**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: итераторы

Студент:

Группа:

Преподаватель:

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Вариант 16:

|  |  |  |  |
| --- | --- | --- | --- |
| 16. | 8-угольник | Список | Очередь |

1. **Описание программы**

Аллкоатор имеет 3 шаблонных параметра: тип хранимых данных, количество блоков памяти и размер одного блока в байтах. Аллокатор хранит очередь свободных блоков. Когда память выделяется, из очереди извлекается первый блок. Если очередь пуста, вызывается исключение. Когда память освобождается, блок возвращается в очередь.

1. **Набор тестов**

|  |  |  |
| --- | --- | --- |
| № | Описание | Ввод |
| 1 | Демонстрация работы основных функций программы | 2  0  0 0  1  0  2  1  400 400  5  0  2  1  100 100  2  0  2  2  300 300  3  0  4  -1  5  -1  6  -1  7  8  10  8  20  8  100  3  1  4  -1  8  20  0 |
| 2 | Демонстрация обработки исключения | 2  0  0 0  1  0  2  0  0 0  1  0  2  0  0 0  1  0  2  0  0 0  1  0  4  -1  2  0  200 200  200  0  4  -1  3  0  2  0  100 100  100  0  4  -1  0 |

1. **Результаты выполнения тестов.**

*test\_01.txt*:

1. Show commands

2. Add figure

3. Delete figure

4. Center point

5. Print points

6. Size of figure

7. Total size

8. Size less than

0. Exit

> 2

Index to insert figure at (0 - 0): 0

Coordinates of center: 0 0

Radius: 1

Angle: 0

> 2

Index to insert figure at (0 - 1): 1

Coordinates of center: 400 400

Radius: 5

Angle: 0

> 2

Index to insert figure at (0 - 2): 1

Coordinates of center: 100 100

Radius: 2

Angle: 0

> 2

Index to insert figure at (0 - 3): 2

Coordinates of center: 300 300

Radius: 3

Angle: 0

> 4

Index (-1 to call for all figures): -1

0: (4.47035e-08, 3.72529e-08)

1: (100, 100)

2: (300, 300)

3: (400, 400)

> 5

Index (-1 to call for all figures): -1

0: (1, 0), (0.707107, 0.707107), (-4.37114e-08, 1), (-0.707107, 0.707107), (-1, -8.74228e-08), (-0.707107, -0.707107), (1.19249e-08, -1), (0.707107, -0.707107)

1: (102, 100), (101.414, 101.414), (100, 102), (98.5858, 101.414), (98, 100), (98.5858, 98.5858), (100, 98), (101.414, 98.5858)

2: (303, 300), (302.121, 302.121), (300, 303), (297.879, 302.121), (297, 300), (297.879, 297.879), (300, 297), (302.121, 297.879)

3: (405, 400), (403.536, 403.536), (400, 405), (396.464, 403.536), (395, 400), (396.464, 396.464), (400, 395), (403.536, 396.464)

> 6

Index (-1 to call for all figures): -1

0: 2.82843

1: 11.3137

2: 25.4557

3: 70.7104

> 7

Total size of all figures: 110.308

> 8

Maximum size: 10

1 figures has size less than 10

> 8

Maximum size: 20

2 figures has size less than 20

> 8

Maximum size: 100

4 figures has size less than 100

> 3

Index: 1

> 4

Index (-1 to call for all figures): -1

0: (4.47035e-08, 3.72529e-08)

1: (300, 300)

2: (400, 400)

> 8

Maximum size: 20

1 figures has size less than 20

> 0

*test\_02.txt*:

1. Show commands

2. Add figure

3. Delete figure

4. Center point

5. Print points

6. Size of figure

7. Total size

8. Size less than

0. Exit

> 2

Index to insert figure at (0 - 0): 0

Coordinates of center: 0 0

Radius: 1

Angle: 0

> 2

Index to insert figure at (0 - 1): 0

Coordinates of center: 0 0

Radius: 1

Angle: 0

> 2

Index to insert figure at (0 - 2): 0

Coordinates of center: 0 0

Radius: 1

Angle: 0

> 2

Index to insert figure at (0 - 3): 0

Coordinates of center: 0 0

Radius: 1

Angle: 0

> 4

Index (-1 to call for all figures): -1

0: (4.47035e-08, 3.72529e-08)

1: (4.47035e-08, 3.72529e-08)

2: (4.47035e-08, 3.72529e-08)

3: (4.47035e-08, 3.72529e-08)

> 2

Index to insert figure at (0 - 4): 0

Coordinates of center: 200 200

Radius: 200

Angle: 0

Exception caught: bad allocation

> 4

Index (-1 to call for all figures): -1

0: (4.47035e-08, 3.72529e-08)

1: (4.47035e-08, 3.72529e-08)

2: (4.47035e-08, 3.72529e-08)

3: (4.47035e-08, 3.72529e-08)

> 3

Index: 0

> 2

Index to insert figure at (0 - 3): 0

Coordinates of center: 100 100

Radius: 100

Angle: 0

> 4

Index (-1 to call for all figures): -1

0: (100, 100)

1: (4.47035e-08, 3.72529e-08)

2: (4.47035e-08, 3.72529e-08)

3: (4.47035e-08, 3.72529e-08)

> 0

1. **Листинг программы**

#include <iostream>

#include <list>

#include <cmath>

#include <memory>

#include <algorithm>

#include <queue>

#define PI 3.14159265f

//класс восьмиугольника

template<class T>

class Octagon

{

public:

//точка многоугольника

using point = std::pair<T, T>;

//заполняем вектор вершин

Octagon(T x, T y, T r, T a)

{

for (int i = 0; i < 8; i++)

{

T phi = a + i \* PI / 4.0;

m\_points[i] = std::make\_pair(r \* cos(phi) + x, r \* sin(phi) + y);

}

}

//конструктор по умолчанию

Octagon()

{

for (int i = 0; i < 8; i++)

m\_points[i] = std::make\_pair(0, 0);

}

//вычисление геометрического центра фигуры

point getCenter()

{

point center = std::make\_pair((T)0, (T)0);

for (point p : m\_points)

{

center.first += p.first;

center.second += p.second;

}

center.first /= 8.0;

center.second /= 8.0;

return center;

}

//вывод координат вершин фигуры

void print()

{

bool comma = false; //печатать запятую перед точкой или нет

for (point p : m\_points)

{

if (comma) std::cout << ", ";

comma = true;

std::cout << "(" << p.first << ", " << p.second << ")";

}

}

//вычисление площади

T size()

{

T S = (T)0;

for (int i = 0; i < 8 - 1; i++)

S += triag(m\_points[0], m\_points[i], m\_points[i + 1]);

return S;

}

private:

//точки многоугольника

point m\_points[8];

//площадь треугольника по координатам вершин

//S = 1/2 \* abs(det(x1 - x3, y1 - y3; x2 - x3, y2 - y3))

T triag(point& a, point& b, point& c)

{

return 0.5 \* abs((a.first - c.first) \* (b.second - c.second) -

(b.first - c.first) \* (a.second - c.second));

}

};

//аллокатор

//N -- число блоков

//SZ -- их размер

template<class T, size\_t N, size\_t SZ>

class QueueAllocator

{

private:

std::shared\_ptr<char[]> memory; //память, из которой будут браться блоки

std::shared\_ptr<std::queue<char\*>> freeBlocks; //свободные блоки памяти

public:

//необходимые объявления для корректной работы C++ с данным аллокатором

using value\_type = T;

using pointer = T\*;

using const\_pointer = const T\*;

using size\_type = std::size\_t;

template<typename U>

struct rebind {

using other = QueueAllocator<U, N, SZ>;

};

//функции для получения параметров аллокатора

//необходимы при копировании аллокатора

std::shared\_ptr<char[]> GetMemory() const { return memory; }

std::shared\_ptr<std::queue<char\*>> GetBlocks() const { return freeBlocks; }

//создание аллокатора

QueueAllocator() noexcept

{

//выделяем память

char\* start = new char[N \* SZ];

memory = std::shared\_ptr<char[]>(

start, //N блоков

[](char\* mem) { delete[] mem; } //деструктор, вызываемый при занулении всех ссылок

);

//заполняем очередь

std::queue<char\*> blocks;

for (int i = 0; i < N; i++) blocks.push(start + i \* SZ);

freeBlocks = std::make\_shared<std::queue<char\*>>(blocks);

}

//копирование аллокатора

template<class U>

QueueAllocator(const QueueAllocator<U, N, SZ>& alloc) noexcept

{

memory = alloc.GetMemory();

freeBlocks = alloc.GetBlocks();

}

~QueueAllocator() {}

//выделить память

T\* allocate(size\_t n)

{

//список блоков

std::queue<char\*>& blocks = \*freeBlocks.get();

//вся память израсходована

if (blocks.empty())

{

throw std::bad\_alloc();

return nullptr;

}

//возвращаем первый элемент очереди

T\* ptr = (T\*)blocks.front();

blocks.pop();

return ptr;

}

//освободить память

void deallocate(T\* ptr, size\_t n)

{

//возвращаем использованный блок

freeBlocks.get()->push((char\*)ptr);

}

//вызвать конструктор объекта

template<typename U, typename ...Args>

void construct(U\* ptr, Args &&...args)

{

new (ptr) U(std::forward<Args>(args)...);

}

//вызвать деструктор объекта

void destroy(pointer ptr)

{

ptr->~T();

}

};

template<class T, size\_t N1, size\_t SZ1, class U, size\_t N2, size\_t SZ2>

bool operator==(QueueAllocator<T, N1, SZ1>&, QueueAllocator<U, N2, SZ2>&) noexcept { return true; }

template<class T, size\_t N1, size\_t SZ1, class U, size\_t N2, size\_t SZ2>

bool operator!=(QueueAllocator<T, N1, SZ1>&, QueueAllocator<U, N2, SZ2>&) noexcept { return false; }

//реализация списка

template<class T, class ALLOCATOR = std::allocator<T>>

class List

{

private:

//элемент списка

struct Node

{

T data; //сам объект, хранимый списком

std::shared\_ptr<Node> next; //следующий элемент

std::weak\_ptr<Node> prev; //предыдущий элемент

Node() {}

Node(T d) : data(d) {}

Node(T d, std::shared\_ptr<Node> n) : data(d), next(n) {}

Node(T d, std::shared\_ptr<Node> n, std::shared\_ptr<Node> p) : data(d), next(n), prev(p) {}

Node(T d, std::shared\_ptr<Node> n, std::weak\_ptr<Node> p) : data(d), next(n), prev(p) {}

};

//первый элемент и элемент после последнего

std::shared\_ptr<Node> first, terminal;

//аллокатор

ALLOCATOR allocator;

public:

//итератор списка

class iterator

{

private:

std::weak\_ptr<List::Node> ptr;

friend class List; //чтобы список имел доступ к переменной ptr

public:

using difference\_type = int;

using value\_type = T;

using reference = T&;

using pointer = T\*;

using iterator\_category = std::forward\_iterator\_tag;

//операторы для итератора

reference operator\*()

{

//если итератор указывает на терминирующий элемент, генерируем исключение

if (!ptr.lock()->next) throw std::out\_of\_range("trying to get value of unexisting element");

return ptr.lock()->data;

}

pointer operator->()

{

//если итератор указывает на терминирующий элемент, генерируем исключение

if (!ptr.lock()->next) throw std::out\_of\_range("trying to access unexisting element");

return &(ptr.lock()->data);

}

iterator& operator++()

{

//мы не можем пройти дальше, чем терминирующий элемент

if (!ptr.lock()->next) throw std::out\_of\_range("moving farther than terminal element");

//переходим к следующему элементу списка

ptr = (\*ptr.lock()).next;

return \*this;

}

bool operator!=(const iterator& other)

{

//итераторы не равны, если они указывают на разные элементы

return ptr.lock() != other.ptr.lock();

}

};

friend class iterator; //чтобы итератор имел доступ к классу Node

//создание списка

List()

{

//первый и терминирующий элемент равны

first = terminal = std::allocate\_shared<Node>(allocator, Node());

}

~List() {}

//итераторы на первый и терминирующий элементы

iterator begin()

{

iterator i;

i.ptr = first;

return i;

}

iterator end()

{

iterator i;

i.ptr = terminal;

return i;

}

//вставка элемента перед элементом, на который указывает итератор

void insert(iterator iter, const T& val)

{

//вставка перед первым элементом (ссылка на предыдущий элемент пустая)

if (!(iter.ptr.lock()->prev.lock()))

{

//новый элемент будет первым

//его следующий элемент -- предыдущий первый элемент

first = std::allocate\_shared<Node>(allocator, Node(val, first));

//создаём ссылку на первый элементр у второго

first->next->prev = first;

}

//вставка в середине списка

else

{

//создаём новый элемент

auto el = std::allocate\_shared<Node>(allocator, Node(val, iter.ptr.lock(), iter.ptr.lock()->prev));

//переставляем ссылки у предыдущего и следующего элемента

el->prev.lock()->next = el;

el->next->prev = el;

}

}

//удаление элемента из списка

iterator erase(iterator iter)

{

//нельзя удалить терминирующий элемент

if (iter.ptr.lock() == terminal) throw std::out\_of\_range("impossible to remove terminal element");

//возвращаемое значение -- итератор после удаляемого элемента

iterator ret\_val = iter;

++ret\_val;

//удаление первого элемента

if (!(iter.ptr.lock()->prev.lock()))

{

//зануляем ссылку на предыдущий элемент у второго элемента

first->next->prev = std::weak\_ptr<Node>();

//теперь второй элемент является первым

first = first->next;

}

//удаление в середине списка

else

{

//удаляемый элемент

auto el = iter.ptr.lock();

//перекидываем ссылки через удаляемый элемент

el->next->prev = el->prev;

el->prev.lock()->next = el->next;

}

return ret\_val;

}

//размер списка

size\_t size() const

{

size\_t sz = 0;

//начинаем с первого элемента

std::weak\_ptr<Node> w = first;

//переходим к следующему значению, пока не достигнем терминального элемента

while (w.lock() != terminal)

{

w = w.lock()->next;

sz++;

}

//длина списка -- количество пройденных шагов

return sz;

}

//элемент по индексу

T operator[](int i) { return \*std::next(begin(), i); }

};

//список команд

void showCommands()

{

std::cout <<

"1. Show commands" << std::endl <<

"2. Add figure" << std::endl <<

"3. Delete figure" << std::endl <<

"4. Center point" << std::endl <<

"5. Print points" << std::endl <<

"6. Size of figure" << std::endl <<

"7. Total size" << std::endl <<

"8. Size less than" << std::endl <<

"0. Exit" << std::endl;

}

int main()

{

//список фигур

List<Octagon<float>, QueueAllocator<Octagon<float>, 5, 256>> figures;

showCommands();

//цикл программы

bool loop = true;

while (loop)

{

//читаем введённую команду

std::cout << "> ";

int command;

std::cin >> command;

switch (command)

{

case 0:

loop = false;

break;

case 1:

showCommands();

break;

case 2:

{

int index, size = figures.size();

std::cout << "Index to insert figure at (0 - " << size << "): ";

std::cin >> index;

if (index < 0 || index > size)

std::cout << "Index out of bounds" << std::endl;

else

{

float x, y, r, a;

std::cout << "Coordinates of center: ";

std::cin >> x >> y;

std::cout << "Radius: ";

std::cin >> r;

std::cout << "Angle: ";

std::cin >> a;

try

{

figures.insert(std::next(figures.begin(), index), Octagon<float>(x, y, r, a));

}

catch (std::exception &e)

{

std::cout << "Exception caught: " << e.what() << std::endl;

}

}

break;

}

case 3:

{

std::cout << "Index: ";

int index;

std::cin >> index;

if (index < 0 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

figures.erase(std::next(figures.begin(), index));

}

break;

}

case 4:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

{

int i = 0;

std::for\_each(figures.begin(), figures.end(), [&i](Octagon<float>& o)

{

std::pair<float, float> p = o.getCenter();

std::cout << i << ": (" << p.first << ", " << p.second << ")" << std::endl;

i++;

});

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

std::pair<float, float> p = figures[index].getCenter();

std::cout << index << ": (" << p.first << ", " << p.second << ")" << std::endl;

}

break;

}

case 5:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

{

int i = 0;

std::for\_each(figures.begin(), figures.end(), [&i](Octagon<float>& o)

{

std::cout << i << ": ";

o.print();

std::cout << std::endl;

i++;

});

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

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

figures[index].print();

std::cout << std::endl;

}

break;

}

case 6:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

{

int i = 0;

std::for\_each(figures.begin(), figures.end(), [&i](Octagon<float>& o)

{

std::cout << i << ": " << o.size() << std::endl;

i++;

});

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

std::cout << index << ": " << figures[index].size() << std::endl;

}

break;

}

case 7:

{

float total = 0.0f;

for (Octagon<float> &f : figures) total += f.size();

std::cout << "Total size of all figures: " << total << std::endl;

break;

}

case 8:

{

std::cout << "Maximum size: ";

float maxSz;

std::cin >> maxSz;

std::cout << std::count\_if(figures.begin(), figures.end(),

[&maxSz](Octagon<float>& o)

{

return o.size() < maxSz;

}) << " figures has size less than " << maxSz << std::endl;

break;

}

default:

std::cout << "Unknown command" << std::endl;

break;

}

}

return 0;

}

1. **Выводы:**

Изучены основы работы с контейнерами и концепцией аллокаторов памяти. Разработан собственный аллокатор на основе очереди, работающий со списком фигур.

**СПИСОК ЛИТЕРАТУРЫ**

1. Альтернативные аллокаторы памяти [электронный ресурс]. URL: <https://habr.com/ru/post/274827/>
2. Управление памятью в C++ [электронный ресурс]. URL: <https://habr.com/ru/post/148657/>