C++

Dr. Hab. Eng. Sergiy Fialko,

http://torus.uck.pk.edu.pl/~fialko
sfialko@riad.pk.edu.pl

Literature

1. **B. Stroustrup. The C++ Programming Language (Fourth Edition). May 2013, Addison Wesley. Reading Mass. USA. May 2013.**

2. **Herbert Schildt. The Complete Reference C++. 2020, Tata McGraw Hill.**
https://www.allabout-engineering.com/download-the-complete-reference-c-by-herbert-schildt/

3. **C++ language documentation. MSDN.**
https://docs.microsoft.com/en-us/cpp/cpp/?view=msvc-160

W1 Object-Oriented Programming (OOP)

number of lines in code (nlc)	Small nlc < 10 000	Medium 10 000 < nlc < 100 000	Large nlc > 100 000
Type of programming language	assembler, FORTRAN	Structured languages C, Pascal	Object-oriented languages C++, Java

Structured languages: „code acting on data” – code operate on any type of data used by the program.

OOL (object-oriented languages): "data controlling access to code„ – data type defines precisely what sort of operations can be applied to that data.

Foundational principals of object-oriented languages.

- **Encapsulation**
- **Polymorphism**
- **Inheritance**

Object-Oriented Programming (OOP)

- **Encapsulation**

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference. When code and data are linked together in this fashion, an ***object*** is created.

Within an object, it functions (methods of object) and data, or both may be *private* to that object or *public*.

- Private methods can be called only by other methods of this object. Private data can be processed only by methods (private or public) of this object. Private methods cannot be called from outside of the object, and private data is not accessible directly from outside of the code.
- Public methods can be called from both: methods of given object and outside of the code. Public data is accessible from the methods of the given object as well as from outside of the code.
- Private data can be accessed outside of the object only by using public methods.

Object-Oriented Programming (OOP)

Example 1 (W1).

The question arises: why complicate access to the members of an object so much by introducing the access specifier private?

Let us consider a large and complex program with hundreds of thousands of lines of code. Now try to check where the values of the variables ii, jj of the MY_CLASS object change.

If it is a *public* variable, you need to search through a huge code for all fragments of the id_object.ii = type.

If it is a *private* variable, its value can only be changed by calling a function of type Set_j. We set a breakpoint in the Set_j function and run the task under the debugger. Now we control all occurrences of the Set_j function from a huge part of the code that is not related to the maintenance of our object. **(W1)**

Polymorphism – “one interface, multiple methods”.

Allows the same piece of code (uniform interface) to control access to different actions (multiple methods).

Distinguish between static and dynamic polymorphism.

- Static polymorphism is about the **function and operator overloading** and is done at compile stage.

- Dynamic polymorphism is based on using **virtual functions** and is implemented at runtime. The decision about which of the many virtual functions should be called is made at runtime.

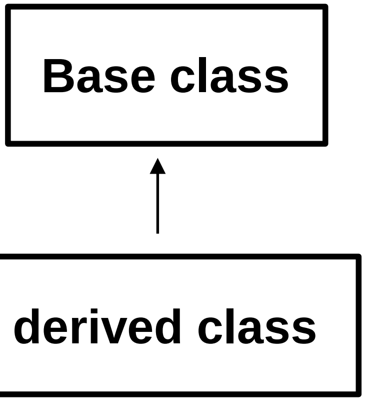
W1

Inheritance - is the process by which one object can acquire the properties of another object.

If the data and actions on it can be organized hierarchically, then it is very convenient at each level of the hierarchy to create only the maintenance of the data that belongs to this level.

This programming technique makes it easier to understand the program and reduces the development time.

Example W2.



Simple sample C++

```
 # include <iostream>   //C++ header (stdio.h)
 using namespace std;

int main()
{
         int i;
        cout  << “hello, C++” << “\n”;   //single line comment
         /* You can use C style comments as well*/

         //input number using >> operator
         cin >> i;   //formatting is produced as default

         //Output using << operator
         cout  <<  ”i = ” << i << ”i *i = ” << i*i << ”\n ” ;

         return 0;
}
```

W1

```
# include <iostream>
```

It is an analog C - #include <stdio.h>. Standard C++ uses a new style which does not require **.h**. The new type headers used in C++ may not be header files, they are abstract constructions which guarantee that the appropriate prototypes and definitions required by the C++ libraries will be declared. For instance,
<vector>   <fstream> <string>

```
using namespace std;
```

The namespace *std* will be used. A namespace creates the declarative region (scope), in which many program elements can be placed. Namespaces help us in organization of large programs.

Statement *using* grants the namespace *std*, in which is declared Standard C++ library (CRT).

C++ has an output operator << as well as input operator >> . C++ compiler depending of context is possible to differ an insertion (output) operator << from the left shift binary operator << as well as the extraction (input) operator >> from right shift binary operator >> .

W1

Insertion to stream cout:
```
cout << a;
```
Extraction from stream cin:
```
cin >> b;
```

Standard stream devices:

cin – input device connected with input stream (keybord → OS driver → stream cin)

cout – output device connected with output stream (api → cout stream → OS driver → screen)

cerr – output device connected with output stream designed for errors.

Statement cout >> object; does not required explicitly shows the format specification, because acts the default formatting.

Exactly the same, statement cin >> object; does not require explicitly formatting too.

To change the default formatting, needs to use the manipulators or the members of *ios* class. We will consider this later.

W1

Examples:

```
double a = 0.3;
int i = 15;
char ch = 'a';
char str[] = “abcde”;

cout << “ a =  “ << a << “ i = “ << i << “ ch = ” << ch << “ str: “ << str << endl;

double b;
cout << “input date: “ << endl;
cin >> b;
..........................................................................
```

Example W3

When entering text lines, the >> operator stops after detecting the first space. Only the first word will be entered, the remainder of the line will not. This works exactly the same as input a text line with scanf (“% s”, str);

Declaration of the local variables

In C, local variables must be declared at the beginning of a block. In C++, local variables can be declared anywhere in the block.

It is advisable to declare variables not at the beginning of the block, only in cases where it is justified, for example, in functions with many lines of code.

No default conversion to *int*

In C, if functions return variables whose type is not explicitly declared, by default the type of that variable is treated as *int*. In the C++ language, the type returned by functions must be declared explicitly.

Type bool

In C++ there is a built-in logical type - *bool*. The *bool* type object can only store *true* and *false* values, which are keywords defined in C++. Values other than 0 are converted to *true*, and zero is converted to *false*. Conversely, *true* is changed to 1 and *false* is changed to 0.

Namespaces

Namespace is a declarative region. Namespaces are used to identify names and avoid collisions. Elements declared in one namespace are separated from elements declared in another.

W1

```
namespace MY_NAMESPACE_1
{
    int i;
    void fun(int j, char *str);
};

namespace MY_NAMESPACE_2
{
    int i;
    void fun(int j, char *str);
};

void MY_NAMESPACE_1::fun(int j, char *str)
{
    cout << “j =  “ << j << “ str:  “ << str << endl;
}

void MY_NAMESPACE_2::fun(int j, char *str)
{
    cout << “j*j =  “ << j*j << “ str:  “ << str << “ !!!” << endl;
}
```

W1

```
int i;                                          //belongs to the global namespace (scope)
void fun(int j, char *str)
{
   cout << “j*j*j = “ << j*j*j << “str: “ << “[ “ << str << “] “ << endl;
}

int main()
{
   MY_NAMESPACE_1::i = 1;        //belongs to namespace  MY_NAMESPACE_1
   MY_NAMESPACE_2::i = 2;        // belongs to namespace MY_NAMESPACE_2
...........................................................................................................
// These are different variables - they are located at different memory locations
// For this, no name collisions arises;    ::   - scope resolution operator (range operator).

MY_NAMESPACE_1::fun(10, “abcd”);  //call the function from scope MY_NAMESPACE_1

MY_NAMESPACE_2::fun(15, “efgk”);  // call the function from scope MY_NAMESPACE_2

fun(20, “tunm”);  // call the function from global namespace
```

W1

```
#include <iostream>

using namespce std;  // makes std available throughout the entire file
. . . . . . . . . . . . . . . . . . .

void fun(list of arguments)
{
        using namespace MY_NAMESPACE_1;  //makes MY_NAMESPACE_1
                                         //available in this block
         . . . . . . . . . . . . . . . . . . .
}

void fun_second()
{
        MY_NAMESPACE_2::i = 10;      //makes member MY_NAMESPACE_1::i
                                     //available only in this block
        . . . . . . . . . . . . . . . . . . .
}
```