Object-Oriented progtamming using C++

ШАГ
компьютерная
АКАДЕМИЯ

Lesson N1

Object-Oriented Programming Using C++

Contents

Basics of object-oriented programming

Hello again!!! We hope that you have successfully passed the exam on programming in C, and now you are ready to move on to the next stage of learning, namely, programming in C++. First of all, let us recall some historical facts about these two programming languages. You have already received this information in the course of the first lesson:

In 1972, Dennis Ritchie, a 31-year-old specialist in system programming of Bell Labs, developed the C programming language. For the first time the language was described in the book of B. Kernighan and D. Ritchie. It has been considered a standard for a long time, but a number of points were interpreted in an ambiguous manner. This gave rise to many interpretations of C. In order to remedy the situation, American National Standards Institute (ANSI) formed a committee to establish a standard specification of C. In 1983, the standard of C has been approved and is known as ANSI C. In the early 80-ies, Bjarne Stroustrup added and elaborated C in the same Bell Laboratory. The result was the invention of new language called "C with Classes". In 1983, this name was changed to C++.

Have you remembered that? However, the history of languages is not the only difference between them :)))). The thing is that there are two fundamental approaches to programming: **procedural and object-oriented**.

Let us consider their definitions:

Procedural programming is an approach, where functions and variables related to any particular object and divorced from each other are freely located in the code (The C programming language).

Object-oriented programming is an approach, where functions and variables related to any particular object are united in the code in a certain way and are closely interlinked (The C++ programming language).

So, starting from today, we will deal with object-oriented programming. Its abbreviated form is OOP.

OOP is a concept that has revolutionized programming. OOP assumes that application is based on a set of parts characterized by independent internal arrangement. OOP rests on three primary principles, which are to be considered now.

"The first pillar" — Encapsulation

Encapsulation is the principle of data independence in OOP. Thus, each part can contain its own data, inaccessible to other parts of the system. It is apparent that these parts cannot be absolutely independent, because they need to interact with each other, use common data and share their own data.

A simple example is a living cellular organism. Each cell is characterized by its own behavior and structure, but it also interacts with other cells and shares substances with them. In programming, such a living organism is an application, cell is an object, substances are the data, and the ways of interaction are methods (functions) and events.

Let's try to define an object:

Object is a unique unit having its own variables and functions processing these variables.

In order to create an object, you must define a data type that will be used in the program and each variable of this type (instance) will be an object. In general, structure data type can be an object.

We hope you still remember what it is. Structure gathers all the data relating to the same object under a single name that is a user-defined data type. In C++, this type of data is called a **CLASS**.

Imagine a black box. In the context of programming, it is an object. However, if the box is an object, what is a class? In this case, class is a drawing according which the box is constructed. Now let us think about this black box. It is obvious that it has certain characteristics. The first is its shape. The second is its content and open/close mechanism by means of which the content is extracted from the box.

In programming, characteristics are called properties. The ways of extracting the content applicable to this box are methods. Properties are used to work with internal data, read and set values. Methods are the actions, which can be executed by the object. In this case, when clicking on the button available on the box that is used to "call the method" meaning to open the box, the box opens automatically. In other words, method is a function belonging to and executed by the object.

So, let’s summarize: encapsulation is a mechanism by means of which all the properties and methods related to one particular object are brought together under the common name (type) that is called a class.

"The second pillar" — Inheritance

Let’s start with definition of this term:

Inheritance is a process, when an object may inherit methods and properties of another object in addition to its own properties and methods.

Let's continue analyzing black box. Assume that we are going to create a colorful gift wrapping on the basis of the black box. In order to do this, we simply add those features typical for a gift wrapping to the black box. For example, one of them is color. Such features as shape and size, as well as the ability to open and close are similar for both the black box and the colored one. Therefore, it makes no sense to describe them again; it suffices to inherit "the colored box" of "the black box". This is the main advantage of using inheritance. We start with creation of a simple structure and add new properties and methods to it. The result is a new enhanced object.

"The third pillar" — Polymorphism

Polymorphism is the ability of object to behave in a different manner depending on the situation and respond to action in a specific way.

Let's consider a common case. You come to the store to buy some product. You choose it. If the seller is your acquaintance, he/she can give an advice whether to buy this product or not. If the seller is a perfect stranger, he/she will indifferently cut off a piece of product. The seller's behavior depends on the situation. The object behaves in a similar way. Independent data type responds to peripheral distractions in various ways. Now it is a bit difficult for you to understand how this will happen. However, the principle of polymorphism is already known to you (remember overloaded functions). Function has called some of its options depending on the parameters passed to it.

Thus, the theoretical part is over. It's time to move on to practice.

Introduction to classes

Defining the class and creating the object

Class is a derivative structured type introduced by the programmer based on the existing types. In other words, the class technique defines a structured set of typed data and allows defining a set of data operations.

General class syntax can be determined by the structure:

```
class_name {component_list};
```

1. **Class_name** — a randomly selected identifier
2. **Component_list** — definitions and descriptions of typed data and functions belonging to the class. Class components may include data, functions, classes, enumerations, bit fields and type names. For the sake of simplicity we assume that class components are data types (basic and derived) and functions.
3. Component list in curly braces is called a class body.
4. Header is preliminary to the class body. In the simplest case, header includes the word "class" and name.
5. Class definition always ends with a semicolon.

So, functions belonging to the class are called **class methods** *or* **component functions**. Class data are called **component data** or **class data elements.**

Let's go back to the definition: class is a type introduced by the programmer. Since each type is used to define objects, let's define the syntax of creation of the class object.

```
class_name object_name;
```

Defining a class object assumes providing the storage area and dividing it into fragments corresponding to the individual object elements, either of which displays a separate component of class data.

Class component access technique

There are several levels of accessing class components. We are going to consider two basic levels.

public — class members are externally accessible.
private — class members are not externally accessible.

Let's interpret this.

By default, all variables and functions belonging to the class are defined as closed (private). This means that they can be used within member functions of the class only. For other parts of the program, such as the main() function, private members are inaccessible. *By the way, this is the only difference between the class and structure. All the members of the structure are public by default.*

Using public access specifier, we can create public class member accessible for all the program functions (both inside and outside the class).

Syntax for accessing the data of a particular object of a given class (just as in case of structures) is the following:

```
object_name.class member_name;
```

It is time to consider an example...

```
#include <iostream>
using namespace std;
class Test{
     //since access specifier is unknown
     //by default this variable will not be
     //externally accessible (private)
     int one;
     //public access specifier
     //all members following it
     //will be externally accessible

public:
     //initializing variables in the class
     //in the process of creation is prohibited,
     //so we define
     //the method implementing the following action
     void Initial(int o,int t){
          one=o;
          two=t;
     }
     //method showing the class variables
     //in display form
     void Show(){
     cout<<"\n\n"<<one<<"\t"<<two<<"\n\n";
     }
     int two;
};

void main(){

     //Test type object is created
     Test obj;
     //the function initializing its properties is called
     obj.Initial(2,5);

     //displaying
     obj.Show(); // 2 5
     //direct entry to the "two" public variable
     //in case of the "one" variable, such record
     //is impossible, because
     //it is not accessible
     obj.two=45;

     //displaying again
     obj.Show(); //2 45
}
```

The above example is intuitively simple, but we should note one of the principal points: **there is no need to pass class variables to class methods in the capacity of parameters, as they are already visible in them.**

Constructors and destructors

Initialization of class objects. Constructors

Sometimes, when creating an object, it is necessary to assign initial values to its elements. As you already know, you cannot do this directly in the class definition. This problem is solved by writing a function that will assign initial values to class variables and calling this function each time after object declaration. We have successfully done this in the example of the previous section of the lesson.

However, C++ has a technique providing another way of solving this problem. It includes constructors.

Constructor is a special class member function declared under the name that is similar to the class name. Constructor does not assume determination of the type of return value. EVEN void!!! Let's consider an example of creating a constructor:

```
#include <iostream>
using namespace std;
class Test{
     //since access specifier is unknown
     //by default this variable will not be
     //externally accessible (private)
     int one;

     //public access specifier
     //all members following it
```

```
	//will be externally accessible

public:
	Test(){
		one=0;
		two=0;
	}

	//initializing variables in the class
	//in the process of creation is prohibited,
	//so we define
	//the method implementing this action
	void Initial(int o,int t){
		one=o;
		two=t;
	}

	//method showing the class variables
	//in display form
	void Show(){
		cout<<"\n\n"<<one<<"\t"<<two<<"\n\n";
	}
	int two;
};

void main(){

	//Test type object is created
	Test obj; (constructor is activated)

	//displaying
	obj.Show(); // 0 0

	//the function initializing its properties is
	//called
	obj.Initial(2,5);

	//displaying
```

```
	obj.Show(); //2 5
	//direct entry to the "two" public variable
	//in case of the "one" variable, such record
	//is impossible, because
	//it is not accessible
	obj.two=45;
	//displaying again
	obj.Show(); //2 45
}
```

1. The constructor is called automatically when an object is created, i.e. it is not necessary to call it.
2. Main purpose of constructors is object initialization.
3. Constructor must always be public!!!

Using the parameters, any data necessary for the class object initialization can be passed to constructor.

Example. A class defining a point

```
#include <iostream>
using namespace std;

//the Point class description
class Point {
	int x, y;
	//by default, coordinates of the point have
	//private access level

public:
	//constructor assigns primary values to the
	//variables of the x and y classes
	//x0 and y0, respectively
	Point(int x0, int y0)
	{
		x = x0;
		y = y0;
	}
```

```
     //the function of displaying coordinates
     //of the point

     void ShowPoint()
     {
          cout << "\nx = " << x;
          cout << "\ny = " << y;
      }
};

void main()
{
     Point A(1,3); //creating A point (object
     //of the Point type) with coordinates x = 1, y = 3
     //(Point(1, 3) constructor is called)
     A.ShowPoint(); //displaying the coordinates
     //of the A point
}
```

Note: *When creating an object, the values of parameters are passed to the constructor using a syntax that is similar to the regular function call.*

More about constructors...

1. Constructor with no parameters is called a default constructor. Usually, such a constructor assigns the most often used values to member variables of the class.

```
Point()
{
     x = 0;
     y = 0;
}
```

2. Each class can include one default constructor only.

3. If a class has no constructor defined, compiler creates a default constructor. This constructor does not set any initial values, it is just there :))).

Destructors

Destructor executes the function opposite to the function of the constructor.

Destructor is a special class function that is automatically called when destroying an object, for example, if an object is out of scope.

Destructor can complete any task when removing the object.

For example, if the constructor has a dynamically allocated memory, destructor must free this memory before removing the class object.

Basic features of working with a destructor

1. Destructor accepts no parameters and returns no values.
2. A class can have one destructor only.

And finally, let's consider the example of sequence of calling constructors and destructors.

```
#include <iostream>
using namespace std;

//the CreateAndDestroy class description
class CreateAndDestroy
{
public:
     CreateAndDestroy(int value) //constructor
     {
         data = value;
```

```
          cout << " Object " << data << " constructor";
     }

     ~CreateAndDestroy() //destructor
     {
          cout << " Object " << data << "
             destructor" << endl;
     }
private:
     int data;
};

void main ()
{
     CreateAndDestroy one(1);
     CreateAndDestroy two(2);
}
PROGRAM OUTPUT

Object 1 constructor
Object 2 constructor
Object 2 destructor
Object 1 destructor
```

This example shows that destructors are called in sequence that is opposite to calling the constructors.

Homework assignment

1. Digital counter is a variable with limited range. Its value is reset when its integer value reaches a certain maximum (for example, k is a value ranging from 0 to 100). The example of such a counter is a digital clock or odometer. Describe the class of such a counter. Provide a possibility to set maximum and minimum values, increasing the counter by 1, returning the current value.

2. Write a class describing a group of students. Student is also implemented by means of the corresponding class.