CS2311 Computer Programming

LT6: Arrays

Example 1

- Input the marks for 10 students
- Store the marks in variables
- Compute the average marks
- Print the marks of the students and the average

```
100 30 44 66 50 60 80 75 80 100
The mark of the students are: 100, 30, 44, 66, 50, 60, 80, 75, 80, 100
Average mark=68
```

The Program

```
// define variables for storing 10 students' mark
int mark1, mark2, mark3, mark4, mark5,
    mark6, mark7, mark8, mark9, mark10, average;
// input marks of student
cin >> mark1 >> mark2 >> mark3 >> mark4 >>
    mark5 >> mark6 >> mark7 >> mark8 >> mark9 >> mark10;
// print the marks
cout << "The mark of the students are: " << mark1 << mark2 <<
    mark3 << mark4 << mark5 << mark6 << mark7 << mark8 << mark9
    << mark10 << endl;
double average = (mark1 + mark2 + mark3 + mark4 + mark5 +
    mark6 + mark7 + mark8 + mark9 + mark10) / 10;
cout << "Average mark = " << average << endl;
```

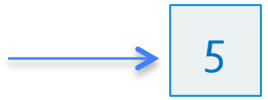
Is it easy to extend the program to handle more students?

```
100 30 44 66 50 60 80 75 80 100
The mark of the students are: 100, 30, 44, 66, 50, 60, 80, 75, 80, 100
Average mark = 68
```

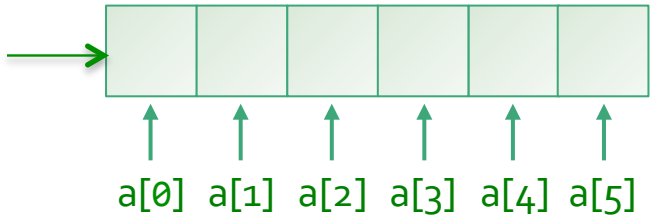
What is an Array?

- Sequence of items of the **same** data type
  - Stored contiguously
  - Can be accessed by index, or subscript

```
int x;
x = 5;
```




```
int a[6];
```




```
a[0] = 5;
a[1] = 7;
a[2] = 2;
```

Outline

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

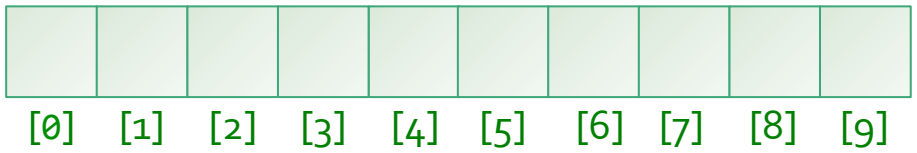
Array Definition

Data type of the array elements

Size of the array

```
int mark[10];
```

Name of the array



There are **ten** elements in this array

`mark[0], mark[1], ……, mark[9]`

The i^{th} array element is `mark[i-1]`.

The range of the subscript **i** is from **0** to **array_size-1**

The location `mark[10]` is invalid. Array out of bound!

Array definition

- Array size
  - **Constant**

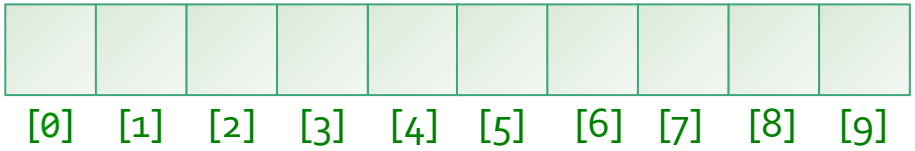
```
int mark[50*50];

int n = 10;
int mark[n];

const int n = 10;
int mark[n];
```

Storing Values in Array Elements

```
int mark[10];
```



- Suppose the mark for the *first* student is *30*. We can use the notation

```
mark[0] = 30;
```

- Reading the marks of the second student

```
cin >> mark[1];
```

- Reading the marks for 10 students from console

```
for (int i=0; i<10; i++)
  cin >> mark[i];
```

  - alternatively,

```
for (int i=1; i<=10; i++)
  cin >> mark[i-1];
```

Retrieving Values From An Array

- Print the mark of the *second* student

```
cout << mark[1];
```

- Print and ***sum*** the marks of all students

```
for (int i=0; i<10; i++) {
  cout << mark[i];
  sum += mark[i];
}
```

Example 1

(using an integer array with 10 elements)

```
// define variables for storing 10 students' marks
int marks[10], sum=0, average;
// input marks of student
for (int i=0;i<10;i++)
   cin >> marks[i];
// print the marks
cout << "The mark of the students are:";
for (int i=0;i<10;i++) {
   cout << ' ' << marks[i];
   sum += marks[i];
}
cout << endl;
// compute and print the average
average = sum/10;
cout << "Average mark = " << average << endl;
```

C++ Macro: #define

- #define is a C++ ***predefined macro*** keyword.
  - It globally replaces all occurrences of ***A*** to ***B*** in the ENTIRE source code listing (all .cpp and all .h files).

```
#define  A  B
```

  - examples

```
#define N 100
#define SIZE 10
```

  - NOT suggested

"global" scope */ more prone to conflicting usages, which can produce hard-to-resolve compilation issues and unexpected run-time results*

- Better to use … for type checking
  - `const int MaxN = 100;`
  - `const int Size = 10;`

#define Example

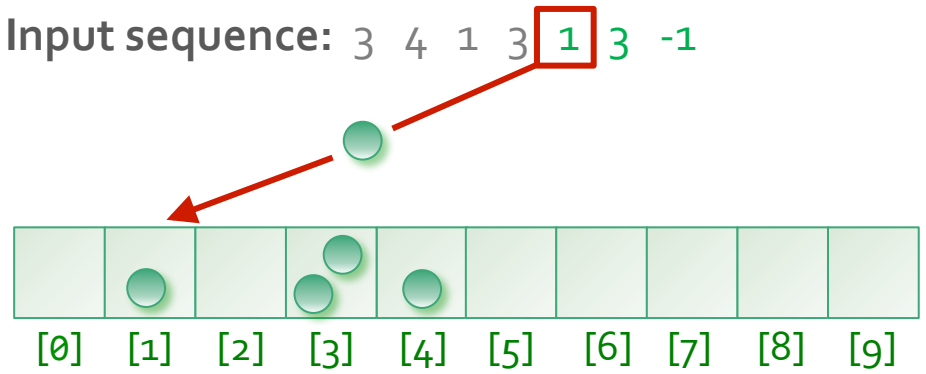
```
#include <iostream>
using namespace std;
const int N = 10;
int main() {
   int marks[N], sum=0, average;

   for (int i=0;i<N;i++)
      cin >> marks[i];

   cout << "The mark of the students are:";
   for (int i=0;i<N;i++) {
      cout << ' ' << marks[i];
      sum = sum + marks[i];
   }
   cout << endl;
   average = sum/N;
   cout << "Average mark = " << average << endl;
   return 0;
}
```

Example 2: counting digits

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



The Program: buggy version

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

int main() {
    int count[10];          //number of occurrence of digits
    int digit;              //input digit
    //read the digits
    do {
      cin >> digit;
      if (digit >= 0 && digit <= 9) //necessary to avoid out-of-bound
         count[digit]++;
    } while (digit != -1);  //stop if the input number is -1
    //print the occurrences
    for (int i=0; i<10; i++)
      cout << "Frequency of " << i << " is " << count[i] << endl;

   return 0;
}
```

The Actual Output (Incorrect!)

For some compilers like VS 2019, this program won't run;
for others like g++, this will run with incorrect output

```
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
```

It's a Good Practice to **Initialize** Arrays

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

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

Array Initializer

```
int mark[2] = {100, 90};

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

- Define an array of 10 elements, set the 1st element to **100** and the 2nd element to **90**
  - If 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};
```

Correct Program for Example 2

```
#include <iostream>
using namespace std;
int main() {
   int count[10] = {0}; //number of occurrence of digits, initialised
   int digit;           //input digit
   //read the digits
   do {
     cin >> digit;
     if (digit >= 0 && digit <= 9) //necessary to avoid out-of-bound
        count[digit]++;
   } while (digit != -1);    //stop if the input number is -1
   //print the occurrences
   for (int i=0; i<10; i++)
     cout << "Frequency of " << i << " is " << count[i] << endl;
  return 0;
}
```

Array Initializer

- **char** array

```
char univ[5] = {'C', 'i', 't', 'y', 'U'};

char univ[] = "CityU";

  or

char *univ = "CityU";
```

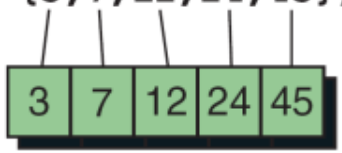
Array Initialization Summary

(a) Basic Initialization

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

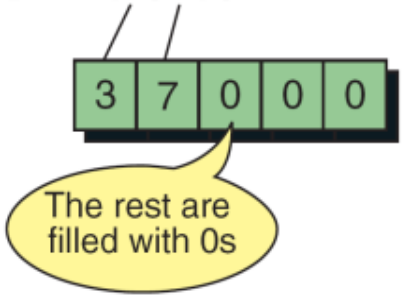
(b) Initialization without Size

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



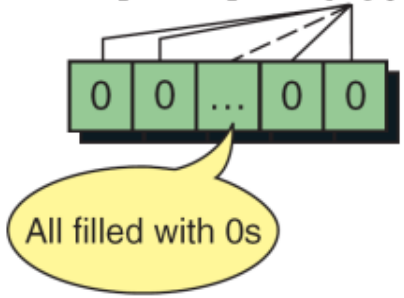
(c) Partial Initialization

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



(d) Initialization to All Zeros

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



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

Summary of Array Declaration and Access

Type	Variable	Array	Variable Access	Array Access
int	`int x;`	`int x[20];`	`x = 1;`	`x[0]=1`
float	`float x;`	`float x[10];`	`x = 3.4;`	`x[0]=3.4;` `x[9]=1.2;`
double	`double x;`	`double x[20];`	`x = 0.7;`	`x[0]=0.7;` `x[3]=3.4;`
char	`char x;`	`char x[5];`	`x = 'a';`	`x[0]='c';` `X[1]='s';`

```
char x[] = "hello";
```

Array declaration and initialization

Example 3: Comparing 2 Arrays

- We have two integers arrays, each with 5 elements

```
int array1[5] = {10, 5, 3, 5, 1};
int array2[5];
```

- The user inputs the values of **array2**
- Compare whether all of 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

```
int main() {
   int array1[5] = {10, 5, 3, 5, 1};
   int array2[5];
   bool arrayEqual = true;
   cout << "Input 5 numbers" << endl;
   for (int i=0; i<5; i++)
      cin >> array2[i];
   for (int i=0; i<5 && arrayEqual; i++)
      if (array1[i] != array2[i])
         arrayEqual = false;
   if (arrayEqual)
      cout << "The arrays are equal." << endl;
   else
      cout << "The arrays are not equal." << endl;
   return 0;
}
```

```
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.
```

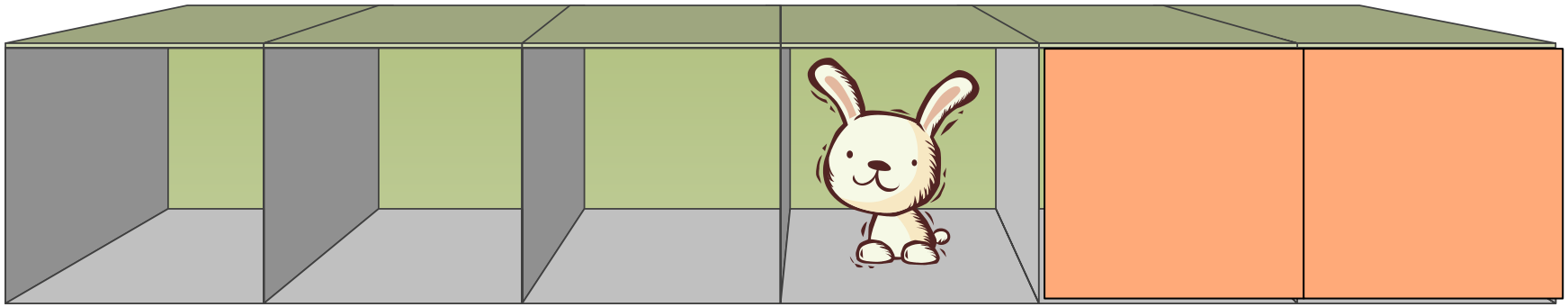
```
arrayEqual = array1[i] == array2[i];
```

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**

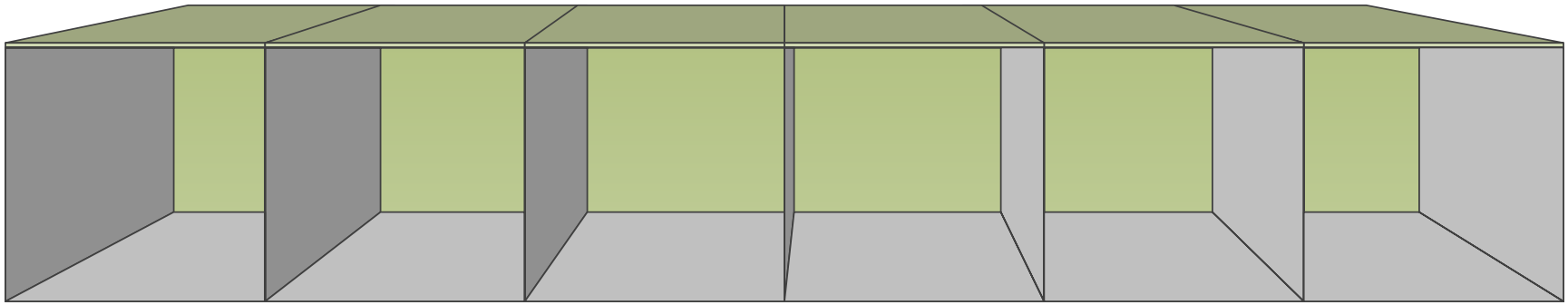
Searching for the Rabbit (Case I)

Search sequentially



If found, skip the rest

Searching for the Rabbit (Case II)

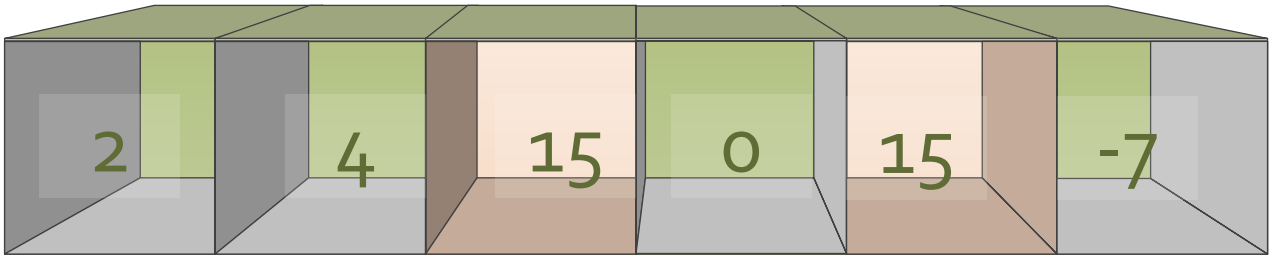


Searching for x=15 (Case 1)

Suppose N = 6

i=1

a[i] != x



i=0

a[i] != x

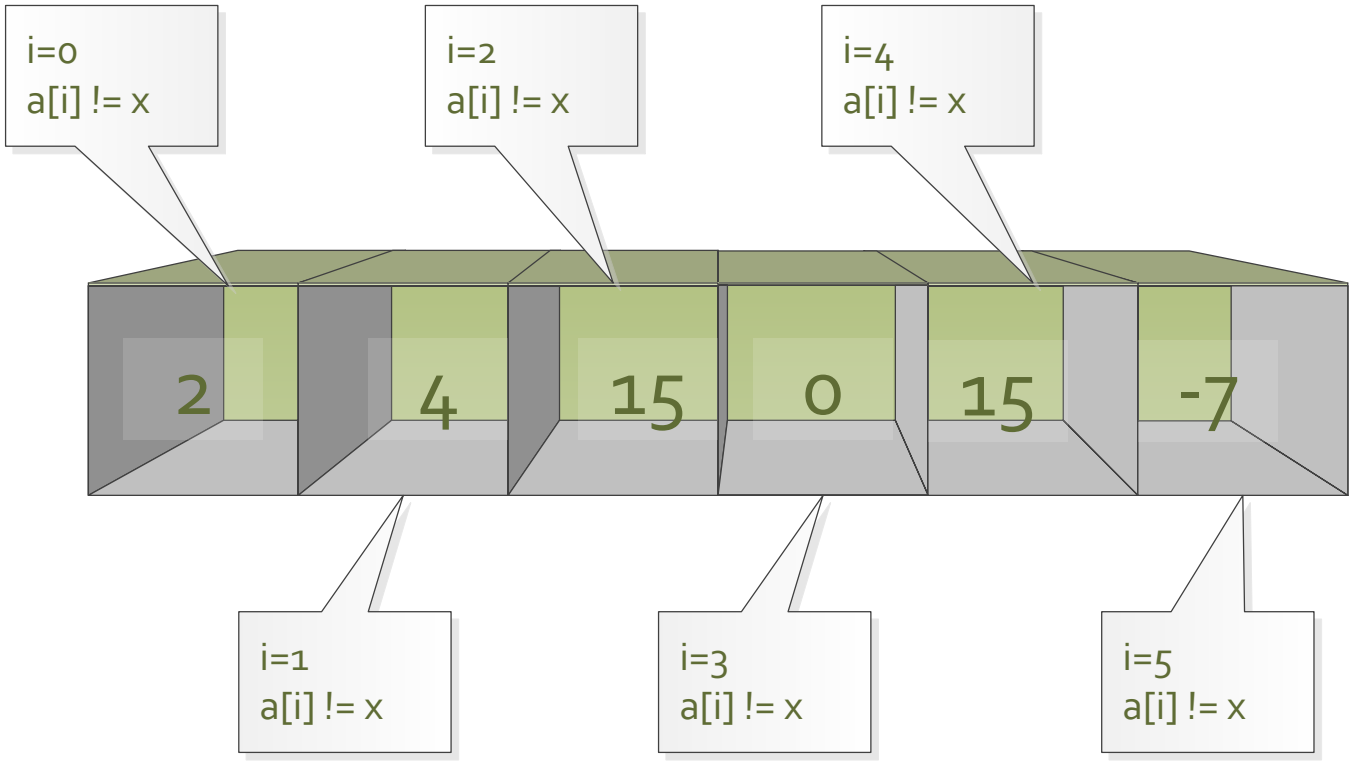
i=2

a[i] == x

Output i = 2

break out of the loop

Searching for x=8 (Case 2)



Output -1

The Program

```
int main() {
   const int N = 6;
   int a[N], x, position = -1;

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

```
bool found = false;
for (int i=0; i<N && !found; i++)
   if (a[i] == x) {
      position = i;
      found = true;
   }
```

```
bool found = false;
for (int i=0; i<N && !found; i++)
   if (found = a[i] == x)
      position = i;
```

Example 5: Sorting

- One of the most common applications is **sorting**
  - arranging data by their values: {1, 5, 3, 2} → {1, 2, 3, 5}
- There are many algorithms for sorting
  - **Selection Sort**
  - **Bubble Sort**
  - **Insertion Sort**

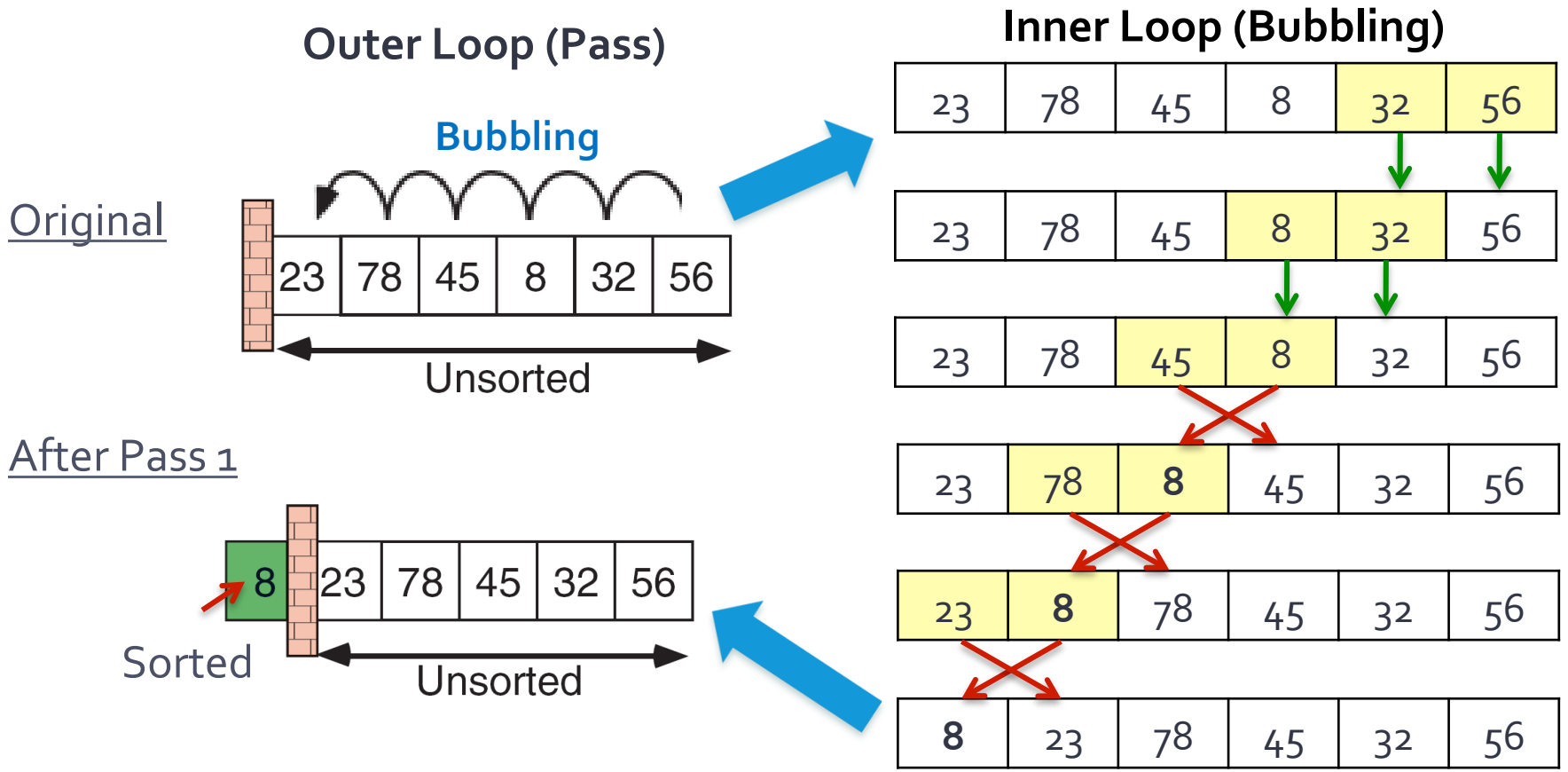
  } "Classic" sorting algorithms

  - **Quick Sort**
  - **Merge Sort**
  - **Heap Sort**

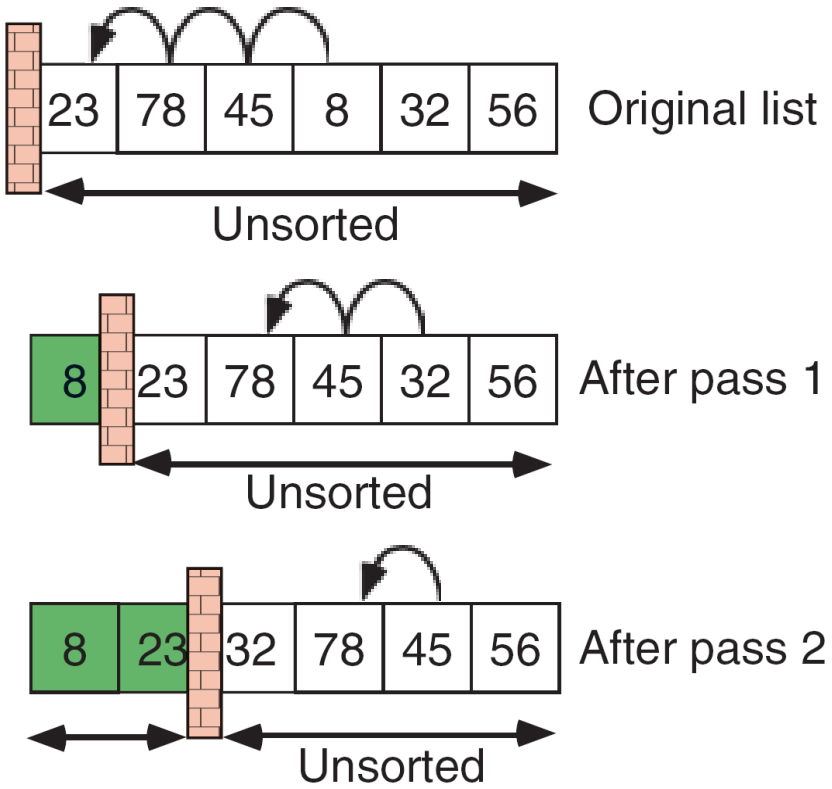
  } Faster, more complex sorting algorithms
- Based on ***iteratively*** swapping two elements in the array so that eventually the array is ordered.
  - The algorithms differ in how they choose the two elements

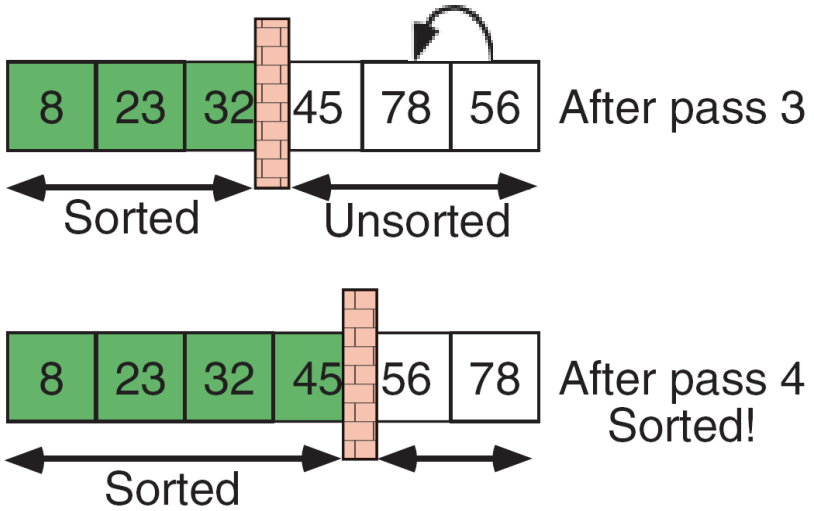
Bubble Sort

- In each pass, start at the end, and swap neighboring elements if they are out of sequence ("bubbling up").
- After i passes, the first i elements are sorted.



Bubble Sort



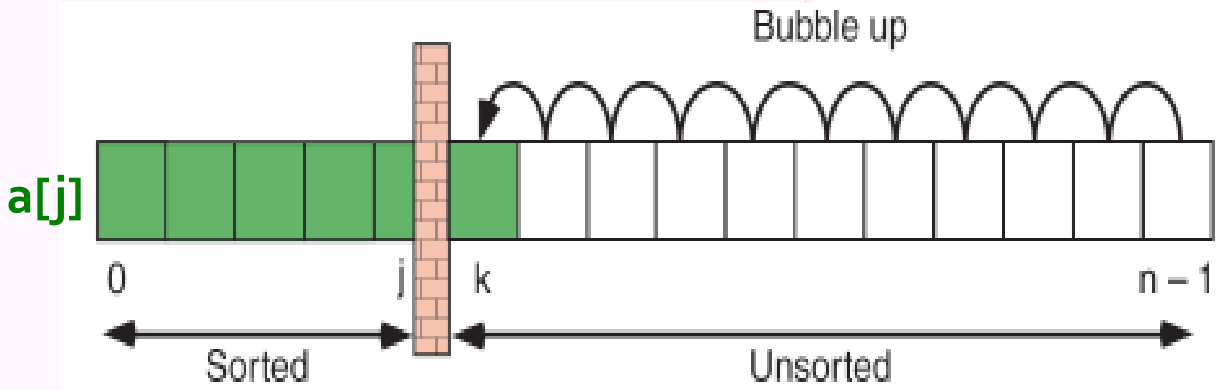


bubble-sort dance: http://youtu.be/lyZQPjUT5B4

insert-sort dance: http://youtu.be/ROalU379l3U

Bubble Sort

```
int main() {
  const int n = 10;
  int a[n], tmp;
  cout <<"Input " << n <<" numbers: ";
  for (int i=0; i<n; i++)
      cin >> a[i];
  for (int j=0; j<n-1; j++) // outer loop
      for (int 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 (int i=0; i<n; i++)
      cout << a[i] << ' ';
  cout << endl;
  return 0;
}
```



Multi-dimensional Array

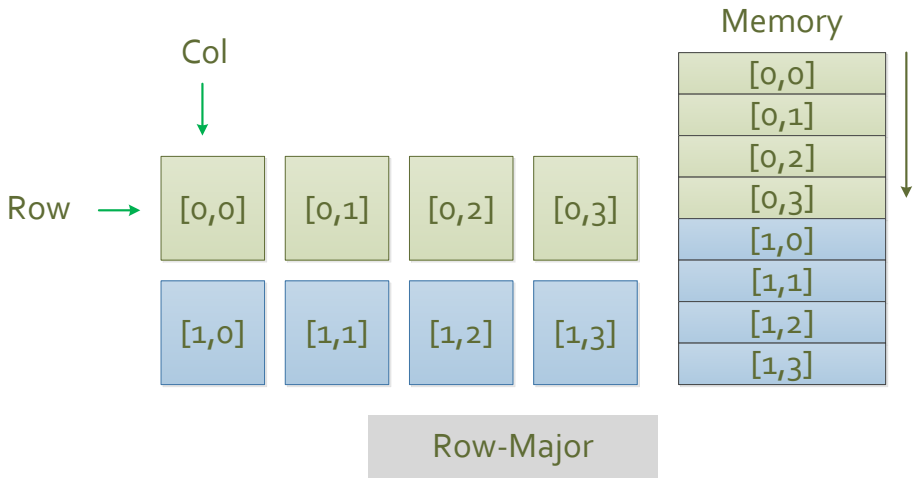
- Multi-dimensional array refers to an array with more than one index.
  - It is a ***logical*** representation.
  - On physical storage, the multi-dimensional array is same as single dimensional array (*stored contiguously* in memory space)
- To define a two-dimensional array, we specify the *size*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

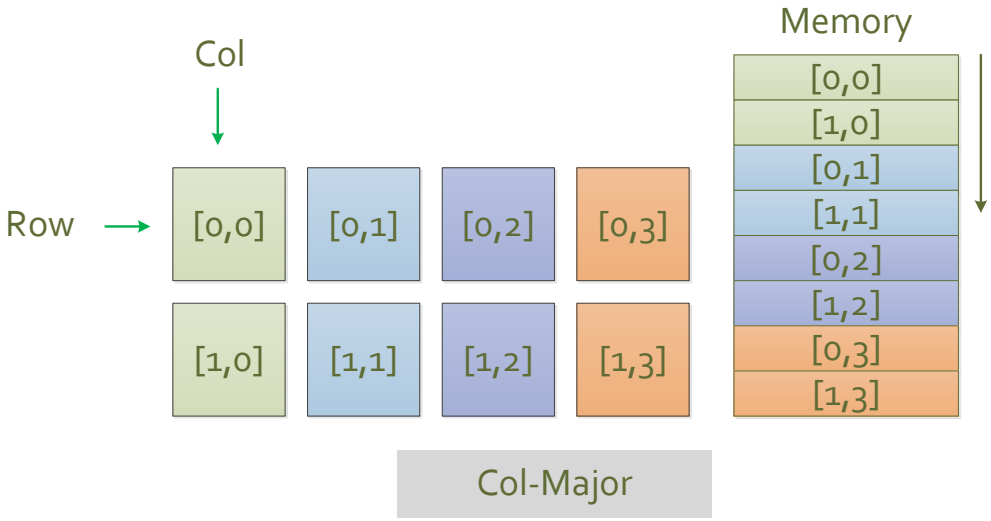
```
int major[2][3]; // [row][column]
```

```
12 34 11
13 56 99
```

- In C++, the array will be stored in the "row-major" order
  - first block of memory will be used to store page [0][0] to page [0][2], the next block for page [1][0] to page [1][2]

Row-major vs Column-major





Two-dimensional Array Initialization

- Assign initial values row by row:

```
int page[2][3] = {{1,2,3},{4,5,6}};
```

- Assign initial values to the elements in the order they are arranged:

```
int page[2][3] = {1,2,3,4,5,6};
```

- Only assign initial values to some elements:

```
int page[2][3] = {{1},{4,5}};
```

```
1  0  0
4  5  0
```

- If all elements are assigned initial values, the length of the first dimension can be left unspecified:

```
int page[][3] = {1,2,3,4,5,6};
```

✗
```
int page[2][] = {1,2,3,4,5,6};
```

Example: swapping row and column

- Swap elements of rows and columns of a two-dimensional array with 2 rows and 3 columns;
- Output the swapped array (please control the precision of each element, so that it contains only **one** digit in the decimal part).

1.2	1.34	2.564
3	4.123	4

1.2	3.0
1.3	4.1
2.6	4.0

Example: swapping row and column

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    // the original array
    double array1[2][3] = {{1.2,1.34,2.564}, {3,4.123,4}};
    // the swapped array
    double array2[3][2];
    // swap elements of row and column
    for (int i=0; i<2; i++)
      for (int j=0; j<3; j++)
          array2[j][i] = array1[i][j];
    // print the swapped array
    for (int i=0; i<3; i++) {
      for (int j=0; j<2; j++)
          cout << fixed << setprecision(1) << array2[i][j] << "  ";
      cout << endl;
   }
   return 0;
}
```

Multi-dimensional Array

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

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

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

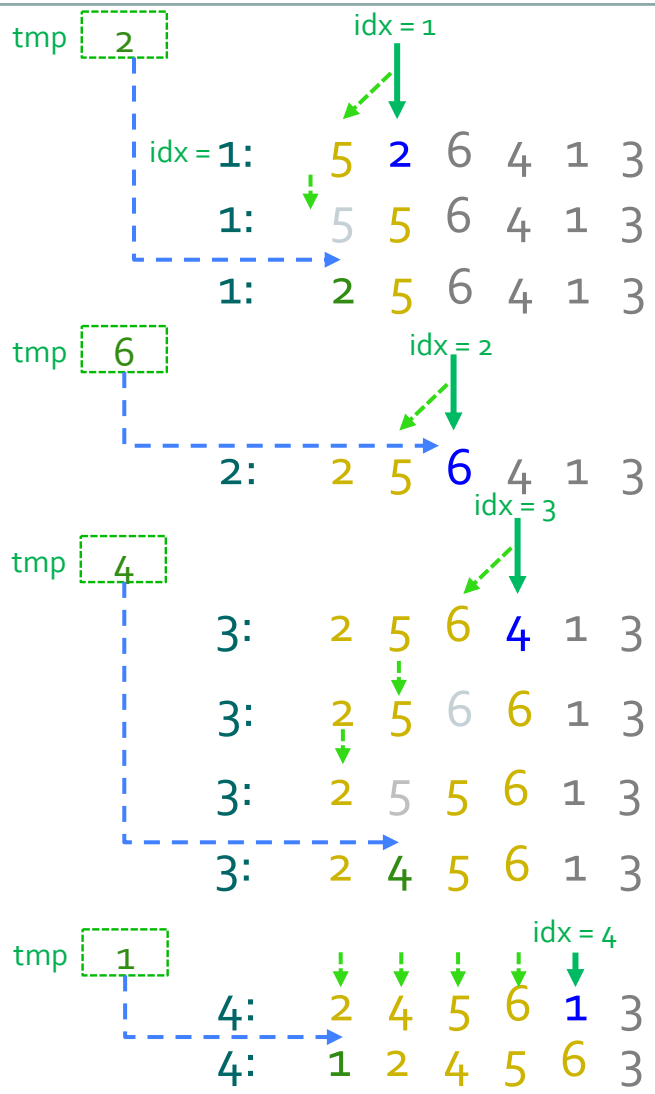
BMI Program

```
int main() {
  const int N=10;
  double data[N][2]; // N records, each holds weight and height
  for (int i=0; i<N; i++) {
     cout << "Weight(kg) Height(m):";
     cin >> data[i][0];
     cin >> data[i][1];
  }
  for (int i=0; i<N; i++) {
     cout << "BMI for " << i+1 << "is :";
     cout << data[i][0] / (data[i][1]*data[i][1]) << endl;
  }
  return 0;
}
```

Summary

- Array is a sequence of variables of the ***same*** data type
- Array elements are indexed and can be accessed by the use of subscripts,
  - e.g. `array_name[1], array_name[4]`
- Array elements are ***stored contiguously*** in memory space
- Array Declaration, Initialization, Searching and Sorting
- Array can be multi-dimensional, i.e. 2D

Insertion Sort



list: 5 2 6 4 1 3

sorted: 1 2 3 4 5 6

```
template <class KeyType>
void SortKeys<KeyType>::InsertSort() {
// assume key[0..maxkeys-1]
        KeyType   tmp;
        for (unsigned int i=1;i<maxkeys;i++) {
                tmp = key[i];
                int j=i;
                while (j > 0 && tmp < key[j-1]) {
                        key[j] = key[j-1];
                        j--;
                }
                key[j] = tmp;
        }
}
```