

**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:  
Аллокаторы**

Студент: Вадим Валов

Группа: 80-208

Преподаватель:

Дата:

Оценка:

Москва, 2019

## 1. Постановка задачи

Создать шаблон динамической коллекции, 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;

3. Коллекция должна содержать метод доступа:

- oСтек – `pop`, `push`, `top`;

- oОчередь – `pop`, `push`, `top`;

- oСписок, Динамический массив – доступ к элементу по оператору `[]`;

4. 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);

5. Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих элементов.

6. Аллокатор должен быть совместим с контейнерами `std::map` и `std::list` (опционально – `vector`).

7. Реализовать программу, которая:

- oПозволяет вводить с клавиатуры фигуры (с типом `int` в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор;

- oПозволяет удалять элемент из коллекции по номеру элемента;

- oВыводит на экран введенные фигуры с помощью `std::for_each`;

Создать набор шаблонов, создающих функции, реализующие:

1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода `std::cout` координат вершин фигуры;
3. Вычисление площади фигуры;

## 2. Описание программы

Программа содержит шаблоны классов `Sixthangle`, `DynamicArray`. Динамический массив, принимает фигуру, которую должен хранить. `Forward iterator` принимает класс на который должен указывать. `Sixthangle` принимает тип данных в котором будут храниться вершины.

`Sixthangle` содержит функции вычисления площади, функцию печати вершин и функцию расчета расстояния между 2 точками.

Программа хранит данные в виде Динамического массива, который позволяет обратиться к элементу по индексу. Удалить элемент по индексу. Вставить элемент по индексу.

Аллокатор представляет собой класс, который выделяет память в начале работы аллокатора и освобождает всю память после окончания работы. Аллокатор на стеке предполагает, что при использовании аллокатора память будет выделяться в стиле LIFO и освобождать также.

Интерфейс взаимодействия с программой предоставляет пользователю 4 различных команд для исполнения. При вводе чисел от 1-4 пользователь выбирает команду для исполнения. Функционал программы поддерживает работу со стандартными `std::for_each` и `std::count_if`.

Набор testcases

Тестам на вход подаются команды для выполнения и вершины

Команды: 1.add 2.delete 3.Print 4.Print less then

**Тест 1:**

1 // Команда  
1,2 2,3 3,4 4,5 // Координаты  
1 //Команда  
2,3 3,4 4,5 5,6 // Координаты  
2 // команда  
1 // Удаление  
3 // Команда  
4 // команда  
10 // Аргумент

## **Тест 2:**

1  
2,4 3,5 4,6 7,9  
1  
3,3 5,4 8,5 1,6  
1  
1,2 2,3 3,4 4,5  
1  
2,3 3,4 4,5 5,6  
2  
3  
3  
4  
10

## **Тест 3:**

1

1,2 2,3 3,4 4,5

1

2,3 3,4 4,5 5,6

1

2,4 3,5 4,6 7,9

1

3,3 5,4 8,5 1,6

2

3

3

4

10

### 3. Результаты выполнения тестов.

#### **Тест 1:**

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>1,2 2,3 3,4 4,5

Enter command:

1.add

2.delete

3.Print

4.Print less than

>1

Enter 4 vertices

>2,3 3,4 4,5 5,6

Enter command:

1.add

2.delete

3.Print

4.Print less than

>2

Enter index to delete

>1

Enter command:

1.add

2.delete

3.Print

4.Print less than

>3

<1, 2>

<2, 3>

<3, 4>

<4, 5>

<2, 3>

<3, 4>

<4, 5>

<5, 6>

Enter command:

1.add

2.delete

3.Print

4.Print less then

>4

Enter area. Programm will print Romds vertices < n

n = 10

Count = 2

Enter command:

1.add

2.delete

3.Print

4.Print less then

>-1

Exit programm...

## **Тест 2:**

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>2,4 3,5 4,6 7,9

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>3,3 5,4 8,5 1,6

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>1,2 2,3 3,4 4,5

Enter command:

1.add

2.delete



3.Print

4.Print less than

>1

Enter 4 vertices

>2,3 3,4 4,5 5,6

Enter command:

1.add

2.delete

3.Print

4.Print less than

>2

Enter index to delete

>3

Enter command:

1.add

2.delete

3.Print

4.Print less than

>3

<2, 4>

<3, 5>

<4, 6>

<7, 9>

<3, 3>

<5, 4>

<8, 5>

<1, 6>

<1, 2>

<2, 3>

<3, 4>

<4, 5>

<2, 3>

<3, 4>

<4, 5>

<5, 6>

Enter command:

1.add

2.delete

3.Print

4.Print less then

>4

Enter area. Programm will print Romds vertices < n

n = 10

Count = 3

Enter command:

1.add

2.delete

3.Print

4.Print less then

>-1

Exit programm...

### **Тест 3:**

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>1,2 2,3 3,4 4,5

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>2,3 3,4 4,5 5,6

Enter command:

1.add

2.delete

3.Print

4.Print less then

>1

Enter 4 vertices

>2,4 3,5 4,6 7,9

Enter command:

1.add

2.delete

3.Print

4.Print less than

>1

Enter 4 vertices

>3,3 5,4 8,5 1,6

Enter command:

1.add

2.delete

3.Print

4.Print less than

>2

Enter index to delete

>3

Enter command:

1.add

2.delete

3.Print

4.Print less than

>3

<1, 2>

<2, 3>

<3, 4>

<4, 5>

<2, 3>

<3, 4>

<4, 5>

<5, 6>

<2, 4>

<3, 5>

<4, 6>

<7, 9>

<3, 3>

<5, 4>

<8, 5>

<1, 6>

Enter command:

1.add

2.delete

3.Print

4.Print less then

>4

Enter area. Programm will print Romds vertices < n

n = 10

Count = 3

Enter command:

1.add

2.delete

3.Print

4.Print less then

>-1

Exit programm...

#### 4. Листинг программы

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "stackallocator.h"
#include "dynamicArray.h"

int main()
{
    DynamicArray<SixthAngle> arr;
    StackAllocator<SixthAngle, sizeof(SixthAngle)> allocator;

    SixthAngle a;
    int command = 1;
    while (command > 0)
    {
        SixthAngle a;
        int k, n;
        std::cout << "Enter command:\n 1.add new sixthangle\n
2.delete sixthangle from array\n 3.Print all sixthangles\n>";
        std::cin >> command;
```

```

switch (command)
{
case 1:
    std::cout << "Enter 6 vertices: Format for enter -
X*any_symbol*Y Example: 4.5\n>";
    try
    {
        arr.add(allocator.Allocate());
        std::cin >> (*arr.end());
        std::cout << arr.end()->a.first << std::endl;
    }
    catch (std::bad_alloc a)
    {
        std::cout << "Not enough storage size. Delete some
of objects to add new\n";
    }
    break;
case 2:
    std::cout << "Enter index to delete\n>";
    std::cin >> k;
    try
    {
        DynamicArray<SixthAngle>::iterator it =
arr.returnIterator(k);
        arr.erase(it);
        allocator.DeAllocate(it);
    }
    catch (int a)
    {
        if (a == OUT_OF_RANGE)
            std::cout << "ERROR: Out of range\n";
        if (a == DOES_NOT_EXIST)
            std::cout << "ERROR: Does not exist\n";
        if (a == ITERATOR_DONT_EXIST)
            std::cout << "ERROR: No iterator in this
array\n";
        if (a == TRY_TO_DELETE_EMPTY)
            std::cout << "ERROR: Position is empty\n";
    }
    break;
case 3:
    try
    {
        for (int i = 0; i < arr.size(); i++)
        {
            if (arr[i] != arr._empty)
            {
                std::cout << "Sixthangle index - " << i <<

```

```

std::endl;
                                arr[i]->printVertices();
                                }
                                }
                                }
catch (int a)
{
    if (a == OUT_OF_RANGE)
        std::cout << "ERROR: Out of range\n";
    if (a == DOES_NOT_EXIST)
        std::cout << "ERROR: Does not exist\n";
    if (a == ITERATOR_DONT_EXIST)
        std::cout << "ERROR: No iterator in this
array\n";
    if (a == TRY_TO_DELETE_EMPTY)
        std::cout << "ERROR: Position is empty\n";
    }

    break;
default:
    std::cout << "Exit programm...\n";
    break;
}
}
return 0;
}

```

## 6. Вывод

Аллокаторы это мощный инструмент для работы с памятью, который позволяет работать с ней достаточно быстро не тратя процессорное время на дополнительное выделение.

Так или иначе для общих случаев проще использовать стандартные методы выделения памяти, и только в частных прибегать к использованию аллокаторов