Задачи на наследование и полиморфизм

Задача 1 ***Чтение и вывод JSON.*** *Требуется реализовать набор классов для чтения и вывода файлов в упрощённом формате JSON. Формат JSON поддерживает организацию данных в виде словарей и списков. В данной задаче ограничимся лишь списками из целых и действительных чисел. Список задаётся в квадратных скобках. Элементами списка могут быть целые и дробные числа, а также другие списки. Элементы списка разделены запятой. Например,*

```
[
  1,
  2,
  3,
  [3.14],
  1.35,
  [
    3.2,
    [ 1, 2, 3 ]
  ]
]
```

Для представления JSON в программе используется следующий набор классов. Каждый элемент этой структуры данных должен быть представлен объектом класса, производного от базового класса ***Node****.*

```
class Node {
 public:
  /* Любые данные можно преобразовать в строку или вывести в поток вывода. */
  virtual std::string ToString() const = 0;
  virtual void Print(std::ostream& out) const = 0;

  /* Каждый объект типа Node в действительности является числом или списком.
     Эти методы должны конвертировать объект в указанный тип данных.
     В случае, если конвертация невозможна, требуется выбросить исключение.
  */
  const Integer& AsInteger() const;
  Integer& AsInteger();

  const Double& AsDouble() const;
  Double& AsDouble();

  const List& AsList() const;
  List& AsList();

  virtual ~Node() { }
};
```

Целому числу, действительному числу и списку должны соответствовать классы, производные от `Node`:

```
class Integer : public Node {
 private:
  int value;

 public:
  Integer();
  Integer(int value);

  /* Методы для чтения/записи приватных полей. */
  int Value() const;
  int& Value();

  std::string ToString() const override;
  void Print(std::ostream& out) const override;
};

class Double : public Node {
 private:
  double value;

 public:
  Double();
  Double(double value);

  double Value() const;
  double& Value();

  std::string ToString() const override;
  void Print(std::ostream& out) const override;
};

class List : public Node {
 private:
  // Элементы списка. Каждый элемент --- указатель на Node для
  // того, чтобы обеспечить полиморфное поведение.
  Node** values;
  size_t count;

 public:
  List();
  List(Node** values, size_t count);

  const Node* const* Values() const;
  Node**& Values();

  // Функция добавляет элемент в список.
  void AddNode(Node* node);

  /* Получить элемент списка для чтения или записи.
```

```
  Методы должны выбрасывать исключение в случае некорректного индекса.
  */
  Node* operator(size_t index);
  const Node* operator(size_t index) const;

  std::string ToString() const override;
  void Print(std::ostream& out) const override;
};
```

Требования к программе:

1. *Реализовать классы с предложенными интерфейсами.*
2. *Реализовать считывание JSON из строки и создание соответствующей системы классов.*

```
Node* FromString(char* jsonCode);
```

   В случае ошибки чтения функция должна выбрасывать исключение.
3. *Реализовать программу, которая считывает JSON из текстового файла, по команде пользователя позволяет изменить любое число в списке на произвольном уровне исходного JSON.*
4. *Реализовать вывод изменённого пользователем JSON в файл.*
5. *Программа должна перехватывать исключения и выдавать информацию об исключении на экран.*

Задача 2 ***Дерево выражения.*** *Требуется реализовать набор классов для представления дерева выражения. В таком дереве операции служат промежуточными узлами, а числа находятся в листьях дерева. Для вычисления значения выражения нужно рекурсивно выполнить все операции, начиная с корня. Аргументами для операций служат значения в дочерних узлах.*

Для представления дерева выражения предлагается использовать следующую систему классов:

```
/* Базовый класс для узла дерева выражения. */
class Node {
 public:
  /* Метод служит для вычисления значения составленного дерева выражения.
     Этот метод вызывается рекурсивно от корня к листьям. */
  virtual double Evaluate() const = 0;

  virtual ~Node() { }
};

class Value : public Node {
 private:
  double value;
 public:
  Value(double value);

  /* Этот метод возвращает само число, записанное в узле дерева. */
  double Evaluate() const override;
};


class BinaryOperation : public Node {
 private:
  Node* left;
  Node* right
 public:
  BinaryOperation(Node* left, Node* right);
};

class UnaryOperation : public Node {
 private:
  Node* argument;
 public:
  UnaryOperation(Node* argument);
};

class Addition : public BinaryOperation {
 public:
  Addition(Node* left, Node* right);

  /* Метод возвращает результат операции сложения для своих аргументов. */
  double Evaluate() const override;
};

class Subtraction : public BinaryOperation {
 public:
```

```
47   Subtraction(Node* left, Node* right);
48
49   /* Метод возвращает результат операции вычитания для своих аргументов. */
50   double Evaluate() const override;
51 };
52
53 class Multiplication : public BinaryOperation {
54  public:
55   Multiplication(Node* left, Node* right);
56
57   /* Метод возвращает результат операции умножения для своих аргументов. */
58   double Evaluate() const override;
59 };
60
61 class Division : public BinaryOperation {
62  public:
63   Division(Node* left, Node* right);
64
65   /* Метод возвращает результат операции деления для своих аргументов.
66   В случае деления на нуль требуется выбросить исключение. */
67   double Evaluate() const override;
68 };
69
70 class Negation : public UnaryOperation {
71  public:
72   Division(Node* argument);
73
74   /* Метод возвращает результат операции унарный минус для своего аргумента. */
75   double Evaluate() const override;
76 };
```

Требования к программе:

1. *Реализовать классы с предложенными интерфейсами.*
2. *Реализовать считывание формулы без скобок из строки и создание соответствующей системы классов.*

```
1   Node* FromString(char* formula);
```

   В случае ошибки чтения функция должна выбрасывать исключение.

3. *Реализовать программу, которая считывает арифметическое выражение без скобок из текстового файла, выводит результат вычисления выражения на экран.*
4. *Программа должна перехватывать исключения и выдавать информацию об исключении на экран.*

Задача 3 ***Вывод SVG.*** *Реализуйте класс для вывода изображений в упрощённом формате SVG. Файл в формате SVG является обычным текстовым XML документом, изображение представляется в виде тегов. Сначала идёт заголовок, затем внутри тега* `<svg> </svg>` *записаны теги, описывающие отдельные графические объекты, например, окружности (тег* `<circle />`*), текст (тег* `<text> </text>`*), ломаные (тег* `<polyline />`*) и т.д. SVG файлы должны отображаться любым браузером. Например,*

```
<?xml version="1.0" encoding="UTF-8" ?>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">

  <polyline points="100,100 500,500 400,50" fill="green"
            stroke="rgb(200,100,15)" stroke-width="3" />

  <circle cx="200" cy="200" r="30" fill="red" stroke="none" stroke-width="1" />

  <text x="25" y="60" dx="-5" dy="10"
        font-size="22" font-family="Verdana"
        fill="black" stroke="yellow" stroke-width="1">
    The text should be placed here.
  </text>

</svg>
```

Для представления SVG документа в программе предлагается следующая иерархия классов:

```
struct Point {
  double x;
  double y;
};

/* Цвет может задаваться либо в формате "rgb(120,10,30)" каждая компонента от 0 до 255,
   либо текстом, например, "none", "white", "black", "red", "green", "blue", и т.д. */
class Color {
 public:
  /* Конструкторы должны выбрасывать исключение в случае некорректного или
     неподдерживаемого цвета.
  */
  Color(const char* color);
  Color(int red, int green, int blue);

  void Print(std::ostream& out) const;
};

/* Общие свойства объектов. */
class Object {
 private:
  Color fillColor;         // Задаёт аттрибут "fill" --- цвет заливки.
  Color strokeColor;       // Задаёт аттрибут "stroke" --- цвет линии.
  double strokeWidth;      // Задаёт аттрибут "stroke-width" --- толщину линии.

 public:
```

```
27
28    /* Реализовать методы, задающие указанные параметры.
29    Методы должны выбрасывать исключение в случае некорректных параметров. */
30
31    /* Вывести объект в поток. */
32    virtual void Print(std::ostream& out) const = 0;
33  };
34
35  /* Тег <circle />, описывающий окружность. */
36  class Circle : public Object {
37   private:
38    Point center;            // Задаёт аттрибуты "cx" и "cy" --- центр окружности.
39    double radius;           // Задаёт аттрибут "r" --- радиус окружности.
40
41    /* Реализовать методы, задающие указанные параметры.
42    Методы должны выбрасывать исключение в случае некорректных параметров. */
43
44   public:
45    void Print(std::ostream& out) const override;
46  };
47
48  /* Тег <polyline />, описывающий ломаную. */
49  class Polyline : public Object {
50   private:
51    /* Задаёт аттрибут "points" в формате "x1,y1 x2,y2 x3,y3 ..."
52       Отдельные точки записываются через пробел. Координаты разделяются запятой. */
53    Point* points;
54    size_t count;
55
56   public:
57
58    /* Реализовать методы, задающие указанные параметры. */
59
60    void Print(std::ostream& out) const override;
61  }
62
63  /* Тег <text> </text>, описывающий текст. */
64  class Text : public Object {
65   private:
66    Point point;               // Задаёт аттрибуты x и y.
67    Point offset;              // Задаёт аттрибуты dx и dy.
68    unsigned fontSize;         // Задаёт аттрибут "font-size" --- размер шрифта.
69    std::string fontFamily;    // Задаёт аттрибут "font-family" --- название шрифта.
70    std::string data;          // Текст, который будет выведен.
71
72   public:
73    void Print(std::ostream& out) const override;
74  };
75
76  class Document {
77   private:
78    // Массив указателей на объекты. Указатели нужны для обеспечения
79    // полиморфного поведения.
```

```
80   Object** objects;
81   size_t count;
82  public:
83   void AddObject(Object* obj);
84
85   /* Выводит заголовок SVG файла и содержимое тега <svg> </svg>,
86      вызывая соответствующие методы у объектов. */
87   void Print(std::ostream& out) const;
88 };
```

Требования к программе:

1. *Реализовать предлагаемый набор классов.*
2. *Написать программу, которая считывает команды пользователя с клавиатуры и формирует SVG документ. Программа должна сформировать набор объектов, для представления SVG документа и вывести его в файл.*
3. *Программа должна перехватывать исключения и сообщать о них пользователю.*

Задача 4 ***Классы для ввода/вывода.*** *Даны абстрактные классы для форматированного ввода/вывода*

```
class Reader {
 public:
  virtual Reader& operator>>(int value) = 0;
  virtual Reader& operator>>(double value) = 0;
  virtual Reader& operator>>(char* str) = 0;

  virtual ~Reader() {};
};

class Writer {
 public:
  virtual Writer& operator<<(int value) = 0;
  virtual Writer& operator<<(double value) = 0;
  virtual Writer& operator<<(char* str) = 0;

  virtual ~Writer() {};
};
```

Требуется написать классы `StringReader`, `StringWriter` *для форматированного ввода/вывода строки, а также классы* `FileReader` *и* `FileWriter` *для форматированного ввода/вывода из/в файл.*

Список требований:

- *Класс* `StringReader` *должен в конструкторе получать строку для считывания из неё данных в формате* `char*`.
- *Классы* `FileReader` *и* `FileWriter` *должны в конструкторе получать имя файла для ввода/вывода.*
- *Все операции ввода/вывода внутри классов следует реализовать исключительно средствами языка C.*
- *В случае ошибок ввода/вывода методы класса должны выбрасывать исключения.*
- *Классы должны корректно работать с памятью. Открытые файлы и выделенная память должны освобождаться в деструкторе. Конструктор копирования и операцию копирующего присваивания следует удалить.*
- *Реализовать пример программы, иллюстрирующей использование всех методов написанных классов.*
- *Программа должна перехватывать и корректно обрабатывать все возможные исключения.*

Задача 5 ***Быстрая сортировка.*** *Реализовать набор классов и функцию для их сортировки алгоритмом Quick Sort.*

```
// Базовый класс для всех объектов.
class Item {
 public:
  virtual bool operator<(const Item&) = 0;
  virtual ~Item() { }
};

// Алгоритм быстрой сортировки.
void QuickSort(Item** items, size_t count);
```

Требуется реализовать производные классы `Real` и `Pair` вещественного числа типа `double` и пары чисел типа `double` на базе класса `Item`.

Требования к программе:

- Требуется реализовать указанные классы и функцию `QuickSort()`.
- Функция `QuickSort()` должна проверить корректность заданного отношения порядка (чтобы операция не выполнялось одновременно `A < B` и `B < A` для элементов массива) и отсортировать массив алгоритмом быстрой сортировки. В случае, если заданное отношение порядка некорректно, выбросить исключение.
- При реализации оператора сравнения можно считать, что сравниваются элементы одинаковых типов (то есть оригинальные производные классы одинаковы). Если типы не совпадают, то следует выбросить исключение.
- Написать пример программы с иллюстрацией работы функции, в том числе и написать классы с некорректным отношением порядка и примеры к ним.
- Программа должна перехватывать и корректно обрабатывать все исключения.

Задача 6 ***Решение линейного уравнения.*** *Реализовать набор классов и функцию для решения линейного уравнения*

$$ax + b = 0.$$

```
class Value {
 public:
  // Эти функции предназначены для создания новых объектов
  // типа производного класса.
  // Эта функция создаёт объект со значением 0.
  virtual Value* CreateZeroValue() = 0;
  // Эта функция создаёт копию текущего объекта.
  virtual Value* Duplicate() = 0;

  virtual Value& operator+=(const Value&) = 0;
  virtual Value& operator-=(const Value&) = 0;
  virtual Value& operator*=(const Value&) = 0;
  virtual Value& operator/=(const Value&) = 0;

  virtual ~Value() { }
};

void Solve(Value& a, Value& b, Value& result);
```

Требуется реализовать производные классы ***RealValue*** *и* ***ComplexValue*** *на базе класса* ***Value****, а также написать функцию* ***Solve****.*

Список требований:

- *Реализовать указанные классы.*
- *Классы должны генерировать исключение при попытке деления на "нуль" (здесь следует учитывать особенности машинной арифметики с плавающей точкой).*
- *При реализации арифметических операций следует считать, что аргументы одинакового типа. При несовпадении оригинальных типов аргументов (производных классов) следует выбросить исключение.*
- *Требуется реализовать программу, иллюстрирующую работу функции* ***Solve****, а также работу исключений.*
- *Программа должна перехватывать и корректно обрабатывать все исключения.*

Задача 7 ***Решение квадратного уравнения.*** *Реализовать набор классов и функцию для решения квадратного уравнения*

$$ax^2 + bx + c = 0.$$

```cpp
class Value {
 public:
  // Эти функции предназначены для создания новых объектов
  // типа производного класса.
  // Эта функция создаёт объект со значением 0.
  virtual Value* CreateZeroValue() = 0;
  // Эта функция создаёт копию текущего объекта.
  virtual Value* Duplicate() = 0;

  virtual Value& operator+=(const Value&) = 0;
  virtual Value& operator-=(const Value&) = 0;
  virtual Value& operator*=(const Value&) = 0;
  virtual Value& operator/=(const Value&) = 0;

  // Извлекает квадратный корень из текущего числа.
  virtual void Sqrt() = 0;

  virtual ~Value() { }
};

void Solve(Value& a, Value& b, Value& c, Value*& result1, Value*& result2);
```

Требуется реализовать производные классы `RealValue` *и* `ComplexValue` *на базе класса* `Value`*, а также написать функцию* `Solve`*.*

Список требований:

- *Реализовать указанные классы.*
- *Классы должны генерировать исключение при извлечении квадратного корня из отрицательного числа в классе действительных чисел, а также при попытке деления на "нуль" (здесь следует учитывать особенности машинной арифметики с плавающей точкой).*
- *При реализации арифметических операций следует считать, что аргументы одинакового типа. При несовпадении оригинальных типов аргументов (производных классов) следует выбросить исключение.*
- *Требуется реализовать программу, иллюстрирующую работу функции* `Solve`*, а также работу исключений.*
- *Программа должна перехватывать и корректно обрабатывать все исключения.*

Задача 8 ***Решение системы линейных уравнений.*** *Реализовать набор классов и функцию для решения СЛАУ методом Гаусса с выбором главного элемента по столбцу.*

```
class Value {
 public:
  // Эти функции предназначены для создания новых объектов
  // типа производного класса.
  // Эта функция создаёт объект со значением 0.
  virtual Value* CreateZeroValue() = 0;
  // Эта функция создаёт копию текущего объекта.
  virtual Value* Duplicate() = 0;

  virtual Value& operator+=(const Value&) = 0;
  virtual Value& operator-=(const Value&) = 0;
  virtual Value& operator*=(const Value&) = 0;
  virtual Value& operator/=(const Value&) = 0;

  virtual ~Value() { }
};

// Здесь везде двойные указатели для обеспечения полиморфного поведения.
// Матрица представлена в виде одномерного массива.
Solve(Value** matrix, Value** rhs, Value** x, size_t size);
```

Требуется реализовать производные классы ***RealValue*** *и* ***AdvancedValue*** *на базе класса* ***Value****, а также написать функцию* ***Solve****. Класс* ***AdvancedValue*** *содержит не только действительное значение, но и абсолютную погрешность, с которой оно задано.*

Машинная точность типа `double` *около значения* `1.0` *вычисляется при помощи следующего кода:*

```
#include <limits>

double eps = std::numeric_limits<double>::epsilon();
```

Список требований:

- *Реализовать указанные классы.*
- *Классы должны генерировать исключение при попытке деления на "нуль" (здесь следует учитывать особенности машинной арифметики с плавающей точкой).*
- *При реализации арифметических операций следует считать, что аргументы одинакового типа. При несовпадении оригинальных типов аргументов (производных классов) следует выбросить исключение.*
- *Арифметические операции для класса* ***AdvancedValue*** *должны корректно вычислять абсолютную погрешность результата.*
- *Требуется реализовать программу, иллюстрирующую работу функции* ***Solve****, а также работу исключений.*
- *Программа должна перехватывать и корректно обрабатывать все исключения.*

- *Требуется заполнить матрицу по формуле* $a_{ij} = \dfrac{1}{i+j+1}$, $i, j = 0, \ldots, N-1$, *правую часть заполнить по формуле* $\mathrm{rhs}_i = \sum\limits_{k=0}^{(N-1)/2} a_{i,2k}$, *решить систему для значений* $N = 1, \ldots, 15$, *вывести решения вместе с погрешностями.*

Задача 9 ***Нахождение НОД и НОК.*** *Дан абстрактный класс целого числа. Требуется реализовать нахождение наименьшего общего кратного и наибольшего общего делителя.*

```
class Value {
 public:
  // Эти функции предназначены для создания новых объектов
  // типа производного класса.
  // Эта функция создаёт объект со значением 0.
  virtual Value* CreateZeroValue() = 0;
  // Эта функция создаёт копию текущего объекта.
  virtual Value* Duplicate() = 0;

  virtual Value& operator+=(const Value&) = 0;
  virtual Value& operator-=(const Value&) = 0;
  virtual Value& operator*=(const Value&) = 0;
  virtual Value& operator/=(const Value&) = 0;

  virtual ~Value() { }
};

// Наибольший общий делитель.
void GCD(Value& a, Value& b, Value& result);
// Наименьшее общее кратное.
void LCM(Value& a, Value& b, Value& result);
```

Целые числа типов `int` *и* `long` *не допускают переполнений, в языке C++ это считается* ***неопределённым поведением****. Максимальное и минимальное значение указанных типов можно узнать при помощи следующего кода:*

```
#include <limits>

int minIntVal = std::numeric_limits<int>::lowest();
int maxIntVal = std::numeric_limits<int>::max();

long minLongVal = std::numeric_limits<long>::lowest();
long maxLongVal = std::numeric_limits<long>::max();
```

Требуется реализовать производные классы `Long` *и* `Int` *на базе класса* `Value`*, которые реализуют арифметические операции с контролем переполнения. Кроме того, требуется реализовать указанные функции, вычисляющие НОД и НОК.*

Требования к задаче:

- *Реализовать указанные классы и функции.*
- *Некорректные операции, приводящие к переполнению, должны генерировать исключение. При этом само переполнение при внутренних вычислениях никогда не должно происходить. Также исключение должно выбрасываться при делении на нуль.*
- *Требуется реализовать программу, иллюстрирующую работу указанных функций, а также исключений.*
- *Программа должна корректно отлавливать и обрабатывать исключения.*

Задача 10 ***Pipeline.*** *Требуется реализовать несколько классов для векторной обработки данных и класс для соединения указанных операторов в один.*

```
// Класс массива. Можно считать, что его размер не меняется.
class Array {
 public:
  Array(size_t size);
  Array(const Array& other);

  ~Array();

  Array& operator=(const Array& other);

  double operator[](size_t index) const;
  double& operator[](size_t index);

  size_t Size() const { return size; }
 private:
  double* data;
  size_t size;
};

// Базовый класс для всех операций. Метод Process() получает на вход
// массив данных и возвращает новый массив после применения к исходному
// массиву требуемой операции.
class Operator {
 public:
  virtual Array Process(const Array& input) const = 0;
  virtual ~Operator() { }
};

// В этом классе метод Process должен добавлять константу к каждому элементу
// входного массива.
class AddConstant : public Operator {
 private:
  double value;
 public:
  AddConstant(double value);

  // Добавить сюда необходимые методы.
};

// В этом классе метод Process ограничивает элементы входного массива заранее
// заданными минимальным и максимальным значением. То есть элементы, выходящие
// за заданные границы заменяются соответствующим граничным значением.
class Clip : public Operator {
 private:
  double minValue;
  double maxValue;
 public:
  Clip(double minValue, double maxValue);

```

```
50   // Добавить сюда необходимые методы.
51 };
52
53 // В этом классе метод Process должен поэлементно вычислить натуральный логарифм
54 // от массива данных.
55 class Log : public Operator {
56   // Добавить сюда необходимые методы.
57 };
58
59 class Pipeline : public Operator {
60  private:
61   // Произвольные операторы. Двойные указатели нужны для
62   // обеспечения полиморфного поведения.
63   Operator** ops;
64   // Их количество.
65   size_t count;
66  public:
67   // Здесь требуется дописать конструкторы, метод для добавления оператора
68   // в цепочку и метод Process() для применения операторов по цепочке.
69 };
```

Требуется реализовать указанные классы. Все классы имеют одинаковый интерфейс для работы с массивом данных. Класс `Pipeline` *применяет добавленные в него операции по цепочке.*

Требования:

- *Реализовать указанные классы.*
- *Метод* `Process` *класса* `Log` *должен генерировать исключение в случае некорректного аргумента (неположительные числа среди элементов массива данных).*
- *Реализовать программу, иллюстрирующую пример использования каждого класса, а также пример, на котором класс* `Log` *выбрасывает исключение.*
- *Программа должна корректно перехватывать и обрабатывать все исключения.*