Лабораторная работа №7

Целью лабораторной работы является:

- Создание сложных динамических структур данных.
- Закрепление принципа OCP.

Задача: Необходимо реализовать динамическую структуру данных – "Хранилище объектов"и алгоритм работы с ней. "Хранилище объектов"представляет собой контейнер стек. Каждым элементом контейнера является динамическая структура список. Таким образом, у нас получается контейнер в контейнере. Элементов второго контейнера является объект-фигура, определенная вариантом задания.
При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5. Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня.
Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта. При удалении объектов должно выполяться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалиться.

Фигуры: октагон, квадрат, треугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: связный список.

1 Описание

Принцип открытости/закрытости (OCP) – принцип ООП, устанавливающий следующее положение: "программные сущности (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменения".
Контейнер в программировании – структура (АТД), позволяющая инкапсулировать в себе объекты любого типа. Объектами (переменными) контейнеров являются коллекции, которые уже могут содержать в себе объекты определенного типа.
Например, в языке C++, std::list (шаблонный класс) является контейнером, а объект его класса-конкретизации, как например, std::list<int> mylist является коллекцией.

2 Исходный код

Описание классов фигур и класса-контейнера списка остается неизменным.

TStack.hpp	
TStack();	Конструктор класса
void Push(const O&);	Добавление элемента в список
void Print();	Печать списка
void RemoveByType(const int&);	Удаление из списка элементов одного типа
void RemoveLesser(const double&);	Удаление из списка элементов, площадь которых меньше, чем заданная
virtual ~TStack();	Деконструктор класса

```
1
2  template <typename Q, typename O> class TNList
3  {
4  private:
5      class Node {
6      public:
7          Q data;
8          std::shared_ptr<Node> next;
9          Node();
10         Node(const O&);
11         int itemsInNode;
12     };
13
14     std::shared_ptr<Node> head;
15     int count;
16 public:
```

```
17      TNList();
18
19      void push(const O&);
20      void print();
21      void removeByType(const int&);
22      void removeLesser(const double&);
23  };
24
25  template <typename T> class TList
26  {
27  private:
28      class TNode {
29      public:
30          TNode();
31          TNode(const std::shared_ptr<T>&);
32          auto GetNext() const;
33          auto GetItem() const;
34          std::shared_ptr<T> item;
35          std::shared_ptr<TNode> next;
36
37          void* operator new(size_t);
38          void operator delete(void*);
39          static TAllocator nodeAllocator;
40      };
41
42      template <typename N, typename M>
43      class TIterator {
44      private:
45          N nodePtr;
46      public:
47          TIterator(const N&);
48          std::shared_ptr<M> operator* ();
49          std::shared_ptr<M> operator-> ();
50          void operator ++ ();
51          bool operator == (const TIterator&);
52          bool operator != (const TIterator&);
53      };
54
55      int length;
56
57      std::shared_ptr<TNode> head;
58      auto psort(std::shared_ptr<TNode>&);
59      auto pparsort(std::shared_ptr<TNode>& head);
60      auto partition(std::shared_ptr<TNode>&);
61
62  public:
63      TList();
64      bool PushFront(const std::shared_ptr<T>&);
65      bool Push(const std::shared_ptr<T>&, const int);
```

```
66      bool PopFront();
67      bool Pop(const int);
68      bool IsEmpty() const;
69      int GetLength() const;
70      auto& getHead();
71      auto&& getTail();
72      void sort();
73      void parSort();
74
75      TIterator<std::shared_ptr<TNode>, T> begin() {return TIterator<std::shared_ptr<
            TNode>, T>(head->next);};
76      TIterator<std::shared_ptr<TNode>, T> end() {return TIterator<std::shared_ptr<TNode
            >, T>(nullptr);};
77
78      template <typename A> friend std::ostream& operator<< (std::ostream&, TList<A>&);
79  };
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab7$ ./run
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
1
Enter side A: 3
Enter side B: 4
Enter side C: 5
Item was added
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
2
Enter side: 10
Item was added
```

```
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
2
Enter side: 8
Item was added
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
2
Enter side: 19
Item was added
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
3
Enter side: 18
Item was added
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
3
Enter side: 15
Item was added
Choose an operation:
```

```
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
3
Enter side: 20
Item was added
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
5
Side A = 3,side B = 4,side C = 5,type: Triangle
Side = 8,type: Foursquare
Side = 10,type: Foursquare
Side = 19,type: Foursquare
Side = 18,type: Octagon

Side = 15,type: Octagon
Side = 20,type: Octagon

Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
4
Enter criteria
1) by type
2) lesser than square
1
1) Foursquare
2) Octagon
3) Triangle
```

```
Enter type
3
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
5
Side = 8,type: Foursquare
Side = 10,type: Foursquare
Side = 19,type: Foursquare
Side = 18,type: Octagon

Side = 15,type: Octagon
Side = 20,type: Octagon

Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
4
Enter criteria
1) by type
2) lesser than square
2
Enter square
1000
Choose an operation:
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
5
Choose an operation:
```

```
1) Add Triangle
2) Add Foursquare
3) Add Octagon
4) Delete Figure
5) Print
0) Exit
0
```

4 Выводы

В данной лабораторной работе был запрограммирован контейнер в контейнере. То есть в контейнере первого уровня – списке хранятся контейнеры второго уровня – списки, внутри которых хранятся фигуры. Фигуры внутри контейнера второго уровня отсортированы по возрастанию площади. Если в ходе выполнения программы контейнер второго уровня полностью освобождается от фигур, то он удаляется из контейнера первого уровня.