Laboratorul 4

Supraîncărcarea funcțiilor și a operatorilor, metode „friend”, vectorii STL

# Exemple de bază

### ***Supraîncărcarea funcțiilor***

Supraîncărcare (override) înseamnă că avem două funcții cu același nume, dar parametrii diferiți

*int* sum(*int* a, *int* b) {

*return* a + b;

}

*int* sum(*int* a, *int* b, *int* c) {

*return* a + b + c;

}

*int* main() {

*// afiseaza 5*

cout << "a) " << sum(2, 3) << '\n';

*// afiseaza 14*

cout << "b) " << sum(3, 5, 6) << '\n';

*// TODO urmatorul cod afiseaza 16. Ce puteti adauga in program a.i. sa afiseze 16.3?*

cout << "c) " << sum(2.3, 14.0) << '\n';

*return* 0;

}

Este foarte util că putem supraincarca functii cu denumiri simple, dar argumente din clase diferite:

LongNumber add(LongNumber a, LongNumber b) {

*// ...*

}

LongNumber add(Fraction f1, Fraction f2) {

*// ...*

}

*void* afisare(LongNumber num){

*// ...*

}

*void* afisare(Fraction f){

*// ...*

}

*int* main() {

*int* cifreA[12] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 1, 1, 1},

cifreB[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2}

LongNumber a(12, cifreA), b(12, cifreB), c;

Fraction f1(12, 5), f2(3, 5), f3;

c = add(a, b);

f3 = add(f1, f2);

afisare(c); // 121932631138190824432332

afisare(f3); // 3/1

}

### ***Operatori supraîncărcați***

Vom simplifica scrierea codului:

suma = add(f1, f2);

Ca să putem scrie așa:

suma = f1 + f2; // chiar daca f1,f2 sunt de tipul Fraction

**Varianta 1 - metoda**

Adaugam in clasa metoda:

*class* Fraction {

*// ...*

*public*:

Fraction *operator*+(*const* Fraction& f) {

*// adunam this cu f2, adica a/b cu f.a/f.b*

*return* Fraction(

a \* f.b + b \* f.a,

b \* f.b

);

}

}

**Varianta 2 - functie friend**

Adăugăm în clasa rândul:

*friend const* Fraction& *operator*+(*const* Fraction& f1, *const* Fraction& f2);

Iar global adăugăm imediat după finalul clasei:

Fraction *operator*+(*const* Fraction &f1, *const* Fraction &f2) {

*return* Fraction(

f1.a \* f2.b + f1.b \* f2.a,

f1.b \* f2.b

);

}

Rezultatul? Adăugăm o funcție globală care **are acces la atributele private** ale clasei Fraction. E ca și cum funcția cu antetul menționat are acces VIP :D.

Putem genera cu CLion definiția/corpul funcției:

**Hover pe linia cu antetul -> Alt+Insert -> “Generate definition for function operator+”**

### ***Funcții „friend”***

Deși au efectul de a simplifica accesul unor funcții la atributele clasei, deci scriere de cod mai simplu, le vom folosi doar pentru a supraîncărca operatori precum operatorul de citire și operatorul de scriere:

1. **<<**

Poate fi generat:

**Alt+Insert (inside class) -> “Stream Output Operator”**

Și obținem:

*class* Fraction {

*// ...*

*friend* ostream &*operator*<<(ostream &os, *const* Fraction &fraction);

}

ostream &*operator*<<(ostream &os, *const* Fraction &fraction) {

os << "a: " << fraction.a << " b: " << fraction.b;

*return* os;

}

1. **>>**

Îl puteți adapta, deoarece seamănă mult cu **<<**

istream &*operator*>>(istream &is, Fraction &fraction) {

*// în schimb, înainte de citire vom afișa numele datelor citite:*

cout << "a: ";

is >> fraction.a;

cout << " b: ";

is >> fraction.b;

*return* is;

}

Reminder: în CLion puteți da **Alt+Enter -> Generate Definition for..**

# 

### ***Vectori „STL”***

Pentru prima parte a laboratorului v-am lăsat să folosiți ceea ce știați. De acum încolo învățăm pas cu pas și STL, pentru a simplifica cu mult tot procesul de creare de constructori, getteri și transmitere a datelor.

Pe scurt, un vector stl este un vector de elemente de un anumit tip.

vector este de fapt o clasă implementată în

#include <vector>

Dacă avem structura;

*struct* Grupa{

*int* number; *// numărul grupei*

*int* cntStudents;

};

Vom crea un vector cu grupe cu sintaxa **vector<tip>**

vector<Grupa> grupe; *// la inceput are 0 elemente*

Avantaje și utilizări:

1. Îi putem transmite și returna din funcții mult mai simplu

vector<Grupa> sortat(vector<Grupa> v) { *// primeste o COPIE a vectorului*

*// sortam vectorul*

*// ...*

*// dupa care il putem returna*

*return* v;

}

1. Îl putem copia cu operator=

vector<Grupa> copie = grupe;

1. Putem parcurge elementele mai simplu

*for* (*auto* grupa : grupe) { *// pentru fiecare grupa din vector*

cout << "Grupa " << grupa.number

<< ": " << grupa.cntStudents << " studenti." << '\n';

}

Metoda „clasică”/„din liceu” ar fi:

*for* (*int* i = 0; i < grupa.size(); ++i) {

cout << "Grupa " << grupe[i].number

<< ": " << grupa[i].cntStudents << " studenti." << '\n';

}

1. Vector, denumit si std::vector, are metode utile:

*// adăugare element la final*

grupe.push\_back({143, 28});

grupe.push\_back({152, 30});

*// adresa vectorului*

grupe.begin(); *// = adresa elementului grupe[0]*

grupe.end(); *// = adresa elementului care urmeaza dupa ultimul.*

*// adica grupe.end()-1 va fi adresa ultimului*

*// ștergere elemente*

grupe.erase(grupe.begin());*// sterge primul element*

grupe.push\_back({102, 0})

grupe.push\_back({103, 0})

grupe.push\_back({104, 0})

grupe.erase(grupe.end() - 1);*// sterge ultimul element*

grupe.clear(); *// sterge toate elementele*

*// adaugare elemente*

Grupa nouaGrupa = {157, 100};

*// argumentele pentru .insert sunt "unde", "pe cine"*

*// adica insereaza la inceput noua grupa*

grupe.insert(grupe.begin(), nouaGrupa);

# 

# Exerciții

*\* înseamnă că este antrenament de bază, iar \*\* că primiți bonus maxim pentru ele.*

## 1. College (solved)

Vom implementa o clasă Student care va conține date despre un student: numele, prenumele, grupa, semigrupa și un vector cu cele n note. Pentru a ne însuși informații cât mai bune, vom începe să folosim **std::vector (vector din STL)** pentru a simplifica crearea clasei (vectorii STL pot fi atribuiți simplu: v2 = v).

1. Se cere să folosiți cunoștințele dobândite până acum pentru a implementa clasa Student, dar să supraîncărcați și operatorii de flux (>>, <<).
2. Verificați în main dacă citirea este corectă:

*int* main() {

Student s;

cin >> s;

cout << s;

}

Adică reține valorile introduse.

1. Supraîncărcați și operator+=(int notaNoua) cu efectul de a adaugă încă o notă

Codul final:

#include < /stdc++.h>

*/\**

*\* Daca nu aveti bits/stdc++.h puteti include fiecare header:*

*\*/*

#include <iostream>

*using namespace* std;

*class* Student {

*private*:

string name;

*int* group, semigroup;

vector<*int*> note;

*public*:

Student(*const* string &name, *int* group, *int* semigroup, *const* vector<*int*> &note) : name(name), group(group),

semigroup(semigroup), note(note) {}

Student() : group(0), semigroup(0) {}

*// b)*

*// operatori de flux*

*friend* ostream &*operator*<<(ostream &os, *const* Student &student) {

os << "name: " << student.name

<< " group: " << student.group

<< " semigroup: " << student.semigroup

<< " note: ";

*for* (*auto* nota: student.note) {

os << nota << ' ';

}

*return* os;

}

*friend* istream &*operator*>>(istream &is, Student &student) { *// FARA const, deoarece modificam valoarea*

cout << "name: ";

is >> student.name;

cout << " group: ";

is >> student.group;

cout << " semigroup: ";

is >> student.semigroup;

cout << " note: ";

cout << "nr. Note = ";

*int* n, nota;

is >> n;

*// golim vectorul*

student.note.clear();

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

is >> nota;

student.note.push\_back(nota);

}

*return* is;

}

*// c)*

*// operatorul +=*

*void operator*+=(*int* notaNoua) {

note.push\_back(notaNoua);

}

};

*int* main() {

Student s;

*// cin >> s;*

*// cout << s;*

s = Student(

"Enrique Neville", *// numele*

154, *// grupa*

2, *// semigrupa*

**{**9, 10, 7**}**); *// pentru vector<int> putem folosi direct {note}.*

s += 10;

cout << s << endl;

*/\**

*\* Va afisa:*

*\* name: Enrique Neville group: 154 semigroup: 2 note: 9 10 7 10*

*\*/*

}

## 2. Construcții \*

Se cere implementarea unei clase **ConstructionProduct** care va contine date despre un material de baza pe care l ati achizitionat pentru renovare. Pentru fiecare produs se retin date despre cantitate, unitatea de masurare (cm, L, buc. etc) si pretul de achizitie pe unitate de masura, dar și denumirea produsului.

1. Veti folosi cunostintele dobandite pana acum, dar veti implementa si operatori de flux (>>, <<) pentru a citi n astfel de obiecte de la tastatura, după care veți afișa prețul total. De asemenea, veți avea la dispoziție o funcție globală care afișează o „chitanța”, adică un mesaj de forma:

*Chitanta renovare:*

*1. Cuie (buc) x 10*

*10 x 0.20 = 2.00 RON*

*2. Lavabila (galeata) x 3*

*3 x 37.12 = 111.36 RON*

*3. PAL (buc) x 5*

*5 x 12.00 = 60.00 RON*

*---*

*Total: 173.36 RON*

1. Să implementați și **operator++**, care va crește cu unu cantitatea

Forma prefixata (++produs) are antetul

ConstructionProduct *operator*++()**);**

**Ca să implementați corect: modificați cantitatea, după care returnați o copie a obiectului. Hint: încercați *return \*this* De ce merge? Oricine nu știe mă poate întreba și aflăm ceva nou.**

Forma postfixata (produs++) are antetul

ConstructionProduct *operator*++(*int*);; // ca sa putem supraincarca trebuie sa difere parametrii

Ca să implementați corect, prima creați o copie a valorii inițiale a obiectului

***ConstructionProduct cpy = \*this;***

După aceea, puteți crește cantitatea, iar la final returnați copia creată.

1. Să implementați **operator +=**,care va crește cantitatea cu valoarea primită ca parametru

Antetul:

ConstructionProduct *operator*+=(*int* crestere);

Ca să implementați corect puteți urma sugestiile de la **++**.

## 3. Cyclism \*

Aici aplicați intuiția construită din exemple și de la punctul 3.

Se cere să implementați două clase: BikeTrail și InterestPoint.

BikeTrail va conține date despre o pistă de biciclete: Denumirea pistei, orașul și un vector stl care conține punctele de interes din acel oraș.

InterestPoint reprezintă o zonă de interes a orașului (muzeu, restaurant, monument istoric sau orice alt punct turistic). FIecare punct de interes are date despre numărul de vizitatori, locație (două coordonate **x** și **y**).

Se cere:

1. Implementați clasele cerute: atributele corecte, constructor default, constructor cu toti parametrii.
2. Supraîncărcați operatorii de flux (>>, <<), atât în BikeTrail cât și în InterestPoint.
3. Creați un BikeTrail în main și folosiți >> pentru a citi un exemplu.
   1. Truc: puteți păstra într-un comentariu pe mai multe linii /\*input\*/.

Ca să vă familiarizați, la acest exemplu input-ul va fi de forma:

*/\**

*Traseu#1*

*Constanta*

*5*

*102 0 0*

*404 10 10*

*3 1 1*

*22 3 2*

*1001 5 0*

*\*/*

1. Se cere să rezolvați un exercițiu de algoritmică: se citește o pistă de biciclete cu cel puțin 5 puncte de interes. Se cere să găsiți 3 puncte de interes care formează un traseu de lungime maximă. Ca output veți afișa cele 3 puncte în ordinea aleasă, lungimea totala a traseului ales (distanta de la primul la al doilea plus distanța de la al doilea la al treilea), dar și suma vizitatorilor.
2. Cerința geamănă. Găsiți cele mai populare trei puncte de interes (după numărul de vizitatori) și afișați cel mai scurt traseu care trece prin toate cele 3 puncte.

Hint: Dacă notăm A,B,C cele mai populare puncte, exista 6 trasee posibile: ABC, ACB, BAC, BCA, CAB, CBA. Îl veți afișa doar pe cel mai scurt dintre acestea.

În această cerință aveți noutăți (implementarea operatorilor în cazul claselor compuse)

***Ca bonus simplu (În loc de +0.2p primiți +0.1p), ziua următoare veți primi și exercițiile 4-5 care vor fi de același nivel ca 2 și 3.***

*4. Formular*

*5. Hărți*

## 6. Algoritmica \*\*

Să implementăm un simplu algoritm de umplere (fill) prin care vom răspunde la o întrebare simplă: care este distanța minimă dintre două puncte de pe o matrice și în câte modalități se poate crea un drum minim între acestea două?

Vom considera matricea pătratică de dimensiune n x n. În interiorul acesteia exista obstacole, notate cu valoarea 0, și zone libere notate cu valoarea 1.

Clasele:

Avem o clasă Maze care va conține o matrice de dimensiune **n x n**, dar și două coordonate **start** și **end**,

o clasă Queue care conține un vector de coordonate

și o clasă Coords care conține doi indici x,y din matrice.

1. Implementați clasele cerute, cu atribute, constructori și inclusiv operatorii de flux.

Observații: pentru a refolosi cod, în constructor dar și în >>, în loc sa citiți întreaga matrice de nxn elemente, puteti folosi clasa Queue pentru a citi obstacolele și să completați voi matricea.

1. Pentru a putea folosi clasa Queue ca pe o coada/ vector de coordonate, aceasta va conține căteva metode și operatori:

***Metode:***

*void* clear(); *// elimină toate elementele*

*void* isEmpty(); *// returneaza true doar atunci cand coada nu are elemente*

***Operatori*:**

Coords *operator*[](*int* i); *// returneaza coordonatele din coada, aflate la un anumit indice. Afișează un mesaj corespunzător și returneaza coordonatele -1, -1 dacă nu există un element pe indicele i.*

Queue *operator*+=(Coords *new*); *// adaugă la final noile coordonate și returnează o copie a noii cozi.*

Queue *operator*–-(); *// elimină primul element din coadă*

1. Implementați o metoda care rulează operația de fill cu ajutorul unei cozi de tipul Queue și care afișează matricea „umplută” la final de algoritm.

**Vedeți exemplul de mai jos.**

1. Extindeți punctul c) astfel încât să puteți calcula și număr de drumuri minime **Vedeți exemplul de mai jos.**

Exemplu input și matrice finală:

Maze:

n=5

x: 0

y: 0

x: 4

y: 0

Queue:

5

x: 1

y: 0

x: 1

y: 2

x: 2

y: 2

x: 3

y: 2

x: 4

y: 1

Adica matricea (1=liber, s=start, e=end, 0=obstacol) este:

s 0 1 1 e

1 1 1 1 0

1 0 0 0 1

1 1 1 1 1

1 1 1 1 1

c) Atunci matricea finala este:

s 0 4 5 e(6)

1 2 3 4 0

2 0 0 0 6

3 4 5 6 7

4 5 6 7 8

d) Prin urmare distanța minimă este 6 și avem 2 drumuri de lungime 6.

## ~~7. Aplicație Bolt\*\*~~

~~Țin să menționez că bonusurile nu sunt mereu cele mai bune implementări. Nu m-aș aștepta să funcționeze în acest mod aplicațiile sau ierarhia, iar dacă găsiți idei mai bune este cu atât mai fain/ bine pentru voi :)~~

~~Scopul exercițiului 7 este să exersați vectorii STL, operatorii STL și compunerea de clase, ca o mare recapitulare a tot ce am făcut până acum.~~

~~Vom implementa o clasă Dispatcher care va avea acces la un vector de obiecte de tip Car, dar și o listă de așteptare reținută într-un vector<Ride>.~~