חלק יבש 1:

א. פירוט שגיאות (לאחריו תיקונים)

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

class B {
private:
   int n;
public:
   B(int x) : n(x) {}
   B operator +(B& b) {
      return B(n+b.n);
   }
   friend ostream& operator <<(ostream &out, const B& b) {
      out << "B: " << n; //operator << is a friend function that doesn't get *this and we
must write b.n
      return out;
   }
   bool operator <(const B& rhs) const {
      return n < rhs.n;
   }
};

B smaller (const B& b1, const B& b2) {
   if(b1 < b2)                    //operator works only when b1 isn't const. changed
operator < to const.
      return b1;
   else
      return b2;
}


int main() {
   B b1(1), b2(2), b3(3);
   const B b4 = b1 + (b2 + b3); // operator () is undefined for class B and isn't needed
in this case
   cout << smaller(b1,b2) << endl;
   return 0;
}
```

קוד מתוקן:

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

class B {
private:
   int n;
public:
   B(int x) : n(x) {}
   B operator +(B& b) {
      return B(n+b.n);
   }
   friend ostream& operator <<(ostream &out, const B& b) {
      out << "B: " << b.n;
   }
   bool operator <(const B& rhs) const {
      return n < rhs.n;
   }
   B smaller (const B& b1, const B& b2);
};

B smaller (const B& b1, const B& b2) {
   if(b1 < b2)
      return b1;
   else
      return b2;
}



int main() {
   B b1(1), b2(2), b3(3);
   const B b4 = b1 + b2 + b3;
   cout << smaller(b1,b2) << endl;
         return 0;
}
```

ב.

applying function f: --- Standard output stream

A copy ctor --- f(*pa) passing a copy of *pa, the function is excepting 'A' type
and copying an A type value. the COPY is 'sliced' from 'B' to 'A' type

This is A --- inside the function f the function type() of the copy we made is being called
as mentioned before the copy is an A type

A copy ctor --- a copy ctor is being called again when returning 'a' by value
the copy is A type

This is A -- after returning a copy of *pa from function f we call type()

A dtor - dtor for the first copy we made when we copied to f

A dtor - dtor for the second copy we made when we returned a value from f by value

applying function g: --- Standard output stream

This is B --- g gets *pa by reference so 'a' is exactly *pa and stays 'B' type
B is subclass of A so 'g(const A& a)' can get A& and B& (not if it's by value)
(this is the outcome from type() inside the function g)

This is B --- g returns again 'a' by reference so returning *pa again so stays 'B' type
B is subclass of A so const 'A& g(const A& a)' can return A& and B& (not if it's by value)
(this is the outcome from type() outside the function g)

B dtor --- *pa which is 'B' type and didn't change , so first this dtor is being called
A dtor --- when B dtor finished he is calling to A dtor

חלק יבש 2:

```
#include <iostream>
#include <vector>


class Road {
public:
   double length();
   int speed();
};

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

double getPetrol(std::vector<Road> roads, const Car& car) {
   double consumption;
   for (std::vector<Road>::iterator it = roads.begin() ; it != roads.end() ; ++it) {
      consumption += car.getFuelConsumption(it->speed());
   }
   return consumption;
}
```