C++

and Object Oriented Programming (OOP)

New in C++

General	OOP

• input / output
• Strings, booleans
• function calls (by reference)
• Templates
• STL (standard template library)
  • Containers (vectors, maps etc.)
  • Algorithms (sort, find etc.)
• memory allocation (malloc vs new)
 	
Threads (Parallel programming)
Exceptions (Error catching)
	Classes and Objects
	
	• Encapsulation
	• Inheritance
	• Polymorphism
	
	RAII (Memory leaks)
	
	Iterators
	Smart pointers

Console I/O

cin >>
cout <<
setw
setprecision
setbase

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main(int argc, char** argv) {
    // OUTPUT
    cout<<setw(10)<<"ten"<<setfill('-')<<setw(10)<<"four"<<setw(6)<<"four"<<"stop";
    cout<<"hello"<<endl;
    cout<<left<<setw(10)<<42<<endl;
    int pr = cout.precision();
    cout << setprecision(3) << 2.71828 << endl;
    cout << 3.141596<<endl;
    cout << 0.00042123 <<" " << 0.0000042123 << endl;
    cout <<setbase(16)<<42<<" "<<123456<<endl;
    cout << 10+20 << endl;
    cout << 18.45268 << "\n" << setprecision(pr) << 18.45268 << endl;
    cout<< pr << endl;
    cout << setprecision(3) << 1.0/3.0 << endl;
    cout << setprecision(20) << 1.0/3.0 << endl;
    cout << setprecision(20) << 1.0/10.0 << endl;
    cout << 42 << endl;
    // INPUT
    int x, y, z;
    cout <<"Enter 3 integers: \n";
    cin >> x >> y >> z;
    cout << "Mean: " << (x+y+z)/3.0 << endl;
    return 0;
}
```

File I/O

Input from file
f >>

Output to file
f <<

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>


using namespace std;

int main() {
    ofstream myfile("out.txt");

    myfile << "hello" << 42 << endl;

    for (int i=0; i<10; i++){
        myfile << setw(12-i) << i << endl;
    }

    ifstream f("test.txt");
    int value = 5;
    f >> value;
    cout << value << endl;
    while(f >> value){
        cout << value << endl;
    }

    return 0;
}

```

New types: String, Boolean

```cpp
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv) {
    string a("Hello");
    string b = "World";
    string c = a + b;
    cout << c << endl;
    cout << a << " " << b << "\t" << 42 << endl;
    char d[] = "old C style.";
    c = d;
    string e(d);
    cout << d << e << endl;

    bool k = 3 < 1;
    cout << k << (13 > 2) << true << false << 2 + true << endl;
    if(k) cout << "Correct";
    else cout << "Wrong";

    cout << endl;
    a = "test"; b = "test";
    char f[] = "old C style.";
    cout << (d==f) << endl;
    cout << (a==b) << endl;
    return 0;
}
```

Swap algorithm

- temp = a
- a = b
- b = temp

Hands on: Write a function [void exchange(int,int)] that accepts two integers and swaps their values. In main() declare int x=5, y =3. Call exchange(x,y) and then cout x and y, to check that x=3 and y=5.

[Don’t use swap(x, y) - swap is the name of a function of the standard library]

References

A reference is an alias to a variable.

```
int a = 5;
int &x = a;
```

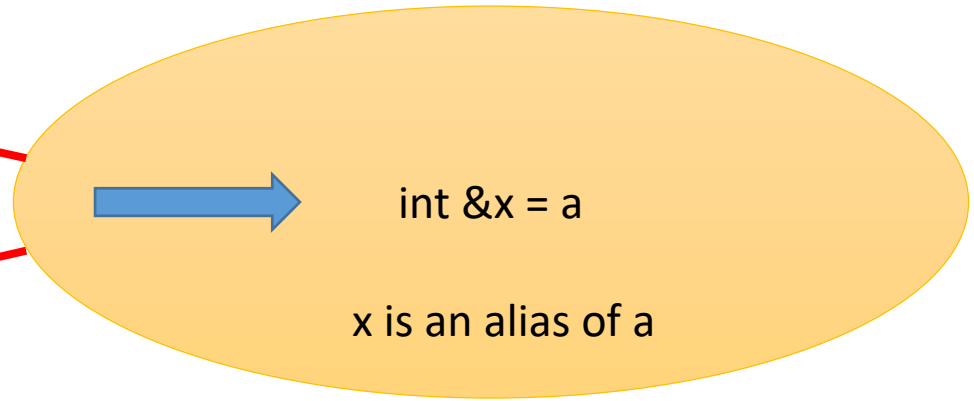
x is an alias (pseudonym) of a.

Used in functions when we want to:

a) return multiple values
b) make changes to the original variables
c) send large arrays / objects

Call by reference

```
void somefunction(int &x){
        x = 42;
}
int main(){
        int a = 23;
        somefunction(a);
        // a is now 42
}
```



Any change in x inside the function, will reflect to variable a in main.

Swapping

```
1   #include <iostream>
2
3   using namespace std;
4
5   void exchange(int& a, int& b){
6       int temp = a;
7       a = b;
8       b = temp;
9   }
10
11  int main(int argc, char** argv) {
12      int x = 5, y = 3;
13      cout << "x= " << x << " y= " << y << endl;
14      exchange(x,y);
15      cout << "x= " << x << " y= " << y << endl;
16      return 0;
17  }
```

But…
What if we want to swap double's?
long int's? unsigned int's?

Templates for the rescue!!

Template function

```
1   #include <iostream>
2
3   using namespace std;
4
5   template<typename T> void exchange(T& a, T& b){
6       T temp = a;
7       a = b;
8       b = temp;
9   }
10
11  int main(int argc, char** argv) {
12      double x = 65.2, y = 73.6;
13      cout << "x= " << x << " y= " << y << endl;
14      exchange(x,y);
15      cout << "x= " << x << " y= " << y << endl;
16      return 0;
17  }
```

T can be any type: int, double, long int, unsigned int, char, unsigned char, etc. It can even be a custom type defined by us (a class)

STL (Standard Template Library)

Containers	Algoriithms	Other stuff
• Vector • Map • Set • Deque • List etc	• Sort • Find • binary_search • Count • Shuffle etc	• Iterators • Functors

Vector (1)

- Replaces C arrays
- Is actually an array, with many extras
  - Dynamically allocated
  - Resizable
  - Knows its size
  - Can be send to and returned by a function
  - We can use algorithms like sort(), shuffle() etc.
  - Can be safe (if we let it!)

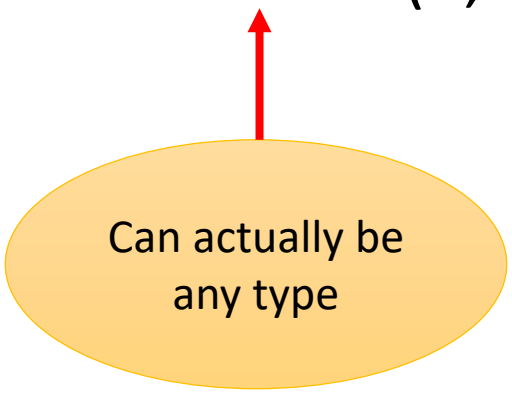
Vector (2)

Declaration / initialization examples:

vector<int> a; // zero sized vector

vector<int> a(n); // n elements

vector<int> a(n, v) // n elements, all initialized to v



Usage:

a[0] = 42;

int x = a[1];

just like an array!

Exercise

In main() create a vector of 100 random integers in the range [1..1000]

Write a function that sorts an array of ints (bubble sort, merge sort etc.) and call it from main(), sending the vector by reference.

Write a function that prints the elements of the (sorted) vector to a file.

Classes

C

```
struct Point {
  int x;
  int y;
}
```

C++

```
class Point {
  int x;
  int y;
 public:
  int getDistance();
  void setX(int);
  void setY(int);
}
```

Procedural vs OO Programming

- Procedural

Think of what needs to be done, the procedures we have to follow and then decide about the data structures

- OOP

Decide what your “Actors” or objects are, describe their state and how this state changes.

Παραδειγμα: Εφαρμογή για Τράπεζα

- Procedural

Άνοιγμα λογαριασμού, Κατάθεση, Ανάληψη, Ερώτημα Υπολοίπου, Χορήγηση δανείου, επιβολή επιτοκίου, υπολογισμός τόκων κλπ.

- OOP

Objects: Πελάτες (Άτομα – Επιχειρήσεις), Λογαριασμοί (καταθέσεως – όψεως), Χαρτοφυλάκια μετοχών, Υπάλληλοι κλπ.

Classes - Encapsulation

```cpp
#include <iostream>

using namespace std;

class Person{
    string name;
    string contactPhone;
    string contactAddress;
    int yearBirth;
};

int main(int argc, char** argv) {
    Person m;
    m.yearBirth = 1990;
    return 0;
}
```

Classes - Encapsulation

Two solutions:

```cpp
#include <iostream>

using namespace std;

class Person{
    string name;
    string contactPhone;
    string contactAddress;
    public:
    int yearBirth;
};

int main(int argc, char** argv) {
    Person m;
    m.yearBirth = 1990;
    return 0;
}
```

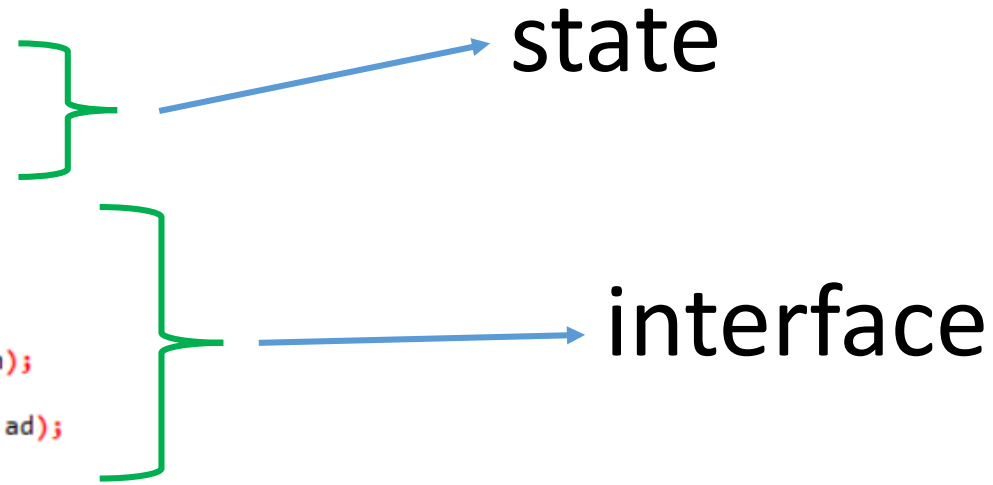
BAD !!!

```cpp
#include <iostream>

using namespace std;

class Person{
    string name;
    string contactPhone;
    string contactAddress;
    int yearBirth;
    public:
    void setYearBirth(int y){
        yearBirth = y;
    }
};

int main(int argc, char** argv) {
    Person m;
    //m.yearBirth = 1990;
    m.setYearBirth(1990);
    return 0;
}
```

GOOD !!!

Class structure

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Person{
6
7      private:
8          string name;
9          string contactPhone;
10         string contactAddress;
11         int yearBirth;
12     public:
13         void setYearBirth(int y);
14         int getYearBirth();
15         void setName(string ph);
16         string getName();
17         void setContactPhone(string ph);
18         string getContactPhone();
19         void setContactAddress(string ad);
20         string getContactAddress();
21 };
22
23 int main(int argc, char** argv) {
24     Person m;
25     //m.yearBirth = 1990;
26     m.setYearBirth(1990);
27     return 0;
28 }
29
```



main.cpp

```cpp
#include <iostream>

using namespace std;
#include "Person.h"

int main(int argc, char** argv) {
    Person m;
    //m.yearBirth = 1990; /* ILLEGAL */
    m.setYearBirth(1990);
    return 0;
}

```

Person.h

```cpp
class Person{

    private:
        string name;
        string contactPhone;
        string contactAddress;
        int yearBirth;
    public:
        void setYearBirth(int y);
        int getYearBirth();
        void setName(string ph);
        string getName();
        void setContactPhone(string ph);
        string getContactPhone();
        void setContactAddress(string ad);
        string getContactAddress();
};

void Person::setYearBirth(int y){
    yearBirth = y;
}
int Person::getYearBirth(){
    return yearBirth;
}

```

Constructor - Κατασκευαστής

- Special method (function) that gets called **automatically**, every time an object is created
- Always has the **same name** as the class and NO return type. It doesn't return anything, but we don't specify a "void" return type.
- Used for initializing the variables of the object (state) and possibly set up any resources needed (e.g. allocate memory, open a filestream, establish a connection etc.)
- If we don't specify one in our code a "default" one is created by the compiler, with no arguments, which does NOTHING.

Constructors

```
5           string contactPhone;
6           string contactAddress;
7           int yearBirth;
8       public:
9           Person(){       // default constructor
10          }
11          Person(string n, string p, string a, int y){    // constructor
12              name = n;
13              contactPhone = p;
14              contactAddress = a;
15              yearBirth = y;
16          }
17          void setYearBirth(int y);
18          int getYearBirth();
19          void setName(string ph);
20          string getName();
21          void setContactPhone(string ph);
22          string getContactPhone();
23          void setContactAddress(string ad);
24          string getContactAddress();
25  };
26
```

Object creation

```
1   #include <iostream>
2
3   using namespace std;
4   #include "Person.h"
5
6   int main(int argc, char** argv) {
7       Person m;
8       //m.yearBirth = 1990; /* ILLEGAL */
9       m.setYearBirth(1990);
10      Person john("John Smith","+306998424242", "10, 3rd September, Thessaloniki", 1992);
11      return 0;
12  }
```

Hands on

- Write a Person class, with private fields ‘age’ and ‘name’. The code must be written in a separate file e.g. “person.h”
- The interface should provide functions (getters – setters) for the private fields and two constructor (the default and one for initializing all fields). Also, a void printPerson() function, which prints: “Name: xxx, Age: yyy” at the console.
- In main() create 2 person objects. One using the default constructor (set the fields immediately after creating the object) and one using the other const’r.

Array of Objects

- We can create an **array** of objects:

```
Person ps[10];
ή
vector<Person> ps(10);
```

The objects are first created and then inserted (copied) into the array.

No default constructor

- If we define a constructor, the compiler does not create the default one. So we cannot declare (create) an object simply like this:

```
Person someone; // ILLEGAL
```

- We can also not create an array of objects, as we did in the previous page

SOLUTIONS

1. Create each object individually and push_back() it into the vector
2. Create an array of Person **pointers**