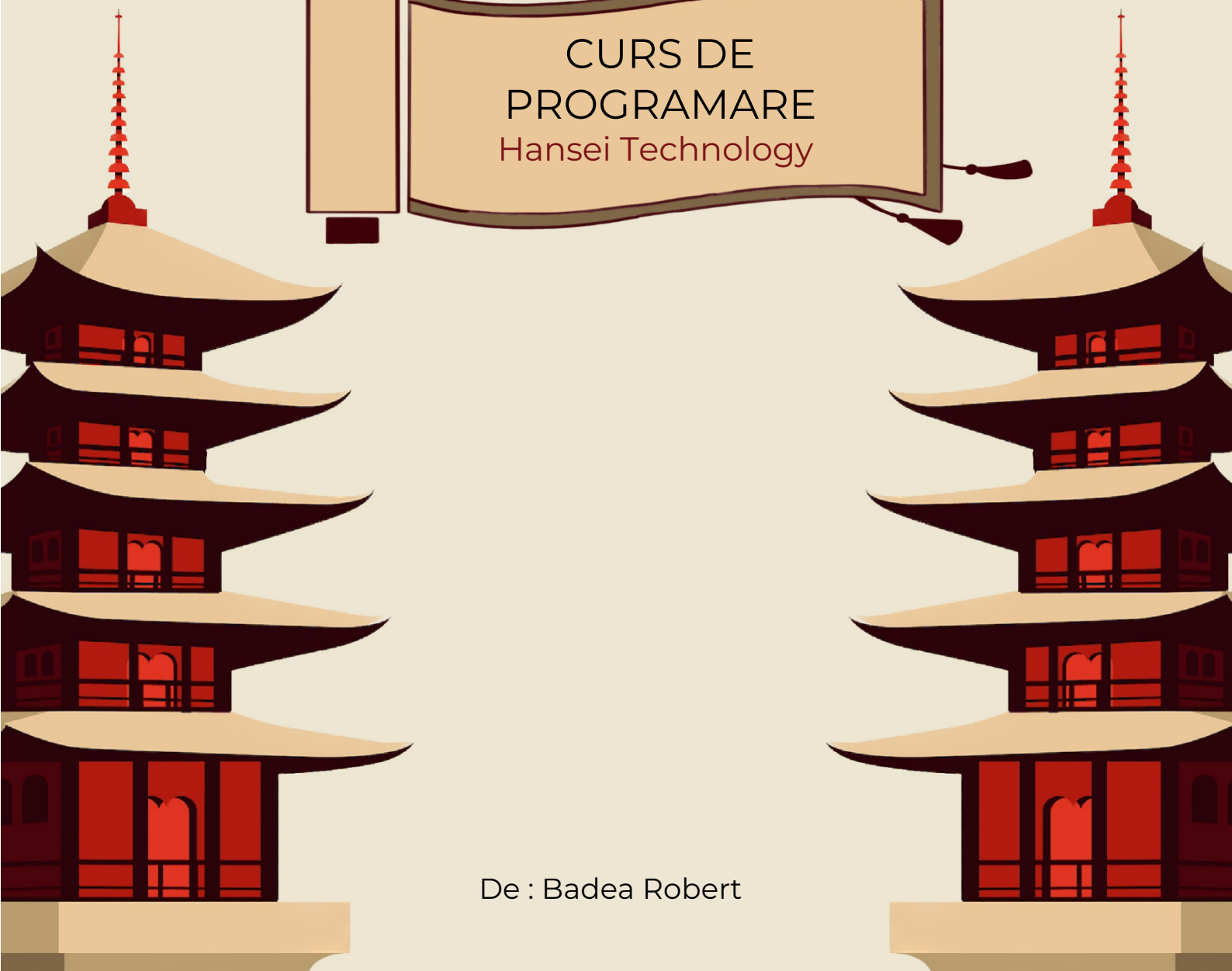
CURS DE PROGRAMARE

Hansei Technology



De : Badea Robert

CUPRINS CAPITOLE CURS:

CAPITOLUL 1:
Introducere

În echipa noastră codul robotului este făcut în java, dar, din păcate, nu stăpânim foarte bine acest limbaj de programare așa că vom începe cu C++ deoarece este mai simplu și pentru noi, dar și pentru voi întrucât asta se face în liceu.

C++:

În prima parte a acestui training veți învăța bazele programării. Chiar dacă le vom parcurge în c++, acestea funcționează identic în orice limbaj de programare. După ce învățați bazele și stăpâniți conceptele de bază vom trece la java, unde veți programa direct roboți, nu vom lucra prea mult teoretic.

De ce nu programăm robotul în c++?

C++ este un limbaj de programare vechi (a fost creat în 1985) și este încă folosit și astăzi pentru crearea sistemelor de operare (un sistem de operare nu poate fi creat în java, acestea pot funcționa decât în c, c++ și pascal), din păcate, așa cum c++ este bun la unele lucruri, la fel și java. Chiar dacă toate limbajele de programare par asemănătoare ele funcționează diferit, iar pentru proiecte diferite este nevoie de limbaje diferite.

Cum va funcționa trainingul?

Acest curs este împărțit în 2 secțiuni: c++ și java. Fiecare dintre ele va avea mai multe lecții, iar la cele de c++, în special, veți primi și teme. Presupunând că sunteți aici deoarece asta vă doriți și nu fiindcă sunteți obligați, așa că va trebuii să vă faceți temele cât considerați voi necesar, însă, trebuie să vă asigurați că înțelegeți fiecare lecție (o să vă pun întrebări din teme și lecțiile trecute ca să mă asigur că ați înțeles, dacă sunteți siguri pe o lecție puteți să nu implementați toate exercițiile, dar trebuie măcar să le citiți).

Eu și colegul meu Radu vom ține lecțiile care vor fi înregistrate video, iar, dacă aveți nevoie de încă o explicație la oricare dintre ele veți putea viziona filmările.

Va trebuii să vă instalați un IDE (un program în care compilați codul scris). Puteți folosi orice IDE vreți, însă recomandările mele sunt: **Visual Studio Community**, **Visual Studio** sau **CodeBlocks**. De asemenea va trebuii să vă faceți cont pe pbInfo, acesta este un site pe care găsiți probleme și teorie pentru c++. Dacă veți avea nevoie de mai multe materiale vă sugerez:

- 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 este cel mai bun prieten al vostru :)

CAPITOLUL 2:
Variabile

O variabilă stochează diferite tipuri de date și permite executarea operațiilor folosind aceste date.

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*" și "*b*" sunt variabile, fiecare dintre acestea are o valoare numerică, astfel, prin operația *"x = a + b"* variabila "x" va primi valoarea sumei dintre a și b.

Exemplu:
Avem variabilele a, b și c și executăm următoarea secvență de cod:

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

Dacă unei variabile îi este atribuită o expresie ("c * 3"), prima dată va fi calculată expresia (c = 15 - > "c * 15" = 45), iar apoi acea valoare va fi atribuită variabilei. La final variabila "c" va avea valoarea 45.

Dacă folosim o variabilă într-o expresie ("c - a"), valoarea acesteia nu se schimbă. Valoarea unei variabile se schimbă decât la o atriburie ("a = ...").

Tipuri de variabile:
Fiecare tip de variabilă poate ține un tip diferit de numere. La robotică noi vom folosi în special 3 tipuri de date: "int"(poate reține numere întregi, între -10^9 și +10^9), "double"(poate reține numere reale) și "boolean"(poate reține 1 și 0, respectiv adevărat sau fals). Chiar dacă nu le vom folosi prea des este important să știți și restul tipurilor de variabile:

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 $\mathbf{9 \cdot 10^{18}}$ (9 cu 18 zerouri)
`unsigned long long`	8	0 ... 18446744073709551615 aproximativ $\mathbf{18 \cdot 10^{18}}$ (18 cu 18 zerouri)
`double`	8	valorile minime și maxime nu au aceeași relevanță, dar erorile de reprezentare cresc cu mărimea numărului

Exemple:

- Vom folosi double pentru viteză.
- Vom folosi boolean pentru o variabilă de tip “RobotulEsteInMiscare”.
- Vom folosi int pentru numarul de cicluri ale codului în autonomie.

String:

String este un tip de date care poate reține un șir de caractere. Spre exemplu o variabilă de tip string poate avea valoarea “Robert”.

NU putem da unei variabile de tip int valoarea unei variabile string. Adică dacă avem variabila a(de tip int) și variabila s(de tip string), atunci nu putem scrie “a = s”(**a** nu poate reține decât numere întregi, iar **s** reprezintă un șir de caractere) sau “s = a”.

Exerciții:

- Ce valoare vor avea variabilele **a**, **b**, **c** și **s** la finalul fiecărui cod?

```
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;
```

Dacă la unul dintre exemple nu vă este clar ce se întâmplă urmăriți codul linie cu linie și scrieți pe o foaie valorile fiecărei variabile. Acest mod de depistare al greșelilor se numește “debug” și este cea mai des întâlnită metodă de a depista erorile unui cod.

```
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;
```

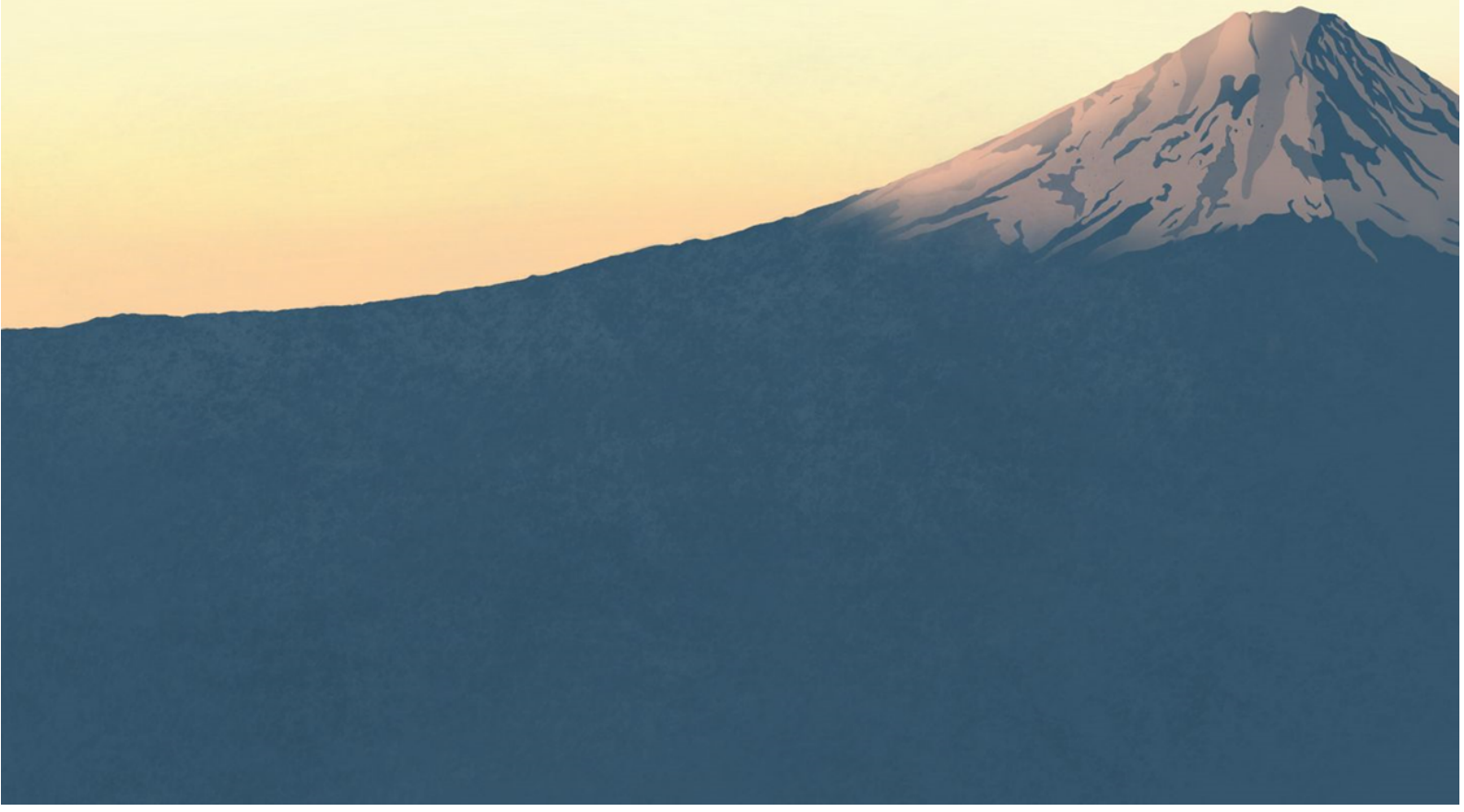
- Precizați valoarea lui **NR** la final și scrieți încă 2 linii de cod astfel încât variabila **x** va avea valoarea 999, iar variabila **NR** va avea valoarea “ALEX”. Nu aveți voie să folosiți “x = 999” sau “NR = “ALEX”.

Probabil vi se pare că acest cod nu are nici un sens. Sunt perfect de acord cu voi, însă, veti ajunge sa lucrati la codul robotului, și vă promit eu că acela chiar nu are nici un sens. Acesta este motivul pentru care trebuie să învățați să urmăriți codul scris de alții și să scrieți chiar voi un cod ușor de urmărit.

Exemplu cod pe robot –>

```
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);
```

CAPITOLUL 3:
The basics



Dacă deschideți un proiect nou în oricare dintre aplicațiile precizate mai sus veți vedea niște linii de cod deja scrise:

```
#include <iostream>

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

În funcție de aplicația folosită veți fi întâmpinați de unul dintre aceste 2 variante, dar ele funcționează identic. Ambele vor afișa pe ecran mesajul ***"Hello world"***. Haideți să vedem ce înseamnă fiecare dintre aceste linii:

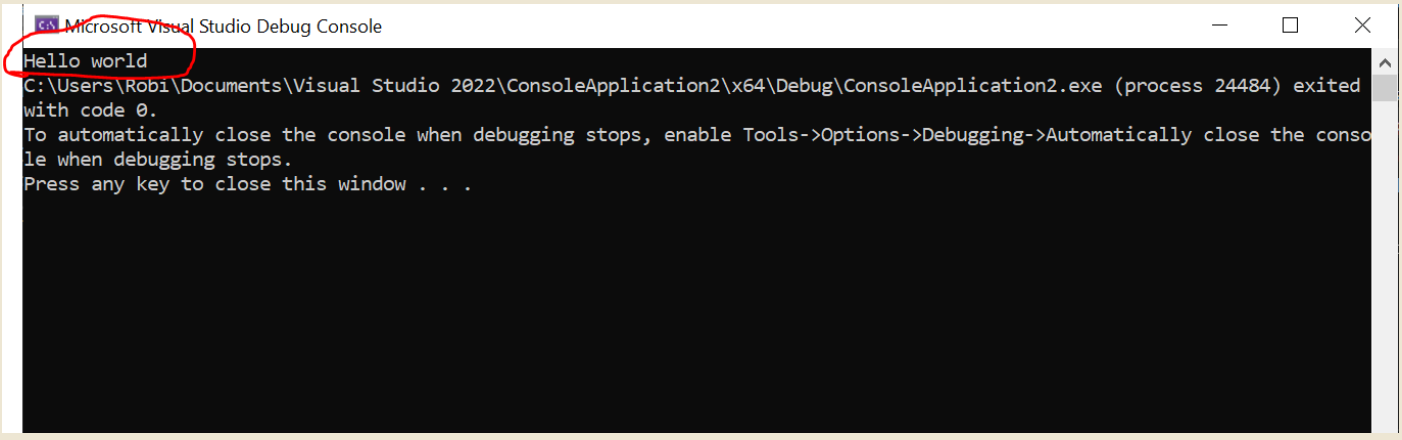
#include <iostream> spune calculatorului că poate folosi libăria ***"iostream"***, librăriile funcționează ca un dicționar, astfel calculatorul știe ce înseamnă celelalte linii de cod deoarece sunt definite în ***"iostream"***. Fără această librărie calculatorul nu ar putea compila operația ***"cout".***

```
#include <iostream>

using namespace std;

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

using namspace std; spune calculatorului că trebuie să adauge singur ***"std::"*** înaintea liniilor de cod. Această linie de cod nu este obligatorie, însă vă va face viața mult mai ușoară. Puteți observa că unul dintre exemple are linia ***"using namespace std;"*** , iar celălalt folosește ***"std::"*** înaintea liniilor de cod. Dacă am continua codul din primul exemplu ar trebuii să scriem ***"std::"*** înaintea fiecărei linii de cod.

Int main() { } definește programul propriuzis. Atunci când compilăm programul el va rula codul din interiorul acoladelor de la ***int main().*** Nu putem folosi comenzi înafara ***int main()***.

cout << ; este operația de scriere: dacă scriem numele unei variabile după ***"<<"*** va fi afișată pe ecran valoarea ei, iar dacă scriem ***"..."*** după ***"<<"*** atunci programul va afișa textul din interiorul ghilimelelor. În cazul nostru va afișa ***Hello world*** deoarece asta este scris între ghilimele.



cin >> ; este folosit pentru a citi de la tastatură valoarea unei variabile.

Spre exeplu, dacă vrem să citim de la tastatură un număr, iar apoi să afișăm dublul sau codul va arăta așa:

```
#include <iostream>

using namespace std;

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

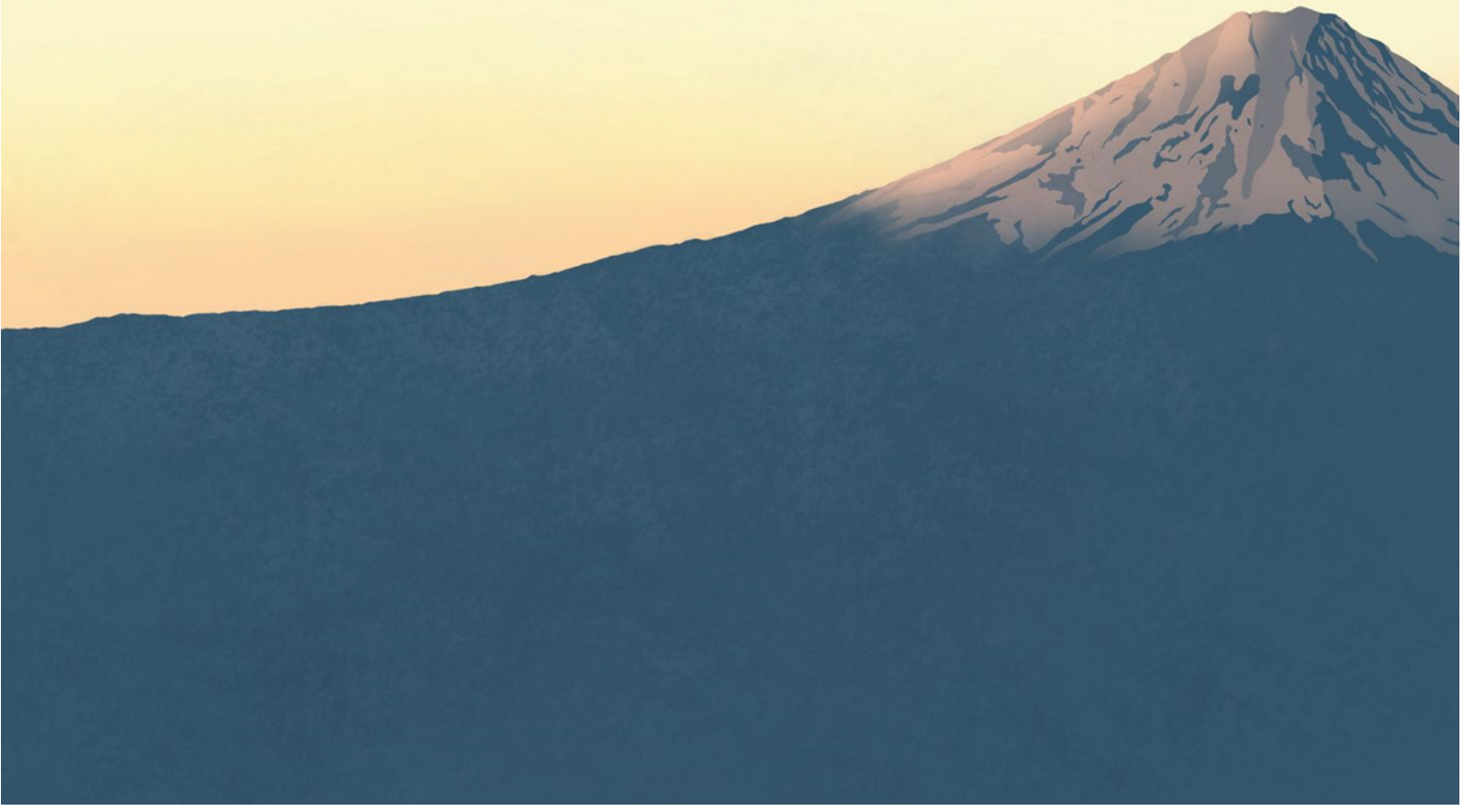
Acest program va aștepta până vom introduce în consolă un număr. Apoi va afișa textul "Dublul numărului cerut este " și valoarea expresiei ***"numar * 2"***.

În consolă arată așa:

Exerciții:

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***

CAPITOLUL 4:
Scrierea în fișiere



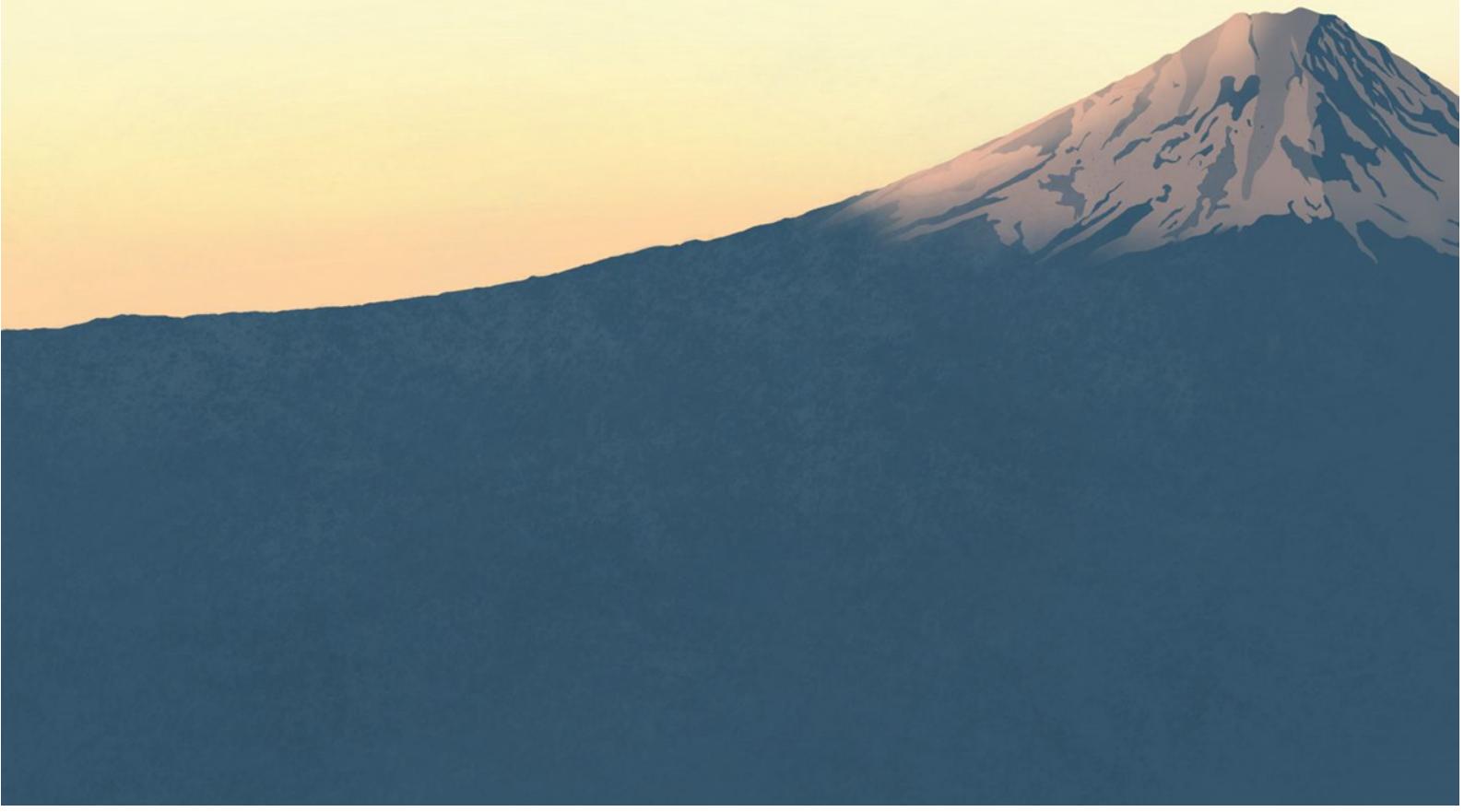
Sintaxa de bază pentru scrierea într-un fișier este:

- ***fin*** funcționează exact la fel ca ***cin***, dar, acesta citește din fișierul "numeleFisierului.in".
- ***fout*** funcționează exact la fel ca ***cout***, dar, acesta scrie în fișierul "numeleFisierului.out".

IMPORTANT: trebuie să adăugați librăria ***<fsteam>***

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

using namespace std;

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

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

CAPITOLUL 5:
Structura de decizie ”if”



În C++, instrucțiunea "if" este folosită pentru a decide dacă o anumită secvență de cod trebuie să fie executată sau nu, în funcție de o condiție dată. Dacă condiția este adevărată, secvența de cod este executată, în caz contrar aceasta este omisă.

Sintaxa instrucțiunii "if" este următoarea:

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

În acest caz, "condiția" este o expresie care poate fi evaluată ca adevărată sau falsă. Dacă condiția este adevărată, secvența de cod dintre acoladele următoare este executată. Dacă condiția este falsă, secvența de cod este omisă și programul trece la următoarea linie de cod după secvența "if".

De exemplu, să presupunem că vrem să scriem un program care citește un număr de la tastatură și afișează mesajul "Numărul este pozitiv" dacă numărul este mai mare decât zero, altfel nu face nimic. Putem folosi instrucțiunea "if" în felul următor:

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

Folosind instrucțiunea **cin** putem citi de la tastatură valoarea unei variabile, iar folosind **cout** putem afișa pe ecran diferite lucruri.

În acest exemplu, "cin" este utilizat pentru a citi un număr de la tastatură, iar "if" este folosit pentru a decide dacă numărul este mai mare decât zero sau nu. Dacă numărul este mai mare decât zero, se afișează mesajul "Numărul este pozitiv" utilizând "cout". În caz contrar, programul nu face nimic și trece la următoarea linie de cod după "if".

În plus față de sintaxa de bază a instrucțiunii "if", există și o variantă numită "if-else", care poate fi folosită pentru a executa o secvență de cod diferită dacă condiția este falsă. Sintaxa este următoarea:

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

De exemplu, să presupunem că dorim să scriem un program care afișează mesajul "Numărul este pozitiv" dacă numărul este mai mare decât zero, mesajul "Numărul este negativ" dacă numărul este mai mic decât zero și un mesaj diferit dacă numărul este zero. Putem folosi instrucțiunea "if-else pentru a realiza acest lucru, în felul următor:

```
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;
    }
}
```

În acest exemplu, instrucțiunea "if" verifică dacă numărul este mai mare decât zero. Dacă acest lucru este adevărat, se afișează mesajul "Numărul este pozitiv". Dacă nu este adevărat, instrucțiunea "else if" verifică dacă numărul este mai mic decât zero. Dacă acest lucru este adevărat, se afișează mesajul "Numărul este negativ". În caz contrar, instrucțiunea "else" este executată, afișând mesajul "Numărul este zero".

În C++, putem utiliza și operatori logici precum "&&", "||" sau "!" în condițiile instrucțiunii "if" pentru a combina mai multe expresii.

operator	ce inseamna	cum il folosim
&&	și	exp. && exp. (este adevarat doar daca ambele sunt adevarate)
||	sau	exp. || exp. (este adevarat daca oricare din ele este adevarata)
!	nu	!exp. (este adevarat daca exp este fals)

De exemplu, putem verifica dacă un număr este între 0 și 10 folosind operatorul "&&", astfel:

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

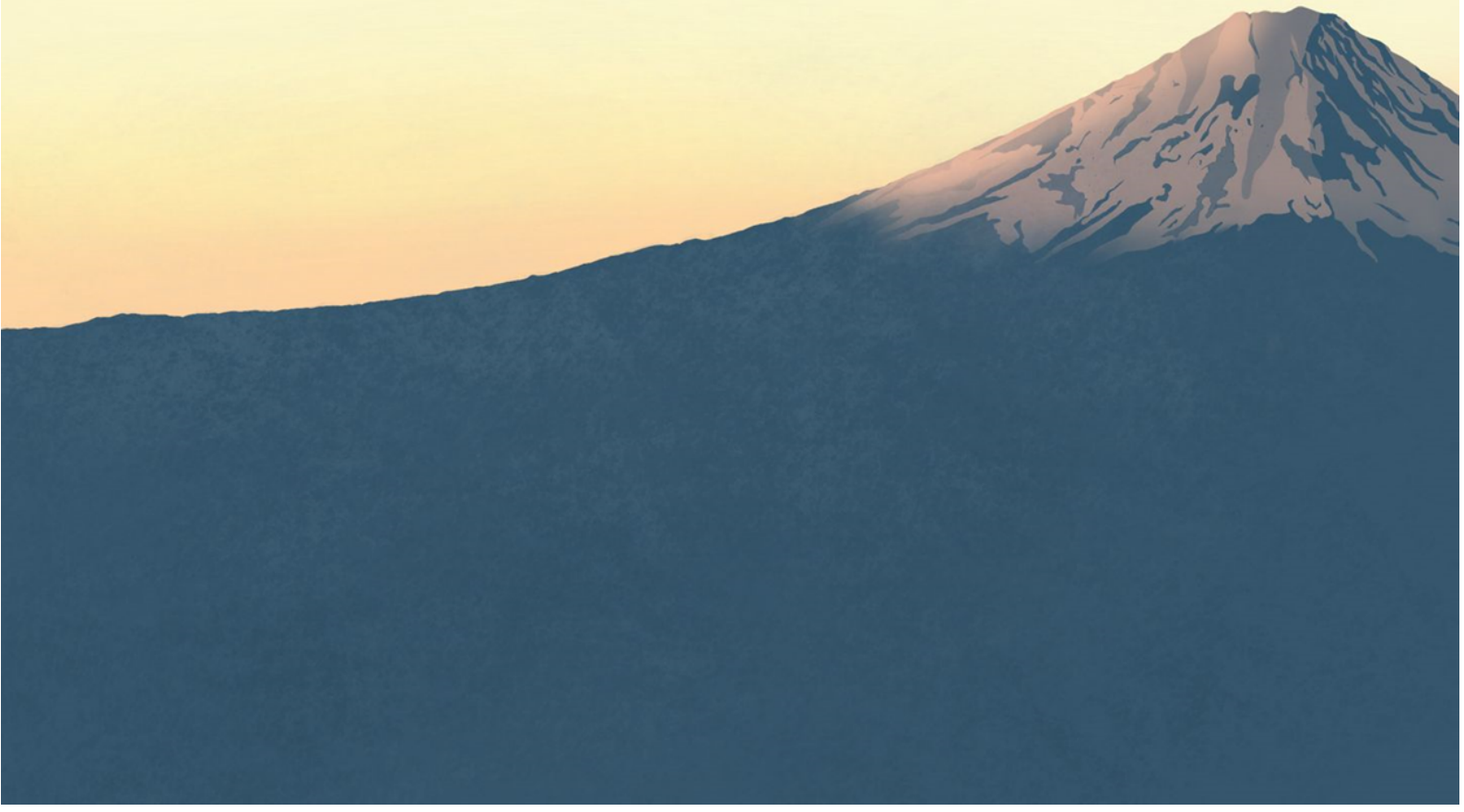
În acest caz, instrucțiunea "if" va fi adevărată doar dacă numărul este mai mare decât zero și mai mic decât 10.

De precizat:
În interiorul unui if (intre acolade) putem introduce orice secvență de cod: putem pune mai multe linii de cod și chiar și alte if-uri.

Exerciții:

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***

CAPITOLUL 6:
Instrucțiuni repetitive



Instrucțiunea "while" permite executarea repetată a unui bloc de cod, atâta timp cât o condiție este adevărată. Structura sintactică a acestei instrucțiuni este următoarea:

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

De exemplu, următorul cod va afișa numerele de la 1 la 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
    }
}
```

Instrucțiunea "while" este utilă atunci când nu cunoști numărul exact de iterări necesare pentru a realiza o anumită sarcină. O condiție este testată înainte de fiecare iterație, iar blocul de cod asociat va fi executat atât timp cât condiția este adevărată.

De precizat:
În interiorul instrucțiunii “while” putem avea alte instrucțiuni repetitive, structuri de decizie sau “break”. Instrucțiunea “brake” face programul să iasă din while și este folosit pentru condiții speciale.

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

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

Exemplu real:
Acesta este scheletul unui cod pentru perioada TeleOp.

Instrucțiunea "for" este un alt tip de instrucțiune repetitivă în C++, care poate fi utilizată pentru a itera printr-o secvență de instrucțiuni de mai multe ori. Acest tip de instrucțiune repetitivă este util atunci când știm exact câte iterații vor fi necesare pentru a realiza o anumită sarcină.

Sintaxa instrucțiunii "for" este următoarea:

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

Inițializare: aceasta reprezintă o instrucțiune care se execută o singură dată, la începutul buclei și este utilizată pentru a inițializa variabilele necesare pentru buclă.

Condiție: este o expresie care este evaluată la fiecare iterație și determină dacă blocul de cod din buclă trebuie să fie executat sau nu. Dacă condiția este adevărată, blocul de cod din buclă este executat. Dacă condiția este falsă, execuția buclei se oprește.

Increment/Decrement: aceasta este o instrucțiune care este executată la sfârșitul fiecărei iterații și este utilizată pentru a actualiza valorile variabilelor necesare pentru următoarea iterație.

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

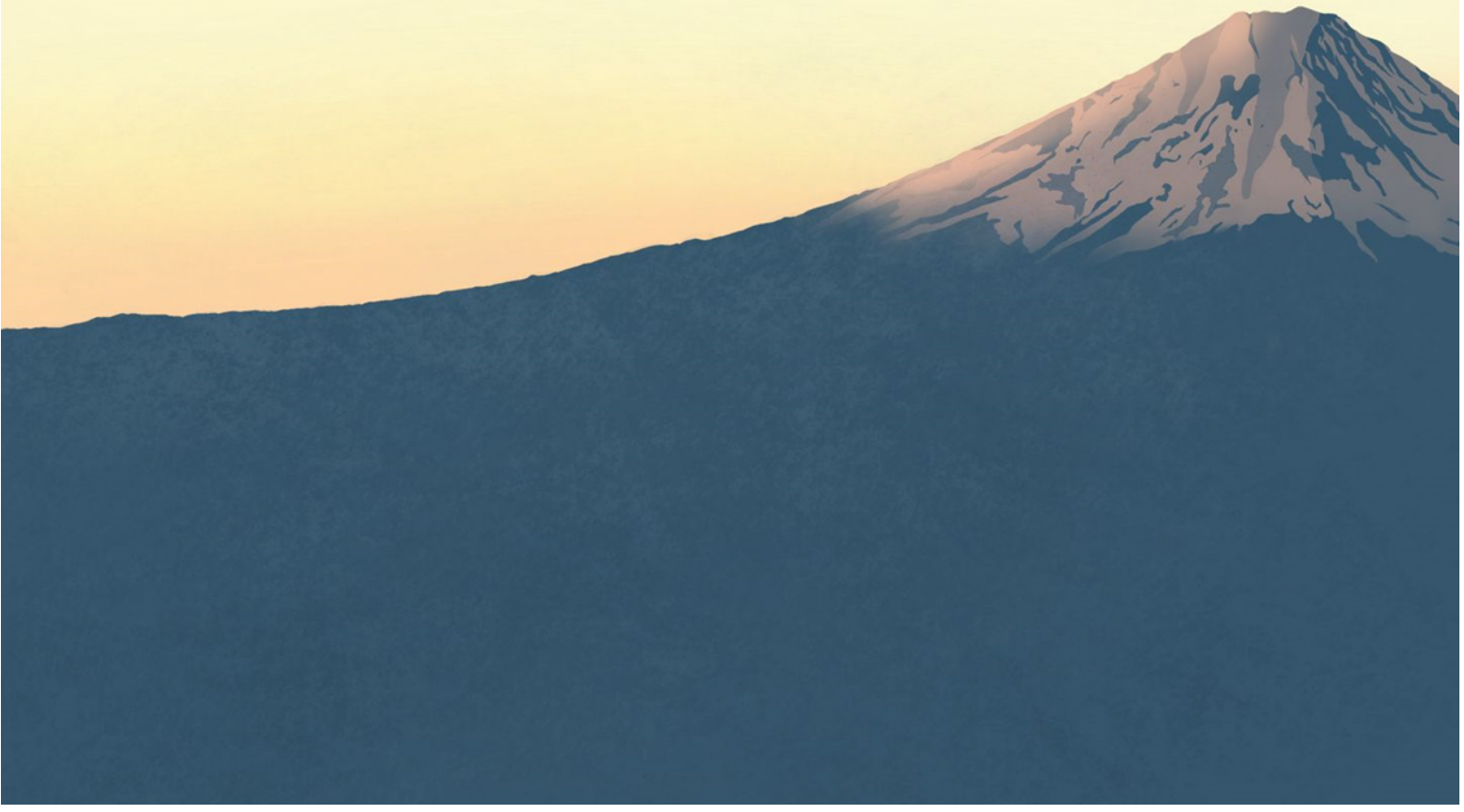
În acest exemplu variabila ***i*** începe cu valoarea 1 și crește până condiția “i <= 10” devine falsă, adică până când ***i*** devine mai mare decât 10. În interior avem instrucțiunea “cout << i” care va afișa valoarea lui ***i*** la fiecare pas. Acest cod va afișa: 1 2 3 4 5 6 7 8 9 10

De precizat:
În interiorul instrucțiunii “for” putem de asemenea să folosim orice alte instrucțiuni, incluzând “break”. Increment/Decrement nu trebuie să fie mereu 1, dacă vrem putem face o instrucțiune ”for” să meargă din 5 în 5 schimbând “i++” cu “i = i + 5”.

Exerciții:

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***

CAPITOLUL 7:
Funcții



În C++, funcțiile sunt blocuri de cod care execută o anumită acțiune atunci când sunt apelate. Acestea pot fi definite într-un fișier separat și apoi apelate dintr-un alt fișier sau pot fi definite în cadrul aceluiași fișier în care sunt apelate. Funcțiile sunt utilizate pentru a organiza codul în bucăți logice și pentru a evita duplicarea codului.

Funcțiile în C++ pot fi clasificate în două categorii:

- Funcții predefinite: Acestea sunt funcții predefinite în biblioteca standard a limbajului C++. Acestea includ funcții matematice precum sqrt(), abs(), funcții de citire și scriere în consolă precum cin și cout, și multe altele.
- Funcții definite de utilizator: Acestea sunt funcții definite de programator pentru a efectua o anumită acțiune specifică. Acestea sunt definite în afara funcției principale a programului și pot fi apoi apelate din interiorul sau din afara programului.

Funcțiile definite de utilizator în C++ pot fi definite cu sau fără parametri. Parametrii sunt variabile care sunt utilizate în interiorul funcției și care pot fi transmise din afara funcției atunci când aceasta este apelată. Funcțiile pot, de asemenea, să returneze o valoare. Această valoare poate fi de orice tip, cum ar fi un întreg, un caracter, un șir de caractere, un obiect etc.

Exemplu:
Dacă avem variabilele de tip **int**: a și b și vrem să aflăm care dintre ele este mai mare avem 2 obțiuni:

Putem folosi instrucțiunea if astfel:

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

Putem folosi funcția ***max()***. Aceasta este o funcție predefinită (este deja implementată în limbajul c++). Aceasta primeste 2 parametrii si returneaza maximul dintre ei.

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

Aceste 2 variante de cod fac același lucru.

În programare există 2 tipuri de variabile:

- **Globale:** acestea sunt declarate înafara funcțiilor (înafara **int main()**) și pot fi folosite oriunde în cod (în interiorul funcțiilor sau în **int main()**).
- **Locale:** acestea sunt declarate în interiorul funcțiilor și nu există înafara lor (putem avea 2 variabile cu același nume în funcții diferite, acestea sunt independente una față de cealaltă).

Putem avea funcții care nu returnează nimic, acestea vor avea tipul variabilei returnate ***void***.

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

Putem folosi funcția ***max()*** pentru a învăța cum se implementează o funcție:

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

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

În dreapta aveți schema generală a unei funcții, iar în stânga este implementarea funcției ***maxim()*** care funcționează similar cu ***max()***.

Funcția primește ca argumente 2 variabile de tip int și le compară returnând care dintre ele este mai mare. Tipul returnat este int deoarece rezultatul va fi de tip 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;
}
```

Exemplu de funcție care returnează 1 dacă numărul dat ca parametru este prim și 0 în caz contrar.

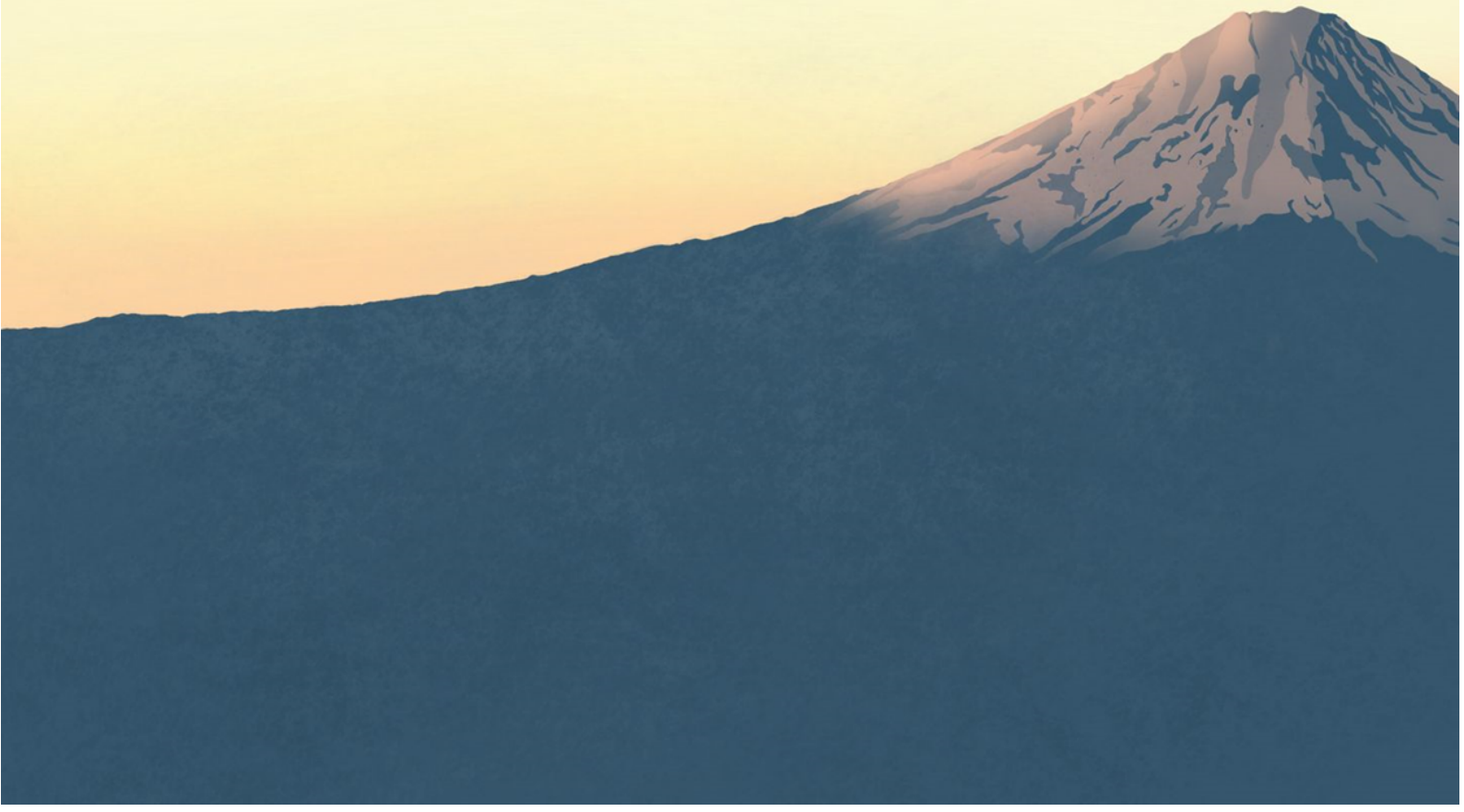
De reținut:
Funcțiile pot avea argumente și pot returna orice tip de variabile. De asemenea, argumentele nu trebuie să fie de același tip și pot exista oricât de multe vrem, iar variabila returnată nu trebuie să aibă același tip ca argumentele (putem avea o funcție care primește ca argumente o variabilă de tip char și 2 de tip long long și returnează o variabilă string).

Exerciții:

1. https://www.pbinfo.ro/probleme/896/factorialf
2. https://www.pbinfo.ro/probleme/897/sumciff
3. https://www.pbinfo.ro/probleme/898/sumfactcif (puteti apela o functie in interiorul altei functii)
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 folosind funcțiile read și solve. ***Int main()*** ar trebuii să arate așa:

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

CAPITOLUL 8:
Librării externe



O librărie externă este o colecție de fișiere de cod precompilat care poate fi utilizată pentru a extinde funcționalitatea programului tău. Acestea sunt adesea scrise în alte limbaje de programare. Spre exemplu **<iostream>** este o librarie, aceasta contine fisierele pentru functiile ***"cin"*** si ***"cout".*** Fara aceasta calculatorul nu ar stii ce inseamna ***"cout"***.

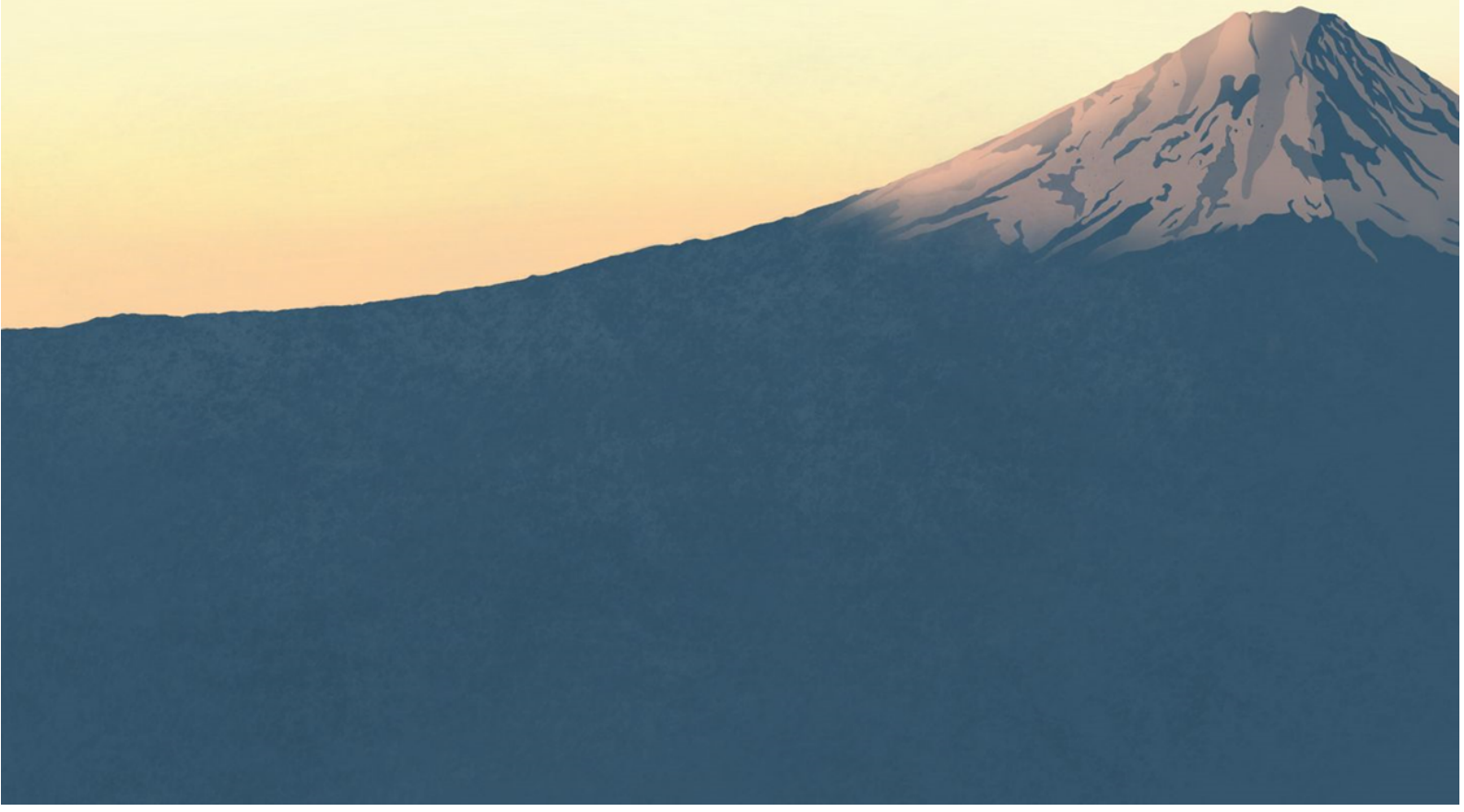
Există o multitudine de astfel de librării externe care ajută la diferite lucruri. Spre exemplu librăria **<cmath>** introduce funcții precum ***"pow(x, y)"*** (ridică numărul x la puterea y).

Majoritatea librăriilor pe care le veți folosi în c++ pe timpul liceului sunt incluse în "biblioteca standard", adică le putem importa folosind comanda ***"#include"***. Dacă vrem să folosim o librărie care nu este inclusă în "biblioteca standard" va trebuii să îi specificăm programului unde să găsească această librărie (nu veți avea nevoie să faceți așa ceva în viitorul apropiat în c++, însă, codul de pe robot este scris cu ajutorul acestor librării "externe").

Așa putem include librăria <string> în programul nostru.

```
#include <string>
```

CAPITOLUL 9: Object oriented programming



În lecțiile trecute am învățat că există mai multe tipuri de variabile. Aceste tipuri de variabile sunt defapt clase, iar variabila în sine este un obiect. Adică ***int*** este o clasă, iar o variabilă de tip ***int*** este un obiect al acestei clase.

Putem defini o clasă nouă dacă vrem, astfel având "un nou tip de variabilă custom". Spre exemplu, putem crea o clasă numită ***"persoana"*** care să aibă implementate 2 variabile (una fiind înălțimea și cealaltă numele).

```
class persoana
{
public:

    int height;
    string name;
};
```

Așa arată declararea unei clase. **Public:** înseamnă că variabilele și funcțiile acestei clase pot fi acccesate de oriunde din cod, dacă am fi făcut aceste variabile **private:** atunci acestea nu ar putea fi accesate decât în interiorul clasei.

```
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;
}
```

Folosind această clasă putem declara obiecte de tipul ***"persoana"*** și putem accesa variabilele din interiorul clasei folosind ***"nume_obiect.height/name"***.

Am declarat 2 obiecte: a și b, iar fiecare dintre ele are câte o variabilă **height** și **name**.

Acest cod va afișa în consolă:

```
Microsoft Visual Studio Debug Console
Babi Are inaltimea 185cm
Emma Are inaltimea 165cm
```

Într-o clasă putem adăuga și funcții.

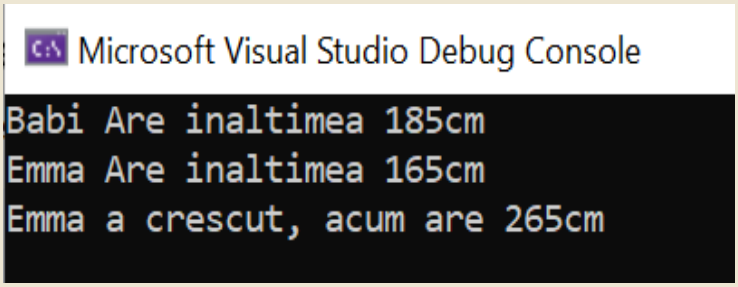
```
class persoana
{
public:

    int height;
    string name;

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

Această funcție primește ca parametru o variabilă int și mărește valoarea **“height”** a obiectului asupra căruia este apelată cu **“val”**.

În exemplu vom folosi funcția **“addheight”** asupra obiectului b, astfel, decât **b.height** va fi schimbat, **a.height** va rămâne la fel.

```
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;
}
```

În consolă:



Un constructor este o funcție care se apelează automat când un obiect este creat. Acesta poate avea și parametrii, spre exemplu putem da ca parametrii numele și înălțimea, astfel, când creem un obiect acesta va avea deja setate numele și înălțimea fără a mai fi nevoie să le scriem noi folosind **“a.height = …”.**

Un constructor este o funcție cu același nume ca și clasa din care face parte și fără un tip de date returnat.

```
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;
    }
};
```

Acest program va afișa exact același lucru ca și celălalt doar că este mai curat.

```
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;
}
```