Laboratorul 1

Primii pași în limbajul C++

Ce ne propunem astăzi

În laboratorul de astăzi ne propunem crearea unor programe simple pentru ca studenții să se familiarizeze cu unele din conceptele de bază în Programarea Orientată pe Obiecte. În cadrul acestei aplicații vom utiliza câteva dintre facilitățile POO oferite de limbajul C++.

Programarea Orientată pe obiecte (POO) a apărut din cauza diferitelor limitări ale programării structurate și a limbajelor care permit aplicarea acestei paradigme. Programarea orientată pe obiecte a luat cele mai bune idei din programarea structurată și le-a combinat cu alte concepte noi, care sunt descrise mai jos.

Principalele concepte (caracteristici) ale **POO** sunt:

- **încapsularea** este mecanismul prin care se contopesc datele cu codul pentru protejarea acestora de interferențele din exterior și pentru o localizare mai bună a erorilor
- **moştenirea** - posibilitatea de a extinde funcționalitățiile unei clase
- **polimorfismul** – se folosește în cazul moștenirii când se obține o ierarhie de clase, o metodă poate să aibă implementări diferite

Avantajul folosirii programării orientată pe obiecte sunt:

- proiectarea ușoară și posibilitatea de reutilizare a codului
- siguranța datelor, deoarece obiectele se comportă ca niște “cutii negre”, din cauza conceptelor POO nu se cunoaște din ce sunt formate
- abstractizare, prin aceea că poate obține o imagine de ansamblu a comportamentului obiectelor și cum interacționează ele

Câteva diferențe față de limbajul C

Pentru operațiile de citire și afișare se folosesc obiectele „cin” și „cout” care nu necesită specificarea formatelor.

```
int variabila;
cout << "Dati valoarea numarului intreg: ";
cin >> variabila;
cout << "Valoarea numarului intreg este:" << variabila << endl;
```

Pentru folosirea obiectelor cin și cout, programatorul va trebui să includă header-ul bibliotecii de stream-uri “iostream”. Un stream este un flux definit ca un concept abstract care desemnează orice flux de date de la o sursă la o destinație. Cu ajutorul acestor stream-uri se vor putea realiza operații de citire sau de scriere. Operatorul << are semnificatia “preia de la…” și se folosește pentru operația de afișare, iar operatorul >> are semnificația de “pune la..” și se folosește pentru

operația de citire. Programul din Anexa 1. exemplu 1 va permite utilizatorului să citească numere întregi până la întîlnirea numărului 0, apoi se vor afișa suma numerelor pare și produsul numerelor impare.

Folosirea spațiilor de lucru este adăugată recent în limbajul C++. În programul din Anexa 1 Anexa 1 Exemplu 1 s-a adăugat namespace std ceea ce înseamnă adăugarea unui spațiu de lucru care include biblioteca Standard C++. În limbajul C nu este nevoie de adăugarea acestui spațiu de lucru deoarece funcțiile sunt deja valabile implicit în spațiul de lucru global.

O altă diferență care o aduce limbajul C++ este faptul că declararea variabilelor se poate face oriunde în program, nu este restricționat ca și în limbajul C, la începutul programului sau a funcției. În Exemplul 1 de mai sus se pot observa că variabilele pentru sumă - s și pentru produs - p sunt declarate în interiorul funcței main.

Supraîncărcarea funcțiilor

Supraîncărcarea funcțiilor este definită ca fiind posibilitatea de a utiliza mai multe funcții cu aceeași nume, dar cu tipul și numărul de parametrii diferiți. În Anexa 1 Exemplu 2 se poate observa că prima funcție „suma" returnează suma a două numere întregi, iar a doua funcție „suma" returnează suma a două numere reale.

Operatorii *new* şi *delete*

În limbajul C pentru alocarea dinamică de memorie se folosesc operatorii "malloc" și "free", în limbajul C++ se folosesc operatorii "new" și "delete".

Operatorul "new" este un operator care obține memorie de la sistemul de operare și apoi returnează un pointer la zona de memorie alocată dinamic. În cazul în care alocarea nu se poate realiza va returna NULL.

Operatorul "delete" se folosește în cazul în care se dorește eliberarea memoriei alocate cu operatorul "new".

În Anexa 1 Exemplul 3 se poate urmării folosirea celor doi operatori de alocare dinamică.

Transferul prin referinţă

O referință este un pointer implicit. Există trei variante de folosire a unei referințe: ca un parametru pentru o funcție, ca și valoarea returnării unei funcții sau o referință de sine stătătoare. În acest caz este prezentat doar transferul prin referință.

În cazul în care se folosește transferul prin referință, se transmite doar adresa ei, nu întreaga structură (adică nu se copiază parametrul de apel). Dacă se dorește accesarea unui membru a unei structuri se poate folosi ".", iar dacă îi pointer se folosește "->". Cel mai indicat este folosirea transferului prin referință.

În Anexa 1 Exemplu 4 este prezentată folosirea transferului prin referință, iar folosirea „&i" se numește "referință la obiectul i".

Noțiuni elementare de Programare Orientată pe Obiecte în C++

Clase în C++

O clasă este folosită pentru definirea naturii unui obiect și este unitatea de bază a limbajul C++ în vederea încapsulării. Declararea unei clase definește un tip care face legătură între cod și date. Acest tip nou este folosit pentru declararea obiectelor unei clase. Un obiect este instanța unei clase. Clasele sunt create folosind cuvântul cheie class. Diferența dintre clase și structuri este că o clasă poate include îmtre membrii săi și funcții.

Forma generală a declarării unei clase este prezentat mai jos:

```
class nume_clasa
{
      specificator_de_acces:
            variabile și funcții;
      specificator_de_acces:
            variabile și funcții;
      ….
} lista_obiecte;
```

Lista de obiecte este opțională. În limbajul C++ există trei tipuri de specificatori de access:

- public – funcțiile sau datele pot fi accesate din orice parte a programului
- protected – se folosește în cazul moștenirii și funcțiile și datele clasei sunt accesibile doar în cadrul clasei respective și a claselor derivate
- private – funcțiile sau datele pot fi accesate numai de membrii aceleiași clase

În rândurile de mai jos s-a declarat clasa **Dreptunghi**. Această clasă conține 4 membri: 2 variabile de tipul **int** (**lungime** și **lățime**) în secțiunea **private** (fiind sectiunea cu permisiune implicită) si 2 funcții in secțiunea **public**: **setare_valori()** și **aria()**, în acest caz s-a declarat numai prototipul acestora.

```
class Dreptunghi
{
        int lungime, latime;
public:
        void setare_valori(int, int);
        int aria(void);
 };
```

În Anexa 1 Exemplu 5 apare operatorul de rezoluție care este notat cu "**::**" din definiția functiei **setare_valori()**. Acest operator este utilizat pentru a declara membrii clasei **în afara** acesteia.

Funcțiile inline

O importantă caracteristică în limbajul C++ o constituie funcțiile *inline*, care de obicei sunt folosite în clase. Funcțiile inline se folosesc pentru a economisi resurse și timp de execuție. Programatorii pot folosi aceste funcții pentru a cere compilatorului să insereze blocul funcției în punctul apelării, în loc să creeze instrucțiunile de apel. Nu se folosesc funcțiile inline în cazul funcțiilor recursive sau complexe.

În Anexa 1 Exemplu 6 este prezentată funcția minim care este extinsă în loc să fie apelată și este echivalentă cu problema din Anexa 2 Exemplu 7.

Avantajul folosirii funcțiilor inline este că permite scriere de cod într-un mod foarte eficient. În momentul apelării unei funcții este invocat un mecanism sofisticat de apelare și returnare și de aceea necesită mai mult timp. În cazul folosirii acestor funcții inline aceste mecanisme nu se mai invocă și în acest fel se va economisi timp și spațiu.

Funcțiile inline se folosesc doar pentru funcții de dimensiuni mici, dacă s-ar folosi pentru funcții cu multe linii de cod va rezulta un cod foarte mare, iar unele instrucțiuni vor fi duplicate.

În Anexa 1 Exemplu 8 se poate vedea că funcțiile inline pot fi funcții membru al unei clase.

În Anexa 1 Exemplu 9 se poate observa cum se pot defini funcții scurte în interiorul unei clase. În acest caz, se vor crea automat funcții inline, în cazul în care acest lucru este posibil. Se poate adăuga și cuvântul cheie inline dar nu este necesar și nu este considerată o eroare dacă lipsește.

Sfaturi utile

În cazul în care nu este precizat nici un specificator atunci respectiva funcție sau variabilă este declarat implicit privat.

Partea practică a laboratorului

1. Să se realizeze un program care preia de la tastatură următoarele informaţii: nume, prenume, vârsta, adresă, telefonul unei persoane. După preluare, aceste informaţii trebuie afişate.
2. Să se realizeze două clase: Clasa Sofer, Clasa Masina.
   Clasa Sofer *va conține următoarele variabile: nume sofer, cnp, varsta, adresa*
   Clasa Mașina *va conține următoarele variabile: producător, model, an fabricație, sofer*
   Se vor crea pentru cele două clase funcții de citire și de afișare.

Mod de lucru

Iată pașii care trebuie urmați pentru dezvoltarea cu succes ale programelor:

- se va parcurge tutorialul din Anexa 1.2, unde se va descrie cum se creează un nou proiect în Visual Studio și pașii care trebuie urmați pentru compilarea și rularea unui program cu succes.
- se vor analiza, rula și înțelege programele și principiile pe cale le exemplifică programele prezentate în laborator care se află în Anexa 1.1
- se vor rezolva problemele de la partea practica a laboratorului

Cu ce ne-am ales?

Prin programele prezentate ca exemple, un tutorial și problemele propuse de la partea practică a laboratorului am reușit să ne familiarizăm pe de-o parte cu Programarea Orientată pe Obiecte iar pe de altă parte cu lucrul în mediul de programare Visual Studio.NET. Astfel, am făcut primii pași în utilizarea unui mediu modern, pentru crearea primelor noastre aplicații orientate pe obiecte. Adică am pornit la drum!

Anexa 1

Exemplu 1

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

using namespace std;

int main()
{
    int nr;
   cout << "Valoare numarului intreg este: ";
   cin >> nr;
   // declararea și inițializarea variabilelor pentru sumă și produs
   int s = 0, p = 1;
   // folosirea instrucțiunii while pentru citirea numerelor până la întâlnirea numărului 0
   while (nr != 0)
   {
      //verificarea numărului dacă este număr par
      if (nr % 2 == 0)
            s = s + nr;
          else
            p = p * nr;
     cout << "Valoare numarului intreg este: "; cin >> nr;
    }
    cout << "Suma numerelor pare este: " << s << endl;
    cout << "Produsul numerelor impare este: " << p;

    getch();
   return 0;
}
```

Exemplu 2

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

int main()
{
        int nr;
        cout << "Valoare numarului intreg este: ";
        cin >> nr;
        int s = 0, p = 1;
        //verificam daca numarul introdus este 0
        while (nr != 0)
        {
                //verificam daca numarul este par
                if (nr % 2 == 0)
                        //daca numarul este par atunci se va aduna la suma anterioara
                        s = s + nr;
                else
                        //daca numarul este impar atunci se va înmulti cu produsul anterior
                        p = p * nr;
                cout << "Valoare numarului intreg este: ";
                //se citeste urmatorul numar
                cin >> nr;
        }
       //se afiseaza suma si produsul
        cout << "Suma numerelor pare este: " << s << endl;
        cout << "Produsul numerelor impare este: " << p;
        getchar();
        return 0;
}
```

Exemplu 3

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

//functie ce returneaza suma a doua numere intregi
int suma(int a, int b)
{
	return a + b;
}
//functie ce returneaza suma a doua numere reale
float suma(float a, float b)
{
	return a + b;
}

int main()
{
	int a, b;
	//citirea celor doua numare intregi
	cout << "Valoare numarului intreg a este: ";
	cin >> a;
	cout << "Valoare numarului intreg b este: ";
	cin >> b;
	//afisarea sumei numerelor intregi
	cout << "Suma celor doua numere intregi este: " << suma(a, b) <<endl;
	float x, y;
	//citirea numerelor reale
	cout << "Valoare numarului real x este: ";
	cin >> x;
	cout << "Valoare numarului real y este: ";
	cin >> y;
	//afisarea sumei numerelor reale
	cout << "Suma celor doua numere reale este: " << suma(x, y);
	getchar();
	return 0;
}
```

Exemplu 4

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

//definirea structurii masina
typedef struct
{
	char marca[20];
	char model[30];
	float consum, pret;
}masina;

int main()
{
	//crearea unui pointer de tip masina
	masina *ma;
	//alocarea dinamica a memoriei pentru pointer
	ma = new masina;
	//citirea informatiilor
	cout << "Dati marca masinii: ";
	cin >> ma->marca;
	cout << "Dati modelul masinii: ";
	cin >> ma->model;
	cout << "Dati consumul de combustibil: ";
	cin >> ma->consum;
	cout << "Dati pretul masinii: ";
	cin >> ma->pret;
//afisarea informatiilor
```

```
cout << "Marca: " << ma->marca << " Model: " << ma->model << " Consum: " << ma->consum << " Pret: " << ma->pret;
        delete ma;
        getch();
        return 0;
}
```

Exemplu 5

```
#include <iostream>

using namespace std;
//functie care va returna o valoarea negativa
int negativ(int &i)
{
        return -i;
}
int main()
{
        int valoare;
        cout << "Dati valoarea: ";
        cin >> valoare;
        cout << "Valoarea negativa este: " << negativ(valoare);
        getchar();
        return 0;

}
```

Exemplu 6

```
#include <iostream>

using namespace std;
//crearea unei clase
class Dreptunghi
{
        int lungime, latime;
public:
        void setare_valori(int, int);
        int aria(void);
};
//funcția setare valori care a fost declarata in clasa Dreptunghi
void Dreptunghi::setare_valori(int a, int b)
{
        lungime = a;
        latime = b;
}
//functia pentru calcularea ariei
int Dreptunghi :: aria(void)
{
        return (lungime * latime);
}

int main()
{
        //crearea unui obiect de tip Dreptunghi
        Dreptunghi drept;
       //setarea valorilor
        drept.setare_valori(3, 8);
       //afisarea ariei
        cout << "Aria este: " << drept.aria();
        getchar();
        return 0;
}
```

Exemplu 7

```
#include <iostream>

using namespace std;

// crearea unei functii inline care va returna numarul minim
inline int minim(int x, int y)
{
	return x < y ? x : y;
}

int main()
{
	//citirea a doua valori intregi
	int x, y;
	cout << "Dati valoarea lui x: ";
	cin >> x;
	cout << "Dati valoarea lui y: ";
	cin >> y;
	//afisarea minimului
	cout << "Minimul este: " << minim(x, y);
	getchar();
	return 0;
}
```

Exemplu 8

```
#include <iostream>

using namespace std;

int main()
{
	// citirea a doua numere intregi
	int x, y;
	cout << "Dati valoarea lui x: ";
	cin >> x;
	cout << "Dati valoarea lui y: ";
	cin >> y;
	//afisarea minimului
	cout << "Minimul este: " << (x < y ? x : y);
	getchar();
	return 0;
 }
```

Exemplu 9

```
#include <iostream>

using namespace std;
//crearea clasei Exemplu
class Exemplu {
	int x, y;
public:
	void initializare(int, int);
	void afisare();
};
//crearea unei functii inline pentru initializarea a doua variabile intregi
inline void Exemplu::initializare(int a, int b)
{
	x = a;
	y = b;
}
//crearea unei functii inline pentru afisarea celor doua variabile intregi
inline void Exemplu::afisare()
```

```
{
	cout << "Valoarea lui x:" << x << endl;
	cout << "Valoarea lui y: " << y;
}

int main()
{
	//crearea unui obiect de tip Exemplu
	Exemplu ex;
	int var1, var2;
	//citirea a doua variabile intregi
	cout << "Valoarea var 1: ";
	cin >> var1;
	cout << "Valoarea var 2: ";
	cin >> var2;
	// apelarea functiei initializare
	ex.initializare(var1, var2);
	// afisarea
	ex.afisare();
	getchar();
	return 0;
}
```

Exemplu 10

```
class Exemplu {
	int x, y;
public:
	//funcție inline într-un mod automat
	void initializare(int a, int b)
	{
		x = a;
		y = b;
	}
	//functie de afisare
	void afisare()
	{
		cout << "Valoarea lui x:" << x << endl;
		cout << "Valoarea lui y: " << y;
	}
};

int main()
{
	//crearea unui obiect de tip Exemplu
	Exemplu ex;
	int var1, var2;
	//citirea a doua variabile intregi
	cout << "Valoarea var 1: ";
	cin >> var1;
	cout << "Valoarea var 2: ";
	cin >> var2;
	ex.initializare(var1, var2);
	ex.afisare();
	getchar();
	return 0;
}
```