МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Контейнеры: список**

Выполнил студент ИВТ-22-2б:

Мифтахов Марат Ринатович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

**Задание**

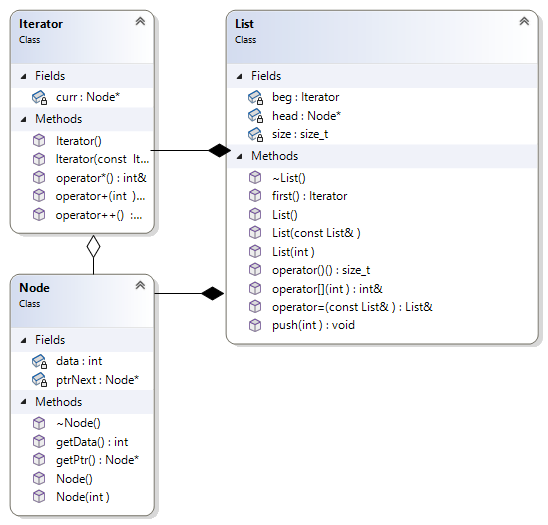
1. Класс-контейнер Список
2. Поля типа int

Операции:

1. [] - доступ по индексу
2. () - определение размера списка
3. вектор — уможение элементов списков a[i]b[i]
4. +n – переход вправо на кол-во элементов n (класс-итератор)

**Диаграмма классов**

Ниже представлена UML-диаграмма классов.



**Программный код**

Файл list.h:

#include <iostream>

class Iterator;

class List;

class Node {

friend class List;

friend class Iterator;

public:

Node();

Node(int);

~Node();

int getData();

Node\* getPtr();

private:

int data;

Node\* ptrNext;

};

class Iterator {

friend class List;

public:

Iterator();

Iterator(const Iterator&);

void operator++();

void operator+(int);

int& operator\*();

private:

Node\* curr;

};

class List {

public:

List();

List(int);

List(const List&);

~List();

int& operator[](int);

List& operator=(const List&);

size\_t operator()();

friend List operator\*(List&, List&);

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

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

void push(int);

Iterator first() { return beg; }

private:

Node\* head;

size\_t size;

Iterator beg;

};

Файл list.cpp:

#include "list.h"

#include <iostream>

#include <exception>

/// <summary>

/// int data,

/// Node\* ptrNext

/// </summary>

Node::Node() {

data = 0;

ptrNext = nullptr;

}

Node::Node(int data) {

this->data = data;

ptrNext = nullptr;

}

Node::~Node() {

if (ptrNext != nullptr) {

delete ptrNext;

}

}

int Node::getData() {

return data;

}

Node\* Node::getPtr() {

return ptrNext;

}

/// <summary>

/// Node\* head;

/// </summary>

List::List() {

head = nullptr;

size = 0;

beg.curr = head;

}

List::List(int data) {

head = new Node(data);

size = 1;

beg.curr = head;

}

List::List(const List& tList) {

Node\* curr = tList.head;

beg.curr = tList.head;

while (curr != nullptr) {

int a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

}

List::~List() {

if (head != nullptr) {

delete head;

head = head->ptrNext;

}

size = 0;

beg.curr = head;

}

void List::push(int data) {

if (head == nullptr) {

head = new Node(data);

beg.curr = head;

}

else {

Node\* curr = head;

while (curr->ptrNext != nullptr) {

curr = curr->ptrNext;

}

curr->ptrNext = new Node(data);

}

size++;

}

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

Node\* curr = head;

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

curr = curr->ptrNext;

}

return curr->data;

}

size\_t List::operator()() {

return size;

}

List operator\*(List& fList, List& sList) {

List tmpList;

int maxSize = fList.size > sList.size ? fList.size : sList.size;

int minSize = fList.size <= sList.size ? fList.size : sList.size;

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

int a = fList[i];

int b = sList[i];

tmpList.push(a \* b);

}

for (int i = 0; i < maxSize - minSize; i++) {

tmpList.push(0);

}

return tmpList;

}

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

Node\* curr = tList.head;

beg.curr = tList.head;

while (curr != nullptr) {

int a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

return \*this;

}

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

Node\* curr = tList.head;

for (int i = 0; i < tList.size; i++) {

out << curr->getData() << " ";

curr = curr->getPtr();

}

return out;

}

std::istream& operator>>(std::istream& in, List& tList) {

int a;

in >> a;

tList.push(a);

return in;

}

/// <summary>

/// Node\* curr

/// </summary>

Iterator::Iterator() {

curr = nullptr;

}

Iterator::Iterator(const Iterator& it) {

curr = it.curr;

}

void Iterator::operator++() {

curr = curr->ptrNext;

}

void Iterator::operator+(int n) {

for (int i = 0; i < n && curr->ptrNext != nullptr; i++) {

curr = curr->ptrNext;

}

}

int& Iterator::operator\*() {

return curr->data;

}

Файл main.cpp:

#include <iostream>

#include "list.h"

#include <Windows.h>

using namespace std;

void listFillFunction(List&, int);

int main() {

int n;

cout << "Enter the number of elements in da list: "; cin >> n;

List ListOne;

listFillFunction(ListOne, n);

cout << "Your list have length " << ListOne() << "\n"

<< "And this are your list: " << ListOne << endl;

cout << "Enter the number of elements in da second list: "; cin >> n;

List ListTwo;

listFillFunction(ListTwo, n);

cout << "Your list have length " << ListTwo() << "\n"

<< "And this are your list: " << ListTwo << endl;

List ListThree = ListOne \* ListTwo;

cout << "Their \* is: " << ListThree;

Iterator it = ListThree.first();

cout << "Enter the number of elements you want to move for: "; cin >> n;

it + n;

cout << "Third list have num " << \*it << " under " << n+1 << " number" << endl;

return 0;

}

void listFillFunction(List& tList, int n) {

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

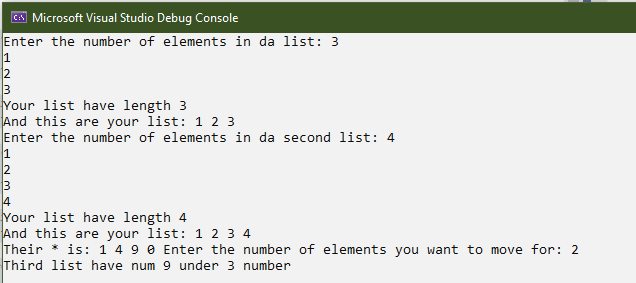
cin >> tList;

}

}

**Вывод программы**

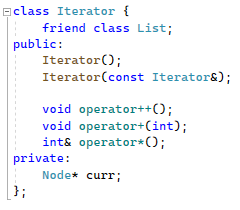
Ниже представлен вывод программы на консоль.

****

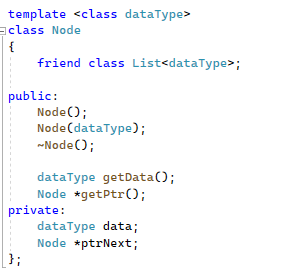
**Контрольные вопросы**

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

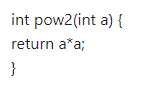
Ответ: АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов.



1. Привести примеры абстракции через параметризацию.

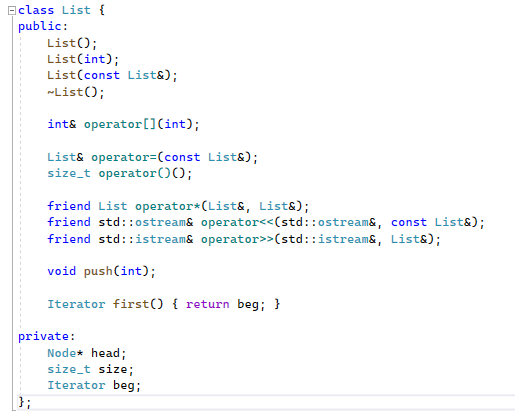
Ответ: класс Node – параметризованный класс

1. Привести примеры абстракции через спецификацию.

Ответ: функция возведения числа в квадрат

1. Что такое контейнер? Привести примеры.

Ответ: контейнер — набор однотипных элементов с операциями над ними.



1. Какие группы операций выделяют в контейнерах?

Ответ:

* + 1. Операции доступа к элементам
    2. Операции добавления и удаления элементов
    3. Операции поиска элементов
    4. Операции объединения контейнеров
    5. Специальные операции

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

Ответ: 1. через внутренние операции 2. через итератор (см. данную лабораторную)

1. Что такое итератор?

Ответ: итератор — объект, обеспечивающий доступ к элементам контейнера.

1. Каким образом может быть реализован итератор?

Ответ: итератор может быть реализован как класс

1. Каким образом можно организовать объединение контейнеров?

Ответ: сцепление контейнеров, объединение упорядоченных контейнеров (с упорядоченным полученным); объединение контейнеров, как пересечения/ объединения множеств

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

Ответ: ассоциативный доступ по ключу

1. 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?

* + 1. int mas=10;
    2. 2. int mas;
    3. 3. struct {char name[30]; int age;} mas;
    4. 4. int mas[100];

Ответ: D

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

* + 1. int a[]={1,2,3,4,5};
    2. int mas[30];
    3. struct {char name[30]; int age;} mas[30];
    4. int mas;

Ответ: h

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

Ответ: прямой доступ через перемещение указателя.

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

Ответ: Последовательный доступ: нужно пройтись по предыдущим элементам, чтобы достичь заданного.