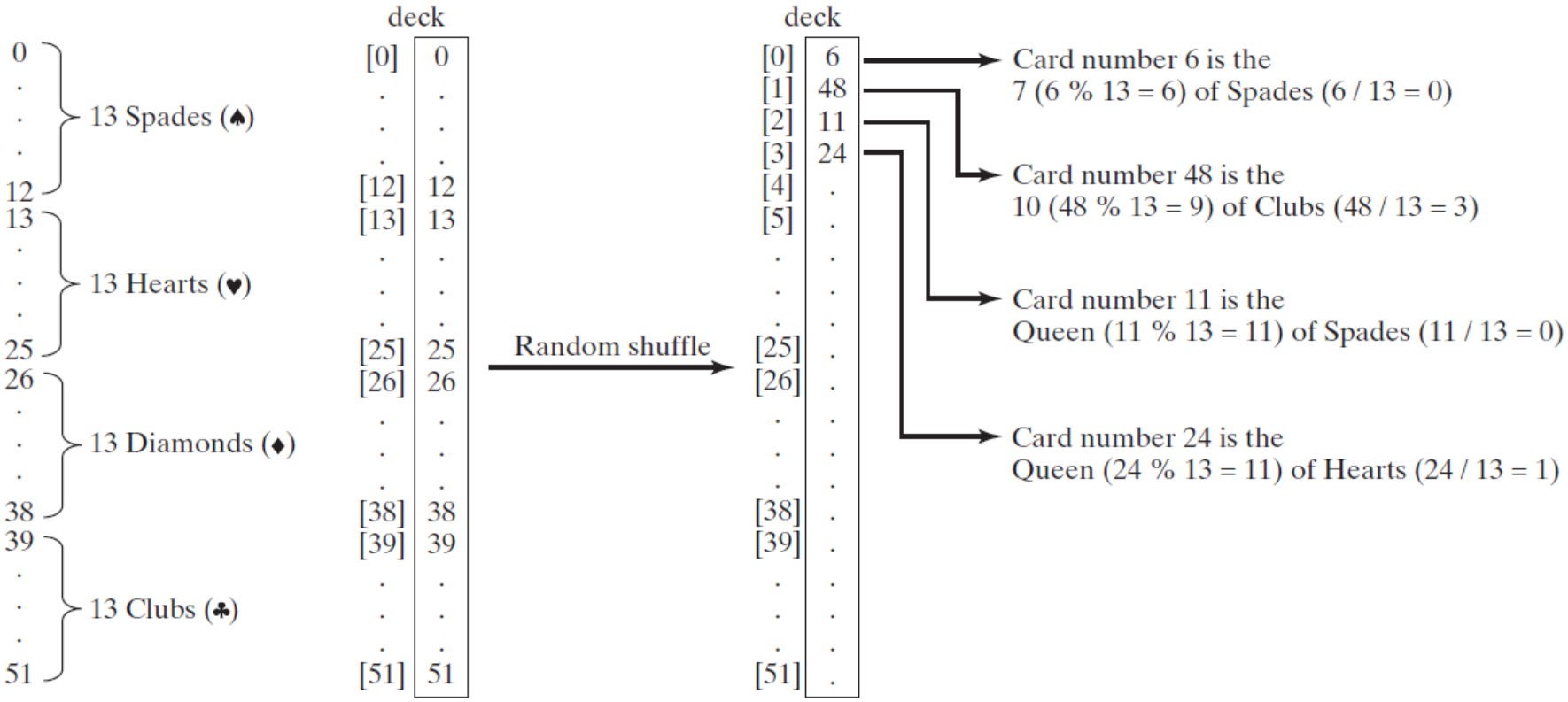
Self-Study Example: Deck of Cards

- The problem is to write a program that picks four cards randomly from a deck of **52** cards.
- All the cards can be represented using an array named **deck**, filled with initial values **0** to **52**, as follows:

```
const int NUMBER_OF_CARDS = 52;
int deck[NUMBER_OF_CARDS];

// Initialize cards
for (int i = 0; i < NUMBER_OF_CARDS; i++)
    deck[i] = i;
```

Self-Study Example: Deck of Cards, cont.



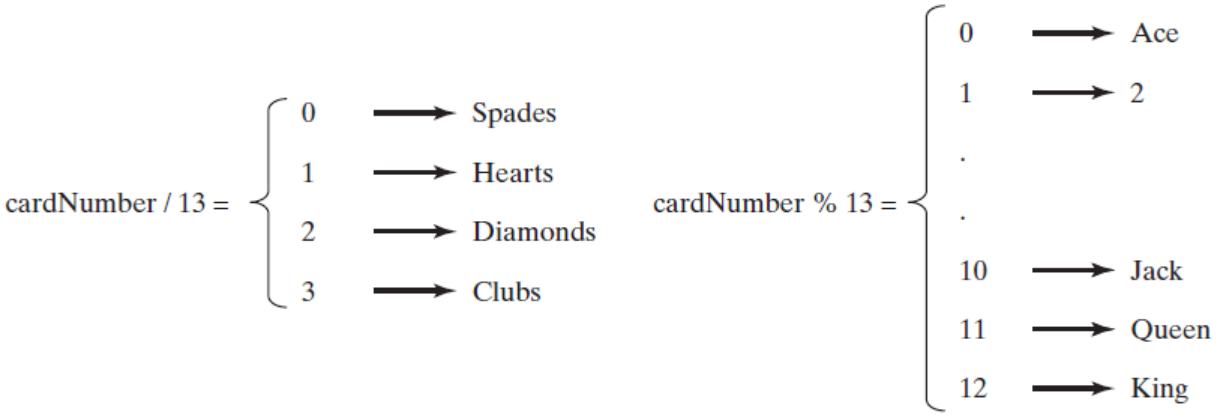
DeckOfCards Run

Self-Study Example:
DeckOfCards.cpp 1/2

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
    const int NUMBER_OF_CARDS = 52;
    int deck[NUMBER_OF_CARDS];
    string suits[] = { "Spades", "Hearts", "Diamonds", "Clubs" };
    string ranks[] = { "Ace", "2", "3", "4", "5", "6", "7", "8",
"9",
      "10", "Jack", "Queen", "King" };

    // Initialize cards
    for (int i = 0; i < NUMBER_OF_CARDS; i++)
        deck[i] = i;
```



Self-Study Example: DeckOfCards.cpp 2/2

```
    // Shuffle the cards
    srand(time(0));
    for (int i = 0; i < NUMBER_OF_CARDS; i++)
    {
        // Generate an index randomly
        int index = rand() % NUMBER_OF_CARDS;
        int temp = deck[i];
        deck[i] = deck[index];
        deck[index] = temp;
    }

    // Display the first four cards
    for (int i = 0; i < 4; i++)
    {
        string suit = suits[deck[i] / 13];
        string rank = ranks[deck[i] % 13];
        cout << "Card number " << deck[i] << ": "
             << rank << " of " << suit << endl;
    }

    return 0;
}
```

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

Passing Arrays to Functions

- You can pass an entire array to a function.
- You need also to pass the size of the array.
- This program gives an example to demonstrate how to declare and invoke this type of functions.

PassArrayDemo Run

PassArrayDemo.cpp

```
#include <iostream>
using namespace std;

void printArray(int list[], int arraySize); // Prototype

int main()
{
    int numbers[6] = { 1, 4, 3, 6, 8, 9 };
    printArray(numbers, 6); // Invoke the function

    return 0;
}

void printArray(int list[], int arraySize)
{
    for (int i = 0; i < arraySize; i++)
    {
        cout << list[i] << " ";
    }
}
```

```
1 4 3 6 8 9
```

Pass-by-Value

- Passing an array variable means that the starting address of the array is passed to the formal parameter by value.
- The parameter inside the function references to the same array that is passed to the function. No new arrays are created.

EffectOfPassArrayDemo Run

EffectOfPassArrayDemo.cpp

```
#include <iostream>
using namespace std;

void m(int, int[]);

int main()
{
    int x = 1; // x represents an int value
    int y[10] = { 0 }; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    cout << "x is " << x << endl;
    cout << "y[0] is " << y[0] << endl;

    return 0;
}

void m(int number, int numbers[])
{
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
}
```

```
x is 1
y[0] is 5555
```

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

Preventing Changes of Array Arguments in Functions

- Passing arrays by reference makes sense for performance reasons. If an array is passed by value, all its elements must be copied into a new array.
- However, passing arrays by its reference value could lead to errors if your function changes the array accidentally.
- To prevent it from happening, you can put the **`const`** to tell the compiler that the array cannot be changed.
- The compiler will report errors if the code in the function attempts to modify the array.

ConstArrayDemo Compile error

ConstArrayDemo.cpp

```
#include <iostream>
using namespace std;

void p(int const list[], int arraySize)
{
  // Modify array accidentally
  list[0] = 100; // Compile error!
}

int main()
{
  int numbers[5] = {1, 4, 3, 6, 8};
  p(numbers, 5);

  return 0;
}
```

```
1>C:\ConstArrayDemo.cpp(7,18): error C3892: 'list': you cannot assign to a
variable that is const
1>Done building project "Testing.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

Returning Arrays from Functions

- How to return an array from a function?
- You may attempt to declare a function that returns a new array that is a reversal of an array as follows:

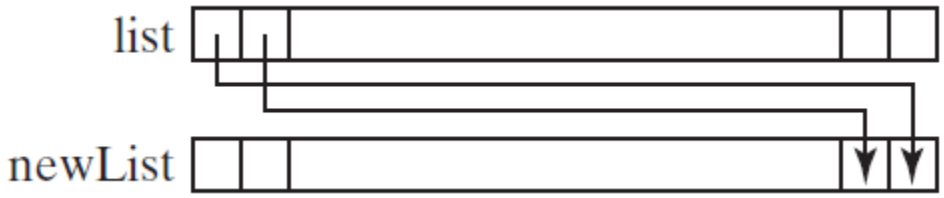
```
// Return the reversal of list
int[] reverse(const int list[], int size);
```

- This is not allowed in C++.

Returning Arrays from Functions, cont.

- However, you can pass two array arguments in the function, as follows:

```
// newList is the reversal of list
void reverse(const int list[], int
newList[],
             int size);
```



ReverseArray Run

ReverseArray.cpp 1/2

```
#include <iostream>
using namespace std;

// newList is the reversal of list
void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}

void printArray(const int list[], int size)
{
    for (int i = 0; i < size; i++)
        cout << list[i] << " ";
```

ReverseArray.cpp 1/2

```
int main()
{
    const int SIZE = 6;
    int list[] = { 1, 2, 3, 4, 5, 6 };
    int newList[SIZE];

    reverse(list, newList, SIZE);

    cout << "The original array: ";
    printArray(list, SIZE);
    cout << endl;

    cout << "The reversed array: ";
    printArray(newList, SIZE);
    cout << endl;

    return 0;
}
```

animation

Trace the reverse Function

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

list

1	2	3	4	5	6

newList

0	0	0	0	0	0

animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 0 and j = 5

list	1	2	3	4	5	6

newList	0	0	0	0	0	0

animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (= 0) is less than 6

list	1	2	3	4	5	6

newList	0	0	0	0	0	0

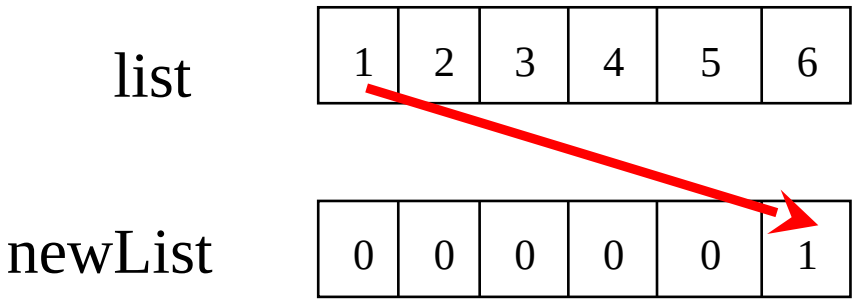
animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 0 and j = 5
Assign list[0] to result[5]



animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

After this, i becomes 1 and j becomes 4

list

1	2	3	4	5	6

newList

0	0	0	0	0	1

animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (=1) is less than 6

list	1	2	3	4	5	6

newList	0	0	0	0	0	1

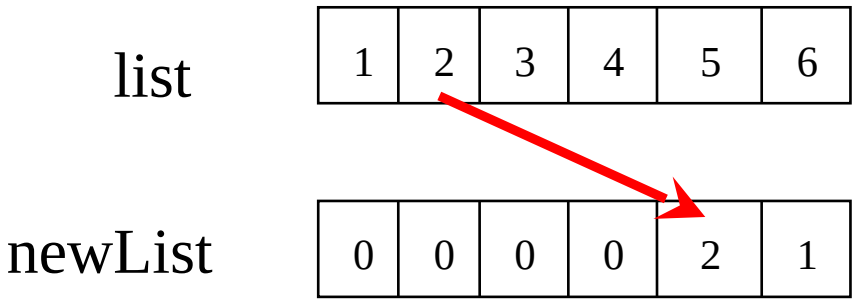
animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 1 and j = 4
Assign list[1] to result[4]



animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

After this, i becomes 2 and j becomes 3

list

1	2	3	4	5	6

newList

0	0	0	0	2	1

animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (=2) is still less than 6

list

1	2	3	4	5	6

newList

0	0	0	0	2	1

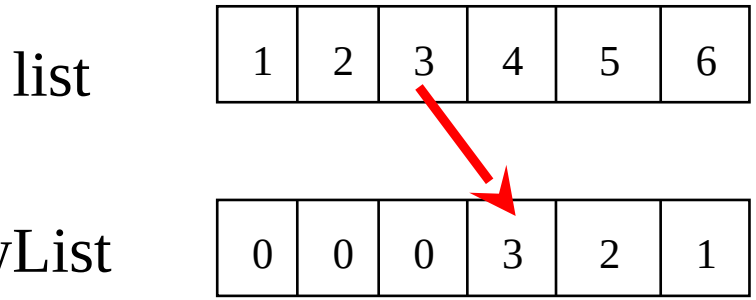
animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 2 and j = 3
Assign list[i] to result[j]



animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

After this, i becomes 3 and j becomes 2

list

1	2	3	4	5	6

newList

0	0	0	3	2	1

animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (=3) is still less than 6

list	1	2	3	4	5	6

newList	0	0	0	3	2	1

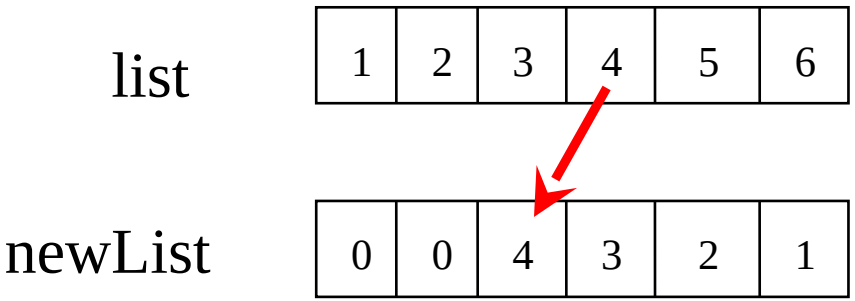
animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 3 and j = 2
Assign list[i] to result[j]

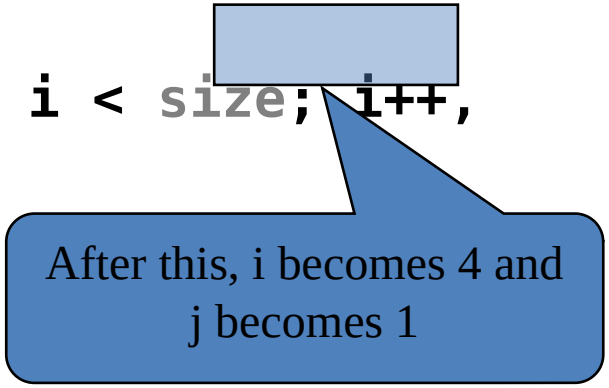


animation

Trace the reverse function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```



list

1	2	3	4	5	6

newList

0	0	4	3	2	1

animation

Trace the reverse Function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (=4) is still less than 6

list	1	2	3	4	5	6

newList	0	0	4	3	2	1

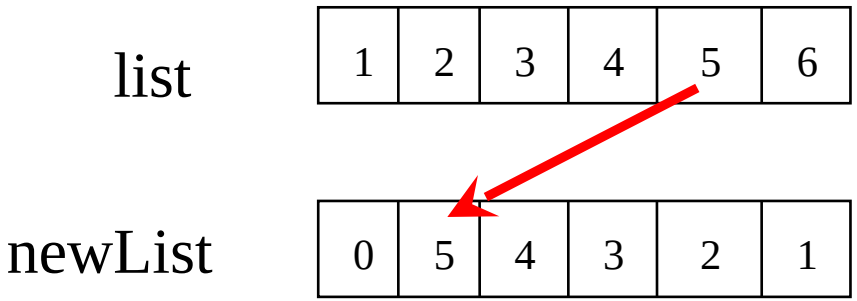
animation

Trace the reverse Function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 4 and j = 1
Assign list[i] to result[j]



animation

Trace the reverse Function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

After this, i becomes 5 and j becomes 0

list

1	2	3	4	5	6

newList

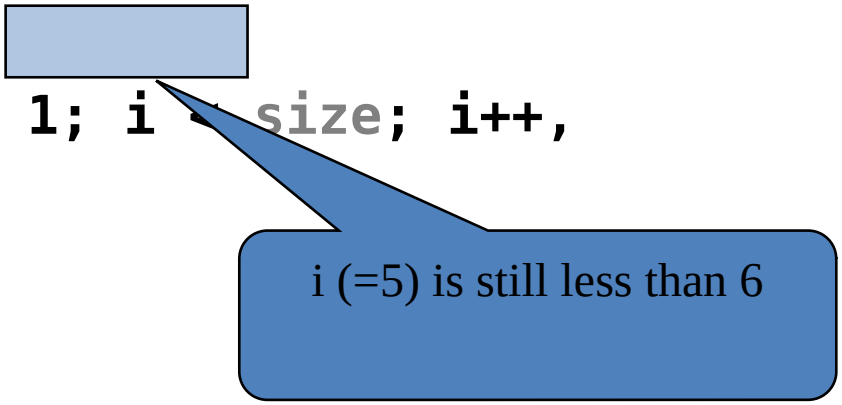
0	5	4	3	2	1

animation

Trace the reverse Function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```



list	1	2	3	4	5	6

newList	0	5	4	3	2	1

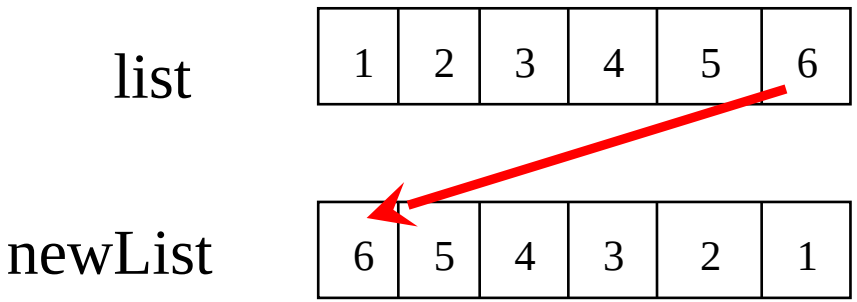
animation

Trace the reverse Function, cont.

```
int list[] = { 1, 2, 3, 4, 5, 6 };
reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i = 5 and j = 0
Assign list[i] to result[j]



animation

Trace the reverse Function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

After this, i becomes 6 and j becomes -1

list	1	2	3	4	5	6

newList	6	5	4	3	2	1

animation

Trace the reverse function, cont.

```
  int list[] = { 1, 2, 3, 4, 5, 6 };
  reverse(list, newList, SIZE);

void reverse(const int list[], int newList[], int
size)
{
    for (int i = 0, j = size - 1; i < size; i++,
j--)
    {
        newList[j] = list[i];
    }
}
```

i (=6) < 6 is false. So exit the loop.

list	1	2	3	4	5	6

newList	6	5	4	3	2	1

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings

C-Strings

- You studied the string type in Chapter 4.
- Example:

```
string s = "welcome to C++";
s.at(0) = 'W';
cout << s.length() << s[0] << endl;
```

```
14W
```

- Here we study the older C-strings because of their popularity.

Initializing Character Arrays

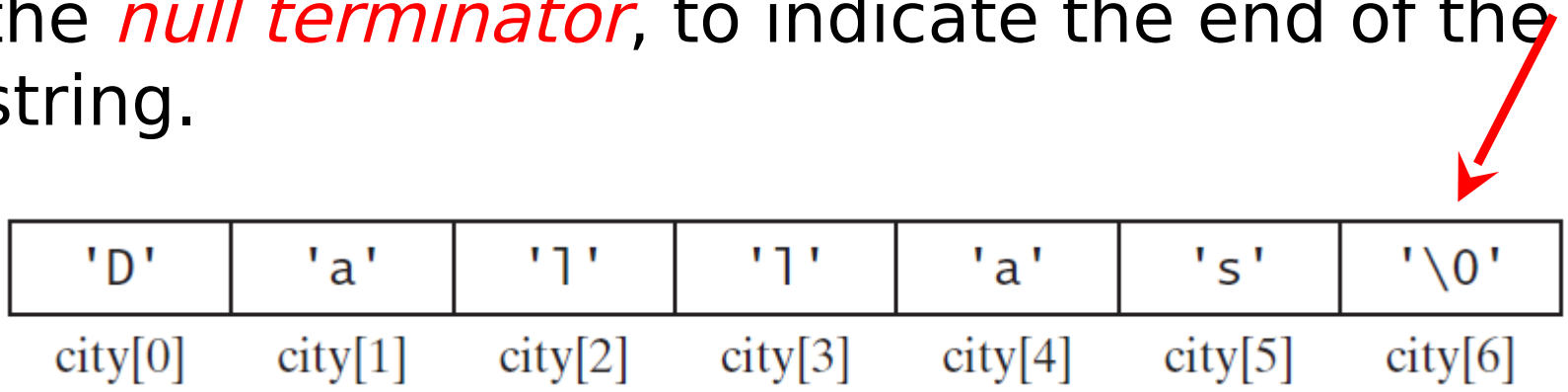
- You can define arrays of characters.

```
char city[] = { 'D', 'a', 'l', 'l', 'a', 's' };
```

- C-strings are defined as follows:

```
char city[] = "Dallas";
```

- In this case, C++ adds the character '\0', called the *null terminator*, to indicate the end of the string.



Reading C-Strings

You can read a string from the keyboard using the `cin` object. For example, see the following code:

```
char city[10];
cout << "Enter a city: ";
cin >> city; // read to array city
cout << "You entered " << city << endl;
```

Printing Character Array

For a character array, it can be printed using one print statement. For example, the following code displays Dallas:

```
char city[] = "Dallas";
cout << city;
```

Reading C-Strings Using getline

- C++ provides the **`cin.getline`** function in the **`iostream`** header file, which reads a string into an array:

```
cin.getline(char array[], int size, char delimitChar);
```

- The function stops reading characters when the delimiter character is encountered or when the **`size - 1`** number of characters are read.
- The last character in the array is reserved for the null terminator (**`'\0'`**).
- If the delimiter is encountered, it is read, but not stored in the array.
- The third argument **`delimitChar`** has a default value (**`'\n'`**).

Working with C-Strings

- The following function finds the length of a C-string:

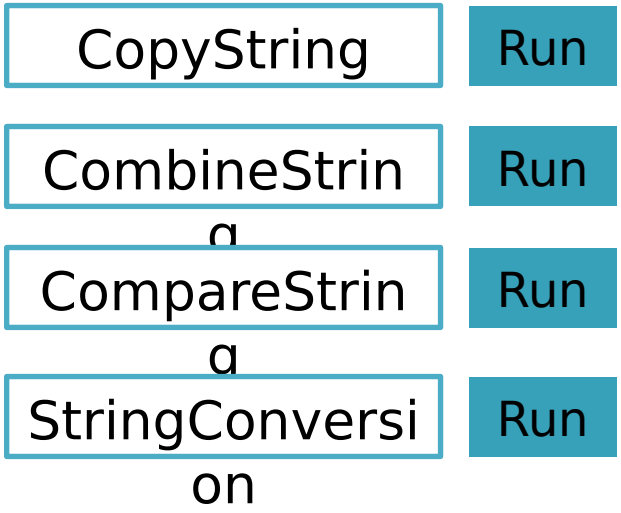
```
unsigned int strlen(char s[])
{
    for (int i = 0; s[i] != '\0'; i++)
;
    return i;
}
```

- The **cstring** and **cstdlib** headers provide many useful C-strings functions.

C-String Functions

Function	*Description*
`size_t strlen(char s[])`	Returns the length of the string, i.e., the number of the characters before the null terminator.
`strcpy(char s1[], const char s2[])`	Copies string s2 to string s1.
`strncpy(char s1[], const char s2[], size_t n)`	Copies the first n characters from string s2 to string s1.
`strcat(char s1[], const char s2[])`	Appends string s2 to s1.
`strncat(char s1[], const char s2[], size_t n)`	Appends the first n characters from string s2 to s1.
`int strcmp(char s1[], const char s2[])`	Returns a value greater than 0, 0, or less than 0 if s1 is greater than, equal to, or less than s2 based on the numeric code of the characters.
`int strncmp(char s1[], const char s2[], size_t n)`	Same as strcmp, but compares up to n number of characters in s1 with those in s2.
`int atoi(char s[])`	Returns an int value for the string.
`double atof(char s[])`	Returns a double value for the string.
`long atol(char s[])`	Returns a long value for the string.
`void itoa(int value, char s[], int radix)`	Obtains an integer value to a string based on specified radix.

C-String Examples



CopyString.CPP

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s1[20];
    char s2[20] = "Dallas
    char s3[20] = "AAAAAAAAAA";

    strcpy(s1, s2);
    strncpy(s3, s2, 6);
    s3[6] = '\0'; // Insert null terminator

    cout << "The string in s1 is " << s1 << endl;
    cout << "The string in s2 is " << s2 << endl;
    cout << "The string in s3 is " << s3 << endl;
    cout << "The length of string s3 is " << strlen(s3) <<
endl;

    return 0;
```

```
The string in s1 is Dallas,
Texas
The string in s2 is Dallas,
Texas
The string in s3 is Dallas
The length of string s3 is 6
```

CombineString.cpp

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s1[20] = "Da
    char s2[20] = "Texas, USA";
    char s3[20] = "Dallas";

    strcat(strcat(s1, ", "), s2);
    strncat(strcat(s3, ", "), s2, 5);

    cout << "The string in s1 is " << s1 << endl;
    cout << "The string in s2 is " << s2 << endl;
    cout << "The string in s3 is " << s3 << endl;
    cout << "The length of string s1 is " << strlen(s1) <<
endl;
    cout << "The length of string s3 is " << strlen(s3) <<
endl;
```

```
The string in s1 is Dallas, Texas,
USA
The string in s2 is Texas, USA
The string in s3 is Dallas, Texas
The length of string s1 is 18
The length of string s3 is 13
```

CompareString.cpp

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s1[] = "abcdefg";
    char s2[] = "abcdg";
    char s3[] = "abcdg";

    cout << "strcmp(s1, s2) is " << strcmp(s1, s2) <<
endl;
    cout << "strcmp(s2, s1) is " << strcmp(s2, s1) <<
endl;
    cout << "strcmp(s2, s3) is " << strcmp(s2, s3) <<
endl;
    cout << "strncmp(s1, s2, 3) is " << strncmp(s1, s2, 3)
         << endl;

    return 0;
}
```

```
strcmp(s1, s2) is -1
strcmp(s2, s1) is 1
strcmp(s2, s3) is 0
strncmp(s1, s2, 3) is
0
```

StringConversion.cpp

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    cout << atoi("4") + atoi("5") << endl;
    cout << atof("4.5") + atof("5.5") << endl;

    char s[10];
    itoa(42, s, 8);
    cout << s << endl;

    itoa(42, s, 10);
    cout << s << endl;

    itoa(42, s, 16);
    cout << s << endl;

    return 0;
}
```

```
9
10
52
42
2a
```

Converting Numbers to Strings

- Note that the **to_string** function is useful to convert numbers to string type.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x = 15;
    double y = 1.32;
    long long int z = 10935;
    string s = "Three numbers: " + to_string(x) + ", " +
        to_string(y) + ", and " + to_string(z);
    cout << s << endl;

    return 0;
}
```

C++11: the to_string function is defined in C++11

```
Three numbers: 15, 1.320000, and
10935
```

Outline

- Introduction
- Array Basics
- Self-Study Example: Lotto Numbers
- Self-Study Example: Deck of Cards
- Passing Arrays to Functions
- Preventing Changes of Array Arguments in Functions
- Returning Arrays from Functions
- C-Strings