

# CS2313 Computer Programming

## LT6 – Array



香港城市大學  
City University of Hong Kong

專業 創新 胸懷全球  
Professional • Creative  
For The World

# Outlines

- 1D and 2D **array** in C++.

# Outcomes

- Array definition
- Array initialization
- Updating array elements
- Printing the content of arrays

# Syntax Summary

- Punctuators-square brackets
  - `[...]`

# 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 = 4;
```

```
double myList[size]; // Wrong
```

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

```
const int size = 4;
```

```
double myList[size]; // Correct
```

# No Bound Checking

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.

# Declaring, creating, initializing Using the Shorthand Notation

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

This shorthand notation is equivalent to the following statements:

```
double myList[4];  
myList[0] = 1.9;  
myList[1] = 2.9;  
myList[2] = 3.4;  
myList[3] = 3.5;
```

# CAUTION

Using the shorthand notation, you have to declare, create, and initialize the array all in **one** statement. Splitting it would cause a syntax error. For example, the following is wrong:

```
double myList[4];
```

```
myList = {1.9, 2.9, 3.4, 3.5};
```



# Implicit Size

C++ allows you to omit the array size when declaring and creating an array using an initializer. For example, the following declaration is fine:

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

C++ automatically figures out how many elements are in the array.

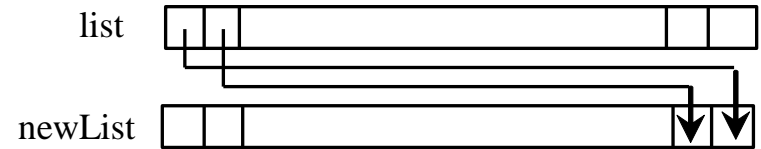
# Partial Initialization

C++ allows you to initialize a part of the array. For example, the following statement assigns values 1.9, 2.9 to the first two elements of the array. The other two elements will be set to zero. Note that if an array is declared, but not initialized, all its elements will contain “garbage”, like all other local variables.

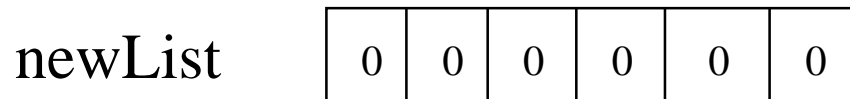
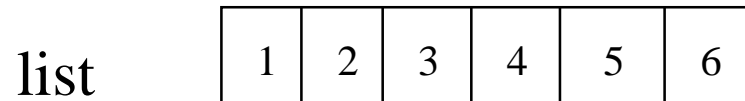
```
double myList[4] = {1.9, 2.9};
```

# Trace the reverse Function

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```



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



# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

**i = 0 and j = 5**

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

**i (= 0) is less than 6**

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
for (int i = 0, j = size - 1; i < size; i++, j--)
```

```
{
```

```
    newList[j] = list[i];
```

```
}
```

**i = 0 and j = 5**  
**Assign list[0] to newList[5]**

list

1	2	3	4	5	6
---	---	---	---	---	---

newList

0	0	0	0	0	1
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

**i (=1) is less  
than 6**

```
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	1
---	---	---	---	---	---

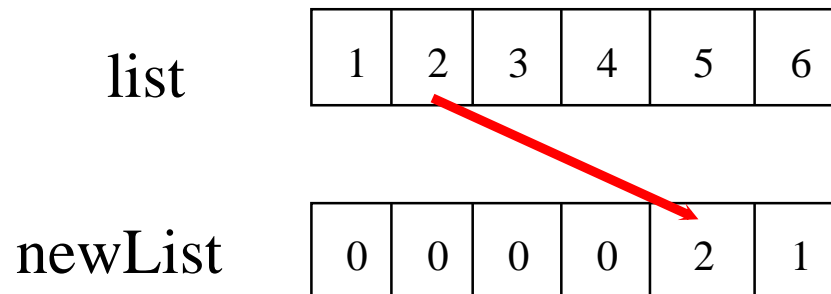


# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

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

**i = 1 and j = 4**  
**Assign list[1] to newList[4]**



# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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

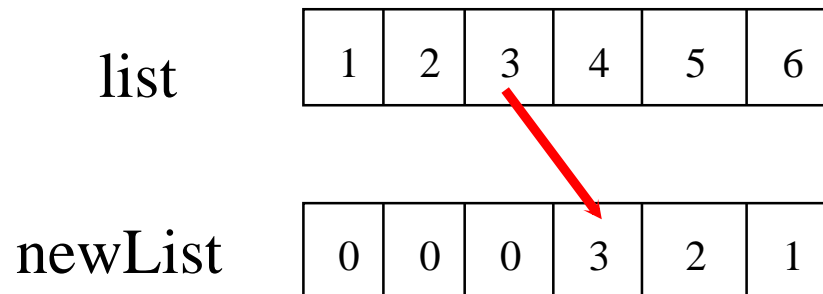
0	0	0	0	2	1
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

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

**i = 2 and j = 3**  
**Assign list[i] to newList[j]**



# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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

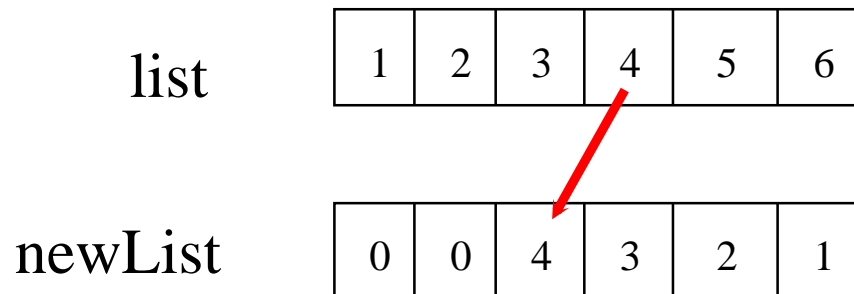
0	0	0	3	2	1
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

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

**i = 3 and j = 2**  
**Assign list[i] to newList[j]**



# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

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

After this, i becomes 4  
and j becomes 1

list

1	2	3	4	5	6
---	---	---	---	---	---

newList

0	0	4	3	2	1
---	---	---	---	---	---



# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

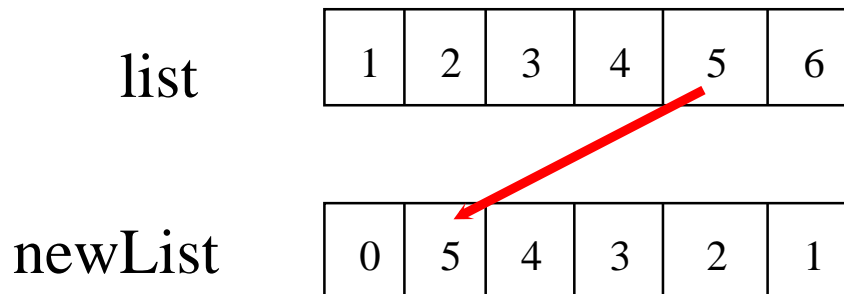
```
for (int i = 0, j = size - 1; i < size; i++, j--)
```

```
{
```

```
    newList[j] = list[i];
```

```
}
```

**i = 4 and j = 1**  
Assign list[i] to newList[j]



# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

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

**i (=5) is still less than 6**

list

1	2	3	4	5	6
---	---	---	---	---	---

newList

0	5	4	3	2	1
---	---	---	---	---	---

# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

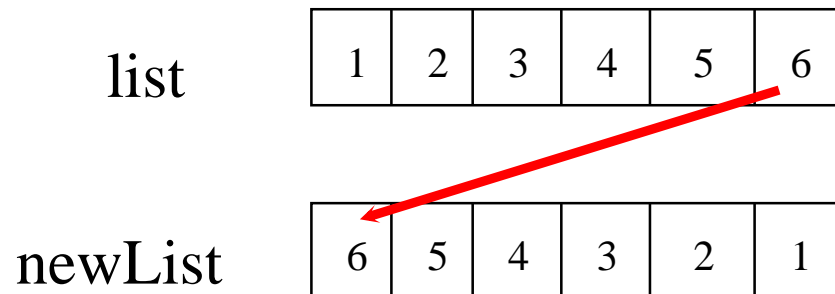
```
for (int i = 0, j = size - 1; i < size; i++, j--)
```

```
{
```

```
    newList[j] = list[i];
```

```
}
```

**i = 5 and j = 0**  
Assign list[i] to newList[j]



# Trace the reverse Function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Trace the reverse function, cont.

```
int list1[] = {1, 2, 3, 4, 5, 6};  
reverse(list1, list2);
```

```
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
---	---	---	---	---	---

# Example 1 - Using An Integer Array

```
/*define variables for storing 10 students' mark*/
int marks[10], sum=0, average;
int i;

/*input marks of student*/
for (i=0; i<10; i++)
    cin >> marks[i];

/*print the marks and sum all the marks*/
cout << "The mark of the students are:";
for (i=0; i<10; i++) {
    cout << marks[i];
    sum = sum + marks[i];
}

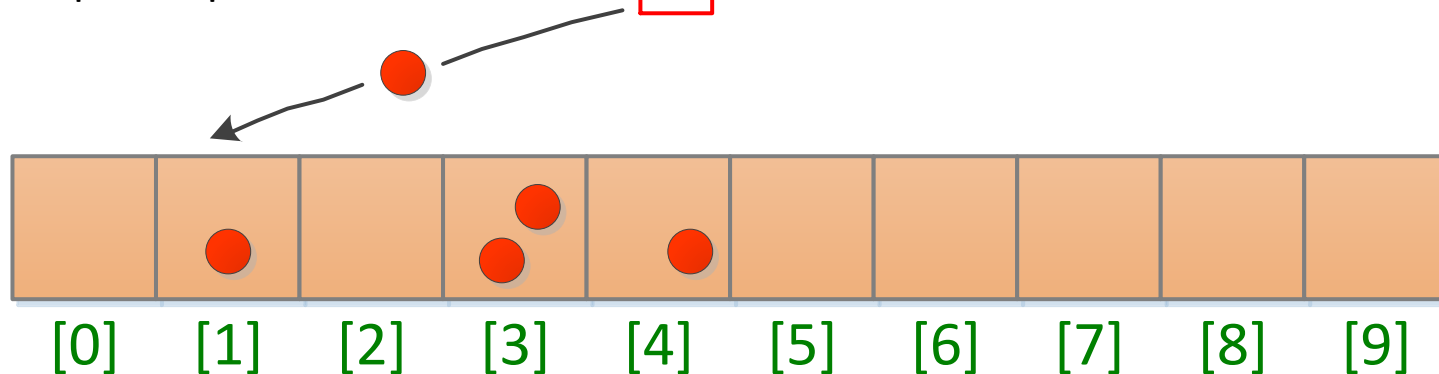
/*compute and print the average*/
average = sum/10;
cout << "Average mark=" << average << endl;
```



# Example 2 - Counting Digits

- Input a sequence of digits  $\{0, 1, 2, \dots, 9\}$ , which is terminated by  $-1$ .
- Count the frequency of occurrence of each digit.
- Use an integer array `count` of 10 elements.
  - `count[i]` stores the frequency of occurrence of digit  $i$ .

Input sequence: 3 4 1 3 1 3 -1



# The Program - Buggy Version

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

void main(){
    int count[10]; //frequency of occurrence of digits
    int digit;      //input digit.
    int i;          //loop counter

    //read the digits
    do {
        cin >> digit;
        if (digit>=0 && digit<=9)
            count[digit]++;
    } while(digit != -1); //stop if the input number is -1

    //print the frequency
    for (i=0; i<10; i++){
        cout << "Frequency of " << i << " is " << count[i] << endl;
    }
}
```

# The Possible Actual Output - Incorrect!

3 4 1 3 1 3 -1

Frequency of 0 is 2089878893

Frequency of 1 is 2088886165

Frequency of 2 is 1376256

Frequency of 3 is 3

Frequency of 4 is 1394145

Frequency of 5 is 1245072

Frequency of 6 is 4203110

Frequency of 7 is 1394144

Frequency of 8 is 0

Frequency of 9 is 1310720

# Always Good Practice to Initialize Arrays

- Otherwise, the values of the array elements is **unpredictable**.
- A common way to **initialize** an array is to set all the elements to zero.


```
for (i=0; i<10; i++)  
    count[i]=0;
```

# Array\_INITIALIZER

```
int mark [10]={100, 90};
```

- Define an array of 10 elements, set the 1<sup>st</sup> element to 100 and the 2<sup>nd</sup> element to 90.
- We list fewer values than the array size (10). The remaining elements are set to 0 by default.
- To initialize all elements to 0:

```
int mark[10]={0};  
int count[10]={0};
```



```
for (i=0; i<10; i++)  
    count[i]=0;
```

# Buggy Version

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

void main(){
    int count[10]; //frequency of occurrence of digits
    int digit; //input digit
    int i; //loop counter

    //read the digits
    do {
        cin >> digit;
        if (digit>=0 && digit<=9)
            count[digit]++;
    } while(digit != -1); //stop if the input number is -1

    //print the frequency
    for (i=0; i<10;i++){
        cout << "Frequency of " << i << " is " << count[i] << endl;
    }
}
```

# Correct Program

```
#include<iostream>
using namespace std;
void main(){
    int count[10]; //frequency of occurrence of digits
    int digit; //input digit.
    int i; //loop counter

    //initialization
    for (i=0;i<10;i++){
        count[i]=0;
    }

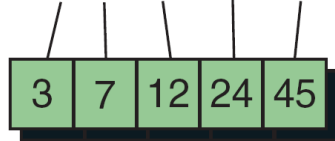
    //read the digits
    do {
        cin >> digit;
        if (digit>=0 && digit<=9)
            count[digit]++;
    } while(digit != -1); //stop if the input number is -1

    //print the frequency
    for (i=0; i<10;i++){
        cout << "Frequency of " << i << " is " << count[i] << endl;
    }
}
```

# Array Initialization Summary

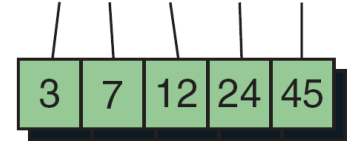
(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```



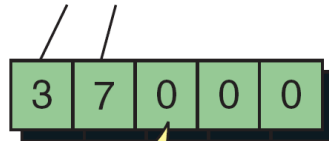
(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```



(c) Partial Initialization

```
int numbers[5] = {3,7};
```



The rest are filled with 0s

(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



All filled with 0s

- Only fixed-length arrays can be initialized when they are defined.
- **Variable length arrays** must be initialized by inputting or assigning the values.



# Example 3: Comparing Two Arrays

- We have two integer arrays, each with 5 elements:  
`int array1[5]={10, 5, 3, 5, 1};`  
`int array2[5];`
- The user input the values of `array2`.
- Compare whether all the elements in `array1` and `array2` are the same.

# Array Equality

- Note that you have to compare array element **one by one**.
- The following code generates **incorrect** results:

```
if (array1 == array2)
    cout << "The arrays are equal ";
else
    cout << "The arrays are not equal ";
```

# The Program

```
#include <iostream>
using namespace std;
void main(){
    int array1[5]={10, 5, 3, 5, 1};
    int array2[5];
    int i;
    bool arrayEqual=true;

    cout << "Input 5 numbers\n";
    for (i=0;i<5;i++){
        cin >> array2[i];

        for (i=0; i<5 && arrayEqual; i++){
            if (array1[i]!=array2[i]){
                arrayEqual=false;
            }
        }

        if (arrayEqual)
            cout << "The arrays are equal";
        else
            cout << "The arrays are not equal";
    }
}
```

Input 5 numbers

10 5 3 5 1

The arrays are equal

Input 5 numbers

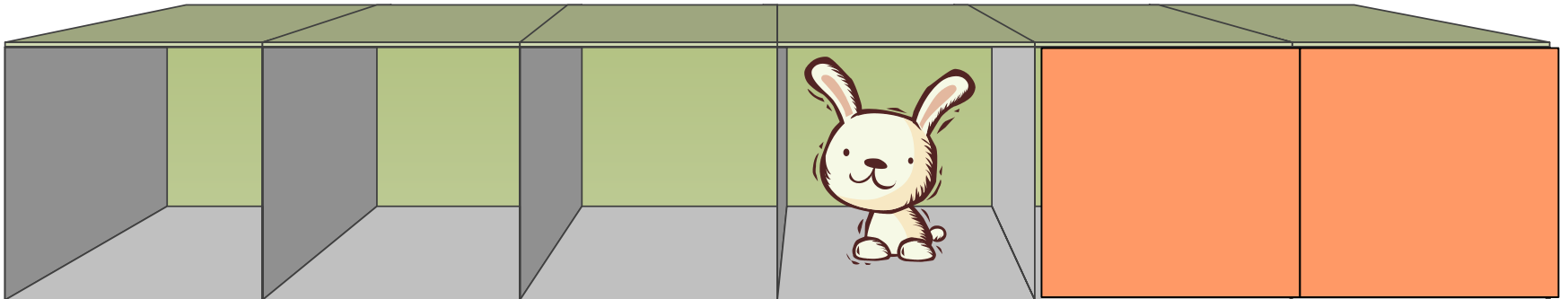
10 4 3 5 2

The arrays are not equal

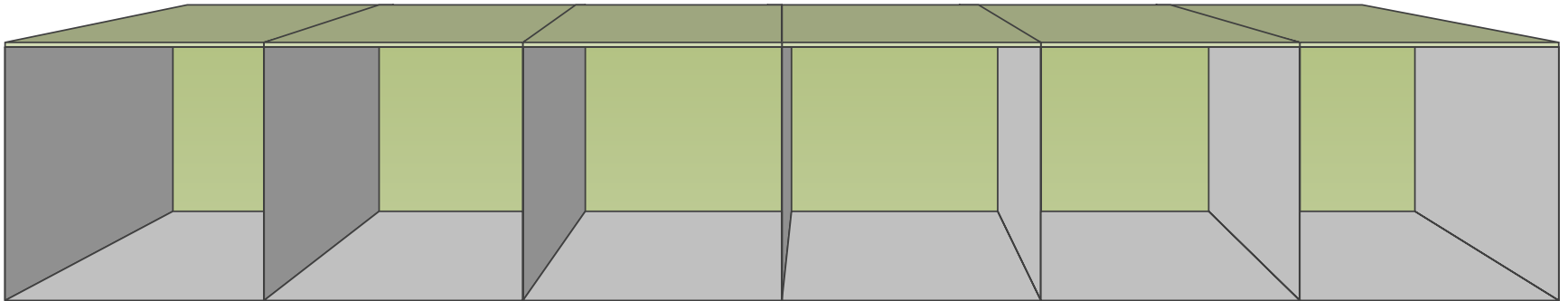
# Example 4: Searching

- Read 10 numbers from the user and store them in an array.
- User input another number  $x$ .
- The program checks if  $x$  is an element of the array
  - If yes, output the index of the element.
  - If no, output  $-1$ .

# Searching for the Rabbit (Case I)



# Searching for the Rabbit (Case II)

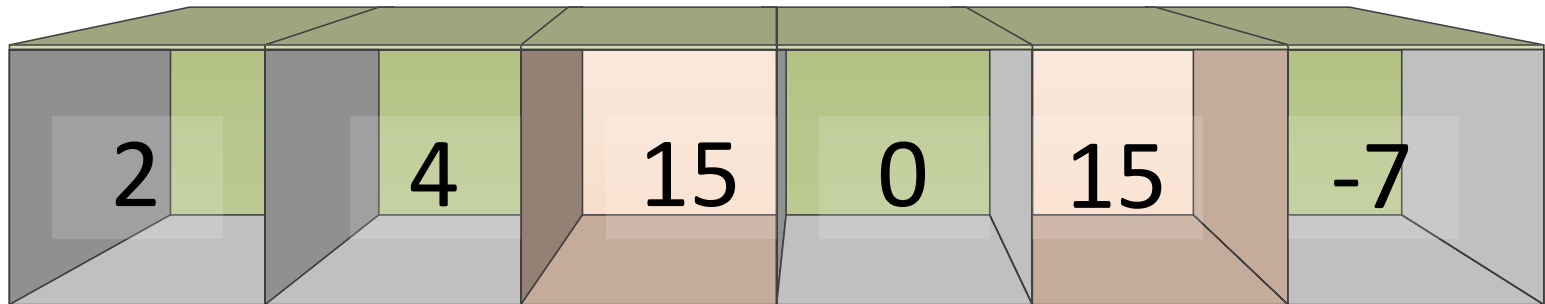


# Searching for $x=15$ (Case 1)

Suppose  $N=6$

$i=1$

$a[i] \neq x$



$i=0$

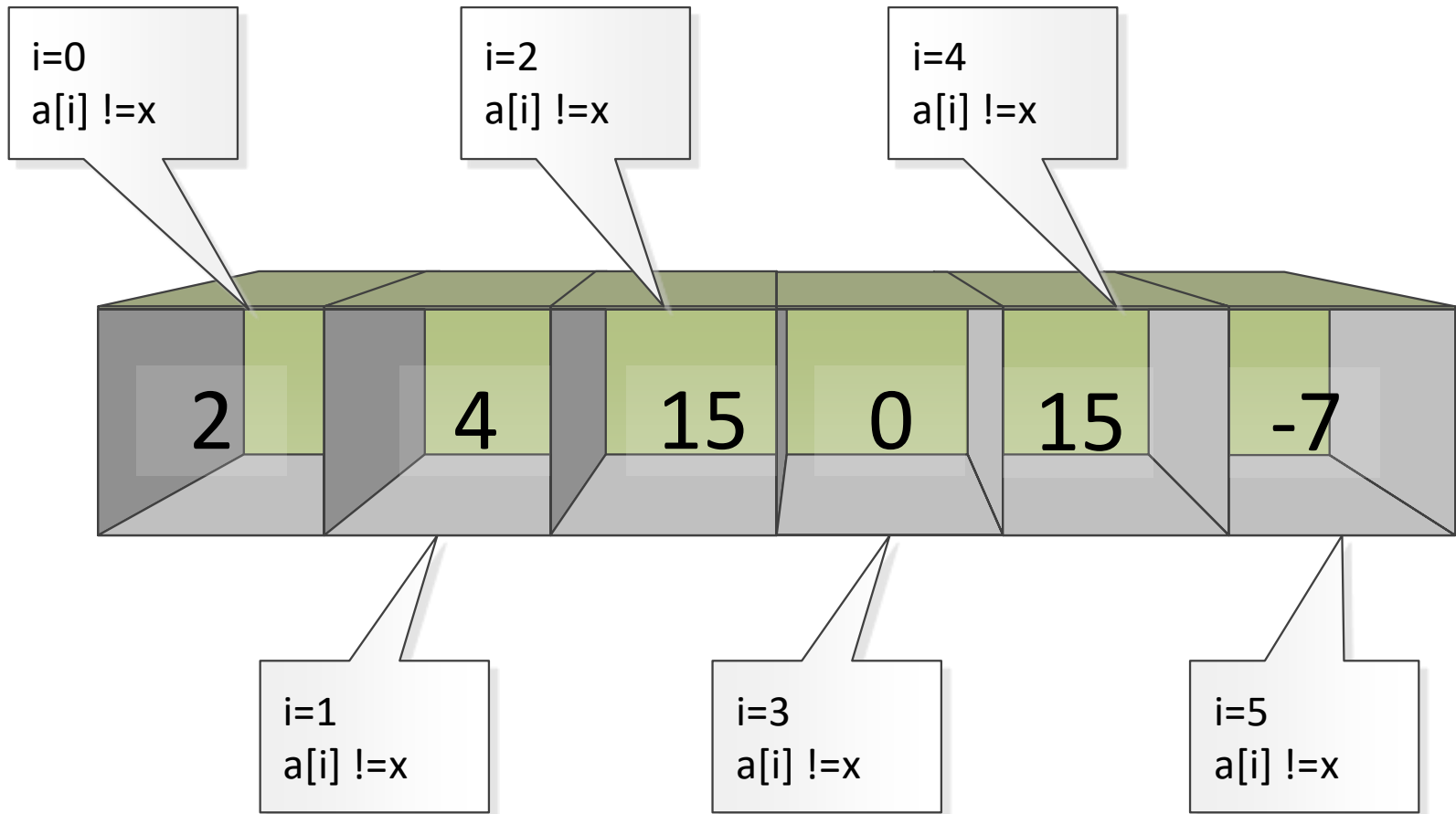
$a[i] \neq x$

$i=2$

$a[i] == x$

Output  $i=2$   
break out of the loop

# Searching for $x=8$ (Case 2)



Output -1



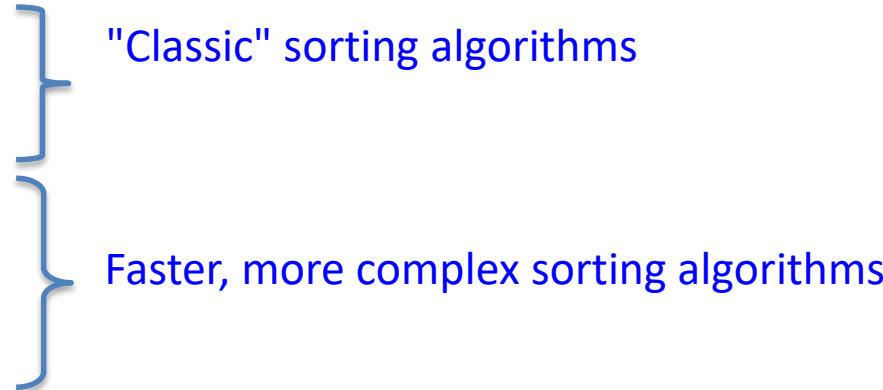
# The Program

```
#include <iostream>
using namespace std;
#define N 10

void main() {
    int a[N], i, x, position;

    for (i=0; i<N; i++)
        cin >> a[i];
    cout << "Input your target: ";
    cin >> x;
    for (i=0, position=-1; i<N; i++){
        if (a[i]==x) {
            position=i;
            break;
        }
    }
    if (position == -1)
        cout << "Target not found!\n";
    else
        cout << "Target found at position" << position << endl;
}
```

# Example 5: Sorting

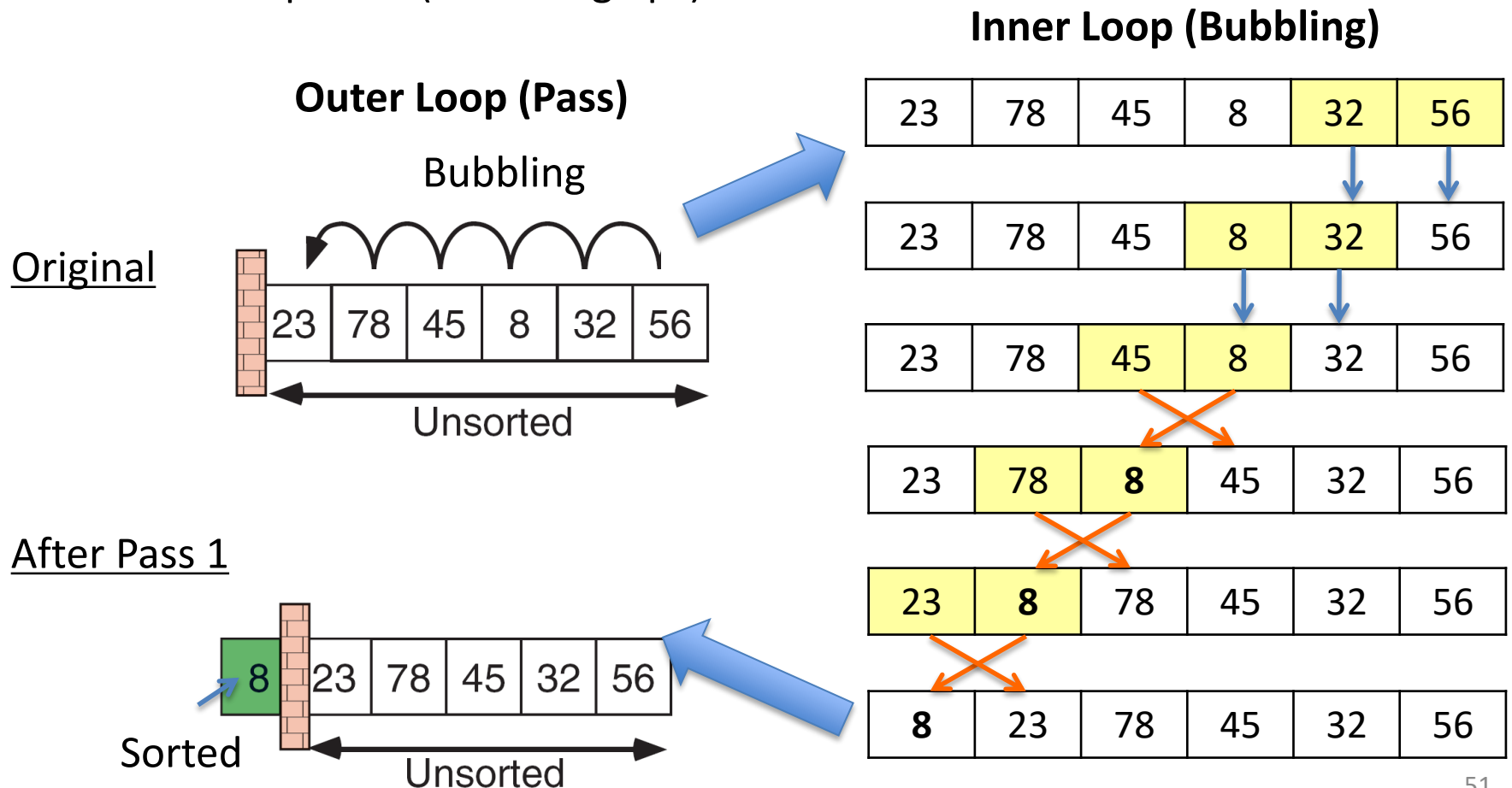
- One of the most common applications in CS is sorting
  - arranging data by their values:  $\{1, 5, 3, 2\} \rightarrow \{1, 2, 3, 5\}$ .
- There are many algorithms for sorting.
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
  - Quick Sort
  - Quicker Sort
  - Merge Sort
  - Heap Sort

"Classic" sorting algorithms

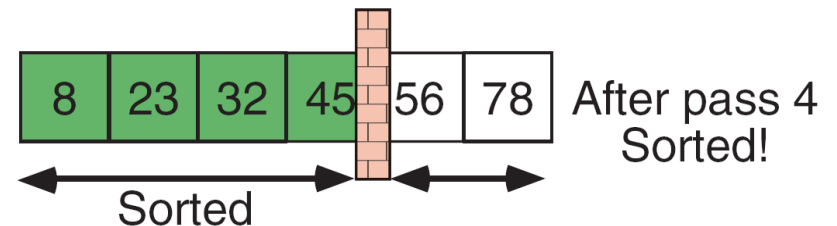
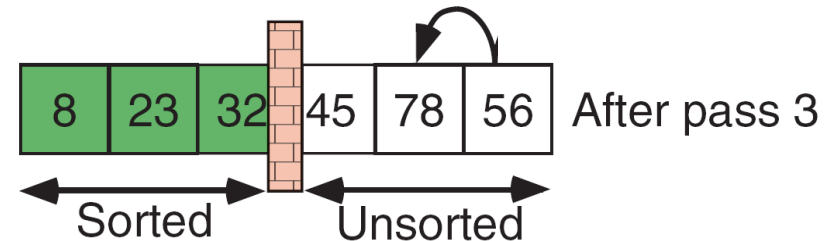
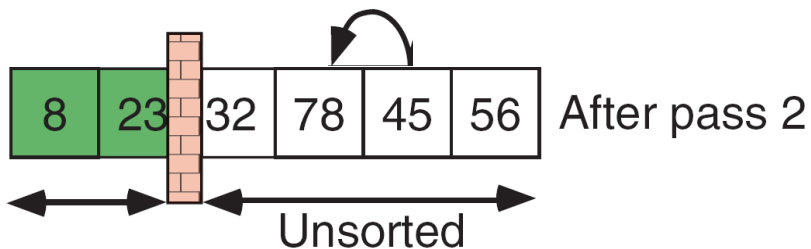
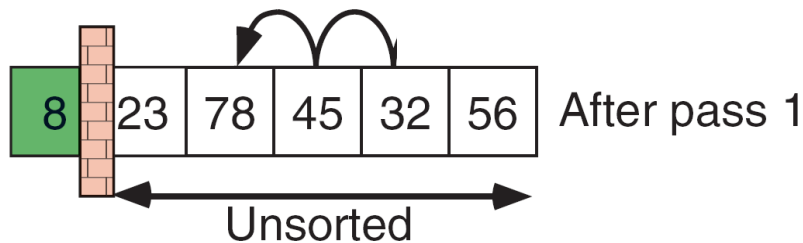
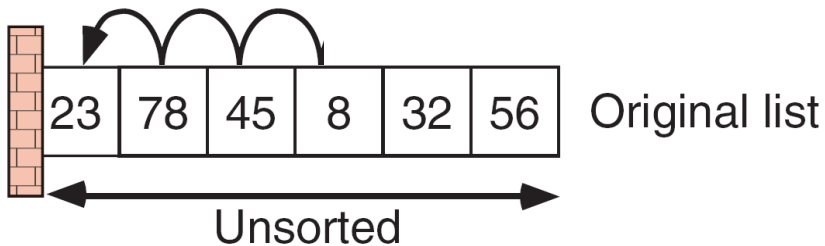
Faster, more complex sorting algorithms
- Based on iteratively swapping two elements in the array so that eventually the array is ordered.
  - The algorithms differ in how they choose the two elements.

# Bubble Sort

- The array is divided into two parts: sorted and unsorted.
- In each pass, start at the end, and swap neighboring elements if they are out of sequence ("bubbling up").



# Bubble Sort



bubble-sort dance: <http://youtu.be/lyZQPjUT5B4>

insert-sort dance: <http://youtu.be/ROaIU379I3U>

# Bubble Sort

Demo!

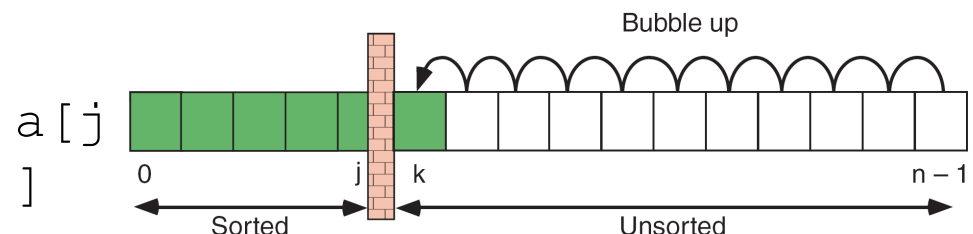
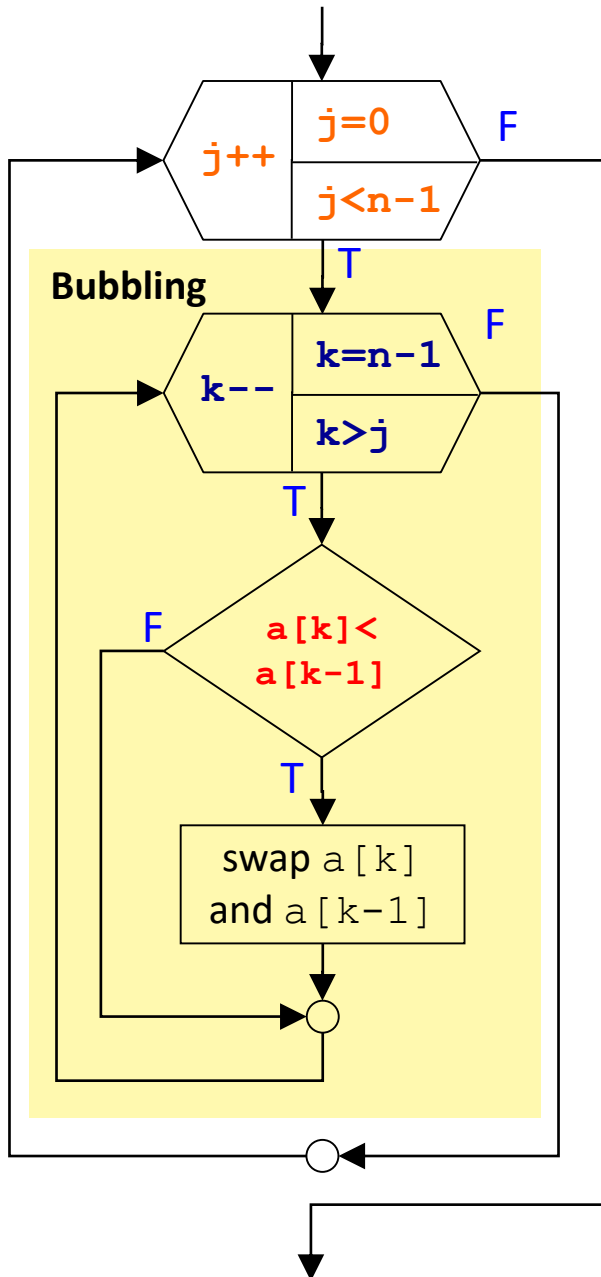
```
#include <iostream>
using namespace std;
#define n 10
void main() {
    int a[n], j, k, tmp;
```

```
    cout << "Input" << n << " numbers: ";
    for (j=0; j<n; j++)
        cin >> a[j];
```

```
    for(j=0; j<n-1; j++)    // outer loop
        for(k=n-1; k>j; k--) // bubbling
            if (a[k]<a[k-1]) {
                tmp      = a[k];    // swap neighbors
                a[k]      = a[k-1];
                a[k-1]    = tmp;
            }
```

```
    cout << "Sorted: ";
    for(j=0; j<n; j++)
        cout << a[j];
```

```
}
```

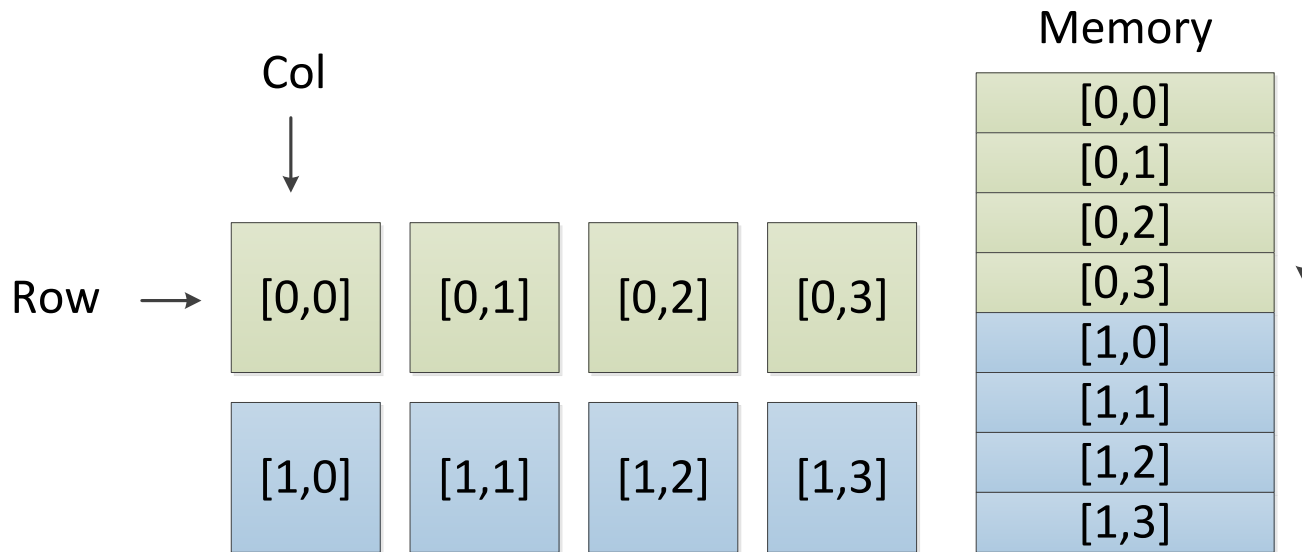


# Multi-Dimensional Array

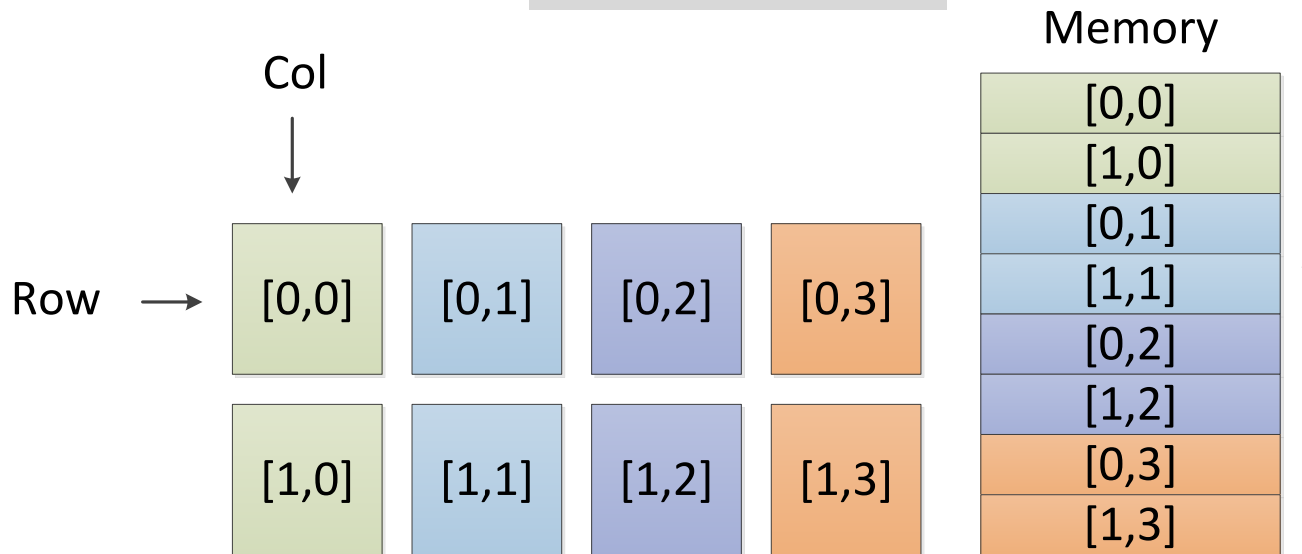
- Multi-dimensional array refers to an array with more than one index. It is a logical representation. On physical storage, the multi-dimensional array is same as single dimensional array (*stored contiguously* in memory space).
- To define a two-dimensional array, we specify the size of each dimension as follows:

```
int page [30][100]; // [row] [column]
```

- In C++, the array will be stored in the “**row-major**” order, i.e. first block of memory will be used to store page [0][0] to page [0][99], the next block for page [1][0] to page [1][99]



Row-Major



Col-Major

# Multi-Dimensional Array

- To access an element of the array, we specify an index for each dimension:

```
cin >> page [i][j]; // [row] [column]
```

- The above statement will input an integer into row *i* and column *j* of the array.



# BMI Program

```
#include <iostream>
using namespace std;
#define N 10

void main() {
    float data[N][2]; // N records, each record holds two values (weight and height)
    int i, count , position;
    do {
        cout << "Number of record (max: 10):";
        cin >> count;
        if (count >10)
            cout << "Maximum number of record is 10, please enter again:" << endl;
    }while (count >10);
    for (i=0; i<count; i++){
        cout << "Weight(kg) Height(m) (" << i+1 << "):";
        cin >> data[i][0];
        cin >> data[i][1];
    }
    for (i=0; i<count; i++){
        cout << "BMI for " << i+ 1 << ": " << data[i][0]/(data[i][1]*data[i][1]);
        cout << endl;
    }
}
```

# Summary

- Array is a sequence of variables of the **same** data type.
- Array elements are **indexed** and can be **accessed** by using subscripts, e.g. `arrayName[1]`, `arrayName[4]`.
- Array elements are **stored contiguously** in memory space.
- Array declaration, initialization, searching, and sorting.
- Array can be multi-dimensional, e.g. 1D , 2D.