Universitatea Tehnică „Gheorghe Asachi" din Iaşi
Facultatea de Automatică şi Calculatoare

PROGRAMARE II

Curs 3

C++.Tipul abstract de dată - Clasa

Principiile de bază are POO

- **Abstractizarea datelor**
- **Incapsularea**
- Polimorfismul
- Moştenirea

Abstractizarea datelor

- Abstractizarea este procesul de recunoaştere şi de concentrare asupra caracteristicilor importante ale unui obiect
- Concentrare asupra semnificaţiei unui tip de date şi asupra comportamentului său
  - Înlăturarea detaliilor de implementare irelevante
  - Generarea unor funcţii care crează şi manipulează datele şi pe care se bazează toate funcţiile de acces
- Abstractizarea datelor nu este o ştiinţă exactă: există nenumerate moduri de abstractizare a unui obiect sau a unei situaţii

Clasa

- Este o extindere a noțiunii de structură din C
- Conține atât date, cât și funcții (metode)
- Permite restricționarea accesului la membri prin folosirea specificatorilor de acces
- Poate fi declarată utilizând atât cuvantul `class` cât și cuvantul `struct`, diferența facând-o numai specificatorul de acces implicit
- Un obiect este o instanțiere a unei clase

Incapsularea

- Posibilitatea de a ascunde utilizatorului unei clase detaliile ei de implementare se numeste incapsulare
- O clasă va conține
  - O parte privată – implementare
  - O parte publică – interfață
- Exemplu: clasă ce implementează o stivă
  - Implementarea se poate schimba (de la o implementare cu tablouri la o implementare cu liste înlănțuite)
  - Utilizatorul nu are acces direct la tablou sau la listă
  - Interfața rămâne aceeași (funcțiile push, pop, etc)
- Incapsularea datelor este realizată în C++ prin utilizarea specificatorilor de acces

Specificatori de acces

- Modifică drepturile de acces către membrii clasei
  - `private`: membrii privați pot fi accesați numai de membri aceleiași clasa *sau de membrii claselor prietene*
  - `protected`: membrii protejați sunt accesibili în cadrul aceleiași clase *sau dintr-o clasă prietenă, dar și din clasele derivate din acestea*
  - `public`: membrii publici sunt accesibili de oriunde din domeniul de vizibilitate al obiectului de tip clasă

Exemplu specificatori de acces

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

class Data
{
private:
  unsigned char zi, luna;
public:
  int an;
};


int main()
{

  Data d1;
  d1.zi = 1;
  d1.luna = 2;
  d1.an = 1900;

  cout << "Data calendaristica este" << (int) d1.zi << "-" << (int)
  d1.luna << "-" << d1.an;
  return 0;
}
```

Exemplu specificatori de acces

- Error 1 error C2248: 'Data::zi' : cannot access private member declared in class 'Data' […]
- Error 2 error C2248: 'Data::luna' : cannot access private member declared in class 'Data'[…]

Specificator de acces implicit

```
class Data
{
private:
  unsigned char zi, luna;
public:
  int an;
};
```

```
class Data
{
  unsigned char zi, luna;
public:
  int an;
};
```

- În cadrul unei clase, specificatorul de acces implicit este `private`
- Membrii celor două clase au aceiaşi specificatori de acces
- În cadrul unei structuri, specificatorul de acces implicit este `public`

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
  - Se apelează atunci când nu există nicio declarație a unui constructor în clasă

Constructori

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

Constructori

```
class Data
{
	unsigned char zi, luna;
	int an;
public:
	void afisare();
	Data(unsigned char z, unsigned char l, int a);
};
void Data::afisare()
{
	cout << "Data calendaristica este" << (int)zi << "-"
<< (int)luna << "-" << an << endl;
}
Data::Data(unsigned char z, unsigned char l, int a)
{
	cout << "S-a apelat:Data(...)" << 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
{
       unsigned char zi, luna;
       int an;
public:
       void afisare();
       Data(unsigned char z, unsigned char l, int a);
       Data();
};

Data::Data()
{
      cout << "S-a apelat Data( )" << 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 de memorie

Exemplu constructor de copiere

```
class Data
{
       unsigned char zi, luna;
       int an;
public:
       void afisare();
       Data(unsigned char z,
unsigned char l, int a);
       Data();
       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 este3-4-2000

Constructorul de copiere

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

```
Data d2 = d1;
```

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

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

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

Main:

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

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

	cout << "dupa apelul functiei increment" << 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 nici un 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
{
	unsigned char zi, luna;
	int an;
public:
	void afisare();
	Data(unsigned char z, unsigned char l, int a);
	Data();
	~Data();
};
Data::~Data()
{
	cout <<"s-a apelat destructorul" <<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

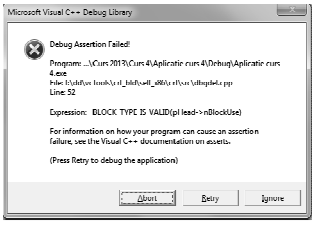
```
Persoana::~Persoana ()
{
	cout << "S-a apelat destructorul" << endl;
	if (nume)
		delete [] nume;
}

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

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

Main.cpp

```
#include <iostream>
#include "header.h"
using namespace std;

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;
}
```