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: треугольник.
Контейнер: связный список.

1 Введение

Развитие теории алгоритмов, а также появление автоматических вычислителей вызвали интерес к таким структурам данных, которые сами изменяются в процессе выполнения алгоритма. Такие структуры данных стали называть динамическими. Как правило, динамические структуры данных рассматривают как набор элементов, некоторым образом связанных друг с другом, количество которых может меняться в процессе работы с такой структурой. Для динамических структур определяют операции, которые изменяют именно организацию (связи) элементов.
В данной лабораторной работе у меня реализован двухсвязный список.В двухсвязном списке каждый элемент имеет поля с данными и два указателя: один указатель хранит адрес предшествующего элемента списка, второй - адрес последующего элемента. Вполне естественно для работы с двухсвязным списком использовать два указателя, хранящие адреса начала и конца такого списка.

2 Код программы

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "TList.h"
#include "Triangle.h"
#include "TListItem.h"



int main() {
setlocale(LC_ALL,"Russian");
int a;
TList list;
Triangle tr;
std::cout <<"----------------------------------------" <<std::endl;
std::cout <<"------------------МЕНЮ------------------" <<std::endl;
std::cout <<"|1-Добавить фигуру                      |" <<std::endl;
std::cout <<"|2-Удалить фигуру                       |" <<std::endl;
std::cout <<"|3-Расспечатать список                  |" <<std::endl;
std::cout <<"|4-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: {
std::cin >>tr;
list.Insert(tr);
break;
}
case 2: {
tr=list.Delete();
break;
}
case 3: {
std::cout <<list;
break;
}
case 4: {
break;
}
default: std::cout <<"Неверный ввод. Попробуйте снова" <<std::endl;
break;
}
} while (a != 4);
return 0;
}
```

Triangle.cpp

```
#include "Triangle.h"
#include <iostream>
#include <cmath>
Triangle::Triangle() : Triangle(0,0,0) {
}

Triangle::Triangle(size_t i,size_t j,size_t k) : side_a(i),side_b(j),side_c(k)
{
// std::cout <<"Создан треугольник со сторонами: " <<side_a <<"," <<side_b
<<"," <<side_c <<std::endl;
}
```

```
Triangle::Triangle(std::istream& is) {
is >>side_a;
is >>side_b;
is >>side_c;
}

Triangle::Triangle(const Triangle &orig) {
side_a = orig.side_a;
side_b = orig.side_b;
side_c = orig.side_c;
}

double Triangle::Square() {
std::cout <<"Площадь = ";
double p = double(side_a + side_b + side_c) / 2.0;
return sqrt(p * (p -double(side_a))*(p -double(side_b))*(p -double(side_c)));
}

Triangle& Triangle::operator=(const Triangle& right) {
if (this == &right)
return *this;
side_a = right.side_a;
side_b = right.side_b;
side_c = right.side_c;
return *this;
}

bool Triangle::operator == (const Triangle& other)
{
return side_a == other.side_a && side_b == other.side_b && side_c == other.side_c;
}

Triangle& Triangle::operator++() {
side_a++;
side_b++;
side_c++;
return *this;
}

Triangle operator+(const Triangle& left,const Triangle& right) {
return Triangle(left.side_a + right.side_a,left.side_b + right.side_b,left.side_c
```

```
+ right.side_c);
}

std::ostream& operator<<(std::ostream& os,const Triangle& obj) {
os <<"Треугольник со сторонами ";
os <<"[" <<obj.side_a <<","<<obj.side_b <<","<<obj.side_c <<"]" <<std::endl;
return os;
}

std::istream& operator>>(std::istream& is,Triangle& obj) {
std::cout <<"Введите значение a:";
while (!(is >>obj.side_a)) {
std::cout <<"Неверный ввод" <<std::endl;
is.clear();
while (std::cin.get() != '\n');
std::cout <<"Введите значение a:";
}
std::cout <<"Введите значение b:";
while (!(is >>obj.side_b)) {
std::cout <<"Неверный ввод" <<std::endl;
is.clear();
while (std::cin.get() != '\n');
std::cout <<"Введите значение b:";
}
std::cout <<"Введите значение c:";
while (!(is >>obj.side_c)) {
std::cout <<"Неверный ввод" <<std::endl;
is.clear();
while (std::cin.get() != '\n');
std::cout <<"Введите значение c";
}
return is;
}

Triangle::~Triangle() {

}
```

Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
```

```
#include <cstdlib>
#include <iostream>

class Triangle {
public:
Triangle();
Triangle(size_t i,size_t j,size_t k);
Triangle(std::istream& is);
Triangle(const Triangle &orig);

Triangle& operator++();
double Square();
friend Triangle operator+(const Triangle& left,const Triangle& right);
friend std::ostream& operator<<(std::ostream& os,const Triangle& obj);
friend std::istream& operator>>(std::istream& is,Triangle& obj);

Triangle& operator=(const Triangle& right);
bool operator==(const Triangle& other);

virtual ~Triangle();
private:
double side_a;
double side_b;
double side_c;
};
#endif
```

TListItem.h

```
#ifndef TLISTITEM_H
#define TLISTITEM_H

#include "Triangle.h"

class TListItem {
public:
TListItem(const Triangle& triangle);
TListItem* GetNext();
TListItem* GetPrev();
void SetNext(TListItem* item);
void SetPrev(TListItem* prev);
Triangle GetTriangle() const;
```

```
virtual ~TListItem();
private:
Triangle triangle;
TListItem *next;
TListItem *prev;
};

#endif
```

TListItem.cpp

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

TListItem::TListItem(const Triangle& triangle) {
this->triangle = triangle;
this->next = nullptr;
this->prev = nullptr;
std::cout <<"Список создан" <<std::endl;
}

TListItem* TListItem::GetNext() {
return this->next;
}

TListItem* TListItem::GetPrev()
{
return this->prev;
}

void TListItem::SetNext(TListItem* item) {
this->next = item;
}

void TListItem::SetPrev(TListItem *prev)
{
this->prev = prev;
}

Triangle TListItem::GetTriangle() const {
return this->triangle;
}
```

```
TListItem::~TListItem() {
std::cout <<"Фигура удалена" <<std::endl;
}
```

TList.cpp

```
#include "TList.h"

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

std::ostream& operator<<(std::ostream& os,const TList& list) {
if (!list.head)
{
os <<"Список пуст." <<std::endl;
}
TListItem *item = list.head;
while (item != nullptr) {
os <<item->GetTriangle();
item = item->GetNext();
}
return os;
}

void TList::Insert(Triangle &tr) {
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(tr);
}
else if (n == this->GetSize() -1) {
this->PushLast(tr);
}
else {
this->PushAtIndex(tr,n);
}
```

```
++size;
}

void TList::PushLast(Triangle &tr)
{
TListItem *newItem = new TListItem(tr);
TListItem *tmp = this->head;

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

void TList::PushFirst(Triangle &tr)
{
TListItem *newItem = new TListItem(tr);
TListItem *oldHead = this->head;
this->head = newItem;
if (oldHead != nullptr) {
newItem->SetNext(oldHead);
oldHead->SetPrev(newItem);
}
}


void TList::PushAtIndex(Triangle &triangle,int n) {
TListItem *p = new TListItem(triangle);
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);
}
```

```
Triangle TList::Delete()
{
int n = 0;
Triangle tr;
std::cout <<"Введите индекс: ";
std::cin >>n;
if (n >this->GetSize() -1 || n <0 || this->IsEmpty()) {
std::cout <<"Неверный ввод.\n";
return tr;
}
if (n == 0) {
tr = this->PopFirst();
}
else if (n == this->GetSize() -1) {
tr = this->PopLast();
}
else {
tr = this->PopAtIndex(n);
}
--size;
return tr;
}

Triangle TList::PopAtIndex(int n)
{
TListItem *tmp = this->head;
for (int i = 0; i <n -1; ++i) {
tmp = tmp->GetNext();
}
TListItem *rem = tmp->GetNext();
Triangle res = rem->GetTriangle();
TListItem *nextItem = rem->GetNext();
tmp->SetNext(nextItem);
nextItem->SetPrev(tmp);
delete rem;
return res;
}

Triangle TList::PopFirst()
{
if (this->GetSize() == 1) {
```

```
Triangle res = this->head->GetTriangle();
delete this->head;
this->head = nullptr;
return res;
}
TListItem *tmp = this->head;
Triangle res = tmp->GetTriangle();
this->head = this->head->GetNext();
this->head->SetPrev(nullptr);
delete tmp;
return res;
}

Triangle TList::PopLast()
{
if (this->GetSize() == 1) {
Triangle res = this->head->GetTriangle();
delete this->head;
this->head = nullptr;
return res;
}
TListItem *tmp = this->head;
while (tmp->GetNext()->GetNext()) {
tmp = tmp->GetNext();
}
TListItem *rem = tmp->GetNext();
Triangle res = rem->GetTriangle();
tmp->SetNext(rem->GetNext());
delete rem;
return res;
}

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

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

```
TList::~TList() {
TListItem* tmp;
while (head) {
tmp = head;
head = head->GetNext();
delete tmp;
}
}
```

TList.h

```
#ifndef TLIST_H
#define TLIST_H
#include "Triangle.h"
#include "TListItem.h"
class TList {
public:
TList();
void Insert(Triangle &triangle);
bool IsEmpty() const;
Triangle Delete();
friend std::ostream& operator<<(std::ostream& os,const TList& stack);
virtual ~TList();
int GetSize();
private:
TListItem *head;
int size;

void PushFirst(Triangle &tr);
void PushLast(Triangle &tr);
void PushAtIndex(Triangle &tr,int n);
Triangle PopFirst();
Triangle PopLast();
Triangle PopAtIndex(int n);
};
#endif
```

3 Вывод программы:

```
----------------------------------------
```

```
-------------------МЕНЮ-------------------
|1-Добавить фигуру                       |
|2-Удалить фигуру                        |
|3-Расспечатать список                   |
|4-Выход                                 |
Выберете действие:
1
Введите значение a:1
Введите значение b:1
Введите значение c:1
Введите индекс: 0
Список создан
Выберете действие:
1
Введите значение a:2
Введите значение b:2
Введите значение c:2
Введите индекс: 1
Список создан
Выберете действие:
1
Введите значение a:3
Введите значение b:4
Введите значение c:5
Введите индекс: 2
Список создан
Выберете действие:
3
Треугольник со сторонами [1,1,1]
Треугольник со сторонами [2,2,2]
Треугольник со сторонами [3,4,5]
Выберете действие:
2
Введите индекс: 1
Фигура удалена
Выберете действие:
7
Неверный ввод. Попробуйте снова
Выберете действие:
3
Треугольник со сторонами [1,1,1]
```

```
Треугольник со сторонами [3,4,5]
Выберете действие:
4
Фигура удалена
Фигура удалена
Для продолжения нажмите любую клавишу . . .
```

4 Вывод

В ходе данной лабораторной работы мною было реализован двухсвязный список. Многие структуры уже существуют в стандартной библиотеке шаблонов, но порой полезно самому уметь реализовывать различные структуры данных.