Imię i nazwisko:..... **V** czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (6pkt)

Lista `LList` zawiera listy o maksymalnej liczbie elementów. Popraw kod, aby poprawnie zwalniać elementy listy. Testem jest funkcja `zad1()`.

```
void zad1() {
   LList l;
   for(int i =0;i< 10;++i) l.push_front();
}
```

```
class LList {
  static const int MAX_SIZE = 3;
  struct Node; typedef std::shared_ptr<Node> PNode; typedef std::weak_ptr<Node> PWNode;
  struct Node { PNode next_; };
  struct List; typedef std::shared_ptr<List> PList; typedef std::weak_ptr<List> PWList;
  struct List {
    List() : count_(0) { }
    ~List() { }
    PNode head_; int count_; PList next_;
    bool push_front() {
      if(count_ < MAX_SIZE) {
        PNode node = PNode(new Node()); node->next_ = head_; head_ = node; ++count_; return true;
      }
      return false;
    }
  };
public:
  LList(){}
  ~LList(){}
  void push_front() {
    if( ! head_ || ! head_->push_front() ) {
      PList l = PList(new List()); l->push_front();
      if( head_ ) { l->next_ = head_; } else { tail_ = l; }
      head_ = l; tail_->next_ = head_;
    }
  }
private:
  PList head_, tail_;
};
```

Zadanie 2 (3pkt)

Liczba liter Twojego nazwiska `LICZBA_LITER`= [ ]. Podaj napis generowany przez `zad2` [ ]

```
class E : public std::exception {
public:
  E(int i) : i_(i) {}
  int i_;
};
struct F {
  F(int i) : i_(i) { cout << i_ << '␣';}
  virtual ~F() { cout << i_ << '␣';}
  int i_;
};
```

```
typedef unique_ptr<F> PF;
void f(PF&& pf) {
  if(pf->i_ > 2)
    f( PF(new F(pf->i_ / 3)));
  else
    throw E(pf->i_);
}
void f(const PF& pf) {
  if(pf->i_ > 0)
    f( PF(new F(pf->i_ / 2)));
}
```

```
void zad2() {
  try {
    PF pb(new F(LICZBA_LITER));
    f(pb);
  } catch(E& e){
    cout << e.i_;
  }
  cout << endl;
}
```

Zadanie 3 (5pkt)

Tworzenie obiektów klasy `Big` jest kosztowne. Przedstaw implementację klasy (o nazwie `LazyBig`, której obiekty pośredniczą przy dostępne do obiektów typu `Big`, minimalizując liczbę tworzonych kopii. Dla przedstawionego poniżej testu obiekt typu `Big` musi być utworzony tylko w dwóch miejscach.

```
class Big {
public:
  Big(int i); //kosztowna
  Big(const Big& f);//kosztowna
  int get() const; //dostęp
  void set(int i); //modyfikacja
};
```

```
LazyBig f(1); //tworzy obiekt
LazyBig f2(f);
int x = f2.get();
f.set(3);//tutaj musi utworzyć obiekt
f2.set(4);
```

Klasa `shared_ptr` posiada metodę `bool unique() const`, która zwraca wartość wyrażenia `use_count() == 1`, gdzie `use_count()` zwraca liczbę wskaźników na obiekt.

```
class LazyBig
```

Zadanie 4 (5pkt)

Dodać klasę `Add`, która dodaje dwie liczby znajdujące się na wierzchołku stosu, wynik wpisuje na stos. Operacja ta powinna być pochodną klasy `Operation`.

```
typedef double Number; typedef std::stack<Number> Stack;
class Operation { //klasa bazowa operacji
public:
  Operation(Stack& st) : stack_(st) {}
  void operator()() { //wykonanie operacji
    Number x = stack_.top();
    stack_.pop();
    stack_.push( doCalculate(x) );
  };
private:
  virtual Number doCalculate(Number) = 0;
  Stack& stack_;
};
```

```
//przyklad operacji, zmiana znaku
class SignChange : public Operation {
public:
  SignChange(Stack& st) : Operation(st) {}
private:
  virtual Number doCalculate(Number x) {
    return -x;
  }
};
```

```
  class Add : public Operation {
```

Zadanie 5 (5pkt)

Dostarczyć funkcję `countBonus`, która sumuje moce obiektów typu `Troll` oraz moce obiektów `Elf` z przynależnością `ENEMY` oraz moce obiektów `Elf` z przynależnością `CONFEDERATE` pomnożnone przez `-2` i moce obiektów `Dwarf` pomnożone przez `-3`. Moc jest zwracana metodą `getPower`.

```
class Visitor {
public:
  virtual void visit(const Elf&) = 0;
  virtual void visit(const Dwarf&) = 0;
  virtual void visit(const Troll&) = 0;
};
class Creature {
public:
  virtual void accept(Visitor& v) const = 0;
  virtual ~Creature() throw() {}
};
typedef shared_ptr<Creature> PCreature;
enum class ATTITUDE { CONFEDERATE, ENEMY };
```

```
class Elf : public Creature {
public:
  Elf() {}
  void accept(Visitor& v) const { v.visit(*this); }
  int getPower() const;
  ATTITUDE getAttitude() const;
};
class Dwarf : public Creature {
public:
  Dwarf() {}
  void accept(Visitor& v) const { v.visit(*this); }
  int getPower() const;
};
class Troll : public Creature {
public:
  Troll() {}
  void accept(Visitor& v) const { v.visit(*this); }
  int getPower() const;
};
```

```
  int countBonus(const vector<PCreature>& v) {
```

Pytanie 1 (1pkt)

Dlaczego kod źródłowy powinien być czytelny ?

Uwagi do prowadzącego: