Introducing C++ classes

Class - This is an abstract construction for creating objects.
Class definition:

class *class_name*
{
    private data and functions
public:
    public data and functions
} [*object name list*] ;

The list of objects can be omitted. class_name becomes the new data type and is used for declaring class objects. Variables and functions that are declared inside a class are called members of the class. Functions declared inside a class are also called class methods.

By default, all class members are private. This means that direct access to these components is forbidden. This is how data and functions are encapsulated.

The keyword public means that the relevant variables and functions are public and accessible directly.

Przykład W01.

```
//class definition
class MY_CLASS
{
    int i;                  //private variable
    public:
    int k;                  //public variable
    int get_i();            //public method - returns the value of a
                            //private variable i
    void set_i(int ii); //public method - sets the value of a
                            //private variable i
    void disp();            //public method - outputs all members of /
                            //class
};
```

Definitions of class methods:

Return type CLASS NAME :: function_name(formal_argument_list)

:: - range operator

```
int k;     //global variable

void fun()
{
        int k;
        k = 10;         // reference to a local variable
        :: k = 10;      // reference to a global variable
}
```

The code would be more readable if different names were used for local and global variables!

```
void MY_CLASS::disp()
{
    //access to private data as well as public data is allowed
    //inside the class
    cout << "obiekt MY_CLASS:" << endl;
    cout << "private  i = "  << i << endl;
    cout << "public   k = "  << k << endl;
}
```

Function disp() returns nothing, is in the scope of MY_CLASS class, and has an empty formal argument list.

Access to public members of a class object (function, data):

class_name object_name; // declarations of the class object

object_name.function_name (list of actual_arguments);
object_name.variable_name = ……;

A class definition defines a new data type, it is a logical abstraction.
Object declaration creates the object itself - the memory allocation is produced.

Each class object has its own instance of all variables declared inside the class.

MY_CLASS ob1, ob2;

Data of objects ob1, ob2 are located in different memory addresses, so ob1.k and ob2.k - these are different variables.

Example W5:

Function Overloading

In C++, many different functions can have the same name as long as they have a different argument list (different quantity or types, quantity and types). We call such functions overloaded functions, and the procedure for creating them is called function overloads.

Example 1: W6

Determining which of the two specified functions with the same name is to be called is carried out at the compilation stage. The compiler makes this decision based on the data type of the actual argument.

A function overload gives you the ability to create a set of functions with the same name. Logically, these functions must perform a general action - retrieving the absolute value, after all, due to different data types, they are different functions.

Example 2

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

void f1(int a);
void f1(int a, int b);

int main()
{
    f1(10);
    f1(10, 20);
    return 0;
}

void f1(int a) {
    cout << " Funkcje f1(int a)" << endl;  }

void f1(int a, int b) {
     cout << " Funkcje f1(int a, int b)" << endl; }
```

If the difference between two (or more) functions with the same name applies only to types that return these functions (the formal argument list is the same), the compiler will not be able to distinguish which function to invoke.

```
//error !!!

int f1(int a);
double f1(int a);
.....................
f1(10);      //which of these functions to call?
```

In C, you can overload not only functions but also operators. We will consider operator overloading later

Constructors and Destructors

Often times, some part of an object needs to be initialized before it can be used. In C++, constructor – a special member-function - is automatically called to initialize an object during creation.
The name of constructor is the same as the class name.

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


class myclass
{
    int  a;
 public:
    myclass();      //constructor
    void show();
};
```

```
myclass::myclass()
{
    cout  << “Constructor\n”;
    a = 10;
}

void myclass::show()
{
    cout  <<  “ a = ” << a << endl;
}

int main()
{
     myclass ob;
     ob.show();
     return 0;
}
```

Output:
Constructor
a = 10

- The constructor is called when creating an object, so when declaring an object, ob. In C++, statements that make a declaration can trigger actions - the variable you declare can have a constructor.

- The constructor returns no value.

- For global and static objects, the constructor is called only once.

Destructors

- Destructor is used to perform a series of actions when destroying an object.

- Local objects are created at the entrance to the block containing them and destroyed when leaving it.

- Global and local static objects are destroyed when the program exits.

- Destructor (if any) is automatically called when the object is destroyed.

- Most often, destructors are used when dynamically allocated memory is freed, when files are closed.

- The destructor has the same name as the constructor, only preceded by the symbol ~ .

- Impossible to get the address of neither the constructor nor the destructor.

- Neither the constructor nor the destructor call explicitly

Example W7
Example W8

Parameterized Constructors

Arguments can be passed to the constructor. To create a parameterized constructor, simply complete it with parameters, just like you do with other functions. When declaring an object, parameters (formal arguments) must be replaced with actual arguments.

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

class myclass
{
        int a;
 public:
        myclass(int x);  //parameterized constructor
        void show();
}

myclass::myclass(int x)
{
        a = x; //initiation of the private member of class using the formal argument x
}
```

```
void myclass:: show()
{
        cout << ” a =  ” << a  <<  endl;
}

int main()
{
        //creation of object ob and passing the actual argument – constant 4 – to constructor.
        myclass ob(4);
        ob.show();     //output of myclass members.

        myclass t(12), rtu(25);
        t.show();
        rtu.show();

        return 0;
}
```

Output:

```
a = 4
a = 12
a  = 25
```

Attention! These are different "a"! The first a - is ob::a - a member of the ob object. The second a - is a member of the t object (t::a), and the third - the rtu object (rtu::a).

The *myclass ob*(4) statement creates an object named *ob* and passes the current argument 4 to the formal argument of the *myclass* constructor. When creating the object, the constructor of the class is called, and the private component named *a* will be assigned the value 4 (initializing the component *a*).

The *myclass ob*(4) statement is an abbreviation of

myclass ob = myclass(4);

In the C++ language the first form of writing is adopted, and we will continue to use only the abbreviation form

myclass ob(4);

The destructor cannot have parameters for differences from the constructor - there is no mechanism for passing arguments to the object that is destroyed.

The list of parameters (formal arguments) of the constructor may contain any number of parameters. For example:

```
class myclass
{
        int a;
        double b;
        char str[256];
 public:
        myclass(int ii, double db, char *sss);
        ……
};

myclass::myclass(int ii, double db, char *sss)
{
        a = ii;
        b = db;
        if(strcpy_s(str, sizeof(str), sss))
        {
            //error handling
        }
}
```

Or:

```
myclass::myclass(int ii, double db, char *sss) : a(ii), b(db)
{
        if(strcpy_s(str, sizeof(str), sss))
        {
            //error handling
        }
}
```

```
int main()
{
        int i_a = - 24;
        double d_pi = 3.141593;
        char strr[] = ”Ann has a cat”;

        myclass tt(i_a, d_pi, strr);
        ……
        return 0;
}
```

A class can have several constructors - constructors overloaded:

```
class myclass
{
        int  a;
 public:
        myclass() : a(0) { }          //constructor with an empty list of arguments (default constructor)
        myclass(int ii) : a(ii) { }   //parameterized constructor
        void set_a(int ii) { a = ii; }
        void  show();
};
```

W2

```
void myclass::show()
{
        cout  << ”a = ”  <<  a  <<  endl;
}

int main()
{
        myclass ob, tt(25);       // ob.a = 0        tt.a    = 25

        tt.show();
        ob.show();

        ob.set_a(35);              //ob.a = 35
        ob.show();

        return 0;
}
```

Output:

```
25
 0
35
```

Copy constructor

- Is used when one object initializes another:
  - myclass A = B
  - passing an object to the function (fun(A))
  - returning an object with the function (myclass A = func())
- By default, if you do not define a copy constructor, a bitwise copy is created in these cases. This can cause side effects.
- The copy constructor is used automatically (only at the time of initialization) by the compiler when an object is used to initialize another object. Then the default bitwise backup is ignored.

```
class_name (const class_name &ob)
{
        //the body of copy constructor
}
```

ob is a reference to the object on the right side of the initialization statement.

The copy constructor can have additional parameters, provided that *default arguments* are defined for them. For the first parameter, a reference to the object must be.

```
NODE_COORD(const NODE_COORD &ob, int i = 0) ;
```

Attention!
In the case:

```
myclass A(10);
myclass B(20);
B = A;
```

The copy constructor is not called, because it is an assignment, but not initialization, and there needs to overload the “=” operator.

```
myclass C = A;    //It is an initialization by right hand side object A

myclass fun();
myclass A = fun();   //It is an initialization by function returned value.
myclass B;
B = fun();           //It is not initialization, it is an assignment.
```

Example W9: copy constructor is called when object is passed to function.

Example W10: copy constructor is called when function returns an object.