Data Structures and Algorithms – Lab2

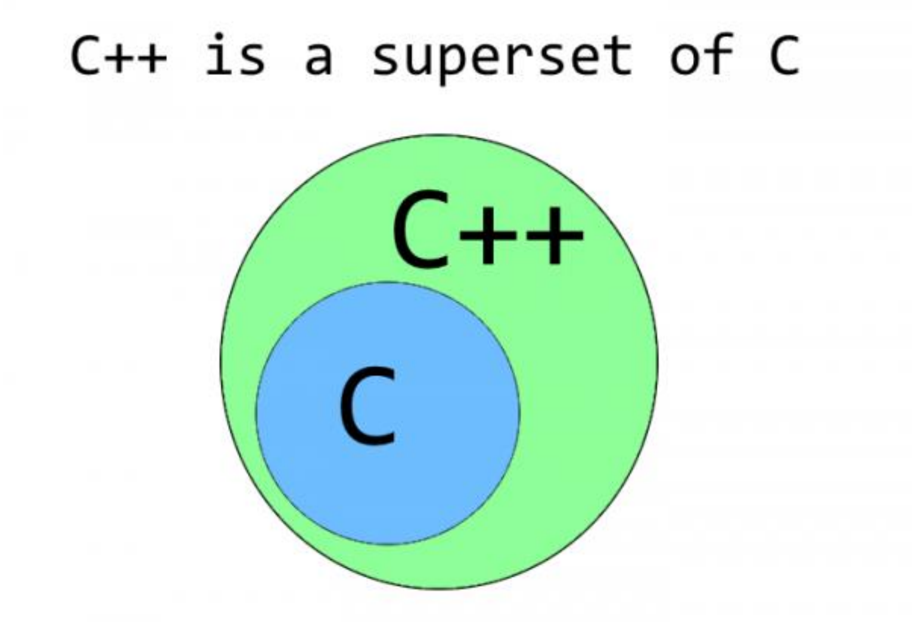
Iulia-Cristina Stănică
iulia.stanica@gmail.com

Roadmap

- Transition from C (first lab) to C++ (2nd lab)
- Structs
- Classes in C++

1. Transition from C to C++

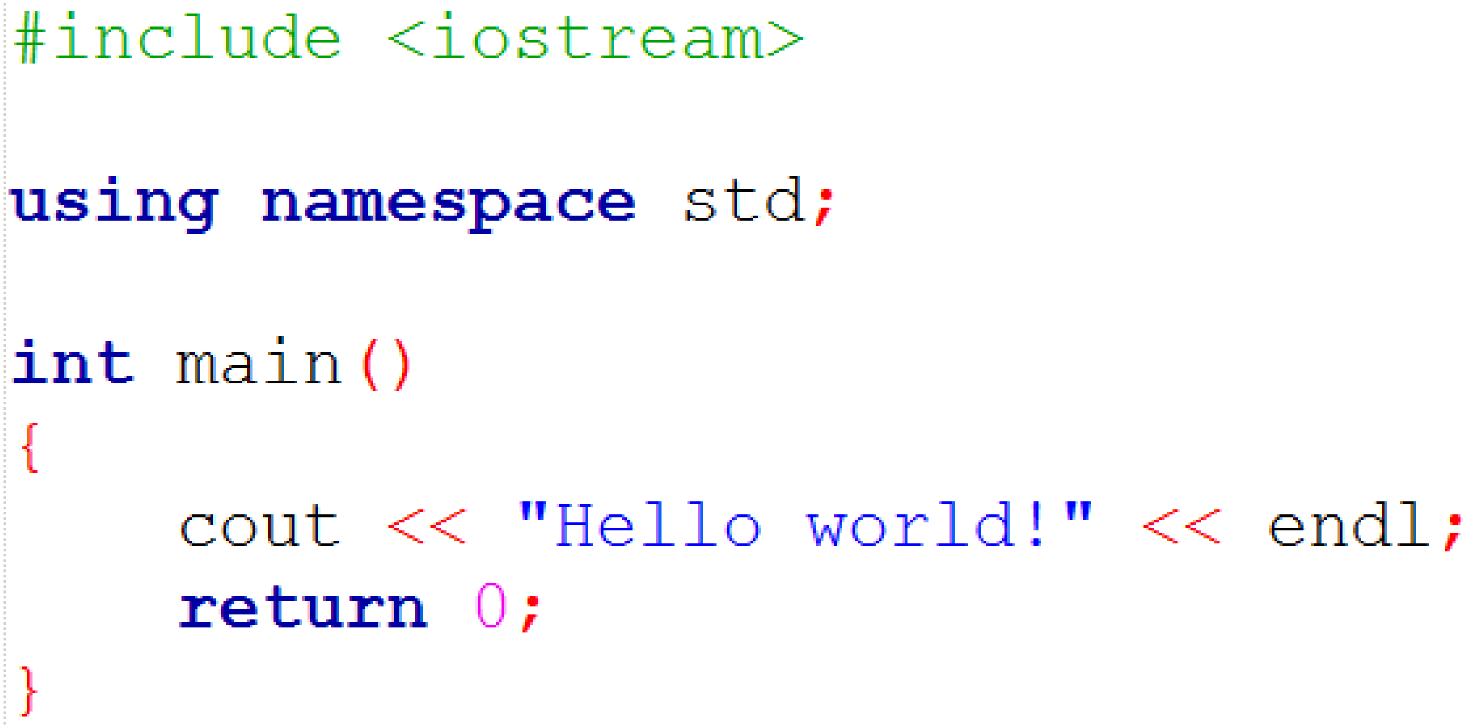
- C++ = superset of C language
- Any program written in C can be compiled by a C++ compiler (“.cpp” extension); not vice versa
- In C we don’t have classes (no OOP)



1. Transition from C to C++

- C++ uses **NAMESPACE** feature (allows the use of functions with the same name, in different classes, for instance)
- **Scanf** and **printf** from C become **cin** >> and **cout**<<;
- Standard input & output functions from **stdio.h** are replaced by the ones from **iostream**

C++ Example



```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

C++ Example

- Observation: In order to avoid conflicts which may appear because of NAMESPACES, it is advised to use the name of the namespace before each function call:
  - std::cin
  - std::cout

Passing by value/Passing by reference

- **Passing by value** means that a copy of the object is made and altering the object means altering a local copy so the caller’s object is unchanged when the function returns.
- **Passing by reference** means that the address of the object is sent so that within the function we can directly alter the original object.
- (See examples)

2. Structures

- a user-defined data type that allows grouping of heterogeneous elements;
- a collection of one or more variables (fields), grouped under one name;
- Format: **struct** [name structure]

  {members};
- the members of a structure are accessed with «.»: struct_name.variable_name;

2. Structures – example

```
struct data{
      unsigned int day;
      unsigned int month;
      unsigned int year;
      char name_day[4];
      char name_year[4];
};
data today;
```

- Use:

```
void writeDDMMMYYYY(data myDate)
{
    printf("%d %d %4d ", myDate.day,
    myDate.month, myDate.year);
}
```

Structures – exercise 1

- Write a structure to represent dates and write functions for:
  - Verifying if a date is valid
  - Calculating the next date (tomorrow) of a certain date
  - Calculating the previous date (yesterday) of a certain date

3. Classes

```
class class_name {
                    access_specifier:
                      members;
                      methods;
                    access_specifier :
                      members;
                      methods;
      ...
                    constructor; // same name as the class
                    destructor; // ~class_name
};
```

- Access specifier = public / private / protected
- A class can have variables and methods
- **Attention!** Do not forget to put “;” at the end of the class definition!

Class instances

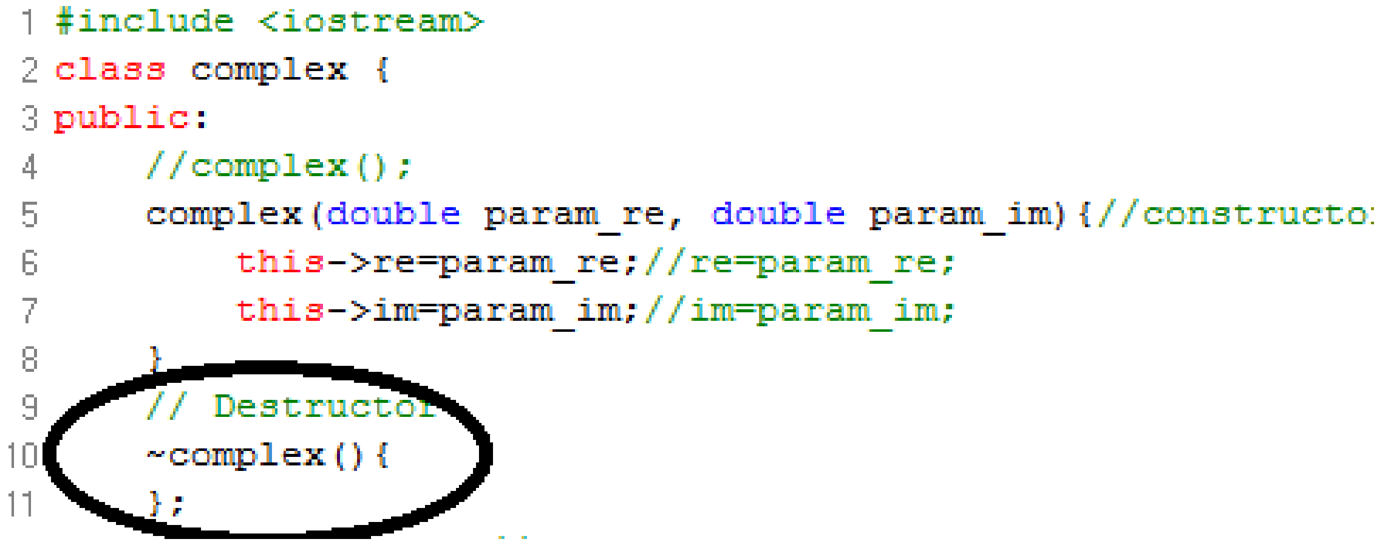
- We create instances of a class:
  - class_name instance_name (param_values);

- We call the methods of the class:
  - instance_name.method_name(param_values);

Ex: Class Complex

```
#include <iostream>
class complex {
private:    //class variables
     double re;
     double im;
public:
     complex() {}; // constructor without params
     complex(double param_re, double param_im) {//constructor
          // used to initialize the members of the class with values and to allocates memory for  some
     members
               this->re=param_re; //re=param_re or (*this).re, later
               this->im=param_im;//im=param_im
     }
     double getRe(){//method: getter
               return re;
     }
     double getIm(){ //method: getter
               return im;
     }
     complex complex_conjugate() { //method- conjugate of a complex number
     complex conjugate(re,-im);//object of type complex
               return conjugate;
      }   };
```

Destructor



```cpp
#include <iostream>
class complex {
public:
    //complex();
    complex(double param_re, double param_im){//constructo
        this->re=param_re;//re=param_re;
        this->im=param_im;//im=param_im;
    }
    // Destructor
    ~complex(){
    };
```

The destructor is automatically called when an object is destroyed, for instance because its scope of existence has finished.

Ex: Class Complex

- Good practice: Separating the body of the methods from their signature
- If smth changes in the implementation, only that specific file will be recompiled; the files containing the declaration (headers) or the files which include the headers will not be recompiled.

```
class complex {
    private:
            double re;
            double im;
    public:
            complex();
            complex(double param_re, double
    param_im);

            void adunare (complex c1);
    };
```

```
complex::complex()
{ }

complex::complex (double param_re, double
param_im)
{
   this->re=param_re;//re=param_re;
   this->im=param_im;//im=param_im;
}

void complex::adunare (complex c1)
{
            // code pour l'addition
}
```

Main for Complex class

```
int main(){
    complex number(2,3);// number is an object and complex is a class

    cout<<"The complex number is:“ << number.getRe() << "+“ <<
    number.getIm()<<"i"<<"\n";

//number.getRe() is a method call

    complex conj=number.complex_conjugate();

     cout<<"The conjugate number is: "<<conj.getRe()<< conj.getIm()<< "I” << "\n";
    return 0;
}
```

Struct vs Class

- **Struct** – **public** by default, no encapsulation; **Class** – **private** by default
- **Struct** also in C (but with no methods); no **Classes** in C

- Usually used for:
  - Struct for POD (plain old data) – only data members, no methods
  - Class – members+ methods

Exercise 2

- Add to the complex class new methods for adding and multiplying complex numbers.

Exercice 3

- We have the following struct called Point.

```
struct Point
{
    //public:
    int coord_x, coord_y; //coordinates

    void reset() //place the point in the origin
    {
        coord_x = coord_y = 0;
    }
    void moveX(int x); //move horizontally
    void moveY(int y); //move vertically
    void moveXY(int x , int y); //move in both ways verticalement
};
```

- Change it in a class, by adding constructors, getters and setters.
- Implement and test the methods.

Homework

- Write a program in C++ in which you define a class named BankAccount (with the members name, address, IBAN, sum etc.) which contains the usual banking operations (deposit, withdraw, display balance, display owner). Test the class.