Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **23Z** 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (2pkt)

Węzeł drzewa binarnego, oprócz uchwytów na lewy i pawy węzeł potomny przechowuje uchwyt na rodzica i na węzeł sąsiedni (tzn. lewy na prawy, a prawy na lewy). Drzewo działa niepoprawnie, przykład przedstawiono obok. Popraw kod drzewa i/lub węzła.

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

```
class Tree {
   using PNode = std::shared_ptr<Node>;
   using WNode = std::weak_ptr<Node>;
   using UNode = std::unique_ptr<Node>;
   struct Node {
      Node(int v) : value_(v) {}
      ~Node() {}
      void add(PNode child) {
         if(child->value_ < value_) {//modify left sub-tree
            if( left_ )
               left_->add(child);
            else {
               left_ = child;
               left_->parent_ = me_.lock();
               if(right_) {
                  left_->sibling_ = right_; right_->sibling_ = left_;
               }
            }
         } else { //modify right sub-tree
            if( right_ )
               right_->add(child);
            else {
               right_ = child;
               right_->parent_ = me_.lock();
               if(left_) {
                  right_->sibling_ = left_; left_->sibling_ = right_;
               }
            }
         }
      }
      int value_;
      PWNode me_; //wskaznik na ten sam wezel
      PNode parent_; //rodzic
      PNode left_; //element nastepny
      PNode right_; //element nasteny
      PNode sibling_; //element sasiedni
   };
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 2 (2pkt)

W aplikacji graf jest reprezentowany przez obiekty Node i Edge, tworzone za pomocą funkcji newNode i newEdge. Obserwujemy, że duże grafy (50 miliony wierzchołków, 100 milionów krawędzi) nie mieści się w pamięci 4GB RAM. Opisz, jak zmieniłbyś kod, aby to naprawić.

```
using PNode = std::shared_ptr<Node>;
using PEdge = std::shared_ptr<Edge>;
struct Node {
  Node(int v) : value_(v) {}
  ~Node() {}
  int value_;
}
struct Edge {
  Edge(PNode s, PNode d) : source_(s), destination_(d) {}
  PNode source_;
  PNode destination_;
}
PNode newNode(int value) {
  return PNode(new Node(value) );
}
PEdge newEdge(PNode source, PNode destination) {
  return PEdge(new Edge(source, destination) );
}
```

Zadanie 3 (2pkt)

Pokazane 3 klasy mają powielony kod. Postaraj się to zminimalizować tworząc kasę bazową Finder.

```
class FinderName {
public:
   FinderName(string name) : name_(name) {}
   bool find(const std::vector<string>& names) const {
     for(string n : names) {
       if(n == name_) return true;
     }
     return false;
   }
private:
   string name_;
};
class FinderShorter {
public:
   FinderShorter(int size) : size_(size) {}
   bool find(const std::vector<string>& names) const {
     for(string n : names) {
       if(n.size() < size_) return true;
     }
     return false;
   }
private:
   int size_;
};
class FinderLength {
public:
   FinderLength(int size) : size_(size) {}
   bool find(const std::vector<string>& names) const {
     for(string n : names) {
       if(n.size() == size_) return true;
     }
     return false;
   }
private:
   int size_;
};
```

Zadanie 4 (2pkt)

Podany kod skutkuje następującym błędem kompilacji:

```
// error[E0599]: no method named `feed` found for mutable reference `&mut T` in the current scope
// --> animals.rs:25:25
// |
// 25 | let result = animal.feed();
// | ^^^^ method not found in `&mut T`
// |
// = help: items from traits can only be used if the type parameter is bounded by the trait
```

Napisz na czym polega problem i jak można go naprawić?

```
trait Animal {
   fn make_sound(&self) -> String;
   fn feed(&mut self) -> Result<String, String>;
}
struct Dog {
   is_hungry: bool,
}
impl Animal for Dog {
   fn make_sound(&self) -> String {
      String::from("Woof!")
   }

   fn feed(&mut self) -> Result<String, String> {
      if self.is_hungry {
         Ok(self.make_sound())
      } else {
         Err(String::from("The␣dog␣is␣not␣hungry!"))
      }
   }
}
fn feed_the_animal<T>(animal: &mut T) {
   let result = animal.feed();
   match result {
      Ok(sound) => println!("The␣animal␣says:␣{sound}"),
      Err(reason) => println!("Animal␣was␣not␣fed:␣{reason}")
   }
}
fn main() {
   let mut bubba = Dog {is_hungry: false};
   feed_the_animal(&mut bubba);
}
```

Zadanie 5 (2pkt)

Porównaj ze sobą mechanizm wyjątków udostępniany w C++ instrukcjami: try, catch, throw z obsługą błędów opartą o enumerację znaną z Rusta. Wskaż wady i zalety każdego z podejść.

Uwagi do prowadzącego (Ł. Neumanna):