**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: Тимофеев Алексей Владимирович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи.**

Реализовать программу, которая:

o Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;

o Позволяет удалять элемент из коллекции по номеру элемента;

o Выводит на экран введенные фигуры c помощью std::for\_each;

o Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if);

14. 5-угольник Список

1. **Описание программы**

Для хранения фигуры создается шаблонный класс Pentagon , который хранит в себе радиус описанной окружности (R) и координаты центра этой окружности. Для хранения коллекции используется умный указатель. При инициализации класса list(список) выделяется память под элементы Pentagon. Также реализован класс iterator, который использует умные указатели. Для того, чтобы итератор был совместим со стандартными алгоритмами (которые мы используем в данной программе: std::for\_each, std::count\_if), необходимо перегрузить операторы - ==, !=, ++, \*, +. В классе list реализованы методы begin() и end(), которые возвращают итератор. Метод Push отвечает за вставку, в нем можно выбрать позицию на которую будет вставлен элемент. Метод Pop отвечает за удаление по индексу в списке. Выводится список с помощью перегруженного оператора << .

1. **Набор  тестов**

coord to center: 0 0 radius: 7 index = 0

coord to center: 50 50 radius: 8 index = 1

coord to center: -50 -50 radius: 8 index = 2

coord to center: 100 100 radius: 10 index = 1

1. **Результаты выполнения тестов**

dude@DESKTOP-9IO9OQQ:/mnt/d/Ycheba2kurs/OOp/OOPlab/mylab5$ make

g++ -std=c++17 -Wall -Werror -Wno-sign-compare -Wno-unused-result -o lab5 \*.cpp

dude@DESKTOP-9IO9OQQ:/mnt/d/Ycheba2kurs/OOp/OOPlab/mylab5$ ./lab5

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 0 0

Enter radius: 7

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 4 4

Enter radius: 5

Enter index = 2

This index doesn't exist

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 50 50

Enter radius: 8

Enter index = 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

idx: 0 (-6.77075,1.77676) (6.09803,-3.43715) (-5.0259,4.87241) (3.62457,-5.98853) (-1.98584,6.71241)

idx: 1 (42.262,52.0306) (56.9692,46.0718) (44.2561,55.5685) (54.1424,43.156) (47.7305,57.6713)

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: -50 -50

Enter radius: 8

Enter index = 2

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 100 100

Enter radius: 10

Enter index = 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

idx: 0 (-6.77075,1.77676) (6.09803,-3.43715) (-5.0259,4.87241) (3.62457,-5.98853) (-1.98584,6.71241)

idx: 1 (90.3275,102.538) (108.711,95.0898) (92.8201,106.961) (105.178,91.445) (97.1631,109.589)

idx: 2 (42.262,52.0306) (56.9692,46.0718) (44.2561,55.5685) (54.1424,43.156) (47.7305,57.6713)

idx: 3 (-57.738,-47.9694) (-43.0308,-53.9282) (-55.7439,-44.4315) (-45.8576,-56.844) (-52.2695,-42.3287)

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

4

Enter square

1000

0 Square 133.295

1 Square 272.031

2 Square 174.1

3 Square 174.1

The number of objects which set area less than the established one: 4

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

4

Enter square

135

0 Square 133.295

1 Square 272.031

2 Square 174.1

3 Square 174.1

The number of objects which set area less than the established one: 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 4 5

Enter radius: 20

Enter index = 3

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

idx: 0 (-6.77075,1.77676) (6.09803,-3.43715) (-5.0259,4.87241) (3.62457,-5.98853) (-1.98584,6.71241)

idx: 1 (90.3275,102.538) (108.711,95.0898) (92.8201,106.961) (105.178,91.445) (97.1631,109.589)

idx: 2 (42.262,52.0306) (56.9692,46.0718) (44.2561,55.5685) (54.1424,43.156) (47.7305,57.6713)

idx: 3 (-15.345,10.0765) (21.4229,-4.82043) (-10.3597,18.9212) (14.3559,-12.1101) (-1.67382,24.1783)

idx: 4 (-57.738,-47.9694) (-43.0308,-53.9282) (-55.7439,-44.4315) (-45.8576,-56.844) (-52.2695,-42.3287)

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

4

Enter square

150

0 Square 133.295

1 Square 272.031

2 Square 174.1

3 Square 1088.13

4 Square 174.1

The number of objects which set area less than the established one: 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

4

Enter square

1000

0 Square 133.295

1 Square 272.031

2 Square 174.1

3 Square 1088.13

4 Square 174.1

The number of objects which set area less than the established one: 4

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

0

dude@DESKTOP-9IO9OQQ:/mnt/d/Ycheba2kurs/OOp/OOPlab/mylab5$ ./lab5

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 54 44

Enter radius: 7

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

idx: 0 (47.2292,45.7768) (60.098,40.5628) (48.9741,48.8724) (57.6246,38.0115) (52.0142,50.7124)

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

2

Enter index = 0

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

The list is empty.

1. **Листинг программы**

**main.cpp**

#include <iostream>

#include <algorithm>

#include <memory>

#include "pentagon.h"

#include "TIterator.h"

#include "TListItem.h"

#include "TList.h"

void menu(){

std::cout << "Choose an operation:" << std::endl;

std::cout << "1) Add pentagon" << std::endl;

std::cout << "2) Delete figure from list" << std::endl;

std::cout << "3) Print list" << std::endl;

std::cout << "4) Find the number of objects which set area less than the established one" << std::endl;

std::cout << "0) Exit" << std::endl;

}

int main(){

int32\_t act = 1;

TList<Pentagon<double>> list;

std::shared\_ptr<Pentagon<double>> ptr;

while (act != 0) {

menu();

std::cin >> act;

switch(act) {

case 1:

double xO, yO, R;

std::cout << "Enter coord to center: ";

std::cin >> xO >> yO;

std::cout << "\nEnter radius: ";

std::cin >> R;

if (R <= 0.0){

std::cout << "The entered values are not correct" << std::endl;

return 0;

}

ptr = std::make\_shared<Pentagon<double>>(xO,yO,R);

list.Push(ptr);

break;

case 2:

list.Pop();

break;

case 3:

std::cout << list << std::endl;

break;

case 4:

double s;

std::cout << "Enter square" << std::endl;

std::cin >> s;

if(s<=0){

std::cout << "Invalid data entered" << std::endl;

}else{

int k = 0, out = 0;

if(!list.IsEmpty()) {

for(auto i : list) {

std::cout << k << " Square " << i->S() << std::endl;

if (i->S()<s){

++out;

}

++k;

}

} else {

std::cout << "List is empty." << std::endl;

}

std::cout << "The number of objects which set area less than the established one: " << out << std::endl;}

break;

case 0:

list.Del();

break;

default:

std::cout << "Incorrect command" << std::endl;;

break;

}

}

return 0;

}

**pentagon.h**

#ifndef PENTAGON\_H

#define PENTAGON\_H

#include <algorithm>

#include <cmath>

#include <iostream>

template <class T>

class Pentagon {

public:

T R, x, y; // радиус и координаты центра

Pentagon() : x(0.0), y(0.0), R(0.0){}

Pentagon(double x1, double y1, double r1) {

x = x1;

y = y1;

R = r1;

}

double S();

void CoordP();

int countS();

};

template <class T>

void Pentagon<T>::CoordP() {

double angle,outX,outY;

for(int i = 2; i < 12; i = i+2) {// расчет точек и вывод

angle = 360 \* 0.2;

outX = R \* cos(i\*angle\*0.5) + x;

outY = R \* sin(i\*angle\*0.5) + y;

std::cout << '(' << outX << ',' << outY << ')'<< '\t';

}

std::cout << std::endl;

}

template <class T>

double Pentagon<T>::S() {

double s = (R \* R \* 5 \* 0.25 \* sqrt((5 + sqrt(20))\*0.5));

return s;

}

#endif

**TListItem.h**

#ifndef TLISTITEM\_H

#define TLISTITEM\_H

#include <memory>

#include <iostream>

#include "pentagon.h"

template <class T>

class TListItem{

private:

std::shared\_ptr<T> item;

std::shared\_ptr<TListItem<T>> next;

std::shared\_ptr<TListItem<T>> prev;

public:

TListItem(const std::shared\_ptr<T> &obj);

std::shared\_ptr<T> GetFigure() const;

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>> item);

template <class A> friend std::ostream& operator<<(std::ostream &os, const TListItem<A> &obj);

virtual ~TListItem(){};

};

template <class T>

TListItem<T>::TListItem(const std::shared\_ptr<T> &obj){

this->item = obj;

this->next = nullptr;

this->prev = nullptr;

}

template <class T>

std::shared\_ptr<T> TListItem<T>::GetFigure() const{

return this->item;

}

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>> item){

this->prev = item;

}

template <class T>

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

os << obj.item << std::endl;

return os;

}

#endif

**TList.h**

#ifndef TLIST\_H

#define TLIST\_H

#include<memory>

#include <iostream>

#include <cstdint>

#include "pentagon.h"

#include "TIterator.h"

#include "TListItem.h"

template <class T>

class TList{

private:

uint32\_t length;

std::shared\_ptr<TListItem<T>> head;

void PushFirst(std::shared\_ptr<T> &obj);

void PushLast(std::shared\_ptr<T> &obj);

void PushAtIndex(std::shared\_ptr<T> &obj, int32\_t ind);

std::shared\_ptr<T> PopFirst();

std::shared\_ptr<T> PopLast();

std::shared\_ptr<T> PopAtIndex(int32\_t ind);

public:

TList();

void Push(std::shared\_ptr<T> &obj);

const bool IsEmpty() const;

uint32\_t GetLength();

std::shared\_ptr<T> Pop();

void Del();

TIterator<TListItem<T>,T> begin();

TIterator<TListItem<T>,T> end();

template <class A> friend std::ostream& operator<<(std::ostream &os, const TList<A> &list);

virtual ~TList();

};

template <class T>

TList<T>::TList(){

head = nullptr;

length = 0;

}

template <class T>

void TList<T>::Push(std::shared\_ptr<T> &obj){

if (this->GetLength() == 0){

this->PushFirst(obj);

++length;

return;

}

int32\_t index = 0;

std::cout << "Enter index = ";

std::cin >> index;

if (index > this->GetLength() || index < 0) {

std::cerr << "This index doesn't exist\n";

return;

}

if (index == 0) {

this->PushFirst(obj);

} else if (index == (this->GetLength()+1)) {

this->PushLast(obj);

} else {

this->PushAtIndex(obj, index);

}

++length;

}

template <class T>

void TList<T>::PushAtIndex(std::shared\_ptr<T> &obj, int32\_t ind){

std::shared\_ptr<TListItem<T>> newItem = std::make\_shared<TListItem<T>>(obj);

std::shared\_ptr<TListItem<T>> tmp = this->head;

for(int32\_t i = 1; i < ind; ++i){

tmp = tmp->GetNext();

}

newItem->SetNext(tmp->GetNext());

newItem->SetPrev(tmp);

tmp->SetNext(newItem);

tmp->GetNext()->SetPrev(newItem);

}

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>

uint32\_t TList<T>::GetLength(){

return this->length;

}

template <class T>

const bool TList<T>::IsEmpty() const{

return head == nullptr;

}

template <class T>

std::shared\_ptr<T> TList<T>::Pop(){

int32\_t ind = 0;

std::cout << "Enter index = ";

std::cin >> ind;

std::shared\_ptr<T> res;

if (ind > this->GetLength() - 1 || ind < 0 || this->IsEmpty()) {

std::cout << "Change index" << std::endl;

return res;

}

if (ind == 0) {

res = this->PopFirst();

} else if (ind == this->GetLength() - 1) {

res = this->PopLast();

} else {

res = this->PopAtIndex(ind);

}

--length;

return res;

}

template <class T>

std::shared\_ptr<T> TList<T>::PopAtIndex(int32\_t ind){

std::shared\_ptr<TListItem<T>> tmp = this->head;

for(int32\_t i = 0; i < ind - 1; ++i) {

tmp = tmp->GetNext();

}

std::shared\_ptr<TListItem<T>> removed = tmp->GetNext();

std::shared\_ptr<T> res = removed->GetFigure();

std::shared\_ptr<TListItem<T>> nextItem = removed->GetNext();

tmp->SetNext(nextItem);

nextItem->SetPrev(tmp);

return res;

}

template <class T>

std::shared\_ptr<T> TList<T>::PopFirst(){

if (this->GetLength() == 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->GetLength() == 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>> removed = tmp->GetNext();

std::shared\_ptr<T> res = removed->GetFigure();

tmp->SetNext(removed->GetNext());

return res;

}

template <class T>

std::ostream& operator<<(std::ostream &os, const TList<T> &list){

if (list.IsEmpty()) {

os << "The list is empty." << std::endl;

return os;

}

std::shared\_ptr<TListItem<T>> tmp = list.head;

for(int32\_t i = 0; tmp; ++i) {

os << "idx: " << i << " ";

tmp->GetFigure()->CoordP();

os << std::endl;

tmp = tmp->GetNext();

}

return os;

}

template <class T>

void TList<T>::Del(){

while(!this->IsEmpty()) {

this->PopFirst();

--length;

}

}

template <class T>

TIterator<TListItem<T>,T> TList<T>::begin(){

return TIterator<TListItem<T>,T>(head);

}

template <class T>

TIterator<TListItem<T>,T> TList<T>::end(){

std::shared\_ptr<TListItem<T>> tmp = this->head;

for(int32\_t i = 0; i < this->length; ++i) {

tmp = tmp->GetNext();

}

return TIterator<TListItem<T>,T>(tmp);

}

template <class T>

TList<T>::~TList(){

}

#endif

**TIterator.h**

#ifndef TITERATOR\_H

#define TITERATOR\_H

#include <memory>

#include <iostream>

template <class N, class T>

class TIterator{

private:

std::shared\_ptr<N> cur;

public:

TIterator(std::shared\_ptr<N> n) {

cur = n;

}

std::shared\_ptr<T> operator\* () {

return cur->GetFigure();

}

std::shared\_ptr<T> operator-> () {

return cur->GetFigure();

}

void operator++() {

cur = cur->GetNext();

}

TIterator operator++ (int) {

TIterator cur(\*this);

++(\*this);

return cur;

}

void operator--() {

cur = cur->GetPrev();

}

TIterator operator-- (int) {

TIterator cur(\*this);

--(\*this);

return cur;

}

bool operator== (const TIterator &i) {

return (cur == i.cur);

}

bool operator!= (const TIterator &i) {

return (cur != i.cur);

}

};

#endif

1. **Вывод**

В данной лабораторной работе я научился реализовывать итератор для списка, проверил на практике его совместимость с алгоритмами из std::. Также я научился работать с умными указателями и еще раз попрактиковался в создание шаблонов и в метапрограммирование.

Список литературы

1. cppreference.com [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/algorithm/count/ (дата обращения: 20.11.20)
2. Метапрограммирование в C++ [Электронный ресурс]. URL: https://habr.com/ru/post/38622/(дата обращения: 19.11.20)