**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «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: Хренов Геннадий

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Создать шаблон динамического массива, при этом: в качестве параметра шаблона коллекция должна принимать тип данных, коллекция должна возвращать итераторы begin() и end(), коллекция должна содержать метод вставки insert, коллекция должна содержать метод удаления erase, итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if).

1. Описание программы

Программа позволяет вводить с клавиатуры фигуры типа int(по заданию) и добавлять их в коллекцию. Из коллекции можно удалять фигуры по индексу, а также выводить на экран уже введенные фигуры с их свойствами. Также можно узнать число объектов, площадь которых меньше введенного значения. Для всех входных данных ловятся исключения, например существующий ли индекс вводится или правильность ввода квадрата. Для проверки, является ли число квадратом, я проверяю равенство всех его сторон и прямой ли один из его углов. В реализации вектора учтены все основные функции коллекции, такие как pushback, size, empty и тд.. Для квадрата определены нахождение площади, геометрический центр и вывод координат.

1. Набор testcases
2. add 1 1 2 1 2 2 1 2 - правильный ввод квадрата, программа должна добавить фигуру в вектор
3. print 0 - программа должна вывести координаты фигуры, площадь, центр
4. del 0 del 0 - сначала программа должна удалить элемент, затем элементов не будет - надо вывести сообщение об ошибке
5. add 1 1 2 2 3 3 4 4 - добавление неправильного квадрата, сообщение об ошибке
6. Результаты выполнения тестов.

add 1 1 2 1 2 2 1 2

Square added

print 0

A = (1, 1) B = (2, 1)

C = (2, 2) D = (1, 2)

centr: (1.00, 1.00)

area: 1.00

del 0

Element was deleted

del 0

Invalid index

add 1 1 2 2 3 3 4 4

It`s not a Square

exit

1. Листинг программы

**figure.hpp**

#include <iostream>

#include <iomanip>

#include <cmath>

template <class A, class B>

class myPair : public std::pair<A, B> {

public:

myPair() : std::pair<A, B>() {

this->first = 0;

this->second = 0;

}

myPair(A firstI, B secondI) : std::pair<A, B>() {

this->first = firstI;

this->second = secondI;

}

friend std::ostream& operator<< (std::ostream &out, const myPair<A,B> &point) {

out << std::fixed << std::setprecision(2) << '(' << point.first << ", " << point.second << ')';

return out;

}

friend std::istream& operator>> (std::istream &in, myPair<A,B> &point) {

in >> point.first;

in >> point.second;

return in;

}

};

template <class T>

double distance(myPair<T,T> a, myPair<T,T> b) {

return sqrt((a.first - b.first) \* (a.first - b.first) + (a.second - b.second) \* (a.second - b.second));

}

template <class T>

class Figure {

public:

using Point = myPair<T,T>;

Point points[4];

};

template <class T>

class Square : public Figure<T> {

public:

using Point = myPair<T,T>;

Square(Point a, Point b, Point c, Point d) {

Figure<T>::points[0] = a;

Figure<T>::points[1] = b;

Figure<T>::points[2] = c;

Figure<T>::points[3] = d;

}

};

template <class T>

void printCoor(Figure<T>& sq) {

std::cout << "A = " << sq.points[0] << "\tB = " << sq.points[1] << "\nC = " << sq.points[2] << "\tD = " << sq.points[3] << '\n';

}

template <class T>

auto centr(Figure<T>& sq) {

myPair<double, double> cnt;

cnt.first = (sq.points[0].first + sq.points[1].first + sq.points[2].first + sq.points[3].first) / 4;

cnt.second = (sq.points[0].second + sq.points[1].second + sq.points[2].second + sq.points[3].second) /4;

return cnt;

}

template <class T>

double area (Figure<T>& sq) {

return pow(distance(sq.points[0], sq.points[1]),2);

}

**vectoriter.hpp**

//Хренов Геннадий

// вектор

#include <algorithm>

#include <cassert>

#include <memory>

#include <iostream>

template <typename T>

class TVector {

public:

using value\_type = T;

using iterator = value\_type\*;

TVector():

already\_used\_(0), storage\_size\_(0), storage\_(nullptr)

{

}

TVector(int size, const value\_type& default\_value = value\_type()):

TVector()

{

assert(size >= 0);

if (size == 0) {

return;

}

already\_used\_ = size;

storage\_size\_ = size;

storage\_ = std::make\_unique<value\_type[]>(size);

std::fill(storage\_.get(), storage\_.get() + already\_used\_, default\_value);

}

int size() const

{

return already\_used\_;

}

bool empty() const

{

return size() == 0;

}

iterator begin() const

{

return storage\_.get();

}

iterator end() const

{

if (storage\_.get()) {

return storage\_.get() + already\_used\_;

}

return nullptr;

}

void insert(iterator pos, value\_type val) {

if (already\_used\_ < storage\_size\_) {

std::copy(pos, storage\_.get() + already\_used\_, pos + 1);

\*pos = val;

++already\_used\_;

return;

}

int next\_size = 1;

if (storage\_size\_) {

next\_size = storage\_size\_ \* 2;

}

TVector next(next\_size);

next.already\_used\_ = already\_used\_;

if (storage\_.get()) {

std::copy(storage\_.get(), storage\_.get() + storage\_size\_, next.storage\_.get());

}

next.insert(pos, val);

Swap(\*this, next);

}

void erase(iterator pos) {

std::copy(pos + 1, storage\_.get() + already\_used\_, pos);

--already\_used\_;

}

friend void Swap(TVector& lhs, TVector& rhs)

{

using std::swap;

swap(lhs.already\_used\_, rhs.already\_used\_);

swap(lhs.storage\_size\_, rhs.storage\_size\_);

swap(lhs.storage\_, rhs.storage\_);

}

TVector& operator=(TVector other)

{

Swap(\*this, other);

return \*this;

}

TVector(const TVector& other):

TVector()

{

TVector next(other.storage\_size\_);

next.already\_used\_ = other.already\_used\_;

if (\*(other.storage\_) ) {

std::copy(other.storage\_.get(), other.storage\_.get() + other.storage\_size\_,

next.storage\_.get());

}

swap(\*this, next);

}

~TVector()

{

storage\_size\_ = 0;

already\_used\_ = 0;

}

void push\_back(const value\_type& value)

{

if (already\_used\_ < storage\_size\_) {

storage\_[already\_used\_] = value;

++already\_used\_;

return;

}

int next\_size = 1;

if (storage\_size\_) {

next\_size = storage\_size\_ \* 2;

}

TVector next(next\_size);

next.already\_used\_ = already\_used\_;

if (storage\_.get()) {

std::copy(storage\_.get(), storage\_.get() + storage\_size\_, next.storage\_.get());

}

next.push\_back(value);

Swap(\*this, next);

}

value\_type& At(int index)

{

if (index < 0 || index > already\_used\_) {

throw std::out\_of\_range("You are doing this wrong!");

}

return storage\_[index];

}

value\_type& operator[](int index)

{

return At(index);

}

private:

int already\_used\_;

int storage\_size\_;

std::unique\_ptr<value\_type[]> storage\_;

};

**lab5.cpp**

//Хренов Геннадий М80-207Б

// cоздание динамического массива

// фигура - квадрат

// Программа позволяет вводить с клавиатуры фигуры типа int(по заданию)

//и добавлять их в коллекцию. Из коллекции можно удалять фигу

//ры по индексу, а также выводить на экран уже введенные фигуры с их свойствами.

//Также можно узнать число объектов, площадь которых меньше введенного значения.

//Для всех входных данных ловятся исключения,

//например существующий ли индекс вводится или правильность ввода квадрата.

#include <iostream>

#include <utility>

#include <algorithm>

#include <map>

#include "vectoriter.cpp"

#include "figure.hpp"

bool isSquare(myPair<int,int> a, myPair<int,int> b, myPair<int,int> c, myPair<int,int> d)

{

if (distance(a, b) == distance(c,d)) {

if (distance(b,c) == distance(a,d)) {

if(distance(a,b) == distance(b,c)) {

if ((int)(pow(distance(a,b),2) + pow(distance(b,c), 2)) == int(pow(distance(a,c),2))) {

return true;

}

}

}

}

return false;

}

template <class T>

void printCoorFE(T In) {

printCoor(\*In);

}

int main(void) {

using Point = myPair<int,int>;

Point P1, P2, P3, P4;

int areaMax;

int index;

TVector<Figure<int>\*> vec;

std::string inpt;

//char inpt[20];

int ex = 0;

while (ex == 0) {

std::cin >> inpt;

if (inpt == "add") {

std::cin >> P1 >> P2 >> P3 >> P4;

if(!(isSquare(P1,P2,P3,P4))) {

std::cout << "It`s not a Square\n";

} else {

vec.push\_back(dynamic\_cast<Figure<int>\*>(new Square<int>(P1, P2, P3, P4)));

std::cout << "Square added\n";

}

} else if (inpt == "print") {

std::cin >> inpt;

if (inpt == "all") {

std::for\_each(vec.begin(), vec.end(), [](auto& k){

printCoor(\*k);

putchar('\n');

});

} else {

try {

index = std::stoi(inpt);

} catch (std::invalid\_argument) {

std::cout << "Invalid argument\n";

}

if (index > (vec.size() - 1) || index < 0) {

std::cout << "Invalid index\n";

} else {

printCoor(\*(vec[index]));

std::cout << "centr: " << centr(\*(vec[index])) << '\n';

std::cout << "area: " << area(\*vec[index]) << '\n';

}

}

} else if (inpt == "lessthan") {

std::cin >> areaMax;

if (areaMax < 0) {

std::cout << "Invalid areaMax\n";

} else {

std::cout << std::count\_if(vec.begin(), vec.end(), [areaMax](auto& k) {

return areaMax > area(\*k);

}) << '\n';

}

} else if (inpt == "del") {

if (!(std::cin >> index)) {

std::cout << "Invalid input\n";

}

if (index > (vec.size() - 1) || index < 0) {

std::cout << "Invalid index\n";

} else {

delete vec[index];

vec.erase(vec.begin() + index);

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

}

} else if (inpt == "exit") {

ex = 1;

for (int i = 0; i < vec.size();i++) {

delete vec[i];

}

} else if (inpt == "help") {

std::cout << "available command:\n";

std::cout << "\tadd (use with 4 point)\n";

std::cout << "\tprint (use with all or with index)\n";

std::cout << "\tlessthan (use with area you want)\n";

std::cout << "\tdel (use with index)\n";

std::cout << "\texit (if you want to quite)\n";

std::cout << "\thelp (if you want to see this menu again)\n";

} else {

std::cout << "unknown comand(use help)\n";

}

}

return 0;

}

6. Выводы

Вектор - коллекция, которую используют чаще всего. Очень удобно, что у этой коллекции есть такой же оператор operator [], что и у обычного массива. Важно понимать, что вместимость вектора изменяется динамически. Для увеличения размера используется мультипликативный подход: выделенная под vector память увеличивается при необходимости в константное число раз, т.е. если добавление нового элемента приведет к тому, что размер массива превысит вместимость, то операционной системой для программы будет выделен новый участок памяти, например, в два раза больший, в который будут скопированы все значения из старого участка памяти и к которому будет приписано новое значение.

СПИСОК ЛИТЕРАТУРЫ

1. Стандартные библиотеки шаблонов [Электронный ресурс]. URL:<https://tproger.ru/articles/stl-cpp/>

(дата обращения: 20.11.2019).

2. Векторы в С++ [Электронный ресурс]. URL:

<https://code-live.ru/post/cpp-vector/>

(дата обращения: 20.11.2019).