

**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: Болотин А.Н.  
Преподаватель: Поповкин А. В.  
Группа: 08-207  
Вариант: 19  
Дата:  
Оценка:  
Подпись:

**Москва, 2017**

# Лабораторная работа №5

## 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 содержать объекты используя `std::shared_ptr<...>`.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется 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 расположены в отдельных файлах: отдельно заголовки (`.h`), отдельно описание методов (`.cpp`).

**Фигуры:** Прямоугольник, ромб, трапеция  
**Контейнер:** N-дерево

### 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.
- Двухнаправленный 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 Исходный код

Описание классов фигур и класса-контейнера остается неизменным.

```
1 |
2 | template <class N, class T>
3 | class TreeIterator {
4 | public:
5 |     TreeIterator(std::shared_ptr<N> n) {
6 |         cur = n;
7 |         q = new std::queue<std::shared_ptr<N>>;
8 |         q->push(cur);
9 |     }
10 |
11 |     TreeIterator(TreeIterator &n) {
12 |         q = n.q;
13 |         cur = n.cur;
14 |     }
15 |
16 |     std::shared_ptr<T> operator * () {
```

```

17     return cur->getFigure();
18 }
19
20 std::shared_ptr<T> operator -> () {
21     return cur->getFigure();
22 }
23
24 void operator++() {
25     if (!cur) {
26         return;
27     }
28     else {
29         q->pop();
30         for (auto n : cur) {
31             q->push(n);
32         }
33         if (!q->empty()) {
34             cur = q->front();
35         }
36         else {
37             cur.reset();
38         }
39     }
40 }
41
42 TreeIterator operator++ (int) {
43     TIterator cur(*this);
44     ++(*this);
45     return cur;
46 }
47
48 bool operator== (const TreeIterator &i) {
49     return (cur == i.cur);
50 }
51
52 bool operator!= (const TreeIterator &i) {
53     return !(cur == i.cur);
54 }
55
56 private:
57     std::shared_ptr<N> cur;
58     std::queue<std::shared_ptr<N>> * q;
59 };

```

## 5 Выводы

Был разработан итератор для N-дерева. Его началом служит корень, концом - nullptr. Объяснение принципа работы займет много времени, но он становится очевиден при прочтении функции getNext() класса TreeItem. Итераторы - удобный инструмент для перебора всех элементов структуры. Благодаря им можно значительно упростить нагромождения кода.