

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №5

по курсу объектно-ориентированное программирование 3 семестр, 2021/22 уч. Год

Студент Абросимов Алексей Дмитриевич, группа М8О-207Б-20
Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 3: Динамический массив, прямоугольник.

Используя структуру данных, разработанную для лабораторной работы №4, спроектировать и разработать итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен позволять работать с любыми типами фигур, согласно варианту задания.

Описание программы

Исходный код лежит в 8 файликах:

1. main.cpp — основная программа
2. item.h — описание класса элемента динамического массива
3. item.cpp — описание методов элемента дин.массива
4. iter.h — описание класса итератора и его методов
5. rectangle.h — описание класса прямоугольника
6. rectangle.cpp — описание методов прямоугольника
7. tvector.h — описание класса дин.массива
8. tvector.cpp — описание методов дин.массива.

Дневник отладки

Результат выполнения программы:

```
Allocated :40bytes
TVector item: created
Allocated :40bytes
TVector item: created
Allocated :40bytes
TVector item: created
Last obj is Rectangle coords (3,3) (3,3) (3,3) (3,3)

Rectangle coords (1,1) (1,1) (1,1) (1,1)

Rectangle coords (2,2) (2,2) (2,2) (2,2)

Rectangle coords (3,3) (3,3) (3,3) (3,3)

TVector item: deleted
TVector item: deleted
TVector item: deleted
Rectangle was deleted
Deleted
Rectangle was deleted
Deleted
Rectangle was deleted
Deleted
```

Недочёты

Выводы

Данная лабораторная работа позволила мне углубить мои знания про итераторы. Их написание мне показалось не таким уж и внешне простым, однако итераторы сильно упрощают дальнейшую работу с контейнером, уменьшают «количество» строк кода в больших проектах и позволяют избежать некоторых ошибок.

Ссылка на гитхаб: https://github.com/yungalexxyey/oop_labs/tree/main/lab5

Исходный код

main.cpp

```
#include <iostream>
#include "tvector.h"
#include "rectangle.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::shared_ptr<Rectangle> t;
t=vec.pop_back();
std::cout<<"Last obj is " <<*t<<std::endl;
for (auto i : vec) {
std::cout << *i << std::endl;
}
return 0;
}

```

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

```

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 << "TVector 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);

```

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
```

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;
}

```

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

```

tvector.cpp

```

#include "tvector.h"
#include "rectangle.h"
#include "item.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> newfig){
std::shared_ptr<Item<T>> other(new Item<T>(newfig));
if(count==length){
length++;
count++;
std::shared_ptr<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){
arr[count-2]->SetNext(narr[count-1]);
}
//free(arr);
arr=narr;
}
else if(count<length){
arr[count]=other;
count++;
if(count-1){
arr[count-2]->SetNext(arr[count-1]);
}
}
}

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<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;
return result;
}

template <class T>
void TVector<T>::resize(int newlength){
if(newlength==length) return;
if(newlength>length){
std::shared_ptr<std::shared_ptr<Item<T>>[]> narr(new std::shared_ptr<Item<T>>[length]);
for(int i=0;i<length;i++)
narr[i]=arr[i];
arr=narr;
length=newlength;
}
else {
std::shared_ptr<std::shared_ptr<Item<T>>[]> narr(new std::shared_ptr<Item<T>>[length]);
for(int i=0;i<newlength;i++)

```

```

narr[i]=arr[i];
arr=narr;
count=newlength;
}
}

```

```

template <class T>
void TVector<T>::clear(){
resize(1);
pop_back();
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<std::shared_ptr<Item<T>>>[]> narr(new std::shared_ptr<Item<T>>>[length]);
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;
}

```

```

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);
}
//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];
}

```

```

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

```


tvector.h

```
#ifndef TVECTOR_H
#define TVECTOR_H
#include <memory>
#include "Iter.h"
#include "item.h"
#include "rectangle.h"
template <class T>
class TVector
{
private:
int length;
int count;
std::shared_ptr<std::shared_ptr<Item<T>>[]> arr;
public:
TVector();
~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
```