Лабораторная работа №5

- Закрепление навыков работы с шаблонами классов.
- Построение итераторов для динамических структур данных.

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: октагон, квадрат, треугольник.
Контейнер: связный список.

1 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).

2 Исходный код

Описание классов фигур и класса-контейнера остается неизменным.

```
1
2  template <class N, class T>
3  class TIterator
4  {
5  public:
6      TIterator(std::shared_ptr<N> n) {
7          cur = n;
8      }
9
10     std::shared_ptr<T> operator* () {
11         return cur->GetFigure();
12     }
13
14     std::shared_ptr<T> operator-> () {
15         return cur->GetFigure();
16     }
```

```
17
18      void operator++() {
19          cur = cur->GetNext();
20      }
21
22      TIterator operator++ (int) {
23          TIterator cur(*this);
24          ++(*this);
25          return cur;
26      }
27
28      bool operator== (const TIterator &i) {
29          return (cur == i.cur);
30      }
31
32      bool operator!= (const TIterator &i) {
33          return (cur != i.cur);
34      }
35
36  private:
37      std::shared_ptr<N> cur;
38  };
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab5$ ./run
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
1
Enter side A: 2
Enter side B: 3
Enter side C: 4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
```

```
5) Print list
6) Print list using iterator
0) Exit
2
Enter side: 10
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
3
Enter side: 10
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
5
idx: 0   Side A = 2,side B = 3,side C = 4,type: Triangle

idx: 1   Side = 10,type: Foursquare

idx: 2   Side = 10,type: Octagon


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
2
Enter side: 19
```

```
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
6
List is printed using iterator
Side A = 2,side B = 3,side C = 4,type: Triangle
Side = 10,type: Foursquare
Side = 10,type: Octagon
Side = 19,type: Foursquare
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
0
```

4 Выводы

В данной лабораторной работе мне было необходимо создать итератор по списку. С этого времени перемещение по списку очень удобно.