UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Programarea Orientată pe Obiecte (POO)

Prelegere

## *Supraîncărcarea operatorilor*

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# SUMAR

1. Privire de ansamblu
2. Reguli privind supraîncărcarea operatorilor
3. Supraîncărcarea operatorilor binari
4. Supraîncărcarea operatorilor unari
5. Conversii

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Privire de ansamblu

§ Un tip de date (obiect) este caracterizat prin:
- Domeniul de valori admisibile;
- Set de operaţii caracteristice tipului.

Spre **exemplu**, tipul de date **număr întreg** (în C++ int)
- Domeniul de valori: -32768 … +32767
- operaţii caracteristice: adunare, scădere, înmulţire ş.a.

§ De regulă, operaţiile caracteristice sunt implementate la nivel de operatori, astfel pentru variabile de tipul de dată întreg:

```
int a,b,c;
```

§ Operaţiile caracteristice sunt implementate la nivel de operatori **c+a*b**

§ Care este mult mai simplă decât prin funcţii: c.adunare(a.produs(b))

§ Un **operator** poate fi privit ca o funcţie, în care termenii sunt argumente.

În lipsa operatorului + expresia **a+b** s-ar calcula apelând funcţia **aduna(a,b)**.

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Să analizăm următoare secvență de cod

```
#include "complex.h"
main(){
 complex x(3,4.4),y(2.5,4.87),z;
 double a=4.25;
 cout<<"Valorile introduse"<<endl;
 cout<<"x="<<x<<endl;
 cout<<"y="<<y<<endl;
 cout<<"a="<<a<<endl;
 cout<<"Operatii cu numere complexe"<<endl;
 z=x+y; cout<<x<<"+"<<y<<"="<<z<<endl;
 z=x+a; cout<<x<<"+"<<a<<"="<<z<<endl;
 z=a+y; cout<<a<<"+"<<y<<"="<<z<<endl;
```

```
z=x/y;
cout<<x<<"/"<<y<<"="<<z<<endl;
z=x-y;
cout<<x<<"-"<<y<<"="<<z<<endl;
z=x*y;
cout<<x<<"*"<<y<<"="<<z<<endl;
cout<<" (x+y)-(a+y)+++x---y";
z= (x+y)-(a+y)+++x---y;
cout<<endl<<"="<<z<<endl;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Noţiuni fundamentale (I)

- Operaţia de adunare + funcţionează pentru variabile de tip **int**, **float**, **double** etc.
  - Operatorul + a fost supraîncărcat chiar în limbajul de programare C++

- Pentru ca un operator să poată lucra asupra obiectelor unei noi clase, el trebuie să fie supraîncărcat pentru acea clasă

- Operatorii se supraîncarcă scriind o definiţie de funcţie obişnuită, având un nume special - cuvântul cheie **operator** urmat de simbolul operatorului care urmează să fie supraîncărcat

- ***Exemplu***
  - **operator+** este numele funcţiei prin care se supraîncarcă operatorul +

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Noţiuni fundamentale (II)

§ **Scopul** supraîncărcării operatorilor
- Scrierea expresiilor concise pentru obiecte care fac parte din clase definite de programatori în acelaşi fel în care se scriu pentru tipurile predefinite

§ **Greşeli** la supraîncărcarea operatorilor
- Operaţia implementată de operatorul supraîncărcat nu corespunde din punct de vedere semantic operatorului

§ ***Exemple***
- Supraîncărcarea operatorului + printr-o operaţie similară scăderii
- Supraîncărcarea operatorului / ca să implementeze o înmulţire

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Reguli pentru supraîncărcarea operatorilor

- Setul de operatori ai limbajul C++ nu poate fi extins prin asocierea de semnificaţii noi unor caractere, care nu sunt operatori (de exemplu nu putem defini operatorul „**”).

- Prin supraîncărcarea unui operator nu i se poate modifica paritatea (astfel operatorul “!” este unar şi poate fi supraîncărcat numai ca operator unar).

- Nu se poate modifica precedenţa şi asociativitatea operatorilor.

- Operatorii supraîncărcaţi într-o clasă sunt moşteniţi în clasele derivate excepţie face operatorul de atribuire „=”.

- Unui operator i se poate atribui orice semnificaţie, însă se recomandă ca aceasta să fie cât mai apropiată de semnificaţia naturală.

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor

- Limbajul C++ permite programatorului să definească diverse operații cu obiecte ale claselor, folosind simbolurile operatorilor standard. Un tip clasă se poate defini împreună cu un set de operatori asociați, obținuți prin supraîncărcarea operatorilor existenți.
- În acest fel, se efectuează operații specifice cu noul tip la fel de simplu ca în cazul tipurilor standard.

- Pot fi supraîncărcaţi următorii operatori: + - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &= |= >>= <<= == != <= >= && || ++ -- ->* , -> << >> [] () **new** **new**[] **delete** **delete**[]

- Nu pot fi supraîncărcaţi operatorii: :: . .* ?: sizeof.

- Procesul de supraîncărcare a unui operator poate fi realizat prin :
  - intermediul unei funcţii membru;
  - intermediul unei funcţii prietene.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor prin funcţii membru

§ Sintaxa:

**class** nume_clasa{

Tip_returnat **operator** S(lista_de_parametri);

};

Tip_returnat nume_clasa::**operator** S(lista_de_parametri){

// descrierea procesului de supraîncărcare

} unde S este operatorul supraîncărcat, tip – tipul rezultatului operaţiei S, iar lista_de_parametri reprezintă

§ !!! **Observaţie** primul argument va fi întotdeauna un operand de tip clasă.

```
class Complex{                  // operația a+b    a este primul operand, b operandul doi
 double x,y;
 public:
 complex operator+(complex arg_doi);
};
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor prin funcţii friend

§ Sintaxa:

**class** nume_clasa{

**friend** Tip_returnat **operator** S(lista_de_parametri);

};

Tip_returnat **operator** S(lista_de_parametri){

// descrierea procesului de supraîncărcare

} unde S este operatorul supraîncărcat, tip – tipul rezultatului operaţiei S, iar lista_de_parametri reprezintă

```
class Complex{          // operația a+b   a este primul operand, b operandul doi
 double x,y;
 public:
 friend complex operator+(complex arg_1, complex arg_doi);
};
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# SUMAR

**Supraîncărcarea operatorilor binari:**

- Supraîncărcarea operatorilor aritmetici
- Supraîncărcarea operatorilor de comparaţie
- Supraîncărcarea operatorilor compuși
- Supraîncărcarea operatorilor << şi >>
- Supraîncărcarea operatorului de atribuire
- Supraîncărcarea operatorului de indexare („[ ]”)
- Supraîncărcarea operatorului apel de funcţie(„( )”)

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Crearea tipului de date complex

- Aşa cum adunarea a două numere complexe este comutativă, vom examina cazurile:
  - complex + complex
  - double + complex
  - complex + double

- În primul şi al treilea caz supraîncărcarea poate fi efectuată prin intermediul funcţiilor membru, însă în cazul doi supraîncărcarea este posibilă numai prin intermediul unei funcţiei prieten, deoarece primul parametru al unei funcţii membru operator este implicit de tipul clasei şi nu poate fi de alt tip.

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor aritmetici

```
class complex{
 double x,y;//complex=x+yi
 public:
 complex(){x=y=0;}    complex(double a, double b){x=a;y=b};
 void afis() {cout<<x<<"+"<<y<<"i\n";}
 complex operator +(complex);
 friend complex operator +(double,complex);
 complex operator+(double);
};
complex complex::operator+(complex b){
 complex c; c.x=x+b.x;  c.y=y+b.y;       return c;}
complex complex::operator+(double b){
 complex c; c.x=x+b; c.y=y;              return c;}
complex operator+(double a, complex b){
 complex c; c.x=a+b.x; c.y=b.y;          return c;}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor de comparaţie

Aşa cum limbajul de programare C++ nu are implementat tipul de date logic, se va considera valoarea 1 echivalenta valorii true, iar valoare 0 echivalenta valorii false. În acest context tipul rezultatului unei operaţii de comparare a două valori numerice poate fi int, avînd valorile posibile 1 sau 0.

```
class complex{
 double x,y;
 public:
 int operator== (complex);
};
int complex::operator==(complex ob){
 if(x==ob.x && y==ob.y) return 1;
 else return 0;
}
```

```
class complex{
 double x,y;
 public:
 friend int operator==(complex,complex);
};
int operator==(complex ob1,complex ob2){
 if(ob1.x==ob2.x && ob1.y==ob2.y)
 return 1;   else return 0;
}
```

```
int main(){  complex a, b;
 if(a==b) cout<<"Numere egale";
 else cout<<"Numere diferite";
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor compuși

```
class complex{
 double x,y;
 public:
 complex(){x=y=0;}
 complex(double a, double b){x=a;y=b;}
 void afis() {cout<<x<<"+"<<y<<"i\n";}
 complex operator += (complex &);
 complex operator += (double);
};
complex complex::operator+=(complex &a){ x+=a.x; y += a.y; return *this;}
complex  complex::operator+=(double d){
  x+= d; return *this;}
int main(){  complex a, b(2,3);
 a+=10; a.afis();      a+=b;  a.afis();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor << şi >>

- Operaţiile de intrare/ieşire în C++ se efectuează prin intermediul operatorilor de inserţie << şi respectiv operatorul de extragere >> din streamuri. Spre deosebire de alţi operatori, operatorii de intrare/ieşire pot fi supraîncărcaţi doar prin intermediul funcţiilor friend.
- Supraîncărcarea operatorilor de intrare/ieşire se poate face respectînd sintaxa:

```
class nume_clasa {
 friend ostream & operator<<(ostream& os, nume_clasa nume);
 friend istream & operator>>(istream& is, nume_clasa &nume);
};
ostream& operator<<(ostream& os, tip_clasa nume){
                    // corpul functiei
 return os; }
 istream& operator>>(istream& is, tip_clasa &nume){
                    // corpul functiei
 return is; }
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorilor intrare/ieşire

```
class complex {  double x, y;
 public:
 friend ostream& operator << (ostream& os, complex z);
 friend istream& operator >>(istream& is, complex& z);
};
ostream& operator<<(ostream& os, complex z){
 os <<z.x;   if(z.y==0) os<<endl;
 if(z.y<0) os<<"-"<<z.y<<"i"<<endl;  if(z.y>0) os<<"+"<<z.y<<"i"<<endl;
 return os;
}
istream& operator>>(istream& is, complex& z){
 cout<<"Dati partea reala "; is >> z.x;
 cout<<"Dati partea imaginara"; is>> z.y;
 return is;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului de atribuire

- Atunci când se utilizează operatorul "=", fără supraîncarcare, se realizează o "copiere bit cu bit". Aceasta nu înseamnă, că întotdeauna obținem rezultatele dorite. Acest lucru se întâmplă, deoarece cele două obiecte care formează atribuirea: ex. a = b, primesc aceeași adresa.
- In cazul în care obiectul b se distruge, atunci, obiectul a va avea o adresa spre "un obiect care nu mai există". Pentru a putea înlătura acest neajuns, trebuie sa supraîncărcăm operatorul "=".
- **Operatorul "=" este binar**
- Dacă acesta este supraîncărcat prin utilizarea unei funcții membru, atunci primul operand este obiectul curent, iar al doilea operand este parametrul funcției.
- **Exemplu** *Considerăm clasa sir, în care fiecare obiect reține adresa unui șir de caractere.*

*Astfel, data membru adr reține adresa unui pointer către șir,*
*iar șirul va fi alocat dinamic cu ajutorul*
*operatorului **new** într-o zonă de memorie disponibilă.*

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului de atribuire

```
class sir{   char *adr;
 public:
 sir(char s[]);      ~sir();
 void afis(){ cout<<adr<<endl; }
 void operator=(sir& sirul);
};
sir::sir(char s[ ]) {adr = new char[strlen(s) + 1]; strcpy(adr, s);}
sir::~sir() {delete[ ] adr;  adr = 0;}
void sir::operator=(sir& sirul) {cout<<"Operator = \n";   delete[ ] adr;
 adr = new char[strlen(sirul.adr)+1]; strcpy(adr,sirul.adr);
}
int main() {
 sir s("un sir"), t("alt sir"); s.afis();  t.afis();
 s = t; t.~sir();  s.afis(); cout<<"Sf. Program\n";
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului de indexare „[ ]”

- Operatorul **de indexare [ ]** este binar și are forma:

  **expresie_1[expresie_2]**

- Pentru supraîncărcarea acestui operator trebuie să folosim o funcție **operator[ ]**, care trebuie să fie membră a clasei, să fie nestatică și să aibă forma:

  **x[n] sau x.operator[ ] (n)**

- Al doilea operand, care are rolul indexului, poate fi la supraîncărcare de orice tip.
- **Exemplu** *Să elaborăm un program prin intermediul căruia vom crea un tablou de înregistrări, ce conțin informații de tip medical despre persoane.*

*Accesul la înregistrări urmează a realizat după nume, după greutate, după numărul de ordine.*
*În prima căutare - supraîncărcarea operatorului de indexare având ca parametru un șir de caractere care indică numele persoanei căutate.*
*În al doilea caz, vom face supraîncărcarea operatorului de indexare având ca parametru de tip **float**.*

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului de indexare „[ ]”

```
class analize;
class pers{
 char nume[35];   double greutate;   int varsta;
 friend class analize;
 public:
 void init(char *s, double gr, int v);
 void tipar();
};
void pers::tipar(){
cout<<"\nPersoana: "<<nume<<"\tGreutatea: "<<greutate<<"Varsta: "<<varsta;
}
void pers::init(char *s, double gr, int v){
 strcpy(nume, s);  greutate=gr;  varsta=v;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

```
class analize{  pers *sir;   int n;
 public:
 analize(){ n=5; sir=new pers[n];}
 analize(int nr){n=nr; sir=new pers[n];}
 pers *operator[] (char *);
 pers *operator[] (double);
 pers *operator[] (int);
 void introd();
};
void analize::introd(){
 for(int i=0;i<n;i++){
  cout<<endl;   cout<<"Persoana "<<i+1<<": ";
  cin>>sir[i].nume;   cout<<"Greutatea: ";
  cin>>sir[i].greutate;   cout<<"Varsta: ";
  cin>>sir[i].varsta;
 }
}
```

```
pers *analize::operator[ ](char *nume){
 for(int i=0;i<n;i++)
 if(strcmp(sir[i].nume, nume)==0) return &sir[i];
 return NULL;
}

pers *analize::operator[](double g){
 for(int i=0;i<n;i++)
 if(sir[i].greutate==g) return &sir[i];
 return NULL;
}

pers *analize::operator[] (int index){
 if(index<=n) return &sir[index-1];
 else return NULL;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

```
int main(){
 char c;  int nr;
 cout<<"Cate analize efectuati ? ";
 cin>>nr;  analize t(nr); t.introd();
 while(1){
  cout<<"\nOptiunea [g]reutate, [n]ume, [i]ndex, [e]xit ? ";   cin>>c;
  switch(c){
  case 'g' : double g; cout<<"Greutatea: ";cin>>g; t[g]->tipar(); break;
  case 'n' : char n[100]; cout<<"Numele: ";cin>>n; t[n]->tipar(); break;
  case 'i' : int i;  cout<<"Nr. de index: ";cin>>i;t[i]->tipar(); break;
  case 'e': return 0;
  }
 }
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului apel de funcţie „( )”

- Când este realizată supraîncărcarea operatorului “apel de funcție”, aceasta nu va însemna o nouă variantă de supraîncărcare a unei funcții, ci supraîncărcarea unui operator binar nestatic de forma:

```
expresie (lista_expresii);
```

- Specificul utilizării acestei funcții operator:
  - evaluarea și verificarea listei de argumente se face întocmai ca pentru o funcție obișnuită;
  - chiar dacă operatorul este binar, nu are limită de argumente, deoarece al doilea argument este considerat o lista de argumente.
  - atunci când lista de parametrii este vidă, al doilea argument poate lipsi.

- **Exemplu** *Să elaborăm un program prin intermediul căruia vom utiliza operatorul “()” pentru accesarea unui element al tabloului bidimensional*

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Supraîncărcarea operatorului apel de funcţie „( )”

```
#define dim 8
class Matrix{
 public:
 void print();
 Matrix();
 int & operator()(int i,int j){
return t[i][j];}
 private:
 int t[dim][dim];
};
Matrix::Matrix(){
 for(int i=0;i<dim;i++)
 for(int j=0;j<dim;j++)
  t[i][j]=rand()%100;
}
```

```
void Matrix::print(){
 for(int i=0;i<dim;i++){
  for(int j=0;j<dim;j++)
    cout<<setw(3)<<t[i][j];
  cout<<endl;}
}
int main(){
 Matrix a;
 cout<<"Elementele matricei"<<endl;
   a.print();
 cout<<"pe pozitia 1 2:";
 cout<<a(1,2)<<endl;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Teme pentru acasă

- A învăța și repeta conceptele prezentate în cadrul lecției
- A depana programele prezentate în cadrul lecției

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

# Prelegerea următoare

1. Supraîncărcarea operatorilor unari
2. Conversii
3. Crearea tipului de date fracție

dr. Silviu GÎNCU