Лабораторная работа №4

Задача: Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

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.
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять вводить произвольное количество фигур и добавлять их в контейнер распечатывать содержимое контейнера, удалять фигуры из контейнера.

1 Введение

Шаблоны классов - обобщенное определение некоторого семейства классов, имеющих схожую структуру, но различных в смысле используемых типов или констант.Шаблоны обеспечивают простой способ введения разного рода общих концепций и простые методы их совместного использования. Получающиеся в результате классы и функции сопоставимы по времени выполнения и требованиям к памяти с написанным вручную специализированным кодом.
Шаблоны обеспечивают непосредственную поддержку обобщенного программирования, т.е. программирования с использованием типов в качестве параметров. Механизм шаблонов в C++ допускает использование типа в качестве параметра при определении класса или функции. Шаблон зависит только от тех свойств параметра-типа, которые он явно использует, и не требует, чтобы различные типы, используемые в качестве параметров, были связаны каким-либо другим образом.
Шаблоны обеспечивают эффективный способ генерации кода из сравнительно коротких определений. Поэтому требуется некоторая осмотрительность во избежание заполнения памяти почти идентичными определениями функций.

2 Код программы

TList.h

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

template<class T>
class TList {
public:
TList();
void Insert(std::shared_ptr<T>&obj);
bool IsEmpty() const;
std::shared_ptr<T>Delete();
template <class A>friend std::ostream& operator<<(std::ostream& os,const TList<A>&list
virtual ~TList();
int GetSize();
private:
```

```
std::shared_ptr<TListItem<T>>head;
int size;

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

TList.cpp

```
#include "TList.h"
#include <iostream>
#include "Figure.h"
template class TList<Figure>;
template std::ostream& operator<<(std::ostream &out,const TList<Figure>&obj);

template<class T>
TList<T>::TList() :head(nullptr),size(0) {
}

template<class T>
void TList<T>::Insert(std::shared_ptr<T>&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;
}

template<class T>
void TList<T>::PushAtIndex(std::shared_ptr<T>&obj,int n) {
std::shared_ptr<TListItem<T>>p = std::make_shared<TListItem<T>>(obj);
std::shared_ptr<TListItem<T>>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);
}

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

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

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

```
template<class T>
int TList<T>::GetSize()
{
return this->size;
}

template<class T>
bool TList<T>::IsEmpty() const
{
return head == nullptr;
}

template<class T>
std::shared_ptr<T>TList<T>::Delete()
{
int n = 0;
std::shared_ptr<T>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;
}

template<class T>
std::shared_ptr<T>TList<T>::PopAtIndex(int n)
{
std::shared_ptr<TListItem<T>>tmp = this->head;
```

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

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

template<class T>
std::shared_ptr<T>TList<T>::PopLast()
{
if (this->GetSize() == 1) {
std::shared_ptr<T>res = this->head->GetFigure();
this->head = nullptr;
return res;
}
std::shared_ptr<TListItem<T>>tmp = this->head;
while (tmp->GetNext()->GetNext()) {
tmp = tmp->GetNext();
}
std::shared_ptr<TListItem<T>>rem = tmp->GetNext();
std::shared_ptr<T>res = rem->GetFigure();
tmp->SetNext(rem->GetNext());
```

```
return res;
}

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


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

TListItem.h

```
#ifndef TLISTITEM_H
#define TLISTITEM_H

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

template <class T>
class TListItem {
public:
```

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

#endif
```

TListItem.cpp

```
#include <iostream>
#include "TListItem.h"
#include "Figure.h"
template class TListItem <Figure>;
template std::ostream& operator<<(std::ostream &out,const TListItem<Figure>&obj);

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

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

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

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

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

template <class T>
std::shared_ptr<T>TListItem<T>::GetFigure() const {
return this->item;
}

template <class T>
std::ostream& operator<<(std::ostream &os,const TListItem<T>&obj)
{
os <<obj.item <<std::endl;
return os;
}
```

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:2
Введите значение b:3
Введите значение c:3
Введите индекс: 0
```

```
Список создан
Выберете действие:
2
Введите значение a:3
Введите значение b:4
Введите индекс: 1
Список создан
Выберете действие:
3
Введите значение a:2
Введите индекс: 2
Список создан
Выберете действие:
5
Треугольник со сторонами [2,3,3]

Прямоугольник со сторонами [3,4]

Квадрат со стороной [2]

Выберете действие:
4
Введите индекс: 1
Выберете действие:
5
Треугольник со сторонами [2,3,3]

Квадрат со стороной [2]

Выберете действие:
6
```

4 Вывод

В данной лабораторной работе я познакомилась с таким важным и полезным инструментом C++, как шаблоны.Шаблоны классов помогут вам избавиться от дублирования кода программы, если вам необходимы объекты похожих классов, которые отличаются только типом. Поскольку шаблоны классов могут быть сложными, они могут вас смутить. Когда вы определяете ваш класс, начните с определения, как будто бы вы создаете класс для конкретного типа. После того как вы полностью

опишете класс, определите какие элементы необходимо изменить, чтобы работать с объектами различных типов.