

**Przyjąć, że udostępniona jest przestrzeń nazw std****Zadanie 1 (2pkt)**

W aplikacji przedmiot (Item) oraz książka (Book) to klasy z niebanalnymi składowymi, agregują obiekty Image. Implementacja nie jest poprawna, są problemy w sytuacji, gdy konstruktor obiektu Image zgłasza wyjątek. Nie potrzebujemy konstruktora kopiującego, ani operatora przypisania.

Popraw kod.

```
class Image { }; //implementacja nieistotna

class Item {
public:
    Item(const std::string& n) : name(n) {
        image = new Image();
    }
    Item(Item&& src) : name(src.name) {
        image = src.image;
        src.image = nullptr;
    }
    virtual ~Item() {
        delete image;
    }
private:
    Item(const Item&) = delete;
    Item& operator=(const Item&) = delete;
    Item& operator=(Item&&) = delete;

    std::string name;
    Image* image;
};

class Book : public Item {
public:
    Book(const std::string& n) : Item(n) {
        front = new Image();
        back = new Image();
    }

    ~Book() {
        delete back;
        delete front;
    }
private:
    Image* front;
    Image* back;
};
```

**Uwagi do prowadzącego (R.Nowak):**

## Zadanie 2 (2pkt)

Drzewo binarne liczb całkowitych (klasa Tree i Node) działa niepoprawnie, przykład przedstawiono obok. Popraw kod drzewa i/lub węzła.

```
Tree t;  
t.add(1);  
t.add(2);  
t.add(3);
```

```
class Tree {  
    struct Node;  
    using PNode = std::shared_ptr<Node>;  
    using PWNode = std::weak_ptr<Node>;  
  
    struct Node {  
        Node(int v) : value_(v) {}  
        int value_;  
  
        void add(PNode child) {  
            if(child->value_ < value_) {  
                //modify left sub-tree  
                if( left_ )  
                    left_->add(child);  
                else {  
                    left_ = child;  
                    left_->parent_ = me_.lock();  
                }  
            } else {  
                //modify right sub-tree  
                if( right_ )  
                    right_->add(child);  
                else {  
                    right_ = child;  
                    right_->parent_ = me_.lock();  
                }  
            }  
        }  
        PWNode me_;  
        PNode parent_;  
        PNode left_;  
        PNode right_;  
    };  
  
public:  
    Tree() {}  
    ~Tree() {}  
    void add(int value) {  
        PNode node( new Node(value) );  
        node->me_ = node;  
        if( head_ )  
            head_->add(node);  
        else  
            head_ = node;  
    }  
private:  
    PNode head_;  
};
```

Uwagi do prowadzącego (R.Nowak):

**Zadanie 3 (2pkt)**

Faktem jest, że w języku Rust stworzenie dwukierunkowej listy jest nietrywialnym zadaniem. Wyjaśnij dlaczego może tak być, opierając się o wiedzę z wykładu, m.in. o reguły borrow checkera.

**Uwagi do prowadzącego (Ł. Neumanna):**

**Zadanie 4 (2pkt)**

Przy użyciu szablonów zaimplementuj funkcję, która (w sposób rekursywny) liczy silnię. Wartość, dla której liczona jest silnia przyjmowana jest jako parametr szablonu będący liczbą bez znaku. Pamiętaj o warunku stopu.

**Zadanie 5 (2pkt)**

Korzystając z algorytmów biblioteki standardowej skonstruuj algorytm, który dla danego zakresu łańcuchów tekstowych (stringów) zwraca liczbę łańcuchów, które składają się wyłącznie z wielkich liter. Załóż, że istnieje funkcja `std::isupper()`, która sprawdza, czy dany znak jest wielką literą.

```
template<typename Iter>
auto count_upper_strings(Iter from, Iter to) {
```

**Uwagi do prowadzącego (W. Wysota):**