Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

АДТ. КОНТЕЙНЕРЫ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**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.

Для варианта 15:

Класс – контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] – доступ про индексу;

int() – определение размера списка;

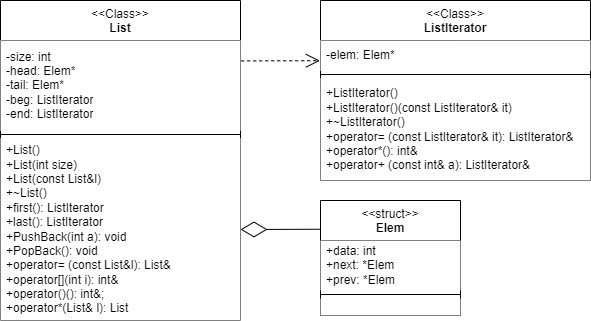
\* вектор – умножение элементов списков a[i] \* b[i];

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

**Анализ задачи:**

1. Создается пара файлов (.h и .cpp), где определяется класс List и его методы.
2. Контейнер работает по принципу списка: каждый элемент указывает на следующий и предыдущий.
3. Класс-итератор определяется в файлах класса List.
4. При умножении двух списков выполняется проверка на соответствие размеров списков.

**UML**



**Код**

List.h

#pragma once

const int MAX\_SIZE = 25;

struct Elem {

int data = 0;

Elem\* next = nullptr;

Elem\* prev = nullptr;

};

class ListIterator

{

friend class List;

Elem\* elem;

public:

ListIterator();

ListIterator(const ListIterator& it);

ListIterator& operator=(const ListIterator& a);

int& operator\*() const;

ListIterator& operator+ (const int& a);

};

class List

{

int size;

Elem\* head = nullptr;

Elem\* tail = nullptr;

ListIterator beg;

ListIterator end;

public:

List() { size = 0; head = nullptr; Elem\* tail = nullptr;};

List(int s);

List(const List& l);

~List();

ListIterator first();

ListIterator last();

void PushBack(int a);

void PopBack();

List& operator= (const List& l);

int& operator[] (int i);

int& operator()();

List operator\*(List& l);

};

List.cpp

#include "List.h"

#include <iostream>

using namespace std;

List::List(int s)

{

size = s;

Elem\* new\_elem = new Elem;

head = new\_elem;

tail = new\_elem;

for (int i = 0; i < size - 1; i++)

{

Elem\* new\_elem = new Elem;

tail->next = new\_elem;

new\_elem->prev = tail;

tail = new\_elem;

}

beg.elem = head;

end.elem = tail;

}

List::List(const List& l)

{

Elem\* elem = l.head;

while (elem != nullptr) {

PushBack(elem->data);

elem = elem->next;

}

beg.elem = head;

end.elem = tail;

}

List::~List()

{

Elem\* curr = head;

while (curr != nullptr)

{

head = curr->next;

delete curr;

curr = head;

}

}

List& List::operator= (const List& l)

{

if (this == &l)return \*this;

if (head != nullptr)

{

Elem\* curr = head;

while (curr != nullptr)

{

head = curr->next;

delete curr;

curr = head;

}

}

Elem\* elem = l.head;

while (elem != nullptr) {

PushBack(elem->data);

elem = elem->next;

}

beg.elem = head;

end.elem = tail;

return \*this;

}

int& List::operator[] (int i)

{

Elem\* curr = head;

for (int j = 0; j < i; j++) {

curr = curr->next;

}

return curr->data;

}

List List::operator\*(List& l)

{

List base = \*this;

int s = this->size;

List temp;

for (int i = 0; i < s; i++)

{

temp.PushBack(base[i] \* l[i]);

}

return temp;

}

int& List::operator()() {

return size;

}

ListIterator List::first()

{

return beg;

}

ListIterator List::last()

{

return end;

}

void List::PushBack(int a)

{

Elem\* new\_elem = new Elem;

new\_elem->data = a;

if (size == 0)

{

head = new\_elem;

tail = new\_elem;

size++;

beg.elem = head;

end.elem = tail;

}

else

{

tail->next = new\_elem;

new\_elem->prev = tail;

tail = new\_elem;

size++;

end.elem = tail;

}

}

void List::PopBack()

{

Elem\* curr = tail;

tail = curr->prev;

delete curr;

tail->next = nullptr;

size--;

end.elem = tail;

}

ListIterator::ListIterator() {

elem = nullptr;

}

ListIterator::ListIterator(const ListIterator& it) {

elem = it.elem;

}

ListIterator& ListIterator::operator=(const ListIterator& a)

{

elem = a.elem;

return \*this;

}

int& ListIterator::operator\*() const

{

return elem->data;

}

ListIterator& ListIterator::operator+ (const int& a)

{

for (int i = 0; i < a; ++i) {

elem = elem->next;

}

return \*this;

}

Main.cpp

#include "List.h"

#include <iostream>

using namespace std;

int main()

{

int count, in\_value, index;

List list1;

ListIterator iter;

cout << "Enter number of elements: ";

cin >> count;

cout << endl;

cout << "Enter elements of list:" << endl;

for (int i = 0; i < count; ++i)

{

cin >> in\_value;

list1.PushBack(in\_value);

}

cout << endl;

iter = list1.first();

cout << "Enter index of element in list to call: ";

cin >> index;

cout << list1[index] << endl;

cout << endl;

cout << "Enter number of elements to move for from first: ";

cin >> index;

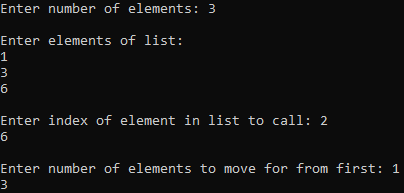
iter + index;

cout << \*iter << endl;

cout << endl;

}

**Решение**



**Выводы**

В ходе выполнения работы были изучены АДТ, контейнеры и принципы работы с ними.

**Ответы на контрольные вопросы:**

1. Тип данных, определяемый только через операции.
2. Абстракция через параметризацию

class Pair {

public:

TT\* first;

TT\* second;

void Show(TT\*, TT\*);

}

1. Абстракция через спецификацию

class Pair {

public:

TT\* first;

TT\* second;

void Show(TT\*, TT\*);

}

1. Набор однотипных элементов. Массивы, списки, векторы.
2. Доступ к элементам, добавление-удаление, поиск, объединение контейнеров, специальные операции.
3. Последовательный (доступ к элементам списка), прямой (доступ по индексу в массиве), ассоциативный (доступ по ключу в словаре).
4. Объект, предоставляющий последовательный доступ к элементам.
5. Может быть реализован как класс.
6. Получая новый контейнер из двух объединяемых.
7. Ассоциативный.
8. Стек.
9. d
10. d
11. Прямой.
12. Последовательный.

**Github**

<https://github.com/Hitikov/Lab_OOP_6>