## **Лабораторная работа №2**

### **Цель работы**

Целью лабораторной работы является:

· Закрепление навыков работы с классами.

· Создание простых динамических структур данных.

· Работа с объектами, передаваемыми «по значению».

### **Задание**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну фигуру ( колонка фигура 1),** согласно вариантам задания. Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
* Классы фигур должны содержать набор следующих методов:
  + Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
  + Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1.
  + Оператор копирования (=)
  + Оператор сравнения с такими же фигурами (==)
* Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
* Класс-контейнер должен содержать набор следующих методов:
  + Метод по добавлению фигуры в контейнер.
    - Очередь: **Push**
    - Динамический массив: **InsertLast**
    - Связанный список: **InsertFirst, InsertLast, Insert**
    - Бинарное дерево: **Push**
    - N-дерево: **Update**
  + Метод по получению фигуры из контейнера.
    - Очередь: **Top**
    - Динамический массив: **operator[]**
    - Связанный список: **First, Last, GetElement**
    - Бинарное дерево: **GetNotLess**
    - N-дерево: **GetItem**
  + Метод по удалению фигуры из контейнера.
    - Очередь: **Pop**
    - Динамический массив: **Remove**
    - Связанный список: **RemoveFirst, RemoveLast, Remove**
    - Бинарное дерево: **Pop**
    - N-дерево: **RemoveSubTree**
  + Перегруженный оператор по выводу контейнера в поток std::ostream (<<).
  + Деструктор, удаляющий все элементы контейнера.
  + Набор специальных методов для класса-контейнера(см. Приложение).

Полное описание всех методов можно найти в приложении к лабораторной.

Нельзя использовать:

* Стандартные контейнеры std.
* Шаблоны (template).
* Различные варианты умных указателей (shared\_ptr, weak\_ptr).

Программа должна позволять:

* Вводить произвольное количество фигур и добавлять их в контейнер.
* Распечатывать содержимое контейнера.
* Удалять фигуры из контейнера.

**Приложение**

**Вариант: Очередь:**

class TQueue {

public:

// Конструктор по умолчанию

TQueue();

// Конструктор копирования очереди.

TQueue(const TQueue& other);

// Метод, добавляющий фигуру в конец очереди.

void Push(const Polygon& polygon);

// Метод, убирающий первую фигуру из очереди.

void Pop();

// Метод, возвращающий ссылку на первую в очереди фигуру

const Polygon& Top();

// Метод, проверяющий пустоту очереди

bool Empty();

// Метод, возвращающий длину очереди

size\_t Length();

// Оператор вывода очереди в формате:

// "=> Sn Sn-1 ... S1 =>", где Si - площадь фигуры,

// а n – номер последней фигуры в очереди

friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TQueue();

};

**Вариант: Динамический массив:**

class TVector {

public:

// Конструктор по умолчанию

TVector();

// Конструктор копирования

TVector(const TVector& other);

// Метод, добавляющий фигуру в конец массива

void InsertLast(const Polygon& polygon);

// Метод, удаляющий последнюю фигуру массива

void RemoveLast();

// Метод, возвращающий последнюю фигуру массива

const Polygon& Last();

// Перегруженный оператор обращения к массиву по индексу

Polygon& operator[] (const size\_t idx)

// Метод, проверяющий пустоту

bool Empty();

// Метод, возвращающий длину массива

size\_t Length();

// Оператор вывода для массива в формате:

// "[S1 S2 ... Sn]", где Si - площадь фигуры

friend std::ostream& operator<<(std::ostream& os, const TVector& arr);

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TQueue();

};

**Вариант: Связный список:**

class TLinkedList {

public:

// Конструктор по умолчанию

TLinkedList();

// Конструктор копирования

TLinkedList(const TLinkedList& other);

// Метод, возвращающий первую фигуру списка

const Polygon& First();

// Метод, возвращающий последнюю фигуру списка

const Polygon& Last();

// Метод, добавляющий элемент в начало списка

void InsertFirst(const Polygon& polygon);

// Метод, добавляющий фигуру в конец списка

void InsertLast(const Polygon& polygon);

// Метод, добавляющий фигуру в произвольное место списка

void Insert(const Polygon& polygon, size\_t position);

// Метод, удаляющий первый элемент списка

void RemoveFirst();

// Метод, удаляющий последний элемент списка

void RemoveLast();

// Метод, удаляющий произвольный элемент списка

void Remove(size\_t position);

// Метод получения фигуры списка по индексу.

const Polygon& GetItem(size\_t idx);

// Метод, проверяющий пустоту списка

bool Empty();

// Метод, возвращающий длину массива

size\_t Length();

// Оператор вывода для массива в формате:

// "S1 -> S2 -> ... -> Sn", где Si - площадь фигуры

friend std::ostream& operator<<(std::ostream& os, const TLinkedList& list);

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TLinkedList();

};

**Вариант: Бинарное дерево:**

// В каждой вершине двоичного дерева хранится фигура и счетчик.

// Если в структуру добавляется фигура, которая уже есть,

// счетчик инкрементируется.

class TBinaryTree {

public:

// Конструктор по умолчанию.

TBinaryTree();

// Метод добавления фигуры согласно правилу:

// При добавлении фигуры в новую вершину,

// в вершине создается счетчик со значением 1.

// Если фигура совпадает с фигурой из вершины,

// счетчик в вершине увеличивается на 1.

// Иначе сравнивается с вершиной из левого поддерева,

// если площадь фигуры < площади в вершине

// или с вершиной правого поддерева, если >=.

void Push(const Polygon& polygon);

// Метод получения фигуры из контейнера.

// Если площадь превышает максимально возможную,

// метод должен бросить исключение std::out\_of\_range

const Polygon& GetItemNotLess(double area);

// Метод, возвращающий количество совпадающих фигур с данными параметрами

size\_t Count(const Polygon& polygon);

// Метод по удалению фигуры из дерева:

// Счетчик вершины уменьшается на единицу.

// Если счетчик становится равен 0,

// вершина удаляется с заменой на корректный узел поддерева.

// Если такой вершины нет, бросается исключение std::invalid\_argument

void Pop(const Polygon &polygon);

// Метод проверки наличия в дереве вершин

bool Empty();

// Оператор вывода дерева в формате вложенных списков,

// где каждый вложенный список является поддеревом текущей вершины:

// "S0: [S1: [S3, S4: [S5, S6]], S2]",

// где Si - строка вида количество\*площадь\_фигуры

// Пример: 1\*1.5: [3\*1.0, 2\*2.0: [2\*1.5, 1\*6.4]]

friend std::ostream& operator<<(std::ostream& os, const TBinaryTree& tree);

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TBinaryTree();

**};**

**Вариант: N-дерево:**

// Если какой-то вершины в tree\_path не существует,

// то функция должна бросить исключение std::invalid\_argument

// Если вершину нельзя добавить из за переполнения,

// то функция должна бросить исключение std::out\_of\_range

class TNaryTree {

public:

// Конструктор по умолчанию. Умолчательное значение n = 3

TNaryTree();

// Конструктор с указанием ограничения на количество детей

TNaryTree(int n);

// Метод добавления или обновления вершины в дереве согласно заданному пути.

// Путь задается строкой вида: "cbccbccc",

// где 'c' - старший ребенок, 'b' - младший брат

// последний символ строки - вершина, которую нужно добавить или обновить.

// Пустой путь "" означает добавление/обновление корня дерева.

void Update(const Polygon& polygon, const std::string& tree\_path="");

// Метод получения фигуры из контейнера

const Polygon& GetItem(const std::string& tree\_path="");

// Метод по удалению вершины и ее поддерева

void RemoveSubTree(const std::string& tree\_path);

// Метод проверки наличия в дереве вершин

bool Empty();

// Оператор вывода дерева в формате вложенных списков, где

// каждый вложенный список является поддеревом текущей вершины:

// "S0: [S1: [S3, S4: [S5, S6]], S2]", где Si - площадь фигуры

friend std::ostream& operator<<(std::ostream& os, const TNaryTree& tree);

// Деструктор

virtual ~TNaryTree();

**};**