**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 5**

Тема: Основы работы с коллекциями: итераторы

Студент: Почечура Артемий Андреевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

**1.** **Постановка задачи**

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения (правильные многоугольники), т.е. равносторонними (кроме трапеции и прямоугольника). Для хранения координат фигур необходимо использовать шаблон std::pair.

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr). Опционально использование std::unique\_ptr;

2. В качестве параметра шаблона коллекция должна принимать тип данных - фигуры;

3. Реализовать forward\_iterator по коллекции;

4. Коллекция должны возвращать итераторы begin() и end();

5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);

6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);

7. При выполнении недопустимых операций (например выход за границы коллекции или удаление несуществующего элемента) необходимо генерировать исключения;

8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)

9. Коллекция должна содержать метод доступа:

* стек – pop, push, top;
* очередь – pop, push, top;
* список, Динамический массив – доступ к элементу по оператору [].

10. Реализовать программу, которая:

* позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;
* позволяет удалять элемент из коллекции по номеру элемента;
* выводит на экран введенные фигуры c помощью std::for\_each;
* выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

**2.** **Описание программы**

Реализована структура данных List, которая представляет из себя линейный односторонний список. Все взаимодействия со списком реализованы с помощью итераторов. Итератор Iterator также реализован как отдельная структура, для которой написано несколько функций. Список может содержать любой тип переменных, в моём случае была выбрана трапеция (по варианту). Класс трапеции Trapezoid в данной лабораторной также поддерживает любой тип переменных.

3. **Набор тестов и результаты их исполнения**

Тест 1:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Section, that join middles of bases, belong to axis Y (centre of section - point (0;0)).

Select a function: add

input index: 0

input first base, second base and height: 3 9 4

Element added

Select a function: add

input index: 1

input first base, second base and height: 4 4 4

Element added

Select a function: add

input index: 1

input first base, second base and height: 4 8 8

Element added

Select a function: print

3 9 4

4 8 8

4 4 4

Select a function: end

Тест 2:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Section, that join middles of bases, belong to axis Y (centre of section - point (0;0)).

Select a function: delete

input index: 0

Wrong iterator!

Select a function: add

input index: 1

input first base, second base and height: 2 2 2

Wrong iterator!

Select a function: add

input index: 0

input first base, second base and height: 2 2 2

Element added

Select a function: get

input index: 1

Wrong index in List!

Select a function: delete

input index: 1

Wrong iterator!

Select a function: end

Тест 3:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Section, that join middles of bases, belong to axis Y (centre of section - point (0;0)).

Select a function: add

input index: 0

input first base, second base and height: 2 3 4

Element added

Select a function: add

input index: 1

input first base, second base and height: 3 4 5

Element added

Select a function: add

input index: 1

input first base, second base and height: 4 5 6

Element added

Select a function: add

input index: 3

input first base, second base and height: 5 6 7

Element added

Select a function: print

2 3 4

4 5 6

3 4 5

5 6 7

Select a function: delete

input index: 2

Element deleted

Select a function: print

2 3 4

4 5 6

5 6 7

Select a function: get

input index: 2

Points: (2.5;3.5), (3;-3.5), (-3;-3.5), (-2.5;3.5)

Base 1: 5

Base 2: 6

Edge: 3.5

Square: 38.5

Select a function: end

**4.** **Листинг программы**

List.h:

//Определение структуры списка

#define LIST\_H

using namespace std;

#include "Iterator3.h"

#include "Trapezoid.h"

#include "Trapezoid.cpp"

#include <memory>

template<typename T>

class List{

private:

shared\_ptr<T\*> buf = make\_shared<T\*>(new T[2]);

size\_t sz = 0;

size\_t cap = 2;

public:

friend class Iterator<T>;

List();

List(size\_t n, const T& value = T());

List(const List<T>& orig);

size\_t size();

size\_t capacity();

void reserve(size\_t n);

void resize(size\_t n, const T& value = T());

void push\_back(const T& value = T());

void pop\_back();

bool empty();

T\* data();

T& front();

const T& front() const;

T& back();

const T& back() const;

T& operator[](size\_t i);

const T& operator[](size\_t i) const;

T& at(size\_t i);

const T& at(size\_t i) const;

Iterator<T> begin();

Iterator<T> end();

void insert(Iterator<T> pos, const T& value);

void erase(Iterator<T> pos);

~List();

};

#endif

List.cpp:

#include <cstddef>

#include <iostream>

#include <cstdlib>

#include "List3.h"

#include "Trapezoid.h"

using namespace std;

template<typename T>

List<T>::List(){}

template <typename T>

List<T>::List(size\_t n, const T& value){

resize(n, value);

}

template <typename T>

List<T>::List(const List<T>& orig){

buf = orig.data();

sz = orig.size();

cap = orig.capacity();

}

template <typename T>

size\_t List<T>::size(){

return sz;

}

template <typename T>

size\_t List<T>::capacity(){

return cap;

}

template <typename T>

void List<T>::reserve(size\_t n){

if(n <= cap) return;

T\* newbuf = new T[n];

//cout << newbuf << " " << sz << endl;

for(size\_t i = 0; i < sz; ++i){

//cout << i << endl;

newbuf[i] = at(i);

}

delete[] (\*buf.get());

buf = make\_shared<T\*>(newbuf);

cap = n;

}

template <typename T>

void List<T>::resize(size\_t n, const T& value){

if(n > cap) reserve(n);

size\_t i = sz;

try{

for(size\_t i = sz; i < n; ++i){

new (\*buf.get() + i) T(value);

}

} catch(...){

for(size\_t j = sz; j < i; ++j){

(\*buf.get() + j)->~T();

}

throw;

}

if(n < sz) sz = n;

}

template <typename T>

void List<T>::push\_back(const T& value){

insert(end(), value);

/\*if(cap == sz){

resize(2 \* cap);

}

new (\*buf.get() + sz) T(value);

++sz;\*/

}

template <typename T>

void List<T>::pop\_back(){

--sz;

(buf.get() + sz)->~T();

}

template <typename T>

bool List<T>::empty(){

return sz == 0;

}

template <typename T>

T\* List<T>::data(){

return (\*buf.get());

}

template <typename T>

T& List<T>::front(){

if(empty()){

throw out\_of\_range("List is empty!");

}

return (\*buf.get())[0];

}

template <typename T>

const T& List<T>::front() const{

if(empty()){

throw out\_of\_range("List is empty!");

}

return (\*buf.get())[0];

}

template <typename T>

T& List<T>::back(){

if(empty()){

throw out\_of\_range("List is empty!");

}

return (\*buf.get())[sz-1];

}

template <typename T>

const T& List<T>::back() const{

if(empty()){

throw out\_of\_range("List is empty!");

}

return (\*buf.get())[sz-1];

}

template <typename T>

T& List<T>::operator[](size\_t i){

return (\*buf.get())[i];

}

template <typename T>

const T& List<T>::operator[](size\_t i) const{

return (\*buf.get())[i];

}

template <typename T>

T& List<T>::at(size\_t i){

if(i >= sz){

throw out\_of\_range("Wrong index in List!");

}

return (\*buf.get())[i];

}

template <typename T>

const T& List<T>::at(size\_t i) const{

if(i >= sz){

throw out\_of\_range("Wrong index in List!");

}

return (\*buf.get())[i];

}

template <typename T>

Iterator<T> List<T>::begin(){

return Iterator<T>(\*buf.get(), 0);

}

template <typename T>

Iterator<T> List<T>::end(){

return Iterator<T>(\*buf.get() + sz, sz);

}

template <typename T>

void List<T>::insert(Iterator<T> pos, const T& value){

if((pos < begin()) || (pos > end())){

throw out\_of\_range("Wrong iterator!");

}

if(sz == cap){

resize(2 \* cap);

}

pos = Iterator<T>(\*buf.get() + pos.index(), pos.index());

++sz;

if(pos == end() - 1){

} else{

Iterator<T> it = end();

for(; it != pos; --it){

new (it.base()) T(\*(it - 1));

//\*it = Trapezoid<T>(\*(it - 1));

//cout << \*it << " " << \*(it - 1) << endl;

}

}

new(pos.base()) T(value);

}

template <typename T>

void List<T>::erase(Iterator<T> pos){

if((pos < begin()) || (pos >= end()) || empty()){

throw out\_of\_range("Wrong iterator!");

}

for(Iterator<T> it = pos; it != end(); ++it){

\*it = \*(it + 1);

}

--sz;

}

template <typename T>

List<T>::~List(){

for(size\_t i = 0; i < sz; ++i){

(\*buf.get() + i)->~T();

}

delete[] (\*buf.get());

}

Iterator.h**:**

//Определение структуры итератора

#define ITERATOR3\_H

#include <memory>

#include "Trapezoid.h"

using namespace std;

template <typename T>

class Iterator{

private:

T\* ptr= nullptr;

size\_t idx = -1;

public:

typedef std::random\_access\_iterator\_tag iterator\_category;

typedef T value\_type;

typedef T difference\_type;

typedef T\* pointer;

typedef T& reference;

Iterator();

Iterator(T\* t, size\_t idx);

Iterator& operator+=(size\_t n);

Iterator& operator-=(size\_t n);

Iterator operator+(size\_t n);

Iterator operator-(size\_t n);

T\* base() const;

size\_t index() const;

T& operator\*();

Iterator\* operator->();

Iterator& operator++();

Iterator operator++(int);

Iterator& operator--();

Iterator operator--(int);

bool operator==(Iterator const& other) const;

bool operator!=(Iterator const& other) const;

template <typename V> friend bool operator==(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend bool operator!=(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend bool operator<(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend bool operator>(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend bool operator<=(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend bool operator>=(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend size\_t operator-(const Iterator<V>& lhs, const Iterator<V>& rhs);

template <typename V> friend Iterator operator+(size\_t n, const Iterator<V>& i);

Iterator& operator=(const Iterator& copy);

};

#endif

Iterator.cpp**:**

#include <cstddef>

#include "Iterator3.h"

#include "Trapezoid.h"

using namespace std;

template <typename T>

Iterator<T>::Iterator() : ptr(nullptr){}

template <typename T>

Iterator<T>::Iterator(T\* t, size\_t index) : ptr(t), idx(index){}

template <typename T>

Iterator<T>& Iterator<T>::operator+=(size\_t n){

ptr += n;

idx += n;

return \*this;

}

template <typename T>

Iterator<T>& Iterator<T>::operator-=(size\_t n){

ptr -= n;

idx -= n;

return \*this;

}

template <typename T>

Iterator<T> Iterator<T>::operator+(size\_t n){

auto copy = \*this;

copy += n;

return copy;

}

template <typename T>

Iterator<T> Iterator<T>::operator-(size\_t n){

auto copy = \*this;

copy -= n;

return copy;

}

template <typename T>

T\* Iterator<T>::base() const{

return ptr;

}

template <typename T>

size\_t Iterator<T>::index() const{

return idx;

}

template <typename T>

T& Iterator<T>::operator\*(){

return \*ptr;

}

template <typename T>

Iterator<T>\* Iterator<T>::operator->(){

return ptr;

}

template <typename T>

Iterator<T>& Iterator<T>::operator++(){

++ptr;

++idx;

return \*this;

}

template <typename T>

Iterator<T> Iterator<T>::operator++(int){

auto copy = \*this;

copy++;

return copy;

}

template <typename T>

Iterator<T>& Iterator<T>::operator--(){

--ptr;

--idx;

return \*this;

}

template <typename T>

Iterator<T> Iterator<T>::operator--(int){

auto copy = \*this;

copy--;

return copy;

}

template <typename T>

bool Iterator<T>::operator==(Iterator const& other) const{

return ptr == other.base() && idx == other.index();

}

template <typename T>

bool Iterator<T>::operator!=(Iterator const& other) const{

return ptr != other.base() || idx != other.index();

}

template <typename T>

bool operator==(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() == rhs.base() && lhs.index() == rhs.index();

}

template <typename T>

bool operator!=(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() != rhs.base() || lhs.index() != rhs.index();

}

template <typename T>

bool operator<(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() < rhs.base() && lhs.index() == rhs.index();

}

template <typename T>

bool operator>(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() > rhs.base();

}

template <typename T>

bool operator<=(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() <= rhs.base();

}

template <typename T>

bool operator>=(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() >= rhs.base();

}

template <typename T>

size\_t operator-(const Iterator<T>& lhs, const Iterator<T>& rhs){

return lhs.base() - rhs.base();

}

template <typename T>

Iterator<T> operator+(size\_t n, const Iterator<T>& i){

return reinterpret\_cast<T>(i.base() + n, i.index() + n);

}

template <typename T>

Iterator<T>& Iterator<T>::operator=(const Iterator& copy){

ptr = copy.base();

idx = copy.index();

return \*this;

}

Trapezoid.h:

//Определение класса фигуры трапеции

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include <utility>

using namespace std;

template<typename T>

class Trapezoid {

public:

Trapezoid(T i,T j, T k);

Trapezoid();

Trapezoid(const Trapezoid<T>& orig);

double square();

double edge();

void print\_points();

void print\_diagonals();

void print\_info();

void print\_base\_and\_height();

template<typename V> friend std::ostream& operator<<(std::ostream& os, Trapezoid<V>& obj);

template<typename V> friend std::istream& operator>>(std::istream& is, Trapezoid<V>& obj);

bool operator==(Trapezoid<T>& copy);

Trapezoid<T>& operator=(const Trapezoid<T>& copy);

virtual ~Trapezoid();

private:

T one;

T two;

T three;

};

#endif

Trapezoid.cpp:

#include "Trapezoid.h"

#include <iostream>

#include <cmath>

using namespace std;

template<typename T>

Trapezoid<T>::Trapezoid(T i, T j, T k) : one(i), two(j), three(k) {

}

template<typename T>

Trapezoid<T>::Trapezoid() : one(0), two(0), three(0) {

}

template<typename T>

Trapezoid<T>::Trapezoid(const Trapezoid<T>& orig) {

one = orig.one;

two = orig.two;

three = orig.three;

}

template<typename T>

double Trapezoid<T>::square(){

return (one + two) \* three / 2.0;

}

template<typename T>

void Trapezoid<T>::print\_points(){

cout << "(" << one/2 << ";" << three/2 << "), ";

cout << "(" << two/2 << ";" << three/2\*(-1) << "), ";

cout << "(" << two/2\*(-1) << ";" << three/2\*(-1) << "), ";

cout << "(" << (one/2)\*(-1) << ";" << three/2 << ")";

}

template<typename T>

void Trapezoid<T>::print\_base\_and\_height(){

cout << one << ", " << two << ", " << three << "\n";

}

template<typename T>

bool Trapezoid<T>::operator==(Trapezoid<T>& copy) {

if (one == copy.one && two == copy.two && three==copy.three){

return true;

} else {

return false;

}

}

template<typename T>

Trapezoid<T>& Trapezoid<T>::operator=(const Trapezoid<T>& copy) {

if (this == &copy) return \*this;

one = copy.one;

two = copy.two;

three = copy.three;

return \*this;

}

template<typename T>

Trapezoid<T>::~Trapezoid() {

}

template<typename T>

double Trapezoid<T>::edge(){

double a=sqrt(abs(two-one))/2\*three;

if(a==0){

return three;

} else {

return a;

}

}

template<typename T>

void Trapezoid<T>::print\_info(){

cout << "Points: ";

print\_points();

cout << "\nBase 1: " << one;

cout << "\nBase 2: " << two;

cout << "\nEdge: " << edge();

cout << "\n" << "Square: "<< square() << "\n";

}

template<typename T>

std::ostream& operator<<(std::ostream& os, Trapezoid<T>& obj) {

os << obj.one;

os << " " << obj.two;

os << " " << obj.three;

return os;

}

template<typename T>

std::istream& operator>>(std::istream& is, Trapezoid<T>& obj) {

if(!(is >> obj.one >> obj.two >> obj.three)){

throw invalid\_argument("Wrong values!");

}

return is;

}

Файл main.cpp:

// Почечура Артемий Андреевич

// М8О-206Б-20

// Лабораторная работа №5

// Вариант 12

// Фигура: Трапеция

// Cтруктура данных для контейнера: список.

#include <iostream>

#include <exception>

#include <vector>

#include <utility>

#include <cmath>

#include <sstream>

#include <algorithm>

#include "Iterator3.h"

#include "Trapezoid.h"

#include "Iterator3.cpp"

#include "List3.h"

#include "List3.cpp"

using namespace std;

bool readnum(int& a){

cin >> a;

if (cin.fail()) {

cout <<"Incorrect input"<<'\n';

cin.clear();

cin.ignore(numeric\_limits<int>::max(),'\n');

return false;

}

return true;

}

void help(){

cout << "Commands:\n";

cout << "\tadd - add element with index.\n\tdelete - delete element with index.\n";

cout << "\tprint - print list";

cout << "\n\tget - print element with index";

cout << "\n\tend - exit\n";

cout << "\thelp\n\n";

cout << "Section, that join middles of bases, belong to axis Y (centre of section - point (0;0)).\n\n";

}

int main(int argc, char\*\* argv) {

List<Trapezoid<double>> list;

string act;

help();

while (1){

try{

cout << "Select a command: ";

cin>>act;

if (act=="end"){

break;

}

else if (act == "add"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

cout << "input first base, second base and height: ";

Trapezoid<double> a;

if(!(cin>>a)){

cout << "Wrong values!";

continue;

}

list.insert(list.begin()+ind,a);

cout << "Element added\n";

}

else if (act == "delete"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

list.erase(list.begin() + ind);

cout << "Element deleted\n";

}

else if (act == "print"){

for(auto i : list){

cout << i << endl;

}

}

else if (act == "get"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

Trapezoid<double> t = list.at(ind);

t.print\_info();

}

else if (act == "help"){

help();

}

else {

cout << "Invalid function. Enter 'help' for help."<<'\n';

}

} catch(const invalid\_argument &arg){

cout << arg.what() << "\n";

} catch(const out\_of\_range &arg){

cout << arg.what() << "\n";

}

}

}

**5.** **Вывод**

В ходе данной работы я реализовал шаблонные классы List, Iterator и Trapezoid для работы с динамическими массивами, в которых содержатся элементы шаблонного класса Trapezoid. Реализовал набор функций и операторов для использования данных классов.

**ЛИТЕРАТУРА**

1. Уроки программирования на языке С++ [Электронный ресурс]. URL:<https://ravesli.com/uroki-cpp> (дата обращения 15.11.2021).