**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: Гребенков Дмитрий Игоревич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

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, 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 с помощью std::for\_each;
* выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

Вариант задания: 12. Трапеция, список.

1. Описание программы

Созданы шаблонные классы: для хранения трапеций (хранятся координаты трех вершин), для хранения элемента списка (содержит объект класса “Трапеция” и ссылку на следующий элемент), для отображения списка, а также класс итератора.

Классы для элемента списка, списка и итератора написаны с использованием умных указателей std::shared\_ptr и std::weak\_ptr.

В классе списка реализованы требуемые по варианту задания методы.

В классе итератора перегружены все операторы, необходимые для корректной работы стандартных алгоритмов.

Реализованы шаблонные функции для вывода вершин трапеций, для вычисления площади трапеций, а также функции, необходимые для корректной работы стандартных алгоритмов.

Взаимодействие с пользователем осуществляется при помощи меню.

При выполнении некорректных операций происходит генерация исключений и вывод описания ошибки в поток вывода std::cerr.

Примечание: для ускорения вычислений и упрощения хранения на вводимые фигуры установлены некоторые ограничения. Во-первых, основания вводимых трапеций должны быть параллельны оси Ox. Во-вторых, большее основание трапеции должно быть снизу.

Трапеция задается координатами трех вершин.

1. Набор тестов и результаты их выполнения

Некорректные команды:

0. Print menu.

1. Show list.

11. Go to the beginning.

12. Go to the ending.

13. Go to the next element.

14. Go to the element with the entered number.

2. Add element in iterator position.

3. Erase element in iterator position.

4. Print figures with "for\_each".

5. Print the number of objects with an area less than the specified one.

9. Quit the program.

Enter option: 0

0. Print menu.

1. Show list.

11. Go to the beginning.

12. Go to the ending.

13. Go to the next element.

14. Go to the element with the entered number.

2. Add element in iterator position.

3. Erase element in iterator position.

4. Print figures with "for\_each".

5. Print the number of objects with an area less than the specified one.

9. Quit the program.

Enter option: 1

List is empty.

Enter option: 13

ERROR: Trying to go out of range.

Enter option: 14

Enter index: 1

ERROR: Index is out of range.

Enter option: 3

ERROR: Trying to delete nonexistent elements.

Enter option: 4

Enter option: 5

Enter square: 10

Number of objects with an area less than the specified one: 0

Демонстрация работы основных функций:

Enter option: 2

Enter the data of the figure:

1 2 0 0 4 0

Enter option: 1

-> |||(1, 2)(0, 0)(4, 0)(3, 2)|||

Enter option: 4

(1, 2)(0, 0)(4, 0)(3, 2)

Enter option: 13

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> ||| |||

Enter option: 2

Enter the data of the figure:

2 4 1 1 6 1

Enter option: 2

Enter the data of the figure:

0 0 0 0 0 0

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> |||(0, 0)(0, 0)(0, 0)(0, 0)|||

-> (2, 4)(1, 1)(6, 1)(5, 4)

Enter option: 12

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> (0, 0)(0, 0)(0, 0)(0, 0)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> ||| |||

Enter option: 2

Enter the data of the figure:

1 3 0 0 3 0

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> (0, 0)(0, 0)(0, 0)(0, 0)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> |||(1, 3)(0, 0)(3, 0)(2, 3)|||

Enter option: 11

Enter option: 1

-> |||(1, 2)(0, 0)(4, 0)(3, 2)|||

-> (0, 0)(0, 0)(0, 0)(0, 0)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 4

(1, 2)(0, 0)(4, 0)(3, 2)

(0, 0)(0, 0)(0, 0)(0, 0)

(2, 4)(1, 1)(6, 1)(5, 4)

(1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 1

-> |||(1, 2)(0, 0)(4, 0)(3, 2)|||

-> (0, 0)(0, 0)(0, 0)(0, 0)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 13

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> |||(0, 0)(0, 0)(0, 0)(0, 0)|||

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 3

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> |||(2, 4)(1, 1)(6, 1)(5, 4)|||

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 12

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

-> ||| |||

Enter option: 14

Enter index: 1

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> |||(2, 4)(1, 1)(6, 1)(5, 4)|||

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 14

Enter index: 3

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

-> ||| |||

Enter option: 5

Enter square: 10

Number of objects with an area less than the specified one: 2

Enter option: 1

-> (1, 2)(0, 0)(4, 0)(3, 2)

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

-> ||| |||

Enter option: 13

ERROR: Trying to go out of range.

Enter option: 14

Enter index: -1

ERROR: Index is out of range.

Enter option: 14

Enter index: 5

ERROR: Index is out of range.

Enter option: 3

ERROR: Trying to delete nonexistent element.

Enter option: 11

Enter option: 1

-> |||(1, 2)(0, 0)(4, 0)(3, 2)|||

-> (2, 4)(1, 1)(6, 1)(5, 4)

-> (1, 3)(0, 0)(3, 0)(2, 3)

Enter option: 8

ERROR: Incorrect command entered.

Enter option: 9

1. Листинг программы

/\*

Гребенков Д. И.

Группа М8О-207Б-19.

Вариант 12.

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, 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 с помощью std::for\_each;

выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

Вариант задания: трапеция, список.

\*/

#include <iostream>

#include <memory>

#include <cmath>

#include <algorithm>

template <typename T>

class Trapezoid {

public:

using type = T;

std::pair <T, T> left\_top, left\_bottom, right\_bottom;

Trapezoid () {}

Trapezoid (T ltx, T lty, T lbx, T lby, T rbx, T rby) {

left\_top.first = ltx;

left\_top.second = lty;

left\_bottom.first = lbx;

left\_bottom.second = lby;

right\_bottom.first = rbx;

right\_bottom.second = rby;

};

};

template <typename T>

void Vertices (Trapezoid <T> tr) {

std::cout << "(" << tr.left\_top.first << ", " << tr.left\_top.second << ")"

<< "(" << tr.left\_bottom.first << ", " << tr.left\_bottom.second << ")"

<< "(" << tr.right\_bottom.first << ", " << tr.right\_bottom.second << ")"

<< "(" << tr.right\_bottom.first - tr.left\_top.first + tr.left\_bottom.first <<

", " << tr.left\_top.second << ")";

}

template <typename T>

T Square (Trapezoid <T> tr) {

return ((tr.right\_bottom.first - tr.left\_top.first) \* (tr.left\_top.second - tr.left\_bottom.second));

}

template <typename T>

class ListMember {

public:

using type = T;

Trapezoid <T> figure;

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

bool barr;

ListMember () {

barr = true;

}

ListMember (T ltx, T lty, T lbx, T lby, T rbx, T rby) {

figure.left\_top.first = ltx;

figure.left\_top.second = lty;

figure.left\_bottom.first = lbx;

figure.left\_bottom.second = lby;

figure.right\_bottom.first = rbx;

figure.right\_bottom.second = rby;

barr = false;

}

};

template <typename T>

class Iterator {

public:

std::weak\_ptr <ListMember <T> > it;

Iterator (std::weak\_ptr <ListMember <T> > al) {

it = al;

}

Iterator \* operator++ () {

std::shared\_ptr <ListMember <T> > temp = it.lock ();

if (temp->next) {

it = temp->next;

}

else {

throw "ERROR: Trying to go out of range.";

}

return this;

}

Trapezoid <T> operator\* () {

std::shared\_ptr <ListMember <int> > temp = it.lock ();

if (!temp->next) {

throw "ERROR: Trying to dereference nullptr.";

}

return (temp->next->figure);

}

bool operator!= (Iterator <T> second) {

bool res;

std::shared\_ptr <ListMember <T> > one = it.lock ();

std::shared\_ptr <ListMember <T> > another = second.it.lock ();

if (one.get () == another.get ()) {

res = false;

}

else {

res = true;

}

return res;

}

};

template <typename T>

class List {

public:

List (std::weak\_ptr <ListMember <T> > b) {

first = b;

last = b;

iter = b;

count = 0;

}

int count;

using type = T;

std::weak\_ptr <ListMember <T> > iter;

std::weak\_ptr <ListMember <T> > first;

std::weak\_ptr <ListMember <T> > last;

Iterator <T> begin () {

return (Iterator <T> (first));

}

Iterator <T> end () {

return (Iterator <T> (last));

}

void insert () {

std::shared\_ptr <ListMember <T> > temp = iter.lock ();

T ltx, lty, lbx, lby, rbx, rby;

std::cout << "Enter the data of the figure: " << std::endl;

std::cin >> ltx >> lty >> lbx >> lby >> rbx >> rby;

std::shared\_ptr <ListMember <T> > curr (new ListMember <T> (ltx, lty, lbx, lby, rbx, rby));

count ++;

if (temp->next) {

std::shared\_ptr <ListMember <T> > copy = temp->next;

temp->next = curr;

curr->next = copy;

}

else {

temp->next = curr;

last = curr;

}

}

void erase () {

std::shared\_ptr <ListMember <T> > temp = iter.lock ();

if (temp->next) {

temp->next = temp->next->next;

count --;

if ((!temp->next) && (!temp->barr)) {

last = temp;

}

}

else {

throw "ERROR: Trying to delete nonexistent element.";

}

}

List \* operator++ () {

std::shared\_ptr <ListMember <T> > temp = iter.lock ();

if (temp->next) {

iter = temp->next;

}

else {

throw "ERROR: Trying to go out of range.";

}

return this;

}

std::weak\_ptr <ListMember <T> > operator[] (int idx) {

std::weak\_ptr <ListMember <T> > pointer = first;

std::shared\_ptr <ListMember <T> > temp;

temp = pointer.lock ();

if ((idx < 0) || (idx > count)) {

throw "ERROR: Index is out of range.";

}

else {

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

temp = pointer.lock ();

pointer = temp->next;

}

temp = pointer.lock ();

}

return (temp);

}

Trapezoid <T> look () {

std::shared\_ptr <ListMember <T> > temp = iter.lock ();

if (!temp->next) {

throw "ERROR: Trying to receive value from nullptr.";

}

return (temp->next->figure);

}

};

double max;

template <typename T>

bool SquareCheck (Trapezoid <T> tr) {

bool res;

if (Square <T> (tr) < max) {

res = true;

}

else {

res = false;

}

return res;

}

void ShowMenu () {

std::cout << "0. Print menu." << std::endl;

std::cout << "1. Show list." << std::endl;

std::cout << "11. Go to the beginning." << std::endl;

std::cout << "12. Go to the ending." << std::endl;

std::cout << "13. Go to the next element." << std::endl;

std::cout << "14. Go to the element with the entered number." << std::endl;

std::cout << "2. Add element in iterator position." << std::endl;

std::cout << "3. Erase element in iterator position." << std::endl;

std::cout << "4. Print figures with \"for\_each\"." << std::endl;

std::cout << "5. Print the number of objects with an area less than the specified one."

<< std::endl;

std::cout << "9. Quit the program." << std::endl;

}

template <class InputIt, class UnaryPredicate>

int count\_if (InputIt first, InputIt last, UnaryPredicate p)

{

int ret = 0;

for (; first != last; ++ first) {

if (p (\*first)) {

ret ++;

}

}

return ret;

}

int main () {

int option = 0;

std::shared\_ptr <ListMember <int> > p (new ListMember <int> ());

List <int> list (p);

ShowMenu ();

while (option != 9) {

std::cout << "Enter option: ";

std::cin >> option;

switch (option) {

case 0: {

ShowMenu ();

break;

}

case 1: {

if (list.count == 0) {

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

}

else try {

std::weak\_ptr <ListMember <int> > temp = list.iter;

std::shared\_ptr <ListMember <int> > check = temp.lock ();

std::shared\_ptr <ListMember <int> > position;

list.iter = list.first; //list.iter = list.begin ();

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

position = list.iter.lock ();

std::cout << " -> ";

if (check == position) {

std::cout << "|||";

}

Vertices (list.look ());

if (check == position) {

std::cout << "|||";

}

std::cout << std::endl;

++list;

list.iter = position->next;

}

if (check->next.get () == nullptr) {

std::cout << " -> ||| |||" << std::endl;

}

else {

std::cout << std::endl;

}

list.iter = temp;

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 11: {

list.iter = list.first;

break;

}

case 12: {

list.iter = list.last;

break;

}

case 13: {

try {

++list;

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 14: {

try {

int idx;

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

std::cin >> idx;

list.iter = list[idx];

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 2: {

try {

list.insert ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 3: {

try {

list.erase ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 4: {

try {

std::for\_each (list.begin (), list.end (), Vertices <int>);

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 5: {

try {

int number = 0;

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

std::cin >> max;

number = count\_if (list.begin (), list.end (), SquareCheck <int>);

std::cout << "Number of objects with an area less than the specified one:"

<< number << std::endl;

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 9: {

break;

}

default: {

std::cout << "ERROR: Incorrect command entered." << std::endl;

break;

}

}

}

return 0;

}

1. Вывод

Изучены основы работы с коллекциями в C++. Получены базовые навыки работы с шаблоном проектирования “Итератор”. Освоена тема “Умные указатели”.

Литература

1. Умный указатель std::shared\_ptr в C++ | Уроки C++ - Ravesli [Электронный ресурс] URL: <https://ravesli.com/urok-194-std-shared_ptr/> (дата обращения 25.11.2020).
2. Умный указатель std::weak\_ptr в C++ | Уроки C++ - Ravesli [Электронный ресурс] URL: <https://ravesli.com/urok-195-std-weak_ptr-tsiklicheskaya-zavisimost-s-std-shared_ptr/> (дата обращения 25.11.2020).
3. std::for\_each - cppreference.com [Электронный ресурс] URL: <https://en.cppreference.com/w/cpp/algorithm/for_each> (дата обращения 25.11.2020)
4. std::count, std::count\_if - cppreference.com [Электронный ресурс] URL: <https://en.cppreference.com/w/cpp/algorithm/count> (дата обращения (25.11.2020).