Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Объектно-ориентированное программирование»

Студент: С. М. Бокоч
Преподаватель:
Группа: М8О-204Б

Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №5

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;

Фигуры. Квадрат, трапеция, прямоугольник.

Контейнер. Массив.

1 Теория

Итератор — объект, позволяющий программисту перебирать все элементы коллекции без учёта особенностей её реализации. Предназначен итератор исключительно для последовательного доступа к элементам

Итераторы позволили алгоритмам получать доступ к данным, содержащимся в контейнере, независимо от типа контейнера. Но для этого в каждом контейнере потребовалось определить класс итератора. Таким образом алгоритмы воздействуют на данные через итераторы, которые знают о внутреннем представлении контейнера.

Существует пять категорий итераторов. Ниже описаны категории в порядке возрастания силы.

1. Итератор вывода.
2. Итератор ввода.
3. Однонаправленный.
4. Двунаправленный.
5. По произвольному доступу.

2 Листинг

```
//TIterator.h
template <class T>
class TIterator {
public:
    TIterator(int n, T *arr);
    T &operator*();
    T &operator->();
    void operator++();
    TIterator operator++(int);
    bool operator==(TIterator const &i);
    bool operator!=(TIterator const &i);
    ~TIterator() = default{};
private:
    size_t _n;
    T *_arr;
};
//TIterator.cpp
#include "TIterator.h"
template <class T>
TIterator:: TIterator(int n, T *arr) {
    _n = n;
    _arr = arr;
}
template <class T>
T & TIterator::operator*() {
    return _arr[_n];
}
template <class T>
T & TIterator::operator->() {
    return _arr[_n];
}
template <class T>
void TIterator::operator++() {
    _n++;
}
template <class T>
TIterator TIterator::operator++(int) {
    ++(*this);
    return *this;
}
template <class T>
bool TIterator::operator==(TIterator const &i) {
    return _n == i._n &&
           _arr == i._arr;
}
template <class T>
bool TIterator::operator!=(TIterator const &i) {
```

```
48 ||     return !(*this == i);
49 || }
```

3 Выводы

В данной лабораторной работе я получил навыки программирования итераторов на языке C++, закрепил навык работы с шаблонами классов. Контейнеры очень удобны в спользовании для реализации различных видов алгоритмов внутри контейнера, к примеру сортировка.