

# Operatorii de citire și afișare la consolă



- Operatori binari ce primesc drept prim parametru un obiect de tip **istream** (pentru citire) sau **ostream** (pentru afișare), respectiv obiectul de citit/afișat
- Pot fi supraîncărcați doar prin funcție globală deoarece primul parametru nu este de tipul clasei
- Pot fi declarați ca friend în clasă (dar nu este obligatoriu, se pot utiliza și metode de acces) pentru a putea citi/afișa attribute private sau protected

# Exemplu: supraîncărcarea operatorului <<



```
Class TelefonMobil
{
    //...
    int nivelBaterie;
    //...
    friend ostream& operator<<(ostream&, TelefonMobil);
};

ostream& operator<<(ostream& out, TelefonMobil t)
{
    out << t.nivelBaterie;
    //...
    return out;
}
```

# Exemplu: supraîncărcarea operatorului >>



```
Class TelefonMobil
{
    //...
    int nivelBaterie;
    //...
    friend istream& operator>>(istream&, TelefonMobil&);
};

istream& operator>>(istream& in, TelefonMobil& t)
{
    in >> t.nivelBaterie;
    //...
    return in;
}
```

# Exemplu: utilizare operatori << și >>



```
int main()
{
    TelefonMobil t;
    cin >> t;
    cout << t;
}
```

# Caz particular: operatorul []



- Poate fi supraîncărcat doar prin funcție membră și are întotdeauna doi parametri (obiectul pentru care se apelează - this și un index - ce nu este neapărat numeric)
- Poate fi supraîncărcat în două moduri:
  - doar citire: va putea fi folosit doar ca getter
  - citire/scriere: va putea fi folosit atât ca getter cât și ca setter

# Exemplu: supraîncărcarea operatorului []



```
class TelefonMobil
{
    //...
    int* durateZilniceUtilizare;
    int nrZile;
    //permite acces doar in modul citire
    int operator[](int index)
    {
        if(index >=0 && index < nrZile)
        {
            return durateZilniceUtilizare[index];
        }
        else return -1;
    }
};
```

# Exemplu: utilizare operator [] precedent



```
int main()
{
    TelefonMobil telefonPersonal;
    //acest lucru este permis
    int valoare = telefonPersonal[1];
    //acest lucru nu este permis
    //telefonPersonal[1] = 7;
}
```

## Exemplu: supraîncărcarea operatorului []



```
class TelefonMobil
{
    //...
    int* durateZilniceUtilizare;
    int nrZile;
    //permite acces citire/scriere
    int& operator[](int index)
    {
        if(index >=0 && index < nrZile)
        {
            return durateZilniceUtilizare[index];
        }
    }
};
```

# Exemplu: utilizare operator [] precedent



```
int main()
{
    TelefonMobil telefonPersonal;
    //acest lucru este permis
    int valoare = telefonPersonal[1];
    //acum si acest lucru este permis
    telefonPersonal[1] = 7;
}
```

# Caz particular: operatorul cast



- Are întotdeauna 1 parametru (obiectul de clasa curentă)
- Nu are tip returnat (numele operatorului este tipul returnat)
- Poate fi supraîncărcat doar prin funcție membră
- Folosit la conversia între diverse tipuri de date
- Are două forme: explicită și implicită

# Exemplu: supraîncărcarea operatorului cast la int



```
class TelefonMobil
{
    //...
    int nivelBaterie
    //este automat implicit
    operator int()
    {
        return nivelBaterie;
    }
};
```

# Exemplu: utilizare operator cast



```
int main()
{
    TelefonMobil telefonPersonal;
    //utilizare operator cast implicit
    int nivel = telefonPersonal;
    //poate fi utilizat si cast explicit
    int nivel = (int)telefonPersonal;
}
```

## Exemplu: supraîncărcarea operatorului cast la int (forma explicită)



```
class TelefonMobil
{
    //...
    int nivelBaterie
    //varianta explicita
    explicit operator int()
    {
        return nivelBaterie;
    }
};
```

# Exemplu: utilizare operator cast explicit



```
int main()
{
    TelefonMobil telefonPersonal;
    //utilizare operator cast explicit
    int nivel = (int)telefonPersonal;
    //castul implicit nu poate fi utilizat
    //int nivel = telefonPersonal;
}
```

# Caz particular: operatorul funcție



- Are număr variabil de parametri
- Permite utilizarea obiectelor, instanțe ale clasei în care este supraîncărcat, ca și funcții
- A nu se confunda cu apelul constructorilor
- Se poate supraîncărca doar prin funcție membră

## Exemplu: supraîncărcarea operatorului funcție



```
class TelefonMobil
{
    //...
    int nivelBaterie
    //...
    //operatorul functie ce primeste un parametru de tip int si returneaza int
    int operator()(int nivel)
    {
        return nivelBaterie + nivel;
    }
};
```

# Exemplu: utilizare operator funcție



```
int main()
{
    TelefonMobil telefonPersonal;
    //obiectul creat este utilizat ca si o functie
    int nivel = telefonPersonal(50);
    //a nu se confunda cu apelul unui constructor de forma
    //TelefonMobil telefonPersonal(50);
}
```

# Reguli pentru a supraîncărca ceilalți operatori



- Cum funcționează operatorul și ce returnează pentru tipurile de date standard (int, float, string, etc)?
- Poate fi apelat în cascadă?
- Modifică sau nu obiectul curent?
- Este un operator comutativ?
- Ce sens îi pot da operatorului pentru clasa curentă astfel încât să fie ușor de utilizat și înțeles de ceilalți programatori?

# Conversia dintre obiecte



- Există mai multe modalități de conversie dintr-un tip de obiect în altul:
  - supraîncărcarea constructorului de copiere din clasa rezultat
  - supraîncărcarea operatorului cast al clasei sursă
- Cele două modalități nu pot coexista, în caz contrar compilatorul va returna eroare de ambiguitate