Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 6 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

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.

*Вариант 12*

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

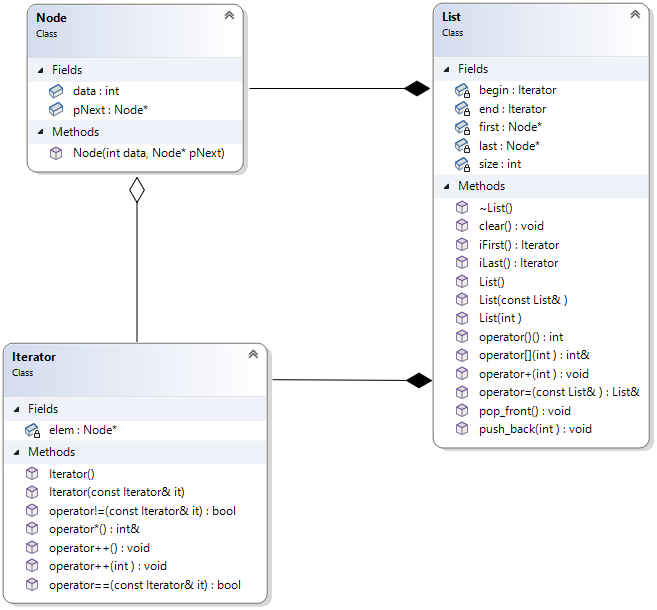
[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам вектора;

++ - переход к следующему элементу (с помощью класса-итератора).

**UML-диаграмма**



**Программное решение**

List.h

#pragma once

#include <iostream>

using namespace std;

class Node

{

public:

int data = 0;

Node\* pNext = nullptr;

Node(int data = 0, Node\* pNext = nullptr) {

this->data = data;

this->pNext = pNext;

}

};

class Iterator

{

private:

friend class List;

Node\* elem;

public:

Iterator() { elem = nullptr; }

Iterator(const Iterator& it) { elem = it.elem; }

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

void operator ++ () { elem = elem->pNext; };

void operator ++ (int) { elem = elem->pNext; };

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

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

};

class List

{

private:

int size;

Node\* first;

Node\* last;

Iterator begin;

Iterator end;

public:

List();

List(int);

List(const List&);

~List();

void push\_back(int);

void pop\_front();

void clear();

List& operator = (const List&);

int& operator [] (int);

int operator () ();

void operator + (int);

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

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

Iterator iFirst() { return begin; }

Iterator iLast() { return end; }

};

List.cpp

#include <iostream>

#include "List.h"

using namespace std;

//конструктор по умолчанию

List::List() {

size = 0;

first = nullptr;

last = nullptr;

}

//конструктор с параметрами

List::List(int s) {

size = s;

first = nullptr;

last = nullptr;

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

int newData = rand() % 10;

Node\* addNode = new Node(newData);

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

}

begin.elem = first;

end.elem = nullptr;

}

//конструктор копирования

List::List(const List& list) {

size = 0;

first = nullptr;

last = nullptr;

Node\* current = list.first;

while (current != nullptr) {

Node\* addNode = new Node();

addNode->data = current->data;

addNode->pNext = nullptr;

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

current = current->pNext;

size++;

}

begin = list.begin;

end = list.end;

}

//деструктор

List::~List() {

clear();

}

//функция для добавления элемента в конец списка

void List::push\_back(int data) {

if (first == nullptr) {

first = new Node(data);

}

else {

Node\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node(data);

}

size++;

}

//функция для удаления элемента из начала списка

void List::pop\_front() {

Node\* temp = first;

first = first->pNext;

delete temp;

size--;

}

//функция для удаления всех элементов списка

void List::clear() {

while (size > 0) {

pop\_front();

}

}

//перегрузка оператора присваивания

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

if (this == &other) {

return \*this;

}

while (first != nullptr) {

Node\* temp = first;

first = first->pNext;

delete temp;

}

Node\* currentOther = other.first;

while (currentOther != nullptr) {

push\_back(currentOther->data);

currentOther = currentOther->pNext;

}

return \*this;

}

//перегрузка квадратных скобок

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

int count = 0;

Node\* current = this->first;

while (current != nullptr) {

if (count == index) {

return current->data;

}

current = current->pNext;

count++;

}

}

//перегрузка круглых скобок

int List::operator () () {

return size;

}

//перегрузка оператора сложения

void List::operator + (int add) {

Node\* current = this->first;

while (current != nullptr) {

current->data = current->data + add;

current = current->pNext;

}

}

//перегрузка операции ввода

istream& operator >> (istream& in, List& list) {

int count;

cout << "Введите количество элементов: ";

in >> count;

cout << "Введите значения: ";

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

int data;

in >> data;

list.push\_back(data);

}

return in;

}

//перегрузка операции вывода

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

if (list.first == nullptr) {

cout << "Список пуст!" << endl;

return out;

}

Node\* current = list.first;

while (current != nullptr) {

cout << current->data << " ";

current = current->pNext;

}

cout << endl;

return out;

}

main.cpp

/\* Вариант 12

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

[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам списка;

++ - переход к следующему элементу ( с помощью класса-итератора).

\*/

#include "List.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "RU");

List list1;

cout << "Список 1: " << list1 << endl;

List list2(5);

cout << "Список 2: " << list2;

list2.push\_back(12);

cout << "Список 2 после добавления: " << list2;

cout << "Элемент с идексом 4 это " << list2[4] << endl;

cout << "Размер списка 2: " << list2() << endl;

list2 + (5);

cout << "Список 2 после увеличения элементов на 5: " << list2;

cout << "Разыменованное значение первого элемента, возвращаемое итератором: " << \*(list2.iFirst()) << endl;

Iterator i = list2.iFirst();

i++;

cout << "Переход к следующему элементу через итератор: " << \*i << endl;

cout << "Вывод списка на экран с помощью итератора: ";

for (i = list2.iFirst(); i != list2.iLast(); ++i) {

cout << \*i << " ";

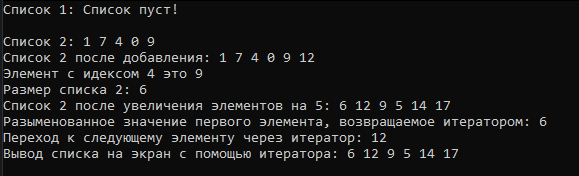
}

cout << endl;

return 0;

}

**Результат работы программы**



**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, содержащий другие однотипные объекты. Примеры: двусвязный список, массив, стек, множество.

class queue() {

int size;

node\* head, \* tail;

};

int main() {

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

return 0;

}

*5. Какие группы операций выделяют в контейнерах?*

* Операции доступа к элементам, которые обеспечивают и операцию замены значений элементов;
* Операции добавления и удаления элементов или групп элементов;
* Операции поиска элементов и групп элементов;
* Операции объединения контейнеров.

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

Виды доступа: последовательный, прямой и ассоциативный.

Последовательный доступ – перемещение от элемента к элементу. Пример: a.next.

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

Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Пример: a[“word”].

*7. Что такое итератор?*

Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера.

*8. Каким образом может быть реализован итератор?*

Итератор реализуется как класс, который имеет такой же интерфейс, как и указатель для совместимости с массивами.

*9. Каким образом можно организовать объединение контейнеров?*

Наиболее часто используется операция объединения двух контейнеров с получением нового контейнера. Она может быть реализована в разных вариантах:

* Простое сцепление двух контейнеров: в новый контейнер попадают сначала элементы первого контейнера, потом второго, операция не коммутативна.
* Объединение упорядоченных контейнеров, новый контейнер тоже будет упорядочен, операция коммутативна.
* Объединение контейнеров как объединение множеств, в новый контейнер попадают только те элементы, которые есть хотя бы в одном контейнере, операция коммутативна.
* Объединение контейнеров как пересечение множеств, в новый контейнер попадают только те элементы, которые есть в обоих контейнерах, операция коммутативна.

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

Ассоциативный доступ.

*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;

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. 2. int mas[30];

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

d. 4. int mas;

Ответ: d

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

Прямой доступ.

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

Последовательный доступ.