# Лабораторная работа №2

**Задача:** Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровная, содержащий одну фигуру, согласно варианту задания
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream(«). Оператор должен распечатывать параметры фигуры.
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream(»). Оператор должен вводить параметры фигуры.
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен содержать объекты фигур "по значению"(не по ссылке).
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера.
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера.
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream(«).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

**Фигура:** трапеция.
**Контейнер:** связный список.

# 1 Описание

Динамические структуры данных используются в тех случаях, когда мы заранее не знаем, сколько памяти необходимо выделить для нашей программы – это выясняется только в процессе работы. В общем случае эта структура представляет собо отдельные элементы, связанные между собой с помощью ссылок. Каждый элемент состоит из двух областей памяти: поля данных и ссылок. Ссылки – это адреса других узлов того же типа, с которыми данный элемент логически связан. При добавлении нового элемента в такую структуру выделяется новый блок памяти и устанавливаются связи этого элемента с уже существующими.

Структура данных список является простейшим типом данных динамической структуры, состоящей из узлов. Каждый узел включает в себя в классическом варианте два поля: данные и указатель на следующий узел в списке. Элементы связного списка можно вставлять и удалять произвольным образом. Доступ к списку осуществляется через указатель, который содержит ядрес первого элемента списка, называемого головой списка.

Параметры в функцию могут передаваться одним из следующих способов: по значению и по ссылке. При передаче аргументов по значению компилятор создает временную копию объекта, который должен быть передан, и размещает его в области стековой памяти, предназначенной для хранения локальных объектов. Вызываемая функция оперирует именно с этой копией, не оказывая влияния на оригинал объекта. Прототипы функций, принимающих аргументы по значению, предусматривают в качестве параметров указание типа объекта,а не его адреса. Если же необходимо, чтобы функция модифицировала оригинал объекта, используется передача параметров по ссылке. При этом в функцию передается не сам объект, а только его адрес. Таким образом, все модификации в теле функции переданных ей по ссылке аргументов воздействуют на объект. Использование передачи адреса объекта весьма эффективный способ работы с большим числом данных. Кроме того, так как передается адрес, а не сам объект, существенно экономится стековая память.

# 2 Исходный код

| trapeze.cpp | |
|---|---|
| Trapeze(); | Конструктор класса |
| Trapeze(std::istream &is); | Конструктор класса из стандартного потока |

| | |
|---|---|
| Trapeze(const Trapeze& orig); | Конструктор копии класса |
| double Square(); | Площадь фигуры |
| void Print(); | Печать фигуры |
| ~Trapeze(); | Деконструктор класса |
| bool operator ==(const Trapeze &obj) const; | Переопределенный оператор сравнения |
| Trapeze& operator =(const Trapeze &obj); | Переопределенный оператор копирования |
| friend std::ostream& operator «(std::ostream &os, const Trapeze &obj); | Переопределенный оператор вывода в поток std::ostream |
| friend std::istream& operator »(std::istream &is, Trapeze &obj); | Переопределенный оператор ввода из потока std::istream |
| TListItem.cpp | |
| TListItem(const Trapeze &obj); | Конструктор класса |
| Trapeze GetFigure() const; | Получение фигуры из узла |
| TListItem* GetNext(); | Получение ссылки на следующий узел |
| TListItem* GetPrev(); | Получение ссылки на предыдущий узел |
| void SetNext(TListItem *item); | Установка ссылки на следующий узел |
| void SetPrev(TListItem *item); | Установка ссылки на предыдущий узел |
| friend std::ostream& operator«(std::ostream &os, const TListItem &obj); | Переопределенный оператор вывода в поток std::ostream |
| virtual ~TListItem(); | Деконструктор класса |
| TList.cpp | |
| TList(); | Конструктор класса |
| void Push(Trapeze &obj); | Добавление фигуры в список |
| Trapeze Pop(); | Получение фигуры из списка |
| const bool IsEmpty() const; | Проверка, пуст ли список |
| uint32t GetLength(); | Получение длины списка |
| friend std::ostream& operator«(std::ostream &os, const TList &list); | Переопределенный оператор вывода в поток std::ostream |
| virtual ~TList(); | Деконструктор класса |

```
1
2  class TList
3  {
4  public:
5      TList();
6      void Push(Trapeze &obj);
```

```
7     const bool IsEmpty() const;
8     uint32_t GetLength();
9     Trapeze Pop();
10    friend std::ostream& operator<<(std::ostream &os, const TList &list);
11    virtual ~TList();
12
13  private:
14    uint32_t length;
15    TListItem *head;
16
17    void PushFirst(Trapeze &obj);
18    void PushLast(Trapeze &obj);
19    void PushAtIndex(Trapeze &obj, int32_t ind);
20    Trapeze PopFirst();
21    Trapeze PopLast();
22    Trapeze PopAtIndex(int32_t ind);
23  };
24
25  class TListItem
26  {
27  public:
28    TListItem(const Trapeze &obj);
29
30    Trapeze GetFigure() const;
31    TListItem* GetNext();
32    TListItem* GetPrev();
33    void SetNext(TListItem *item);
34    void SetPrev(TListItem *item);
35    friend std::ostream& operator<<(std::ostream &os, const TListItem &obj);
36
37    virtual ~TListItem(){};
38
39  private:
40    Trapeze item;
41    TListItem *next;
42    TListItem *prev;
43  };
44
45  class Trapeze : public Figure
46  {
47  public:
48    Trapeze();
49    Trapeze(std::istream &is);
50    Trapeze(int32_t small_base, int32_t big_base, int32_t l_side, int32_t r_side);
51    Trapeze(const Trapeze &orig);
52
53    bool operator ==(const Trapeze &obj) const;
54    Trapeze& operator =(const Trapeze &obj);
55    friend std::ostream& operator <<(std::ostream &os, const Trapeze &obj);
```

```
56      friend std::istream& operator >>(std::istream &is, Trapeze &obj);
57
58      double Square() override;
59      void Print() override;
60      virtual ~Trapeze();
61
62  private:
63      int32_t small_base;
64      int32_t big_base;
65      int32_t l_side;
66      int32_t r_side;
67  };
```

# 3 Консоль

```
karma@karma:~/mai_study/OOP/lab2$ valgrind --leak-check=full ./run
==4059== Memcheck,a memory error detector
==4059== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==4059== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4059== Command: ./run
==4059==
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
The list is empty.

Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 5
Enter smaller base: 2
Enter left side: 1
Enter right side: 1
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
```

```
3) Print list
0) Exit
1
Enter bigger base:
4
Enter smaller base: 3
Enter left side: 3
Enter right side: 3
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 5
Enter smaller base: 5
Enter left side: 5
Enter right side: 5
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 2
Enter smaller base: 2
Enter left side: 2
Enter right side: 2
Enter index = 2
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 4 3 3)

idx: 1   (2 5 1 1)
```

```
idx: 2   (5 5 5 5)

idx: 3   (2 2 2 2)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 4 3 3)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
```

```
Enter bigger base: 3
Enter smaller base: 3
Enter left side: 3
Enter right side: 3
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 2
Enter smaller base: 2
Enter left side: 2
Enter right side: 2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 3 3 3)

idx: 1   (3 4 3 3)

idx: 2   (2 2 2 2)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
0
==4059==
==4059== HEAP SUMMARY:
==4059==     in use at exit: 72,704 bytes in 1 blocks
==4059==   total heap usage: 9 allocs,8 frees,75,040 bytes allocated
==4059==
==4059== LEAK SUMMARY:
```

```
==4059==    definitely lost: 0 bytes in 0 blocks
==4059==    indirectly lost: 0 bytes in 0 blocks
==4059==      possibly lost: 0 bytes in 0 blocks
==4059==    still reachable: 72,704 bytes in 1 blocks
==4059==         suppressed: 0 bytes in 0 blocks
==4059== Reachable blocks (those to which a pointer was found) are not shown.
==4059== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==4059==
==4059== For counts of detected and suppressed errors,rerun with: -v
==4059== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

# 4 Выводы

В этой лабораторной было предложено написать свою структуру данных. В моем случае это список. Я сделала его двунаправленным для облегчения перемещения по списку. К тому же в будущем будет удобно (в отличие от однонаправленного) писать для такого списка итератор. Конечно, все эти контейнеры есть в стандартной библиотеке шаблонов, но любому профессиональному программисту не составит труда реализовать свой список, стек, очередь или любую другую широко используемую структуру данных. Фигуры передаются в список "по значению чтобы в них хранился сам объект, а не его копия.