Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «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: Павлова К.А.
Группа: М8О-306Б-18
Преподаватель:
Дата:
Оценка:

Москва, 2021

1. Постановка задачи

Вариант 16:

16.	8-угольник	Список

2. Описание программы

Программа реализует двунаправленный список с терминальным элементом и forward_iterator для этого списка. Список имеет все основные функции: insert, erase, begin, end и оператор [] для обращения по индексу. Данная реализация поддерживает работу с count_if. Программа реализует пользовательский интерфейс для работы со списком.

3. Набор testcases

№	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

4. Результаты выполнения тестов.

```
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
```

5. Листинг программы

```
#include <iostream>
#include <list>
#include <cmath>
#include <memory>
#include <algorithm>

#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));
        }
};

//реализация списка
template<class 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;

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::make_shared<Node>(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::make_shared<Node>(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::make_shared<Node>(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>> 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;

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

6. Выводы:

Изучение основ работы с коллекциями и с шаблоном проектирования "Итератор". Разработана программа на языке C++, реализующая список и обеспечивающая работу с ним.

СПИСОК ЛИТЕРАТУРЫ

1. Smart pointers для начинающих [электронный ресурс]. URL: https://habr.com/ru/post/140222/
2. Контейнеры [электронный ресурс]. URL: https://ru.cppreference.com/w/cpp/container