Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра: 806 "Вычислительная математика и программирование"
Факультет: "Информационные технологии и прикладная математика"
Дисциплина: "Объектно-ориентированное программирование"

Группа:
Студент: Пашкевич Андрей Романович
Преподаватель: Поповкин Александр Викторович

Москва, 2017

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Вариант №17

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 вариантов задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<>).
- Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен соджержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (опередеряется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеряется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<)
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

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.

Описание структуры классов и алгоритма работы программы:

Triangle.cpp	
Triangle()	Конструктор класса треугольник
Triangle(std::istream &is)	Конструктор класса треугольник из стандартного потока ввода
Triangle(size_t i, size_t j, size_t k)	Конструктор класса треугольник по задаваемым в тексте программы значениям
Triangle(const Triangle& orig)	Конструктор копии класса треугольник
double Square()	Функция для вычисления площади фигур по её параметрам
void Print()	Функция для вывода фигуры на экран
~Triangle()	Деструктор класса треугольник
Triangle& operator =(const Triangle& right);	Переопределенный оператор копирования класса треугольник
Friend std::ostream& operator<<(std::ostream& os, const Triangle& obj);	Перегруженный оператор ввода для класса треугольник
Friend std::ostream& operator>>(std::ostream& os, const Triangle& obj);	Перегруженный оператор вывода для класса треугольник
Tree.cpp	
Tree();	Конструктор класса
Tree(TreeNode* node);	Конструктор класса с заданным узлом
Tree(const Tree& orig);	Конструктор класса с заданной фигурой
void add(Triangle triangle);	Добавление фигуры в дерево
TreeNode* del(Triangle triangle);	Удаление из дерева по параметрам фигуры
bool empty();	Проверка дерева на пустоту
friendstd::ostream& operator<<(std::ostream& os, const Tree& tree);	Перегруженный оператор вывода
virtual ~Tree();	Деструктор класса
TreeNode.cpp	
TreeNode(const Triangle& triangle);	Конструктор Класса
TreeNode(const TreeNode& orig);	Функция копирования
friend std::ostream& operator<<(std::ostream& os, const TreeNode& obj);	Перегруженный оператор ввода
TreeNode* SetLeft(TreeNode* ptr)	Установка ссылки на левый узел
TreeNode* GetLeft();	Получение ссылки на левый узел
TreeNode* SetRight(TreeNode* ptr);	Установка ссылки на правый узел
TreeNode* GetRight();	Получение ссылки на правый узел
Triangle GetTriangle() const;	Получение фигуры из узла
virtual ~TreeNode();	Деструктор класса

Динамические структуры данных – это структуры данных, *память* под которые выделяется и освобождается по мере необходимости. Преимуществом таких структур данных является, то что память под отдельные элементы выделяется в момент, когда они "начинают существовать" в процессе выполнения программы, а не во *время компиляции*.

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями.

Листинг программы:

Вывод:

В данной лабораторной работе описывается динамическая структура данных, элементами которой является класс фигур. С помощью неё возможно получать доступ к объектам в то время, когда они понадобятся. Также я познакомился с перегрузкой операторов в языке программирования C++. Перегрузка оператора – переопределение этого оператора для работы с данным классом. С помощью неё можно писать более понятный код, что облегчает дальнейшее его развитие.