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

Przyjąć, że udostępniona jest przestrzeń nazw std

Zadanie 1 (6pkt)

Elementy macierzy (obiekty `Node`) nie są zwalniane w momencie niszczenia obiektu `Matrix`. Popraw implementację.

```
class Matrix {
   struct Node; typedef std::shared_ptr<Node> PNode; typedef std::weak_ptr<Node> PWNode;
   struct Node {
      Node(int v = 0) : val_(v) { }
      ~Node() { }
      int val_;
      PNode up_, down_, left_, right_;
   };
public:
   Matrix(int size) { //konstruktor tworzy siatkę poprawnie powiązanych obiektów Node
      PNode tmp[size][size]; //tablica pomocnicza (patrz komentarz w treści zadania)
      for(int i=0; i<size;++i) for(int j=0;j<size;++j) tmp[i][j] = PNode(new Node(i*size + j));
      for(int i=0; i<size;++i) for(int j=0;j<size;++j) {
            PNode n = tmp[i][j];
            if(i>0) n->up_ = tmp[i-1][j]; if(i<size-1) n->down_ = tmp[i+1][j];
            if(j>0) n->left_ = tmp[i][j-1]; if(j<size-1) n->right_ = tmp[i][j+1];
      }
      node_ = tmp[0][0];
   }
   ~Matrix() {}
private:
   PNode node_;
};
```

Nie należy poprawiać konstruktora `Matrix`. Używa on VLA (variable length array), które nie są obecne w C++11. Można w tym miejscu użyć typu `vector<vector<PNode>>`.

Zadanie 2 (4pkt)

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;
```

```
F f(F&& b) {
   if (b.i_ < 10) {
      throw E(b.i_);
   }
   return F(b.i_-2);
}
F f(F& b) {
   if(b.i_ > 2) {
      return f(F(b.i_ / 2));
   }
   return F(b.i_-2);
}
```

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

Zadanie 3 (3pkt)

Podaj napis generowany przez `zad3` [ ]

```
class B {
   int i_;
public:
   B(int i) : i_(i) {}
   void inc() { ++i_; }
   int get() { return i_; }
};
```

```
class D1 : virtual public B {
public:
   D1(int i) : B(i-1) {}
   void inc() {
     B::inc(); B::inc();
   }
};
class D2 : virtual public B {
public:
   D2(int i) : B(i-2) {}
   void inc() { B::inc(); }
};
```

```
class M : public D1, public D2 {
public:
   M(int i)
   : B(i), D1(i), D2(i) {}
};

void zad3() {
   M m(3);
   static_cast<D1&>(m).inc();
   cout << m.get() << endl;
}
```

Pytanie (2pkt)

Do czego stosuje się słabe sprytne wskaźniki (obiekty typu std::weak_ptr) ?

Uwagi do prowadzącego:

Zadanie 4 (4pkt)

Produkty `ProfitConstant` mają narzut kwotowy (patrz metoda `getPrice`, zaś `ProfitPercent` narzut procentowy. Zmodyfikuj kod, aby można było tworzyć produkty złożone (obiekty typu `Composite`), które składają się z innych produktów i które dodają narzut kwotowy. Modyfikacja to, między innymi, dodanie metod `add` oraz `getPrice` do klasy `Composite`. Przykład przedstawiony obok (funkcja `zad4`) opisuje produkt `p3` złożony z dwóch produktów, cena `p3` to `321`.

```
void zad4() {
   ProfitConstant p1(100,1);
   ProfitPercent p2(200,5);
   Composite p3(10);
   p3.add(&p1);
   p3.add(&p2);
   cout << p3.getPrice() << endl;
}
```

```
class ProfitConstant {
   int cost_, profit_;
public:
   ProfitConstant(int cost, int profit) : cost_(cost), profit_(profit) {}
   virtual int getPrice() const { return cost_ + profit_; }
};
class ProfitPercent {
   int cost_, profit_;
public:
   ProfitPercent(int cost, int profit) : cost_(cost), profit_(profit) {}
   virtual int getPrice() const { return cost_ + cost_ * profit_ / 100; }
};
class Composite {
   int profit_;
public:
   Composite(int profit) : profit_(profit) {}
};
```

Zadanie 5 (5pkt)

Błędy w systemie są reprezentowane przez klasy `BaseException`, dostarczono wzorzec wizytatora. Uzupełnij implementację funkcji `updateCounters` oraz klasy `Counters`, aby poprawnie zliczać każdy z błędów. Przykład użycia to funkcja `zad5`.

```
class BaseException : public exception {
public:
  virtual void accept(Visitor& v) = 0;
};
class Visitor {
public:
  virtual ~Visitor() {}
  virtual void visit(NetworkError&) = 0;
  virtual void visit(UserBreak&) = 0;
  virtual void visit(BadData&) = 0;
};
```

```
class NetworkError : public BaseException {
public:
  void accept(Visitor& v) { v.visit(*this); }
};
class UserBreak : public BaseException {
public:
  void accept(Visitor& v) { v.visit(*this); }
};
class BadData : public BaseException {
public:
  void accept(Visitor& v) { v.visit(*this); }
};
```

```
class Counters {
public:
   Counters() : networkErrors_(0), userBreaks_(0), badData_(0) {}

   int networkErrors_, userBreaks_, badData_;
};

void updateCounters(Counters& counters, BaseException& ex)
```

```
ostream& operator<<(ostream& os, const Counters& c) {
  os << "Network:" << c.networkErrors_ << ',' << "␣User:" << c.userBreaks_ << ',' << "␣Data:" << c.badData_;
}
Counters counters;
void zad5() {
   try {
      //kod ktory moze zglosic wyjatek typu BaseException
   } catch(BaseException& e) { updateCounters(counters, e); }
   cout << counters << endl;
}
```