#include <iostream>

using namespace std;

class B {

 private:

    int n;

 public:

    B(int x) : n(x) {}

    B operator +(B b) {

        b = n+b.n;

return b;

 }

 friend ostream& operator <<(ostream &out, const B& b) {

    out << "B: " << 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)

        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;

 }

**שאלה 1, סעיף א:**

1.(שורה 13)

ניסיון לגשת ל this מפונ' friend.

במקום n יש לכתוב b.n .

2.(שורה 22)

ניסיון להפעיל operator< כאשר אחד האופרנדים אינו מתאים להצהרת הפונקציה.

תיקון : הוספת const בהצהרת הפונ'.

3.(שורה 30)

operator + לא תומך בשרשור.

תיקון : שינוי הצהרת וגוף הפונ' – הורדנו את קבלת הפרמטר כרפרנס, כך שאנו לא פוגעים באף אחד מהאיברים b1,b2,b3.

#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;

}

**שאלה 1 סעיף ב:**

הסבר על שורות הקוד שבתוך main:

|  |  |
| --- | --- |
| **השורה בקוד:** | **הסבר:** |
| A\* pa = new B(); | creates a new pointer with type A\* to a B object – no output. |
| cout << "applying function f:" << endl; | prints to the screen through cout 1. |
| f(\*pa).type(); | going inside f function we insert a copy c’tor and prints 2. Then there is a call to the method a.type() that prints 3.  After that we go to the return and we returning back another copy of a, that calls the copy c’tor and prints 4.  Then we call the method .type again from the main so it prints 5.  So we had 2 copies of a, so the destructor needs to be called twice – once for the first copy that the function received 6., and once for the object that was returned out 7. |
| cout << "applying function g:" << endl; | prints to the screen through cout 8. |
| g(\*pa).type()*;* | g receives an object by reference thus there is no calls to the copy constructor nor in the reception nor on the return. So, first we go through the method inside g that prints 9. Then we have the method .type inside the main function that causes the print 10. Now we are going out of the scope of g function so the destructor of B is called and causes the print 11., and then because of B is derived from A the destructor of A is called afterwards so we get the output 12. |
| delete pa; | Removes the pointer from the heap. |
| return 0; | Go out of main. |

**להלן הפלט של התוכנית:**

1. applying function f:
2. A copy ctor
3. This is A
4. A copy ctor
5. This is A
6. A dtor
7. A dtor
8. applying function g:
9. This is B
10. This is B
11. B dtor
12. A dtor