ÎNTREBĂRI INTERVIU C++

# Care este diferența dintre C și C++?

* 1. C este un limbaj de programare procedural, în timp ce C++ este atât procedural, cât și orientat pe obiecte.
     1. În C, programarea se concentrează pe funcții și proceduri, unde codul este organizat pe baza unor funcții care procesează datele. În schimb, C++ introduce programarea orientată pe obiecte (OOP), care se concentrează pe obiecte și interacțiunea dintre ele, permițând o structurare mai naturală a codului și o mai bună modularizare.
  2. C++ folosește concepte de programare orientată pe obiecte (OOP) precum clase și obiecte, fiind mai ușor de asociat cu lumea reală și de rezolvat probleme. De exemplu, în jocuri, un jucător poate interacționa cu un inamic.
     1. C++ permite crearea de clase care modelează obiecte reale. Prin intermediul acestor clase, poți reprezenta entități din lumea reală precum jucători, inamici sau alte obiecte. Fiecare instanță a unei clase (obiect) poate avea propriile sale caracteristici (proprietăți) și comportamente (metode), ceea ce face rezolvarea problemelor mai intuitivă și mai apropiată de lumea reală.
  3. C++ este mai sigur decât C, datorită mecanismului de ascundere a datelor, folosind specificatori de acces (private, protected, public).
* În C++, ascunderea datelor este realizată prin specificatori de acces:
  + Private: Membrii declarați private sunt accesibili doar din interiorul clasei, oferind control total asupra accesului la date.
  + Protected: Membrii protected pot fi accesați din interiorul clasei și al claselor derivate, dar nu din exteriorul acestora.
  + Public: Membrii public sunt accesibili de oriunde, ceea ce oferă flexibilitate, dar trebuie utilizat cu atenție.
  1. C++ oferă, de asemenea, caracteristici precum moștenirea, polimorfismul, încapsularea, abstractizarea, supraîncărcarea operatorilor și constructorii.

 Moștenirea: Permite unei clase să deriveze dintr-o altă clasă, moștenind proprietățile și metodele acesteia, ceea ce ajută la reutilizarea codului.

 Polimorfismul: Permite folosirea mai multor forme pentru același concept, de exemplu, o funcție cu același nume poate avea comportamente diferite în clase diferite.

 Încapsularea: Reunește datele și metodele care operează asupra acestor date într-o singură unitate (clasă), protejând datele de acces neautorizat.

 Abstractizarea: Permite concentrarea asupra esenței obiectelor, ascunzând detaliile complexe ale implementării.

 Supraîncărcarea operatorilor: Permite redefinirea modului în care funcționează operatorii pentru obiectele definite de utilizator, făcând interacțiunea cu acestea mai intuitivă.

 Constructorii: Sunt funcții speciale care inițializează obiectele atunci când sunt create, oferind control asupra modului în care obiectele își inițializează datele.

* 1. C++ este un limbaj de programare generic – STL
  + În C++, programarea generică permite scrierea de cod care poate funcționa cu orice tip de date, fără a specifica în mod explicit tipul acestora la momentul scrierii codului. Un exemplu central al acestei abordări este STL (*Standard Template Library*).

# Ce este un namespace în C++?

**Dacă există două sau mai multe funcții cu același nume definite în biblioteci diferite, cum va ști compilatorul la care să se refere?**

De aceea, punem acele funcții, clase, sau obiecte sub un nume (namespace).

### Explicație pe scurt:

Pentru a evita conflictele între funcții sau clase cu același nume din biblioteci diferite, **C++** folosește conceptul de **namespace** (spațiu de nume). Prin utilizarea unui namespace, putem organiza funcțiile, clasele sau obiectele astfel încât compilatorul să știe exact la ce element să se refere, evitând ambiguitățile.

# Ce este using namespace std în C++?

Aceasta instruiește compilatorul că vei folosi spațiul de nume (namespace) numit std.  
Dorești să folosești toate funcțiile, clasele și obiectele care sunt sub std.  
Astfel, nu mai trebuie să scrii std::cout, ci poți scrie simplu cout.

În C++, pentru a evita să scrii în mod repetat std:: în fața funcțiilor și obiectelor standard (precum cout sau cin), poți folosi instrucțiunea using namespace std. Aceasta îți permite să accesezi toate elementele din spațiul de nume std fără a-l specifica de fiecare dată.

# Ce este iostream în C++?

Este un fișier header care include obiecte precum cin pentru a obține date din fluxul de intrare și cout pentru a trimite date către fluxul de ieșire.  
Flux (stream) - este fluxul de date (0/1) către și dinspre computer.

Fișierul iostream din C++ oferă funcționalități esențiale pentru manipularea inputului și outputului, prin intermediul obiectelor standard cin și cout. Acestea permit citirea și scrierea de date în terminal (sau alte fluxuri) utilizând conceptele de flux de date, ceea ce facilitează interacțiunea cu utilizatorul.

# Numărul minim de funcții într-un program C++?

Un program C++ trebuie să aibă cel puțin o funcție - main().

În **C++**, funcția main() este punctul de intrare al oricărui program. Aceasta este funcția care este apelată prima dată atunci când programul este executat, iar fără ea, compilatorul nu ar ști de unde să înceapă execuția. Chiar și în cele mai simple programe C++, main() este esențială pentru a asigura că programul funcționează corect.

# Este C++ un limbaj de programare orientat pe obiect? Și de ce?

Da, C++ poate fi considerat un limbaj de programare orientat pe obiect (OOP) datorită mai multor concepte esențiale, cum ar fi:

1. Clase: O clasă este un model sau un tip de dată definit de utilizator care poate conține date (atribute) și funcții (metode) pentru a manipula aceste date.
2. Obiecte: Obiectele sunt instanțe ale claselor. Ele combină datele și funcțiile într-un singur entitate, permițând interacțiunea între diferitele părți ale programului.
3. Ascunderea datelor: C++ oferă specificatori de acces (private, protected, public) care controlează accesul la datele dintr-o clasă, asigurând astfel că datele sensibile sunt protejate de accesul neautorizat.
4. Polimorfism: Acest concept permite utilizarea aceleași funcții sau metode în diferite contexte, având comportamente diferite în funcție de tipul de obiect utilizat.
5. Moștenire: C++ permite crearea de noi clase bazate pe clase existente, moștenind atributele și metodele acestora, facilitând reutilizarea codului și extensibilitatea programului.

Aceste concepte permit C++ să implementeze principiile programării orientate pe obiect, făcându-l un limbaj puternic și flexibil pentru dezvoltarea de aplicații complexe.

# Ce este o clasă?

O clasă este un tip definit de utilizator care conține variabile de membru și funcții. Prin urmare, o clasă este, de fapt, un model sau un șablon pentru un obiect.

Aceasta înseamnă că o clasă nu ocupă spațiu sau memorie în sine; în schimb, este utilizată pentru a crea obiecte care conțin date și funcționalitate.

1. Ce este un obiect?

**Ce este un obiect?**

Un obiect este o instanță a unei clase. (Un obiect este creat prin referință la clasă.)  
De exemplu, **Planul unei case** reprezintă clasa, iar **Casa efectivă** reprezintă obiectul.

Aceasta înseamnă că un obiect ocupă spațiu și memorie.

# Câte obiecte putem crea dintr-o clasă?

Putem crea un număr nelimitat de obiecte dintr-o singură clasă.

# Caracteristicile Programării Orientate pe Obiect (OOP)

* 1. Clasă:
  2. Obiect:
  3. Încapsulare:
  4. Ascunderea datelor:
  5. Abstracție:
  6. Moștenire:
  7. Polimorfism:

# Ce este Încapsularea?

Încapsularea este procesul de legare a variabilelor și funcțiilor împreună. Este aplicată pentru a preveni accesul direct la date pentru motive de securitate.

# Ce este Abstractizarea?

Abstractizarea în C++ reprezintă procesul de ascundere a implementărilor interne și prezentarea doar a detaliilor necesare. Aceasta permite utilizatorilor să interacționeze cu un obiect fără a fi nevoie să înțeleagă complexitatea sa internă, facilitând astfel utilizarea și înțelegerea codului.

# Ce este Moștenirea?

Moștenirea este capacitatea unei clase de a prelua proprietățile și caracteristicile unei alte clase. Aceasta se referă la următoarele relații:

* Clasa Părinte -> Clasa Copil
* Clasa de Bază -> Clasa Derivată

Moștenirea permite reutilizarea codului, facilitând extinderea funcționalităților unei clase fără a fi nevoie să se rescrie codul existent.

# Ce este Supraîncărcarea Funcțiilor?

Supraîncărcarea funcțiilor se referă la capacitatea de a avea două sau mai multe funcții cu același nume, dar cu tipuri de parametri diferite sau cu un număr diferit de parametri. Aceasta permite utilizarea aceluiași nume de funcție pentru operații similare, facilitând astfel scrierea unui cod mai clar și mai organizat.

# Ce este Suprascrierea Funcțiilor?

Suprascrierea funcțiilor apare atunci când o clasă derivată moștenește o funcție dintr-o clasă de bază și îi oferă o implementare nouă. Aceasta permite clasei derivate să își redefinească comportamentul funcției moștenite, adaptându-se astfel la nevoile specifice ale clasei respective.

# Ce este Polimorfismul?

Polimorfismul înseamnă "forme multiple". Acesta se referă la capacitatea de a avea mai multe funcții cu același nume, dar cu funcționalități diferite. Polimorfismul se poate manifesta prin:

* Supraîncărcarea funcțiilor: Acelasi nume de funcție, dar cu parametri diferiți.
* Suprascrierea funcțiilor: O funcție definită în clasa de bază este redefinită în clasa derivată, având astfel un comportament specific pentru fiecare clasă.

Această caracteristică permite utilizarea unei funcții în mai multe contexte, îmbunătățind flexibilitatea și reutilizabilitatea codului.

# Ce tipuri de Polimorfism există?

* 1. Polimorfismul la timpul de execuție: Este cunoscut și sub denumirea de polimorfism dinamic. Acesta se referă la decizia care se ia la momentul execuției cu privire la ce funcționalitate trebuie apelată. Un exemplu este suprascrierea funcțiilor, unde funcția apelată (fie din clasa de bază, fie din clasa derivată) este determinată la timpul de execuție, în funcție de obiectul care invocă funcția. Acest tip de polimorfism permite implementarea unor comportamente diferite în funcție de contextul specific al aplicației.
  2. Polimorfismul la timpul de compilare: Este cunoscut și sub denumirea de polimorfism static. Acest tip de polimorfism se referă la decizia luată la timpul compilării cu privire la ce funcționalitate trebuie apelată. Un exemplu de polimorfism la timpul de compilare este supraîncărcarea funcțiilor, unde funcțiile cu același nume, dar cu tipuri sau numere diferite de parametri sunt diferențiate de compilator înainte de execuție. Acest tip de polimorfism permite o alegere clară a funcției care trebuie apelată, îmbunătățind eficiența și claritatea codului.

# Ce este o Funcție Inline?

O funcție inline în C++ este o funcție care este definită cu ajutorul cuvântului cheie inline.Aceste funcții sunt utilizate pentru a reduce suprasarcina apelurilor de funcții.Când o funcție este marcată ca inline, compilatorul are permisiunea de a înlocui apelurile la acea funcție cu codul efectiv al funcției, evitând astfel costul asociat cu apelurile de funcție. Acest lucru poate îmbunătăți performanța, mai ales în cazul funcțiilor mici care sunt apelate frecvent. Totuși, utilizarea funcțiilor inline poate crește dimensiunea codului generat, deoarece codul funcției este replicat de fiecare dată când este apelat.

# Ce sunt Specificatorii de acces și caracteristicile acestora?

Specificatorii de acces în C++ sunt utilizati pentru a controla vizibilitatea și accesibilitatea membrilor unei clase. Aceștia definesc cum pot fi accesate variabilele și funcțiile dintr-o clasă. Există trei specificatori de acces principali:

* 1. Public: Membrii declarați ca fiind publici sunt accesibili din afara clasei.
  2. Protected: Membrii declarați ca fiind protejați nu pot fi accesați din afara clasei, dar pot fi accesibili în clasele derivate (moștenite).
  3. Private: Membrii declarați ca fiind privați nu pot fi accesați din afara clasei.

# Ce este o variabilă statică?

Membru static ar fi alocat memorie doar o singură dată. Și această memorie este partajată de ambele obiecte. Membrii de date statice aparțin clasei și sunt comuni pentru toate obiectele.

# Ce este un constructor?

Un constructor în C++ este o metodă specială care este apelată automat atunci când un obiect al unei clase este creat.

class Test { // numele clasei

public:

Test() { // Constructor

cout << "Hello";

}

};

Constructorul are același nume ca și numele clasei. Acesta inițializează obiectele unei clase.

# Ce este un constructor de copiere?

Constructorul de copiere creează un obiect prin inițializarea acestuia cu un obiect din aceeași clasă, care a fost creat anterior.

# Care este diferența între delete și delete[]?

delete este folosit pentru a elibera o unitate de memorie, în timp ce delete[] este folosit pentru a elibera un array.

# Ce este o funcție virtuală?

O funcție virtuală este o funcție membru care este prezentă în clasa de bază și este redefinită de clasa derivată.

# Ce este un limbaj de programare generic?

Genericul înseamnă generalizat, adică funcționalitatea scrisă pentru un tip de date funcționează și pentru alte tipuri de date. C++ este un limbaj de programare generic datorită:

* **Template-urilor**: funcții template și clase template.
* **STL (Standard Template Library)**: containere care pot stoca orice tip de date și algoritmi care pot efectua operații pe acel tip de date.

# Ce este STL?

STL (Standard Template Library) este o bibliotecă standard care facilitează programarea generică. STL are trei componente principale:

* 1. Containere: structuri care stochează datele.
  2. Algoritmi: funcții care aplică operații asupra datelor stocate în containere.
  3. Iteratori: instrumente care indică poziția într-un container, permițând parcurgerea acestuia, cum ar fi container\_name.start().

# Poți numi câteva containere în STL?

* 1. Array: Un container care stochează elemente de dimensiune fixă.
  2. Vector: Un container dinamic care permite redimensionarea automată.
  3. List: O structură de date care stochează elemente într-o secvență dublu înlănțuită.
  4. Stack: Un container care urmează principiul LIFO (Last In, First Out).
  5. Queue: Un container care urmează principiul FIFO (First In, First Out).
  6. Priority Queue: Un container în care elementele sunt procesate în funcție de prioritate.
  7. Map: Un container care stochează perechi cheie-valoare, permițând accesul rapid la valori prin intermediul cheilor.
  8. Set: Un container care stochează elemente unice, fără ordonare specifică.

# Care este diferența între array și vector?

* Array: Are o dimensiune fixă și nu poate fi extins la timpul de execuție.
* Vector: Este un array dinamic, al cărui dimensiune se poate crește automat la timpul de execuție.

# Ce este/De ce este un template?

Un template este un bloc de cod generic care ne ajută să limităm utilizarea supraîncărcării funcțiilor.

# Ce container ai folosi dacă vrei să accesezi rapid un element?

* 1. Array - [], at()
  2. Vector - [], at()
  3. Map - [key\_value], at(key\_value)

# Întrebări teoretice

* 1. Dispozitivul colectează temperatura întreaga zi (la fiecare minut) și generează un fișier (multe numere, în orice ordine, valori duplicate). Cum ai elimina valorile duplicate și ai sorta temperatura?
     1. Încărcați datele într-un container de tip Set. (10, 10, 20, 15, 30, 31, 25) -> (10, 15, 20, 25, 30)
     2. Încărcați datele într-un container de tip Vector -> customUni() -> algoritm de sortare.
  2. Ai un record de angajați (emp\_id, nume, număr de telefon). Ce container ai folosi pentru a procesa aceste date?
     1. Container de tip map
     2. Container în Container (map în map) std::map<int, std::map<string, string>>