Programare Orientata pe Obiecte

-teorie-

0. Class vs Struct

Clasele reprezintă o extindere a structurilor de date. Acestea pot să conțină date, la fel ca structurile, dar îninteriorul lor se pot afla și funcții. Noțiunea aceasta de funcții din interiorul unei structuri de date constituie diferența majoră dintre cele două modalități de reprezentare a datelor.
Mențiune: Aici vorbim de struct-ul din C standard. În C++, se pot adăuga funcții în interiorul unei structuri de date obișnuite și nu diferă atât de mult de o clasă.

1. Descrieti pe scurt in ce consta mecanismul de incapsulare:

Mecanismul de incapsulare este procesul in care se creeaza un nou tip de date (abstract) definind o clasa ca fiind formata din campuri(structuri de date) si metode(functii/algoritmi). Fiecare camp sau metoda are un atribut de acces **: private** (poate fii accesat doar de metodele clasei) **, protected** (poate fi accesat si de metodele claselor derivate), **public** (poate fii accesat de oriunde din program). Daca nu este pus un atribut de acces atunci atributul de acces implicit va fi private.

Exemplu de definire a unei clase :

```
class NumeClasa {
        [modificator_de_acces]:
        date;
        metode;
        } [numeObiecte de tipul NumeClasa];
```

2. Spuneti ce este obiectul implicit al unei metode si descrieti pe scurt proprietatile pe care le cunoasteti despre acesta.

Obiectul implicit este obiectul care apeleaza metoda. In cadrul metodei campurile obiectului implicit nu vor mai avea specificat carui obiect apartin, subintelegandu-se ca ele apartin obiectului implicit (adica a celui pe care o apeleaza). Obiectul implicit al un metode se poate accesa si cu ajutorul pointerului *this in modul acesta : this->a; (pentru variabile) / this-

>f(); (pentru metode).(este transmit in metoda prin referinta(referinta constanta daca metoda este constanta))

3. Descrieti pe scurt mostenirea virtuala si scopul in care este folosita.

Mostenirea virtuala este o metoda de implementare a polimorfismului de executie . Mostenirea virtuala este necesara atunci cand avem 2 clase derivate dintr-o clasa de baza iar o a patra clasă derivată din cele două. Se foloseşte derivarea virtuală pentru a evita o problemă de ambiguizare (problema diamantului).

4. Descrieti pe scurt proprietatile unui camp constant al unei clase.

Un camp constant al unei clase este un camp a carei valoare nu se poate modifica. El se poate initializa doar in lista de initializare a constructorului si nicaieri in alta parte. Pentru a implementa un camp constant se adauga inainte tipului de date cuvantul cheie const. De asemenea daca acest camp constant este un obiect , el nu poate apela.

5. Descrieti pe scurt mecanismul de tratare a exceptiilor.

Sunt folosite cuvintele cheie try, throw si catch. Se implementeaza in felul urmator :

* Semnalizarea aparitie un exceptii prin intermediul unei valori care reprezinta exceptia respectiva
* Receptionarea valorii prin care este semnalizata exceptia
* Tratarea (rezolvarea) exceptiei semnalizate si receptionate

In C++ se reprezinta astfel :

```
try {
        // instructiuni care se executa pana la posibila aparitie a exceptiei
if(test) throw valoare
        // instructiuni care se executa daca nu survine exceptia}
catch (tip valoarea){
        //instructiuni pentru tratarea / rezolvarea exceptiei }
```

Daca aruncam o valoare de anumit tip si nu exista un catch care sa prinda valori de acel timp, atunci vom avea eroare de executie ("Unhandled exception"). Exista deasemenea si posibilitatea de a pune catch(...) (fara tip) care prinde orice tip de valoare.

6. Descrieti pe scurt diferenta dintre un pointer si o referinta.

Referinta este practic un pointer constant care se dereferentiaza automat (un alias pentru un obiect care odata atribuita unui obiect nu se mai schimba). Prin modificarea referintei se modifica si obiectul in sine. Pointerul "pointeaza" (arata) catre un obiect , dar

obiectul spre care pointeaza poate fii schimbat. In schimb, pointerul cand este declarat de un anumit tip nu mai poate pointa catre un tip diferit.

7. Descrieti pe scurt functiile virtuale si scopul in care sunt folosite.

O functie virtuala este o functie care este declarata ca fiind virtual in clasa de baza si *redefinita* de o clasa derivata. Pentru a declara o functie ca fiind virtuala , declararea sa este precedata de cuvantul cheie "virtual". Redefinirea functiei in clasa derivata modifica si are prioritate fata de definitia functiei din clasa de baza. Functiile virtuale reprezinta o metoda de implementare a polimorfismului de executie. O functie virtuala pura este o functie virtuala care nu are definitie in clasa de baza,deci trebuie definite in clasa derivate.

Functie virtual pura: virtual tip nume_functie(//lista de parametri) = 0;
Functie virtuala: virtual tip nume_functie(//lista de parametri);

*O clasa care contine cel putin o functie virtuala pura s**e numeste abstracta (***si nu poate fi instantiata)

8. Descrieti pe scurt functiile statice si scopul in care sunt folosite.(metodele statice)

Declararea unei functii statice este precedata de cuvantul cheie "static". O astfel de functie are urmatoarele restrictii :

*Ea poate sa aiba acces doar la alti membri (campuri de date/functii) de tip static ai clasei si bineinteles la functiile si datele globale.

*Ele nu pot avea un pointer de tip *this (deci nu au obiect implicit)

*Nu poate exista o versiune static si una non-static ale aceleasi functii.

Functiile statice pot fi folosite fara a mai declara un obiect, prin intermediul operatorului de rezolutie :: , ele fiind independente de obiect. Functiile statice si campurile de date statice au fost create pentru a se pastra principiul incapsularii. Scopul functiilor statice este acela ca pot "preinitializa" datele particulare de tip static, inainte de creearea efectiva a unui obiect.

9. Descrieti pe scurt diferenta dintre transferul parametrilor unei functii prin valoare si prin referinta constanta.

Transferul prin valoare reprezinta copierea valorii transmise ca parametru actual in parametru formal , care este creat pe stiva la lansarea in executie a functiei , ca o variabila locala (automatica), in timp ce **transferul prin referinta** constanta evita consumul suplimentare

de memorie pentru crearea unei alte variabile si ne asigura in acelasi timp ca variabila transmisa ca referinta constanta nu va fi modificata .

OBS. Daca transmitem obiecte prin apel prin referinta, nu se mai creeaza noi obiecte temporare, ci se lucreaza direct pe obiect trimis ca referinta.(nu se mai apeleaza copy-constructorul si nici destructorul).

10. Descrieti pe scurt pe scurt cum se comporta destructorii la mostenire.

Ordinea de executie a destructorilor este inversa ordinii de executie a constructorilor (adica de la clasele derivate spre clasa de baza). In cazul mostenirii multiple ordinea de executie a destructorilor e de la dreapta la stanga in lista de derivare.

11. Creare dinamica de obiecte.

Obiectele se pot crea dinamic folosind instructiunea "new" pentru alocarea memoriei in momentul executiei in heap. Constructorul initializeaza zona de memorie alocata. Eliberarea memoriei alocata astfel se poate face cu ajutorul instructiunii "delete". Cand un obiect este creat dinamic, metodele si campurile de date se acceseaza cu ajutorul operatorului "->" si nu cu ajutorul operatorului "." .

12. Proprietatile campurilor statice.

Proprietatile campurilor statice sunt urmatoarele :

* Campurile statice sunt comune tuturor obiectelor din clasa.
* Sunt independente de obiectele din clasa.
* Sunt create la lansare programului.
* Functiile statice pot opera doar asupra variabilelor statice din clasa.
* Sunt declarate in interiorul clasei, dar trebuie declarate si global.
* In cadrul acestor functii nu exista pointerul *this , pentru ca aceste metode se apeleaza indiferent daca exista sa nu obiecte instantiate.
*functiile statice nu pot avea natura virtuala

Sintaxa: class A{ static void numara(); }

Exemplu de utilizare: metodele statice sunt utilizate pentru accesarea datelor statice, care au fost declarate ca fiind private(principiul incapsularii)

13. Proprietatile destructorului.

Proprietatile destructorilor sunt :

* Intr-o clasa exista un singur destructor.

* Exista un destructor implicit, pe care il putem suprascrie.

* In cazul mostenirii ordinea de executie a destructorilor este inversa ordinii de executie a constructorilor (adica de la clasele derivate spre clasa de baza).

**** Se declara astfel*** : class A { ~A();};.

* Destructorul se apeleaza automat in momentul iesirii din blocul in care a fost declarat obiectul.

14. Descrieti diferenta dintre transferul parametrilor unei functii prin pointeri si prin referinta.

Transferul parametrilor prin pointeri reprezinta copierea adresei transmise ca parametru actual in parametrul formal, care este creat pe stiva la lansarea in executie a functiei ca variabila locala, modificarile asupra parametrilor fiind vizibile si in afara functiei, in timp ce transferul parametrilor prin referinta reprezinta crearea unei referinte locale catre variabila transmisa ca parametru actual la lansarea in executie a functiei.

15. Descrieti pe scurt cum este implementat mecanismul de control al tipului in timpul executiei (RTTI).

RTTI cuprinde operatorul typeid si operatia dynamic_cast<>.

->"typeid()" permite aflarea tipului obiectului (clasa din care face parte) la executie atunci cand ai doar un pointer sau o referinta catre acel tip.

Exemplu: int myint=50; cout<<typeid(myint).name()

->"dynamic_cast<>" se poate aplica pe pointeri sau referinta, realizeaza convertirea unui obiect de un anumit tip la alt tip si are urmatoarea implementare :

Sintaxa: dynamic_cast <tipul la care vrem sa convertim> (ceea ce vrem sa convertim).

16. Descrieti pe scurt diferenta dintre o clasa si un obiect.

Clasa reprezinta un tip abstract de date , format din campuri de date (structuri de date) si metode (functii/algoritmi), in timp ce, un obiect este o instanta a unei clase.

Atunci cand o clasa este create nus e alca memorie, pe cand atunci cand un obiect este create se aloca memorie

17. Descrieti pe scurt functiile sablon si dati exemplu de trei situatii in care un apel de functie nu genereaza o versiune a functiei dintr-un sablon disponibil pentru functia respectiva .

Functiile sablon sunt o metoda de implementare a polimorfismului de compilare. Cu un sablon este posibil sa cream functii generice si clase generice. O functie generica defineste un set general de operatii care vor fi aplicate unor tipuri de date variate. Unei astfel de functii tipul de date asupra caruia va opera ii este transmis ca parametru. O astfel de functie este creata cu ajutorul cuvantului cheie "template" iar forma generala a unei defenitii de functie de tip template este :

template <class Tip> tip_returnat nume_functie (lista parametri) {//corpul functiei}

Trei situatii in care un apel de functie nu genereaza o versiune a functiei dintr-un sablon:

*Cand functia sablon are in lista de parametrii doi parametrii de tipul sablonului , iar noi apelam functia pentru doua tipuri de date diferite.

*Cand functia sablon are in lista de parametrii un parametru de tipul sablonului si cand mai avem o functie definita cu acelasi nume si un tip specificat , atunci se va executa functia cu tipul deja specificat.

*Cand functia sablon are in lista de parametrii doi parametrii de tipul sablonului, iar noi apelam functia pentru un tip si un pointer la acel tip.

18. Descrieti pe scurt diferenta dintre polimorfismul de compilare si cel de executie.

Polimorfismul este o caracteristica a programarii orientate pe obiecte care se refera la comportamentul metodelor (posibilitatea ca o functie sa se comporte diferit in contexte diferite). Exista doua tipuri de polimorfism : **de compilare** (care se decide la compilare) si **de executie** (care se decide la excutie in functie de anumite informatii disponibile la momentul executiei). Polimorfismul de compilare cuprinde : supraincarcarea functiilor/operatorilor , sabloane si parametrii cu valori implicite , in timp ce polimorfismul de executie cuprinde : insantiere dinamica, mostenire si metode virtuale.

19. Descrieti pe scurt diferenta dintre functiile care returneaza valoarea si cele care returneaza referinta

Intoarcerea rezultatului prin valoare reprezinta copierea valorii furnizate de functia apelata pentru o instructiune return intr-o variabila temporara, de tipul functiei , creata in functia apelanta, in timp ce intoarcerea rezultatului prin referinta reprezinta crearea unei referinte temporare de tipul functiei in functia apelanta , catre variabila intoarsa ca rezultat de functia apelata.

20. Descrieti pe scurt cum se pot defini functii de conversie intre tipuri (clase).

Avem situatia urmatoare : B - tip de date deja existent (clasa sau tip predefinit) si definim o clasa noua A .

*B->A : se defineste un constructor in clasa A cu un singur parametru de tip B : A(B).

*A->B : supraincarcam operatorul "cast" al tipului B in clasa A : (B)A. Sintaxa pentru acest caz este : class A{ public: operator B(); }

Observatie : Constructorii cu un parametru sunt tratati ca metode de conversie implicita, chiar daca nu pentru asta au fost creati. Solutie : se pune in fata constructorului cuvantul explicit.

21. Spuneti care este diferenta dintre clasa generica (template) si clasa abstracta si in ce situatii se foloseste fiecare dintre ele.

O clasa generica este o clasa care defineste toti algoritmii definiti de ea , dar tipul de date care este manevrat efectiv va fi specificat ca un parametru la crearea obiectului acelei clase. Forma generala a declararii unei clase generice este **: template<class Tip>class nume-clasa{ ....**} . Odata ce am construit o clasa generica putem crea un anumit exemplar al acesteia folosind forma generala : nume-clasa<tip>ob;. Pe de alta parte **clasa abstracta** este o clasa in care avem cel putin o metoda virtuala pura. O alta diferenta este ca **clasa generica** reprezinta o metoda a polimorfismului de compilare , in timp ce **clasa abstracta** reprezinta o metoda a polimorfismului de executie **Sintaxa: class A{ virtual f()=0}.** O clasa abstracta constituie un tip incomplet care este folosit ca fundament pentru clasele derivate. Clasele generice sunt folositoare cand o clasa contine caracteristici generale.

22. Descrieti pe scurt diferenta dintre transferul prin valoare si transferul prin referinta al parametrilor in cazul apelului unei functii.

Transferul prin valoare reprezinta copierea valorii transmise ca parametru actual in parametru formal , care este creat pe stiva la lansarea in executie a functiei , ca o variabila

locala (automatica), in timp ce transferul prin referinta reprezinta crearea unei referinte locale catre variabila transmisa ca parametru actual la lansarea in executie a functiei.

23. Spuneti care este diferenta dintre incluziunea de clase si mostenirea de clase si cand se foloseste fiecare metoda.

Mostenirea este mecanismul prin care o clasa noua este creata prin preluarea tuturor elementelor unei clase existente si adaugarea unor elemente noi specifice. Clasa de la care se pleaca se numeste clasa de baza, iar clasa la care se ajunge se numeste clasa derivata. La derivare putem asocia clasei de baza un atribut de acces : public (elementele din clasa de baza isi pastreaza atributele de acces in clasa derivata), private (elementele din clasa de baza devin private in clasa derivata) si protected (elementele publice din clasa de baza devin protected in clasa derivata). **Incluziunea de clase** se manifesta atunci cand intr-o clasa avem campuri de date de tipul altor clase. Mostenirea se foloseste cand vrem ca o clasa sa incorporeze in declararea sa alta clasa, astfel v-om construi o ierarhie de clase, in timp ce incluziunea este folosita cand vrem ca doar anumite parti a unei clase deja existente sa fie incluse in declararea unei clase noi. Observatie : Incluziunea -> verbul “a avea” ; Mostenire -> verbul “a fi”.

*O clasă D **MOȘTENEȘTE** o clasă B, dacă se poate spune **”D ESTE (IS A) un obiect de tip B**”.*

Persoană ===> Student / Profesor / ... (Studentul ESTE o persoană).

***Sintaxa**:* class A{...};

class B: public A {...}

*O clasă D **APARȚINE** (**compunere**) de o clasă B, dacă se poate spune **”B ARE (HAS A) un obiect de tip D”.***

Facultate ===> Student / Profesor / ... (Facultatea ARE studenți).

***Sintaxa**:* class A{...};

class B { A o1; int x; }

24. Descrieti pe scurt constructorul de copiere si situatiile in care acesta este folosit.

Constructorul de copier este o functie membra care ajuta la initializarea unui obiect utilizand alt obiect din aceasi clasa

Constructorul de copiere este un constructor cu un singur parametru de tipul clasei. El are o forma implicita care copiaza octet cu octet .

Sintaxa: class_name(const classname &ob)

Utilizarea constructorului de copiere :

* Pentru initializarea obiectelor din alte obiecte de acelasi tip:

```
Class A{} => A a, b(a);
```

* La transferul parametrilor unei functii prin valoare

```
Class A{
        F(A ob)}
```

* La intoarcerea rezultatului unei functii prin valoare

```
Class A{
A F(){A ob;
    Return ob;}}
```

25. Spuneti daca o variabila constanta poate fi transmisa ca parametru al unei functii si daca da in ce situatii. Justificati.

O variabila constanta poate fi transmisa ca parametru al unei functii doar cand tipul parametrului functiei nu este de tip referinta neconstanta si cand nu este de tip neconstant , deoarece in momentul in care unul dintre parametrii functiei este de tipul mentionat atunci am avea in primul caz : convertirea unei variabile constante la o referinta neconstanta (eroare) , iar in al doilea caz : se apeleaza constructorul de copiere care este de tipul A(A &a) , deci din nou va fi aceeasi eroare.

26. Spuneti ce reprezinta o functie prietena a unei clase.

Este posibil sa permitem unei functii care nu este membru sa aiba acces la membri paritculari ai unei clase folosint un friend (o functie prietena). O functie friend are acces la membrii private si protected ai clasei careia ii este prietena . Pentru a declara o functie friend includem prototipul ei in acea clasa precedat de cuvantul cheie friend .

Exemplu : supraincarcarea operatorului „>>” pentru citire, respectiv „<< „ pentru afisare.

friend ostream &operator<<(ostream&os, const clasa& p);

27. Descrieti pe scurt ce reguli verifica supraincarcarea operatorilor.

Regulile care verifica supraincarcarea operatorilor sunt urmatoarele :

* Nu putem defini operatori noi. Doar supraincarcam operatori deja existenti.

* Nu putem modifica numarul operanzilor unui operator existent si nici sintaxa lor.

* Nu putem modifica prioritatea si nici asociativitatea operatorilor existenti.

* Exista operatori care nu se pot supraincarca (exemplu: “.”, “::”, “?:” ,”.*”.)

* Operatorii pot fi supraincarcati ca metode sau ca functii independente. Nu toti operatorii care pot fi supraincarcati ca metode pot fi supraincarcati ca functii independente ! (exemplu: “= “, “( )”, “[ ] “,”->”)

28. Descrieti trei metode de proiectare diferite prin care elementele unei clase se pot regăsi în dublu exemplar, sub diverse forme, în definitia altei clase.

Cele trei metode de proiectare sunt urmatoarele:

* mostenire multipla, nevirtuala (cand mostenesti o clasa si inca una care e derivata din prima clasa mostenita)

* prin compunere (poti sa ai doua obiecte din doua clase diferite care mostenesc aceeasi clasa de baza

* mostenire si compunere (a aceleiasi clase de baza)

29. Enumerati trei metode de implementare a polimorfismului de compilare.

Polimorfismul de compilare se poate realiza prin :

* Supraincarcare functii si operatori

* Sabloane(template-uri)

* Parametrii cu valori implicite

30. Descrieti pe scurt comportamentul operatorului dynamic_cast.

Operatorul "dynamic_cast<>" se poate aplica pe pointeri sau referinta, realizeaza convertirea unui obiect de un anumit tip la alt tip si are urmatoarea implementare : dynamic_cast <tipul la care vrem sa convertim> (ceea ce vrem sa convertim).

31. Tipuri de mostenire

a. Mostenire private

Această moștenire presupune că totul din clasa de BAZĂ devine private în clasa DERIVATĂ. Singurele excepții sunt datele și metodele private care devin inaccesibile.
Acest tip de moștenire este cel implicit când nu specificăm niciun modificator după simbolul :

b. Moștenire protected

Tot ce nu e private în clasa de bază devine protected în clasa derivată.

c. Mostenire publica

Aceasta este cea mai des folosită moștenire. Totul rămâne la fel ca în clasa de bază. Datele și metodele private tot inaccesibile rămân.

Accesul asupra membrilor mosteniti:

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

32. Pointeri constanti vs pointeri catre constante

POINTER CATRE OBIECT CONSTANT: aloarea obiectului nu poate fi modificata, dar putem modifca obiectul catre care arata pointerul.

Sintaxa: const tip de date * p

Exemplu:

```
const int* p; // pointer catre un intreg constant
                // pointerul se poate muta, dar valoarea NU se poate schimba
int x = 3;
p = &x; // ok
*p = 5; // not ok
```

POINTER CONSTANT CATRE OBIECT-pointerul nu poate fi modifica, dar valoarea catre care pointeaza da

Sintaxa: tip de date const*p

Exemplu:

```
int* const p; // pointer constant catre un intreg
              // pointerul NU se poate muta, dar valoarea se poate schimba
int x = 3;
p = &x; // not ok
*p = 5; // ok
```

33. Prin ce se caracterizeaza o variabila statica a unei clase:

Proprietetati:

- variabilele statice sunt precedate de cuvantul- cheie static.

- o singura copie din acea variabila va exista pentru toata clasa

- o variabila statica declarata in clasa nu este definita(nu are zona de memorie alocata). Pentru a putea fi definite, sunt redeclarate in exteriorul clasei respective, folosind operatorul de rezolutie.

- Dat fiind ca variabilele statice sunt instantiate indiferent daca sunt declarate sau nu obiecte, putem accesa variabilele statice fara a ne folosi de vreun obiect, prin intermediul operatorului de rezolutie.

```
Exemplu: Class A {
                static int x;
                 }
                int A::x=5;
```

Exemplu de utilizare: pentru a retine numar de instante ale unui obiect.(si a limita instatierea mai multor obiecte ex Singletone)

34. Spuneti care dintre urmatoarele reprezinta mecanisme prin care se obtine polimorfismul de functii:

functiile friend , functiile inline , constructorii, functiile virtuale, destructorii.

Raspuns: functii inline,constructori, functii virtuale. Obs: nu poate exista decat un singur destructor pentru fiecare clasa.

35. Descrieţi pe scurt diferenţa dintre funcţiile care returnează valoare şi cele carereturnează referinţă.

INTOARCERE PRIN VALOARE-copierea valorii furnizate de functie intr-o variabila temporara,de tipul funtiei.

INTOARCERE PRIN REFERINTA - crearea unei referinte temporare de tipul functiei in functia apelanta , catre variabila intoarsa ca rezultat de functia apelata.

36. Descrieţi pe scurt cum se pot defini funcţii de conversie între tipuri (clase).

Conversiile de tip se pot realiza prin:

- supraincarcarea operatorului unar „cast"(se poate face conversia dintrun tip clasa intr-un tip fundamental sau intr-un alt tip clasa)

sintaxa: operator TipData();

- prin intermediul constructorilor (consta in definirea unui constructor care primeste ca parametru tipul la care se face conversia). Constructorul intoarce intotdeauna un rezultat de tipul clasei de care apartine, prin urmare prin intermediul cosntructorilor se poate face conversia doar de la un tip fundamental sau un tip clasa la un alt tip clasa.

OBS: in cazul conversiei de la un tip clasa la alt tip clasa, constructorul trebuie sa aiba acces la campurile private ale tipului clasa in care de doreste a se face conversia si prin urmare trebuie declarat ca functie friend.

Exemplu : Fie clasa B (tip de date predefinit sau clasa) si se doreste a se face conversia la tipul A

1) **supraincarcam operatorul de cast al** tipului in clasa B. class B { public : operator A(); }

2) **supraincarcare constructorului de copiere** cu un singur parametru de tip B: A(B);

OBS. Constructorii cu un parametru sunt tratati ca metode de conversie implicita, chiar daca nu pentru asta au fost creati. Solutie: se pune in fata constructorului cuvantul explicit.

37. Descrieti in ce consta polimorfismul de executie folosind metode virtuale:

Mostenirea virtuala este o metoda de implementare a polimorfismului de executie.

-mostenirea virtuala este necesara cand avem 2 clase derivate din clasa de baza si o a treia care mosteneste cele 2 clase(apare problema diamantului – anumite elemente sunt mostenite de doua ori, ceea ce conduce la ambiguitate).

38. Descrieti cele doua feluri de folosire a cuvantului „virtual" la mostenire si in ce cazuri se folosesc:

A) Mostenirea virtuala se realizeaza folosind cuvantul cheie virtual.

Exemplu: Class D: virtual public A

Utilitate: pentru a rezolva ambiguitatea diamantului(doua clase deriva din clasa de baza si avem o a treia clasa care deriva din cele doua clase).

B) Cand definim functiile utilizand cuvantul cheie virtual, pentru a le putea redefini in clasa derivata

Exemplu:

```
class A { int x;
        public: virtual int sum(){ //definitie functie };
        // functie initiala }
```

```
class B:public A { int y;
        public: virtual int sum()
        {//redefinire functie }; }
```

39. Cum se poate supraincarca operatorul [] si care este utilizarea uzuala a acestuia.

Operatorul [], alaturi de operatorul () si operatorul „=" NU se pot supraincarca folosind functie friend. Acestia trebuie supraincarcati ca functii membre
-este considerat operator binar;
-operatorul [] poate fi folosit si la stanga unei atribuiri(obiectul intors este referinta)
Exemplu: int v[100]
int & operator[](int i){return vector[i]}

40. Ce este lista de initializare a unui constructor si care este utilitatea ei.

-apare in implementarea constructorului, intre antentul si corpul acestuia.
- lista contien operatorul „ : " urmat de numele fiecarui membru si valoarea acestuia, in ordinea in care memebrii apar in definita clasei
Utilizare: -initializarea variabilelor statice
-initializarea referintelor
-initializarea campurilor pentru care nu exista un constructor implicit.
-initializare membrilor clasei de baza
-din motive de eficienta.
Exemplu: class A { int x, y;
public: A(int x, int y)::A(7,9){}; }

41. Polimorfism la compilare:

Polimorfismul este o caracteristica a programarii orientate pe obiecte care se refera la comportamentul metodelor (posibilitatea ca o functie sa se comporte diferit in contexte diferite). Polimorfism **de compilare** (care se decide la compilare) cuprinde : supraincarcarea functiilor/operatorilor , sabloane si parametrii cu valori implicite .

42. Polimorfism la executie

Polimorfismul este o caracteristica a programarii orientate pe obiecte care se refera la comportamentul metodelor (posibilitatea ca o functie sa se comporte diferit in contexte diferite). Polimorfismul **de executie** (care se decide la excutie in functie de anumite informatii disponibile la momentul executiei) cuprinde : insantiere dinamica, mostenire si metode virtuale.

43. Descrieti pe scurt comportamentul functiilor virtuale in constructori si in destructori.

-daca vrem sa eliminam portiuni alocate dinamic si pentru clasa derivata, dar facem upcasting

trebuie sa folosim destructori virtuali

```
class AbstractBase {
public:
virtual ~AbstractBase() = 0;
};
class Derived : public AbstractBase {};
int main() { Derived d; }
```