Лабораторная №5

Вам требуется написать свой аллокатор, а также произвести сравнение его производительности с std::allocator Принцип работы аллокатора должен быть следующий:

1. Аллокатор при конструировании выделяет большой объем памяти. Освобождение происходит только в конце “жизни”. Никакие аллокаций памяти в процессе “работы” быть не должно
2. Вся память должна быть “разбита” на “кусочки”. Размеры кусочков, а также их количество параметризуется. “Кусочки” одинакового размера разумно объединить в группы
3. При запросе на выделение памяти размера “N” аллокатор ищет свободный подходящий кусочек нужного размера (ближайший свободный, подходящий по размеру). Если такого нет, то происходит исключение
4. При освобождении, аллокатор возвращает кусочек к списку свободных.
5. Ваш аллокатор должен соответствовать требованиям к аллокаторам для C++17

main.cpp

```
#include <iostream>
#include<vector>
#include "allocator.h"
int main()
{

    std::vector<int, MyAllocator<int>> g(85);
    g[0] = 1;
    return 0;
}
```

allocator.h

```
#ifndef LAB5_ALLOCATOR_H
#define LAB5_ALLOCATOR_H

#include <algorithm>
#include <cmath>
#include  <vector>
#define MEM_ALLOC 100
#define k 10
template<typename T>
class MyAllocator
{
public:
    typedef size_t size_type ;
    typedef ptrdiff_t difference_type ;
    typedef T* pointer;
```

```
    typedef const T* const_pointer ;
    typedef T& reference ;
    typedef const T& const_reference ;
    typedef T value_type ;
    MyAllocator()
    {
        data = (T*)malloc(sizeof(T) * MEM_ALLOC);
    }
    pointer allocate(size_t size)
    {
        std::cout << "Allocate" << std::endl;
        pointer current = data;
        std::cout << len << std::endl;
        int needed = size / len + (size % len != 0);
        for (int i = 0; i < k; i++)
        {
            if (!used[i])
            {
                bool check = true;
                std::cout << needed << std::endl;
                for (int j = i + 1; j < i + needed - 1; j++)
                {
                    std::cout << j << ' ';
                    if (used[j])
                    {
                        check = false;
                    }
                }
                std::cout << '\n';
                if (check)
                {
                    for(int j = i; j < i + needed; j++)
                    {
                        used[j] = true;
                    }
                    break;
                }
            }
            current += len;
        }
        std::cout << data << ' ' << current;
        return current;
    }
    void deallocate(pointer p, size_t n)
    {

        int needed = n / len + (n % len != 0), pos = 0;
        pointer current = data;
        while (current != p)
        {
            pos++;
            current = current + len;
        }
        for (int i = 0; i < needed; i++)
```

```
        {
            used[pos] = false;
            pos++;
        }
        std::cout << "Deallocate" << std::endl;
    }

private:
    T* data;
    bool used[k];
    size_t len = MEM_ALLOC / k;

};

#endif
```

Ввод и вывод

Программа показывает результат взаимодействия с аллокатором.

Вывод

Использую принципы ООП, я выполнил предложенное мне задание, понял, каким образом реализовано выделение памяти в STL.