1) operatori binari => ca metoda au un sg param
2) supraincarcare operatori nume_tip
3) return genereaza un obiect
4) i-- => verificare dupa decrementare dupa instr
5) prop de friend nu e tranzitiva
6) pointeri de clase abstracte merge
7) pt mostenire vituala toate clasele trb sa mosteneasca virtual si sa fie mostenite virtual
8) malloc doar aloca memorie, nu apeleaza so constructorul clasei
9) conversia non static la static merge
10) conversia const la non const merge
11) pt fucntile cee mostenesc virtual se ia o sg data functia mostenita + cate o data pt fiecare functie care mosteneste normal
12) la distrugerea unui pointer se apeleaza destrucotrul tipului pointerului
13) se realizeaza converisa cu operatorul int si in cazul ob = int
14) daca apelez o fucntie cu param de tip int cu un elem de tip float se realizeaza conversia
15) var statice nu se initiaza in lista de initializare
16) nu se pot declara operatori cls inaintea declararii clasei respective
17) un obiect const nu poate fi trimis prin referinta non const
18) v=new int[x] daca exista deja v se distruge si apoi este reinitializat
19) nu se poate realiza conversia int& la const int
20) constr sau destruct nu pot returna val
21) obiectele temporare sunt const deci nu pot fi pasate prin referinte non const
22) la return x++ se realizeaza incrementarea lui x
23) obiectele statice pot fi date ca val pt param default
24) o functie care arunca erori nu trebuie sa aiba neaparat un block try catch dedicat (poate sa se afla intr-un bloc mai mare)
25) o functie const face toate intsntele obiectului const
26) dc o clasa mosteneste private o alta clasa, nu se poate face downcasting la clasa de baza
27) chiar daca mostenirea este private, se poat accesa prin intermediul obiectului derivta merodele virtual din baza chiar daca nu sunt suprascrise
28) daca un obiect mosteneste de 2 ori o cls de baza nu se poate face downcasting la un pointer de tip baza (nu se stie pe care ruta de mostenire sa mearga)
29) se poate apela o metoda dintr-o clasa cu un obiect null
30) doar operatorul () poate avea param default
31) daca o clasa mosteneste privat alta clasa, nu se mai poate face downcasting la clasa de baza

Constructori:
1) obiectul trimis ca param se distruge la sf functiei
2) obiectele temp definite in int functiei nu se distrug la return (constructor de mutare) => li se prelungeste durata de viata

```
#include <iostream>
using namespace std;
```

```
struct C {
    C () {cout << "C" << endl;}
    C (const C &c) {cout << "C-cpy" << endl;}
    ~C () {cout << "~C" << endl;}
};

void f (C c) {}

C g () {
//    C c;
    return C();
}

C h (C c) {
    return c;
}

int main () {
   C c;
//    f(c);
//   C c = g();
    h(c);
    return 0;
}
```

POLIMORFISM LA EXECUTIE

Polimorfismul la executie, dupa cum ii spune si numele, se decide la executia programului. Acesta cuprinde alocarile dinamice de memorie, metodele si mostenirea virtuala.

Alocarea dinamica permite realizarea downcastingului, obiectul derivat putand fi pasat print-un pointer de tip baza, astfel evitandu-se verificarile de tip.

Metodele virtuale se definesc astfel: virtual tip_retur nume_metoda(parametrii){ instructiuni }. Acestea sunt folosite la downcasting, permitand fiecarui obiect sa acceseze metoda definita in clasa din care acesta face parte, chiar daca obiectul este transmis printr-un pointer de tip baza. Acest lucru este posibil datorita tabelului numit vtable care pastreaza pentru fiecare clasa un pointer catre metoda corespunzatoare acesteia.

Mostenirea virtuala se foloseste in cazul in care o clasa mosteneste mai multe clase derivate din acceasi clasa de baza (ex: in cazul mostenirii de tip diamant). In astfel de cazuri, se creaza ambiguitate cand vrem sa ne referim la o metoda sau o instanta din baza, acestea fiind mostenite de mai multe ori. Prin mostenirea virtuala se evita aceasta ambiguitate, aceasta asigurandu-se ca baza este mostenita o singura data in noua derivata. Aceasta se realizeaza prin adaugarea cuvantului cheie virtual inainatea modificatorului de acces la mostenire: class nume_clasa1: virtual modificator_acces nume_clasa2 {};