Лабораторная работа №3

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Знакомство с умными указателями.

Задача: Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры, согласно варианту задания Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен содержать объекты, используя std::shared ptr<...>.
- Класс-контейнер должен иметь метод по добалению 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 ostream.
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

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

1 Описание

Умный указатель – класс (обычно шаблонный), имитирующий интерфейс обычного указателя и добавляющий некую новую функциональность, например, проверку границ при доступе или очистку памяти.
Существует 3 вида умных указателей стандартной библиотеки C++:

- unique ptr – обеспечивает, чтобы у базового указателя был только один владелец. Может быть передан новому владельцу, но не может быть скопирован или сделан общим. Заменяет auto ptr, использовать который не рекомендуется.
- shared ptr – умный указатель с подсчитанными ссылками. Используется, когда необходимо присвоить один необработанный указатель нескольким владельцам, например, когда копия указателя возвращается из контейнера, но требуется сохранить оригинал. Необработанный указатель не будет удален до тех пор, пока все владельцы shared ptr не выйдут из области или не откажутся от владения.
- weak ptr – умный указатель для особых случаев использования с shared ptr. weak ptr предоставляет доступ к объекту, который принадлежит одному или нескольким экземплярам shared ptr, но не участвует в подсчете ссылок. Используется, когда требуется отслеживать объект, но не требуется, чтобы он оставался в активном состоянии.

2 Исходный код

TListItem.cpp	
TListItem(const std::sharedptr<Figure> &obj);	Конструктор класса
std::sharedptr<Figure> GetFigure() const;	Получение фигуры из узла
std::sharedptr<TListItem> GetNext();	Получение ссылки на следующий узел
void SetNext(std::sharedptr<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(std::sharedptr<Figure> &obj);	Добавление фигуры в список

std::sharedptr<Figure> 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(std::shared_ptr<Figure> &obj);
7      const bool IsEmpty() const;
8      uint32_t GetLength();
9      std::shared_ptr<Figure> Pop();
10     friend std::ostream& operator<<(std::ostream &os, const TList &list);
11     virtual ~TList();
12
13 private:
14     uint32_t length;
15     std::shared_ptr<TListItem> head;
16
17     void PushFirst(std::shared_ptr<Figure> &obj);
18     void PushLast(std::shared_ptr<Figure> &obj);
19     void PushAtIndex(std::shared_ptr<Figure> &obj, int32_t ind);
20     std::shared_ptr<Figure> PopFirst();
21     std::shared_ptr<Figure> PopLast();
22     std::shared_ptr<Figure> PopAtIndex(int32_t ind);
23 };
24
25 class TListItem
26 {
27 public:
28     TListItem(const std::shared_ptr<Figure> &obj);
29
30     std::shared_ptr<Figure> GetFigure() const;
31     std::shared_ptr<TListItem> GetNext();
32     void SetNext(std::shared_ptr<TListItem> item);
33     friend std::ostream& operator<<(std::ostream &os, const TListItem &obj);
34
35     virtual ~TListItem(){};
36
37 private:
38     std::shared_ptr<Figure> item;
39     std::shared_ptr<TListItem> next;
```

```
40 };
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/lab3$ ./run
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
1
Enter side A: 1
Enter side B: 2
Enter side C: 3
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
2
Enter side: 5
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
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
0) Exit
```

```
5
idx: 0   Side A = 1,side B = 2,side C = 3,type: Triangle

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

idx: 1   Side = 5,type: Foursquare


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
3
Enter side: 123
Choose an operation:
1) Add triangle
2) Add foursquare
```

```
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
5
idx: 0   Side A = 1,side B = 2,side C = 3,type: Triangle

idx: 1   Side = 5,type: Foursquare

idx: 2   Side = 123,type: Octagon


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
```

```
5) Print list
0) Exit
5
The list is empty.

Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
0) Exit
0
```

4 Выводы

В данной лабораторной работе необходимо было работь с технологией умных указателей. Контейнер первого уровня был переписан с использованием shared ptr. Использование умных указателей значительно упрощает жизнь программисту.