Laboratory lesson №3
(The spring session)

Class constructors. Class destructor.

Constructor is a method that runs automatically when an object is created. It can not only initialize class variables, but also perform any other initialization tasks necessary to prepare the object for use. A constructor must be a method of type *public*.

The main constructor properties:

- ✓ The constructor name **is the name of the class.** So, the compiler distinguishes the constructor from other class methods.
- ✓ The constructor **does not return any value**, even of type void. The constructor is automatically called by the system, and therefore, there is no program or function that calls it, and to which the constructor could return a value.
- ✓ A class can have several constructors with different parameters for different types of initialization, and an overload mechanism is used.
- ✓ The constructor can accept any number of arguments (including zero).
- ✓ Construcor parameters can be of any type except this class. You can set default parameter values.
- ✓ Constructors are NOT inherited.
- ✓ Constructors cannot be declared with *const*, *virtual*, and *static* modifiers.

Constructor types:

- ➢ Constructor with parameters
- ➢ Constructor with initialization list.
- ➢ Default constructor.
- ➢ Copy constructor.

Constructor with parameters

The constructor can initialize the class fields through parameters, for example

```
someClass (int mm1, int mm2, int mm3)
{M1 = mm1; m2 = mm2; m3 = mm3;}
```

In this case, the creation of the object may take the form

```
someClass obj (5, 20, 13);
```

The constructor is called here, into which the parameter values are passed. In addition, when creating an instance of the class, parameter values can be passed to the constructor using the ***new*** operator, for example:

```
someClass * Pobj = new someClass (5, 20, 13);
```

The constructor may have default parameters that are passed automatically, for example, to class fields, when creating class objects. If the values of the constructor parameters are not specified, the object is initialized with the default parameter values.

```
someClass (int mm1=1, int mm2=1, int mm3=1)
{ m1 = mm1; m2 = mm2; m3 = mm3;}
someClass obj;
```

Constructor with initialization list.

The constructor described above cannot be used for initialization of constant fields or reference fields, because the values cannot be assigned to them. There is a special property of the constructor called the initialization list, which allows you to initialize one or more variables and not give them values. It can be made in such:

```
someClass(): m1(0), m2(10), m3(15) {}
```

The field objects can be initialized using the list of initializers, where values can be expressions, for example:

```
someClass (int mm1, int mm2, int mm3): m1 (mm1), m2 (mm2),
m3 (mm3) {}
```

In the first case, the constructor is called when the object is created:

```
someClass obj1, obj2;
```

In the second case, when creating the *obj* object, the constructor with the corresponding parameters indicated in parentheses will be called, for example:

```
someClass obj(5,20,13);
```

Default constructor

The constructor may have default parameters that are passed automatically, for example, to class fields, when creating class objects. If the values the constructor parameters are not specified, the object is initialized with default parameter values.

```
someClass (int mm1 = 1, int mm2 = 1, int mm3 = 1)
{M1 = mm1; m2 = mm2; m3 = mm3;}
someClass obj;
```

This constructor is called the **default constructor**.

A constructor without parameters is also called the default constructor. If the constructor for any class is not defined, then the compiler itself generates a default constructor. Such constructors do not assign initial values to the class fields; it defines a class object without passing parameters, for example:

```
someClass obj1, obj2;
```

Each class can include only one default constructor.

If any constructor is defined in the class, the compiler will not create a default constructor.

Copy constructor

Now consider another way to initialize an object that uses the field values of an existing object. Such a constructor is created by the compiler for each class and is called ***the default copy constructor***. It has only one argument that is an object of the same class.

For example, let’s create three objects t1, t2, t3 in different ways:

```
void main ()
{ Interval t1(2, 30);
Interval t2(t1);
Interval t3 = t1;
cout << "\nt1 = "; t1.showinterval();
cout << "\nt2 = "; t2.showinterval();
cout << "\nt3 = "; t3.showinterval();
cout << endl;
}
```

The result of it:

```
t1 = 2 год. 30 хв.
t2 = 2 год. 30 хв.
```

t3 = 2 год. 30 хв.

```
If the class contains pointers or references, the calling the copy constructor
will cause the situation, when both the copy and the original  point to the same
piece of memory. In this case, the copy constructor should be created by the
programmer and have the form
```

```
T :: T(const T&)
```

where T – class name.

For example:

```
someClass :: someClass (const someClass &D) { . . . }
```

Destructor

A **destructor** is a special form of a method that is used to free the memory occupied by an object. The destructor is essentially the antipode of the constructor. It is called automatically when an object goes out of scope.

The name of the destructor begins with a tilde (~), immediately followed by the class name. The destructor has the following format:

```
~ <Class Name> () {}
```

The destructor has no arguments and the value that is returned. But it can perform some actions, for example, outputting the final values of data elements of a class, it is convenient when debugging a program.

If the destructor is not explicitly defined, the compiler automatically creates an empty destructor.

It is necessary to place the destructor in the class if the object contains pointers to memory that is allocated dynamically. The pointer to the destructor cannot be determined. The destructor is not inherited.

Example: Try the use of constructors and a destructor using the example of the “Worker Company class” (**lab.lesson №3**).

Add to the methods of the **Employee class**:

- default constructor;
- constructor with parameters, including default parameters;
- constructor with an initialization list;
- copy constructor;
- destructor;

These methods are in the *public* section.

Here are the representations of the class being developed in UML:

Employee
- name: string - age: int - position: string - salary: double
+ Employee() + Employee (n:string, a:int, s:double) + Employee (n:string, a:int) + Employee (&src: const Employee) ~ Employee() + setName(n: string) + getName(): string + setAge(a: int) + getAge(): int + setSalary(s:double) + getSalary(): double + showEmployee()

Here is an example of a code for the declaration and implementation of the “Work” class:

Header file **Employee.h**

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <iostream>
#include <string>

using namespace std;
class Employee
{
    string name;
    int age;
    string position;
    double salary;

public:
    Employee();//default constructor
    Employee (string n, int a, double s);// constructor with
//parameters
    Employee (string n, int a=18);// constructor with parameters,
including default parameters
    Employee (const Employee &src);//copy constructor
    ~Employee();// Destructor
    void setName(string n);
    string getName();
    void setAge(int s);
    int getAge();
    void setSalary(double s);
    double getSalary();
    void showEmployee();
    };
    #endif // EMPLOYEE_H
```

File with class implementation **Employee.cpp**

```
#include <iostream>
#include "Employee.h"
#include <string>

 using namespace std;

//The default constructor
  Employee::Employee():name("N/A"),age(0),salary(0.0)
    {
      cout<<"Class instance was created by default constructor\n";
    }
// constructor with parameters
  Employee::Employee (string n, int a, double s)
    {
      name=n; age=a;
      if (s< 50000.0)
      salary = s;
     else // Invalid salary
      salary = 0.0;
      cout<<"<<"Class instance was created by default constructor with parameters\n";
    }
// constructor with parameters, with an initialization list
  Employee::Employee (string n, int a):name(n),age(a)
    {
        do  {
        cout << "Input salary " << name << "< $50000: ";
        cin >> salary;
    }
        while (salary >= 50000.0);
        cout<<" Class instance was created by default constructor with parameters  \n";
    }
  Employee::Employee (const Employee &src)//Copy constructor
    {
       cout<<" Copy constructor \n";
        name=src.name;
        age=src.age;
        salary=src.salary;
    }
  Employee::~Employee()// Destructor
    {
        cout<<" The Destructor is working\n";
    }
  void Employee::setName(string n)
```

```
    {
     name=n;
    }
  string Employee::getName()
    {
     return name;
    }
  void Employee::setAge(int s)
    {
     age=s;
    }
  int Employee::getAge()
    {
     return age;
    }
  void Employee::setSalary(double s)
    {
     salary = s;
    }
  double Employee::getSalary()
    {
     return salary;
    }
  void Employee::showEmployee()
    {
     cout<<"Employee: "<<name<<"\t"<<"Age: "<<age<<"\t"<<"Salary:"
<<salary<<endl;
    }
```

The file **main.cpp**

```
#include <iostream>
#include "Employee.h"

using namespace std;

int main()
{
    Employee emp1;
    emp1.showEmployee();
    Employee emp2("Ivan", 35,100000.0);
    emp2.showEmployee();
    Employee emp3("Andrew",18);
    emp3.showEmployee();
    Employee department[5]={Employee("A",30), Employee("B",25),
    Employee("C",61)};
     for (int i=0;i<5;i++)
     {
         department[i].showEmployee();
     }
     Employee emp4=emp3;
     emp4.showEmployee();
    return 0;
}
```

Laboratory work №3.

Task progress:

1. Create the class for a given concept.
   - ✓ define class attributes and their data types;
   - ✓ define class methods
   - ✓ To the class methods add:

     -default constructor, with parameters, with an initialization list;

     -destructor.
2. Describe the class in UML notation.
3. Write a program in which the user will be able to manipulate the created class. Organize work with an array of class instances.

Варіанти завдань.

1. Data;
2. Train;
3. Triangle;
4. Auto;
5. Book.