חלק יבש שאלה 1 א

```
1 #include <iostream>
2 using namespace std;
3
4 class B {
5 private:
6     int n;
7 public:
8     B(int x) : n(x) {}
9     B operator +(B& b) {
10            return B(n+b.n);
11    }
12    friend ostream& operator <<(ostream &out, const B& b) {
13            out << "B: " << n;
14            return out;
15    }
16    bool operator <(const B& rhs) {
17            return n < rhs.n;
18    }
19 };
20
21 B smaller (const B& b1, const B& b2) {
22    if(b1 < b2)
23            return b1;
24    else
25            return b2;
26 }
27
28 int main() {
29    B b1(1), b2(2), b3(3);
30    const B b4 = b1 + (b2 + b3);
31    cout << smaller(b1,b2) << endl;
32    return 0;
33 }
```

	שגיאה	תיקון
שורה 13	עבור אופרטור >> אגף שמאל של האופרטור הוא מטיפוס ostream, ולכן האופרטור מופעל כמתודה על אובייקט ממחלקה זו ולכן אין משמעות לכתיבת n כי this הוא לא מטיפוס המחלקה שלנו.	שמירה על ההצהרה בתוך המחלקה ומימוש הפונקציה מחוץ למחלקה כאשר במקום n נכתוב כמובן B.n -אנחנו יכולים לגשת לשדה הפנימי כי הגדרנו את החתימה בשורה 13 כfriend.
שורה 22	אופרטור > פועל על ארגומנט משמאל עם קלט מימין. הקלט שלו אכן const כמו בחתימה שלו, אבל הטיפוס שעליו הוא פועל הוא const בניגוד לחתימה של אופרטור > שאינה מחייבת את this לconst.	1. פתרון גורף - הוספת const לחתימה של > : bool operator <(const B& rhs) const הגיוני גם מבחינה לוגית של שימוש באופרטור > 2. פתרון נקודתי לפונקציה smaller – שימוש ב copy ctor דיפולטיבי: if(B(b1) < b2) כעת נוצר איבר מטיפוס B זמני שהוא לא const עם הערך של b1.
שורה 30	מאסוציאטיביות, קודם יחושב (b2+b3) התוצאה של חישוב זה היא B **זמני** שנוצר מהcopy ctor בעת החזרה מהפוקנציה. הB הזמני הזה כעת מועבר כארגומנט לפעולה + שהופעלה על האובייקט b1. הארגומנט מוגדר בפונקציה + להיות מסוג &. כפי שלמדנו - & לא יכול לקבל קבועים או משתנים זמניים.	אנו מעוניינים לייצר אופציה לאסוציאטיביות כזאת, לכן כדי להשאיר את הסוגריים עלינו לאפשר לB הזמני להיכנס כארגומנט לפונקציה +. לכן נשנה את החתימה שלה: B operator +(const B& b) זה מעיד לקומפיילר שהארגומנט שאנחנו מעבירים לאופרטור+ הוא "נטו קלט" ולכן משתנים זמניים יתכנו כארגומנטים אפשריים בפונקציה.

חלק יבש שאלה 1 ב

```
#include <iostream>
using namespace std;

class A {
public:
	A() {}
	A(const A& a) { cout << "A copy ctor" << endl; }
	virtual ~A() { cout << "A dtor" << endl; }
	virtual void type() const { cout << "This is A" << endl; }
};
class B: public A {
public:
	virtual ~B() { cout << "B dtor" << endl; }
	void type() const override { cout << "This is B" << endl; }
};
A f(A a) {
	a.type();
	return a;
}
const A& g(const A& a) {
	a.type();
	return a;
}
int main()
{
	A* pa = new B();
	cout << "applying function f:" << endl;
	f(*pa).type();
	cout << "applying function g:" << endl;
	g(*pa).type();
	delete pa;
	return 0;
}
```

שורה מתוכנית main	קריאות	פלט
A* pa = new B;()	קריאה ל**ctor** דיפולטיבי של **מחלקה B**, היות והוא דיפולטיבי אין בו רשימת אתחול לגבי מחלקת האב – לכן נקרא **ctor** ריק של האב – של **מחלקת A**. המופע נוצר בheap והכתובת נשמרת במצביע מסוג האב.	-
cout << "applying function f:" << endl;	-	applying function f:
f(*pa).type();	קודם נקראת הפונקציה f – היא מקבלת איבר by value A לכן נקרא **copy ctor של מחלקה A** (מעין slicing על pa). לכן מופעל **type של A** . ביציאה מהפונקציה ההחזרה היא by value ולכן מופעל **copy ctor של מחלקה A**, ביציאה מהפונקציה מופעל **הורס של A** על המופע של A בפונקציה f. על ההחזרה של f (ערך A) **מופעל type של A.** בתום השורה הנ"ל A ערך ההחזרה הזמני נמחק לכן נקרא שוב **הורס של A**	A copy ctor This is A A copy ctor A dtor This is A A dtor
cout << "applying function g:" << endl;	-	applying function g:
g(*pa).type();	הפונקציה g מקבלת איבר by reference A לכן מועבר הערך עליו מצביע pa (שהוא מסוג B). לכן מופעל **type של B** . ערך ההחזרה הוא כreference לכן הtype החיצוני גם מופעל על B ולכן מופעל שוב **type של B** .	This is B This is B
delete pa;	קריאה להורס היא מהבן לאב לכן נקרא **dtor של B** ואז **dtor של A**	B dtor A dtor

חלק יבש שאלה 2

```
#include <vector>
class Road {
public:
	double length(); // km
	int speed(); // km per hour
};

class Car {
public:
	virtual double getFuelConsumption(const int speed) const = 0;
	virtual ~Car() {}
};

double getPetrol(std::vector<Road> roads, const Car& car)
{
	double petrol_sum = 0;
	for (std::vector<Road>::iterator it = roads.begin(); it != roads.end(); ++it)
	{
		int cur_road_speed = it->speed();
		double cur_road_length = it->length();
		double curr_fuel_consumption = car.getFuelConsumption(cur_road_speed);
		sum += (cur_road_length / curr_fuel_consumption);
	}
	return petrol_sum;
}
```

חלק יבש שאלה 2