PROGRAMMING TRAINING

Hansei Technology

By : Badea Robert

CONTENTS:

CHAPTER 1: Introduction

In our team, the robot's code is written in Java, but unfortunately, we are not very proficient in this programming language. Therefore, we will start with C++ since it is simpler for us, as well as for you, as this is what is taught in high school.

C++:

In the first part of this training, you will learn the basics of programming. Although we will cover them in C++, these fundamentals work the same way in any programming language. After mastering the basics and understanding the core concepts, we will move on to Java, where you will directly program robots, and we won't spend too much time on theory.

Why aren't we programing the robot in C++?

C++ is an old programming language (created in 1985) and is still used today for creating operating systems (an operating system cannot be created in Java; they can only function in C, C++, and Pascal). Unfortunately, just as C++ is good for certain things, so is Java. Even though all programming languages might seem similar, they function differently, and different projects require different languages.

How will the training work?

This course is divided into 2 sections: C++ and Java. Each of them will have multiple lessons, and especially for the C++ section, you will also receive assignments. Assuming you are here because you want to be and not because you're forced to, you'll need to complete the assignments as you see fit. However, you must ensure that you understand each lesson (I will ask you questions from assignments and previous lessons to make sure you've understood; if you're confident about a lesson, you don't have to implement all exercises, but you should at least read them).

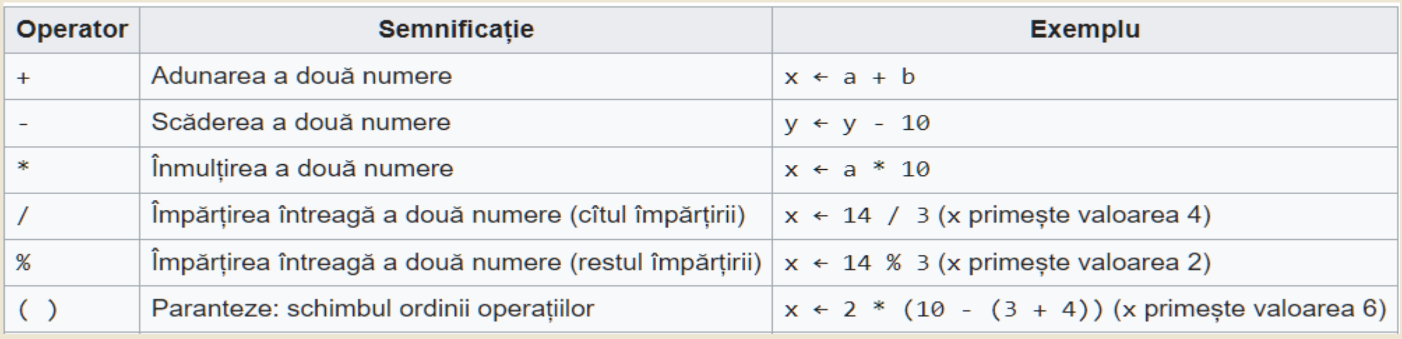
My colleague Radu and I will conduct the lessons, which will be recorded as videos. If you need further explanations for any of the lessons, you can watch the recordings.

You will need to install an IDE (a program where you compile your written code). You can use any IDE you want, but my recommendations are: **Visual Studio Community**, **Visual Studio**, or **CodeBlocks**. Additionally, you will need to create an account on pbInfo, which is a website where you can find problems and theory for C++. If you need more materials, I suggest:

- https://www.pbinfo.ro/articole/5547/informatica-clasa-a-ix-a
- https://www.algopedia.ro/wiki/index.php/Cercul_de_informatic%C4%83,_IQ_Academy,_clasa_a_V-a,_anul_2019-2020
- Chat GPT is your best friend :)

CHAPTER 2: Variabile

A variable stores different types of data and enables the execution of operations using this data.



Operator	Semnificație	Exemplu
+	Adunarea a două numere	`x ← a + b`
-	Scăderea a două numere	`y ← y - 10`
*	Înmulțirea a două numere	`x ← a * 10`
/	Împărțirea întreagă a două numere (cîtul împărțirii)	`x ← 14 / 3` (x primește valoarea 4)
%	Împărțirea întreagă a două numere (restul împărțirii)	`x ← 14 % 3` (x primește valoarea 2)
( )	Paranteze: schimbul ordinii operațiilor	`x ← 2 * (10 - (3 + 4))` (x primește valoarea 6)

"x", "a", and "b" are variables, each of them holding a numerical value. Thus, through the operation "x = a + b," the variable "x" will receive the value that is the sum of "a" and "b."

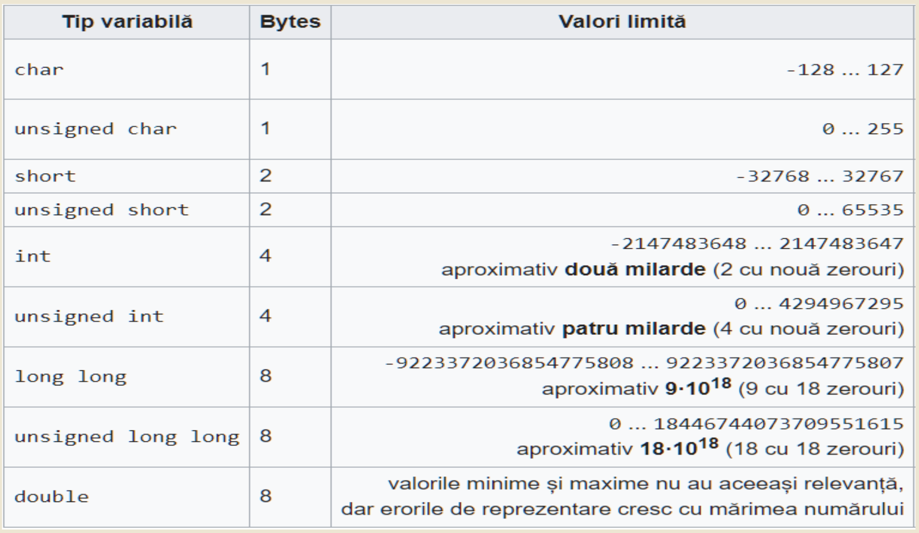
Example:
We have variables a, b, and c, and we execute the following code sequence:

a = 5;
b = 3;
c = a * b; -> c = 15;
c = c * 3; -> c = 45;
a = c - a; a = 40, -> c = 45;

If an expression ("c * 3") is assigned to a variable, first the expression will be evaluated (c = 15 -> "c * 15" = 45), and then that value will be assigned to the variable. In the end, the variable "c" will have the value 45.

If we use a variable in an expression ("c - a"), its value does not change. The value of a variable changes only through an assignment ("a = ...").

Variable Types:
Each type of variable can hold a different kind of number. In robotics, we will mainly use 3 types of data: "int" (can hold integers between -10^9 and +10^9), "double" (can hold real numbers), and "boolean" (can hold 1 and 0, representing true or false). Even though we won't use them frequently, it's important for you to know about the other variable types as well:



Tip variabilă	Bytes	Valori limită
`char`	1	-128 ... 127
`unsigned char`	1	0 ... 255
`short`	2	-32768 ... 32767
`unsigned short`	2	0 ... 65535
`int`	4	-2147483648 ... 2147483647 aproximativ **două milarde** (2 cu nouă zerouri)
`unsigned int`	4	0 ... 4294967295 aproximativ **patru milarde** (4 cu nouă zerouri)
`long long`	8	-9223372036854775808 ... 9223372036854775807 aproximativ **$9 \cdot 10^{18}$** (9 cu 18 zerouri)
`unsigned long long`	8	0 ... 18446744073709551615 aproximativ **$18 \cdot 10^{18}$** (18 cu 18 zerouri)
`double`	8	valorile minime și maxime nu au aceeași relevanță, dar erorile de reprezentare cresc cu mărimea numărului

Examples:

- We will use "double" for speed.
- We will use "boolean" for a variable like "RobotIsMoving."
- We will use "int" for the number of autonomous code cycles.

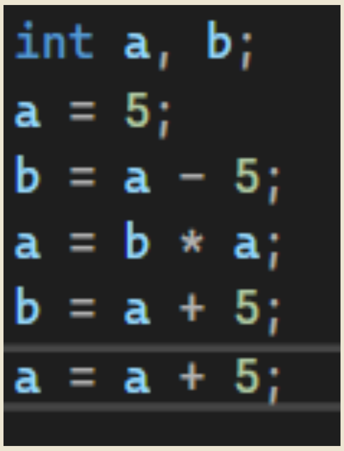
String:

"String" is a data type that can hold a sequence of characters. For instance, a variable of type string can have the value "Robert."

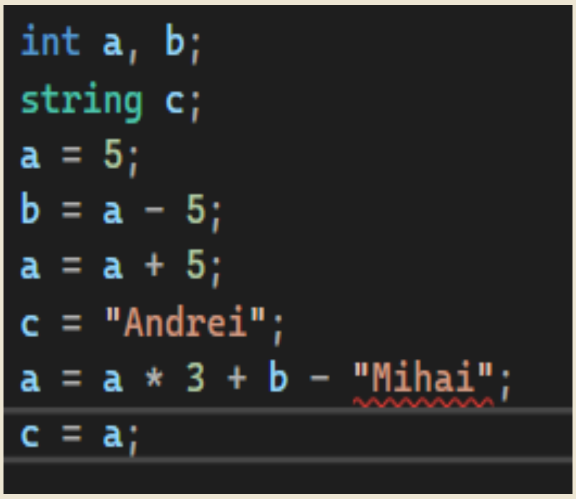
We cannot assign a string variable's value to an int variable. In other words, if we have a variable "a" (of type int) and a variable "s" (of type string), we cannot write "a = s" (because **"a"** can only hold integer numbers, while **"s"** represents a sequence of characters) or "s = a."

Exercitii:

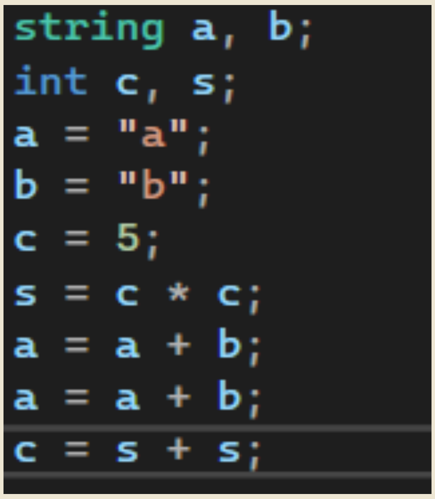
- What values will the variables **a**, **b**, **c**, and **s** have at the end of each code?



```
int a, b;
a = 5;
b = a - 5;
a = b * a;
b = a + 5;
a = a + 5;
```

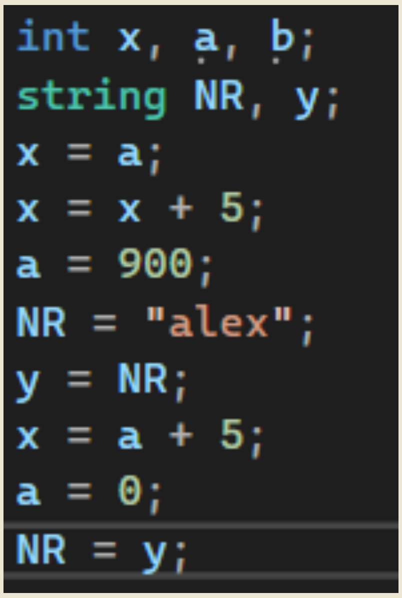


```
int a, b;
string c;
a = 5;
b = a - 5;
a = a + 5;
c = "Andrei";
a = a * 3 + b - "Mihai";
c = a;
```



```
string a, b;
int c, s;
a = "a";
b = "b";
c = 5;
s = c * c;
a = a + b;
a = a + b;
c = s + s;
```

If in any of the examples you are not clear about what is happening, follow the code line by line and write down the values of each variable on a sheet of paper. This method of error detection is called "debugging" and is the most common way to identify errors in code.

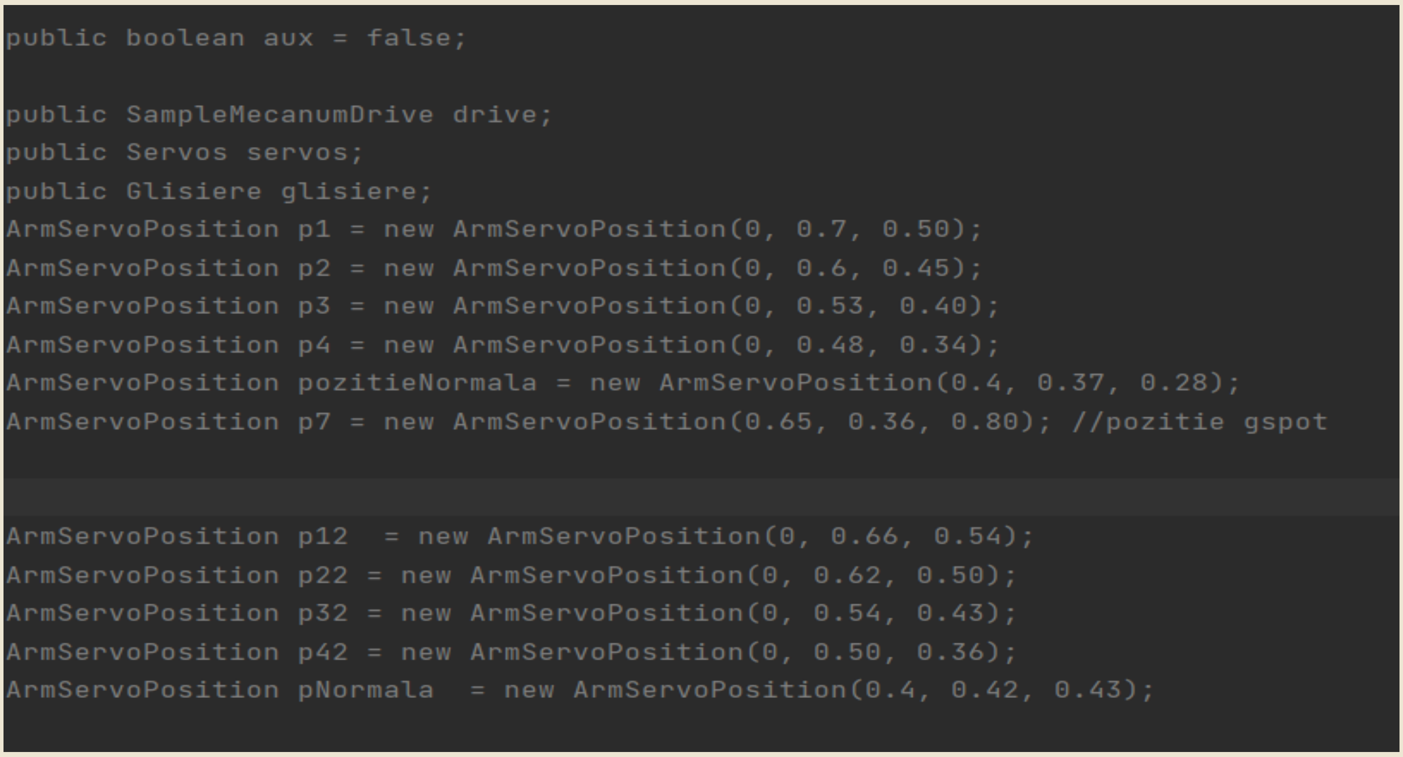


```
int x, a, b;
string NR, y;
x = a;
x = x + 5;
a = 900;
NR = "alex";
y = NR;
x = a + 5;
a = 0;
NR = y;
```

- Specify the value of **NR** at the end and write 2 more lines of code so that the variable **x** will have the value 999, and the variable **NR** will have the value "ALEX." You are not allowed to use "x = 999" or "NR = "ALEX"."

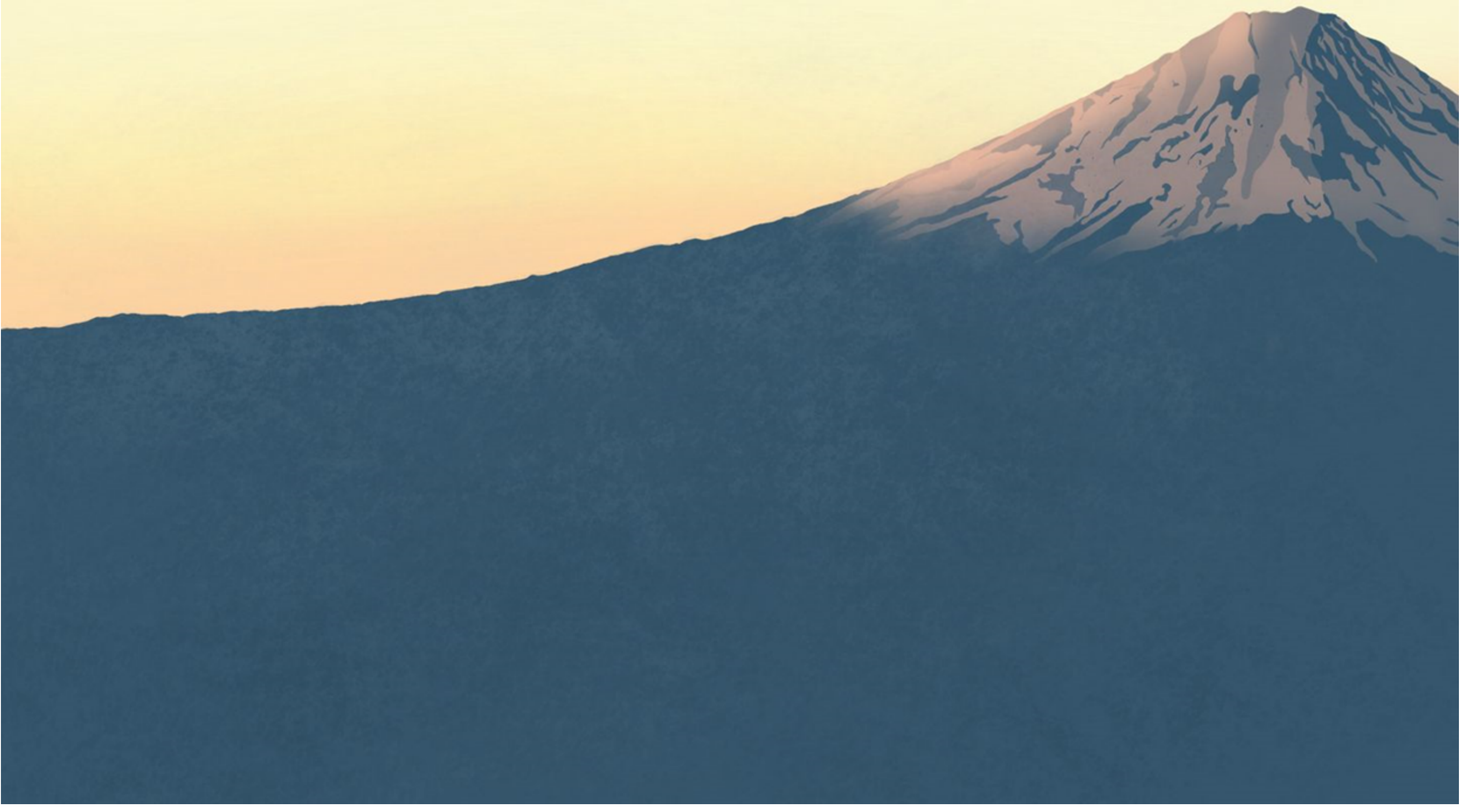
You probably find this code to be nonsensical. I completely agree with you. However, you will end up working with the robot's code, and I promise you that it truly doesn't make any sense. This is why you need to learn to follow code written by others and write code that is easy to understand.

Example of code from the robot –>

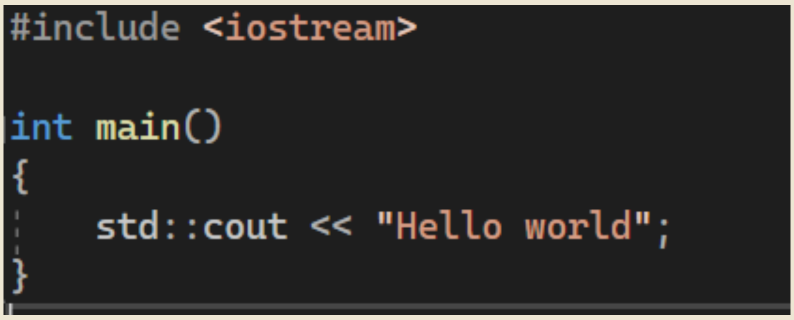


```java
public boolean aux = false;

public SampleMecanumDrive drive;
public Servos servos;
public Glisiere glisiere;
ArmServoPosition p1 = new ArmServoPosition(0, 0.7, 0.50);
ArmServoPosition p2 = new ArmServoPosition(0, 0.6, 0.45);
ArmServoPosition p3 = new ArmServoPosition(0, 0.53, 0.40);
ArmServoPosition p4 = new ArmServoPosition(0, 0.48, 0.34);
ArmServoPosition pozitieNormala = new ArmServoPosition(0.4, 0.37, 0.28);
ArmServoPosition p7 = new ArmServoPosition(0.65, 0.36, 0.80); //pozitie gspot


ArmServoPosition p12  = new ArmServoPosition(0, 0.66, 0.54);
ArmServoPosition p22 = new ArmServoPosition(0, 0.62, 0.50);
ArmServoPosition p32 = new ArmServoPosition(0, 0.54, 0.43);
ArmServoPosition p42 = new ArmServoPosition(0, 0.50, 0.36);
ArmServoPosition pNormala  = new ArmServoPosition(0.4, 0.42, 0.43);
```

CHAPTER 3: The basics

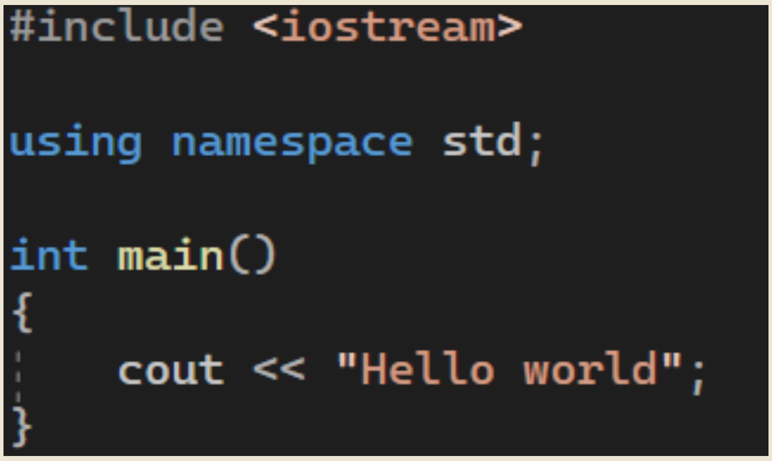


If you open a new project in any of the before mentioned applications, you will see some lines of code already written:



```
#include <iostream>

int main()
{
    std::cout << "Hello world";
}
```

Depending on the application used, you will encounter one of these two variants, but they function the same way. Both will display the message **"Hello world"** on the screen. Let's see what each of these lines means:



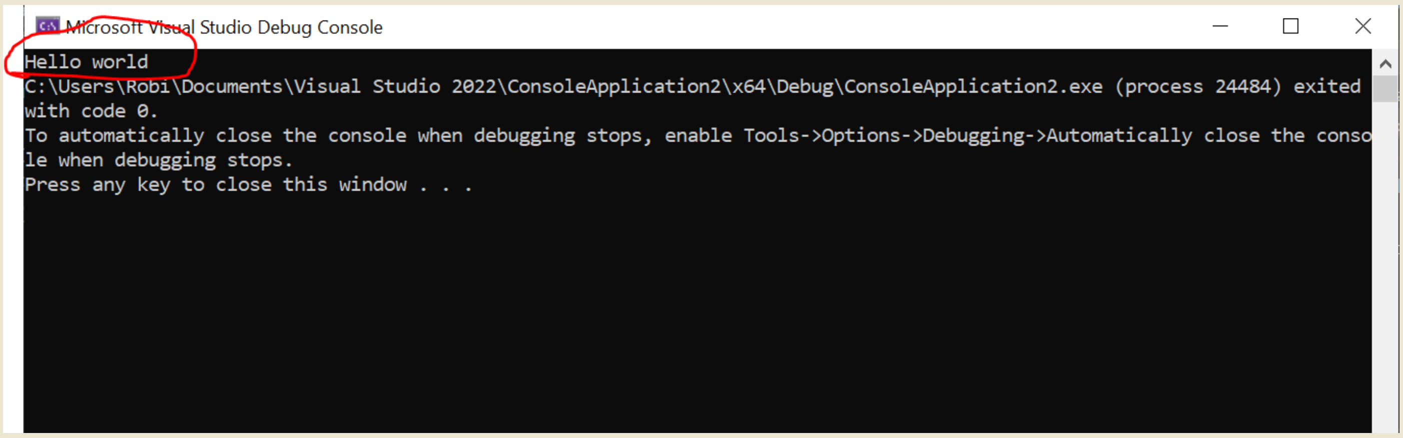
```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world";
}
```

#include <iostream> *tells the computer that it can use the* ***"iostream"*** *library. Libraries function like dictionaries, allowing the computer to understand the other lines of code because they are defined in* ***"iostream."*** *Without this library, the computer wouldn't be able to compile the* ***"cout"*** *operation.*

using namspace std; It tells the computer to automatically add ***"std::"*** before lines of code. This line of code is not mandatory, but it will make your life much easier. You can see that one of the examples has the line ***"using namespace std;"***, while the other uses ***"std::"*** before lines of code. If we were to continue the code from the first example, we would need to write ***"std::"*** before each line of code.

Int main() { } defines the actual program. When we compile the program, it will only run the code inside the curly braces of int main().

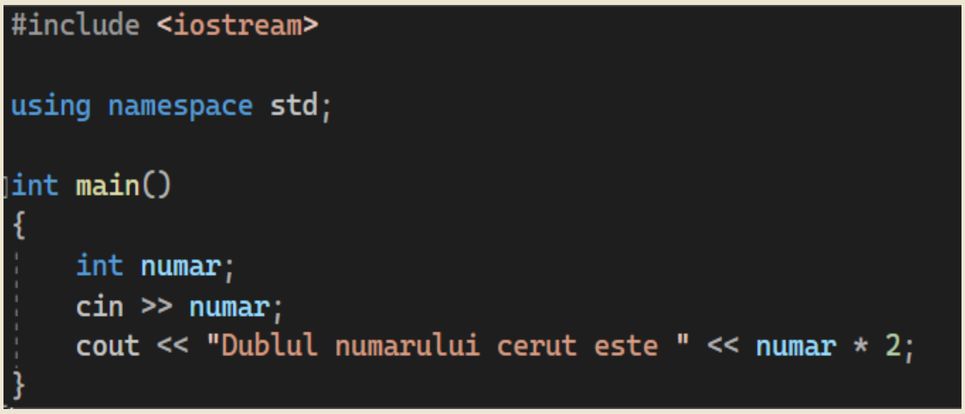
cout << ; This is the write operation: if we write the name of a variable after ***"<<"*** its value will be displayed on the screen, and if we write ***"..."*** after ***"<<"*** then the program will display the text within the quotation marks. In our case, it will display ***"Hello world"*** because that's what is written between the quotation marks.



```
Hello world
C:\Users\Robi\Documents\Visual Studio 2022\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (process 24484) exited
with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

cin >> ; is used to read the value of a variable from the keyboard.

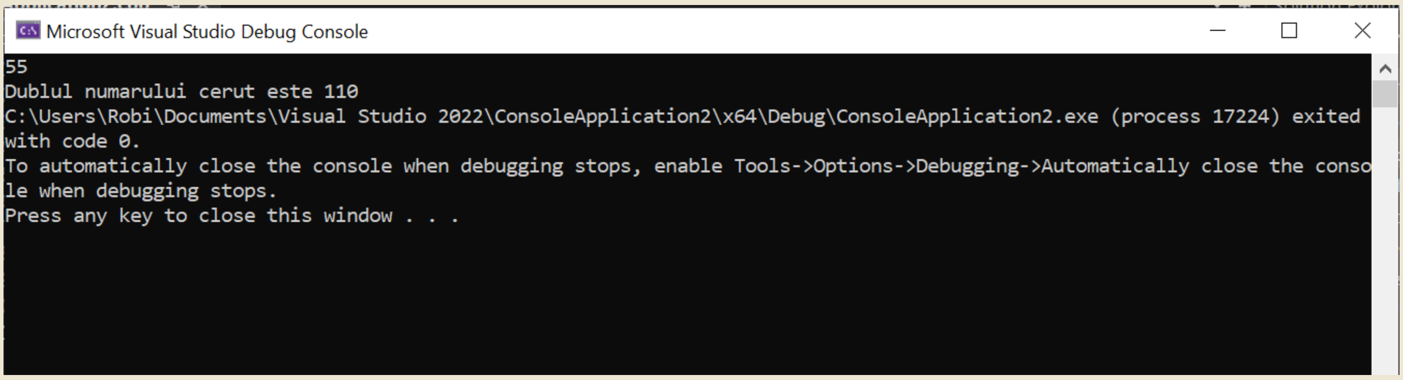
For example, if we want to read a number from the keyboard and then display its double, the code will look like this:



```cpp
#include <iostream>

using namespace std;

int main()
{
    int numar;
    cin >> numar;
    cout << "Dublul numarului cerut este " << numar * 2;
}
```

This program will wait until we input a number in the console. Then it will display the text "Dublul numarului cerut este " followed by the value of the expression ***"numar * 2."***

In the console it looks like this:

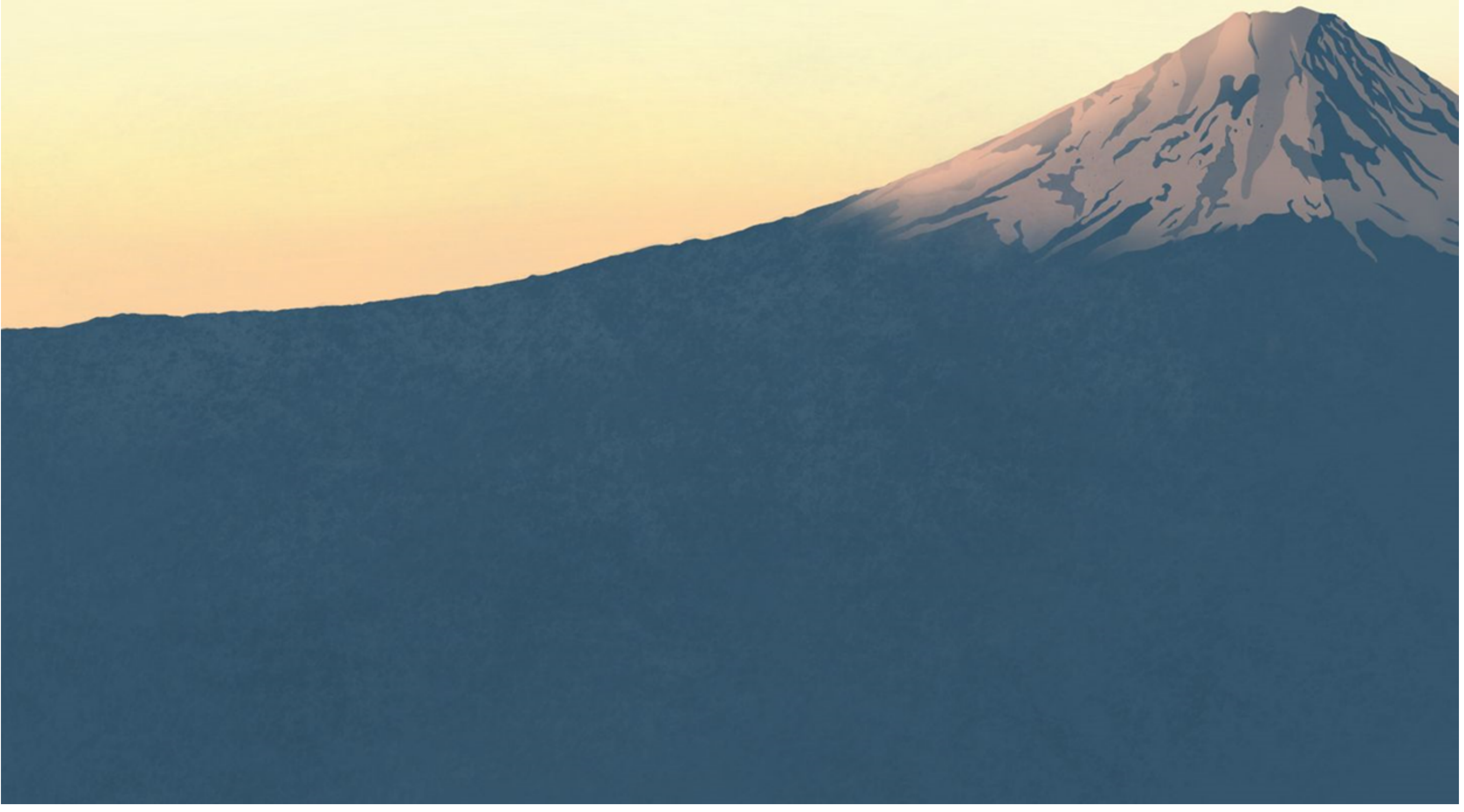


```
55
Dublul numarului cerut este 110
C:\Users\Robi\Documents\Visual Studio 2022\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (process 17224) exited
with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Exercises:

1. ***https://www.pbinfo.ro/probleme/939/sum00***
2. ***https://www.pbinfo.ro/probleme/1258/scadere2***
3. ***https://www.pbinfo.ro/probleme/1260/asii***
4. ***https://www.pbinfo.ro/probleme/3178/copii2***

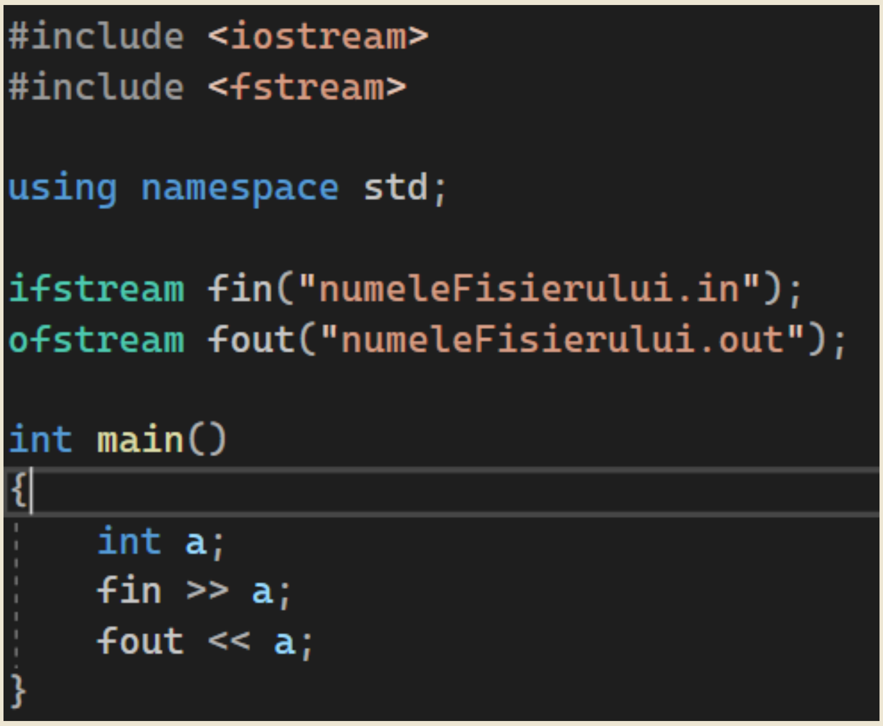
CHAPTER 4: Writing in files



The basic syntax for writing to a file is:

- ***fin*** works exactly the same as "cin," but it reads from the file "filename.in."
- ***fout*** works exactly the same as "cout," but it writes to the file "filename.out."

IMPORTANT: you need to add the library ***<fsteam>***

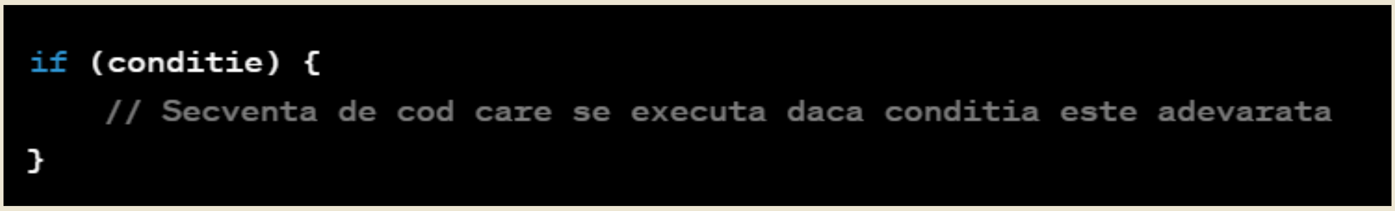


```
#include <iostream>
#include <fstream>

using namespace std;

ifstream fin("numeleFisierului.in");
ofstream fout("numeleFisierului.out");

int main()
{
    int a;
    fin >> a;
    fout << a;
}
```

CHAPTER 5: The ”if” decision structure

In C++, the "if" statement is used to decide whether a certain sequence of code should be executed or not, based on a given condition. If the condition is true, the code sequence is executed; otherwise, it is skipped.

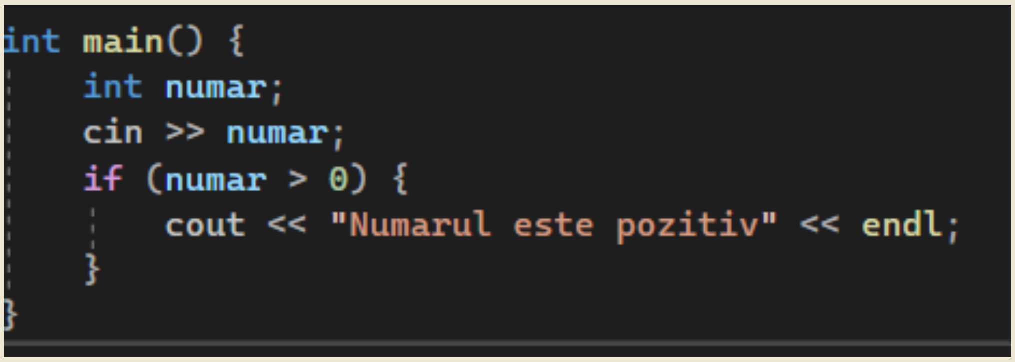
The syntax of the "if" statement is as follows:



```
if (conditie) {
    // Secventa de cod care se executa daca conditia este adevarata
}
```

In this case, the "condition" is an expression that can be evaluated as true or false. If the condition is true, the code sequence within the following curly braces is executed. If the condition is false, the code sequence is skipped, and the program moves to the next line of code after the "if" sequence.

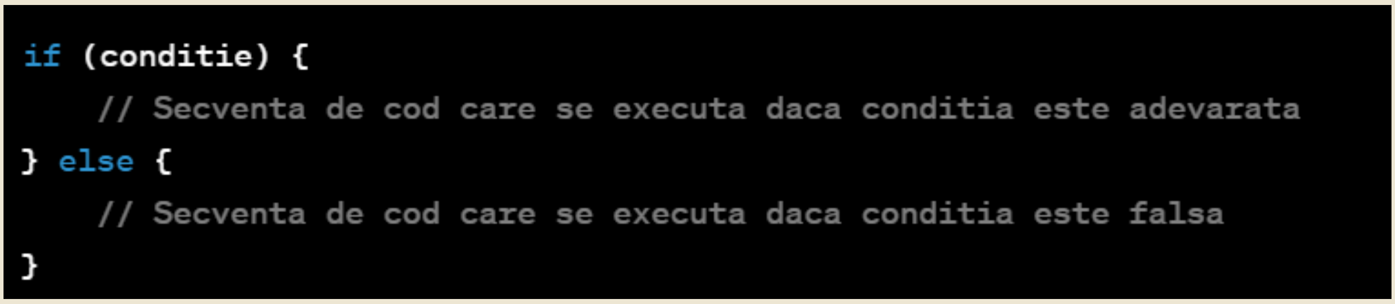
For example, let's say we want to write a program that reads a number from the keyboard and displays the message "The number is positive" if the number is greater than zero, otherwise it does nothing. We can use the "if" statement as follows:



```
int main() {
    int numar;
    cin >> numar;
    if (numar > 0) {
        cout << "Numarul este pozitiv" << endl;
    }
}
```

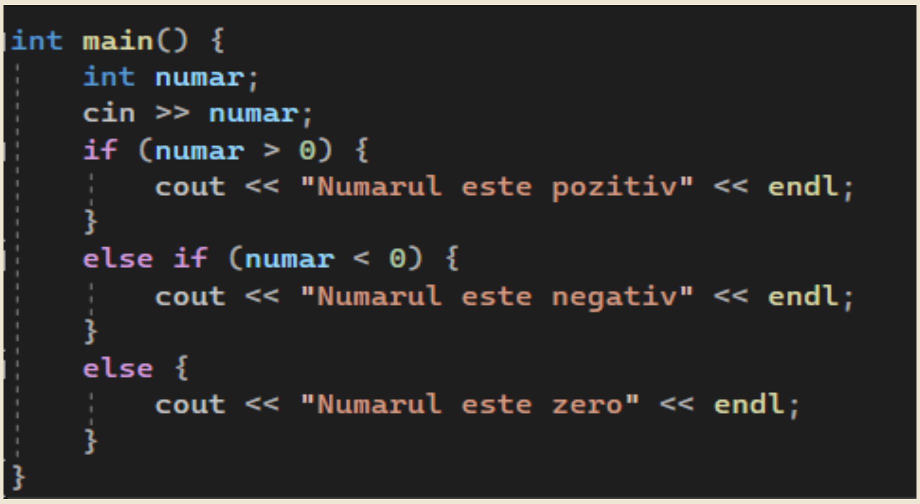
In this example, "cin" is used to read a number from the keyboard, and "if" is used to decide whether the number is greater than zero or not. If the number is greater than zero, the message "The number is positive" is displayed using "cout." Otherwise, the program does nothing and moves to the next line of code after the "if" statement.

In addition to the basic syntax of the "if" statement, there's also a variation called "if-else" that can be used to execute a different code sequence if the condition is false. The syntax is as follows:



```
if (conditie) {
    // Secventa de cod care se executa daca conditia este adevarata
} else {
    // Secventa de cod care se executa daca conditia este falsa
}
```

For example, let's say we want to write a program that displays the message "The number is positive" if the number is greater than zero, the message "The number is negative" if the number is less than zero, and a different message if the number is zero. We can use the "if-else" statement to achieve this as follows:



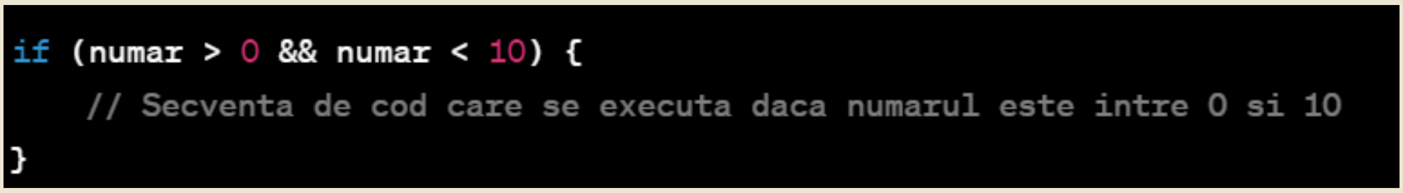
```
int main() {
    int numar;
    cin >> numar;
    if (numar > 0) {
        cout << "Numarul este pozitiv" << endl;
    }
    else if (numar < 0) {
        cout << "Numarul este negativ" << endl;
    }
    else {
        cout << "Numarul este zero" << endl;
    }
}
```

In this example, the "if" statement checks whether the number is greater than zero. If this is true, the message "The number is positive" is displayed. If it's not true, the "else if" statement checks whether the number is less than zero. If this is true, the message "The number is negative" is displayed. Otherwise, the "else" statement is executed, displaying the message "The number is zero."

In C++, we can also use logical operators like "&&" (and), "||" (or), or "!" (not) in the conditions of the "if" statement to combine multiple expressions.

operator	meaning	How it's used
&&	and	exp. && exp. (is true if both are true)
||	or	exp. || exp. (is true if any of them are true)
!	no	!exp. (is true if exp is false)

For example, we can check if a number is between 0 and 10 using the "&&" operator like this:



```
if (numar > 0 && numar < 10) {
    // Secventa de cod care se executa daca numarul este intre 0 si 10
}
```

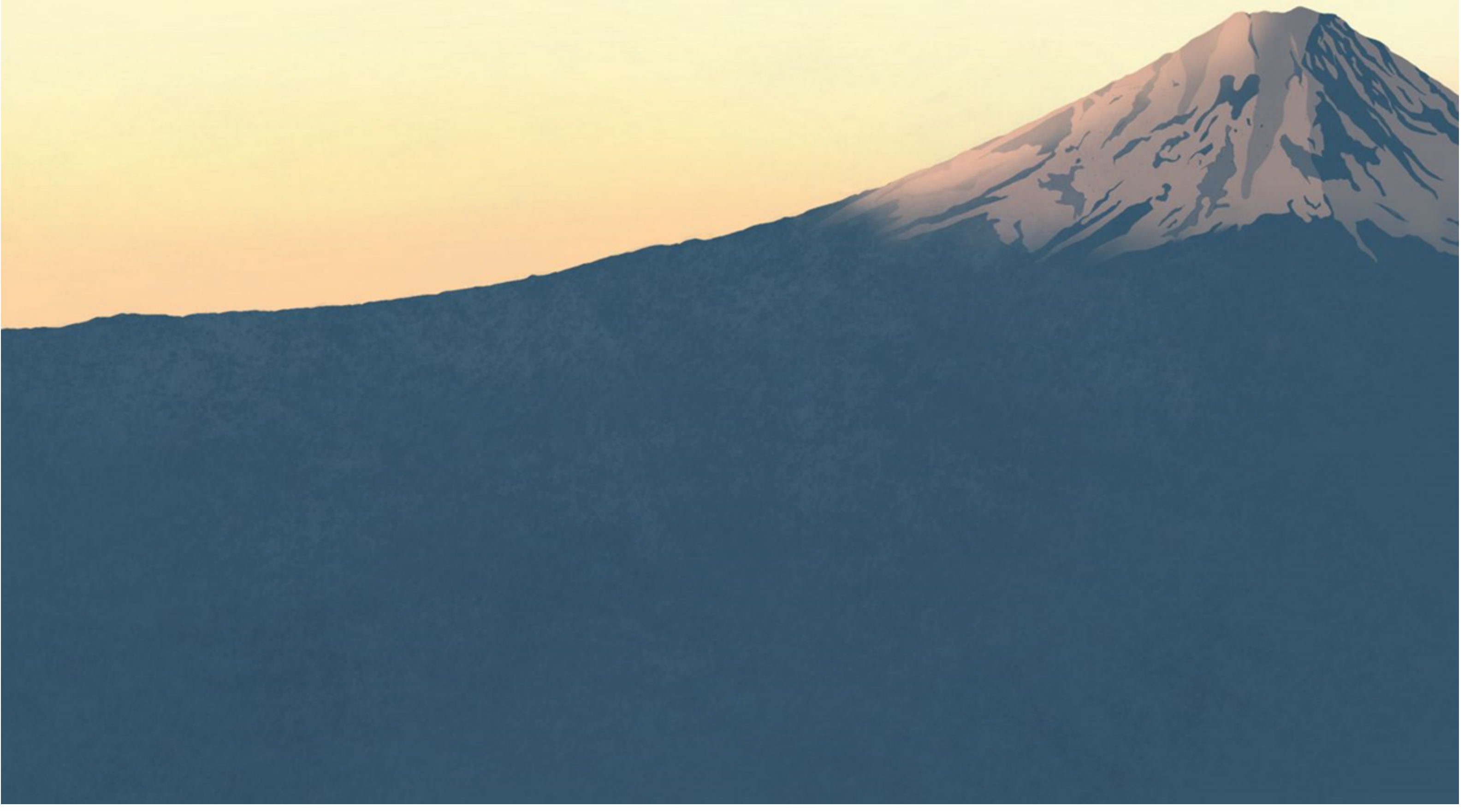
In this case, the "if" statement will be true only if the number is greater than zero and less than 10.

Note:
Inside an "if" statement (between curly braces), we can include any code sequence: we can have multiple lines of code and even other "if" statements.

Exercises:

1. ***https://www.pbinfo.ro/probleme/109/paritate***
2. ***https://www.pbinfo.ro/probleme/105/max2***
3. ***https://www.pbinfo.ro/probleme/469/interval2***
4. ***https://www.pbinfo.ro/probleme/106/minim3***
5. ***https://www.pbinfo.ro/probleme/4339/pare-impare***
6. ***https://www.pbinfo.ro/probleme/452/cifimp***
7. ***https://www.pbinfo.ro/probleme/1311/cifegale***
8. ***https://www.pbinfo.ro/probleme/168/semn***

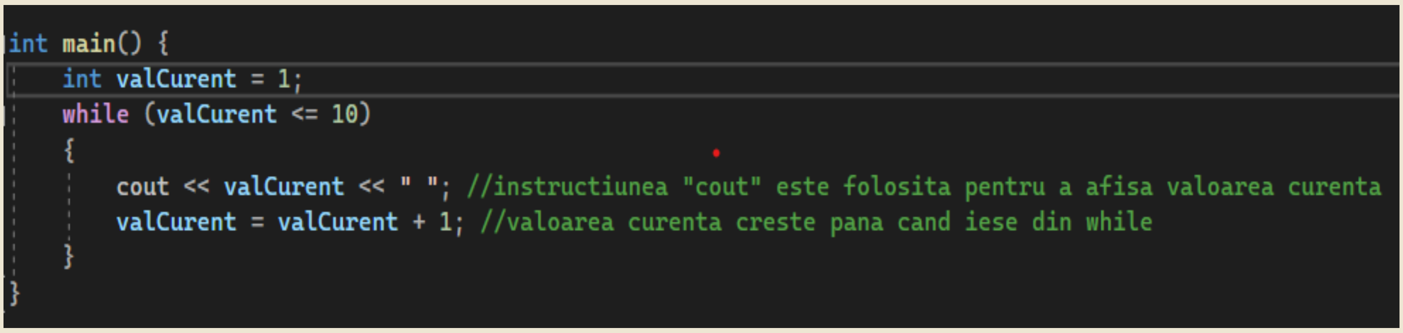
CHAPTER 6: Repetitive instructions



The "while" statement allows for the repeated execution of a code block as long as a condition is true. The syntactic structure of this statement is as follows:

```
while (conditie) {
    // blocul de cod ce va fi executat cat timp conditia este adevarata
}
```

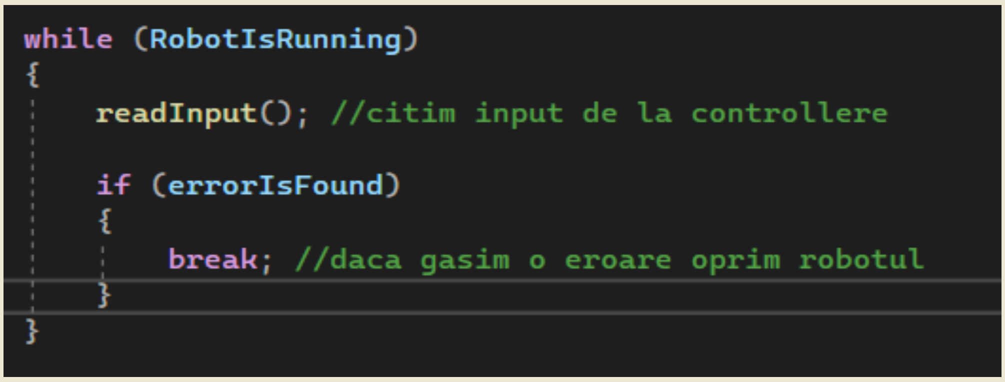
For example, the following code will display numbers from 1 to 10:



```
int main() {
    int valCurent = 1;
    while (valCurent <= 10)
    {
        cout << valCurent << " "; //instructiunea "cout" este folosita pentru a afisa valoarea curenta
        valCurent = valCurent + 1; //valoarea curenta creste pana cand iese din while
    }
}
```

The "while" statement is useful when you don't know the exact number of iterations required to accomplish a certain task. A condition is tested before each iteration, and the associated code block will be executed as long as the condition is true.

Note:
Inside the "while" statement, we can have other repetitive statements, decision structures, or the "break" statement. The "break" statement makes the program exit the "while" loop and is used for special conditions.



```
while (RobotIsRunning)
{
    readInput(); //citim input de la controllere

    if (errorIsFound)
    {
        break; //daca gasim o eroare oprim robotul
    }
}
```

Real example:
This is the skeleton of a code for the TeleOp period.

The "for" statement is another type of repetitive instruction in C++ that can be used to iterate through a sequence of statements multiple times. This type of repetitive statement is useful when we know exactly how many iterations will be needed to accomplish a certain task.

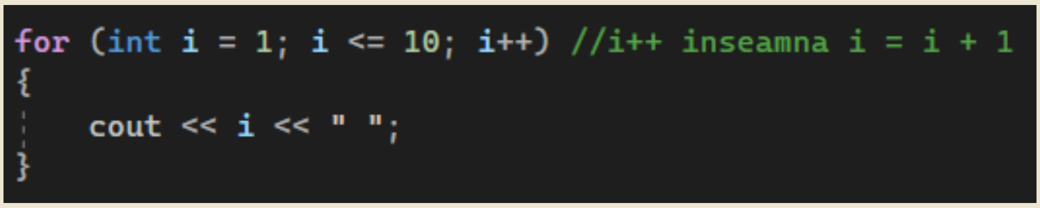
The syntax of the "for" statement is as follows:

```
for (inițializare; condiție; increment/decrement) {
    // bloc de cod care se execută de fiecare dată când condiția este adevărată
}
```

Initialization: This represents an instruction that is executed only once, at the beginning of the loop, and is used to initialize the variables required for the loop.

Condition: It is an expression that is evaluated at each iteration and determines whether the code block inside the loop should be executed or not. If the condition is true, the code block inside the loop is executed. If the condition is false, the loop execution stops.

Increment/Decrement: This is an instruction that is executed at the end of each iteration and is used to update the values of the variables required for the next iteration.



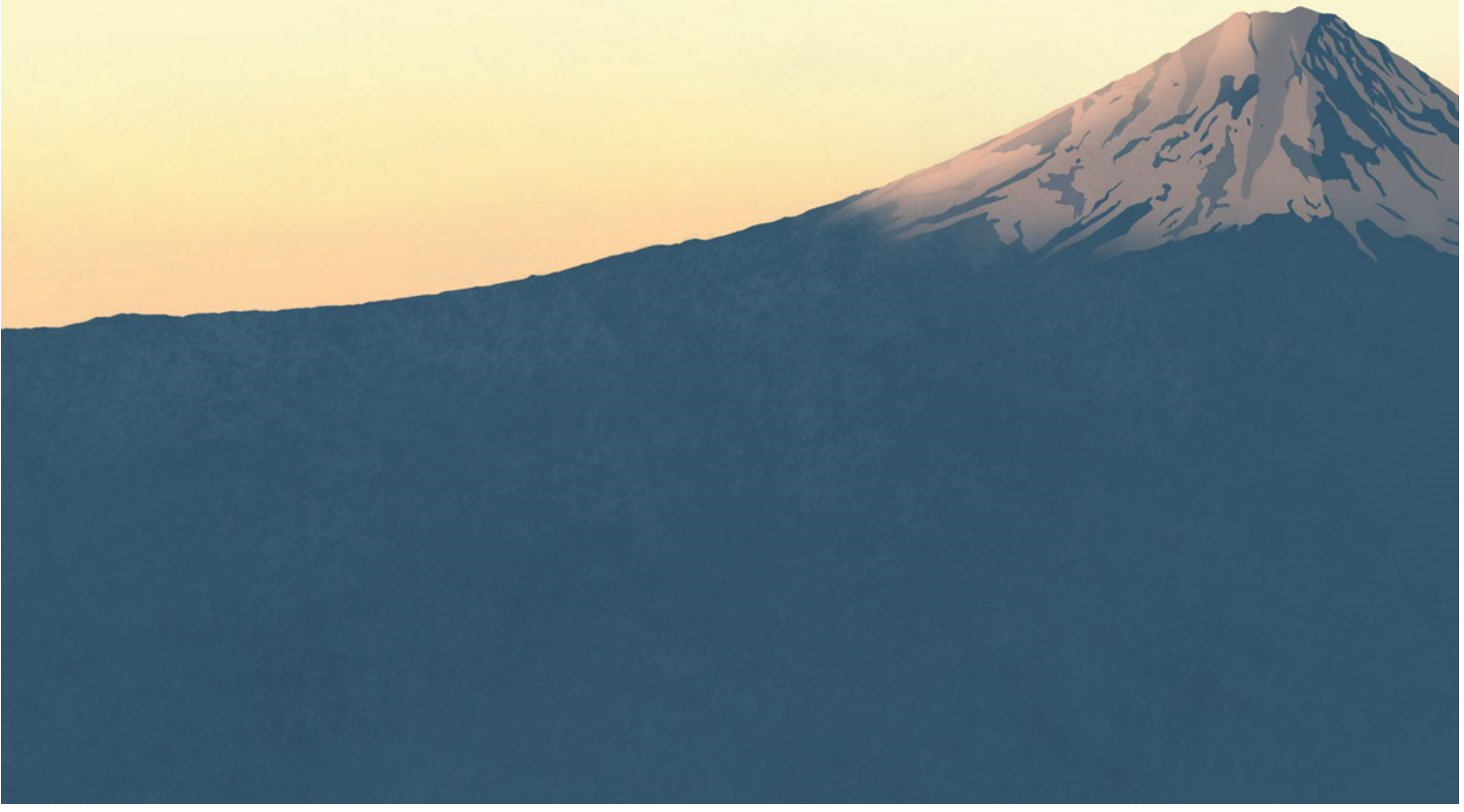
```
for (int i = 1; i <= 10; i++) //i++ inseamna i = i + 1
{
    cout << i << " ";
}
```

In this example, the variable "i" starts with the value 1 and increases until the condition "i <= 10" becomes false, meaning until "i" becomes greater than 10. Inside, we have the instruction "cout << i" which will display the value of "i" at each step. This code will display: 1 2 3 4 5 6 7 8 9 10

Note:
Inside the "for" statement, we can also use any other instructions, including "break." Increment/Decrement doesn't always have to be 1; if desired, we can make a "for" loop go in steps of 5 by changing "i++" to "i = i + 5."

Exercises:
1. ***https://www.pbinfo.ro/probleme/327/afisarenumere***
2. ***https://www.pbinfo.ro/probleme/328/afisarenumere1***
3. ***https://www.pbinfo.ro/probleme/3984/afisare-m2***
4. ***https://www.pbinfo.ro/probleme/3976/prodimpare***
5. ***https://www.pbinfo.ro/probleme/10/suma-cifrelor***
6. ***https://www.pbinfo.ro/probleme/65/produscifreimpare***
7. ***https://www.pbinfo.ro/probleme/3979/suma37***
8. ***https://www.pbinfo.ro/probleme/354/n-maxim***
9. ***https://www.pbinfo.ro/probleme/54/maxim***
10. ***https://www.pbinfo.ro/probleme/171/primaciframinima***

CHAPTER 7: Functions



In C++, functions are blocks of code that perform a specific action when called. They can be defined in a separate file and then called from another file, or they can be defined within the same file where they are called. Functions are used to organize code into logical pieces and to avoid code duplication.

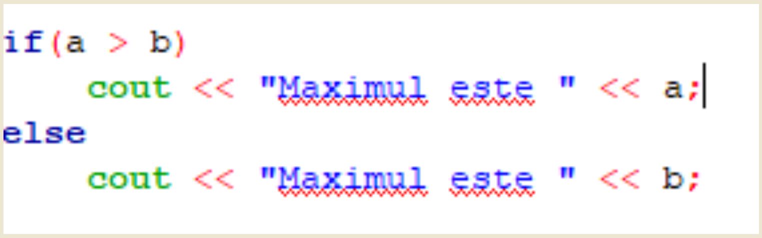
Functions in C++ can be classified into two categories:

- Predefined functions: These are functions predefined in the C++ standard library. They include mathematical functions like sqrt(), abs(), input and output functions like cin and cout, and many others.
- User-defined functions: These are functions defined by programmers to perform a specific action. They are defined outside the main function of the program and can be called from within or outside the program.

User-defined functions in C++ can be defined with or without parameters. Parameters are variables used inside the function and can be passed from outside the function when it is called. Functions can also return a value. This value can be of any type, such as an integer, a character, a string, an object, etc.

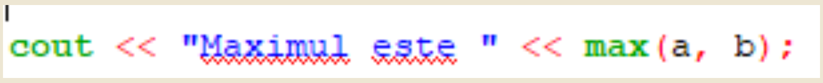
Example:
If we have integer variables a and b and we want to determine which one is larger, we have two options:

We can use the "if" statement like this:



```
if(a > b)
    cout << "Maximul este " << a;
else
    cout << "Maximul este " << b;
```

We can use the ***max()*** function. This is a predefined function (already implemented in C++). It takes 2 parameters and returns the maximum of the two.



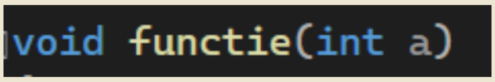
```
cout << "Maximul este " << max(a, b);
```

These two code variants do the same thing.

In programming, there are two types of variables:

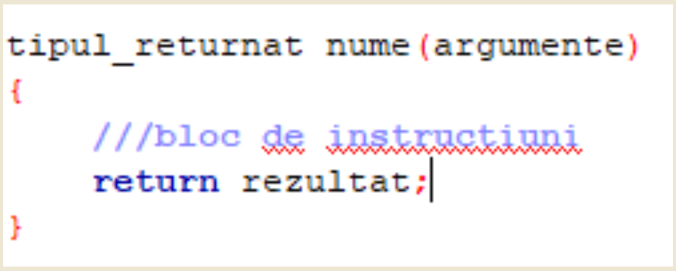
- **Global:** These are declared outside of functions (outside of **int main()**) and can be used anywhere in the code (inside functions or in **int main()**).
- **Local:** These are declared inside functions and do not exist outside of them (you can have two variables with the same name in different functions, and they are independent of each other).

We can have functions that don't return anything; these functions will have a return type of ***void***.

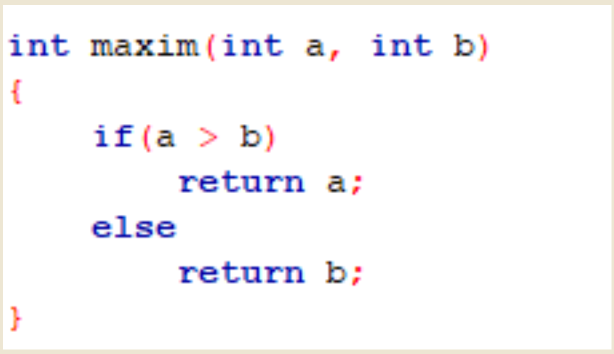


```
void functie(int a)
```

We can use the ***max()*** function as an example to learn how to implement a function.



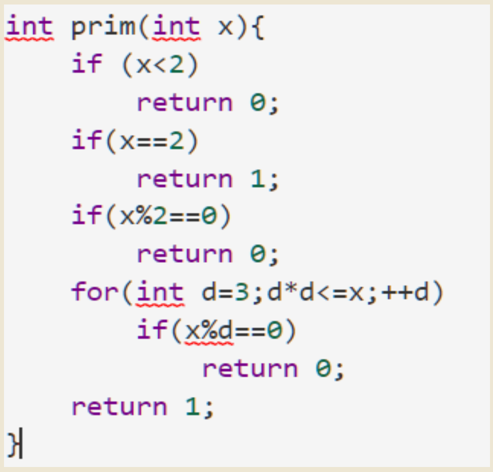
```
tipul_returnat nume(argumente)
{
    ///bloc de instructiuni
    return rezultat;
}
```



```
int maxim(int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

On the right, you have the general structure of a function, and on the left is the implementation of the maxim() function, which functions similarly to max().

The function takes two int variables as arguments, compares them, and returns the larger one. The return type is int because the result will be of type int.



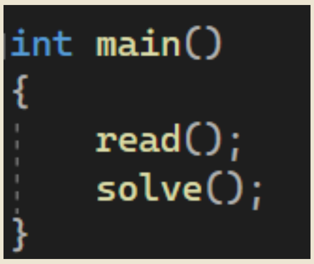
```
int prim(int x){
    if (x<2)
        return 0;
    if(x==2)
        return 1;
    if(x%2==0)
        return 0;
    for(int d=3;d*d<=x;++d)
        if(x%d==0)
            return 0;
    return 1;
}
```

Example of a function that returns 1 if the given number is prime and 0 otherwise:

Remember:
Functions can have arguments and can return any type of variable. The arguments don't have to be of the same type, and there can be as many as needed. The returned variable doesn't have to be of the same type as the arguments (you can have a function that takes a char, two long long variables as arguments, and returns a string).

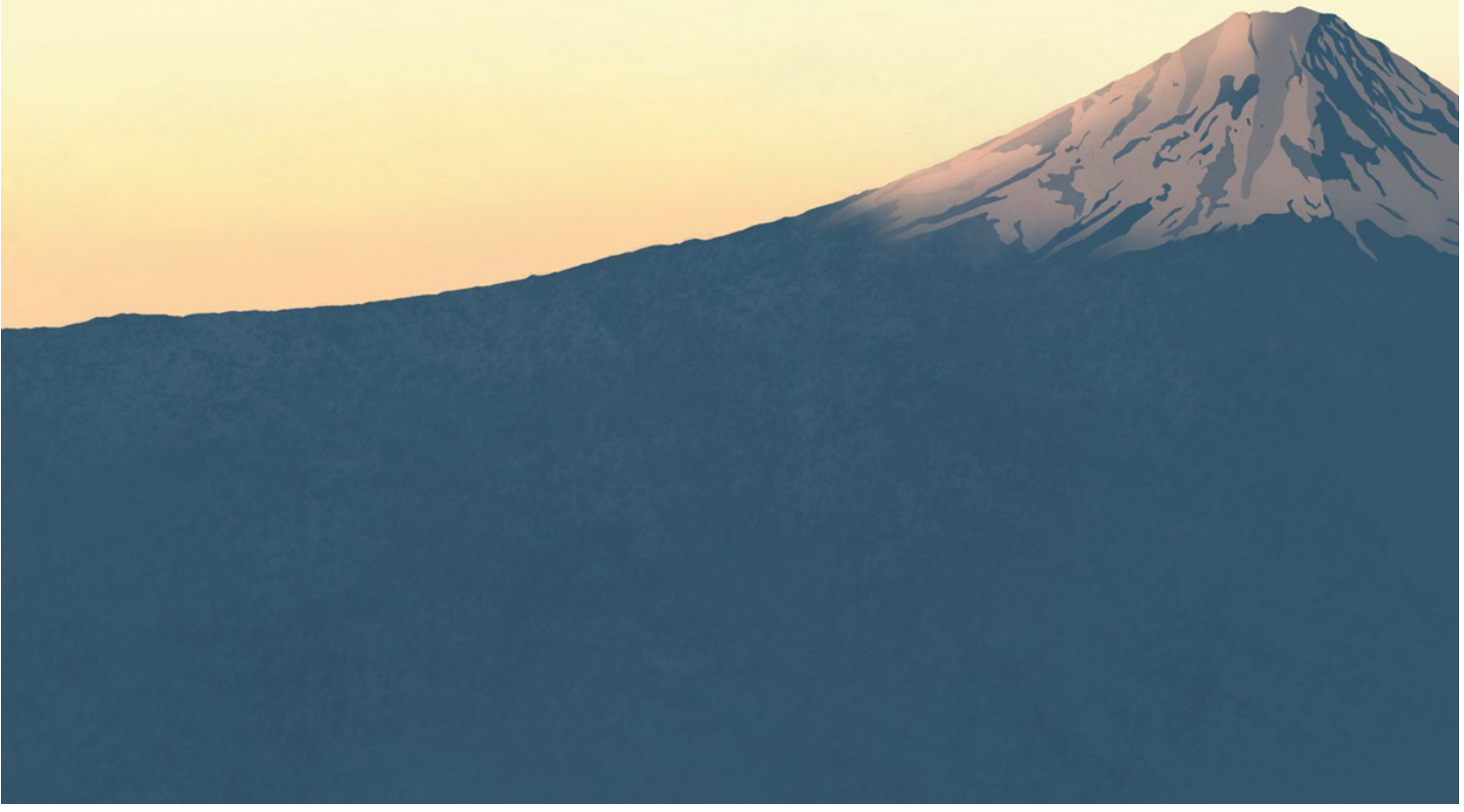
Exercises:

1. https://www.pbinfo.ro/probleme/896/factorialf
2. https://www.pbinfo.ro/probleme/897/sumciff
3. https://www.pbinfo.ro/probleme/898/sumfactcif (you can call a function inside another function)
4. https://www.pbinfo.ro/probleme/24/oglindit2
5. https://www.pbinfo.ro/probleme/2859/treicifimp
6. Rezolvati problema https://www.pbinfo.ro/probleme/4272/prodpare Using the read and solve functions, the **int main()** should look like this:



```
int main()
{
    read();
    solve();
}
```

CHAPTER 8: External libraries

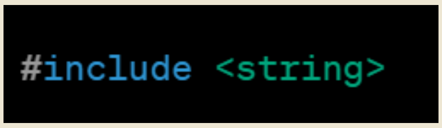


An external library is a collection of precompiled code files that can be used to extend the functionality of your program. These are often written in other programming languages. For example, **<iostream>** is a library containing files for the functions **"cin"** and **"cout".** Without it, the computer wouldn't understand what **"cout"** means.

There are numerous such external libraries that aid in various tasks. For instance, the **<cmath>** library introduces functions like **"pow(x, y)"** (raises the number x to the power of y).

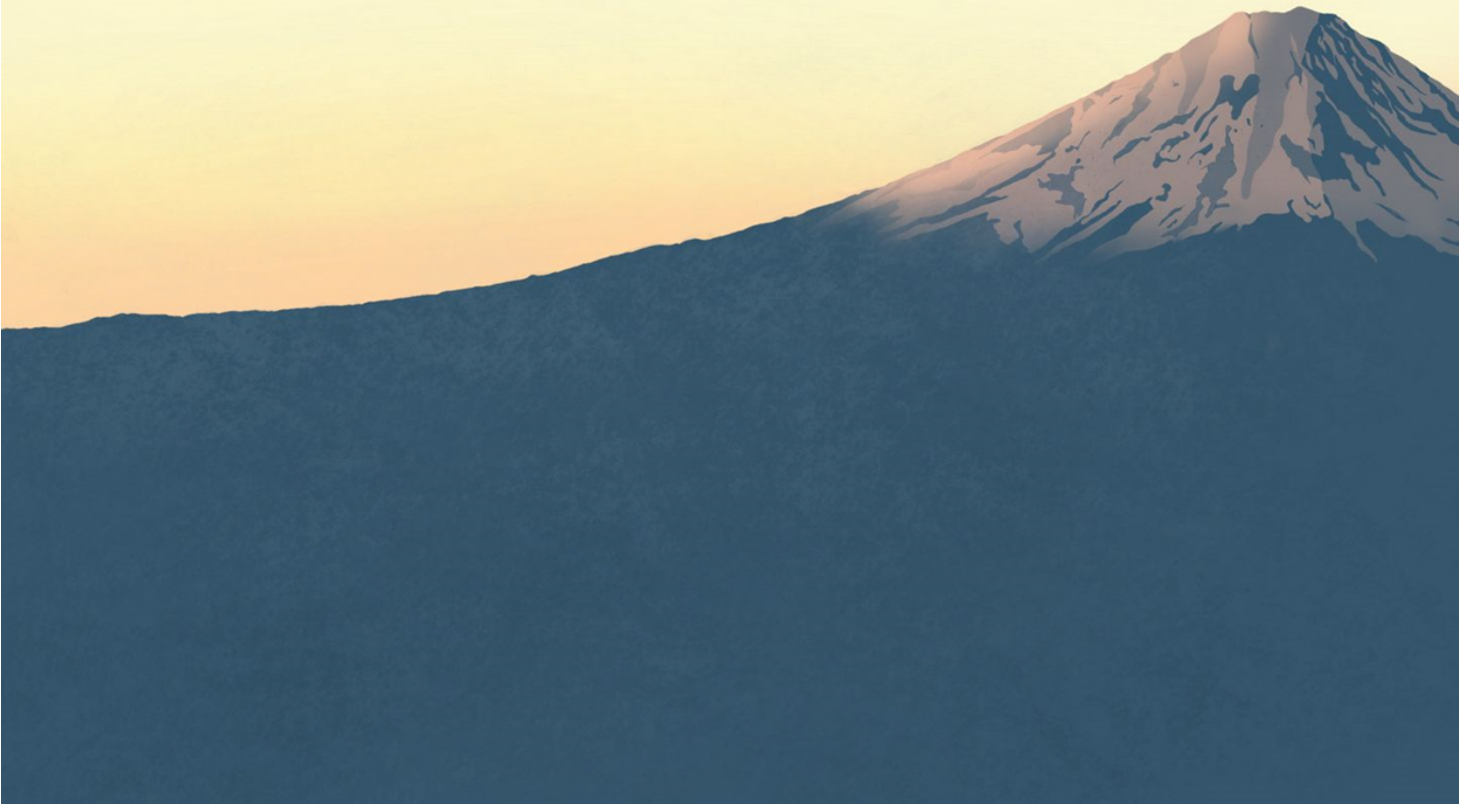
Most libraries you'll use in C++ during high school are included in the "standard library," meaning you can import them using the ***#include*** command. If you want to use a library that isn't part of the "standard library," you'd need to specify to your program where to find that library (you won't need to do this in the near future with C++, but the robot's code is written using these "external" libraries).

This is how we can import the <string> library into our code.



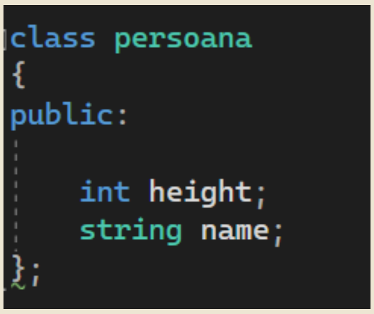
```
#include <string>
```

CHAPTER 9: Object oriented programming



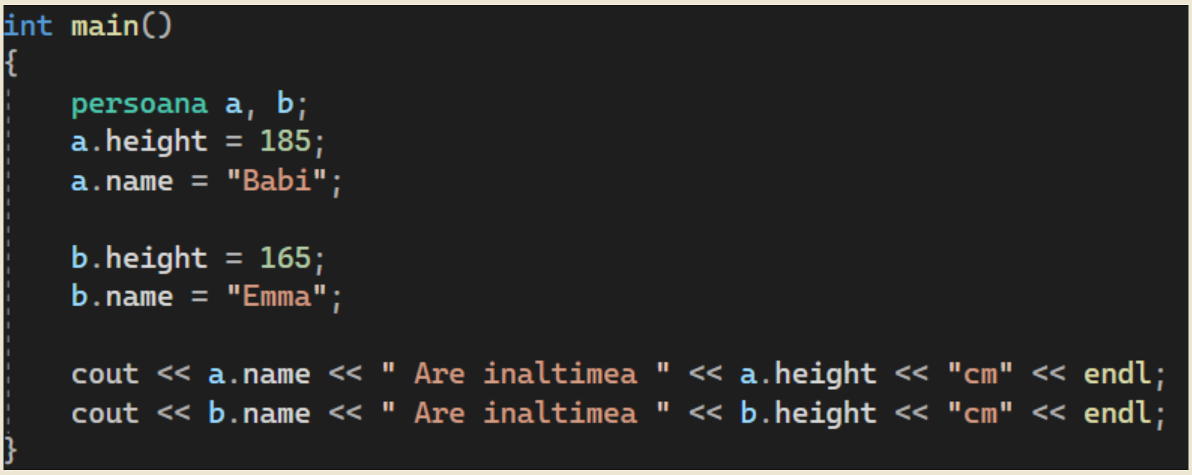
In the previous lessons, we learned that there are various types of variables. These types of variables are actually classes, and the variable itself is an object. For instance, "int" is a class, and a variable of type ***"int"*** is an object of that class.

We can define a new class if we want, creating a "custom new variable type." For example, we can create a class named ***"Person"*** that has two variables implemented (one for height and the other for name).



```
class persoana
{
public:

    int height;
    string name;
};
```

Here's how you declare a class. **"Public:"** means that the variables and functions of this class can be accessed from anywhere in the code. If we had made these variables **"private:",** then they could only be accessed within the class.

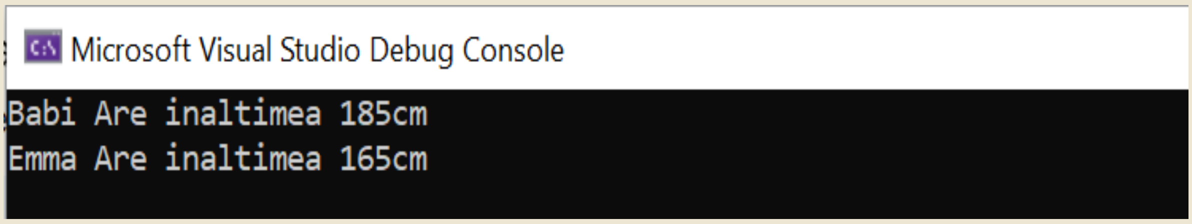


```
int main()
{
    persoana a, b;
    a.height = 185;
    a.name = "Babi";

    b.height = 165;
    b.name = "Emma";

    cout << a.name << " Are inaltimea " << a.height << "cm" << endl;
    cout << b.name << " Are inaltimea " << b.height << "cm" << endl;
}
```

Using this class, we can declare objects of type ***"Person"*** and access the variables within the class using ***"object_name.height/name".***

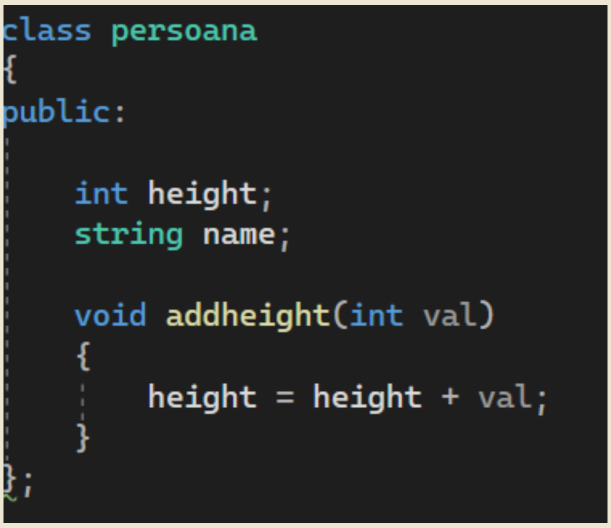
We've declared 2 objects, "a" and "b", each with its own ***"height"*** and ***"name"*** variables.

This code will display in the console:



```
Babi Are inaltimea 185cm
Emma Are inaltimea 165cm
```

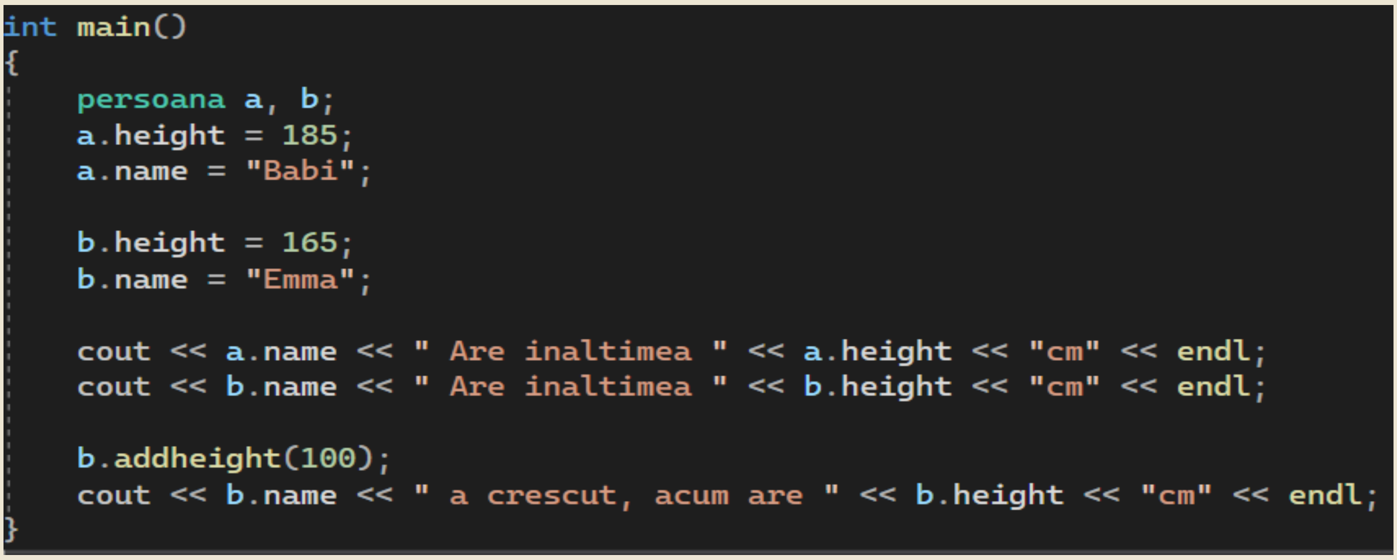
We can also add functions to a class.



```cpp
class persoana
{
public:

    int height;
    string name;

    void addheight(int val)
    {
        height = height + val;
    }
};
```

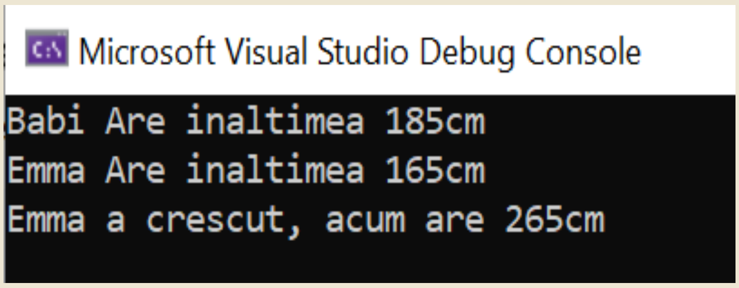
This function takes an integer variable as a parameter and increases the **"height"** value of the object it is called on by the value of **"val".**

In this example, we will use the **"addheight"** function on the object "b." As a result, only **"b.height"** will be changed, while **"a.height"** will remain the same.



```cpp
int main()
{
    persoana a, b;
    a.height = 185;
    a.name = "Babi";

    b.height = 165;
    b.name = "Emma";

    cout << a.name << " Are inaltimea " << a.height << "cm" << endl;
    cout << b.name << " Are inaltimea " << b.height << "cm" << endl;

    b.addheight(100);
    cout << b.name << " a crescut, acum are " << b.height << "cm" << endl;
}
```

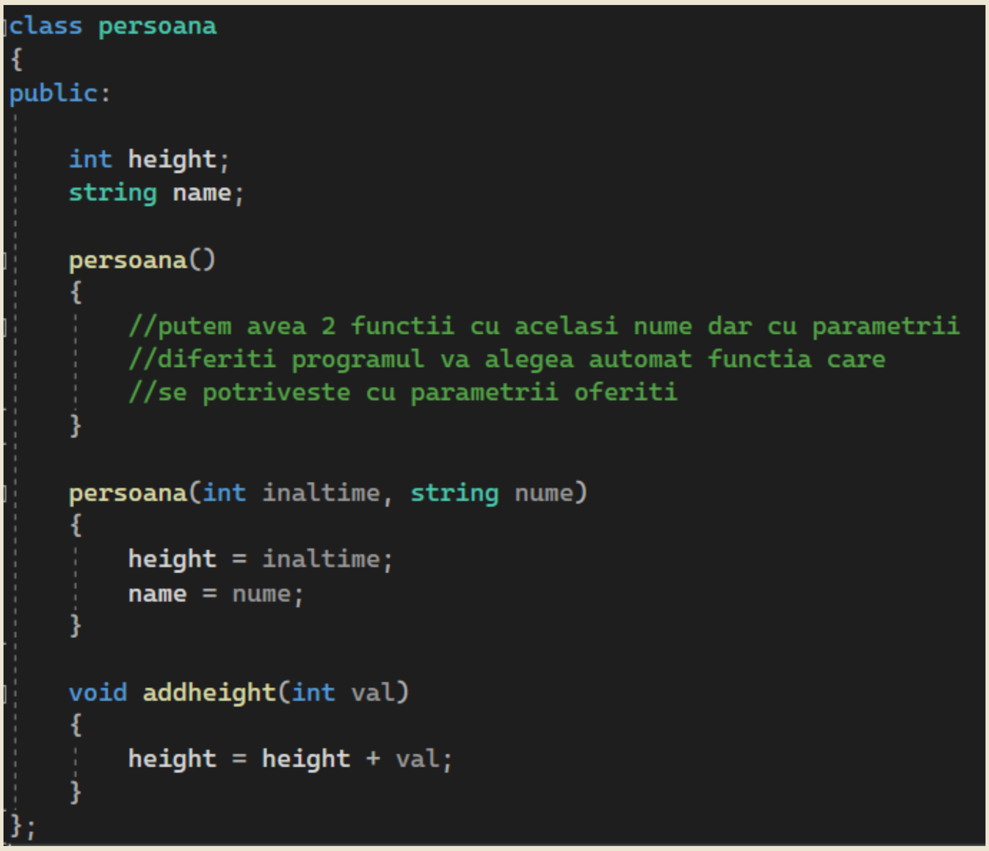
In the console:



```
Babi Are inaltimea 185cm
Emma Are inaltimea 165cm
Emma a crescut, acum are 265cm
```

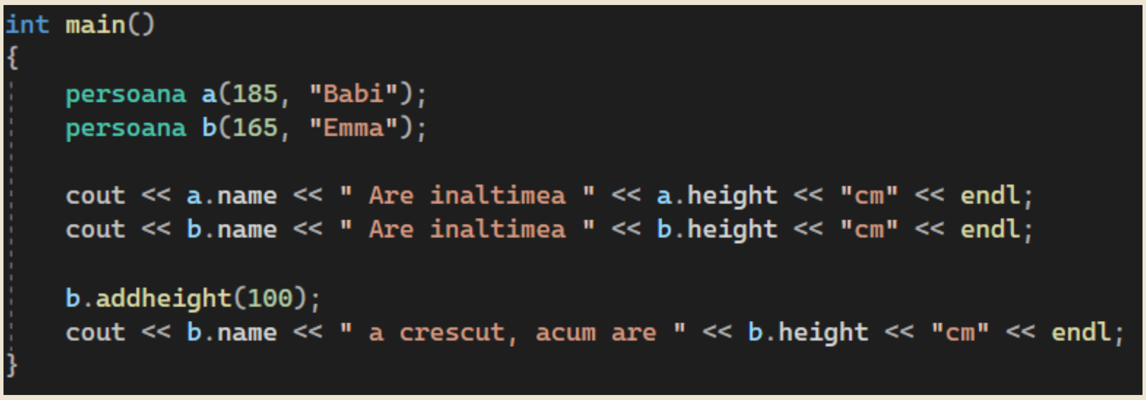
A constructor is a function that is automatically called when an object is created. It can also have parameters. For example, we can provide parameters like name and height, so when we create an object, it will already have the name and height set without us needing to write them using **"a.height = …".**

A constructor is a function with the same name as the class it belongs to and without a return data type



```
class persoana
{
public:

    int height;
    string name;

    persoana()
    {
        //putem avea 2 functii cu acelasi nume dar cu parametrii
        //diferiti programul va alegea automat functia care
        //se potriveste cu parametrii oferiti
    }

    persoana(int inaltime, string nume)
    {
        height = inaltime;
        name = nume;
    }

    void addheight(int val)
    {
        height = height + val;
    }
};
```

This program will display the exact same thing as the previous one, just in a cleaner way.



```
int main()
{
    persoana a(185, "Babi");
    persoana b(165, "Emma");

    cout << a.name << " Are inaltimea " << a.height << "cm" << endl;
    cout << b.name << " Are inaltimea " << b.height << "cm" << endl;

    b.addheight(100);
    cout << b.name << " a crescut, acum are " << b.height << "cm" << endl;
}
```