Inheritance in C++

Objectives:

- Derived classes
- Inheritance
- Access control
- Multiple inheritance
- Overloading member functions
- Constructors and the destructor of the derived class

Inheritance is a mechanism which allows a class A to inherit members (data and functions) of a class B. We say "A inherits from B". Objects of class A thus have access to members of class B without the need to redefine them. The syntax is:

class Derived_Class : [access modifier] Base_Class { …. } ;
or
class Derived_Class: [access modifier] Base_Class1, [access modifier] Base_Class2, [access modifier] Base_Class3 ……..

B is called superclass, supertype, base class or parent class.

A is called subclass, subtype, derived class or child class.

class A : **public** B { /* declarations */};

class A : **protected** B { /* declarations */};

class A : **private** B { /* declarations */};

If the access modifier is missing, then it is considered private by default.
Member functions from the derived class can access only public and protected members of the base class, they cannot access the private ones.

The following table illustrates the access to the members of the base class:

Base class	Access control	Access in derived class	External access
private protected public	private private private	not accessible private private	not accessible not accessible not accessible
private protected public	protected protected protected	not accessible protected protected	not accessible not accessible not accessible
private protected public	public public public	not accessible protected public	not accessible not accessible accessible

Study the following example, comment and detect/explain the errors:

```
#include <iostream>
class One
{
    int x;
    protected:
    int y;
    public:
    int z;
};

class Two:One
{
    void f()
    {
            //x=10; //err
            y=20;
            z=30;
    }
};

class Three:public One
{};

class Four:Three
{
    void f2();
};

void f()
{
    One u;
    Two d;
    Three t;
    u.x=11; //err
    u.y=22; //err
    u.z=33;

    d.x=11; //err
    d.y=22; //err
    d.z=33; //err

    t.x=11; //err
    t.y=22;//err
    t.z=33;
}
```

```
void Four::f2()
{
    x=11;//???
    y=22;  //??
    z=33;  //??
}
```

Derived class:

```
class Library:public Book
{
    char author[], publisher[];
    public:
    Library(char *title, char *author, char *publisher):Book(title)
    {
            strcpy(Library::author,author);
            strcpy(Library::publisher,publisher);
    }
};
```

In the base class Book we define and we initialize the title which will be inherited in the class Library. There is no need to redefine it.

In the moment when an object of the class Library is created, first it will be called the constructor of the class Book and then the constructor of the class. The constructor of the class Book has to be specified, has one single parameter, otherwise the compiler will generate an error. If the constructor of the class Book has no parameter, then the compiler does not generate an error and there is no need to specify this constructor.

Protected member from the base class are accessible in the derived class and they can be modified.

Example:

```
class Book
{
    char title[64];
    public:
    Book (char *title)
    {
            strcpy(Book::title,title);
    }
    void print_book()
    {
            cout<<"the title of the book is "<<title<<endl;
    }
    protected:
    float price;
    void print_price()
    {
            cout<<"the price of the book is "<<price<<endl;
    }
```

```
};
class Library:public Book
{
    char author[64], publisher[64];
    public:
    Library (char *title, char *author, char * publisher): Book (title)
    {
            strcpy(Library::author,author);
            strcpy(Library:: publisher, publisher);
            price=49.9;
    }
    void print_library()
    {
            print_book();
            print_price();
            cout<<"author: "<<author<<" publisher "<< publisher <<endl;
    }
};
void main()
{
Library lib("C++ Programming", "Stroustrup", "Teora");
lib.print_library();
}
```

Multiple inheritance

Constructors and destructors of the base classes and of the derived class:

```
#include <iostream>
class Book
{
    char title[64];
    float static const price2=100;
    protected:
    void print_price2()
    {
            cout<<"the price is "<<price2<<endl;
    }
    public:
    Book (char *title)
    {
            strcpy(Book::title,title);
            cout<<"the constructor of the base class 1"<<endl;
    }
    void print_book()
    {
            cout<<"the title of the book is "<<title<<endl;
    }
```

```
    ~ Book ()
    {
            cout<<"the destructor of the base class 1"<<endl;
    }
    protected:
    float price;
    void print_price()
    {
            cout<<"the price of the book is "<<price<<endl;
    }
};

class Pages
{
    protected:
    int lines;
    public:
    Pages (int lines)
    {
            Pages::lines = lines;
      cout<<"the constructor of the base class 2 "<<endl;
    }
    void print_ Pages ()
    {
            cout<<"the book has " <<lines<<" lines<<endl;
    }
    ~ Pages ()
    {
            cout<<"the destructor of the base class 2"<<endl;
    }

};

class Library: Book, Pages
{
    char author[64], publisher [64];
    public:
    Library (char *title, char *author, char * publisher): Book (title), Pages (166)
    {
            strcpy(Library::author,author);
            strcpy(Library:: publisher, publisher);
            price=49.98;
            cout<<"The constructor of the derived class"<<endl;
    }
    void print_library()
    {
            print_book();
            print_price();
            print_price2();
            cout<<"author: "<<author<<" publisher "<< publisher <<endl;
```

```
            print_ Pages ();
        }
        ~Library()
        {
            cout<<"The destructor of the derived class"<<endl;
        }
};
void main()
{
Library  lib("C++ Programming", "Stroustrup", " Publisher ");
lib.print_library();
}
```

Exercices:

1. Use your imagination and create a program in C++ which illustrates inheritance, multiple inheritance. (Example: consider a family: the parents have two children, a girl and a boy, each of them inherits something different from mother, father, grandfather, grandmother, cousins, like the eyes color, the hair color….).

2. When we have multiple inheritance what we can do in order to avoid the inheritance of multiple copies from the base class. Justify your answer by a correct program!

3. How do we solve the name conflicts from the base classes and from the derived classes?

Answer to exercise 3:

-we use the resolution operator in order for the compiler to know exactly from where to take the correcponding data.

Example:

```
class Base
{
  public:
  void print()
  {
        cout<<"We are in the base class"<<endl;
  }
};

class Derived:public Base
{
  public:
  void print()
  {
        cout<<"We are in the derived class "<<endl;
  }
};
void main()
{
  Derived object;
  object.print(); //calls the function from the Derived class
  object.Base::print(); //calls the function from the Base class
}
```