Laboratorul 6

Upcasts, smart pointers and pointer vectors

# Takeaways

* We’ll learn how derived classes can be casted to the base class, just like a float can be casted to an int: (*int*) 3.4
* After that, we’ll learn how to write nicer pointers, using smart pointers. The good and the bad.
* Using these insights we’ll soon be able to create a vector with elements derived from a base class:

*int main() {*

*vector<shared\_ptr<Chair> > allKindsOfChairs =* ***{***

*make\_shared<ArmChair>(*

*ArmChair(200, "brown", "Metal")*

*// valorile de sus se retin in noul "smart pointer"*

*),*

*make\_shared<Sofa>(*

*Sofa(790, "cream", 5, 6.7)*

*),*

*make\_shared<MassageChair>(*

*MassageChair(590, "silver", "Plastic", 3, 2.7)*

*),*

***}****;*

*for (auto chair: allKindsOfChairs) {*

*// datele afisate o sa difere in numar,*

*// in functie de tipul de scaun:*

*cout << \*chair << '\n'; // \* este folosit pentru a lua de la acel pointer*

*}*

*}*

Final Objective for next 2 laboratories and the second project? Being able to implement functions and methods that work for derived classes.

# Exercitii (conțin și exemple)

## Exercițiul 0 - Magazin de scaune (rezolvat)

***Codul și cerințele corespund cu introducerea***

Avem de implementat clase care vor fi utile in inventarierea unui magazin de scaune. Implementăm doar clasele si operator<< (varianta cu metode virtuale).

Din discuțiile cu owner-ul (sau proprietarul, dar în echipele de dezvoltatori discutăm despre **owners**) ați aflat următoarele:

Magazinul comercializează scaune de diverse tipuri, dar toate vor avea un preț și o culoare. În plus,

* Fotoliile vor conține și tipul cotierelor (piele, lemn, metal, plastic);
* Canapelele contin numarul de persoane și lungimea;
* Scaunele de mesaj sunt un tip special de fotoliu (deci si ele conțin tipul cotierelor) despre care reținem numărul de trepte și lungimea cablului de alimentare.

### *Ierarhia de clase:*

#include <bits/stdc++.h>

*// sau daca nu avem bits/*

*// putem include <memory>*

#include <memory> *// shared\_ptr, make\_shared*

*// includem interfata din laboratorul trecut*

#include "IoBase.h"

*using namespace* std;

*class* Chair : *public* IoBase {

*protected*:

*double* price;

string color;

*public*:

Chair(*double* price, *const* string &color) : price(price), color(color) {}

*// Afisarea si citirea cu ajutorul functiilor virtuale va deveni mai utila astazi,*

*// deoarece am mostenit IoBase, in CLion folosim Ctrl+O pentru a implementa read/write*

ostream &write(ostream &os) *const override* {

IoBase::write(os);

cout << "price: " << price;

cout << ", color: " << color;

*return* os;

}

};

*class* ArmChair : *public* Chair {

*protected*:

string armRestMaterial;

*public*:

ArmChair(*double* price, *const* string &color, *const* string &armRestMaterial) : Chair(price, color),

armRestMaterial(armRestMaterial) {}

ostream &write(ostream &os) *const override* {

Chair::write(os);

cout << ", armRestMaterial: " << armRestMaterial;

*return* os;

}

};

*class* Sofa : *public* Chair {

*private*:

*int* maxPeople, length;

*public*:

Sofa(*double* price, *const* string &color, *int* maxPeople, *int* length) : Chair(price, color), maxPeople(maxPeople),

length(length) {}

ostream &write(ostream &os) *const override* {

Chair::write(os);

cout << ", maxPeople: " << maxPeople;

cout << ", length: " << length;

*return* os;

}

};

*class* MassageChair : *public* ArmChair {

*private*:

*int* cntLevels, cableLength;

*public*:

MassageChair(*double* price, *const* string &color, *const* string &armRestMaterial, *int* cntLevels, *int* cableLength)

: ArmChair(price, color, armRestMaterial), cntLevels(cntLevels), cableLength(cableLength) {}

ostream &write(ostream &os) *const override* {

ArmChair::write(os);

cout << ", cntLevels: " << cntLevels;

cout << ", cableLength: " << cableLength;

*return* os;

}

};

### *Crearea și afișarea unui vector de smart pointeri*

*int* main() {

vector<shared\_ptr<Chair> > allKindsOfChairs = **{**

make\_shared<ArmChair>(

ArmChair(200, "brown", "Metal") *// valoarea de aici este pusa in noul "smart pointer"*

),

make\_shared<Sofa>(

Sofa(790, "cream", 5, 6.7)

),

make\_shared<MassageChair>(

MassageChair(590, "silver", "Plastic", 3, 2.7)

),

**}**;

*for* (*auto* chair: allKindsOfChairs) {

*// datele afisate o sa difere in numar,*

*// in functie de tipul de scaun:*

cout << \*chair << '\n'; *// \* este folosit pentru a lua valoarea de la acel pointer*

}

}

## Exercitiul 1 - De la suc la plăcinte

Ați participat la un curs opțional de antreprenoriat. În cadrul lui ai format o echipă și creat un plan de afacere pentru un magazin de sucuri. Ați decis inclusiv locația.  
Pentru a crește popularitatea afacerii în mediul online ai decis să creezi în C++ o aplicație.

În cadrul cursului, ați decis că veți aproviziona magazinul cu fructe naturale.

Pe acestea le veți putea folosi pentru două tipuri de produse: plăcinte și sucuri. Plăcintele le veți crea în cazul fructelor extra, iar sucurile vor fi la rândul lor de trei tipuri: sucuri cu un singur tip de fructe, sucuri mixte și sucuri cu gheata.

Despre fiecare fruct reținem numele și soiul.

Despre fiecare suc reținem cantitatea în litri.

Sucurile cu un singur tip de fructe rețin și fructul folosit.

Spre deosebire de cele simple, pentru sucurile mixte, au fost folosite mai multe fructe.

Sucurile cu gheata au aceleași date ca cele mixte, dar se adaugă cantitatea de gheața.

1. Se cere implementarea ierarhiei de clase (adică a fiecărei clase, cu atributele din cerință, constructor default și parametrizat, citirea și scrierea virtuală);
2. Se cere să implementați o metoda care va putea fi folosi de orice tip de suc pentru a afișa cantitatea de suc (în sensul că sucurile cu gheața conțin atât sucul, cat și gheața);

Adică următorul cod:

vector<shared\_ptr<Juice> > stock = **{**

make\_shared<SimpleJuice>(

SimpleJuice(550, Fruit("Apple", "Granny Smith"))

),

make\_shared<MixedJuice>(

MixedJuice(

560,

**{**

Fruit("Apple", "Granny Smith"),

Fruit("Banana", "Musa balbisiana"),

**}**

)

),

make\_shared<IcedJuice>(

IcedJuice(

560,

**{**

Fruit("Apple", "Granny Smith"),

Fruit("Banana", "Musa balbisiana"),

**}**,

150

)

),

**}**;

*for* (*auto* juice : stock) {

cout << juice->quantity();

}

Va afișa cantitățile:

**550 560 710**

## Exercitiul 2 - Coding Contest

### *Cerințele*

Se cere să implementați clasele necesare pentru o competitie de codare pe echipe care va avea loc în cadrul facultății. Participantii pot fi studenti, elevi și programatori profesionali, iar fiecare echipa va avea un mentor care trebuie să fie programator profesional.

Toți participanții s-au înregistrat cu numele, CNP-ul și limbajele de programare pe care le vor folosi. Elevii vor conține un singur limbaj de programare, iar studenții și programatorii profesionali oricât de multe, deoarece au avut timp să le caute în seara dinaintea înregistrării. Programatorii profesionali au oferit și numele firmelor la care lucrează, iar studenții au completat și anul de studii.

Fiecare echipă a decis de la început un număr maxim de participanți. În afară de membrii săi (vector<shared\_ptr<...> >), fiecare echipă și-a ales un nume și un domeniu aplicat al aplicației, dar și un mentor care întotdeauna va fi programator profesional.

1. Se cere să implementați ierarhia de clase.
2. În echipă veți implementa următoarele metode:

void eliminaMembru(int index);

void adaugaMembru(shared\_ptr<Participant> p);

**void inregistreazaNouMembru(); // Citit precum in exemplu**

void modificaMentor(ProfessionalProgrammer p)

int locuriLibere(); // cate locuri libere au in echipa

### *Exemple utile*

Continuam exemplul de la exercitiul 0, adica magazinul de scaune:

*// citirea de la tastatura a unui nou Chair*

shared\_ptr<Chair> newChair;

*int* type;

*// Pasul 1: alegem tipul*

cout << "Choose chair type (0=simple,1=ArmChair,2=Sofa,3=Massage): ";

cin >> type;

*// Pasul 2: cream pointer de tip potrivit.*

*if* (type == 0) {

newChair = make\_shared<Chair>();

} *else if* (type == 1){

newChair = make\_shared<ArmChair>();

} *else if* (type == 2){

newChair = make\_shared<Sofa>();

} *else if* (type == 3){

newChair = make\_shared<MassageChair>();;

}

*// Pasul 3: citire virtuala*

cin >> \*newChair; *// va citi in functie de ce am implementat in read() in fiecare clasa in parte.*

*// BONUS: Avem nevoie de citirea de scaune in mai multe zone ale codului (e.g.: atat in main, cat si intr-o clasa magazin).*

*// Ce am putea implementa ca sa refolosim usor codul?*

TODO În timpul rămas mai aveți câteva exemple pentru utilitatea metodelor virtuale, STL și auto. Va fi nevoie pentru puțina teorie pe care v-o atașez. Inclusiv variabile statice.

## 

## Exercitiul 3 - Firma Dezvoltare Software (bonus +20p/200)

O firma IT planuieste sa dezvolte un mini soft care va ajuta sa creeze echipe pentru proiectele pe care le au in prezent.

Firma iși dorește sa introducă o listă de angajați și să primească o lista cu toate echipele pe care le pot crea.

Pentru fiecare angajat cunoaștem salariul de baza, funcția, numele și câți ani de experiență deține. În firmă lucrează trei tipuri de angajați: programatori Fullstack, programatori DevOps și manageri. Programatorii se deosebesc prin faptul că pot lucra ore extra, care vor fi plătite cu o cotă de 2.5 ori mai mare față de orele obișnuite.

Prin echipă înțelegem orice grup de programatori și manageri cu condițiile că:

1. Minimul pentru a constitui o echipă este să conțină un manager, un programator DevOps și un programator Fullstack, dar cel puțin unul dintre programatori va trebui să aibă 2 ani de experiență;
2. De asemenea, în cazul echipelor mai mari este nevoie ca cel puțin jumătate dintre programatori să fie **experimentați.**

**Cerințe:**

1. Implementați clasele Employee, FullstackProgrammer, DevOpsProgrammer, Manager.

\* moșteniți inclusiv clasa IoBase ca să vă familiarizați suficient de bine.

1. Implementați o metodă virtuală care va calcula salariul net al fiecărui angajat, prin utilizarea regulilor:
2. Pentru orice manager, salariul net va fi 50% din salariul de bază ~~+ 3% din salariul angajatorului.~~
3. Pentru orice programator, salariul net este de 50% din salariul de bază, plus cntBonusHours \* BONUS\_HOUR\_INCOME

Unde BONUS\_HOUR\_INCOMEeste calculat cu ajutorul formulei:

BONUS\_HOUR\_INCOME = 2.5 \* baseSalary / 160

1. Implementați trei funcție globale care pot fi folosite în rezolvarea cerinței:

Prima funcție primește ca parametru un vector de angajați și va spune dacă se pot forma echipe.

A doua funcție primește ca parametru un vector de angajați și va returna numărul de echipe care se pot forma.

A treia funcție primeste ca parametru un vector de angajați și va returna un vector<vector<Employee> >, adică echipele care pot fi formate

1. Dacă implementați inclusiv d) puteți primi întregul punctaj bonus de +20p/200.

Modificați ce ați lucrat până acuma (puteți păstra o copie cu originalul).

Vom implementa clasa Team care conține un vector de angajați ( vector<shared\_ptr<Employee> > ) și bugetul maxim al companiei.

1. Veți implementa cel puțin clasa și constructorul cu toți parametrii.
2. Veți implementa cele trei funcții de la punctul c), dar de această dată ca pe niște metode.
3. Veți implementa o metodă

vector<Team> sortedTeams();

Care va folosi metodele de mai sus pentru a crea toate echipele, după care le va sorta crescător în funcție de bugetul alocat fiecărei echipe (suma salariilor nete \* 2). **TODO exemplu**

1. **Veți implementa o metodă**

int maximumTeamCount();

Care va calcula numărul maxim de echipe care se pot forma, dar și încadra în bugetul companiei. **TODO exemplu**

*You received more* ***hardcore*** *exercise sets because it’s better sooner than later. It will be our way of improving ourselves for interviews, and even more importantly, developing a solution oriented mindset capable of great attention. Take care!*