Lecture 3

Arrays

Topics covered last week

- Unit introduction
- Concept of functions
- Create a function
- Parameter passing in a function
  - Call-by-value
  - Call-by-reference
- Function overloading
- Default arguments

Call-By-Reference vs Call-By-Value

Call-By-Reference vs Call-By-Value

```
#include<iostream>

using namespace std;

void Swap(int a, int b) {
    int temp;

    temp=a;
    a=b;
    b=temp;
}

int main() {
    int a=5;
    int b=8;

    cout << "a=" << a << "; b=" << b <<endl;

    Swap(a,b);

    cout << "a=" << a << "; b=" << b <<endl;
    return 0;
}
```

```
#include<iostream>

using namespace std;

void Swap(int& a, int& b) {
    int temp;

    temp=a;
    a=b;
    b=temp;
}

void main() {
    int a=5;
    int b=8;

    cout << "a=" << a << "; b=" << b <<endl;

    Swap(a,b);

    cout << "a=" << a << "; b=" << b <<endl;
}
```

Function Overloading

C++ is a strongly typed language. Change parameter list of a function will lead to "*another*" function.

- Function *signature*: function name & parameter list
  - Must be "unique" for each function definition
- Function overloading: the same function name with different parameters (different types or different number of parameters) represents different functions.

```
double average(int n1, int n2) {
    return ((n1 + n2) / 2.0);
}
double average(double n1, double n2) {
            return ((n1 + n2) / 2.0);
}
```

Function call: overloading resolution

Given following functions:

1) void f(int n, double m);
2) void f(double n, int m);
3) void f(double n, double m);

These calls:

f(5.3, 4); → Calls 2)
f(4.3, 5.2); → Calls 3)
f(98, 99); → Calls ??

See functioncall.cpp

Default Arguments in a function

When we define a function, it is possible to set some arguments of the function default values.

```
double roundup(double, int);//function prototype
void main() {
  cout << roundup(3.14159, 2); //set roundup to 2 digits
}

double roundup(double, int = 2);//function prototype
void main() {
  cout << roundup(3.14159); //means roundup to 2 digits
}

void main() {
  cout << roundup(3.14159, 4); //set roundup to 4 digits
}
```

Rules for Default Arguments

The two rules of using default parameters are:

1. Default values can only be assigned in the function prototype (declaration).
2. It is not necessary that all arguments of a function have default values but the default values must bee filled from right to left.

```
double roundup(double = 0.0, int = 2);//function prototype
double roundup(double, int = 2);//function prototype
```

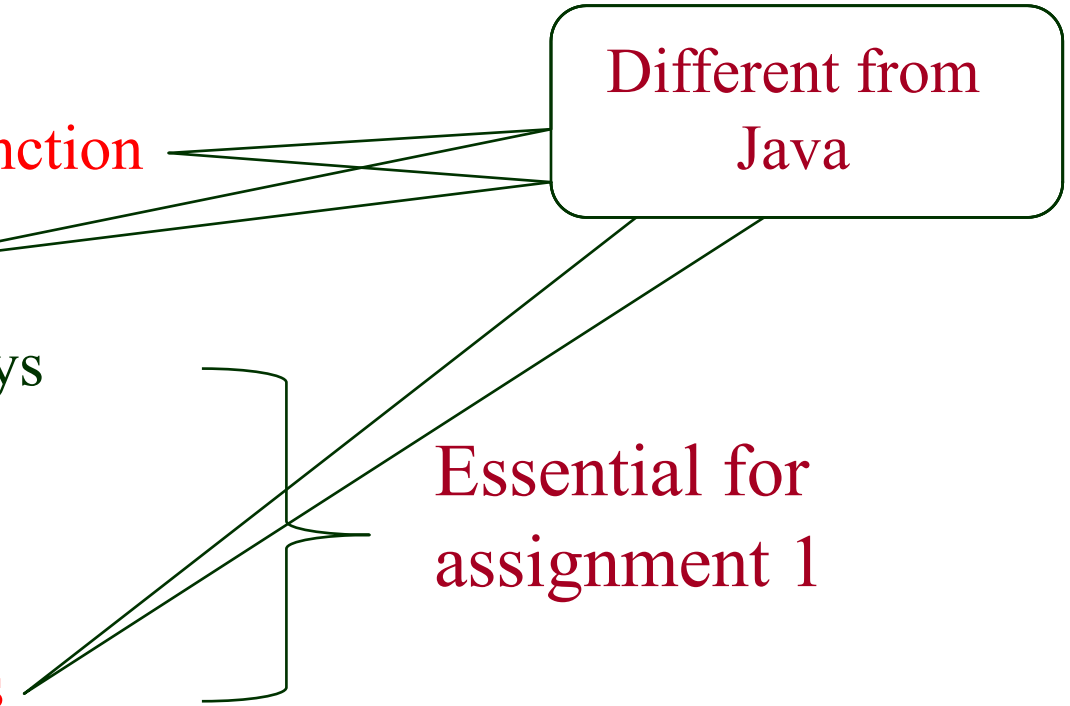
These rules allow C++ not to get 'mixed up' when matching actual arguments with formal arguments.

```
roundup(); roundup(3.14159); roundup(3.14159, 4);
```

Class constructors may take advantage of this capability to initialise data members either explicitly through actual arguments or defaulting to a set of values specified in the function prototype.

Content of this lecture

- Introduction to Arrays
  - Declare arrays
  - Access arrays
- Passing an arrays to a function
- Array initialisation
- Common process in arrays
  - Linear search
  - Find the largest
- Multidimensional Arrays



Introductory example

Task: Design a program which *accepts ten integers, store them and display the total.*

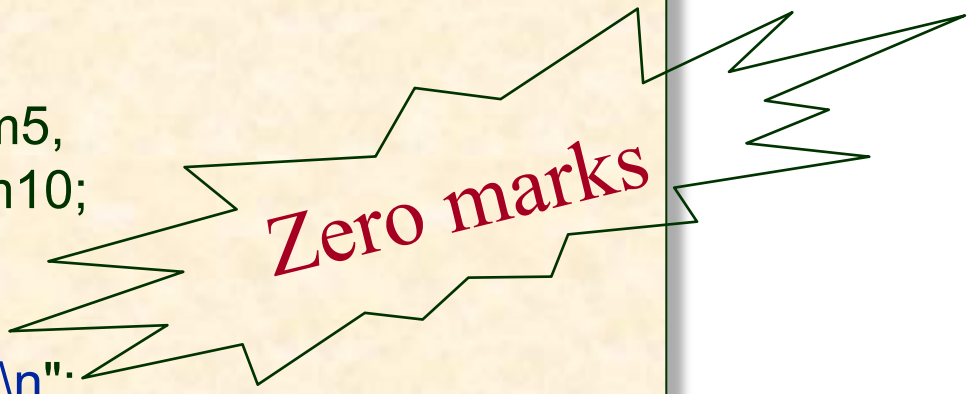
An incorrect solution.
Find the problem.

```
// Bad solution 1
#include <iostream>
using namespace std;
int main() {
    int num, total;
    total = 0;
    cout << "Please input ten integers:\n";
    for ( int i=0; i<10; i++) {
        cin >> num;
        total = total + num;
    } // end for
    cout << "\n The total of the ten integers is " << total;
    return 0;
}  // end main
```

Introductory example

```
// Bad solution 2
#include <iostream>

int main() {
    int num1, num2, num3, num4, num5,
        num6, num7, num8, num9, num10;
    int total;

    cout << "Please input ten integers:\n";
    cin >> num1 >> num2 >> num3 >> num4 >> num5;
    cin >> num6 >> num7 >> num8 >> num9 >> num10;

    total = num1+num2+num3+num4+num5
            +num6+num7+num8+num9+num10;
    cout << "\n The total of the ten integers is " << total;
    return 0;
} //end main
```

A bad solution.
Do not follow.



Introductory example

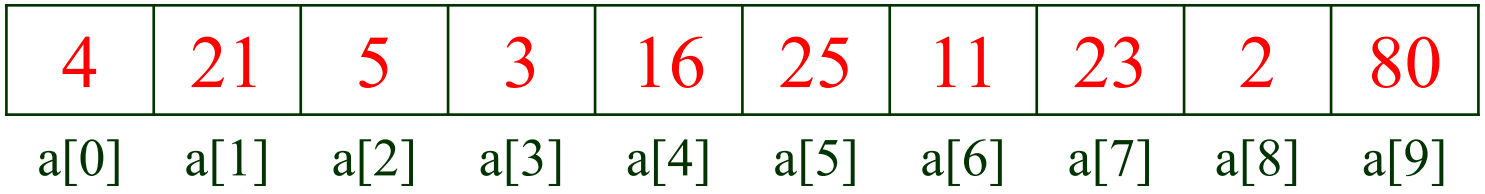
```
// Good solution
#include <iostream>

int main() {
    int num[10];
    int total=0;

    cout << "Please input ten integers:\n";
    for ( int i=0; i< 10; i++) {
        cin >> num[i];
        total = total + num[i];
    } // end for
    cout << "\n The total of the ten integers is " << total;
    return 0;
} // end main
```

A good solution: general and clean.

Definition of Array

Note the difference between Java and C++

An array is an indexed/subscripted list of elements of the same type (a homogeneous collection).



Declaration:

```
int a[10];
```

Declare ten memory cells to store ten integers. The size must be constant.

access:

```
a[2] = 5;
cout << a[2];
```

Put integer 5 to the third memory cell.

An array in
- C++: *a collection of objects*
- Java: *an object that contains a collection of objects*

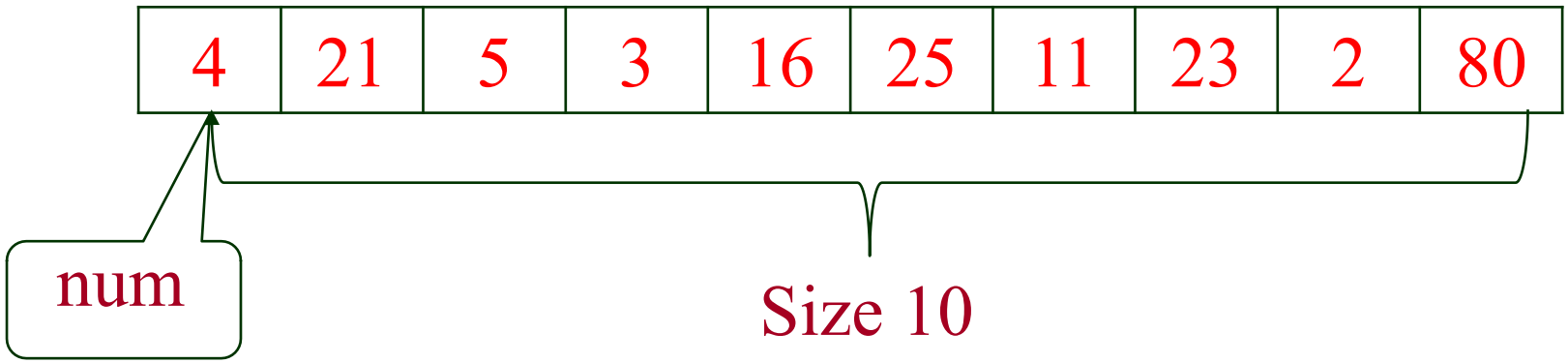
Java:

```
int[] a = new int[10];
```

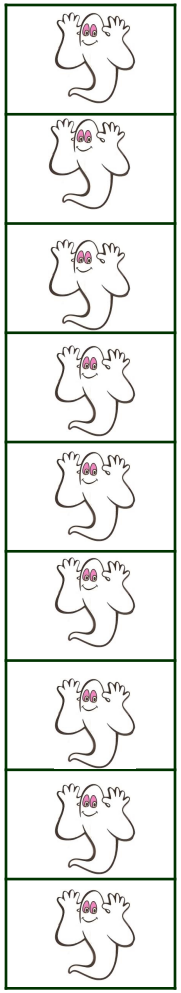
Three components of an array

- Think of array as 3 "pieces"
  - Array **type**: What is the type of the elements in the array?
  - Array **name**: Where does it starts?
  - **Size** of the array: How many elements it can contain?
- The array name is actually a pointer (see lecture 6)

```
int num[10]
```



Initialising an Array

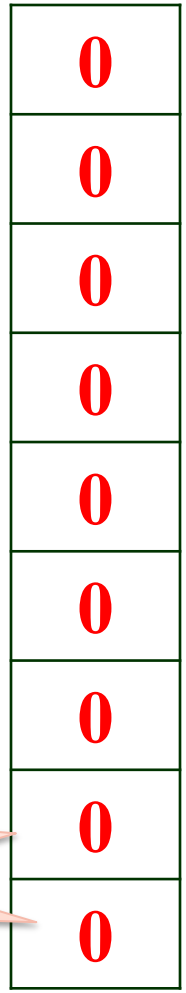


scores

```
int scores[9];

for (int i = 0; i <9; i++) {
   scores[i] = 0;
}
```

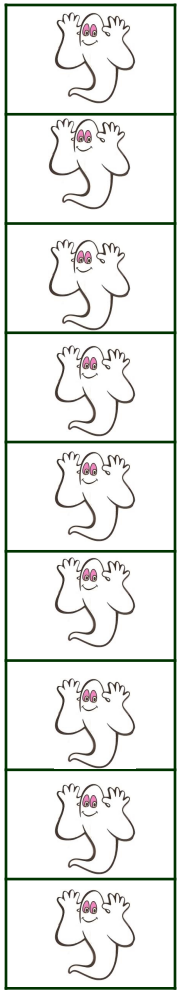
What are in the memory cells before you initialise them?

Note the difference between Java and C++



scores

Initialising an array: Enumerating



```
int scores[ ] = {2, 5, 9, 10, 4, 2, 7, 8, 5};
```

Compiler sets the size to match the number of elements in the list

2
5
9
10
4
2
7
8
5

Pass an array to a function

Note the difference between Java and C++

- When you pass an array to a function, you also need to pass the information of type, the name and the size of the function.
- Formal arguments in the function:
  - **Type**: the same as the array you want to pass
  - **Formal name** of the array with indicator
  - **Array size**: can be smaller than the actual array

  e.g., `function(int list[], int size);`
- Actual arguments:
  - The name of the array that carries the data (it's a pointer)
  - The actual size of data you pass to the function

  e.g., `function(num, 10);`

Pass an array to a function

Array as a formal parameter

```
void printArray(int a[], int length) {
  for (int i = 0; i < length; i++)
    cout << a[i] << endl;
}

int main() {
  int a[] = { 1, 2, 3, 4, 5 };
  int b[] = { 4,-1,3,20,7,9,4,11,-2};
  printArray(a, 5);
  printArray(b, 9);
  return 0;
}
```

Arrays passed to the function

See code 05-03.cpp

Pass an array to a function

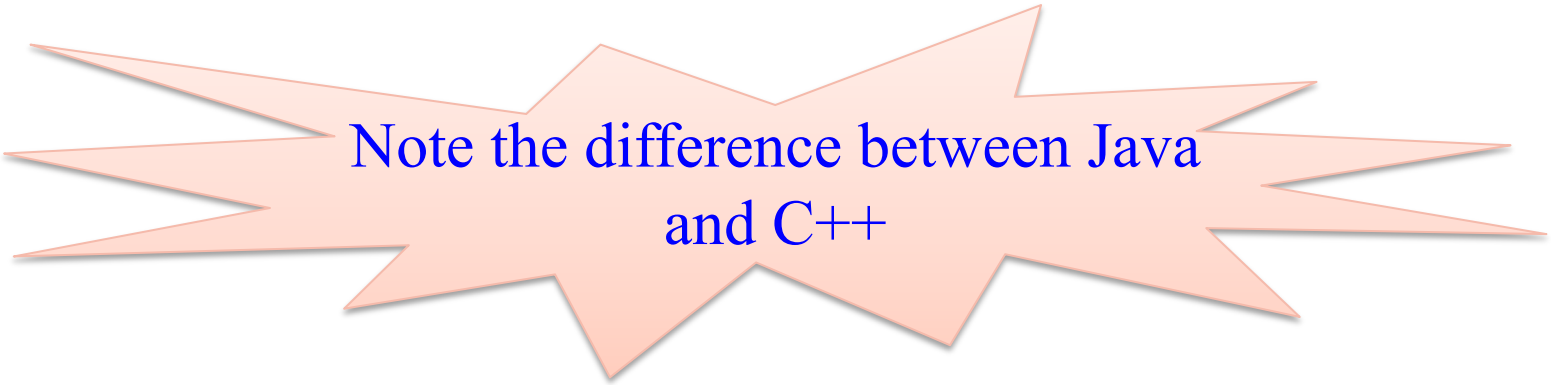
- Why do we specify the size of an array separately in C++:
  - The array name is a pointer, which points to the first element of the array
  - We can pass partial data of an array to a function
  - We can pass different arrays to the same function if the type of array is the same
  - Example:

```
int score[5], time[10];
printArray(score, 5);
printArray(time, 10);
```

See code 05-03.cpp again

Function that returns an array

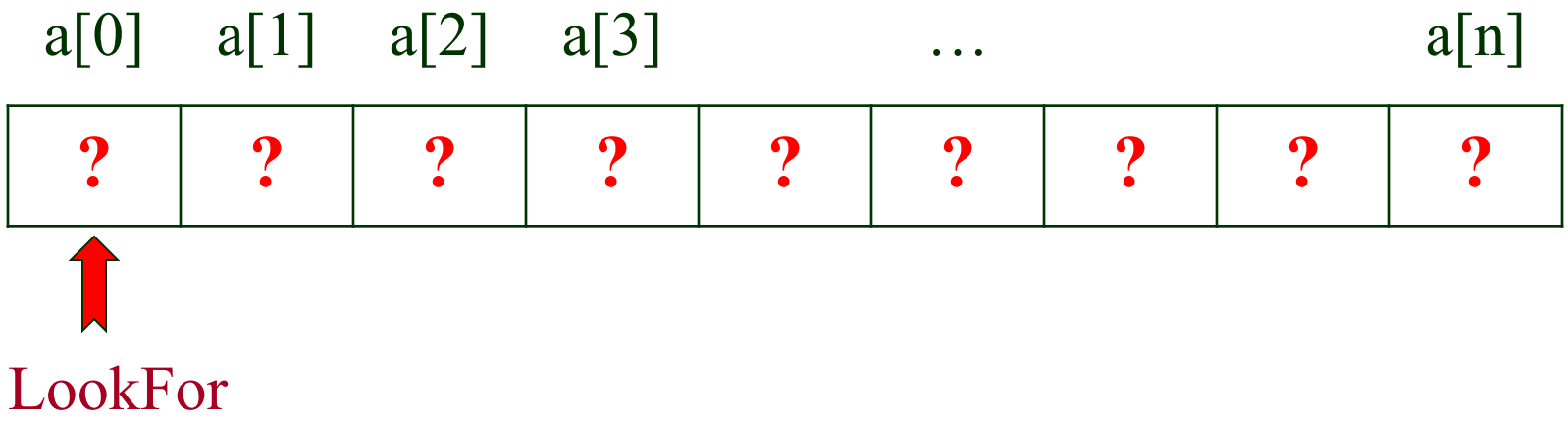
- In principal, a function **cannot** return an array. There is no such thing as `int[] function()` in C++.
- However, you can use a pointer to return a static array.
- Will talk about it Lecture 6…



Common Array Processing

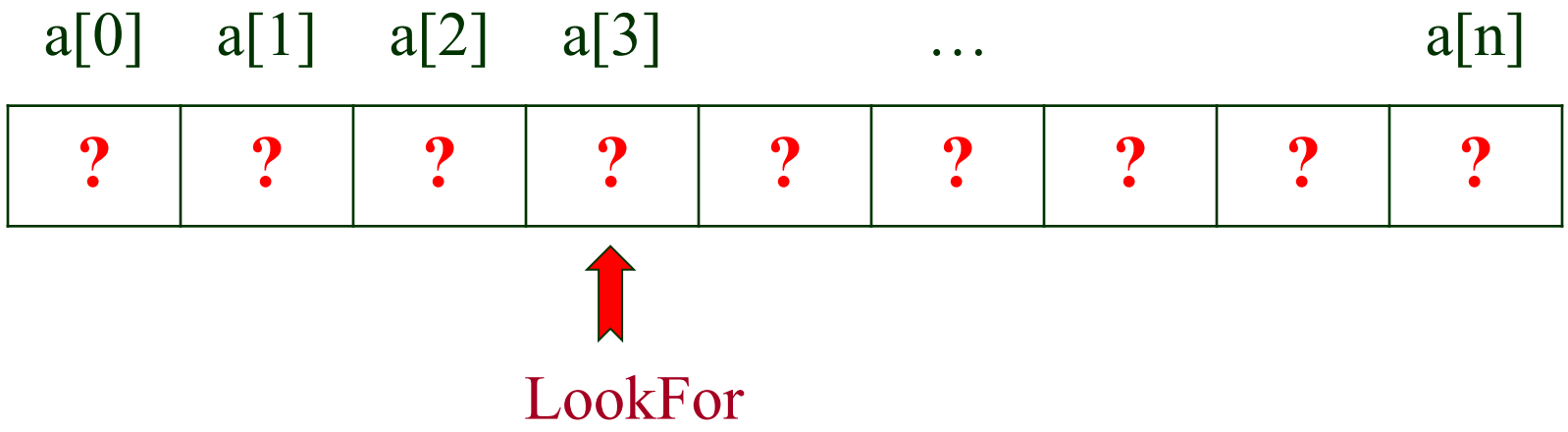
- Search: *finding a value in the array*
- Sort: *arrange the values in order*
- Calculate the average
- Calculate the minimum
- Calculate the maximum
- …

Linear Search



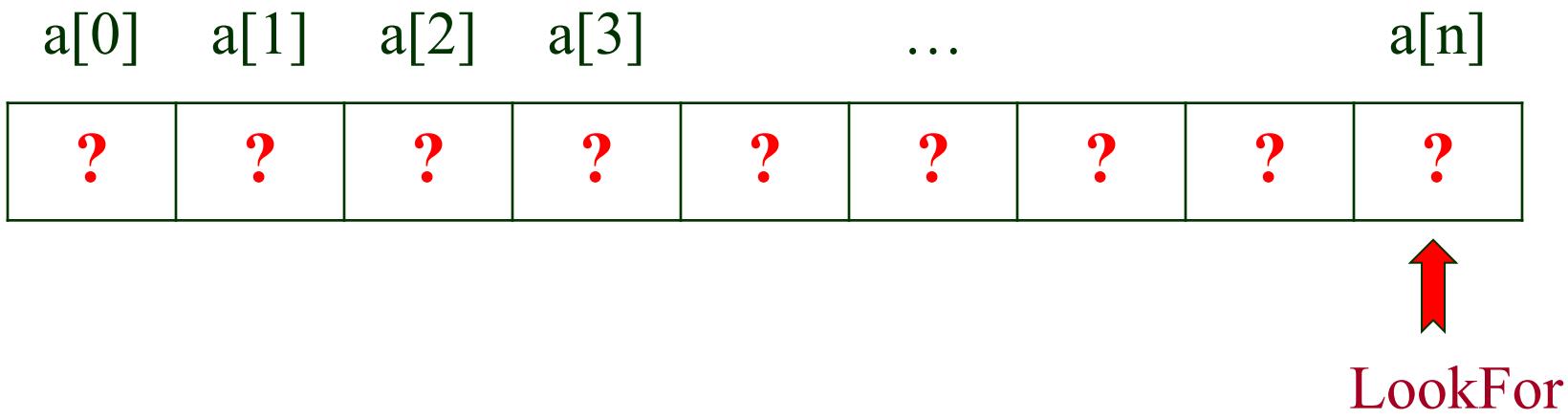
- Search: find a specific value in a set of data
- Linear search: travel through the elements one by one until find.

Linear Search



- Search: find a specific value in a set of data
- Linear search: travel through the elements one by one until find.

Linear Search



- Search: find a specific value in a set of data
- Linear search: travel through the elements one by one until find.

C++ Code for Linear Search

```
int linearSearch(int data[ ], int size, int lookFor) {
    bool found = false;
    int  index;
    for( index  = 0; index < size && !found; index++ ) {
         if ( data[index] == lookFor )
             found = true;
    }

    if (found)
        return index;
    else
        return -1;
}  // end linearSearch
```

The index of an array start from 0!

You get punishment if you do not remember it!

Calculate maximum

2
5
9
10
4
2
7
8
5

```
int main( )
{
    int data[ ] = {2, 5, 9, 10, 4, 2, 7, 8, 5};

    int largest = data[0]; //or a number less than any
                                    numbers in the array
    for (int i = 1; i < 9; i++)  {
        if (data[i] > largest)  {
            largest = data[i];
        }
    }
   cout << “The largest number is” << largest<<endl;
   return 0;
}
```

Two-Dimensional Arrays

A two-dimensional array is made up of rows and columns (like a spread sheet or a table)

	0	...	n
0			
1			
...			
m			

In C++, a two-dimensional array is an array of arrays.

Declaration

```
const int ROWS = 4;
const int COLS = 3;
int table[ROWS][COLS];
```

Must contain integer values.

OR

```
int table[ROWS][COLS] = {{2,1,3},
                         {4,12,7},
                         {6,2,4},
                         {10,1,2}};
```

OR

```
int table[ROWS][COLS] = {2,1,3,4,12,7,6,2,4,10,1,2};
```

Two dimensional array is simply an array of arrays.

Accessing array elements

```
amount = table[2][1]; //take values
table[0][0] = 2; //set values
newValue = 4*(table[1][0]-5);
table[1][2]=table[3][2]+table[2][0];

for(int i=0;i<4;i++)
  for(int j=0;j<3;j++)
    table[i][j] = i*j;
```

2	21	-3
9	4	21
11	2	-5
2	3	10

Processing 2-D Arrays

- **Getting input**

```
for (int i = 0; i < ROWS; i ++)
   for (int j = 0; j < COLS; j++)   {
      cout << "Enter the next value: ";
      cin  >> table[i][j];
   }
```

- **Displaying output**

```
for (int i = 0; i < ROWS; i ++)  {
   for (int j = 0; j < COLS; j++)
      cout << setw(3) << table[i][j];

   cout << endl;
}
```

See code io2Array.cpp

Passing 2-D array to a function

```
const int ROWS=2;
const int COLS=3;
void printTab(int table[ROWS][COLS]) {
   for (int i = 0; i < ROWS; i ++)  {
        for (int j = 0; j < COLS; j++)
                cout << setw(3) << table[i][j];
        cout << endl;
    }
}
int main() {
   int table[ROWS][COLS];
   for (int i = 0; i < ROWS; i ++)
        for (int j = 0; j < COLS; j++)   {
                cout << "Enter the next value: ";
                cin  >> table[i][j];
        }
   printTab(table);
   return 0;
}
```

Can be omitted

See code display2D.cpp

In class practice

Write a program that calculate the total of each row in a matrix.

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

Summary

- Array is collection of data in the same type
- Array elements stored sequentially
  - "Contiguous" portion of memory (arranged by operation system)
  - The name of the array is the point to the 1st element of the array
- The use of an array element is the same as the use of any other variable
- Programmer responsible for staying "*inbound*"
- Multidimensional arrays
  - Create "array of arrays"

Homework

- Read textbook Chapter 5
- Check out the tasks of Assignment 1
- Complete practical tasks for week 3 (Practical 2).
- Warm up Chapter 6