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: Лагуткина Мария Сергеевна
Группа: М8О-206Б-19
Преподаватель: Чернышов Л. Н.
Дата: 13.11.2020
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 аз границы коллекции или удаление не существующего 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: 14.
Фигура: пятиугольник.
Контейнер: список.

2 Описание программы

Для решения задачи пятиугольник хранится в виде координат одной его вершины и координат центра. Список создается из пятиугольников. Для класса пятиугольника реализованы функции нахождения остальных вершин и подсчета площади пятиугольника. Для работы со списком реализован forward_iterator.
Ввод элемента списка производится после команды insert: сначала вводится индекс по которому нужно вставить элемент, затем элементы пятиугольника: координаты одной вершины и центра. При этом, если индекс указан за пределами конца списка, то элемент будет вставлен в конец списка. Если вставка произошла успешно, выведется «Ok».
Удаление элемента из списка производится командой erase, затем вводится индекс удаляемого пятиугольника. Если указанный индекс выходит за пределы списка, удаление не выполнится, на стандартный поток ошибок выведется «BORDER OVERLAY». Если удаление будет произведено успешно, выведется «Ok».
Печать координат всех пятиугольников, находящихся в списке (в том числе и если список пустой) производится с помощью команды print. Печать количества пятиугольников, у которых площадь меньше заданной, осуществляется с помощью команды area, затем указывается значение.
С помощью команды quit осуществляется выход из программы. Если введенная команда не является одной из указанных выше, на стандартный поток ошибок выводится «INCORECT INPUT» программа переходит в состояние ожидания другой команды.

3 Набор тестов

Программа получает на ввод команду и аргументы к ней. Ознакомится со списком команд можно через команду help:

```
Usage: <element>is first vertex and center in pentagon.
insert <index><element>-insertion new element of list
erase <element>        -delete element of list
print                  -print vertex all pentagons
area <number>          -displaying the number of objects with an area less
than the specified one
```

```
help                    -print usage
quit                    -quit out program
```

Тест №1.
В первом тесте проверяется корректность ввода пятиугольника и вычисления площади и печати. Для пятиугольника с радиусом описанной окружности равным 1 и центром в точке (0,0) площадь примерна равна 2.34. Для пятиугольника с радиусом описанной окружности равным $2\sqrt{2}$ и центром в точке (0,0) площадь примерна равна 19.

```
insert 0
0 1
0 0
insert 1
3 3
1 1
insert 0
0 0
0 0
print
area 0
area 20
area 19
area 3
area 2
quit
```

Тест №2.
Во втором тесте проверяется корректность работы со списком, проверяется работа всех обрабатываемых команд.

```
insert 2
0 2
1 1
insert 0
0 1
1 2
print
erase 2
print
insert 2
2 0
```

```
0 0
insert 10
1.3 4
-34 0
print
erase 2
erase 2
erase 2
erase 0
erase 0
print
qwe
quit
```

4 Результаты выполнения тестов

При каждом запуске программы печатается справка по ее работе. Для улучшения читаемости в данном разделе она не будет приводиться.
Тест №1.

```
insert 0
0 1
0 0
Ok
insert 1
3 3
1 1
Ok
insert 0
0 0
0 0
Ok
print
(0,0)
(0,0)
(0,0)
(0,0)
(0,0)
```

```
(0,1)
(0.951059,0.309008)
(0.587808,-0.809001)
(-0.587733,-0.809055)
(-0.951088,0.30892)

(3,3)
(3.52019,-0.283986)
(0.557744,-1.79364)
(-1.79356,0.557226)
(-0.284453,3.51996)

Ok
area 0
0
Ok
area 20
3
Ok
area 19
2
Ok
area 3
2
Ok
area 2
1
Ok
quit
```

Тест №2.

```
insert 2
0 2
1 1
Ok
insert 0
0 1
1 2
Ok
print
(0,1)
```

```
(1.64205,3.26007)
(2.39681,1.77881)
(1.22132,0.603212)
(-0.260008,1.35783)

(0,2)
(0.358007,-0.260097)
(-0.396819,1.22113)
(0.778613,2.39678)
(2.25998,1.64223)

Ok
erase 2
BORDER OVERLAY
print
(0,1)
(1.64205,3.26007)
(2.39681,1.77881)
(1.22132,0.603212)
(-0.260008,1.35783)

(0,2)
(0.358007,-0.260097)
(-0.396819,1.22113)
(0.778613,2.39678)
(2.25998,1.64223)

Ok
insert 2
2 0
0 0
Ok
insert 10
1.3 4
-34 0
Ok
print
(0,1)
(1.64205,3.26007)
(2.39681,1.77881)
(1.22132,0.603212)
```

```
(-0.260008,1.35783)

(0,2)
(0.358007,-0.260097)
(-0.396819,1.22113)
(0.778613,2.39678)
(2.25998,1.64223)

(2,0)
(0.618104,-1.90209)
(-1.61795,-1.17569)
(-1.61816,1.17539)
(0.617752,1.9022)

(1.3,4)
(-19.2863,-32.3357)
(-60.2054,-23.9868)
(-64.9114,17.5093)
(-26.9011,34.8094)

Ok
erase 2
Ok
erase 2
Ok
erase 2
BORDER OVERLAY
erase 0
Ok
erase 0
Ok
print
Ok
qwe
INCORECT INPUT
quit
```

5 Листинг программы

```
#include <iostream>
#include <cmath>
#include <array>
#include <algorithm>
#include <iterator>
#include <memory>
#include <cmath>
#include <string>

using namespace std;
const double PI = 3.1415;
template <class T>
using vertex_t = pair<T, T>;
template <class T>
istream& operator>> (istream& input, vertex_t<T>& v) {
  input >> v.first >> v.second;
  return input;
}
template<class T>
ostream& operator<< (ostream& output, const vertex_t<T> v) {
  output << "(" << v.first << "," << v.second << ")" << '\n';
  return output;
}
template <class T>
vertex_t<T> operator+(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
  return { lhs.first + rhs.first, lhs.second + rhs.second };
}
template <class T>
vertex_t<T> operator-(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
  return { lhs.first - rhs.first, lhs.second - rhs.second };
}
template <class T>
vertex_t<T> operator/(const vertex_t<T>& lhs, const double& rhs) {
  return { lhs.first / rhs, lhs.second / rhs };
}
template <class T>
//расстояние между двумя точками
double distance(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
  return sqrt((lhs.first - rhs.first) * (lhs.first - rhs.first)
    + (lhs.second - rhs.second) * (lhs.second - rhs.second));
}
template <class T>
class Pentagon {
public:
  vertex_t<T> a, center;
};
template <class T>
```

```
48 vertex_t<T> polar_to_vertex(double ro, double fi) { //переход
49   return { ro * cos(fi), ro * sin(fi) };// из полярной  системы  координат
50 }
51 template <class T>
52 array<vertex_t<T>, 5> find_pentagon_vertexes(const Pentagon<T>& pt) {
53 //определение всех вершин пятиугольника
54   vertex_t<T> ast, b, c, d, e;
55   ast = pt.a - pt.center;
56   double ro = distance(pt.a, pt.center);
57   double fi = 0;
58   if (ast.second >= 0) {
59     if (ast.first != 0) {
60       fi = atan(ast.second / ast.first);
61     }
62     else {
63       fi = PI / 2;
64     }
65   }
66   else {
67     if (ast.first != 0) {
68       fi = atan(ast.second / ast.first) + PI / 2;
69     }
70     else {
71       fi = 3 * PI / 2;
72     }
73   }
74   fi -= 2 * PI / 5;
75   b = polar_to_vertex<T>(ro, fi);
76   fi -= 2 * PI / 5;
77   c = polar_to_vertex<T>(ro, fi);
78   fi -= 2 * PI / 5;
79   d = polar_to_vertex<T>(ro, fi);
80   fi -= 2 * PI / 5;
81   e = polar_to_vertex<T>(ro, fi);
82   const auto& center_of_figure = pt.center;
83   return { pt.a, b + center_of_figure, c + center_of_figure,
84            d + center_of_figure, e + center_of_figure };
85 }
86 template <class T>
87 double area(array<vertex_t<T>, 5> &a) {
88   return abs(a[0].first*a[1].second + a[1].first*a[2].second
89        + a[2].first*a[3].second + a[3].first*a[4].second
90        + a[4].first*a[0].second - a[1].first*a[0].second
91        - a[2].first*a[1].second - a[3].first*a[2].second
92        - a[4].first*a[3].second - a[0].first*a[4].second)/2;
93 }
94 template <class T>
95 istream& operator>> (istream& input, Pentagon<T>& p) {
96   input >> p.a >> p.center;
```

```
97    return input;
98  }
99  template <class T>
100 ostream& operator<< (ostream& output, array<vertex_t<T>,5> &a) {
101   output << a[0] << ' ' << a[1] << ' ' << a[2] <<' '
102          << a[3] << ' ' << a[4] << '\n';
103   return output;
104 }
105
106 template <class T>
107 class List {    //контейнер список
108 private:
109   class List_el;
110   unique_ptr<List_el> first;
111   List_el *tail = nullptr;
112   size_t size = 0;
113   public:
114     class Forward_iterator{
115     public:
116       using value_type = T;
117       using reference = value_type &;
118       using pointer = value_type *;
119       using difference_type = std::ptrdiff_t;
120       using iterator_category = std::forward_iterator_tag;
121       explicit Forward_iterator(List_el *ptr) {
122         it_ptr = ptr;
123       }
124       T &operator*() {
125         return this->it_ptr->value;
126       }
127       Forward_iterator &operator++() {
128         if(it_ptr == nullptr)
129           throw std::length_error("out of list");
130         *this = it_ptr->next();
131         return *this;
132       }
133       Forward_iterator operator++(int) {
134         Forward_iterator old = *this;
135         ++*this;
136         return old;
137       }
138       bool operator==(const Forward_iterator &other) const {
139         return it_ptr == other.it_ptr;
140       }
141       bool operator!=(const Forward_iterator &other) const {
142         return it_ptr != other.it_ptr;
143       }
144     private:
145       List_el *it_ptr;
```

```
146       friend List;
147     };
148     Forward_iterator begin() {
149       return Forward_iterator(first.get());
150     }
151     Forward_iterator end() {
152       return Forward_iterator(nullptr);
153     }
154     void erase(size_t index) {          //удаление элемента по индексу и списка
155       Forward_iterator it = this->begin();
156       for (size_t i = 0; i < index; ++i){
157         ++it;
158       }
159       Forward_iterator begin = this->begin(), end = this->end();
160       if (it == end) { throw length_error("out of border"); }
161       if (it == begin){                    //удаление из начала списка
162         if (size == 0) { throw length_error("can't pop from empty list"); }
163         if (size == 1){
164           first = nullptr;
165           tail = nullptr;
166           --size;
167           return;
168         }
169         unique_ptr<List_el> tmp = std::move(first->next_el);
170         first = std::move(tmp);
171         first->prev_el = nullptr;
172         --size;
173         return;
174       }
175       if (it.it_ptr == tail){     //удаление из конца списка
176         if (size == 0) { throw length_error("can't pop from empty list"); }
177         if (tail->prev_el){
178           List_el *tmp = tail->prev_el;
179           tail->prev_el->next_el = nullptr;
180           tail = tmp;
181         }else{
182           first = nullptr;
183           tail = nullptr;
184         }
185         --size;
186         return;
187       }
188       if (it.it_ptr == nullptr) { throw std::length_error("out of broder"); }
189       auto tmp = it.it_ptr->prev_el;
190       unique_ptr<List_el> temp1 = move(it.it_ptr->next_el);
191       it.it_ptr = it.it_ptr->prev_el;
192       it.it_ptr->next_el = move(temp1);
193       it.it_ptr->next_el->prev_el = tmp;
194       --size;
```

```
195     }
196     void insert(size_t index, T &value) {     //вставка элемента в список
197       Forward_iterator it = this->begin();
198       if (index >= this->size) { it = this->end(); }
199       else {
200         for (size_t i = 0; i < index; ++i) {
201           ++it;
202         }
203       }
204       unique_ptr<List_el> tmp = make_unique<List_el>(value);
205       if (it == this->begin()){     //вставка в начало списка
206         size++;
207         unique_ptr<List_el> tmp = move(first);
208         first = make_unique<List_el>(value);
209         first->next_el = move(tmp);
210         if (first->next_el != nullptr) {
211           first->next_el->prev_el = first.get();
212         }
213         if (size == 1){
214           tail = first.get();
215         }
216         if (size == 2){
217           tail = first->next_el.get();
218         }
219         return;
220       }
221       if (it.it_ptr == nullptr){  //вставка в конец списка
222         if (!size){
223           first = make_unique<List_el>(value);
224           tail = first.get();
225           size++;
226           return;
227         }
228         tail->next_el = make_unique<List_el>(value);
229         List_el *tmp = tail;
230         tail = tail->next_el.get();
231         tail->prev_el = tmp;
232         size++;
233         return;
234       }
235       tmp->prev_el = it.it_ptr->prev_el;
236       it.it_ptr->prev_el = tmp.get();
237       tmp->next_el = std::move(tmp->prev_el->next_el);
238       tmp->prev_el->next_el = std::move(tmp);
239       size++;
240     }
241     List &operator=(List &other) {
242       size = other.size;
243       first = std::move(other.first);
```

```
244     }
245     T &operator[](size_t index) {
246       if (index < 0 || index >= size){
247         throw std::out_of_range("out of list");
248       }
249       Forward_iterator it = this->begin();
250       for (size_t i = 0; i < index; i++){
251         it++;
252       }
253       return *it;
254     }
255 private:
256   class List_el {
257   public:
258     T value;
259     unique_ptr<List_el> next_el;
260     List_el *prev_el = nullptr;
261     List_el() = default;
262     List_el(const T &new_value) : value(new_value) {}
263     Forward_iterator next() {
264       return Forward_iterator(this->next_el.get());
265     }
266   };
267 };
268 namespace Interface {
269   void help() {
270     cout <<
271       "Usage: <element> is first vertex and center in pentagon.\n"
272       "insert <index><element> - insertion new element of list\n"
273       "erase <element>         - delete element of list\n"
274       "print                   - print vertex all pentagons\n"
275       "area <number>           - displaying the number of objects with an area less
      than the specified one\n"
276       "help                    - print usage\n"
277       "quit                    - quit out program\n";
278   }
279 }
280 int main() {
281   string input_s;
282   int input_n;
283   List<Pentagon<double>> list;
284   Interface::help();
285   while (true) {
286     cin >> input_s;
287     if (input_s == "insert") {
288       cin >> input_n;
289       Pentagon<double> p;
290       cin >> p;
291       list.insert(input_n, p);
```

```
292       cout << "Ok\n";
293     }
294     else if (input_s == "erase") {
295       cin >> input_n;
296       try {
297         list.erase(input_n);
298         cout << "Ok\n";
299       }
300       catch (out_of_range) { cerr << "BORDER OVERLAY" << '\n'; }
301       catch(length_error) { cerr << "BORDER OVERLAY" << '\n'; }
302     }
303     else if (input_s == "print") {
304       for_each(list.begin(), list.end(), [](Pentagon<double> &P) {
305         auto a = find_pentagon_vertexes(P);
306         cout << a;
307       }
308       );
309       cout << "Ok\n";
310     }
311     else if (input_s == "area") {
312       cin >> input_n;
313       cout << count_if(list.begin(), list.end(), [=](Pentagon<double> &P) {
314         auto a = find_pentagon_vertexes(P);
315         return area(a) < input_n;
316       }) << '\n';
317       cout << "Ok\n";
318     }
319     else if (input_s == "help") {
320       Interface::help();
321     }
322     else if (input_s == "quit") {
323       return 0;
324     }
325     else { cout << "INCORECT INPUT\n"; }
326   }
327   return 0;
328 }
```

6 GitHub

https://github.com/marianelia/MAI/tree/main/OOP/oop_exercise_05

7 Вывод

Выполняя лабораторную работу, я начала разбираться в устройстве итераторов в C++. Хотя и в начале использовать итераторы практически для любых действий со списком, было непривычно. В этой лабораторной работе я узнала о различных видах итераторов, а также о переборе элементов с помощью for_each.

Список литературы

[1] *Курс «Основы разработки на C++: белый пояс».* [Электронный ресурс] URL: https://www.coursera.org/learn/c-plus-plus-white (дата обращения 10.11.2020).

[2] *Документация Microsoft по C++.* [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/?view=msvc-16 (дата обращения 10.11.2020).