C++ Programming

Arrays

M1 Math
Michail Lampis
michail.lampis@dauphine.fr

Arrays

- Correspond to notion of 1-dimensional vector
- Allow to store many items of **same type**
- Items are accessed with an int index

```
int a[5]; //Allocate an array of 5 integers
a[2] = 23; //Write the value at position 2
cout << (a[2] + 1) << endl;
```

Arrays

- Arrays should be declared only inside functions
- Their size must be given in declaration

```
int a[5]; //Array of five integers
int x;
x=…;
int b[2*x]; //Array whose size depends on x
    // Note! Is x>0?
```

Accessing array elements

- Use the [ ] operators to access individual elements
- Numbering starts at **0** and goes up to **size-1**

```
int a[3];
a[0] = 17; a[1] = 23; a[2] = 75; // a = [17,23,75];
```

- Allowed to initialize array at declaration

```
int a[3] = {17,23,75}; //Same as before
```

- Note: a[i] is an lvalue (it can be referenced)

Out of bounds

- Careful with the following:

```
int a[5];
a[0] = 2; //OK
a[-2] = 5; //??
a[7] = 8; //??
a[a[0]] = 3; //OK
```

- No compiler or (predictable) run-time error!

Easy example

- The following loop finds the max in an int array

```
int data[size];
int max = data[0];
//The following loop follows a standard pattern
for (int i = 0; i<size; i++){
    max = data[i] > max? data[i] : max ;
}
```

Example 2

- Write a loop that calculates the average value of an int array

Example 2

- Write a loop that calculates the average value of an int array

```
int data[size];

double avg;

for(int i=0; i<size; i++)
    avg += data[i];

avg = avg/size;
```

Arrays and Functions

- It is possible to pass an array as a function parameter
- Semantics are call-by-reference, not call-by-value
  - Reason: array is not copied
  - Only reference to array is passed

find_max

```
int find_max(int data[], int size)
{
int max = data[0];
for (int i = 0; i<size; i++)
    max = data[i] > max? data[i] : max ;
return max;
}
```

Arrays and Functions

- Note the prototype

  int find_max(int data[], int size)

- The first parameter has type array, of unspecified size
  - The size **cannot** specified (it will be ignored)
- Imperative to supply the array size as parameter
  - Impossible to recover otherwise inside the function

Why we need the size

```
int f(int c[])
{
	cout << sizeof(c) << endl;
}

int main()
{
	int a[10];
	cout << sizeof(a) << endl; //Prints 40
	f(a); //Prints what?
}
```

Array pitfalls

- Common operators may not compile, or not work as expected
- Example 1

```
int a[3] = {3,4,5};

int b[3];

b = a; //??
```

Array pitfalls

- Common operators may not compile, or not work as expected
- Example 2

```
int a[3] = {3,4,5};

int b[3] = {3,4,5};

if( a == b) cout << “The same”! << endl;

//Will this print anything?
```

Explanation

- The identifiers a,b in the previous programs are allowed to be considered as variables themselves (of type array) BUT
  - They cannot be written on (**const** type)
  - Their actual value is **NOT** the array, but the place in memory where the array is stored
- This will become more familiar once we talk about **pointers**

Returning arrays

- Suppose that I want to write a function that, given n, return the array [1,2,3,...,n]
- First try

```
int [] myarray(int n)  //WRONG!!
{
int a[n];
for(int i=0;i<n;i++) a[i] = i+1;
return a;
}
```

Returning Arrays

- Arrays cannot be returned for two reasons
  - Returning an array is forbidden by the C++ rules (stupid reason)
  - The memory where the array is stored is destroyed when the function terminates!
  - Recall: Normally, **return** copies its operand (with the = operation). The = operation does not work on arrays...
- Solution: pointers (next class)

An example

- The sieve of Eratosthenes
- Write a program that decides which of the integers 1,2,...,n are primes
- For each i
  - If i is prime mark all multiples of i as non-prime

Eratosthenes

```
Bool isPrime[n];
int i;
for(i=0;i<n;i++) isPrime[i]=true; //Initially all prime
for(i=2;i<n;i++)
    if(isPrime[i]){
        cout << i  << " is prime" << endl;
        int j = 2*i;
        while(j<n){ isPrime[j] = false; j+=i; }
    }
```

Arrays and Strings

- Traditionally in C strings are just char arrays

  ```
  char myword[] = “hello”;
  if(myword[1] == 'e') { .. } //this will be executed
  ```

- myword[5] == 0 // note that this is not '0' but '\0'
- C++ has a string class
  - Much nicer, more convenient, less buggy
  - Avoid C-type strings if possible

Resizing Arrays?

- Once an array has been declared its value is **fixed**
- What if I need more/less space?
- Two solutions:
  - **Vector** class (to be seen later)
  - Dynamic memory management (**new, delete)**
- Second solution is needed also to return arrays