Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

**ОТЧЁТ**

Тема: «АТД. Контейнеры»

Выполнил

Студент РИС-22-2б

Зубов Р.А.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

**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.

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

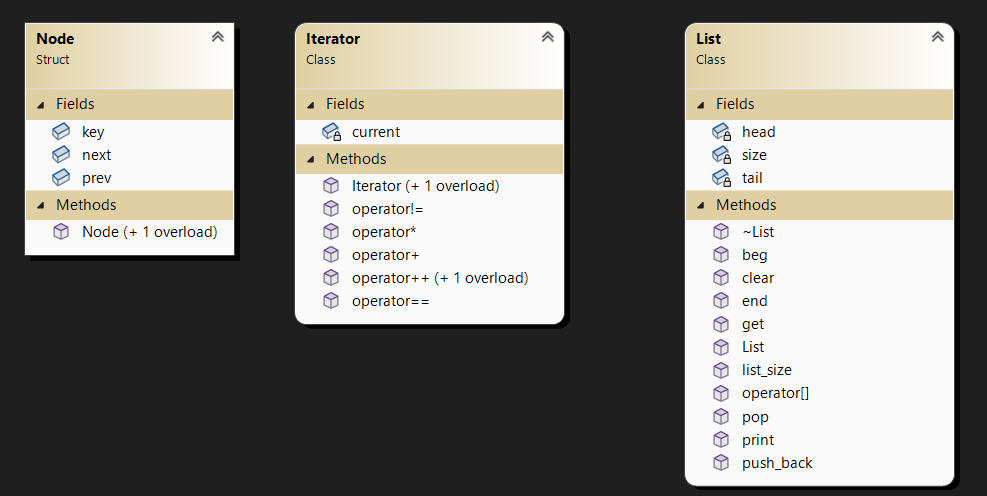
[] – доступа по индексу;

int() – определение размера списка;

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

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

**UML**

****

**Код программы**

**List.h:** #pragma once

#include <cstddef>

#include <iostream>

struct Node {

int key;

Node\* next = NULL;

Node\* prev = NULL;

Node();

Node(int k, Node\* n, Node\* p);

};

class Iterator {

friend class List;

private:

Node\* current = NULL;

public:

Iterator() { current = NULL; }

Iterator(Node\* node) : current(node) {}

bool operator==(const Iterator& other) const { //сравнивают текущий узел

return current == other.current; // и узел другого итератора

} //

bool operator!=(const Iterator& other) const { // на равенство и неравенство.

return current != other.current;

}

int& operator\*() const { // операция разыменования итератора

return current->key;

}

Iterator& operator++() { //++i

current = current->next;

return \*this;

}

Iterator operator++(int) { // i++

Iterator old = \*this;

current = current->next;

return old;

}

Iterator operator+(int n) const { // переход вправо к n элементу

Iterator it(\*this);

while (n > 0 && it.current != NULL) {

it.current = it.current->next;

n--;

}

return it;

}

};

class List {

Node\* head;

Node\* tail;

int size;

public:

List() {

head = NULL;

tail = NULL;

size = 0;

}

~List() {

clear();

}

int& get(int index);

int& operator[](int index) {

return get(index); // возвращаем ссылку на элемент по индексу

}

// Метод добавления элемента в конец списка

void push\_back(int k);

void pop(int key);

int list\_size() {

return size;

}

void clear();

void print() const;

friend std::ostream& operator<<(std::ostream& out, const List& list);

friend std::istream& operator>>(std::istream& in, List& list);

Iterator beg() {

return Iterator(head); // возвращаем итератор на первый элемент

}

Iterator end() {

return Iterator(tail); // возвращаем итератор на последний элемент

}

friend List operator\*(List& left, List& right) {

List result; // создаем новый список

Iterator it1 = left.beg(); // итератор по первому списку

Iterator it2 = right.beg(); // итератор по второму списку

while (it1 != left.end() && it2 != right.end()) {

result.push\_back(\*it1 \* \*it2); // добавляем произведение элементов в новый список

++it1; // переходим к следующему элементу первого списка

++it2; // переходим к следующему элементу второго списка

}

return result; // возвращаем новый список

}

};

**List.cpp:**

#include "List.h"

#include "List.h"

Node::Node(int k, Node\* n, Node\* p)

{

key = k;

next = n;

prev = p;

}

int& List::get(int index)

{

Node\* current = head; // начинаем с головы списка

int i = 0;

while (i < index && current->next != NULL) {

current = current->next; // переходим к следующему узлу

}

return current->key; // возвращаем ссылку на данные узла

}

void List::push\_back(int k)

{

Node\* newNode = new Node(k, NULL, tail); // создаем новый узел с ключом key, указателем на следующий узел равным NULL и указателем на предыдущий узел равным tail

if (tail != NULL) {

tail->next = newNode; // если tail не равен NULL, то устанавливаем указатель на следующий узел у tail на новый узел

}

tail = newNode; // устанавливаем tail на новый узел

if (head == NULL) {

head = newNode; // если список был пустой, то устанавливаем head на новый узел

}

size++;

}

void List::pop(int key) {

Node\* p = head;

while (p != NULL && p->key != key) { // Проходим по списку до нужного ключа

p = p->next;

}

if (p != NULL) { // Если нашли узел по позиции

if (p->prev != NULL) { // Если узел не является головным

p->prev->next = p->next;

}

else { // Если узел является головным

head = p->next;

}

if (p->next != NULL) { // Если узел не является хвостовым

p->next->prev = p->prev;

}

else { // Если узел является хвостовым

tail = p->prev;

}

delete p; // Удаляем текущий узел из памяти

size--;

}

}

void List::clear()

{

Node\* p = head;

while (p != NULL) {

Node\* q = p;

p = p->next;

delete q; // Удаляем текущий узел из памяти

}

head = NULL;

tail = NULL;

size = 0;

}

void List::print() const

{

Node\* node = head;

while (node != NULL) {

std::cout << node->key << "->";

node = node->next;

}

std::cout << std::endl;

}

std::ostream& operator<<(std::ostream& out, const List& list)

{

list.print();

return out;

}

std::istream& operator>> (std::istream& in, List& list)

{

int k;

in >> k;

list.push\_back(k);

return in;

}

**Source.cpp:**

#include "List.h"

#include <iostream>

using namespace std;

int main()

{

List l;

cout << "Made a empty list: " << l;

cout <<"List size: " << l.list\_size() << "\n";

cout << "Add 5 to list: ";

l.push\_back(5);

cout << l;

cout << "in index 0 elem is: " << l[0];

cout << "\nList size: " << l.list\_size();

cout << "\nAdding two elem in list: ";

l.push\_back(34);

l.push\_back(55);

cout << l;

cout << "List size: " << l.list\_size();

cout << "\nFirst elem in list: " << \*(l.beg());

cout << "\nLast elem in list: " << \*(l.end());

cout << "\nDelete 5 from list: ";

l.pop(5);

cout << l;

List l1;

l1.push\_back(5); cout << "Made a new list l1: " << l;

l[2] = l[1] \* l1[0];

cout << "List l After mult: " << l;

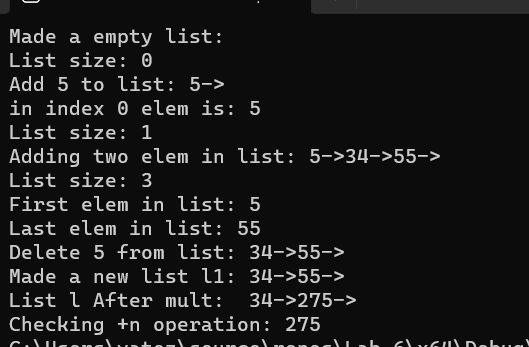
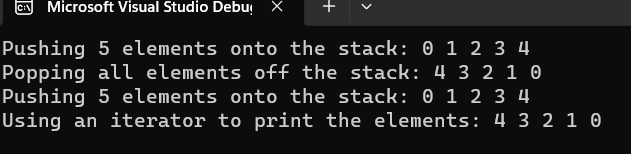
Iterator i = l.beg();

i = i + 1;

cout << "Checking +n operation: " << \*i;

return 0;

}

**Результаты программ** 

**Контрольные вопросы**

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

Абстрактный тип данных (АТД) - это тип данных, определенный не через его внутреннюю реализацию, а через набор операций, которые можно выполнять над этим типом данных. Примеры АТД: стек, очередь, список, дерево и т.д.

1. Привести примеры абстракции через параметризацию.

template<typename T>

void swap(T& a, T& b) {

T temp = a;

a = b;

b = temp;

}

int main() {

int a = 10, b = 20;

swap<int>(a, b);

return 0;

}

1. Привести примеры абстракции через спецификацию.

class Animal {

private:

string name;

int age;

public:

Animal(string n, int a) {

name = n;

age = a;

}

virtual void speak() = 0; // абстрактный метод

};

class Dog : public Animal {

public:

Dog(string n, int a) : Animal(n, a) {}

void speak() {

cout << "Гав-гав!" << endl;

}

};

class Cat : public Animal {

public:

Cat(string n, int a) : Animal(n, a) {}

void speak() {

cout << "Мяу-мяу!" << endl;

}

};

4. Что такое контейнер? Привести примеры.

Контейнер - это объект, который содержит другие объекты и предоставляет методы для добавления, удаления и доступа к этим объектам. Примеры: массив, список, стек, очередь, хэш-таблица и т.д.

5. Какие группы операций выделяют в контейнерах?

Группы операций в контейнерах: операции добавления и удаления элементов, операции доступа к элементам, операции поиска элементов, операции сортировки и т.д.

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

Доступ по индексу(index access) - это доступ к элементу контейнера по индексу, например, vector<int> myVec{ 1, 2, 3 }; int x = myVec[1];

Доступ по итератору(iterator access) - это доступ к элементу контейнера с использованием итератора, который представляет указатель на элемент контейнера, например, vector<int> myVec{ 1, 2, 3 }; vector<int>::iterator it = myVec.begin(); int x = \*it;

Доступ с использованием алгоритмов(algorithmic access) - это доступ к элементам контейнера с использованием функций стандартной библиотеки, таких как find, accumulate, for\_each и т.д., например, vector<int> myVec{ 1, 2, 3 }; auto it = find(myVec.begin(), myVec.end(), 2); int x = \*it;

7. Что такое итератор?

Итератор - это объект, который позволяет перебирать элементы контейнера последовательно, без доступа к его внутренней реализации.

8. Каким образом может быть реализован итератор?

Итератор может быть реализован как указатель на элемент контейнера, либо как отдельный объект, который хранит информацию о текущей позиции в контейнере.

9. Каким образом можно организовать объединение контейнеров?

Объединение контейнеров можно организовать с помощью операций объединения, пересечения и разности множеств.

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

Контейнер, состоящий из элементов "ключ-значение" (например, ассоциативный массив или словарь), предоставляет доступ к элементам по ключу.

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

Контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера, называется стеком.

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];**

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

a. int a[]={1,2,3,4,5};

**b. 2. int mas[30];**

c. 3. struct {char name[30]; int age;} mas[30];

d. 4. int mas;

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

Доступ к элементам контейнера будет осуществляться по индексу (для массива).

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

Доступ к элементам контейнера будет осуществляться с помощью итератора или указателя на текущий элемент списка.