Московский авиационный институт
(национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №6
Тема: Основы работы с коллекциями: аллокаторы

Студент: Лагуткина Мария Сергеевна
Группа: М8О-206Б-19
Преподаватель: Чернышов Л. Н.
Дата: 27.11.2020
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 доступа:
   - Стек – pop, push, top;
   - Очередь – pop, push, top;
   - Список, Динамический массив — доступ к элементу по оператору [];
   - Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти — является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь).
   - Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих элементов;
   - Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально — vector);

7. Реализовать программу, которая:

   - Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор;
   - Позволяет удалять элемент из коллекции по номеру элемента;
   - выводит на экран введенные фигуры с помощью std::for_each;

Вариант задания: 14.
Фигура: пятиугольник.
Контейнер: список.
Аллокатор: список.

2 Описание программы

Аллокатор памяти — часть программы, обрабатывающая запросы на выделение и освобождение памяти. Основное назначение аллокатора — реализация динамического выделения памяти.
Для решения поставленной задачи был реализован аллокатор, в котором работа с выделенной памятью производится на двусвязном списке. Реализация списка для аллокатора взята из STL. Но для хранения фигур написан отдельный класс List (как и требуется в задании).
Для создания аллокатора используется шаблон, где первым аргументом передается тип хранимых данных, вторым аргументом — размер памяти, которую нужно выделить. Для аллокатора реализован конструктор, деструктор, метод allocate, позволяющий выделять память, метод deallocate, позволяющий освобождать память. Размер выделяемой памяти был задан значением 500.
После создания аллокатора, функционально программа не изменилась, однако аллокатор позволяет более грамотно распоряжаться памятью, выделяемой для хранения фигур.
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

```
1  // Лагуткина Мария Сергеевна
2  //Вариант 14: пятиугольник, список, список
3  #include <iostream>
4  #include <array>
5  #include <algorithm>
6  #include <iterator>
7  #include <memory>
8  #include <cmath>
9  #include <list>
10 #include <exception>
11
12 using namespace std;
13 const double PI = 3.1415;
14
15 template <class T>
16 using vertex_t = pair<T, T>;
17 template <class T>
18 istream& operator>> (istream& input, vertex_t<T>& v) {
19     input >> v.first >> v.second;
20     return input;
21 }
```

```
22 template<class T>
23 ostream& operator<< (ostream& output, const vertex_t<T> v) {
24     output << "(" << v.first << "," << v.second << ")" << '\n';
25     return output;
26 }
27 template <class T>
28 vertex_t<T> operator+(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
29     return { lhs.first + rhs.first, lhs.second + rhs.second };
30 }
31 template <class T>
32 vertex_t<T> operator-(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
33     return { lhs.first - rhs.first, lhs.second - rhs.second };
34 }
35 template <class T>
36 vertex_t<T> operator/(const vertex_t<T>& lhs, const double& rhs) {
37     return { lhs.first / rhs, lhs.second / rhs };
38 }
39 //расстояние между двумя точками
40 template <class T>
41 double distance(const vertex_t<T>& lhs, const vertex_t<T>& rhs) {
42     return sqrt((lhs.first - rhs.first) * (lhs.first - rhs.first)
43         + (lhs.second - rhs.second) * (lhs.second - rhs.second));
44 }
45 template <class T>
46 class Pentagon {
47 public:
48     vertex_t<T> a, center;
49 };
50 //переход из полярной системы координат
51 template <class T>
52 vertex_t<T> polar_to_vertex(double ro, double fi) {
53     return { ro * cos(fi), ro * sin(fi) };
54 }
55 //определение всех вершин пятиугольника
56 template <class T>
57 array<vertex_t<T>, 5> find_pentagon_vertexes(const Pentagon<T>& pt) {
58     vertex_t<T> ast, b, c, d, e;
59     ast = pt.a - pt.center;
60     double ro = distance(pt.a, pt.center);
61     double fi = 0;
62     if (ast.second >= 0) {
63         if (ast.first != 0) {
64             fi = atan(ast.second / ast.first);
65         } else {
66             fi = PI / 2;
67         }
68     } else {
69         if (ast.first != 0) {
70             fi = atan(ast.second / ast.first) + PI / 2;
```

```
71          } else {
72              fi = 3 * PI / 2;
73          }
74      }
75      fi -= 2 * PI / 5;
76      b = polar_to_vertex<T>(ro, fi);
77      fi -= 2 * PI / 5;
78      c = polar_to_vertex<T>(ro, fi);
79      fi -= 2 * PI / 5;
80      d = polar_to_vertex<T>(ro, fi);
81      fi -= 2 * PI / 5;
82      e = polar_to_vertex<T>(ro, fi);
83      const auto& center_of_figure = pt.center;
84      return { pt.a, b + center_of_figure, c + center_of_figure, d + center_of_figure, e
         + center_of_figure };
85  }
86  template <class T>
87  double area(array<vertex_t<T>, 5> &a) {      //подсчет площади
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
91          - a[1].first*a[0].second - a[2].first*a[1].second
92          - a[3].first*a[2].second - a[4].first*a[3].second
93          - a[0].first*a[4].second) / 2;
94  }
95  template <class T>
96  istream& operator>> (istream& input, Pentagon<T>& p) {
97      input >> p.a >> p.center;
98      return input;
99  }
100 template <class T>
101 ostream& operator<< (ostream& output, array<vertex_t<T>, 5> &a) {
102     output << a[0] << ' ' << a[1] << ' ' << a[2] << ' ' << a[3] << ' ' << a[4] <<'\n';
103     return output;
104 }
105 template <class T, size_t ALLOC_SIZE> //ALLOC_SIZE - размер, который требуется
        выделить
106 struct TAllocator {
107 private:
108     char *blocks_begin; //указатель на начало выделенной памяти
109     char *blocks_end;   //указатель на конец выделенной памяти
110     char *blocks_tail;  //указатель на конец заполненного пространства
111     std::list<char *> free_blocks;
112 public:
113     using value_type = T;
114     using size_type = size_t;
115     using pointer = T * ;
116     using const_pointer = const T *;
117     using difference_type = std::ptrdiff_t;
```

```
118     template <class U>
119     struct rebind {
120         using other = TAllocator<U, ALLOC_SIZE>;
121     };
122     //обертка для вызова конструктора: строит объект типа U,
123     //преобразует свои аргументы к соответствующему конструктору
124     template <typename U, typename... Args>
125     void construct(U *p, Args &&... args){
126         new (p) U(std::forward<Args>(args)...);
127     }
128     TAllocator(const TAllocator &) = delete;
129     TAllocator(TAllocator &&) = delete;
130     ~TAllocator() {
131         delete[] blocks_begin;
132     }
133     T *allocate(size_t n) {     // выделение память
134         if (n != 1) {
135             throw logic_error("can't allocate arrays");
136         }
137         if (size_t(blocks_end - blocks_tail) < sizeof(T)) {
138             if (free_blocks.size()) {   //ищем свободное место
139                 char *ptr = free_blocks.front();
140                 free_blocks.pop_front();
141                 return reinterpret_cast<T *>(ptr);
142             }
143             cout << "Bad Alloc" << '\n';
144             throw bad_alloc();
145         }
146         T *result = reinterpret_cast<T *>(blocks_tail); //приведение к типу
147         blocks_tail += sizeof(T);
148         return result;
149     }
150     void deallocate(T *ptr, size_t n) {     // удаляем память
151         if (n != 1) { throw logic_error("can't allocate arrays, thus can't deallocate
    them too"); }
152         if (ptr == nullptr) { return; }
153         free_blocks.push_back(reinterpret_cast<char *>(ptr));
154     }
155 };
156 template <class T, class Allocator = std::allocator<T>>
157 class List {    //контейнер список
158 private:
159     class List_el;
160     unique_ptr<List_el> first;
161     List_el *tail = nullptr;
162     size_t size = 0;
163 public:
164     class Forward_iterator {
165     public:
```

```
166         using value_type = T;
167         using reference = value_type & ;
168         using pointer = value_type * ;
169         using difference_type = std::ptrdiff_t;
170         using iterator_category = std::forward_iterator_tag;
171         explicit Forward_iterator(List_el *ptr) {
172             it_ptr = ptr;
173         }
174         T &operator*() {
175             return this->it_ptr->value;
176         }
177         Forward_iterator &operator++() {
178             if (it_ptr == nullptr)
179                 throw std::length_error("out of list");
180             *this = it_ptr->next();
181             return *this;
182         }
183         Forward_iterator operator++(int) {
184             Forward_iterator old = *this;
185             ++*this;
186             return old;
187         }
188         bool operator==(const Forward_iterator &other) const {
189             return it_ptr == other.it_ptr;
190         }
191         bool operator!=(const Forward_iterator &other) const {
192             return it_ptr != other.it_ptr;
193         }
194     private:
195         List_el *it_ptr;
196         friend List;
197     };
198     Forward_iterator begin() {
199         return Forward_iterator(first.get());
200     }
201     Forward_iterator end() {
202         return Forward_iterator(nullptr);
203     }
204     void erase(size_t index) {  //удаление элемента по индексу и списка
205         Forward_iterator it = this->begin();
206         for (size_t i = 0; i < index; ++i) {
207             ++it;
208         }
209         Forward_iterator begin = this->begin(), end = this->end();
210         if (it == end) { throw length_error("out of border"); }
211         if (it == begin) {                    //удаление из начала списка
212             if (size == 0) { throw length_error("can't pop from empty list"); }
213             if (size == 1) {
214                 first = nullptr;
```

```
215                 tail = nullptr;
216                 --size;
217                 return;
218             }
219             unique_ptr<List_el> tmp = std::move(first->next_el);
220             first = std::move(tmp);
221             first->prev_el = nullptr;
222             --size;
223             return;
224         }
225         if (it.it_ptr == tail) { //удаление из конца списка
226             if (size == 0) { throw length_error("can't pop from empty list"); }
227             if (tail->prev_el) {
228                 List_el *tmp = tail->prev_el;
229                 tail->prev_el->next_el = nullptr;
230                 tail = tmp;
231             }
232             else {
233                 first = nullptr;
234                 tail = nullptr;
235             }
236             --size;
237             return;
238         }
239         if (it.it_ptr == nullptr) { throw std::length_error("out of broder"); }
240         auto tmp = it.it_ptr->prev_el;
241         unique_ptr<List_el> temp1 = move(it.it_ptr->next_el);
242         it.it_ptr = it.it_ptr->prev_el;
243         it.it_ptr->next_el = move(temp1);
244         it.it_ptr->next_el->prev_el = tmp;
245         --size;
246     }
247     void insert(size_t index, T &value) { //вставка элемента в список
248         Forward_iterator it = this->begin();
249         if (index >= this->size) { it = this->end(); }
250         else {
251             for (size_t i = 0; i < index; ++i) {
252                 ++it;
253             }
254         }
255         unique_ptr<List_el> tmp = make_unique<List_el>(value);
256         if (it == this->begin()) {  //вставка в начало списка
257             size++;
258             unique_ptr<List_el> tmp = move(first);
259             first = make_unique<List_el>(value);
260             first->next_el = move(tmp);
261             if (first->next_el != nullptr) {
262                 first->next_el->prev_el = first.get();
263             }
```

```
264             if (size == 1) {
265                 tail = first.get();
266             }
267             if (size == 2) {
268                 tail = first->next_el.get();
269             }
270             return;
271         }
272         if (it.it_ptr == nullptr) { //вставка в конец списка
273             if (!size) {
274                 first = make_unique<List_el>(value);
275                 tail = first.get();
276                 size++;
277                 return;
278             }
279             tail->next_el = make_unique<List_el>(value);
280             List_el *tmp = tail;
281             tail = tail->next_el.get();
282             tail->prev_el = tmp;
283             size++;
284             return;
285         }
286         tmp->prev_el = it.it_ptr->prev_el;
287         it.it_ptr->prev_el = tmp.get();
288         tmp->next_el = std::move(tmp->prev_el->next_el);
289         tmp->prev_el->next_el = std::move(tmp);
290         size++;
291     }
292     List &operator=(List &other) {
293         size = other.size;
294         first = std::move(other.first);
295     }
296     T &operator[](size_t index) {
297         if (index < 0 || index >= size) {
298             throw std::out_of_range("out of list");
299         }
300         Forward_iterator it = this->begin();
301         for (size_t i = 0; i < index; i++) {
302             it++;
303         }
304         return *it;
305     }
306 private:
307     using allocator_type = typename Allocator::template rebind<List_el>::other;
308     struct deleter {
309     private:
310         allocator_type *allocator_;
311     public:
312         deleter(allocator_type *allocator) : allocator_(allocator) {}
```

```
313         void operator()(List_el *ptr) {
314             if (ptr != nullptr) {
315                 std::allocator_traits<allocator_type>::destroy(*allocator_, ptr);
316                 allocator_->deallocate(ptr, 1);
317             }
318         }
319     };
320     struct List_el {
321         T value;
322         unique_ptr<List_el> next_el;
323         List_el *prev_el = nullptr;
324         List_el() = default;
325         List_el(const T &new_value) : value(new_value) {}
326         Forward_iterator next() {
327             return Forward_iterator(this->next_el.get());
328         }
329     };
330 };
331 namespace Interface {
332     void help() {
333         cout <<
334             "Usage: <element> is first vertex and center in pentagon.\n"
335             "insert <index><element> - insertion new element of list\n"
336             "erase <element>         - delete element of list\n"
337             "print                   - print vertex all pentagons\n"
338             "area <number>           - displaying the number of objects with an area
    less than the specified one\n"
339             "help                    - print usage\n"
340             "quit                    - quit out program\n";
341     }
342 }
343 int main() {
344     string input_s;
345     int input_n;
346     List<Pentagon<double>, TAllocator<Pentagon<double>, 500>> list_pentagons;
347     Interface::help();
348     while (true) {
349         cin >> input_s;
350         if (input_s == "insert") {
351             cin >> input_n;
352             Pentagon<double> p;
353             cin >> p;
354             list_pentagons.insert(input_n, p);
355             cout << "Ok\n";
356         }
357         else if (input_s == "erase") {
358             cin >> input_n;
359             try {
360                 list_pentagons.erase(input_n);
```

```
361                 cout << "Ok\n";
362             }
363             catch (out_of_range) { cerr << "BORDER OVERLAY" << '\n'; }
364             catch (length_error) { cerr << "BORDER OVERLAY" << '\n'; }
365         }
366         else if (input_s == "print") {
367             for_each(list_pentagons.begin(), list_pentagons.end(),
368             [](Pentagon<double> &P) {
369                 auto a = find_pentagon_vertexes(P);
370                 cout << a;
371             }
372             );
373             cout << "Ok\n";
374         }
375         else if (input_s == "area") {
376             cin >> input_n;
377             cout << count_if(list_pentagons.begin(), list_pentagons.end(),
378             [=](Pentagon<double> &P) {
379                 auto a = find_pentagon_vertexes(P);
380                 return area(a) < input_n;
381             }) << '\n';
382             cout << "Ok\n";
383         }
384         else if (input_s == "help") {
385             Interface::help();
386         }
387         else if (input_s == "quit") {
388             return 0;
389         }
390         else { cout << "INCORECT INPUT\n"; }
391     }
392     return 0;
393 }
```

6 GitHub

https://github.com/marianelia/MAI/tree/main/OOP/oop_exercise_06

7 Вывод

Выполняя лабораторную работу, я узнала о существовании такого инструмента для работы с памятью, как аллокатор. Аллокатор управляет выделением памяти для контейнера. Хотя и в большинстве случаев удобно использовать аллокатор по умолчанию, я узнала, что также есть возможность написать свой аллокатор.

Список литературы

[1] *Курс «Основы разработки на C++: белый пояс».* [Электронный ресурс] URL: https://www.coursera.org/learn/c-plus-plus-white (дата обращения 20.11.2020).

[2] *Документация Microsoft по C++.* [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/?view=msvc-16 (дата обращения 20.11.2020).