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>
double distance(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {   //расстояние
    между двумя точками
  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>
```

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

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

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

```
194     }
195     void insert(size_t index, T &value) {     //вставка элемента в список
196       Forward_iterator it = this->begin();
197       if (index >= this->size) { it = this->end(); }
198       else {
199         for (size_t i = 0; i < index; ++i) {
200           ++it;
201         }
202       }
203       unique_ptr<List_el> tmp = make_unique<List_el>(value);
204       if (it == this->begin()){     //вставка в начало списка
205         size++;
206         unique_ptr<List_el> tmp = move(first);
207         first = make_unique<List_el>(value);
208         first->next_el = move(tmp);
209         if (first->next_el != nullptr) {
210           first->next_el->prev_el = first.get();
211         }
212         if (size == 1){
213           tail = first.get();
214         }
215         if (size == 2){
216           tail = first->next_el.get();
217         }
218         return;
219       }
220       if (it.it_ptr == nullptr){  //вставка в конец списка
221         if (!size){
222           first = make_unique<List_el>(value);
223           tail = first.get();
224           size++;
225           return;
226         }
227         tail->next_el = make_unique<List_el>(value);
228         List_el *tmp = tail;
229         tail = tail->next_el.get();
230         tail->prev_el = tmp;
231         size++;
232         return;
233       }
234       tmp->prev_el = it.it_ptr->prev_el;
235       it.it_ptr->prev_el = tmp.get();
236       tmp->next_el = std::move(tmp->prev_el->next_el);
237       tmp->prev_el->next_el = std::move(tmp);
238       size++;
239     }
240     List &operator=(List &other) {
241       size = other.size;
242       first = std::move(other.first);
```

```
243     }
244     T &operator[](size_t index) {
245       if (index < 0 || index >= size){
246         throw std::out_of_range("out of list");
247       }
248       Forward_iterator it = this->begin();
249       for (size_t i = 0; i < index; i++){
250         it++;
251       }
252       return *it;
253     }
254 private:
255   class List_el {
256   public:
257     T value;
258     unique_ptr<List_el> next_el;
259     List_el *prev_el = nullptr;
260     List_el() = default;
261     List_el(const T &new_value) : value(new_value) {}
262     Forward_iterator next() {
263       return Forward_iterator(this->next_el.get());
264     }
265   };
266 };
267 namespace Interface {
268   void help() {
269     cout <<
270       "Usage: <element> is first vertex and center in pentagon.\n"
271       "insert <index><element> - insertion new element of list\n"
272       "erase <element>         - delete element of list\n"
273       "print                   - print vertex all pentagons\n"
274       "area <number>           - displaying the number of objects with an area less
    than the specified one\n"
275       "help                    - print usage\n"
276       "quit                    - quit out program\n";
277   }
278 }
279 int main() {
280   string input_s;
281   int input_n;
282   List<Pentagon<double>> list;
283   Interface::help();
284   while (true) {
285     cin >> input_s;
286     if (input_s == "insert") {
287       cin >> input_n;
288       Pentagon<double> p;
289       cin >> p;
290       list.insert(input_n, p);
```

```
      cout << "Ok\n";
    }
    else if (input_s == "erase") {
      cin >> input_n;
      try {
        list.erase(input_n);
        cout << "Ok\n";
      }
      catch (out_of_range) { cerr << "BORDER OVERLAY" << '\n'; }
      catch(length_error) { cerr << "BORDER OVERLAY" << '\n'; }
    }
    else if (input_s == "print") {
      for_each(list.begin(), list.end(), [](Pentagon<double> &P) {
        auto a = find_pentagon_vertexes(P);
        cout << a;
      }
      );
      cout << "Ok\n";
    }
    else if (input_s == "area") {
      cin >> input_n;
      cout << count_if(list.begin(), list.end(), [=](Pentagon<double> &P) {
        auto a = find_pentagon_vertexes(P);
        return area(a) < input_n;
      }) << '\n';
      cout << "Ok\n";
    }
    else if (input_s == "help") {
      Interface::help();
    }
    else if (input_s == "quit") {
      return 0;
    }
    else { cout << "INCORECT INPUT\n"; }
  }
  return 0;
}
```

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

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

[0] [1] *Документация Microsoft по C++.* [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/?view=msvc-16 (дата обращения 10.11.2020).