**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

# Программирование

Лабораторная работа №5

*Allocator.*

**Выполнил студент группы № M3109**
Гумбатов Владислав Юрьевич
**Подпись:**

Санкт-Петербург
2022

**Задача**

Целью лабораторной работы является написание собственного аллокатора, а также сравнение его производительности с std::allocator.

**Входные данные:**

Входными данными в программе является набор ячеек размеров памяти.

**Выходные данные:**

В качестве выходных данных программа выдает запрошенную память.

**Код:**

```
#include <list>
#include <iostream>
#include <typeinfo>
#include <algorithm>
#include <cmath>
#include <vector>


template <class T>
class alloc
{


public:

    typedef T value_type;

    alloc(std::vector<int> listA) { // in first version it was for list

        list = listA;

        setlocale(LC_ALL, "Russian");

        sort(list);

        sizeForAllocate(list);

        if (list.size() != 0) {

            std::vector<int>::iterator endIter = list.end();

            --endIter;

            sizeOfLast = *endIter;

        }

        std::vector<int>::iterator iter = list.begin();

        value_type* point = (static_cast<int *>(malloc(allSize * sizeof(T))));


        for (int i = 0; i < list.size(); i++) {

            this->dinArr[i] = point;

            point += *iter * sizeof(T);

            listOfLastIndex.push_back(0);

            if (*iter % 10 == 1) {

                std::cout << "Выделил " << *iter << " блок памяти под тип " << typeid(T).name() << std::endl;
```

```
        }

        else if (*iter % 10 == 0) {

            std::cout << "Выделил " << *iter << " блоков памяти под тип " << typeid(T).name() << std::endl;
        }

        else if (*iter % 2 == 0 || *iter % 3 == 0 || *iter % 4 == 0) {

            std::cout << "Выделил " << *iter << " блока памяти под тип " << typeid(T).name() << std::endl;

        }

        else if (*iter % 10 == 0 || *iter % 12 == 0 || *iter % 13 == 0 || *iter % 14 == 0
        || *iter % 15 == 0 || *iter % 16 == 0 || *iter % 17 == 0 || *iter % 18 == 0 || *iter % 19 == 0) {

            std::cout << "Выделил " << *iter << " блоков памяти под тип " << typeid(T).name() << std::endl;

        }

        else {

            std::cout << "Выделил " << *iter << " блока памяти под тип " << typeid(T).name() << std::endl;
        }

        ++iter;

        //secDinArr[i] = true;

    }


    std::cout << "Выделение заверешно. Размер последнего блока: " << sizeOfLast << std::endl;

}



T* allocate(size_t search) {
    if (this->allSize == 0)
    {
        return static_cast<T*>(malloc(search * sizeof(T)));
    }

    auto elementId = binsearch(list, search);

    std::cout << "Нашел блок " << search << " на позиции " << elementId + 1 << std::endl;

    secDinArr[elementId] = true;


    return dinArr[elementId];
```

```
    template<class Y> alloc(const alloc<Y>& other) {}

    void deallocate(T *point, size_t val) {

        for (int i = 0; i < list.size(); i++) {
            if (dinArr[i] == point) {
                std::cout << "Деллоцировал блок " << list[i] << " по адрессу " << point << std::endl;
                secDinArr[i] = false;
                listOfLastIndex[i] = i;
            }
        }
    }


    //~alloc() {
    //  std::cout << "Destructed " << std::endl;
    //  list.clear();
    //  listOfLastIndex.clear();
    //}

private:
    void sizeForAllocate(std::vector<int> &list) {
        std::vector<int>::iterator iter = list.begin();

        for (int i = 0; i < list.size(); i++) {

            this->allSize += *iter;
            ++iter;
        }
    }
    void sort(std::vector<int> &list) {
```

```
void sort(std::vector<int> &list) {


    //list.sort(); was for list
    std::sort(list.begin(), list.end());


}
int binsearch(std::vector<int> &list, const int &search) {

    auto findElem = std::lower_bound(list.begin(), list.end(), search); // Левый бин поиск за O(logn)
    std::vector<int>::iterator end = list.end();
    end--;
    if (findElem > end) {
        std::cout << "Такого блока нет" << std::endl;
        throw std::logic_error("Такого блока нет");
    }
    auto elementIndex = std::distance(list.begin(), findElem);
    if (secDinArr[elementIndex] == false) {
            listOfLastIndex[elementIndex] = elementIndex;

    }
    if (list[elementIndex] != search) {
        std::cout << "Такого блока нет" << std::endl;
        throw std::logic_error("Такого блока нет");
    }
    if (secDinArr[elementIndex] == true) { //Если первый найденный уже занят
        if (elementIndex + 1 == list.size()) {
            std::cout << "Все блоки этого типа заняты" << std::endl;
            throw std::logic_error("Все блоки этого типа заняты");
        }
```

```
            for (int i = listOfLastIndex[elementIndex]; i < list.size(); i++) { // Поиск следующего за O(1)
                if (list[i] != search) {
                    std::cout << "Все блоки этого типа заняты" << std::endl;
                    throw std::logic_error("Такого блока нет");
                }
                if (list[i] == search && (secDinArr[i] == false)) {
                    listOfLastIndex[elementIndex] = i;
                    elementIndex = i;
                    listOfLastIndex[elementIndex] = i;
                    break;
                }
            }
        }
        return elementIndex;
    }

    bool secDinArr[100] = { false };
    int numberOfCells;
    int sizeOfLast;
    int allSize = 0;
    T* dinArr[100];
    std::vector<int> list;
    std::vector<int> listOfLastIndex;
    int lastIndex;
};
```

```
int main() {


    //std::list<int> b{ 5,4,6 }; much longer with list

    std::vector<int> c{ 10,20,30,30,30,40,42 };

    //alloc<int> A(c);

    //auto b = A.allocate(30);

    ////A.deallocate(b);

    //A.allocate(30);

    //A.allocate(30);

    //A.allocate(10);

    //A.allocate(40);

    std::vector<int, alloc<int>> d(c);

    d.reserve(10);



}
```

**Вывод:**

Программа успешно работает. Аллокатор успешно выделяет запрошенную память, было проверено выделение памяти на векторе и листе. Поиск свободного участка был реализован с помощью бинарного поиска, за счет чего сложность поиска O(logn), а значит аллокатор работает быстрее стандартного.