# Lectii despre C++

C++ este un limbaj de programare puternic și versatil, folosit atât pentru dezvoltarea de aplicații de sistem, cât și pentru aplicații de înaltă performanță.

Iată o scurtă prezentare a lecțiilor esențiale în C++:

**1. Introducere în C++:**

Istoric și caracteristici: C++ a fost dezvoltat de Bjarne Stroustrup ca o extensie a limbajului C, adăugând caracteristici de programare orientată pe obiect (OOP).

Structura unui program C++: Programul C++ începe de obicei cu #include pentru a include biblioteci, urmat de funcția principală main() care este punctul de intrare al aplicației.

**2. Sintaxa de bază:**

Tipuri de date: int, float, double, char, bool.
Variabile: Declararea și inițializarea variabilelor.
Operatori: Aritmetici (+, -, *, /), relaționali (==, !=, <, >), logici (&&, ||, !).

**3. Structuri de control:**

Instrucțiuni de ramificare: if, else, switch.
Instrucțiuni de repetare: for, while, do-while.

**4. Funcții:**

Definirea funcțiilor: Cum se declară și se implementează funcții.
Funcții cu parametri și valori de returnare.
Funcții recursive: Funcții care se apelează pe ele însele.

**5. Programare orientată pe obiect (OOP):**

Clase și obiecte: Definirea și utilizarea claselor, crearea obiectelor.
Encapsulare: Folosirea specificatorilor de acces (public, private, protected).
Moștenire: Crearea de clase derivate, suprascrierea funcțiilor.
Polimorfism: Funcții virtuale și metode pentru a permite diferite implementări ale aceleași funcții în clase derivate.

**6. Manipularea erorilor:**

Excepții: Utilizarea blocurilor try, catch, și throw pentru gestionarea erorilor.

**7. Structuri de date:**
**Tablouri (arrays): Declarație, inițializare, și accesarea elementelor.**
**Structuri (struct): Definirea și utilizarea structurilor de date personalizate.**
**Liste legate, stive și cozi: Implementarea și utilizarea acestora în C++.**

**8. Intrare și ieșire:**
**Fluxuri de date: Utilizarea cin și cout pentru intrare și ieșire.**
**Manipulatoare: Utilizarea manipulatoarelor pentru formatul ieșirii, cum ar fi setw, setprecision.**

**9. Biblioteci standard:**
**Standard Template Library (STL): Utilizarea containerelor, iteratorilor, algoritmilor.**
**Vecuri (vectori), liste, seturi, hărți (maps).**

**10. Memorie și resurse:**
**Gestionarea memoriei: Utilizarea operatorilor new și delete pentru alocarea și eliberarea memoriei dinamice.**
**Pointeri: Declarația și manipularea pointerilor, pointeri la funcții, pointeri la membrii claselor.**

**11. Programare concurentă:**
**Threade și sincronizare: Utilizarea bibliotecii <thread>, std::mutex, std::lock_guard pentru a lucra cu fire de execuție și sincronizarea acestora.**

**12. Proiecte și practică:**
**Dezvoltarea de proiecte: Aplicarea cunoștințelor dobândite în proiecte practice, rezolvarea problemelor și dezvoltarea unor aplicații simple.**

**Să discutăm pe larg fiecare subiect din lista de mai sus.**

**1. Introducere în C++:**
**Istoric și caracteristici: C++ a fost creat în anii 1980 de Bjarne Stroustrup ca o extensie a limbajului C. Este un limbaj de programare general, cu suport pentru programare procedurală, orientată pe obiect și generice.**

**Structura unui program C++:**

```
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

**Acest exemplu include biblioteca iostream pentru intrare/ieșire și definește funcția main care este punctul de intrare al programului.**

**2. Sintaxa de bază:**

**Tipuri de date:**

```
int a = 10;
float b = 5.5;
double c = 10.12345;
char d = 'A';
bool e = true;
```

**Variabile:**

```
int var = 10;
float pi = 3.14f;
```

**Operatori:**

```
int sum = a + b; // Aritmetic
bool isEqual = (a == b); // Relațional
bool isTrue = (a > b) && (c < d); // Logic
```

**3. Structuri de control:**

**Instrucțiuni de ramificare:**

```
if (a > b) {
    std::cout << "a este mai mare decât b" << std::endl;
} else {
```

```
    std::cout << "a nu este mai mare decât b" << std::endl;
}
switch (a) {
    case 1:
        std::cout << "a este 1" << std::endl;
        break;
    case 2:
        std::cout << "a este 2" << std::endl;
        break;
    default:
        std::cout << "a nu este nici 1, nici 2" << std::endl;
}
```

**Instrucțiuni de repetare:**

```
for (int i = 0; i < 10; ++i) {
    std::cout << i << " ";
}
int i = 0;while (i < 10) {
    std::cout << i << " ";
    ++i;
}

i = 0;do {
    std::cout << i << " ";
    ++i;
} while (i < 10);
```

## 4. Funcții:

**Definirea funcțiilor:**

```
int add(int x, int y) {
    return x + y;
}
int main() {
    int result = add(5, 3);
    std::cout << "Rezultatul este: " << result << std::endl;
    return 0;
```

```
}
```

**Funcții recursive:**

```
int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}
```

**5. Programare orientată pe obiect (OOP):**

**Clase și obiecte:**

```
class Car {
public:
    std::string brand;
    std::string model;
    int year;

    void display() {
        std::cout << brand << " " << model << " " << year << std::endl;
    }
};
int main() {
    Car car1;
    car1.brand = "Toyota";
    car1.model = "Corolla";
    car1.year = 2020;
    car1.display();
    return 0;
}
```

**Encapsulare:**

```
class Person {
private:
    std::string name;
    int age;
public:
    void setName(std::string n) {
```

```
        name = n;
    }
    std::string getName() {
        return name;
    }
    void setAge(int a) {
        age = a;
    }
    int getAge() {
        return age;
    }
};
```

Moștenire:

```
class Animal {
public:
    void eat() {
        std::cout << "Eating..." << std::endl;
    }
};
class Dog : public Animal {
public:
    void bark() {
        std::cout << "Barking..." << std::endl;
    }
};
int main() {
    Dog dog;
    dog.eat();
    dog.bark();
    return 0;
}
```

Polimorfism:

```
class Base {
public:
    virtual void display() {
        std::cout << "Base class" << std::endl;
    }
```

```
};
class Derived : public Base {
public:
    void display() override {
        std::cout << "Derived class" << std::endl;
    }
};
int main() {
    Base* basePtr;
    Derived derivedObj;
    basePtr = &derivedObj;
    basePtr->display(); // Va apela funcția din Derived
    return 0;
}
```

**6. Manipularea erorilor:**
**Excepții:**

```
int divide(int a, int b) {
    if (b == 0) {
        throw std::invalid_argument("Diviziune prin zero!");
    }
    return a / b;
}
int main() {
    try {
        int result = divide(10, 0);
        std::cout << "Rezultatul este: " << result << std::endl;
    } catch (std::invalid_argument& e) {
        std::cerr << "Eroare: " << e.what() << std::endl;
    }
    return 0;
}
```

**7. Structuri de date:**
**Tablouri (arrays):**

```
int arr[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; ++i) {
    std::cout << arr[i] << " ";
}
```

**Structuri (struct):**

```
struct Point {
    int x;
    int y;
};
int main() {
    Point p1;
    p1.x = 10;
    p1.y = 20;
    std::cout << "Point: (" << p1.x << ", " << p1.y << ")" << std::endl;
    return 0;
}
```

**Liste legate, stive și cozi: Exemplele următoare sunt simplificate pentru ilustrare.**

```
struct Node {
    int data;
    Node* next;
};
class LinkedList {
private:
  Node* head;
  public:
    LinkedList() : head(nullptr) {}

    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    void display() {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
```

```
    }
};
int main() {
    LinkedList list;
    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.display();
    return 0;
}
```

**8. Intrare și ieșire:**

**Fluxuri de date:**

```
int main() {
    int num;
    std::cout << "Introdu un număr: ";
    std::cin >> num;
    std::cout << "Numărul introdus este: " << num << std::endl;
    return 0;
}
```

**Manipulatoare:**

```
#include <iomanip>
int main() {
    double pi = 3.14159;
    std::cout << "Pi cu două zecimale: " << std::fixed <<
std::setprecision(2) << pi << std::endl;
    std::cout << "Pi cu cinci zecimale: " << std::fixed <<
std::setprecision(5) << pi << std::endl;
    return 0;
}
```

**9. Biblioteci standard:**

**Standard Template Library (STL):**

```
#include <vector>
```

```
#include <algorithm>
int main() {
    std::vector<int> vec = {4, 1, 8, 3, 9};
    std::sort(vec.begin(), vec.end());
    for (int num : vec) {
        std::cout << num << " ";
    }
    return 0;
}
```

Vecuri (vectori), liste, seturi, hărți (maps):

```
#include <vector>
#include <list>
#include <set>
#include <map>
int main() {
    // Vector
    std::vector<int> vec = {1, 2, 3, 4};
    vec.push_back(5);
    for (int num : vec) {
        std::cout << num << " ";
    }

    // List
    std::list<int> lst = {1, 2, 3, 4};
    lst.push_back(5);
    for (int num : lst) {
        std::cout << num << " ";
    }

    // Set
    std::set<int> s = {1, 2, 3, 4};
    s.insert(5);
    for (int num : s) {
        std::cout << num << " ";
    }

    // Map
    std::map<int, std::string> m;
```

```
    m[1] = "one";
    m[2] = "two";
    for (const auto& pair : m) {
        std::cout << pair.first << " => " << pair.second << std::endl;
    }

    return 0;
}
```

**10. Memorie și resurse:**

**Gestionarea memoriei:**

```
int* p = new int;
*p = 10;
std::cout << *p << std::endl;
delete p;
```

**Pointeri:**

```
int var = 20;
int* ptr = &var;
std::cout << "Valoarea lui var: " << var << std::endl;
std::cout << "Adresa lui var: " << ptr << std::endl;
std::cout << "Valoarea la care pointează ptr: " << *ptr << std::endl;
```

**11. Programare concurentă:**

**Threade și sincronizare:**

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mtx;
void printThreadID(int id) {
    mtx.lock();
    std::cout << "Thread ID: " << id << std::endl;
    mtx.unlock();
```

```
}
int main() {
    std::thread t1(printThreadID, 1);
    std::thread t2(printThreadID, 2);

    t1.join();
    t2.join();

    return 0;
}
```

**12. Proiecte și practică:**

**Aplicarea cunoștințelor dobândite prin dezvoltarea unor proiecte practice este esențială. De exemplu, un proiect simplu ar putea fi un sistem de gestionare a bibliotecii, care include funcționalități precum adăugarea, ștergerea și afișarea cărților, gestionarea utilizatorilor și împrumuturilor.**