***Ministerul Educaţiei,Tineretului si Sportului a Republicii Moldova***

### **Universitatea Tehnică a Moldovei**

***Facultatea Calculatoare, Informatică si Microelectronică***

Lucrarea de laborator Nr.4

la disciplina Programarea orientată pe obiecte

***Tema:* Moştenirea şi compoziţia**

**Varianta 10**

## A efectuat

## studentul grupei TI-161

***Grosu Elena***

***A verificat***

***asist. univ. Botnaru S.***

**Tema:** Supraîncărcarea operatorilor

**Scopul lucrării:**

* studierea moştenirii, avantajele şi dezavantajele;
* studierea compoziţiei;
* studierea regulilor de definire a moştenirii şi compoziţiei;
* studierea formelor de moştenire;
* studierea iniţializatorilor;
* principiul de substituţie;
* moştenirea şi compoziţia – ce să alegem.

**Întrebări de control:**

1. **Prin ce se deosebesc formele de moştenire?**

Mostenirea publica ( metodele publice ale clasei derivate sunt met publice ale clasei care mosteneste , metoda protected sunt mostenite ca protected , iar metodele private nu sunt mostenite)

Mostenirea protejata/protected ( met. public si protected sunt mostenite ca metode protected )

Mostenirea privata/private ( met. public si protected sunt mostenite ca metode private)

1. **Ce reprezintă compoziţia ?**

Compozitia reprezinta relatiile dintre obiecte , atunci cind mostenirea reprezinta relatia dintre clase . Compozitia realizeaza relatia de “a contine”

1. **Care sunt avantajele moştenirii?**

Avantajul mostenirii consta in aceea ca nu trebuie sa declaram de fiecare data clasele care sunt asemantoare, trebuie sa declaram o clasa generala cara va fi mostenita si modificata de altele.

1. **Cum lucrează constructorii la moştenire?**

după parantezele rotunde a constructorului derivat se scrie constructorul de bază.

1. **Ce reprezintă principiul de substituţie?**

obiectul clasei derivate poate fi utilizat în locul obiectului de bază. Animal \*ptr = new Dog;

1. **În ce cazuri se utilizează moştenirea şi în care se utilizează compoziţia?**

Daca noua clasa este un subtip atunci utilizam mostenirea , in caz contrar apare intrbarea :nu este noua clasa un container , atunci utilizam compozitia.

Sarcina:

Varianta 10

а) De creat o ierarhie a claselor *transport* – *transport aerian – elicopter*. Determinaţi fluxul de ieşire şi fluxul de intrare, constructorul de copiere, operatorul de atribuire prin funcţia corespunzătoare a clasei de bază.

b) Definiţi clasa *element chimic*, care conţine informaţia despre denumirea elementului şi proprietăţile ei chimice. Determinaţi clasa *medicamente*, care conţine diferite elemente chimice şi diferite cantităţi. Determinaţi constructorii, funcţiile fluxului de ieşire şi alte funcţii necesare.

***Listingul programului:***

***a)***

#include <iostream>

#include <cstring>

#include <stdio.h>

#include <process.h>

#include <stdlib.h>

#include <conio.h>

#include<ctime>

#include "windows.h"

using namespace std;

class transport

{

protected:

char \*marca;

char \*producator;

int pret;

public:

transport()

{

\*(marca=new char[1])=NULL;\*(producator=new char[1])=NULL;pret=0;

}

transport(char \*m, char \*p, int pr)

{

marca=new char[strlen(m)+1];

strcpy(marca,m);

p=new char[20];

producator=new char[strlen(p)+1];

strcpy(producator,p);

pret=pr;

}

void display()

{

cout <<"Marca transportului:"<<marca<< endl;

cout<<"Producator trans.:"<<producator<<endl;

cout<<"Pret trans.:"<<pret;

cout << endl;

}

public:

char\* getmarca(){

return marca;

}

char\* getproducator(){

return producator;

}

int getpret()

{

return pret;

}

friend ostream &operator<<(ostream &input,transport&);

friend istream &operator >>(istream &output, transport&);

~transport(){delete []marca;delete[]producator;}

transport operator=(transport& s)

{

delete[]marca;

delete[]producator;

marca=new char[strlen(s.marca)];

producator=new char[strlen(s.producator)];

strcpy(marca,s.marca);

strcpy(producator,s.producator);

pret=s.pret;

}

};

istream &operator >> (istream &input, transport &f) {

char x[20];

delete f.marca;

delete f.producator;

cout << "Marca: ";

input>>x;

f.marca=new char[strlen(x)+1];

cout << "Producator: ";

input>>x;

f.producator=new char[strlen(x)+1];

cout << "Pret: ";

input>>f.pret;

return (input);

}

ostream &operator << (ostream &output, transport &p)

{

output <<"Marca transportului:"<<p.marca<< endl;

output<<"Producator trans.:"<<p.producator<<endl;

output<<"Pret trans.:"<<p.pret;

output << endl;

return (output);

}

class transport\_aerian: public transport

{

protected:char \*destinat; //ce transporta

char \*tipul; //civic sau militar

int viteza;

int capacitatea;

public:

friend ostream& operator <<(ostream&,transport\_aerian&);

~transport\_aerian(){delete[]destinat;delete[] tipul;};

friend ostream &operator<<(ostream &input,transport&);

transport\_aerian():transport()

{

\*(destinat=new char[1])=NULL;\*(tipul=new char[1])=NULL;viteza=0;capacitatea=0;}

transport\_aerian(char \*m1, char \*p1, int pr1, char \*h, char \*q,int c, int v)

:transport(m1,p1,pr1)

{

destinat=new char[strlen(h)+1];

strcpy(destinat,h);

tipul=new char[strlen(q)+1];

strcpy(tipul,q);

capacitatea=c;

viteza=v;}

};

ostream& operator<<(ostream& os, transport\_aerian &t)

{

cout <<"Marca transportului:"<<t.marca<< endl;

cout<<"Producator trans.:"<<t.producator<<endl;

cout<<"Pret trans.:"<<t.pret;

cout << endl;

os<<"Avionul este utilizat la transportarea: "<<t.destinat<<endl;

os<<"Tipul avionului: "<<t.tipul<<endl;

os<<"Vitez maxima cu care se poate deplasa:"<<t.viteza<<" km/h"<<endl;

os<<"Capacitatea:"<<t.capacitatea<<" tone"<<endl;

return os;

}

class elicopter:public transport\_aerian

{

// de lupta, de atac, elicopter specializat, elicopter de sprijin de lupta

protected: char \*t;

int echipaj;

int capacitate;

int lungime;

int diametru;

public:

friend ostream& operator <<(ostream&,elicopter&);

~elicopter(){delete[]t;};

elicopter(char \*m1, char \*p1, int pr1, char \*h, char \*q,int c, int v,char \*a, int x1, int x2,int x3, int x4):transport\_aerian(m1,p1,pr1,h,q,c, v)

{

t=new char[strlen(a)+1];

strcpy(destinat,a);

echipaj=x1;

capacitate=x2;

lungime=x3;

diametru=x4;

}

elicopter():transport\_aerian()

{

\*(t=new char[1])=NULL;echipaj=0;capacitate=0;lungime=0;diametru=0;

}

};

ostream& operator<<(ostream& os, elicopter &t)

{

cout <<"Marca transportului:"<<t.marca<< endl;

cout<<"Producator trans.:"<<t.producator<<endl;

cout<<"Pret trans.:"<<t.pret;

cout << endl;

os<<"Avionul este utilizat la transportarea: "<<t.destinat<<endl;

os<<"Tipul avionului: "<<t.tipul<<endl;

os<<"Vitez maxima cu care se poate deplasa:"<<t.viteza<<" km/h"<<endl;

os<<"Capacitatea:"<<t.capacitatea<<" tone"<<endl;

os<<"Tipul avionului: "<<t.t<<endl;

os<<"Echipaj: "<<t.echipaj<<" persoane"<<endl;

os<<"Capacitate:"<<t.capacitate<<endl;

os<<"Lungime: "<<t.lungime<<endl;

os<<"Diametru: "<<t.diametru<<endl;

return os;

}

int main()

{

transport b("azxk","transportrit",12000);

cout<<"Date despre transport";

cout<<b;

cout<<"------------------"<<endl;

transport\_aerian i("airxzip","air",5000,"oameni","militar",300,400);

cout<<"------------------"<<endl;

cout<<"Datele despre transportul aerian";

cout<<i<<endl;

cout<<"------------------"<<endl;

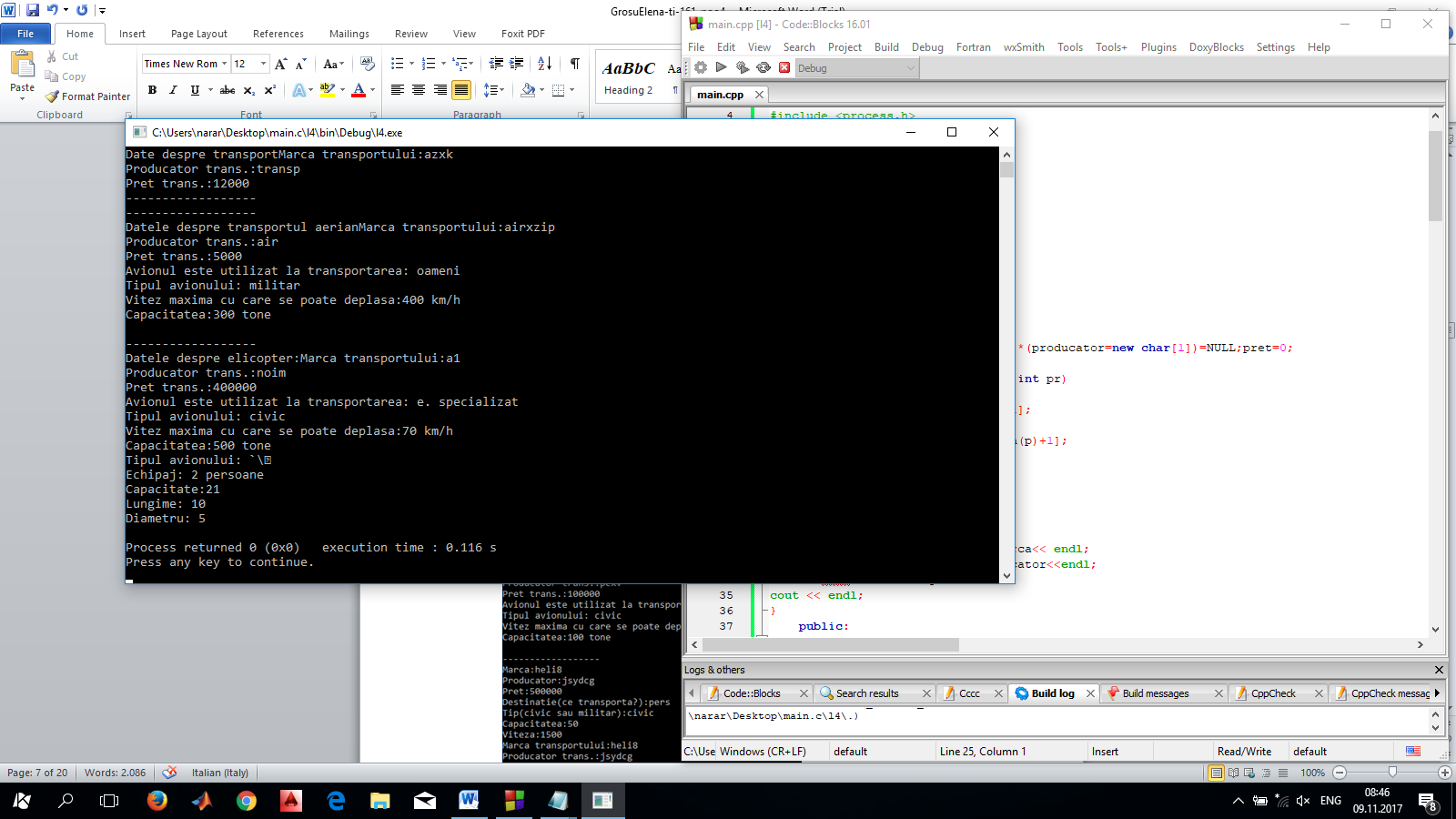
elicopter p("a1","noim",400000,"marfa","civic",500,70,"elicoper specializat",2,21,10,5);

cout<<"Datele despre elicopter:";

cout<<p;

return 0;

}



**b)**

#include <iostream>

#include <cstring>

#include <stdio.h>

#include <process.h>

#include <stdlib.h>

#include <conio.h>

#include<ctime>

#include "windows.h"

using namespace std;

class elementchimic

{

public:

char \*nume;

char \*simbol;

char \*culoare;

int grupa;

int perioada;

int masa;

int raza;

int nrdeoxidare;

elementchimic(char\*n,char\*s,char \*c,int g,int p, int m, int r, int nrox)

{

nume=new char[strlen(n)+1];

strcpy(nume,n);

simbol=new char[strlen(s)+1];

strcpy(simbol,s);

culoare=new char[strlen(c)+1];

strcpy(culoare,c);

grupa=g;

perioada=p;

masa=m;

raza=r;

nrdeoxidare=nrox;

}

public:

void set (char\*,char\*,char\*,int,int,int,int,int);

friend ostream &operator<<(ostream &input,elementchimic&);

friend istream &operator >>(istream &output, elementchimic&);

elementchimic()

{

\*(nume=new char[1])=NULL;

\*(simbol=new char[1])=NULL;

\*(culoare=new char[1])=NULL;

grupa=0;perioada=0;masa=0;raza=0;nrdeoxidare=0;

}

~elementchimic(){delete []nume;delete[]simbol;delete[]culoare;}

elementchimic operator=(elementchimic& s)

{

delete[]nume;

delete[]simbol;

delete[]culoare;

nume=new char[strlen(s.nume)];

simbol=new char[strlen(s.simbol)];

culoare=new char[strlen(s.culoare)];

strcpy(nume,s.nume);

strcpy(simbol,s.simbol);

strcpy(culoare,s.culoare);

grupa=s.grupa;

perioada=s.perioada;

masa=s.masa;

raza=s.raza;

nrdeoxidare=s.nrdeoxidare;

}

};

istream &operator >> (istream &input, elementchimic &f) {

char x[20];

delete f.nume;

delete f.simbol;

delete f.culoare;

cout << "Nume: ";

input>>x;

f.nume=new char[strlen(x)+1];

cout << "Simbol: ";

input>>x;

f.simbol=new char[strlen(x)+1];

cout << "Culoare: ";

input>>x;

f.culoare=new char[strlen(x)+1];

cout<<"Grupa:";

input>>f.grupa; cout<<"Perioada:";

input>>f.perioada; cout<<"Masa:";

input>>f.masa;cout<<"Raza:";

input>>f.raza;cout<<"Nr de oxidare:";

input>>f.nrdeoxidare;

return (input);

}

ostream &operator << (ostream &output, elementchimic &p)

{

output <<"Denumirea elementului:"<<p.nume<< endl;

output<<"Simbol :"<<p.simbol<<endl;

output<<"Culoare :"<<p.culoare;output << endl;

output<<"Grupa:"<<p.grupa;output << endl;

output<<"Perioada:"<<p.perioada;output << endl;

output<<"Masa :"<<p.masa;output << endl;

output<<"Raza:"<<p.raza;output << endl;

output<<"Nr de oxidare :"<<p.nrdeoxidare;

output << endl;

return (output);

}

void elementchimic::set(char \*numek, char \*simbolk,char \*culoarek,int gk,int pk, int mk,int rk, int nrk )

{

delete[]nume;

delete[]simbol;

delete[]culoare;

nume=new char[strlen(numek)];

simbol=new char[strlen(simbolk)];

culoare=new char[strlen(culoarek)];

strcpy(nume,numek);

strcpy(simbol,simbolk);

strcpy(culoare,culoarek);

grupa=gk;

perioada=pk;

masa=mk;

raza=rk;

nrdeoxidare=nrk;

}

class medicament:public elementchimic

{

public:

char \*numemed;

int n;

elementchimic elem[10];

int cantit[10];

friend ostream &operator << (ostream &output, elementchimic &p);

medicament(int a,char \*num1)

{

numemed=num1;

n=a;

cout<<endl<<"Medicamentul "<<numemed<<" contine "<<a<<"elemente chimice";

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

{

cout<<endl<<"------------------"<<endl;

cout<<"Dati datele elementului "<<i+1<<endl;

char num[20], simbol1[20],culoare1[20];

int gr,per,m,r,nrox;

cout<<"Nume: "<<endl;

cin>>num;

cout<<"Simbol: "<<endl;

cin>>simbol1;

cout<<"Culoare: "<<endl;

cin>>culoare1;

cout<<"Grupa: "<<endl;

cin>>gr;

cout<<"Perioada: "<<endl;

cin>>per;

cout<<"Masa: "<<endl;

cin>>m;

cout<<"Raza: "<<endl;

cin>>r;

cout<<"Nr. oxidare: "<<endl;

cin>>nrox;

elementchimic a(num,simbol,culoare1,gr,per,m,r,nrox);

elem[i]=a;

cout<<"Cantitatea elementului dat:"<<endl;

cin>>cantit[i];

}

}

friend ostream &operator << (ostream &output, medicament &p);

};

ostream &operator << (ostream &output, medicament &p)

{

cout<<"Medicamentul contine "<<p.n<<" elemente chimice"<<endl<<endl;

elementchimic d[p.n];

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

{

cout<<"Elementul chimic nr "<<i<<endl;

d[i].nume=p.elem[i].nume;

d[i].simbol=p.elem[i].simbol;

d[i].culoare=p.elem[i].culoare;

d[i].grupa=p.elem[i].grupa;

d[i].perioada=p.elem[i].perioada;

d[i].masa=p.elem[i].masa;

d[i].raza=p.elem[i].raza;

d[i].nrdeoxidare=p.elem[i].nrdeoxidare;

cout<<d[i];

cout<<"Cantitatea de substanta:"<<p.cantit[i]<<" grame"<<endl<<endl;

}

output << endl;

return (output);

}

int main()

{

elementchimic b("aluminiu","Al","sur",10,2,3,45,5),c;

cout<<"Elementul chimic nr.1"<<endl<<"-------------------"<<endl<<endl;

cout<<b;

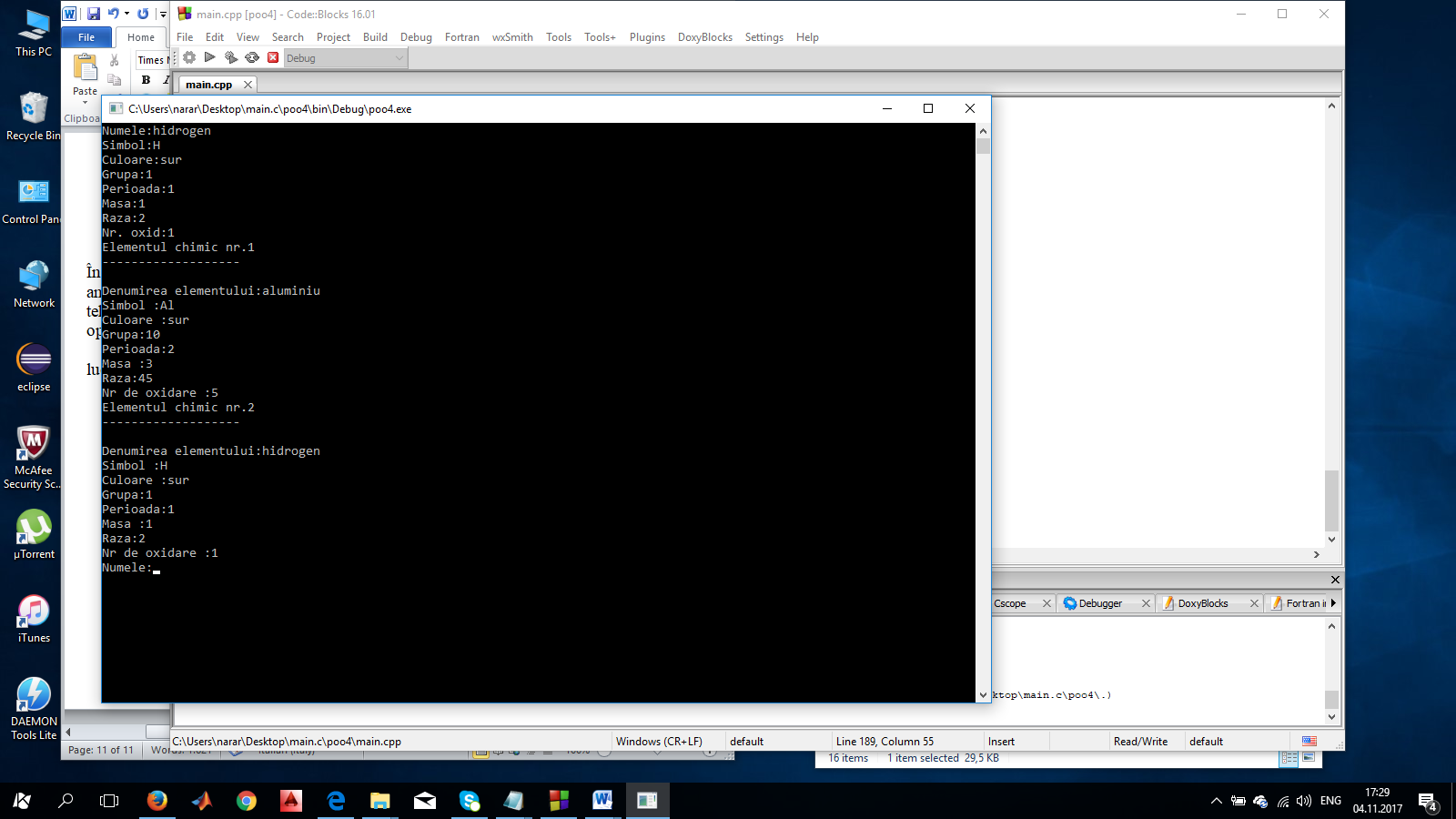
cout<<"Elementul chimic nr.2"<<endl<<"-------------------"<<endl<<endl;

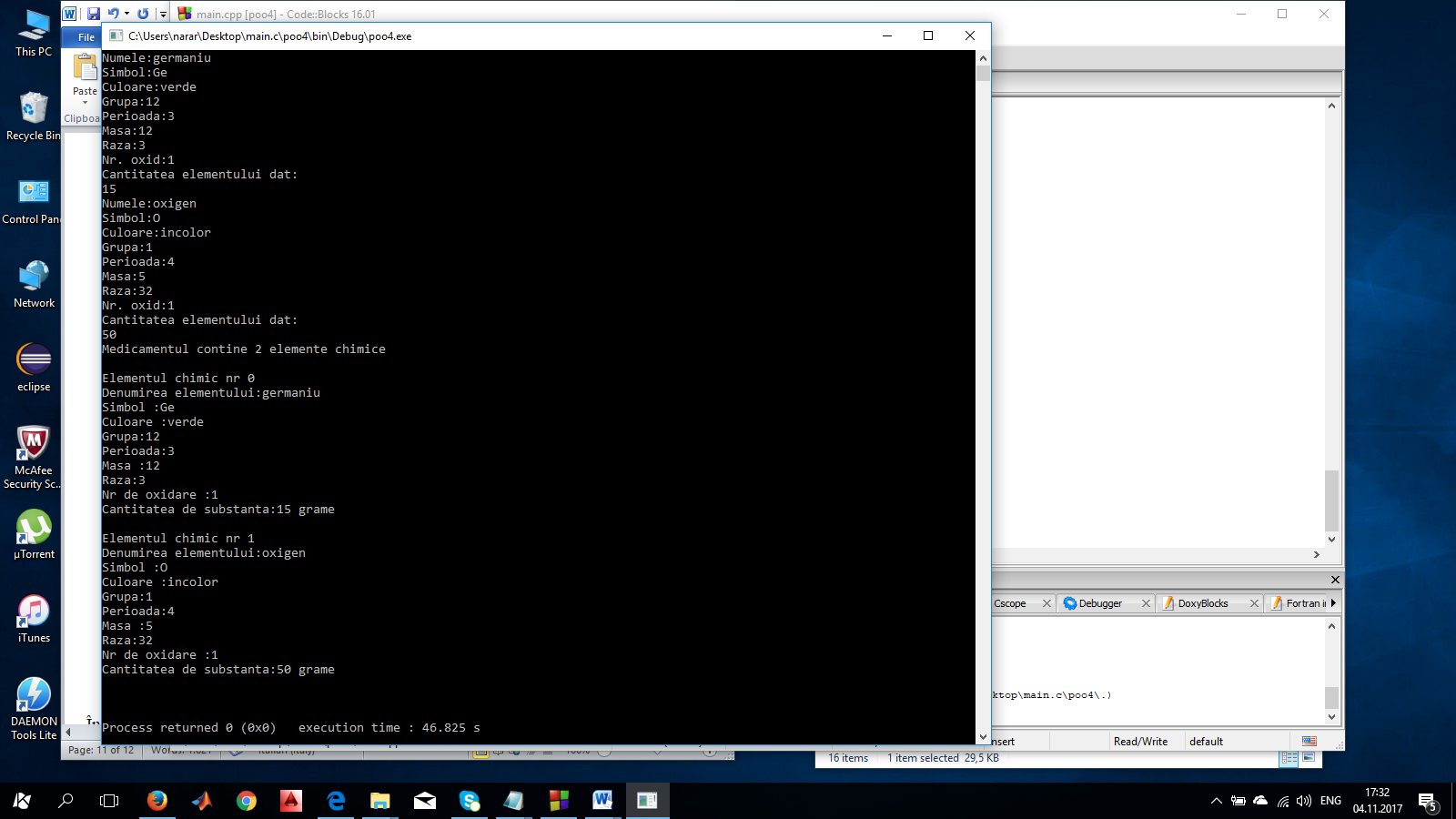
cout<<c;

medicament a(2,"sxcmvf");

cout<<a;

return 0;

}



**Tema pentru acasa**

1. Să se creeze clasa Persoană care are name, profession pointer spre char și age de tip întreg. Să se definească constructorii destructorul. Să se definească o clasele derivate MathTeatcher, Footballer, Unemployed care derivă din clasa de bază
2. Să se creeze clasa Transport care are următoarele metode: fuelAmount, capacity, transportModel. Să se definească constructorul implicit, constructorul cu parametrii, constructorul de copier, operatorul de atribuire și de ieșire(<<). Să se definească clasele derivate: Bus-care conține number\_of\_seats, Car- care conține type\_of\_car(atchback, universal), Truck- care conține destination(long distance, offroad). Pentru toate acestea să se definească Constructorii, să se folosească inițializarea listelor;

**1)**

#include <iostream>

#include <cstring>

#include <stdio.h>

#include <process.h>

#include <stdlib.h>

#include <conio.h>

#include<ctime>

#include "windows.h"

using namespace std;

class Persoana

{

protected:char \*name;

char \*profession;

int age;

Persoana(){\*(name=new char[1])=NULL;\*(profession=new char[1])=NULL;age=0; }

public:Persoana(char \*n, char \*p, int a)

{

name=new char[strlen(n)+1];

profession=new char[strlen(p)+1];

name=n;

profession=p;

age=a;

}

~Persoana(){delete []name;delete[]profession;}

void display()

{

cout<<endl<<"Datele Persoanei:"<<endl;

cout<<"Name: "<<name<<endl;

cout<<"Profession: "<<profession<<endl;

cout<<"Age: "<<age<<endl;

}

};

class Mathteacher:public Persoana

{

protected :int graddidactic;

char \*locdemunca;

int andelucru;

public: Mathteacher(char \*n1, char \*p1, int a1, int g, int an, char \*loc):Persoana(n1,p1,a1)

{

locdemunca=new char[strlen(loc)+1];

locdemunca=loc;

graddidactic=g;

andelucru=an;

}

~Mathteacher()

{

delete []name;delete[]profession;delete []locdemunca;

}

void display()

{

Persoana::display();

cout<<"Mathteacher caracteristici:"<<endl;

cout<<"Graddidactic: "<<graddidactic<<endl;

cout<<"Loc de munca: "<<locdemunca<<endl;

cout<<"Anide lucru: "<<andelucru<<endl;

}

};

class Footballer:public Persoana

{

protected:char \*echipa;

int experinta;

public:Footballer(char \*n1, char \*p1, int a1, char \*ec, int exp):Persoana(n1,p1,a1)

{

echipa=new char [strlen(ec)+1];

echipa=ec;

experinta=exp;

}

~Footballer()

{

delete []name;delete[]profession; delete []echipa;

}

void display()

{

Persoana::display();

cout<<endl<<"Footballer caracteristici:"<<endl;

cout<<"Echipa: "<<echipa<<endl;

cout<<"Experienta:"<<experinta<<"ani"<<endl;

}

};

class Unemployed:public Persoana

{

protected:char \*motiv;

public: Unemployed(char \*n1, char \*p1, int a1, char \*m):Persoana(n1,p1,a1)

{

motiv=new char [strlen(m)+1];

motiv=m;

}

~Unemployed()

{

delete []name;delete[]profession; delete []motiv;

}

void display()

{

Persoana::display();

cout<<endl<<"Unemployed caracteristici:"<<endl;

cout<<"Motiv: "<<motiv<<endl;

}

};

int main()

{

Persoana x("Ion Trofim","coreograf",24);

x.display();

cout<<endl<<endl<<"------------------------";

Mathteacher y("Daniel Frunza","profesor",33,2,7,"UTM") ;

y.display();

cout<<endl<<endl<<"------------------------";

Footballer z("Alexandru Rusu","jucator de footbal",22,"FC Dacia",5);

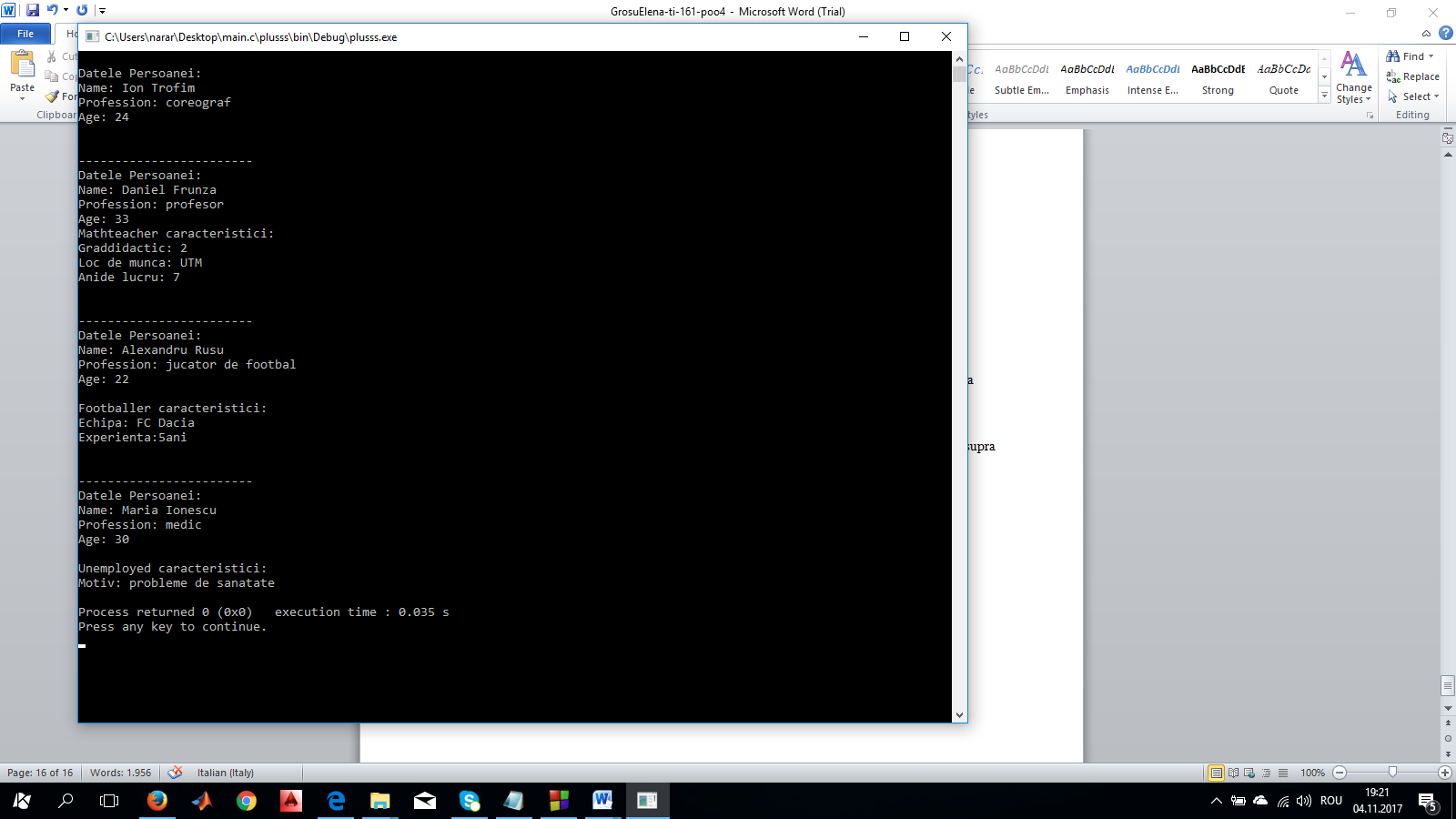
z.display();

cout<<endl<<endl<<"------------------------";

Unemployed q("Maria Ionescu","medic",30,"probleme de sanatate");

q.display();

}



2) #include <iostream>

#include <cstring>

#include <stdio.h>

#include <process.h>

#include <stdlib.h>

#include <conio.h>

#include<ctime>

#include "windows.h"

using namespace std;

class Transport

{

protected:

char \*transportModel;

int capacity;

int fuelamount;

public:

Transport(){\*(transportModel=new char[1])=NULL;capacity=0;fuelamount=0;}

Transport(char \*t,int c, int f)

{

transportModel=new char[strlen(t)+1];

transportModel=t;

capacity=c;

fuelamount=f;

}

Transport(const Transport &d)

{

transportModel=new char[strlen(d.transportModel)+1];

transportModel=d.transportModel;

capacity=d.capacity;

fuelamount=d.fuelamount;

}

friend ostream &operator <<(ostream &input, Transport&);

friend istream &operator >>(istream &output, Transport&);

~Transport(){delete []transportModel;}

};

istream &operator >> (istream &input, Transport &f) {

char x[20];

int cap, fuel;

cout << "Model: ";

input>>f.transportModel;

cout << "Capacitate: ";

input>>f.capacity;

cout << "Fuel amount: ";

input>>f.fuelamount;

return (input);

}

ostream &operator << (ostream &output, Transport &p)

{

output <<"Modelul transportului:"<<p.transportModel<< endl;

output<<"Capaciatatea:"<<p.capacity<<endl;

output<<"Fuel amount:"<<p.fuelamount;

output << endl;

return (output);

}

class Bus:public Transport

{

protected: int nr\_of\_seats;

public: Bus(char \*t1,int c1, int f1, int nr):Transport(t1,c1,f1)

{

nr\_of\_seats=nr;

cout<<endl<<"Constructorul Bus cu parametri a fost apelat";

}

Bus():Transport()

{

nr\_of\_seats=0;

cout<<endl<<"Constructorul implicit Bus a fost apelat";

}

~Bus()

{

delete []transportModel;

}

Bus(const Bus &d)

{

transportModel=new char[strlen(d.transportModel)+1];

transportModel=d.transportModel;

capacity=d.capacity;

fuelamount=d.fuelamount;

nr\_of\_seats=d.nr\_of\_seats;

cout<<endl<<"Constructorul de copiere Bus";

}

};

class Car:public Transport

{

protected: char \*t; //hatchback, universal

public: Car(char \*t1,int c1, int f1, char \*tip):Transport(t1,c1,f1)

{

t=new char [strlen(tip)+1];

t=tip;

cout<<endl<<"Constructorul Car cu parametri a fost apelat";

}

Car():Transport()

{

\*(t=new char[1])=NULL;

cout<<endl<<"Constructorul implicit Car a fost apelat";

}

~Car()

{

delete []transportModel;

delete []t;

}

Car(const Car &d)

{

transportModel=new char[strlen(d.transportModel)+1];

transportModel=d.transportModel;

capacity=d.capacity;

fuelamount=d.fuelamount;

t=new char [strlen(d.t)+1];

t=d.t;

cout<<endl<<"Constructorul de copiere Car";

}

};

class Track:public Transport

{

protected: char \*destination;

public: Track(char \*t,int c, int f, char \*d):Transport(t, c, f)

{

destination=new char [strlen(d)+1];

destination=d;

cout<<endl<<"Constructorul Track cu parametri a fost apelat";

}

Track():Transport()

{

\*(destination=new char[1])=NULL;

cout<<endl<<"Constructorul implicit Track a fost apelat";

}

~Track()

{

delete []transportModel;

delete []destination;

}

Track(const Track &d)

{

transportModel=new char[strlen(d.transportModel)+1];

transportModel=d.transportModel;

capacity=d.capacity;

fuelamount=d.fuelamount;

destination=new char [strlen(d.destination)+1];

destination=d.destination;

cout<<endl<<"Constructorul de copiere Track";

}

};

int main()

{

Transport a("acaaa",15,8),b;

cin>>b;

cout<<a;

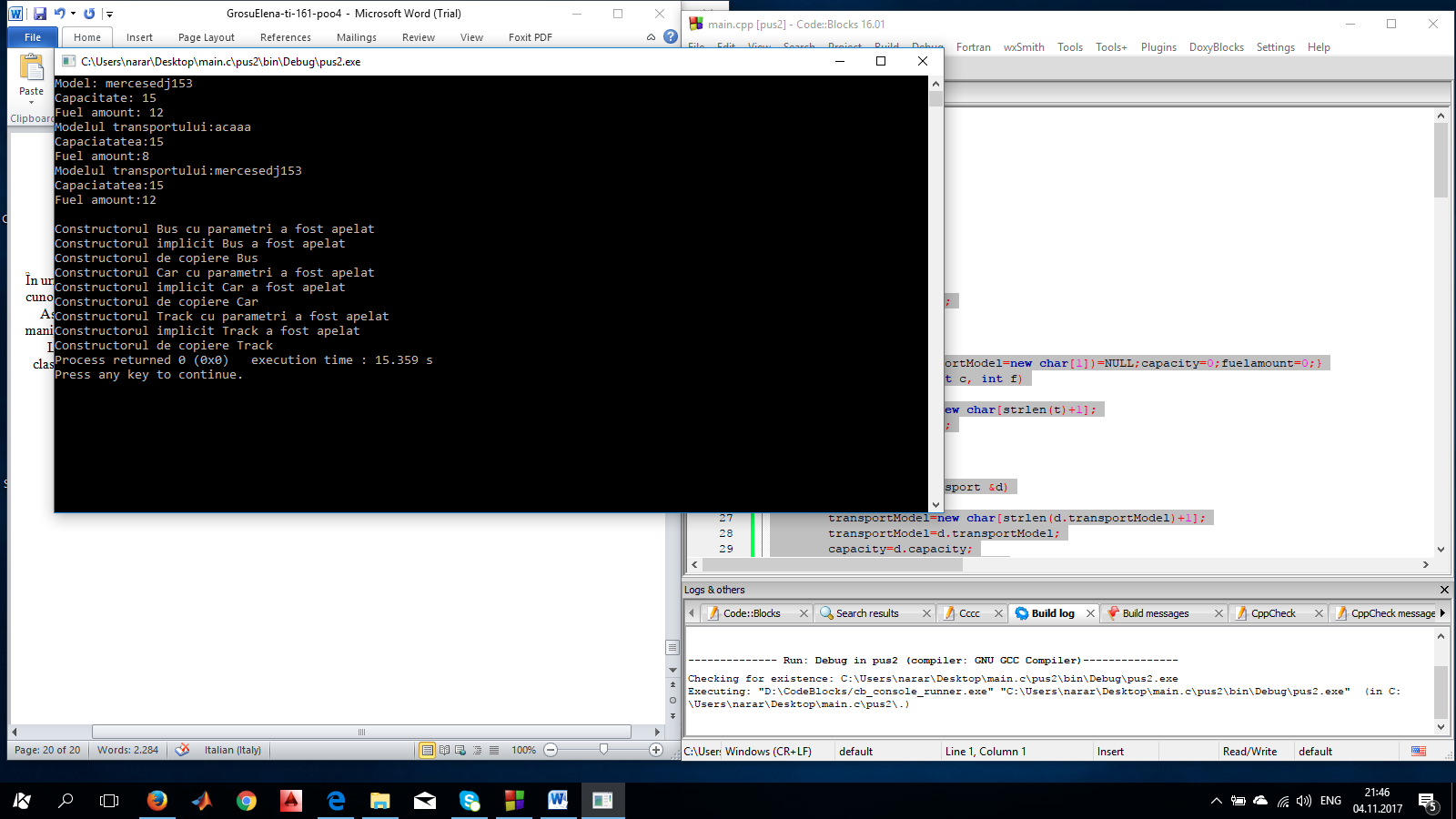
cout<<b;

Bus x("Danke",15,20,20),x1,x4(x);

Car y("helll",10,15,"hatckback"),x2,x5(y);

Track z("exprim",5,8,"long distance"),x3,x6(z);

}



**Concluzie:**

În urma efectuării lucrării de laborator nr.4 „*Moștenirea.*” , am însușit aplicarii in practica a cunostintelor teoretice referitoare la mostenirea multipla.

Astfel se poate judeca despre posibilitatile largi acordate de limbajul C++ referitor la manipularea claselor.

La fel am perfectionat tehnica de programare in limbajul C++ și anume lucrul asupra claselor, constructorilor și destructorilor.