Московский авиационный институт

(национально исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра прикладной математики и информатики

Лаборатория №5 по курсу «Объектно-ориентируемое программирование»

Студент: Рождественских Аднрей Сергеевич

Преподаватель: А. В. Поповкин

Группа: М80-206Б

Вариант: 15

Дата:

Оценка:

Подпись:

Москва, 2018.

1 Цель работы

• Закрепление навыков работы с шаблонами классов.

• Построение итераторов для динамических структур данных.

2 Задача

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР №4) спроектировать и разработать Итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например: for(auto i : stack) std::cout « \*i « std::endl;

• Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.

• Шаблон класса-контейнера должен соджержать объекты используя std:sharedptr.

• Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.

• Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (опеределяется структурой контейнера).

• Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (опеределяется структурой контейнера).

• Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток std::ostream («).

• Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.

• Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

3 Описание

Для доступа к элементам некоторого множества элементов используют специальные объекты, называемые итераторами. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки, так что обычный указатель тоже может использоваться как итератор.

Категории итераторов:

• Итератор ввода (input iterator) – используется потоками ввода.

• Итератор вывода (output iterator) – используется потоками вывода.

• Однонаправленный итератор (forward iterator) – для прохода по элементам в одном направлении.

• Двунаправленный итератор (bidirectional iterator) – способен пройти по элементам в любом направлении. Такие итераторы реализованы в некоторых контейнерных типах stl (list, set, multiset, map, multimap).

• Итераторы произвольного доступа (random access) – через них можно иметь доступ к любому элементу. Такие итераторы реализованы в некоторых контейнерных типах stl (vector, deque, string, array).

4 Изменения

Добавился файл Titerator.h, изменились файлы TBinaryTree.cpp, TbinaryTree.h

5 Выводы

Был разработан итератор для массива. Объяснение принципа работы займет много времени, но он становится очевиден при прочтении функции GetNext(). Итераторы - удобный инструмент для перебора всех элементов структуры. Благодаря им можно значительно упростить нагромождения кода.