**TEST Nr.14-2**

**pentru atestarea nr.2 la disciplina ”Programare orientată pe obiecte”**

1. Formulați definiția de *funcție prietenă* a unei clase.

O **functie** este **prietena** cu o clasa daca are acces la datele membru private ale acelei clase. O functie prietena poate fi o functie **globala**sau chiar o functie **membru** a altei clase. O **clasa** este **prietena** cu o alta clasa daca ea are acces la datele membru ale acesteia.

O **functie,**respectiv o **clasa** **prietena** se declara utilizand cuvantul ***friend*** astfel:

a)       ***friend Tip\_functie Nume\_functie****Lista\_parametri\_formali* **Functie** ***friend*** globala

b)       ***friend Tip\_functie Nume clasa* *Nume\_functie****Lista\_par.\_formali***Functie** ***friend*** membru

c)       ***friend Nume clasa* Clasa** ***friend***

# include <math.h> #include <iostream.h>

class **Punct**;

class **Cerc**

**int TaieOx();**

};

class **Punct**

Punct (float x0, float y0)

**friend int *Cerc*::TaieOx();**

};

**int *Cerc*::TaieOx()**

void main ()

**(10 p.)**

1. Formulați regulile de supraîncărcare a funcțiilor.

C ++ permite scrierea codului flexibil și ușor de înțeles folosind conceptul cunoscut ca suprasarcină. Permite realizarea diferitelor funcționalități în cadrul codului existent, cu modificări foarte minime, prin reducerea codului duplicat. Practic, există în principal două tipuri principale de supraîncărcare care sunt suportate de C ++.

C ++ ne permite să scriem funcții cu același nume, dar cu diferența de tipuri de date sau în numărul de argumente care i-au fost transmise, această caracteristică este cunoscută sub denumirea de supraîncărcare a funcției în C ++. Această caracteristică permite dezvoltatorilor să definească funcțiile cu același nume în același domeniu de aplicare. Cu același nume, funcțiile reprezintă aceeași funcționalitate, permițând astfel realizarea unui polimorfism în timp. Supraîncărcarea funcției are un avantaj că îmbunătățește lizibilitatea codului.

Există o altă caracteristică furnizată în C ++, care permite redefinirea funcționalității existente a operatorilor încorporați, care este cunoscută sub denumirea de supraîncărcare a operatorilor **.** Această caracteristică permite utilizarea operatorilor integrați pe tipuri definite de utilizator. Supraîncărcarea operatorului simplifică codul prin redefinirea funcționalității conform cerințelor utilizatorului. În acest articol ne vom concentra atât asupra supraîncărcării funcției, cât și asupra supraîncărcării operatorului, vom vedea detaliile acesteia și cum este utilizată în C ++.

* **Supraîncărcarea funcțiilor:** Ne permite să definim funcția cu același nume, dar distinge funcțiile în funcție de tipul de parametri care le-a fost trecut sau de numărul de parametri care le-au fost transmise. Deci, toate funcțiile vor avea același nume, dar vor avea fie un tip de date diferit, fie vor avea un număr diferit de parametri. Când funcția este apelată, compilatorul va alege funcția cu tipul de parametru care se potrivește și care va potrivi numărul de argumente. Acum, dezvoltatorii săi pot alege ce funcție să apeleze în conformitate cu cerințele. Aceștia pot alege funcția corespunzătoare trecând parametrii urmând regulile.
* **Suprasarcina operatorului:** permite operatorilor să lucreze pentru tipuri de date definite de utilizator, adică clase. Operatorii existenți sunt supraîncărcați și li se oferă puterea de a opera pe clasa și obiectele definite de utilizator. Supraîncărcarea operatorului se realizează prin definirea funcției cu numele special. Funcția va avea numele de „operator” urmată de simbolul operator. Putem utiliza simbolul operator direct pe tipul de date definit de utilizator și să efectuăm operația. Acțiunea sau operațiunea necesară este definită de acea funcție specială de către noi. Prin intermediul supraîncărcării operatorului, putem efectua operațiuni de diferite tipuri pe același tip de date.

**(10 p.)**

3. Determinați dacă sunt erori în următorul program și explicați cum ele pot fi corectate:

class Char

{ char c;

public:

Char(char C){c=C;}

friend int mai\_mic(Char,Int);

};

class Int

{ int i;

friend int mai\_mic(Char,Int);

public:

Int(int I){i=I;}

};

int mai\_mic(Char a,Int b)

{ return a.c<b.i;}

main()

{

Char x8(’a’); Int xa(8);

cout<<mai\_mic(xa,x8);

} **(15 p.)**

4. Supraîncărcați funcția ***sortare()***, care având ca parametri un tablou de elemente și dimensiunea tabloului, realizează:

a) tablourile de numere întregi – le sortează crescător,

b) tablourile de numere reale – le sortează descrescător.

Apelați funcția dată în programul principal ***main()***, pentru un tablou de numere intregi și pentru un tablou de numere reale, și afișați tablourile sortate la ecran.

void Sortare(int Vector[], int dim)

{

for(int i = 0; i < dim - 1; i++)

{

for(int j = i + 1; j < dim; j++)

{

if(Vector[i] > Vector[j])

{

int aux = Vector[i];

Vector[i] = Vector[j];

Vector[j] = aux;

}

}

}

}

void Sortare(double tablou[],int n)

{

for(int i=0;i<n;i++)

{

for(int j=i+1;j<n;j++)

{

if(tablou[i]<tablou[j])

{

int temp =tablou[i];

tablou[i]=tablou[j];

tablou[j]=temp;

}

}

}

}

main()

{ int tab\_int[]{2,5,4,-1,1,3};

double tab\_real[]{7.2,4.9,9.6,-3.2,5.3};

Sortare(tab\_real,5);

for (int i = 0; i < 5; i++)

{

cout<<tab\_real[i]<<" ";

}

cout<<endl;

Sortare (tab\_int,6);

for (int i = 0; i < 6; i++)

{

cout<<tab\_int[i]<<" ";

}

}

**(30 p.)**

5. Scrieți programul C++ în care se declară clasa ”Produs” cu proprietățile: Denumirea produsului, Prețul pentru o unitate de produs, Cantitatea. Definiți un constructor cu parametri al clasei. Fie ***ob*** – un obiect al clasei ”Produs”; ***n*** – un număr întreg; ***α*** – orice operator unar; ***β*** – orice operator binar. Supraîncărcați operatorul ***α*** prin metoda funcțiilor membre, iar operatorul ***β*** prin metoda funcțiilor prietene, în felul următor:

a) expresia: ***α ob;*** va mări prețul produsului ***ob*** cu 1 leu,

b) expresia: ***ob β n;*** va mări cantitatea produsului ***ob*** cu ***n*** unități.

class Produs{

string denumire;

double pret;

int cantitate;

public:

void Print()

{

cout<<denumire<<endl;

cout<<cantitate<<endl;

cout<<pret<<endl;

}

Produs(string D,double P,int C)

{

denumire=D;

pret=P;

cantitate=C;

}

friend Produs operator\*(Produs &other,int n);

Produs operator++()

{

this->pret+=1;

return \*this;

}

};

Produs operator\*(Produs &other ,int n){

other.cantitate +=n;

return other; }

int main()

{

Produs ob("Bomboane"20,4);

++ob;

ob\*15;

return 0 ;

}

**(35 p.)**

**BAREM DE NOTARE**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nota** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Punctaj acumulat** | **E** | **0-10** | **11-20** | **21-29** | **30-45** | **46-60** | **61-75** | **76-85** | **86-95** | **96-100** |
|  | | | | |  | | | | | |