МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №5**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Абросимов Алексей Дмитриевич, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Задание

Спроектировать и разработать итератор для динамической структуры данных.

**Вариант №3:**

* Фигура : Прямоугольник
* Контейнер: Вектор (TVector)

**Описание программы:**

Исходный код разделён на 7 файлов:

* rectangle.h – описание класса прямоугольника (наследуется от фигуры)
* rectangle.cpp – реализация класса прямоугольника
* Iter.h - описание и реализация итератора
* Item.h - описание элемента вектора
* Item.cpp - реализация элемента вектора
* tvector.h – описание класса квадрата (наследуется от прямоугольника)
* tvector.cpp – реализация класса квадрата
* main.cpp – основная программа

**Дневник отладки:**

**Вывод:**  
 Выполнение лабораторной работы позволило мне ознакомиться с итераторами.

**Исходный код:**

**Iter.h**

#ifndef ITER\_H

#define ITER\_H

#include <iostream>

#include <memory>

template <class node, class T>

class Iter {

public:

Iter(std::shared\_ptr<node> n) { node\_ptr = n; }

std::shared\_ptr<T> operator\*() { return node\_ptr->Get(); }

std::shared\_ptr<T> operator->() { return node\_ptr->Get(); }

void operator++() { node\_ptr = node\_ptr->GetNext(); }

Iter operator++(int) {

Iter iter(\*this);

++(\*this);

return iter;

}

bool operator==(Iter const& i) { return node\_ptr == i.node\_ptr; }

bool operator!=(Iter const& i) { return !(\*this == i); }

private:

std::shared\_ptr<node> node\_ptr;

};

#endif // ITER\_H

**Item.cpp**

#include "item.h"

#include <iostream>

template <class T>

Item<T>::Item(const std::shared\_ptr<T>& item)

: item(item){

std::cout << "TVector item: created" << std::endl;

}

template <class T>

std::shared\_ptr<T> Item<T>::Get() const {

return this->item;

}

template <class T>

std::shared\_ptr<Item<T>> Item<T>::GetNext() {

return this->next;

}

template <class T>

Item<T>::~Item() {

std::cout << "Stack item: deleted" << std::endl;

}

template <class T>

void Item<T>::SetNext(std::shared\_ptr<Item<T>>& next) {

this->next=next;

}

template <class A>

std::ostream& operator<<(std::ostream& os, const Item<A>& obj) {

os << "Item: " << \*obj.item << std::endl;

return os;

}

template <class T>

void Item<T>::forget(){

next=nullptr;

}

template <class T>

void\* Item<T>::operator new(size\_t size) {

std::cout << "Allocated :" << size << "bytes" << std::endl;

return malloc(size);

}

template <class T>

void Item<T>::operator delete(void\* p) {

std::cout << "Deleted" << std::endl;

free(p);

}

#include "rectangle.h"

template class Item<Rectangle>;

template std::ostream& operator<<(std::ostream& os,

const Item<Rectangle>& obj);

**Item.h**

#ifndef ITEM\_H

#define ITEM\_H

#include <memory>

template <class T>

class Item {

public:

Item(const std::shared\_ptr<T>& triangle);

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

template <class A>

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

void SetNext(std::shared\_ptr<Item<T>>& next);

std::shared\_ptr<Item<T>> GetNext();

void forget();

void\* operator new(size\_t size);

void operator delete(void\* p);

virtual ~Item();

private:

std::shared\_ptr<T> item;

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

};

#endif // ITEM\_H

**rectangle.h:**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

class Rectangle{

public:

Rectangle();

Rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3, int y4);

Rectangle(std::istream&is);

bool isit();

void Print(std::ostream&os);

size\_t VertexesNumber();

double Area();

~Rectangle();

friend std::ostream &operator<<(std::ostream &out,const Rectangle &rec);

friend std::istream &operator>>(std::istream &in,Rectangle &rec);

private:

double x1;

double y1;

double x2;

double y2;

double x3;

double y3;

double x4;

double y4;

};

#endif // RECTANGLE\_H

**rectangle.cpp:**

#include "rectangle.h"

#include <math.h>

Rectangle::Rectangle():x1(0),y1(0),x2(1),y2(1),x3(0),y3(0),x4(0),y4(0){

}

Rectangle::Rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4){

this->x1=x1;

this->x2=x2;

this->x3=x3;

this->x4=x4;

this->y1=y1;

this->y2=y2;

this->y3=y3;

this->y4=y4;

}

Rectangle::~Rectangle(){

std::cout<<"Rectangle was deleted\n";

}

Rectangle::Rectangle(std::istream&is){

std::cout <<"set x1 and y1:";

is >> x1 >> y1;

std::cout <<"set x2 and y2:";

is >> x2 >> y2;

std::cout <<"set x3 and y3:";

is >> x3 >> y3;

std::cout <<"set x4 and y4:";

is >> x4 >> y4;

}

void Rectangle::Print(std::ostream&os){

os << "Rectangle " << "(" <<x1<<" "<<y1<<")"<< "(" <<x2<<" "<<y2<<")"<< "(" <<x3<<" "<<y3<<")"<< "(" <<x4<<" " <<y4<<")" <<std::endl;

}

size\_t Rectangle::VertexesNumber(){

return 4;

}

bool Rectangle::isit(){

double perp;

double perp2;

perp=(x4-x1)\*(x2-x1)+(y4-y1)\*(y2-y1);

perp2=(x3-x4)\*(x3-x2)+(y3-y4)\*(y3-y2);

if((perp+perp2)==0) return true;

else return false;

}

double Rectangle::Area(){

double r1 = sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

double r2 = sqrt((x2 - x3) \* (x2 - x3) + (y2 - y3) \* (y2 - y3));

double r3 = sqrt((x1 - x3) \* (x1 - x3) + (y1 - y3) \* (y1 - y3));

double p=(r1+r2+r3)/2;

double s= 2\*sqrt((p \* (p - r1) \* (p - r2) \* (p - r3)));

return s;

}

std::ostream& operator<<(std::ostream &out, const Rectangle &rec){

out << "Rectangle coords " <<"("<< rec.x1 << "," << rec.y1 << ")"<< " " <<"("<< rec.x2 << "," << rec.y2 << ")"<< " "<< "("<< rec.x3 << "," << rec.y3 << ")"<< " " << "("<<rec.x4 << "," << rec.y4 << ")\n";

return out;

}

std::istream& operator>>(std::istream &in,Rectangle &rec){

in >> rec.x1;

in >> rec.y1;

in >> rec.x2;

in >> rec.y2;

in >> rec.x3;

in >> rec.y3;

in >> rec.x4;

in >> rec.y4;

return in;

}

**Tvector.h;**

#ifndef TVECTOR\_H

#define TVECTOR\_H

#include <memory>

#include "item.h"

#include "Iter.h"

template <class T>

class TVector

{

private:

int length;

int count;

std::shared\_ptr<Item<T>> \*arr;

public:

TVector();

virtual ~TVector();

int size();

bool empty();

void resize(int nindex);

void push\_back(std::shared\_ptr<T> &&newrec);

void erase(int pos);

std::shared\_ptr<T> pop\_back();

void clear();

Iter<Item<T>, T> begin();

Iter<Item<T>, T> end();

std::shared\_ptr<Item<T>>& operator[] (int i) ;

template <class A>

friend std::ostream& operator<<(std::ostream &out, TVector<A> &cont);

};

#endif // TVECTOR\_H

**Tvector.cpp;**

#include "tvector.h"

#include "rectangle.h"

template <class T>

TVector<T>::TVector():length(0),count(0) { }

template <class T>

int TVector<T>::size(){

return this->length;

}

template <class T>

bool TVector<T>::empty(){

if(this->length>0) return true;

else return false;

}

template <class T>

void TVector<T>::push\_back(std::shared\_ptr<T> &&newrec){

std::shared\_ptr<Item<T>> other(new Item<T>(newrec));

if(count==length){

length++;

count++;

std::shared\_ptr<Item<T>> \*narr=new std::shared\_ptr<Item<T>>[length];

for(int i=0;i<length-1;i++) narr[i]=arr[i];

narr[length-1]=other;

if(count-1) narr[length-2]->SetNext(other);

//free(arr);

arr=narr;

}

else if(count<length){

arr[count]=other;

if(count)

arr[count-1]->SetNext(other);

count++;

}

}

template <class T>

TVector<T>::~TVector(){

}

template <class T>

std::shared\_ptr<T> TVector<T>::pop\_back(){

std::shared\_ptr<T> result;

std::shared\_ptr<Item<T>> \*narr=new std::shared\_ptr<Item<T>>[length];

for(int i=0;i<count-1;i++){

narr[i]=arr[i];

}

result=arr[count-1]->Get();

count--;

length--;

arr=narr;

arr[count-1]->forget();

return result;

}

template <class T>

void TVector<T>::resize(int newlength){

if(newlength==length) return;

if(newlength>length){

std::shared\_ptr<Item<T>> \*narr=new std::shared\_ptr<Item<T>>[newlength];

for(int i=0;i<length;i++)

narr[i]=arr[i];

arr=narr;

length=newlength;

}

else {

std::shared\_ptr<Item<T>> \*narr=new std::shared\_ptr<Item<T>>[newlength];

for(int i=0;i<newlength;i++)

narr[i]=arr[i];

arr=narr;

count=newlength;

}

arr[count-1]->forget();

}

template <class T>

void TVector<T>::clear(){

free(arr);

length=0;

count=0;

}

template <class T>

void TVector<T>::erase(int pos){

if(count==0)

{

std::cout<<"Container is empty"<<std::endl;

return;

}

std::shared\_ptr<Item<T>> \*narr=new std::shared\_ptr<Item<T>>[length-1];

int current\_index=0;

for(int i=0;i<count;i++){

if(i!=pos-1) {

narr[current\_index]=arr[i];

current\_index++;

}

}

count--;

length--;

arr=narr;

arr[count-1]->forget();

}

//перегрузка операций

template <class T>

std::shared\_ptr<Item<T>>& TVector<T>::operator[] (int i)

{

if(i >= 0 && i < this->length)

return this->arr[i];

else perror("Out of bounds\n");

}

template <class T>

std::ostream& operator<<(std::ostream &out, TVector<T> &cont){

for(int i=0;i<cont.count;i++){

out<<"figure #"<< i+1<<"coords is " << \*cont[i];

}

return out;

}

template <class T>

Iter<Item<T>, T> TVector<T>::begin() {

return Iter<Item<T>, T>(arr[0]);

}

template <class T>

Iter<Item<T>, T> TVector<T>::end() {

return Iter<Item<T>, T>(nullptr);

}

template class TVector<Rectangle>;

template std::ostream& operator<<(std::ostream& out, TVector<Rectangle>& cont);

**main.cpp**

#include <iostream>

#include "rectangle.cpp"

#include <tvector.h>

#include "Iter.h"

int main()

{ TVector<Rectangle> vec;

vec.push\_back(std::shared\_ptr<Rectangle>(new Rectangle(1, 1, 1,1,1,1,1,1)));

vec.push\_back(std::shared\_ptr<Rectangle>(new Rectangle(2, 2, 2,2,2,2,2,2)));

vec.push\_back(std::shared\_ptr<Rectangle>(new Rectangle(3, 3, 3,3,3,3,3,3)));

std::cout<<"Last obj is " <<\*vec.pop\_back()<<std::endl;

for (auto i : vec) {

std::cout << \*i << std::endl;

}

return 0;

}