Лабораторная работа №2

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

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: Октагон.
Контейнер: связный список.

1 Описание

Динамические структуры данных используют в тех случаях, когда не известно, сколько памяти необходимо выделить для нашей программы – это выясняется только в процессе работы. В общем случае эта структура представляет собой 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

Octagon.cpp	
Octagon();	Конструктор класса
Octagon(std::istream &is);	Конструктор класса из стандартного потока

Octagon(const Octagon& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Octagon();	Деконструктор класса
bool operator ==(const Octagon &obj) const;	Переопределенный оператор сравнения
Octagon& operator =(const Octagon &obj);	Переопределенный оператор копирования
friend std::ostream& operator «(std::ostream &os, const Octagon &obj);	Переопределенный оператор вывода в поток std::ostream
friend std::istream& operator »(std::istream &is, Octagon &obj);	Переопределенный оператор ввода из потока std::istream
TListItem.cpp	
TListItem(const Octagon &obj);	Конструктор класса
Octagon GetFigure() const;	Получение фигуры из узла
TListItem* GetNext();	Получение ссылки на следующий узел
void SetNext(TListItem *item);	Установка ссылки на следующий 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(Octagon &obj);	Добавление фигуры в список
Octagon 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  public:
4   TList();
5   TList(const TList& orig);
6   TList(Octagon& item);
7
8   bool push(Octagon &obj);
```

```
 9    const bool empty() const;
10    int getLength();
11    TListItem* getHead();
12    Octagon pop();
13    friend std::ostream& operator<<(std::ostream& os, const TList& list);
14    ~TList();
15
16   private:
17    int length;
18    TListItem *head;
19  };
20
21  class TListItem {
22   public:
23    TListItem(const Octagon& obj);
24    TListItem(const TListItem& orig);
25
26    TListItem* getNext();
27    Octagon getItem() const;
28    Octagon remove();
29    void setNext(TListItem* item);
30
31    ~TListItem(){};
32
33   private:
34    Octagon item;
35    TListItem *next;
36  };
37
38  class Octagon: public Figure {
39   public:
40    Octagon();
41    Octagon(const Octagon &obj);
42    Octagon(double i);
43    Octagon(std::istream &is);
44
45    Octagon operator++();
46    Octagon operator+(const Octagon& obj) const;
47    friend std::ostream& operator<<(std::ostream& os, const Octagon& obj);
48    friend std::istream& operator>>(std::istream& is, Octagon& obj);
49    bool operator==(const Octagon& obj) const;
50    Octagon operator=(const Octagon& obj);
51
52    double Square() override;
53    void Print() override;
54    void setParams(std::istream &is);
55    ~Octagon() {};
56
57   private:
```

```
58 ||   double side_a;
59 || };
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab2$ ./a.out
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
1
Enter an octagon's side
10
Item was added
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
1
Enter an octagon's side
20
Item was added
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
1
Enter an octagon's side
30
Item was added
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
2
Octagon's side is 10
Octagon's side is 20
Octagon's side is 30
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
```

```
0) Exit
3
Item was removed. It contained:
Octagon's side is 30
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
2
Octagon's side is 10
Octagon's side is 20
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
3
Item was removed. It contained:
Octagon's side is 20
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
2
Octagon's side is 10
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
3
Item was removed. It contained:
Octagon's side is 10
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
3
The list is empty
Item was not removed
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
```

```
0) Exit
2
The list is empty!
1) Insert an octagon
2) Print list
3) Delete the last octagon from list
0) Exit
0
```

4 Выводы

Во данной лабораторной работе необходимо было реализовать сложную структуру данных – односвязный список. Объекты передаются в методы контейнера по значению. Это обладает преимуществом: отсутствие создания копии.