Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Классы и объекты. АТД. Контейнеры»

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Желнин Н.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

Постановка задачи

Определить класс - контейнер, конструкторы, операции ввода, вывода, присваивания.

Класс – контейнер СПИСОК.

Операции:

Доступ по индексу []

Определение размера списка ()

Умножение всех элементов на число \*

Переход влево к элементу в номером n -n

Алгоритм решения

1. Создаем класс – контейнер СПИСОК.
2. Создаем методы инициализации, ввода, вывода и т.д.
3. Создаем класс – итератор для проходки по списку.
4. В главной функции последовательно проверяем работоспособность всех функции.

Код программы

*MAIN*

*#include "list\_head.h"*

*#include <iostream>*

*using namespace std;*

*int main() {*

*srand(time(NULL));*

*List a(10);*

*cout << a << endl;*

*for (int i = 0; i < a(); i++) a[i] = rand() % 100 - 40;*

*cout << a << endl;*

*List b(a);*

*cout << "List have been copied\n";*

*cout << b << endl;*

*cout << a\*b << endl;*

*int num;*

*cout << "Enter int number: "; cin >> num;*

*b.pushback(num);*

*b.popfront();*

*for (Iterator iter = b.first(); iter != b.last(); ++iter)*

*cout << \*iter << " ";*

*return 0;*

*}*

*LIST.H*

*#pragma once*

*#include <iostream>*

*using namespace std;*

*struct Node{*

*Node\* prev = nullptr;*

*Node\* next = nullptr;*

*int data = 0;*

*};*

*class Iterator{*

*friend class List;*

*Node\* n;*

*public:*

*Iterator(){*

*n = nullptr;*

*}*

*Iterator(const Iterator& it){*

*n = it.n;*

*}*

*Iterator& operator=(const Iterator& a) {*

*n = a.n;*

*return \*this;*

*}*

*bool operator==(const Iterator& it) { return n == it.n; }*

*bool operator!=(const Iterator& it) { return n != it.n; };*

*Iterator& operator++() {*

*n = n->next;*

*return \*this;*

*};*

*Iterator& operator--() {*

*n = n->prev;*

*return \*this;*

*}*

*Iterator& operator+(const int& a) {*

*for (int i = 0; i < a; i++) n = n->next;*

*return \*this;*

*}*

*Iterator& operator-(const int& a) {*

*for (int i = 0; i < a; i++) n = n->prev;*

*return \*this;*

*}*

*int& operator \*() const { return n->data; }*

*};*

*class List {*

*private:*

*int size;*

*Node\* head = nullptr;*

*Node\* tail = nullptr;*

*Iterator start, end;*

*public:*

*List();*

*List(int s, int k = 0);*

*List(const List&);*

*~List();*

*int front();*

*int back();*

*void pushback(int data);*

*void pushfront(int data);*

*void popback();*

*void popfront();*

*bool empty();*

*List& operator=(const List&);*

*int& operator[](int index);*

*int& operator()();*

*List operator\*(List&);*

*friend ostream& operator <<(ostream&, const List&);*

*friend istream& operator >>(istream&, const List&);*

*Iterator first() { return start; }*

*Iterator last() { return end; }*

*};*

*LIST.CPP*

#include "list\_head.h"

#include <iostream>

using namespace std;

List::List(int s, int k) {

size = s;

Node\* node = new Node;

node->data = k;

head = node;

tail = node;

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

Node\* node = new Node;

node->data = rand() % 10;

tail->next = node;

node->prev = tail;

tail = node;

}

tail->next = nullptr;

start.n = head;

end.n = tail->next;

}

List::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

List::List(const List& a) {

Node\* node = a.head;

while (node != nullptr) {

pushback(node->data);

node = node->next;

}

start = a.start;

end = a.end;

}

List::~List() {

Node\* th = head;

while (th != nullptr) {

head = th->next;

delete th;

th = head;

}

}

void List::pushback(int data) {

Node\* newNode = new Node;

newNode->data = data;

if (head == nullptr) {

head = newNode;

tail = newNode;

size++;

end.n = tail->next;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

size++;

end.n = tail->next;

}

}

void List::pushfront(int data) {

Node\* newNode = new Node;

newNode->data = data;

if (head == nullptr) {

head = newNode;

tail = newNode;

size++;

start.n = head;

}

else {

head->prev = newNode;

newNode->next = head;

head = newNode;

size++;

start.n = head;

}

}

int List::front() { return head->data; }

int List::back() { return tail->data; }

void List::popback() {

if (head != nullptr) {

Node\* th = tail;

tail = th->prev;

delete th;

tail->next = nullptr;

size--;

end.n = tail->next;

}

}

void List::popfront() {

if (head != nullptr) {

Node\* th = head;

head = th->next;

delete th;

head->prev = nullptr;

size--;

start.n = head;

}

}

bool List::empty() { return size == 0; }

List& List::operator=(const List& a) {

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

Node\* node = head;

while (node != nullptr) {

head = node->next;

delete node;

node = head;

size--;

}

Node\* th = a.head;

while (th != nullptr) {

pushback(th->data);

th = th->next;

}

start = a.start;

end = a.end;

return \*this;

}

int& List::operator[](int index) {

if (index < size && index >= 0) {

Node\* th = head;

for (int i = 0; i < index; i++) th = th->next;

return th->data;

}

else {

cout << "Bad index, man!" << endl;

exit(222);

}

}

int& List::operator ()() {

return size;

}

List List::operator\*(List& a) {

int tempsize = size;

int b = 5;

List templist(tempsize);

for (int i = 0; i < tempsize; i++) templist[i] = (\*this)[i] \* b;

return templist;

}

ostream& operator<<(ostream& out, const List& a) {

Node\* th = a.head;

while (th != nullptr) {

cout << th->data << " ";

th = th->next;

}

cout << endl;

return out;

}

istream& operator>>(istream& in, const List& a) {

Node\* th = a.head;

while (th != nullptr) {

in >> th->data;

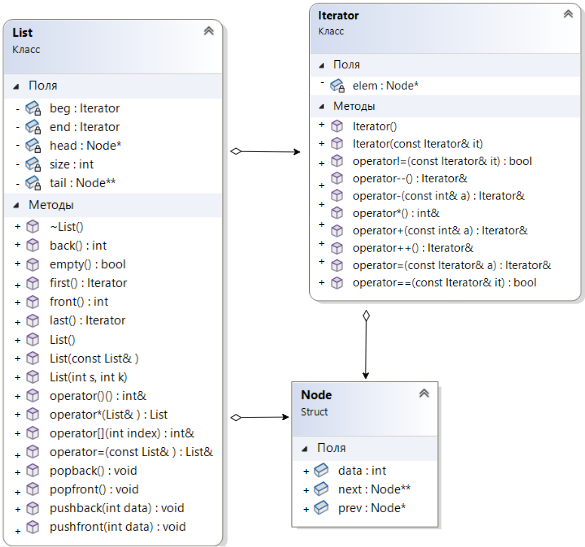
th = th->next;

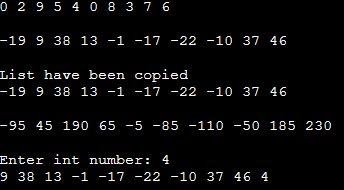
}

return in;

}

Диаграмма и вывод в консоль





Контрольные вопросы

1. Что такое абстрактный тип данных? Привести примеры АТД

АТД - тип данных, определяемый только через операции, которые могут

выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

template <typename T>

T & Stack<T>::push(){ //операция добавления элементов в АТД типа стек

return head->data;

}

2. Привести примеры абстракции через параметризацию.

template <typename T>

class node {

T data;

node\* next, \*prev;

}

3. Привести примеры абстракции через спецификацию.

class complex {

double re, im;

friend complex operator+(complex, complex);

public:

complex() { re = im = 0;}

complex(double r) { re = r; im = 0; }

complex(double r, double i) { re = r; im = i; }

};

complex operator+(complex a1, complex a2) {

return complex(a1.re + a2.re, a1.im + a2.im);

}

Для поддержки смешанной арифметики комплексных и действительных чисел (Re x, In y) + double A требуется специфицировать перегруженную функцию.

complex operator+(complex,complex);

complex operator+(complex,double);

complex operator+(double,complex);

Абстракция через спецификацию достигается за счет того, что операции

представляются как часть типа (абстракция вычислений, перегрузка операторов).

4. Что такое контейнер? Привести примеры.

Контейнер – это объект. Имя контейнера – это имя переменной. Контейнер, так же как и другие объекты, обладает временем жизни. Время жизни контейнера в общем случае не зависит от времени жизни его элементов. Элементами контейнера могут любые объекты, в том числе, и другие контейнеры.

class queue() {

int size;

node \*head, \*tail;

…

};

//в основной функции

queue\* q; //контейнер-очередь

float a = 7.5;

q->head = a; //головой очереди q является float переменная

queue \*c;

c->tail = q; //хвостом очереди c является контейнер queue

5. Какие группы операций выделяют в контейнерах?

* Операции доступа к элементам

template <typename T>

class queue() {

int size;

node \*head, \*tail;

node & getHead() { return head->data; }

…

};

* Операции замены значений элементов;

void queue::setHead(node head){ this->head = head; }

* Операции добавления и удаления элементов или групп элементов;

T queue::pop(){

T data = tail->data;

size–;

tail = tail->prev;

tail->next = nullptr;

return data;

}

* Операции поиска элементов и групп элементов;

int queue::find\_index(T data) {

int counter = 0;

node \* curr = this->head;

while (counter < size) {

if (curr->data = data) return counter;

curr = curr->next;

counter++;

}

}

* Операции объединения контейнеров;

queue queue::merge(queue q1, queue q2) {

q1->tail->next = q2->head;

q1->tail = q2->tail;

delete q2;

return q1;

}

* Специальные операции, которые зависят от вида контейнера.

bool queue::isCircular(queue q) {

return (q->tail->next == q->head); }

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

Доступ к элементам контейнера бывает: последовательный, прямой и

ассоциативный.

Прямой доступ – это доступ по индексу. Например, a[10] – требуется найти

элемент контейнера с номером 10 (с учетом индексации с нуля)

Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Пусть имеется контейнер–словарь, в котором хранится информация, состоящая, как минимум из двух полей: слово и его перевод.

Индексом будут данные поля, например, q[T data]. Поле, с содержимым которого ассоциируется элемент контейнера, называется ключом или полем доступа. Элемент, с которым ассоциируется ключ, называется

значением. Контейнер, который представляет ассоциативный доступ, состоит из пар «ключ-значение» (ключ - T data, значение - node name).

При последовательном доступе осуществляется перемещение от элемента к элементу контейнера.

Например, дана очередь, и для того, чтобы найти индекс элемента, требуется поэлементно пройти по каждому узлу.

int queue::find\_index(T data) {

int counter = 0;

node \* curr = this->head;

while (counter < size) {

if (curr->data = data) return counter;

curr = curr->next;

counter++;

}

}

7. Что такое итератор?

Итератор (перечислитель) — интерфейс, предоставляющий доступ к элементам контейнера и навигацию по ним. Главное предназначение итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы пользователя с ним как с простой последовательностью или списком. Итератор можно реализовать как класс, представляющий такой же набор операций.

В С++ итератор реализуется как класс, который имеет такой же интерфейс, как и указатель для совместимости с массивами.

8. Каким образом может быть реализован итератор?

class iterator() {

friend class queue; //поддержка работы с классом очередей

public:

iterator() { elem = nullptr; }

…

iterator & operator++() {  
 elem = elem->next; //реализация части кода с переходом на следующий элемент

}

…

};

9. Каким образом можно организовать объединение контейнеров?

queue queue::merge(queue q1, queue q2) {

q1->tail->next = q2->head;

q1->tail = q2->tail;

delete q2;

return q1;

}

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

Ассоциативный доступ (индексом будут данные поля, например, q[T data]. Поле, с содержимым которого ассоциируется элемент контейнера, называется ключом или полем доступа. Элемент, с которым ассоциируется ключ, называется значением)

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

Стек

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

a. int mas=10; //переменная типа int

b. 2. int mas; //переменная типа int

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

d. 4. int mas[100]; //массив переменных int

Ответ: d

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

a. int a[]={1,2,3,4,5}; //заполненный массив переменных int

b. 2. int mas[30]; //пустой массив переменных int

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

d. 4. int mas; //переменная типа int

Ответ: d

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

В С++ массивы поддерживают доступ по индексу. Доступ по индексу - это прямой доступ.

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

В С++ в линейной списке нет доступа по индексу, и требуется пройти по каждому элементу. Это последовательный доступ.