

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Пермский национальный исследовательский политехнический
университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа №6

" АТД. Контейнеры "

Вариант: 12

Выполнил студент ИВТ-24-26:
Шишкин Максим Григорьевич

(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2025

Содержание

1 Постановка задачи	3
2 Код на C++	4-5
3 Результаты работы	6
4 UML-диаграмма классов	6
5 Ответы на контрольные вопросы	7
6 Ссылка на github.....	7

1 Постановка задачи

1. Определить класс-контейнер.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
5. Написать тестирующую программу, иллюстрирующую выполнение операций.

Вариант: Класс- контейнер СПИСОК с ключевыми значениями типа `int`.
Реализовать операции:

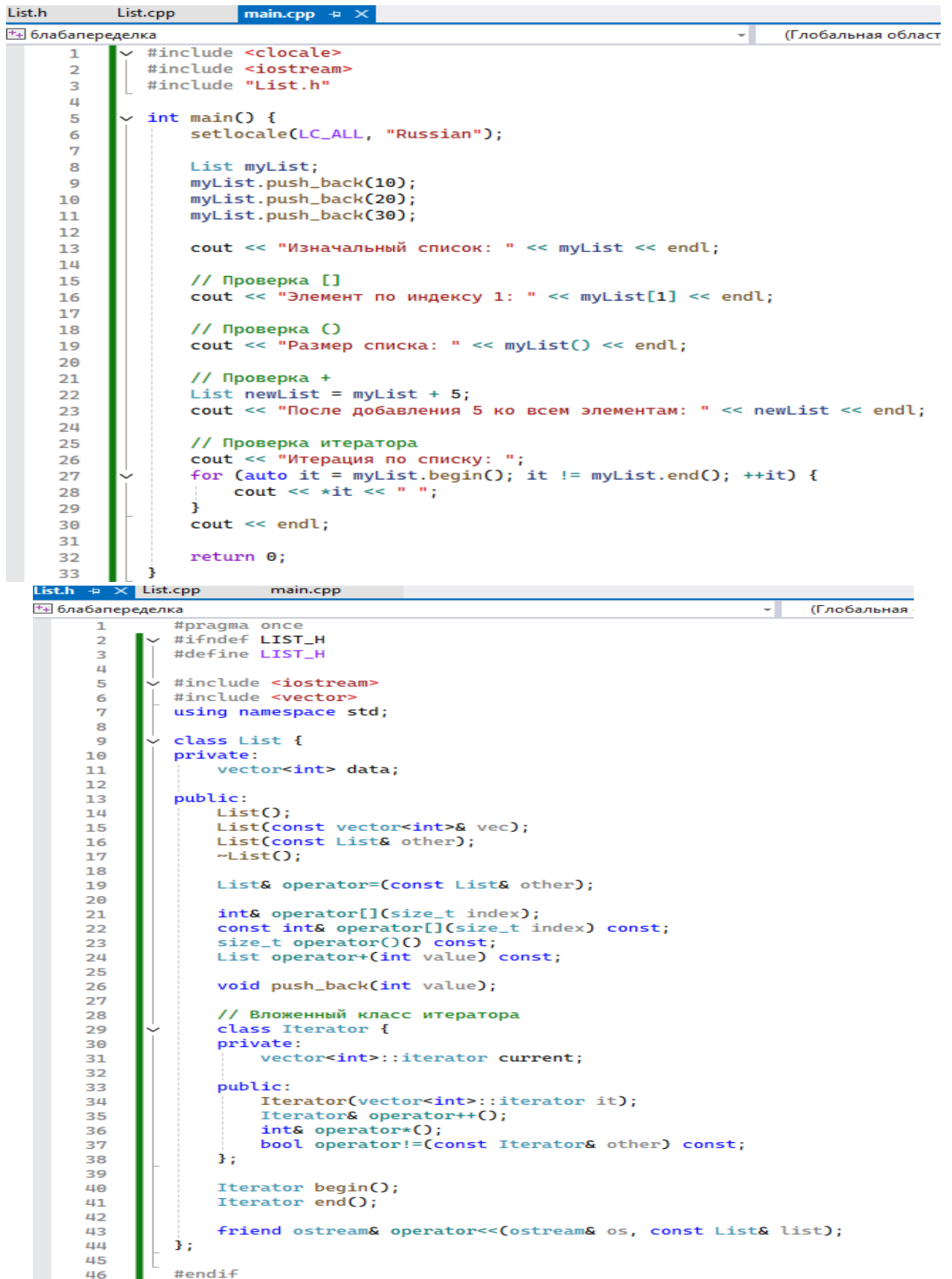
`[]` – доступа по индексу;

`()` – определение размера вектора;

`+` число – добавляет константу ко всем элементам вектора;

`++` - переход к следующему элементу (с помощью класса-итератора).

2 Код на C++



```
1 #include <locale>
2 #include <iostream>
3 #include "List.h"
4
5 int main() {
6     setlocale(LC_ALL, "Russian");
7
8     List myList;
9     myList.push_back(10);
10    myList.push_back(20);
11    myList.push_back(30);
12
13    cout << "Изначальный список: " << myList << endl;
14
15    // Проверка []
16    cout << "Элемент по индексу 1: " << myList[1] << endl;
17
18    // Проверка ()
19    cout << "Размер списка: " << myList() << endl;
20
21    // Проверка +
22    List newList = myList + 5;
23    cout << "После добавления 5 ко всем элементам: " << newList << endl;
24
25    // Проверка итератора
26    cout << "Итерация по списку: ";
27    for (auto it = myList.begin(); it != myList.end(); ++it) {
28        cout << *it << " ";
29    }
30    cout << endl;
31
32    return 0;
33 }
```

```
1 #pragma once
2 #ifndef LIST_H
3 #define LIST_H
4
5 #include <iostream>
6 #include <vector>
7 using namespace std;
8
9 class List {
10 private:
11     vector<int> data;
12
13 public:
14     List();
15     List(const vector<int>& vec);
16     List(const List& other);
17     ~List();
18
19     List& operator=(const List& other);
20
21     int& operator[](size_t index);
22     const int& operator[](size_t index) const;
23     size_t operator()() const;
24     List operator+(int value) const;
25
26     void push_back(int value);
27
28     // Вложенный класс итератора
29     class Iterator {
30 private:
31         vector<int>::iterator current;
32
33 public:
34         Iterator(vector<int>::iterator it);
35         Iterator& operator++();
36         int& operator*();
37         bool operator!=(const Iterator& other) const;
38     };
39
40     Iterator begin();
41     Iterator end();
42
43     friend ostream& operator<<(ostream& os, const List& list);
44 };
45
46 #endif
```

```

List.h  List.cpp  main.cpp
благонепредделка (Глобальная область)

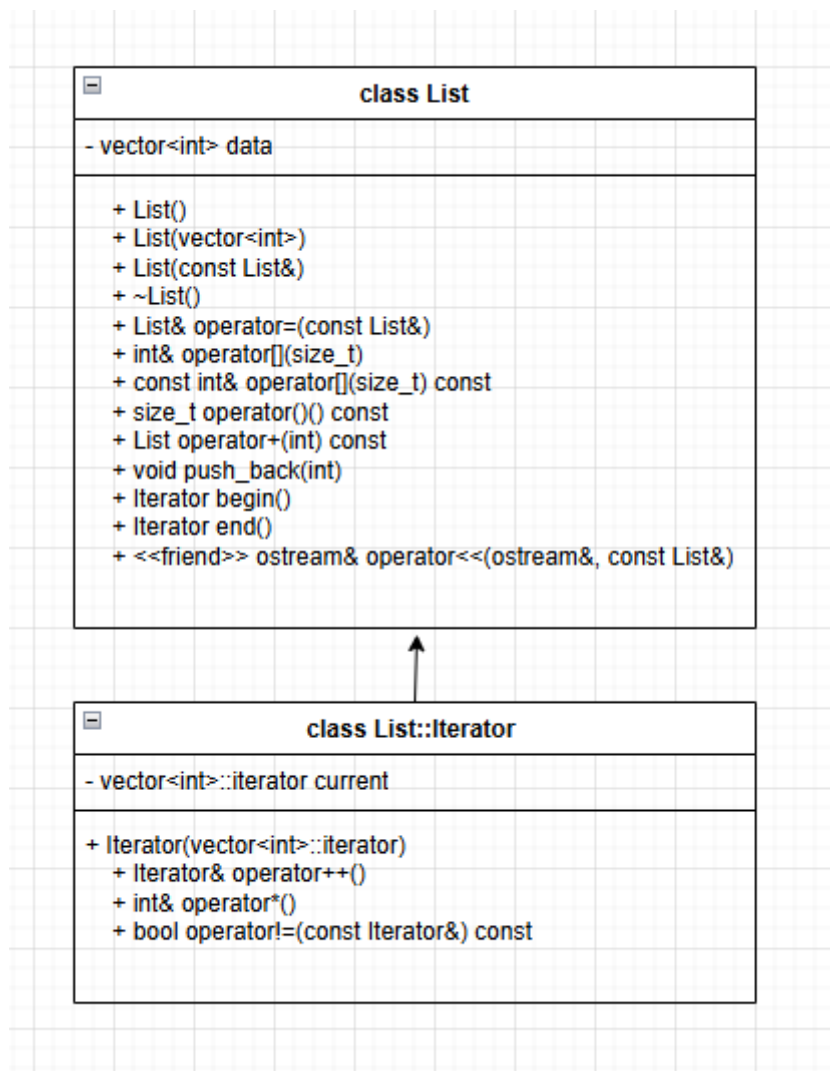
1  #include "List.h"
2
3  List::List() = default;
4
5  List::List(const vector<int>& vec) : data(vec) {}
6
7  List::List(const List& other) : data(other.data) {}
8
9  List::~List() = default;
10
11  List& List::operator=(const List& other) {
12  |   if (this != &other) {
13  |       data = other.data;
14  |   }
15  |   return *this;
16  | }
17
18  int& List::operator[](size_t index) {
19  |   return data.at(index);
20  | }
21
22  const int& List::operator[](size_t index) const {
23  |   return data.at(index);
24  | }
25
26  size_t List::operator()() const {
27  |   return data.size();
28  | }
29
30  List List::operator+(int value) const {
31  |   List result(*this);
32  |   for (auto& el : result.data) {
33  |       el += value;
34  |   }
35  |   return result;
36  | }
37
38  void List::push_back(int value) {
39  |   data.push_back(value);
40  | }
41
42  // Реализация итератора
43  List::Iterator::Iterator(vector<int>::iterator it) : current(it) {}
44
45  List::Iterator& List::Iterator::operator++() {
46  |   ++current;
47  |   return *this;
48  | }
49
50  int& List::Iterator::operator*() {
51  |   return *current;
52  | }
53
54  bool List::Iterator::operator!=(const Iterator& other) const {
55  |   return current != other.current;
56  | }
57
58  List::Iterator List::begin() {
59  |   return Iterator(data.begin());
60  | }
61
62  List::Iterator List::end() {
63  |   return Iterator(data.end());
64  | }
65
66  ostream& operator<<(ostream& os, const List& list) {
67  |   for (const auto& el : list.data) {
68  |       os << el << " ";
69  |   }
70  |   return os;
71  | }

```

3 Результаты работы

```
Изначальный список: 10 20 30
Элемент по индексу 1: 20
Размер списка: 3
После добавления 5 ко всем элементам: 15 25 35
Итерация по списку: 10 20 30
```

4 UML-диаграмма классов



5 Ответы на контрольные вопросы

1. Абстрактный тип данных — модель данных с операциями без реализации. Примеры: стек, очередь, список.
2. Пример параметризации — шаблон `template<typename T> class Vector`.
3. Пример спецификации — абстрактный класс с чисто виртуальной функцией.
4. Контейнер — структура данных для хранения элементов. Примеры: массив, вектор, список.
5. Операции: вставка, удаление, поиск, доступ, изменение
6. Виды доступа: по индексу (`vec[i]`), по итератору, по ключу (`map[key]`).
7. Итератор — объект для последовательного доступа к элементам.
8. Итератор реализуется как класс с перегрузкой `*`, `++`, `!=`.
9. Объединение контейнеров: через цикл, `+`, `insert`, `merge`.
10. Доступ по ключу (`map[key]`).
11. Стек.
12. `d. int mas[100];`
13. `d. int mas;`
14. Произвольный доступ (random access).
15. Последовательный доступ.

6 Ссылка на github

ссылка на github - <https://github.com/MAKSPOWERO/mas1>