Data Structures and Algorithms
Lab 2

Marin Iuliana
FILS

Roadmap

- Transition from C (first lab) to C++ (2nd lab)
- Struct vs. classes in C++
- Templates

Transition from C to C++ (1)

- C++ is a superset of C language
- Any program written in C (“.c” extension) can be compiled by a C++ compiler (“.cpp” extension); not vice versa
- C follows the procedural programming paradigm while C++ can follow both procedural paradigm and OOP: in C we don’t have classes!!!
- The NAMESPACE feature in C++ is absent in case of C: avoid name collisions (namespaces are similar, to a certain extant, to Java packages)
- Standard input & output functions differ in the two languages: in C, we have scanf and printf; in C++ we have cin>> and cout<<

Transition from C to C++ (2)

```
// the inclusion of iostream for I/O operations
// instead of #include <stdio.h> from C
/* All the elements of the standard C++ library are declared within
the std namespace*/

#include <iostream>
using namespace std;

int main()
{
    char c;
    cout<<"Insert a character:\n";
    cin>>c;
    cout<<"You have inserted\n "<<c <<"\n";
}
```

Transition from C to C++ (3)

- Parameter passing in functions: both by value (as in C or Java) and 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.

Ex: void foo(int &i) // Note the reference operator!!!!!

```
{
    i++;
    cout << "In the function: " << i << endl;
}
```

Exercise

Implement a function to sort an array of 5 elements of type double. Use a swap mechanism, which has to be implemented in another function.

```
template <ReturnType> void swap(Type& ob1, Type& ob2)
{
    Type aux = ob1;
    ob1 = ob2;
    ob2 = aux;
}
```

Transition from C to C++ (4)

- Structures exists both in C and C++, but there are small differences.
- Structures are user-defined data types that can group several variables which can be of different types (as opposed to vectors containing only the same data type)

```
struct student {
    char nume[40]; // field in a structure
    int an;
    float medie;
};

struct complex {
    double re;
    double im;
};
```

Examples (2)

```
// Declaration and initialization of variables of type struct
struct student s1 = {"Popa Ionel", 3, 9.25};
struct complex c1, c2;
struct complex v[10];

//or we use typedef

typedef struct student Student;
.....
Student s1, s2, s3;

// Access to struct fields
s1.nume = "Ionescu Raluca";
```

More examples…(1)

```
#include <iostream>
using namespace std;
typedef struct {
    int data;
    int text;
} S1; // a typedef for S1, functional in C and C++
struct S2 {
    int data;
    int text;
}; // a typedef for S2, functional only in C++
struct {
    int data;
    int text;
    void foo() { cout<<data;} //defining function within  a structure works only in C++
} S3; // it's a declaration for S3, a variable of type struct (it isn't a type definition)
int main(){
    S1 mine1; // corect
    S2 mine2; // corect
    //S3 mine3; // It won't work: S3 is not a typedef
    //S1.data = 5; //it won't work: S1 is a typedef, not a variable
    //S2.data = 5; //it won't work: S1 is a typedef, not a variable
    S3.data = 5; //correct
    S3.foo();
    return 0;
}
```

More examples…(2)

```
#include <iostream>
using namespace std;
struct complex {
      double re;
      double im;
      void complex_initialize(double param_re, double param_im) {
                  this->re = param_re;// re = param_re;
                  this->im = param_im;// im = param_im;
        }
       struct complex complex_conjugate() {
                  struct complex conjugate;
                  conjugate.complex_initialize(this->re, -(this->im));
                   return conjugate;
    }
};
int main()
{
    struct complex number;
    number.complex_initialize(2, 3);
    cout<<"The complex number is: "<<number.re<<"+"<<number.im<<"i"<<"\n";
    cout<<"The conjugate number is:
      "<<number.complex_conjugate().re<<number.complex_conjugate().im<<"i\n";
    return 0;
}
```

Struct vs. Classes

- What is a class?
- What is an object?

- Replace the keyword *struct* with the keyword *class* in the example from the previous slide!!!

The difference stays in the default accessibility level:

-*public* for *struct*
-*private* for *class*

```
1 #include <iostream>
2 class complex {
3 public:
4     double re;
5     double im;
6
7     void complex_initialize(double param_re, do
8     {
9         this->re = param_re;// re = param_re;
10        this->im = param_im;// im = param_im;
11    }
12
13    struct complex complex_conjugate()
```

Add keyword "public"!!!

Classes in C: similar with Java

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

Destructor

```
1  #include <iostream>
2  class complex {
3  public:
4      //complex();
5      complex(double param_re, double param_im){//constructo
6          this->re=param_re;//re=param_re;
7          this->im=param_im;//im=param_im;
8      }
9      // Destructor
10     ~complex(){
11     };
```

- The destructor is automatically called when an object is destroyed, either because its scope of existence has finished (for example, if it was defined as a local object within a function and the function ends) or because it is an object dynamically assigned.

Later….

Dynamic allocation and deallocation: with *new* and *delete*!!!

```
28 int main()
29 {
30      complex *nr=new complex(2,3);
31      delete(nr);
```

Separating the body of the methods from their signature…

```
class complex {
public:
    complex(double param_re, double param_im);
      ~complex();
                double getRe();
                double getIm();
                complex complex_conjugate();
    private:
                double re;
       double im;
    };
    complex::complex(double param_re, double param_im){//constructor
                                this->re=param_re;//re=param_re;
                                this->im=param_im;//im=param_im;
                }
    complex::~complex(){}
    double complex::getRe(){//method
                return re;
                }
    double complex::getIm(){//method
                                return im;
                }
    complex complex::complex_conjugate()//method
       {
          complex conjugate(re,-im);//object of type complex
          return conjugate;
       }
    int main(){.....}
```

Exercise

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

```
class complex{
public:
    complex addition(const complex& );
    complex div(const complex& );
    complex mult(const complex& );
private:.....};

complex complex::addition(const complex &b){
    complex c(o, o);
    c.re=....; c.im=....;
    return c;
}
int main(){
    complex a(x,y), b(z,t), c(o,o);
    c=a.addition(b);
}
```

Template class

```
class KeyStorage
{
    public:
      int key;
        T member; //a generic member: we don't know its type when creating the class
    };
    int main()
    {
      //Everything happens to compile time, not to run time
     //The compiler analyses the way in which you use the class
              KeyStorage<long> keyElement1;
               KeyStorage<int> keyElement2;
     return 0;
}
```

Exercise

Make a constructor, a destructor, a getter and a setter for the template class *KeyStorage*.

```
#include<iostream>
template<typename T>
class KeyStorage{
      public:
      int key;
         T member; //a generic member: we don't know its type when creating the class
      KeyStorage();
      ~KeyStorage();
      T get();
      T set(T arg);
};
template<typename T>
KeyStorage<T>::KeyStorage(void){
       //TODO
}
template<typename T>
KeyStorage<T>::~KeyStorage(void){
       //TODO
}
template<typename T>
T KeyStorage<T>::get(){
      //TODO
}
template<typename T>
T KeyStorage<T>::set(T arg){
       //TODO
}
```

Function Templates

```
template<typename T>
 T getMax(T a, T b) {
        return a > b ? a : b;
 }


getMax<int>(2, 3);
 getMax<double>(3.2, 4.6);
```

Homework
Ex. 1

- Does overloading exist in C++?
- Do abstract classes exist in C++?
- When is it better to use abstract classes and when templates?
- Do static classes exist in C++?

Use www.cplusplus.com and other books, tutorials…

Ex. 2

Implement the following structure named Point:

```
struct Point {
//public:
int coord_x, coord_y;//the coordinates of a point
void reset() //places the point into the origin
{
coord_x = coord_y = 0;
}
void moveX(int x = 1); //moves the point horizontally with x positions
void moveY(int y = 1); // moves the point vertically with y positions
void moveXY(int x = 1, int y = 1); // moves the point horizontally and vertically
};
```

Make a program in which you implement the movement of a point within the coordinates of a rectangle with the dimensions 1 x N (N given by the user), starting from the origin.

Also considering a point that is placed at the left side of the origin, at a certain distance t, write its coordinates as the point moves around the origin with 360 degrees, maintaining the distance t between the point's new coordinates and the origin (it does not need to be (0, 0)).

Compile and run the application! Then, replace struct with class and do the same!

Ex. 3

Consider the structure from Ex.2 and the following function which interchanges two objects of type Point.

```
void Schimba(Point &A, Point &B) {
    int x = A.coord_x;
    A.coord_x = B.coord_x;
    B.coord_x = x;

    int y = A.coord_y;
    A.coord_y = B.coord_y;
    B.coord_y = y;
}
```

Make a program which sorts the triangles that can be made out of 5 points, in the ascending order of the triangle area.

Ex. 4

Write a program in C++ in which you define the operations performed while going shopping (put cash inside the wallet, pay with cash, pay by credit card, pay by debit card, test pin number, display total wallet balance). Test the class.