

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**Пермский национальный исследовательский
политехнический университет**

Факультет электротехнический

Кафедра ИТАС

ОТЧЁТ

о лабораторной работе №6 по классам

Выполнил:
Студент группы ИВТ-23-1Б
Пискунов Д. А.

Проверил:
Доцент кафедры ИТАС
Яруллин Д.В.

Пермь 2024

Задача:

15

Класс- контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] – доступа по индексу;

int() – определение размера списка;

* вектор – умножение элементов списков $a[i]*b[i]$;

+n - переход вправо к элементу с номером n (с помощью класса-итератора).

Текст программы
файл Lab_Main_6.cpp

```
1  #include <iostream>
2  #include "List.h"
3
4  using namespace std;
5
6  int main() {
7      system("chcp 1251 > NULL");
8
9      List a(5);
10     cout << "Введите 5 элементов:\n";
11     cin >> a;
12     cout << "Список a:\n";
13     cout << a << '\n';
14     cout << "меняем 3 элемент\n";
15     a[2] = 100;
16     cout << "Список a:\n";
17     cout << a << '\n';
18
19     List b(10);
20     cout << "Список b:\n";
21     cout << b << '\n';
22     cout << "Умножение элементов списков";
23     cout << " a[2] * b[3] = " << a[2] * b[3] << endl;
24     a = b;
25     cout << "a = b\n";
26     cout << "Размер списка a = " << a() << endl;
27     cout << "Список a:\n";
28     cout << a << '\n';
29
30
31     cout << "Выведем список b с помощью итератора:\n";
32     for (Iterator i = b.first(); *i != NULL; ++i) {
33         cout << (*i)->key << ' ';
34     }
35     cout << '\n';
36     cout << "Добавим 3 элемента в список b\n";
37     b << 14;
38     b << 22;
39     b << 13;
40     cout << "Список b:\n";
41     cout << b << '\n';
42     cout << "Проверим +\n";
43     cout << "Ставим итератор на первый элемент b\n";
44     Iterator i = b.first();
45     cout << "Смещаем на 11 элементов\n";
46     i + 11;
47     cout << "Текущий элемент: " << (*i)->key << '\n';
48
49     return 0;
50 }
```

Файл List.cpp

```
1  #include "List.h"
2
3  List::List(int count) {
4      n = count;
5      head = new node;
6      head->key = 0;
7      lastnd = head;
8      for (int i = 1; i < n; i++) {
9          cur = new node;
10         cur->key = 0;
11         lastnd->next = cur;
12         lastnd = cur;
13     }
14     lastnd->next = NULL;
15 }
16
17 List::~List() {
18     lastnd = head;
19     while (lastnd != NULL) {
20         cur = lastnd->next;
21         delete lastnd;
22         lastnd = cur;
23     }
24     n = 0;
25 }
26
27 List& List::operator=(List& l) {
28     if (this != &l) {
29         if (this != 0) {
30             lastnd = head;
31             while (lastnd != NULL) {
32                 cur = lastnd->next;
33                 delete lastnd;
34                 lastnd = cur;
35             }
36             n = 0;
37         }
38         lastnd = head = new node;
39         l.lastnd = l.head->next;
40         lastnd->key = l.head->key;
41         while (l.lastnd != NULL) {
42             lastnd->next = new node;
43             lastnd->next->key = l.lastnd->key;
44             l.lastnd = l.lastnd->next;
45             lastnd = lastnd->next;
```

```

45         lastnd = lastnd->next;
46     }
47     lastnd->next = NULL;
48     n = l.n;
49 }
50 return *this;
51 }
52
53 int& List::operator[](int index) {
54     if (index < n) {
55         lastnd = head;
56         for (int i = 0; i < index; i++) {
57             lastnd = lastnd->next;
58         }
59         return lastnd->key;
60     }
61     else {
62         cout << "Çàïðáááäëüíûé èíáâëñ";
63     }
64 }
65
66 int List::operator () () {
67     return n;
68 }
69
70 Iterator List::first() {
71     beg.cur = head;
72     return beg;
73 }
74
75 Iterator List::last() {
76     lastnd = head;
77     while (lastnd->next != NULL) {
78         lastnd = lastnd->next;
79     }
80     end.cur = lastnd;
81     return end;
82 }
83
84 ostream& operator<<(ostream& out, List& l) {
85     if (l.n) {
86         l.lastnd = l.head;
87         while (l.lastnd != NULL) {
88             out << l.lastnd->key << ' ';

```

```

88             out << l.lastnd->key << ' ';
89             l.lastnd = l.lastnd->next;
90         }
91     }
92     else {
93         out << "İöñöî";
94     }
95     return out;
96 }
97
98 istream& operator>>(istream& in, List& l) {
99     l.lastnd = l.head;
100    while (l.lastnd != NULL) {
101        in >> l.lastnd->key;
102        l.lastnd = l.lastnd->next;
103    }
104    return in;
105 }
106
107 void List::operator << (int number) {
108     if (head == NULL) {
109         head = new node;
110         head->key = number;
111         head->next = NULL;
112     }
113     else {
114         lastnd = head;
115         while (lastnd->next != NULL) {
116             lastnd = lastnd->next;
117         }
118         lastnd->next = new node;
119         lastnd->next->key = number;
120         lastnd = lastnd->next;
121         lastnd->next = NULL;
122     }
123     n += 1;
124 }

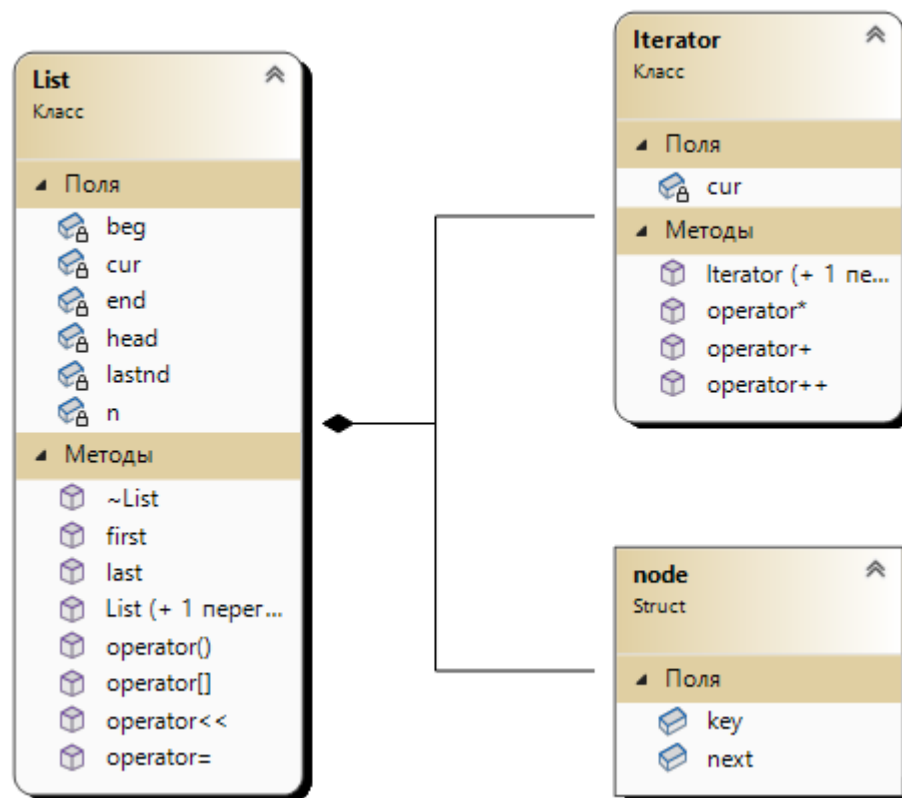
```

Файл List.h

```
1  #pragma once
2
3  #include<iostream>
4  #include<string>
5
6  using namespace std;
7
8  struct node {
9      int key;
10     node* next = nullptr;
11 };
12
13 class Iterator {
14 private:
15     node* cur;
16     friend class List;
17 public:
18     Iterator() {
19         cur = nullptr;
20     };
21     Iterator(node* node) {
22         cur = node;
23     };
24     void operator ++ () {
25         cur = cur->next;
26     }
27     node* operator *() const {
28         return cur;
29     }
30     void operator + (int shift) {
31         node* tmp = cur;
32         int i = 0;
33         while (i < shift && cur != nullptr && cur->next != nullptr) {
34             cur = cur->next;
35             i++;
36         }
37         if (i < shift) {
38             cout << "\n";
39             cur = tmp;
40         }
41     }
42 };
43
44 class List {
45 private:
46     node* lastnd, * cur, * head;
47     int n = 0;
48     Iterator beg, end;
```

```
48         Iterator beg, end;
49     public:
50         List() {};
51         List(int);
52         ~List();
53         List& operator=(List&);
54         int& operator[](int);
55         int operator () ();
56         friend ostream& operator << (ostream&, List&);
57         friend istream& operator >> (istream&, List&);
58         Iterator first();
59         Iterator last();
60         void operator << (int);
61     };
```


UML-диаграмма



Тест

```
Введите 5 элементов:
12
151
65
78
45
Список a:
12 151 65 78 45
меняем 3 элемент
Список a:
12 151 100 78 45
Список b:
0 0 0 0 0 0 0 0 0
Умножение элементов списков a[2] * b[3] = 0
a = b
Размер списка a = 10
Список a:
0 0 0 0 0 0 0 0 0
Выведем список b с помощью итератора:
0 0 0 0 0 0 0 0 0
Добавим 3 элемента в список b
Список b:
0 0 0 0 0 0 0 0 0 14 22 13
Проверим +
Ставим итератор на первый элемент b
Смещаем на 11 элементов
Текущий элемент: 22

C:\Users\M0kASiH\Desktop\Test\x64\Debug\Test.exe (процесс 5280) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Ответы на вопросы

1. Что такое абстрактный тип данных? Привести примеры АТД.

Ответ: АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов. Примером абстрактного типа данных является класс в языке C++.

Пример абстрактного типа данных - класс Итератор

2. Привести примеры абстракции через параметризацию.

```
template <class T>
class List {
public:
    List() {};
    List(int count) { ... }
    ~List() { ... }
    List& operator = (List<T>& l) { ... }
    int& { ... }
    int operator () () { ... }
    friend ostream& operator << (ostream& out, List<T>& l) { ... }
    friend istream& operator >> (istream& in, List<T>& l) { ... }
    void operator << (T t) { ... }
private:
    Node<T>* lastNd, * current, * head;
    int n = 0;
};
```

В этом примере класс «List» параметризован типом «Т», который определяется при создании объекта класса. Это позволяет использовать один и тот же класс для работы с различными типами данных.

3. Привести примеры абстракции через спецификацию.

```
class Shape
{
public:
    virtual double area() const = 0;
};

class Circle : public Shape {
private:
    int x, y;
public:
    double area();
};

class Triangle : public Shape {
private:
    int x1, y1, x2, y2, x3, y3;
public:
    int area();
};
```

Класс Shape содержит чисто виртуальный метод area().

Класс Triangle является производным от класса Shape и реализует метод area(). Он содержит приватные поля x1, y1, x2, y2, x3, y3, которые используются для вычисления площади.

Класс Circle тоже является производным от класса Shape и реализует метод area(). Он содержит приватные поля x, y, которое используется для вычисления площади.

Оба класса «Rectangle» и «Circle» реализуют интерфейс, определенный в абстрактном классе «Shape», что позволяет использовать полиморфизм для работы с различными типами геометрических фигур.

4. Что такое контейнер? Привести примеры.

Ответ: Контейнер – набор однотипных элементов.

```
#include <iostream>
using namespace std;
#include <list>

int main() {
    list<int> a = { 1, 3, 4 };
    return 0;
}
```

5. Какие группы операций выделяют в контейнерах?

Ответ:

1. Операции доступа к элементам
2. Операции добавления и удаления
3. Операции поиска
4. Операции объединения контейнеров

6. Какие виды доступа к элементам контейнера существуют? Привести примеры.

Ответ: Доступ к элементам контейнера бывает: последовательный, прямой и ассоциативный.

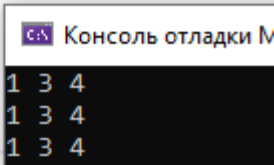
```

#include <iostream>
using namespace std;
#include <list>
#include <vector>
#include <map>

int main() {
    list<int> a = { 1, 3, 4 };
    vector<int> b = { 1, 3, 4 };
    map<char, int> c = { {'a', 1}, {'b', 3}, {'c', 4} };

    for (auto i = a.begin(); i != a.end(); i++) { //последовательный доступ
        cout << *i << ' ';
    }
    cout << '\n';
    for (int i = 0; i < b.size(); i++) { //прямой доступ
        cout << b[i] << ' ';
    }
    cout << '\n';
    for (char i : (string)"abc") { //ассоциативный доступ
        cout << c[i] << ' ';
    }
    return 0;
}

```



Консоль отладки IV

```

1 3 4
1 3 4
1 3 4

```

7. Что такое итератор?

Ответ: Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера. Итератор может быть реализован как часть класса-контейнера в виде набора методов.

8. Каким образом может быть реализован итератор?

Ответ: Как класс

9. Каким образом можно организовать объединение контейнеров?

Ответ:

1. Сложение множеств
2. Пересечение множеств
3. Вычитание множеств

10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?

Ответ: По ключу

11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?

Ответ: Стек

12. Какой из объектов (a,b,c,d) является контейнером?

- a. `int mas=10;`
- b. `2. int mas;`
- c. `3. struct {char name[30]; int age;} mas;`
- d. `4. int mas[100];`

Ответ: d

13. Какой из объектов (a,b,c,d) не является контейнером?

- a. `int a[]={ 1,2,3,4,5};`
- b. `int mas[30];`
- c. `struct {char name[30]; int age;} mas[30];`
- d. `int mas;`

Ответ: d

14. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?

Ответ: Прямой доступ.

15. Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?

Ответ: Последовательный доступ.