Суть программы: допустим, у нас игра конструктор, по типу Crossout, но создание автомобилей происходит не из деталей, а путем добавлений личных характеристик одного автомобиля, другому (например, автобусу добавляется характеристика грузовика «грузоподъемность»). Но в чем же смысл… я думаю, что можно сделать так: Игрок перемещается по карте, у него есть базовый класс автомобилей «автобус», он проходит миссии и за них открывает дополнительные класса («грузовик», «Танк» и т д) а при прохождении «финальной миссии главы» ему открывается подкласс какой-то (например, грузовой автобус – автобус, в котором можно перевозить не только пассажиров, а ещё и полезные грузы, или танк-автобус, для перевозки людей по захваченный территориям). Вот))).

//Полный код программы:

#include <iostream>

#include <stdio.h>

#include <cstring>

using namespace std;

class Avtomobil { //абстрактный класс

protected:

int skorost, mochnost, rashod\_benzina; char \*nazvanie;

public:

virtual ~Avtomobil() {delete[] nazvanie;}; //деструктор

virtual void print()=0; //читсо виртуальный метод

};

class Gruzovik: virtual public Avtomobil {

protected:

float gruzopodemnost;

public:

Gruzovik() { //конструктор по умолчанию

skorost = 180; mochnost = 300; rashod\_benzina = 11;

gruzopodemnost=14.7;

nazvanie = new char[strlen("Chappy") + 1];

strcpy(nazvanie,"Chappy");

}

Gruzovik (int sk, int mos,int rb ,float gr, char \*n){

skorost = sk; mochnost = mos; rashod\_benzina = rb;

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

strcpy(nazvanie,n);

gruzopodemnost=gr;

}

~Gruzovik() {delete[] nazvanie; }; //а почему бы и не деструктор

Gruzovik& operator =(const Gruzovik &G) { //присваивание

if (&G == this) return \*this;

delete[] nazvanie;

nazvanie = new char[strlen(G.nazvanie+1)];

strcpy(nazvanie, G.nazvanie);

gruzopodemnost=G.gruzopodemnost;

skorost=G.skorost;

mochnost=G.mochnost;

rashod\_benzina=G.rashod\_benzina;

return \*this;

}

void vivod\_gruzovika() { //просто вывод всех полей объекта

printf("%15s %3d %3d %3d",nazvanie,skorost,mochnost,rashod\_benzina);

printf("%4f \n", gruzopodemnost);

}

void print() { //перепопределенный виртуальный метод, так надо

{cout<<"novi gruzovik "<<nazvanie<<endl;}

}

};

class Avtobus: virtual public Avtomobil {

protected:

int colvo\_csi, colvo\_stoi;

public:

Avtobus() { //конструктор по умолчанию

skorost = 180; mochnost = 300; rashod\_benzina = 11;

colvo\_csi=20;

colvo\_stoi=25;

nazvanie = new char[strlen("Chappy") + 1];

strcpy(nazvanie,"Chappy");

}

Avtobus (int sk, int mos,int rb ,int csi,int cst, char \*n){

skorost = sk; mochnost = mos; rashod\_benzina = rb;

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

strcpy(nazvanie,n);

colvo\_csi=csi;

colvo\_stoi=cst;

}

~Avtobus() {delete[] nazvanie; }; //а почемy бы и не деструктор 2

Avtobus& operator =(const Avtobus &O) { //присваивание

if (&O == this) return \*this;

delete[] nazvanie;

nazvanie = new char[strlen(O.nazvanie+1)];

strcpy(nazvanie, O.nazvanie);

skorost=O.skorost;

mochnost=O.mochnost;

rashod\_benzina=O.rashod\_benzina;

colvo\_csi=O.colvo\_csi;

colvo\_stoi=O.colvo\_stoi;

return \*this;

}

void vivod\_avtobusa() { //просто вывод

printf("%15s %3d %3d %3d",nazvanie,skorost,mochnost,rashod\_benzina);

printf("%4d %4d\n",colvo\_csi,colvo\_stoi);

}

void print() { //переопределенный виртуальный метод...

{cout<<"novi avtobus "<<nazvanie<<endl;}

}

};

class Gruzovoi\_Avtobus: public Avtobus, public Gruzovik { //тот самый класс, созданный через множественное наследование

char \*name\_creater; //это имя создателя "щедевра"

public:

Gruzovoi\_Avtobus(): //конструктор по умолчанию

Avtobus() {

gruzopodemnost=14.7;

name\_creater = new char[strlen(("Danila")+1)];

strcpy(name\_creater, "Danila");

}

Gruzovoi\_Avtobus(int sk, int mos,int rb ,int csi,int cst,float gr, char \*n,char \*m): //конструктор не по умолчанию, работает не полностью и не очень

Avtobus(sk,mos,rb,csi,cst,n) {

gruzopodemnost=gr;

name\_creater = new char[strlen((m)+1)];

strcpy(name\_creater, m);

}

~Gruzovoi\_Avtobus() {delete[] name\_creater; }; //деструктор(ну да, опять)

void vivod\_Gruzovoi\_Avtobus() { //просто вывод (ну да, опять)

printf("%15s %10s %3d %3d %3d %4d %4d %4f\n",nazvanie,name\_creater,skorost,mochnost,rashod\_benzina,colvo\_csi,colvo\_stoi,gruzopodemnost);

}

void print() { //переопределенный виртуальный метод, тут даже 2 поля выводит)))

{cout<<"novi gruzovoi avtobus: "<<nazvanie<<" Created:"<<name\_creater<<endl;}

}

Gruzovoi\_Avtobus& operator =(const Gruzovoi\_Avtobus &GA) { //присваивание :/

if (&GA == this) return \*this;

if (GA.gruzopodemnost){

gruzopodemnost= GA.gruzopodemnost;

}

else gruzopodemnost = 0;

if (GA.name\_creater){

name\_creater= GA.name\_creater;

}

else name\_creater = 0;

Avtobus::operator = (GA);

return \*this;

}

};

void func(Avtomobil \*p) { //полиморфная функция, которая вызывает виртуальный метод, потому что я денивый

p->print();

}

int main()

{

Gruzovik topovii(180,300,15,12.7, "12.7 tonn)))"); //объект грузовит

Avtobus mark(180,300,15,25,20, "Mark"); //объект автобус

Avtobus mark3(200,350,14,30,20, "Mark3"); //объект автобус

mark.vivod\_avtobusa(); //вывод полей объекта автобуса

mark3.vivod\_avtobusa(); //вывод полей объекта автобуса

Avtobus mark2=mark; //присваивание объекту автобуса значений объекта автобуса

mark2.vivod\_avtobusa(); //вывод полей объекта автобуса после присваивания...

func(&topovii); //проверка полиморфной функции для грузовика

func(&mark); //проверка полиморфной функции для автобуса

Gruzovoi\_Avtobus \*zevs; //указатель на объект грузового автобуса

zevs=new Gruzovoi\_Avtobus; //память под объект грузового автобуса

func(zevs); //полиморфная функция для объекта грузового автобуса, который адрес

zevs->vivod\_Gruzovoi\_Avtobus(); //вывод полей объекта грузового автобуса, который адрес

Gruzovoi\_Avtobus zevs2(134,248,15,14,17,28.8, "zevs2","Magister"); //попытка на передачу данных пользователем (удалась на половину)

func(&zevs2); //полиморфная функция для объекта грузового автобуса, который не по адресу

zevs2.vivod\_Gruzovoi\_Avtobus(); //вывод полей объекта грузового автобуса, который не по адресу

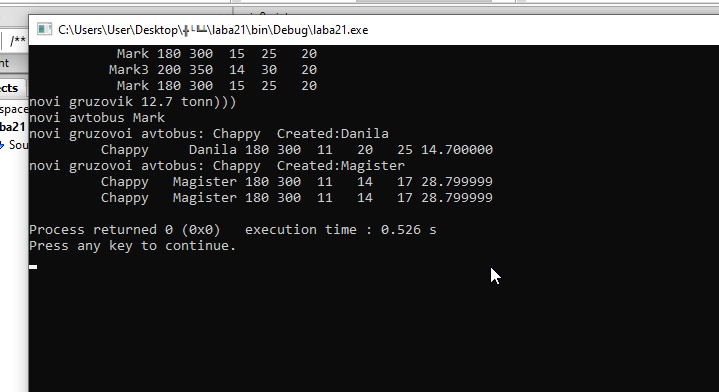
Gruzovoi\_Avtobus zevs3=zevs2; //проверка операции присваивания

zevs3.vivod\_Gruzovoi\_Avtobus(); //вывод полей после проверки, чтобы удебиться

return 0; //return 0

}

Результат выполнения:



А теперь по порядку, что это за результаты.



1. Mark 180 300 15 25 20

2. Mark3 200 350 14 30 20

3. Mark 180 300 15 25 20

4. novi gruzovik 12.7 tonn)))

5. novi avtobus Mark

6. novi gruzovoi avtobus: Chappy Created:Danila

7. Chappy Danila 180 300 11 20 25 14.700000

8. novi gruzovoi avtobus: Chappy Created:Magister

9. Chappy Magister 180 300 11 14 17 28.799999

10. Chappy Magister 180 300 11 14 17 28.799999

Итак, 1 строка вывода, это 157 строка кода, вывод полей объекта автобуса.

2 строка – то же, что и первая, но другой объект.

3 строка – 160 строка кода, вывод объекта, который мы операцией присваивания прировняли к объекту 1 (соответственно, строки вывода одинаковые).

4 строка – проверка полиморфной функции, вызывающей виртуальный метод для грузовика

(161 строка кода)

5 строка – проверка полиморфной функции, вызывающей виртуальный метод для автобуса

(162 строка кода)

6 строка вывода – полиморфная функция, а которую передается указатель на объект класса «грузовой автобус». (166 строка)

7 строка – вывод полей «грузовика автобуса» по адресу.

8 строка – полиморфная функция, но туда уже указателем передается объект (170 строка)

9 строка – вывод полей того объекта (171 строка)

10 строка – вывод полей объекта, который был создан операцией присваивания (173)

**Дополнительный вопрос:** зачем нужны (чем удобны, как используются) абстрактные классы?

Если реализация некоторых методов у подтипов заранее известна и в случае если данная реализация общая для определенной группы типов, то эти общие реализации методов можно поместить в абстрактном классе, который будет базовым для этой группы типов. Методы, реализации которых не являются общими для подтипов - оставляются абстрактными.

Чисто-виртуальная заставляет обязательно писать реализацию в каждом наследнике.  
 Это нужно например для тех случаев, когда на уровне интерфейса в принципе нельзя предугадать какая может быть реализация. Но при этом эта реализация нужна.

Например, для работы некоторой библиотеки нужно получить какие-то важные данные. Но заранее не известно, где и как они будут храниться.  
Может быть просто в файле на диске, а может быть в базе данных.  
Здесь не получится реализовать поведение по умолчанию. Чисто-виртуальный метод не позволит разработчику случайно забыть реализовать важный метод.