```
#include <iostream>
#include <conio>

class TabladeJoc
{
public:
  TabladeJoc() { cout << "TabladeJoc()\n"; }

  TabladeJoc(const TabladeJoc&) {
    cout << "TabladeJoc(const TabladeJoc&)\n";
  }

  TabladeJoc& operator=(const TabladeJoc&) {
    cout << "TabladeJoc::operator=()\n";
    return *this;
  }

  ~TabladeJoc() { cout << "~TabladeJoc()\n"; }
};

class Joc {
  TabladeJoc tj; // Compunere

public:
  // implica automat apel de constr. implicit TabladeJoc
  Joc() { cout << "Joc()\n"; }

  // Trebuie apelat explicit constructorul de copiere al clasei
  // obiectului inclus (TabladeJoc), altfel se apeleaza
  //constructorul implicit pt. componenta respectiva


Joc(const Joc& j) : tj(j.tj) {
    cout << "Joc(const Joc&)\n";
  }

Joc(int)
  { cout << "Joc(int)\n"; }

  Joc& operator=(const Joc& j) {
    // Trebuie apelat explicit operatorul de atribuire
    // al clasei TabladeJoc , altfel nu se fac atribuiri
    // pentru componenta respectiva

    tj = j.tj;
    cout << "Joc::operator=()\n";
    return *this;
  }

  ~Joc() { cout << "~Joc()\n"; }
};

class Sah : public Joc {};

class Dame : public Joc {
public:
  // implica automat apel de constr. implicit Joc
  Dame() { cout << "Dame()\n"; }

  // Trebuie apelat explicit constructorul de copiere al clasei
  // de baza, altfel se apeleaza constructorul implicit pt.
  // membri mosteniti din clasa de baza

Dame(const Dame& c) : Joc(c)
{
    cout << "Dame(const Dame& c)\n";
  }

Dame& operator=(const Dame& c) {
 // Trebuie apelata explicit versiunea din clasa de baza
 // pentru operator=() altfel nu se realizeaza atribuirea
 // intre componentele mostenite din clasa de baza

    Joc::operator=(c); //  (Joc&)(*this)=c;
    cout << "Dame::operator=()\n";
    return *this;
  }
};


int main()
{
  Sah s1;  // Constructor implicit
  Sah s2(s1); // Constructor de copiere
//! Sah s3(1); // Eroare nu exista constructor cu param
  s1 = s2; // Operator = sintetizat

  Dame d1, d2(d1);
  d1 = d2;
  getch();
  return 0;
}
```

```
TabladeJoc()
Joc()
TabladeJoc(const TabladeJoc&)
Joc(const Joc&)
TabladeJoc::operator=()
Joc::operator=()
TabladeJoc()
Joc()
Dame()
TabladeJoc(const TabladeJoc&)
Joc(const Joc&)
Dame(const Dame& c)
TabladeJoc::operator=()
Joc::operator=()
Dame::operator=()
~Dame
~Joc()
~TabladeJoc()
~Dame
~Joc()
~TabladeJoc()
~Joc()
~TabladeJoc()
~Joc()
~TabladeJoc()
```

Ex. 1 Fara metode virtuale

```
#include <iostream.h>
#include <conio.h>

class Paralelogram
{
public:
  Paralelogram(){ cout<<"Constr. Paral\n";}
  void afis() const {
    cout << "Paralelogram::afis" << endl;
  }
  ~Paralelogram(){cout<<"Destr. Paral\n";}
};

class Dreptunghi : public Paralelogram {
public:
  Dreptunghi(){ cout<<"Constr. Dreptunghi\n";}
  void afis() const {
    cout << "Dreptunghi::afis" << endl;
  }
  ~Dreptunghi(){cout<<"Destr. Drept\n";}
};

class Patrat : public Dreptunghi {
public:
  Patrat(){cout<<"Constr. Patrat\n";}
  void afis() const {
    cout << "Patrat::afis" << endl;
  }
  ~Patrat(){cout<<"Destr. Patrat\n";}
};

void display(Paralelogram& p) {
  p.afis();
}

int main(void) {
  Paralelogram par;  Dreptunghi dre;  Patrat pat;
  Paralelogram tab[]={par, dre, pat};// se apel. Constr. Cop Paralelog.
  Paralelogram *tabp[]={&par, &dre, &pat};
  for(int i=0;i<3;i++)
     tabp[i]->afis();
  cout<<"Display ...\n";
  display(par);  display(dre);  display(pat);
  getch();  return 0;
}
```

```
Constr. Paral
Constr. Paral          Constr. Dreptunghi
Constr. Paral          Constr. Dreptunghi    Constr. Patrat
Paralelogram::afis  Paralelogram::afis  Paralelogram::afis
Display ...
Paralelogram::afis  Paralelogram::afis Paralelogram::afis
Destr. Paral
Destr. Paral
Destr. Paral
// merge pt. ca sunt obiecte!
Destr. Patrat          Destr. Drept          Destr. Paral
Destr. Drept           Destr. Paral
Destr. Paral
```

Ex. 2 Cu metode virtuale

```
#include <iostream.h>
#include <conio.h>

class Paralelogram
{
public:
  Paralelogram(){ cout<<"Constr. Paral\n";}
  virtual void afis() const {
    cout << "Paralelogram::afis" << endl;
  }
  virtual ~Paralelogram(){cout<<"Destr. Paral\n";}
};

class Dreptunghi : public Paralelogram {
public:
  Dreptunghi(){ cout<<"Constr. Dreptunghi\n";}
  virtual void afis() const {
    cout << "Dreptunghi::afis" << endl;
  }
  virtual ~Dreptunghi(){cout<<"Destr. Drept\n";}
};
class Patrat : public Dreptunghi {
public:
  Patrat(){cout<<"Constr. Patrat\n";}
  virtual void afis() const {
    cout << "Patrat::afis" << endl;
  }
  virtual ~Patrat();//{cout<<"Destr. Patrat\n";}
};

Patrat::~Patrat(){    cout<<"Destr. Patrat\n";}

void display(Paralelogram& p) {
  p.afis();
}
int main(void){
  Paralelogram par;  Dreptunghi dre;  Patrat pat;
  Paralelogram tab[]={par, dre, pat};
  Paralelogram *tabp[]={new Paralelogram, new Dreptunghi, new
Patrat};
  for(int i=0;i<3;i++)
     tabp[i]->afis();
  cout<<"Display ...\n";
  display(par);          display(dre);          display(pat);
  for(i=0;i<3;i++)
     delete tabp[i];
  // Warning delete will invoke a non-virtual destructor
  // daca destructorii nu sunt virtuali
  getch();  return 0;
}
```

cu destructori nevirtuali

```
Constr. Paral
Constr. Paral          Constr. Dreptunghi
Constr. Paral          Constr. Dreptunghi          Constr. Patrat
Constr. Paral
Constr. Paral          Constr. Dreptunghi
Constr. Paral          Constr. Dreptunghi          Constr. Patrat
Paralelogram::afis  Dreptunghi::afis            Patrat::afis
Display ...
Paralelogram::afis  Dreptunghi::afis            Patrat::afis
// TABP
Destr. Paral           Destr. Paral          Destr. Paral
// TAB
Destr. Paral           Destr. Paral          Destr. Paral
Destr. Patrat          Destr. Drept          Destr. Paral          // pat
Destr. Drept           Destr. Paral                                // dre
Destr. Paral                                                       // par
```

cu destructori virtuali

```
Constr. Paral
Constr. Paral          Constr. Dreptunghi
Constr. Paral          Constr. Dreptunghi          Constr. Patrat
Constr. Paral
Constr. Paral          Constr. Dreptunghi
Constr. Paral          Constr. Dreptunghi          Constr. Patrat
Paralelogram::afis  Dreptunghi::afis            Patrat::afis
Display ...
Paralelogram::afis  Dreptunghi::afis            Patrat::afis
// TABP
Destr. Paral
Destr. Drept           Destr. Paral
Destr. Patrat          Destr. Drept          Destr. Paral
// TAB
Destr. Paral           Destr. Paral          Destr. Paral
Destr. Patrat          Destr. Drept          Destr. Paral          // pat
Destr. Drept           Destr. Paral                                // dre
Destr. Paral                                                       // par
```