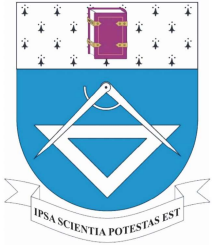


Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare

PROGRAMARE II

Curs 3

C++. Tipul abstract de dată – Clasa.

Principiile de bază are POO

- **Abstractizarea datelor**
- **Incapsularea**
- Moștenirea
- Polimorfismul

Constructori şi destructori

- Constructorul:
  - metodă specială a unei clase
  - are același nume ca și clasa
  - nu are un tip de return
  - este apelat automat la declararea obiectului
- Cu ajutorul constructorului:
  - se creează un obiect pentru care se face alocarea spațiului de memorie necesar
  - se pot inițializa variabilele membre ale clasei
- Constructor implicit (generat de compilator)
  - se apelează atunci când nu există nicio declarație a unui constructor în clasă

Constructori

- Alte tipuri de constructori:
  - Constructor fără listă de argumente (numit și constructor implicit scris de utilizator)
  - Constructor de inițializare (cu listă de argumente sau cu parametri)
  - Constructor de copiere
  - Constructor de mutare
- Dacă se definește cel puțin un constructor, nu se va mai genera constructorul implicit generat de compilator
- Pot exista mai mulți constructori ai aceleiași clase
- Toate declarațiile obiectelor trebuie să respecte un model din mulțimea constructorilor clasei

Constructori

```
class Data
{
	int zi, luna, an;
public:
	void afisare();
	Data(int z, int l, int a);
};
void Data::afisare()
{
	std::cout << "Data calendaristica este" << zi << "-"
<< luna << "-" << an << std::endl;
}
Data::Data(int z, int l, int a)
{
	std::cout << "S-a apelat:Data(...)" << std::endl;
	zi = z;
	luna = l;
	an = a;
}
int main()
{
	Data d1(1,2,1900),d2(3,4,2000);
	d1.afisare();
	return 0;
}
```

Constructori

- Pentru o variabilă definită astfel în funcția main:

```
Data d;
```

- Error 1 error C2512: 'Data' : no appropriate default constructor available

Constructor fără listă de argumente

```
class Data
{
        int zi, luna, an;
public:
        void afisare();
        Data(int z, int l, int a);
        Data();
};


Data::Data()
{
         std::cout << "S-a apelat Data( )" << std::endl;
}
```

Constructorul de copiere

- Este un constructor special al clasei
- Este utilizat pentru a copia un obiect existent de tipul clasei
- Are un singur argument, o referință către obiectul ce va fi copiat
- Forma generală este:

```
MyClass( const MyClass& other );
MyClass( MyClass& other );
```

- În cazul în care o clasă conține variabile de tip pointer, trebuie făcută alocare dinamică de memorie

Exemplu constructor de copiere

```
class Data
{
        int zi, luna, an;
public:
        void afisare();
        Data(int z, int l, int a);
        Data();
        Data(Data &);
};
```

```
Data::Data(Data & d)
{
        zi=d.zi;
        luna=d.luna;
        an=d.an;
}
```

În funcția main:

```
Data d3(d2);

d3.afisare();
```

Se afisează pe ecran: Data calendaristica este 3-4-2000

Constructorul de copiere

- Alte cazuri când se apelează constructorul de copiere:
  - La inițializarea unui obiect nou creat cu un alt obiect

```
Data d2 = d1;
```

  - La transferul parametrilor unei funcții prin valoare
  - **Funcție ce returnează un obiect**

Transferul parametrilor prin valoare

- La transferal parametrilor prin valoare se apelează constructorul de copiere
- Pentru a evita acest apel și implicit o copiere a obiectului:
  - Se transferă parametrul prin referință
  - Se face obiectul transferat `const` pentru a semnala că nu se modifică în funcție, deci e dată de intrare

```
bool Data::comparaAn(const Data &d)
{
    if (an > d.an)
        return true;
    return false;
}
```

Funcție ce returnează un obiect

- In clasă se adaugă:

```
public:
    Data increment ();
```

- Definiția funcției:

```
Data Data::increment ()
{
        Data temp;
        temp.zi = zi + 1;
        temp.luna = luna + 1;
        temp.an = an;

        cout << "inainte de return"<<endl;
        return temp;
}
```

Constructori

```
Data::Data()
{
        std::cout << "S-a apelat constr. fara argumente" << std::endl;
}

Data::Data(int z, int l, int a)
{
       std::cout << "S-a apelat constr. cu lista de argumente" <<
std::endl;
       zi = z;
       luna = l;
       an = a;
}
Data::Data(Data & d)
{
       std::cout << "S-a apelat constr. de copiere" << std::endl;
       zi=d.zi;
       luna=d.luna;
       an=d.an;
}
```

Main:

```
int main()
{
        std::cout<<"d1: ";
        Data d1 (9,1,2012);

        std::cout<<"d2: ";
        Data d2;
        d2 = d1.increment();

        std::cout << "dupa apelul functiei increment"<< std::endl;

        d2 = d1;
        return 0;
}
```

Rezultatul rulării programului:

```
d1: S-a apelat constr. cu lista de argumente
d2: S-a apelat constr. fara argumente
S-a apelat constr. fara argumente
inainte de return
S-a apelat constr. de copiere
dupa apelul functiei increment
```

Constructor de copiere implicit

- Exista un constructor de copiere implicit care copiază bit cu bit obiectul sursă în obiectul destinație
  - Atunci când există alocări dinamice în cadrul membrilor obiectului trebuie creat un constructor ce va face alocările necesare de memorie

Destructorul

- Ajută la eliberarea zonelor de memorie
- Are rol opus constructorului
- Nu are tip de return
- Are numele clasei precedat de caracterul ~
- Nu primește niciun parametru
  - O clasă are un singur destructor!
- **Se apelează automat** când domeniul de vizibilitate a obiectului s-a terminat
- Se recomandă să se realizeze atunci când clasa conține pointeri care sunt alocați dinamic în constructor sau în alte metode prezente în clasă

Destructorul

```
class Data
{
	int zi, luna, an;
public:
	void afisare();
	Data(int z, int l, int a);
	Data();
	~Data();
};
Data::~Data()
{
	std::cout <<"s-a apelat destructorul" <<std::endl;
}
```

Destructorul

```
int main()
{
        Data d;

        Data d2 (3,4,2000);

        Data d3(d2);

        d2.afisare();

        d3.afisare();
        return 0;
}
```

- Se va afișa mesajul:

```
S-a apelat Data( )
S-a apelat:Data(...)
Data calendaristica este3-4-2000
Data calendaristica este3-4-2000
s-a apelat destructorul
s-a apelat destructorul
s-a apelat destructorul
```

Constructorul de copiere – exemplu .h

```
class Persoana
{
        char * nume;
        int varsta;
public:
        Persoana();
        Persoana (char * n, int v);

        ~Persoana ();
        void afiseaza();
        Persoana schimbaVarsta(int);
};
```

Fct.cpp

```
#include <iostream>
#include "header.h"

using namespace std;

Persoana::Persoana()
{
	nume = new char [1] ();
	varsta = 0;
}

Persoana::Persoana(char * n, int v)
{
	nume = new char [strlen (n) + 1];
	strcpy_s (nume, strlen (n) + 1, n);
	varsta = v;
}
```

Fct.cpp

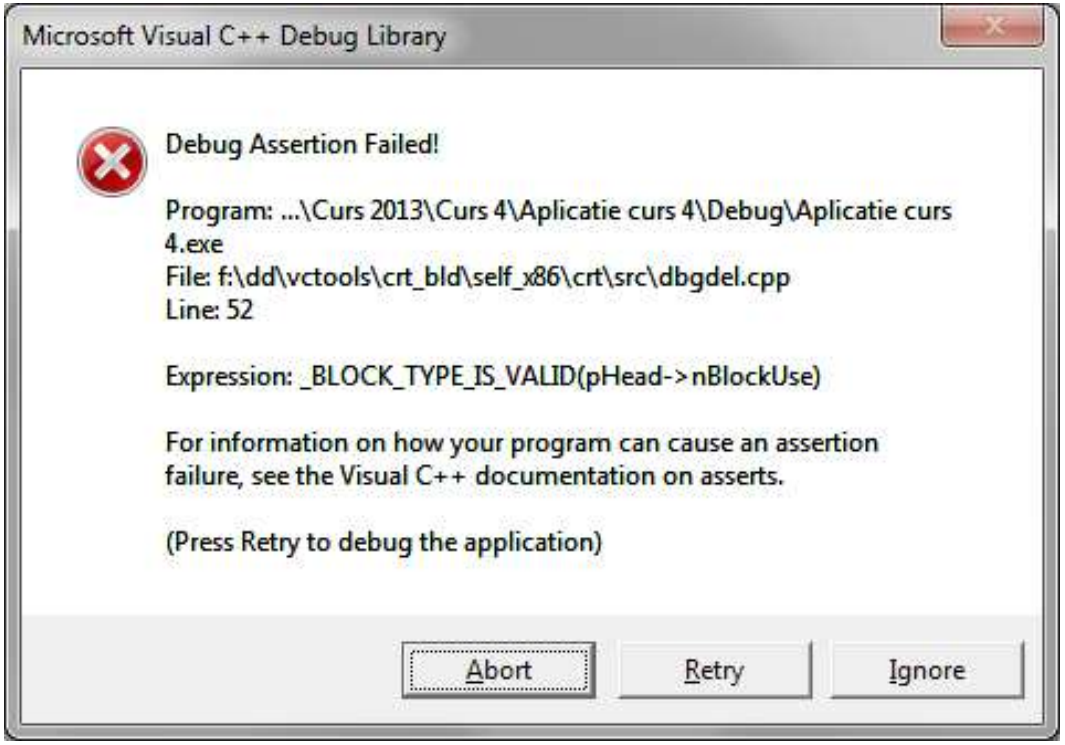
```
Persoana::~Persoana ()
{
	std::cout << "S-a apelat destructorul" << std::endl;
	if (nume)
		delete [] nume;
}

void Persoana::afiseaza()
{
	std::cout << "Nume=" << nume << " varsta=" << varsta
<< std::endl;
}

Persoana Persoana::schimbaVarsta(int n)
{
	Persoana temp (nume, varsta);
	varsta = n;
	return temp;
}
```

Main.cpp

```
#include <iostream>
#include "header.h"

int main ()
{
	Persoana p1 ("Pavel",22);
	p1.afiseaza();

	Persoana p2 ("Ana",23);

	Persoana p3 = p2.schimbaVarsta(25);

	p2.afiseaza();

	p3.afiseaza();
	return 0;

}
```

Ce se afiseaza?

```
Nume=Pavel varsta=22
Nume=Ana varsta=25
Nume=ŢŢŢŢŢŢŢŢEtł2ŢŢ varsta=23
```



De ce?

- Pentru ca lipseste constructorul de copiere! Debug.

- Se adauga in clasa un constructor de copiere:

```
Persoana (Persoana & p);
```

- Cu definitia

```
Persoana::Persoana (Persoana & p)
{
        nume = new char [strlen (p.nume) + 1];
        strcpy_s (nume, strlen (p.nume) + 1, p.nume);
        varsta = p.varsta;
}
```

Liste de inițializare

- Prin utilizarea listelor de inițializare se apelează numai constructorii de copiere ai claselor membrilor pe care îi inițializează
- Utilizarea constructorului de copiere este mai eficace dpdv al timpului în comparație cu apelarea constructorului fară argumente (sau a celui implicit) urmată de o atribuire
- *Sunt folosite în principal pentru a controla ce constructori se apelează în cadrul claselor derivate*

Liste de inițializare

```
Data::Data(int z, int l, int a)
{
        std::cout << "S-a apelat constr. cu lista de argumente"
                << std::endl;
        zi = z;
        luna = l;
        an = a;
}
```

Liste de inițializare

```
Data::Data(int z, int l, int a)
{
        std::cout << "S-a apelat constr. cu lista de argumente"
                << std::endl;
        zi = z;
        luna = l;
        an = a;
}
```

```
Data::Data(int z, int l, int a):zi(z),luna(l),an(a)
{
        std::cout << "S-a apelat constr. cu lista de argumente"
                << std::endl;
}
```

- Listele de inițializare sunt părți ale definițiilor constructorilor și nu ale declarațiilor!

Constructori expliciți

- Se declară folosind cuvântul cheie `explicit`
- Sunt constructori ce nu pot lua parte în nici o conversie implicită
- Numai constructorii cu un singur argument pot lua parte într-o conversie implicită

```
class A
{
public:
        explicit A(int);
};

A::A(int)
{}


 A a1 = 37;
```

```
error C2440: 'initializing' : cannot convert from 'int' to 'A'
```

Alocarea și dealocarea dinamică de memorie

- In C++ alocarea și dealocarea dinamică de memorie se face cu ajutorul operatorilor:
  - `new`
  - `delete`
- La folosirea operatorului `new` pentru alocarea unui singur obiect sau variabile, este apelat automat constructorul implicit sau, dacă există constructori declarati, cel ce corespunde listei de argumente
- La folosirea operatorului `delete`, este apelat automat destructorul clasei

```
int main()
{
        std::cout << "d1: ";
        Data d1;
    ------------------------------------------------------------------------------------------
        std::cout << "d2: ";
        Data * d2 = new Data;

        std::cout << "d3: ";
        Data * d3 = new Data (1,2,2003);

        std::cout << "d4: ";
        Data * d4 = new Data (*d3);

        std::cout << "d5: ";
        Data * d5 = (Data*)malloc(sizeof(Data));

        std::cout << std:: endl << "final:";

        delete d2;
        delete d3;
        delete d4;

        free(d5);
        //delete d5;

        return 0;
    ------------------------------------------------------------------------------------------
}
```

Rezultatul rulării programului

```
d1: S-a apelat constr. fara argumente
d2: S-a apelat constr. fara argumente
d3: S-a apelat constr. cu lista de argumente
d4: S-a apelat constr. de copiere
d5:
final:s-a apelat destructorul
s-a apelat destructorul
s-a apelat destructorul
s-a apelat destructorul
```