Лабораторная работа №3

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: треугольник, квадрат, прямоугольник
Контейнер: связный список.

1 Введение

Умные указатели призваны для борьбы с утечками памяти,которые сложно избежать в больших проектах. Они особенно удобны в местах,где возникают исключения,так как при последних происходит процесс раскрутки стека и уничтожаются локальные объекты. В случае обычного указателя - уничтожится переменная-указатель,при этом ресурс останется не освобожденным. В случае умного указателя - вызовется деструктор,который и освободит выделенный ресурс.

В новом стандарте появились следующие умные указатели: unique_ptr,shared_ptr и weak_ptr.
Все они объявлены в заголовочном файле <memory>

unique_ptr -этот указатель пришел на смену старому и проблематичному auto_ptr. Основная проблема последнего заключается в правах владения. Объект этого класса теряет права владения ресурсом при копировании.

shared_ptr-это самый популярный и самый широкоиспользуемый умный указатель. В отличии от рассмотренных выше указателей,shared_ptr реализует подсчет ссылок на ресурс. Ресурс освободится тогда,когда счетчик ссылок на него будет равен 0.

weak_ptr -данный класс позволяет разрушить циклическую зависимость,которая,несомненно,может образоваться при использовании shared_ptr.

2 Код программы

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "TList.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "FSquare.h"
#include <memory>



int main() {
setlocale(LC_ALL,"Russian");
int a;
TList list;
std::shared_ptr<Figure>in,del;
std::cout <<"-----------------------------------------" <<std::endl;
std::cout <<"-------------------МЕНЮ-------------------" <<std::endl;
std::cout <<"|1-Добавить треугольник                  |" <<std::endl;
std::cout <<"|2-Добавить прямоугольник                |" <<std::endl;
std::cout <<"|3-Добавить квадрат                      |" <<std::endl;
std::cout <<"|4-Удалить фигуру                        |" <<std::endl;
std::cout <<"|5-Расспечатать список                   |" <<std::endl;
std::cout <<"|6-Выход                                 |" <<std::endl;
```

```
do {
std::cout <<"Выберете действие:" <<std::endl;
if (!(std::cin >>a)) {
std::cin.clear();
while (std::cin.get() != '\n');
}
switch (a) {
case 1: {
in = std::make_shared<Triangle>(std::cin);
list.Insert(in);
break;
}
case 2: {
in = std::make_shared<Rectangle>(std::cin);
list.Insert(in);
break;
}
case 3: {
in = std::make_shared<FSquare>(std::cin);
list.Insert(in);
break;
}
case 4:
del=list.Delete();
break;
case 5:
std::cout <<list;
break;
case 6: {
break;
}
default: std::cout <<"Неверный ввод. Попробуйте снова" <<std::endl;
break;
}
} while (a != 6);
return 0;
}
```

TList.h

```
#ifndef TLIST_H
#define TLIST_H
#include "FSquare.h"
#include "TListItem.h"
#include "Triangle.h"
#include "Rectangle.h"

class TList {
public:
TList();
```

```
void Insert(std::shared_ptr<Figure>&obj);
bool IsEmpty() const;
std::shared_ptr<Figure>Delete();
friend std::ostream& operator<<(std::ostream& os,const TList& list);
virtual ~TList();
int GetSize();
private:
std::shared_ptr<TListItem>head;
int size;

void PushFirst(std::shared_ptr<Figure>&obj);
void PushLast(std::shared_ptr<Figure>&obj);
void PushAtIndex(std::shared_ptr<Figure>&obj,int n);
std::shared_ptr<Figure>PopFirst();
std::shared_ptr<Figure>PopLast();
std::shared_ptr<Figure>PopAtIndex(int n);
};
#endif
```

TList.cpp

```
#include "TList.h"
#include <iostream>

TList::TList() :head(nullptr),size(0) {
}

void TList::Insert(std::shared_ptr<Figure>&obj) {
int n;
std::cout <<"Введите индекс: ";
std::cin >>n;
if (n <0 || n >this->GetSize()) {
std::cout <<"Такого индекса нет.\n";
return;
}
if (n == 0) {
this->PushFirst(obj);
}
else if (n == this->GetSize() -1) {
this->PushLast(obj);
}
else {
this->PushAtIndex(obj,n);
}
++size;
}

void TList::PushAtIndex(std::shared_ptr<Figure>&obj,int n) {
std::shared_ptr<TListItem>p = std::make_shared<TListItem>(obj);
std::shared_ptr<TListItem>tmp = this->head;
```

```
for (int i = 1;i <n;i++) {
tmp = tmp->GetNext();
}
p->SetNext(tmp->GetNext());
p->SetPrev(tmp);
tmp->SetNext(p);
tmp->GetNext()->SetPrev(p);
}

void TList::PushLast(std::shared_ptr<Figure>&obj)
{
std::shared_ptr<TListItem>newItem = std::make_shared<TListItem>(obj);
std::shared_ptr<TListItem>tmp = this->head;

while (tmp->GetNext() != nullptr) {
tmp = tmp->GetNext();
}
tmp->SetNext(newItem);
newItem->SetPrev(tmp);
newItem->SetNext(nullptr);
}

void TList::PushFirst(std::shared_ptr<Figure>&obj)
{
std::shared_ptr<TListItem>newItem = std::make_shared<TListItem>(obj);
std::shared_ptr<TListItem>oldHead = this->head;
this->head = newItem;
if (oldHead != nullptr) {
newItem->SetNext(oldHead);
oldHead->SetPrev(newItem);
}
}

int TList::GetSize()
{
return this->size;
}

bool TList::IsEmpty() const
{
return head == nullptr;
}

std::shared_ptr<Figure>TList::Delete()
{
int n = 0;
std::shared_ptr<Figure>f;
std::cout <<"Введите индекс: ";
std::cin >>n;
```

```
if (n >this->GetSize() -1 || n <0 || this->IsEmpty()) {
std::cout <<"Неверный ввод.\n";
return f;
}
if (n == 0) {
f = this->PopFirst();
}
else if (n == this->GetSize() -1) {
f = this->PopLast();
}
else {
f = this->PopAtIndex(n);
}
--size;
return f;
}

std::shared_ptr<Figure>TList::PopAtIndex(int n)
{
std::shared_ptr<TListItem>tmp = this->head;
for (int i = 0; i <n -1; ++i) {
tmp = tmp->GetNext();
}
std::shared_ptr<TListItem>rem = tmp->GetNext();
std::shared_ptr<Figure>res = rem->GetFigure();
std::shared_ptr<TListItem>nextItem = rem->GetNext();
tmp->SetNext(nextItem);
nextItem->SetPrev(tmp);
return res;
}

std::shared_ptr<Figure>TList::PopFirst()
{
if (this->GetSize() == 1) {
std::shared_ptr<Figure>res = this->head->GetFigure();
this->head = nullptr;
return res;
}
std::shared_ptr<TListItem>tmp = this->head;
std::shared_ptr<Figure>res = tmp->GetFigure();
this->head = this->head->GetNext();
this->head->SetPrev(nullptr);
return res;
}

std::shared_ptr<Figure>TList::PopLast()
{
if (this->GetSize() == 1) {
std::shared_ptr<Figure>res = this->head->GetFigure();
```

```
this->head = nullptr;
return res;
}
std::shared_ptr<TListItem>tmp = this->head;
while (tmp->GetNext()->GetNext()) {
tmp = tmp->GetNext();
}
std::shared_ptr<TListItem>rem = tmp->GetNext();
std::shared_ptr<Figure>res = rem->GetFigure();
tmp->SetNext(rem->GetNext());
return res;
}

std::ostream& operator<<(std::ostream& os,const TList& list) {
if (list.IsEmpty())
{
os <<"Список пуст." <<std::endl;
return os;
}
std::shared_ptr<TListItem>item = list.head;
for (int i = 0;item;++i) {
item->GetFigure()->Print();
os <<std::endl;
item = item->GetNext();
}
/*while (item != nullptr) {
os <<item->GetFigure();
item = item->GetNext();
}*/
return os;
}

TList::~TList()
{
while (!this->IsEmpty()) {
this->PopFirst();
--size;
}
}
```

TListItem.cpp

```
#include <iostream>
#include "TListItem.h"

TListItem::TListItem(const std::shared_ptr<Figure>&obj) {
this->item = obj;
this->next = nullptr;
this->prev = nullptr;
std::cout <<"Список создан" <<std::endl;
```

```
}

std::shared_ptr<TListItem>TListItem::GetNext() {
return this->next;
}

std::shared_ptr<TListItem>TListItem::GetPrev()
{
return this->prev;
}

void TListItem::SetNext(std::shared_ptr<TListItem>item) {
this->next = item;
}

void TListItem::SetPrev(std::shared_ptr<TListItem>prev)
{
this->prev = prev;
}

std::shared_ptr<Figure>TListItem::GetFigure() const {
return this->item;
}
```

TListItem.h

```
#ifndef TLISTITEM_H
#define TLISTITEM_H

#include <memory>
#include "FSquare.h"
#include "Triangle.h"
#include "Rectangle.h"


class TListItem {
public:
TListItem(const std::shared_ptr<Figure>&obj);
std::shared_ptr<TListItem>GetNext();
std::shared_ptr<TListItem>GetPrev();
void SetNext(std::shared_ptr<TListItem>item);
void SetPrev(std::shared_ptr<TListItem>prev);
//friend std::ostream& operator<<(std::ostream &os,const TListItem &obj);
std::shared_ptr<Figure>GetFigure() const;
virtual ~TListItem(){};
private:
std::shared_ptr<Figure>item;
std::shared_ptr<TListItem>next;
std::shared_ptr<TListItem>prev;
};
```

```
#endif
```

3 Вывод программы:

```
----------------------------------------
------------------МЕНЮ------------------
|1-Добавить треугольник                |
|2-Добавить прямоугольник              |
|3-Добавить квадрат                    |
|4-Удалить фигуру                      |
|5-Расспечатать список                 |
|6-Выход                               |
Выберете действие:
1
Введите значение a:1
Введите значение b:1
Введите значение c:1
Введите индекс: 0
Список создан
Выберете действие:
3
Введите значение a:2
Введите индекс: 1
Список создан
Выберете действие:
2
Введите значение a:1
Введите значение b:1
Введите индекс: 2
Список создан
Выберете действие:
2
Введите значение a:2
Введите значение b:2
Введите индекс: 0
Список создан
Выберете действие:
5
Прямоугольник со сторонами [2,2]

Треугольник со сторонами [1,1,1]

Квадрат со стороной [2]

Прямоугольник со сторонами [1,1]

Выберете действие:
```

```
4
Введите индекс: 0
Выберете действие:
5
Треугольник со сторонами [1,1,1]

Квадрат со стороной [2]

Прямоугольник со сторонами [1,1]

Выберете действие:
6
```

4 Вывод

В данной лабораторной работе я познакомилась с умными указателями, которые оказались очень полезными и удобными. При правильном и аккуратном использовании умные указатели могут существенно облегчить жизнь C++-программисту, однако следует очень внимательно изучить их поведение, их сильные и слабые стороны.