

## 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:** треугольник, квадрат, прямоугольник.

**Контейнер:** связный список.

# 1 Введение

Итератор — это объект, который может выполнять итерацию элементов в контейнере STL и предоставлять доступ к отдельным элементам. Вы можете использовать итераторы явно, с помощью члена и глобальных функций, таких как `begin()` и `end()`, а также операторов `++` и `--` для перемещения вперед или назад. Вы можете также использовать итераторы неявно, с циклом `range-for` или (для некоторых типов итераторов) подстрочным оператором `[]`.

Принцип работы итераторов очень похожий на работу указателей: для получения значения также используется оператор разыменования, операции инкремента и декремента обеспечивают доступ в прямом и обратном направлении соответственно.

## 2 Код программы

### TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <memory>
#include <iostream>

template <class node, class T>
class TIterator
{
public:
    TIterator(std::shared_ptr<node>n) {
        node_ptr = n;
    }

    std::shared_ptr<T>operator* () {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T>operator->() {
        return node_ptr->GetFigure();
    }

    void operator++() {
        node_ptr = node_ptr->GetNext();
    }

    TIterator operator++ (int) {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator == (const TIterator &i) {
        return (node_ptr == i.node_ptr);
    }
};
```

```

}

bool operator!= (const TIterator &i) {
return !(*this == i);
}

private:
std::shared_ptr<node>node_ptr;
};

#endif

```

### 3 Вывод программы:

```

-----
-----МЕНЮ-----
|1-Добавить треугольник      |
|2-Добавить прямоугольник    |
|3-Добавить квадрат          |
|4-Удалить фигуру            |
|5-Распечатать список        |
|6-Выход                     |
Выберете действие:
2
Введите значение a:1
Введите значение b:1
Введите индекс: 0
Список создан
Выберете действие:
1
Введите значение a:1
Введите значение b:1
Введите значение c:2
Введите индекс: 0
Список создан
Выберете действие:
3
Введите значение a:2
Введите индекс: 0
Список создан
Выберете действие:
2
Введите значение a:3
Введите значение b:4
Введите индекс: 0
Список создан
Выберете действие:
5

```

Прямоугольник со сторонами [3,4]  
Квадрат со стороной [2]  
Треугольник со сторонами [1,1,2]  
Прямоугольник со сторонами [1,1]  
Выберете действие:  
4  
Введите индекс: 0  
Выберете действие:  
5  
Квадрат со стороной [2]  
Треугольник со сторонами [1,1,2]  
Прямоугольник со сторонами [1,1]  
Выберете действие:  
6

## 4 Вывод

В данной лабораторной работе я применила итератор для списка. Основное преимущество итераторов, бесспорно, заключается в том, что они помогают систематизировать код и повышают коэффициент его повторного использования. Один раз реализовав некоторый алгоритм, использующий итераторы, его можно использовать с любым типом контейнера.