Imię i nazwisko: ..... U czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (6pkt)

Zmień implementację listy, wykorzystaj sprytne wskaźniki.

```
class List {
  struct Node;
  typedef std::shared_ptr<Node> PNode;
  typedef std::weak_ptr<Node> PWNode;
public:
  List() : head_(nullptr) {}
  ~List() {
    Node* n = head_;
    while( n != nullptr ) {
      Node* p = n;
      n = n->next_;
      delete p;
    }
  }
  void push_front(int val);
private:
  Node* head_;
};
```

```
struct List::Node {
  Node(int v = 0) : val_(v),
                    next_(nullptr), prev_(nullptr) {}
  ~Node() { }
  int val_;
  Node* next_;
  Node* prev_;
};
void List::push_front(int val) {
  Node* n = new Node(val);
  if(head_) head_->prev_ = n;
  n->next_ = head_;
  head_ = n;
}
```

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(const PF& pf) {
   if(pf->i_ > 2) {
      f(PF(new F(pf->i_/2)));
   }
   else {
      throw E(pf->i_);
   }
}
```

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

Zadanie 3 (3pkt)

Podaj napis generowany przez zad3 [ ], stała LICZBA_LITER jest użyta w zad2.

```
class B {
  int i_;
public:
  B(int i) : i_(i) {}
  int inc(){ return doInc();}
protected:
  virtual int doInc(){ return ++i_;}
};
class D1 : public B {
public:
  D1(int i) : B(i-1) {}
protected:
 int doInc() { return B::doInc();}
};
```

```
class D2 : public B {
public:
  D2(int i) : B(i-2) {}
protected:
  int doInc() { return 3; }
};
class M : public D1, public D2 {
public:
  M(int i) : D1(i), D2(i) {}
protected:
  int doInc() {
    return D1::doInc()+D2::doInc();
  }
};
```

```
void zad3() {
   M m(LICZBA_LITER);
   cout << static_cast<D1&>(m).inc();
   cout << static_cast<D2&>(m).inc();
   cout << endl;
}
```

Pytanie 1 (1pkt)

Dlaczego kod źródłowy powinien być czytelny ?

Uwagi do prowadzącego:

Zadanie 4 (6pkt)

Dostarczyć klasy, które będą tworzyły parę: komenda oraz komenda odwrotna. Para taka jest przechowywana w pamięci komend (obiekt `History`). Klasa `Button` wykorzystuje dostarczone rozwiązanie, używa sprytnego wskaźnika `PCreator`.

```
class Command {
  public: virtual void execute() = 0;
};
typedef shared_ptr<Command> PCommand;
class LightOnCommand : public Command {
public: void execute(){ /* implementacja */ }
};
class LightOffCommand : public Command {
public: void execute(){ /* implementacja */ }
};
class FanUpCommand : public Command {
public: void execute(){ /* implementacja */ }
};
class FanDownCommand : public Command {
public: void execute(){ /* implementacja */ }
};
```

```
class History {
public:
  void add(PCommand cmd, PCommand rev) {
    commands_.push_back(cmd); reverse_.push_front(rev);
  }
private:
  list<PCommand> commands_; list<PCommand> reverse_;
};
typedef shared_ptr<History> PHistory;
typedef shared_ptr<Creator> PCreator;
class Button {
public:
  Button(PCreator creator, PHistory history)
  : creator_(creator), history_(history) {}
  void click() {
    PCommand cmd = creator_->createCmd();
    cmd->execute();
    history_->add(cmd, creator_->createReverse() );
  }
private:
  PCreator creator_;
  PHistory history_;
};
```

Zadanie 5 (6pkt)

Dostarczyć funkcje używane w metodzie `Logger::printError`. Dla `LevelErrorEx` pisać na `cout` "Blad poziomu N" w wersji polskiej, zaś "N level error" w wersji angielskiej. N to liczba, pochodzi ze składowej `level_`. Dla `UnknownErrorEx` pisać "Blad bez opisu" i "unknown error".

```
class Logger {
  Logger() : language_("en") {}
  Logger(const Logger&) = delete;
  string language_;
public:
  static Logger& getInstance() {
     static Logger logger;
     return logger;
  }
  void printError(const BaseException& e) {
     if(language_ == "pl") printPolishError(e);
     else printEnglishError(e);
  }
  void setLanguage(string l) { language_ = l; }
};
```

```
class Visitor {
public:
   virtual void visit(const LevelErrorEx& e) = 0;
   virtual void visit(const UnknownErrorEx& e) = 0;
};
class BaseException : public std::exception {
public:
   virtual void accept(Visitor& v) const = 0;
   const char* what() { return "base exception"; }
};
class LevelErrorEx : public BaseException {
public:
   LevelErrorEx(int level) : level_(level) {}
   void accept(Visitor& v) const { v.visit(*this); }
   int getLevel() const { return level_; }
private:
   int level_;
};
class UnknownErrorEx : public BaseException {
public:
   void accept(Visitor& v) const { v.visit(*this); }
};
```