Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **26L-1** 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std i są dodane niezbędne nagłówki z biblioteki standardowej

Zadanie 1 (2pkt) Jeżeli trzeba, to popraw kod, aby lista działała poprawnie.

```
class List {
   using PNode = std::shared_ptr<Node>; using PWNode = std::weak_ptr<Node>;
   struct Node {
      Node(int v) : value_(v) {}
      ~Node() {}
      int value_;
      PWNode next_; //element nastepny
      PWNode prev_; //element poprzedni
   };
public:
   List() {}
   ~List() {}
   void push(int val) {
      PNode node = make_shared<Node>(val);
      if( head_ ) { //not empty
         node->prev_ = head_->prev_;
         node->next_ = head_;
         head_->prev_->next_ = node;
         head_->prev_ = node;
      }
      else { //empty
         head_ = node;
         head_->prev_ = head_; head_->next_ = head_;
      }
   }
private:
   PNode head_;
};
```

Zadanie 2 (2pkt)

Obiekty typu `Point3D` reprezentują punkty, wykorzystują wektor liczb. Popraw kod, aby uniknąć niepotrzebnego kopiowania. Przykład użycia (gdzie są niepotrzebne kopie) obok.

```
int main() {
   const Point3D p1(1.0,1.0,1.0);
   Point3D p2 = p1.center(p1.center(Point3D(2.0,2.0,2.0)));
   cout << (p2 == p1) << endl;
   return 0;
}
```

```
class Point3D {
   vector<double> v_;
   static const int X = 0; static const int Y = 1; static const int Z = 2;
public:
   Point3D(double x = 0.0, double y = 0.0, double z = 0.0) {
      v_.push_back(x); v_.push_back(y); v_.push_back(z);
   }
   Point3D(const Point3D& p) : v_(p.v_) { }
   Point3D(Point3D&& p) { v_ = std::move(p.v_); }
   ~Point3D() {}
   //przesuwa w kierunku wskazanego punktu
   Point3D& moveToCenter(const Point3D& p) {
      v_[X] += p.v_[X]; v_[X] /= 2.0;
      v_[Y] += p.v_[Y]; v_[Y] /= 2.0;
      v_[Z] += p.v_[Z]; v_[Z] /= 2.0;
      return *this;
   }
   //zwraca punkt centralny
   Point3D center(const Point3D& p) const {
      Point3D out(*this);
      out.moveToCenter(p);
      return out;
   }
   friend bool operator==(Point3D a, Point3D b);
};
bool operator==(Point3D a, Point3D b) {
   return a.v_[0] == b.v_[0] && a.v_[1] == b.v_[1] && a.v_[2] == b.v_[2];
}
```

Zadanie 3 (2pkt)
W jakim celu do języka Rust został wprowadzony mechanizm lifetime? Odpowiedź uzasadnij.

Zadanie 4 (2pkt)

Poniższy kod skutkuje następującym błędem:

```
error[E0277]: `RefCell<i32>` cannot be shared between threads safely
  --> counters.rs:9:36
   |
 9 | let handle = thread::spawn(move || {
   | ___________________________-------------_^
   | | |
   | | required by a bound introduced by this call
10 | | let mut num = counter.borrow_mut();
11 | |
12 | | *num += 1;
13 | | });
   | |_________^ `RefCell<i32>` cannot be shared between threads safely
   |
   = help: the trait `Sync` is not implemented for `RefCell<i32>`
```

Wyjaśnij na czym polega ten błąd i zaproponuj rozwiązanie problemu.

```
use std::{cell::RefCell, sync::Arc, thread};
fn main() {
   let counter = Arc::new(RefCell::new(0));
   let mut handles = vec![];
   for _ in 0..10 {
      let counter = Arc::clone(&counter);
      let handle = thread::spawn(move || {
         let mut num = counter.borrow_mut();

         *num += 1;
      });
      handles.push(handle);
   }
   for handle in handles {
      handle.join().unwrap();
   }
   println!("Result: {}", *counter.borrow());
}
```

Zadanie 5 (2pkt)

Obiekty typu `SafeOut` bezpiecznie zapisują napis. Zmień strukturę, aby unikać powielania kodu. Dla przypomnienia: metoda `good` jest z klasy `std::ios_base`, a to klasa bazowa dla `std::ostream`.

```
class SafeOut {
public:
   SafeOut(std::ostream& out) : out_(out) {}
   virtual ~SafeOut() = default;
   virtual void store(const std::string& s) = 0;
private:
   std::ostream& out_;
};
class FileOut : public SafeOut {
public:
   FileOut(const std::string& filename) : SafeOut(file_), file_(filename) { }
   ~FileOut() override { file_.close(); }
   void store(const std::string& s) override {
      if(!s.empty() && file_.good() ) {
         file_ << s << std::flush;
      }
   }
private:
   std::ofstream file_;
};
class StdOut : public SafeOut {
public:
   StdOut() : SafeOut(std::cout) {}
   void store(const std::string& s) override {
      if(!s.empty() && cout.good() ) {
         std::cout << s << std::flush;
      }
   }
};
```

Uwagi do prowadzących lub notatki