Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №6

Тема: «АТД. Контейнеры.»

Выполнили

Студенты группы РИС-22-2б

Худеньких В.Д.

Проверил доц. 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.

ВАРИАНТ 15:

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).

**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. Примером абстрактного типа данных является класс в языке С++.

Пример абстрактного типа данных - класс Итератор

* 1. Привести примеры абстракции через параметризацию.

В этом примере класс «Stack» параметризован типом «T», который определяется при создании объекта класса. Это позволяет использовать один и тот же класс для работы с различными типами данных, не переписывая его каждый раз.

* 1. Привести примеры абстракции через спецификацию.

Класс «Shape» является абстрактным классом, так как содержит чисто виртуальные методы «area()» и «perimeter()», которые не имеют конкретной реализации в этом классе. Вместо этого, эти методы должны быть переопределены в производных классах, таких как «Rectangle» и `Circle`.

Класс «Rectangle» является производным от класса «Shape» и реализует методы «area()» и «perimeter()» для прямоугольника. Он содержит приватные поля «width» и «height», которые используются для вычисления площади и периметра.

Класс «Circle» также является производным от класса «Shape» и реализует методы «area()» и «perimeter()» для круга. Он содержит приватное поле «radius», которое используется для вычисления площади и периметра.

Оба класса «Rectangle» и «Circle» реализуют интерфейс, определенный в абстрактном классе «Shape», что позволяет использовать полиморфизм для работы с различными типами геометрических фигур.

* 1. Что такое контейнер? Привести примеры.

**Контейнер** – набор однотипных элементов. Встроенные массивы в С++ - частный случай контейнера.

* 1. Какие группы операций выделяют в контейнерах?

Среди всех операций контейнера можно выделить несколько типовых групп:

* + - 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;
    - Специальные операции, которые зависят от вида контейнера.
  1. Какие виды доступа к элементам контейнера существуют? Привести примеры.

Доступ к элементам контейнера бывает: последовательный, прямой и ассоциативный.

* 1. Что такое итератор?

Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера. Итератор может быть реализован как часть класса-контейнера в виде набора методов:

v.first() перейти к первому элементу

v.last() перейти к последнему элементу

v.next() перейти к следующему элементу

v.prev() перейти к предыдущему элементу

v.skip(n) перейти на n элементов вперед

v.skip(-n) перейти на n элементов назад

v.current() получить текущий элемент

* 1. Каким образом может быть реализован итератор?

Как класс

* 1. 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.
      * Для контейнеров-множеств может быть еще реализована операция вычитания, в контейнер попадают только те элементы первого контейнера, которых нет во втором, операция не коммутативна.
      * Извлечение части элементов из контейнера и создание нового контейнера. Эта операция может быть выполнена с помощью конструктора, а часть контейнера задается двумя итераторами.
  1. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?

По ключу

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

Стек

* 1. 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];

D – массивы являются контейнером

* 1. 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;

d, т.к. обычное целочисленная переменная под названием mas

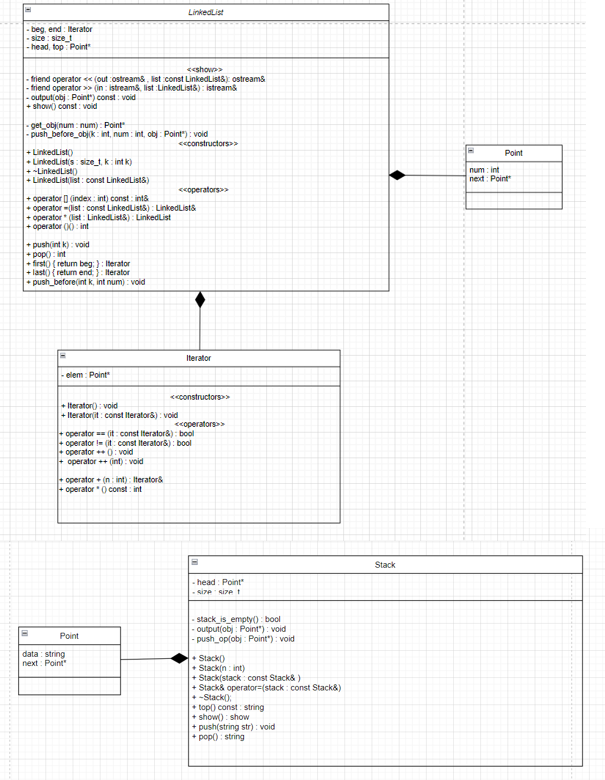
* 1. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?

Прямой доступ: через смещение указателя

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

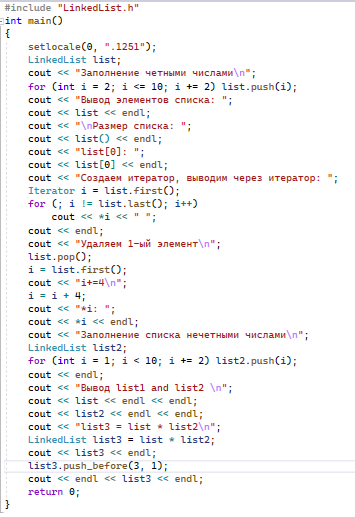
Последовательный доступ: нужно пройтись по предыдущим элементам, чтобы достичь заданного.

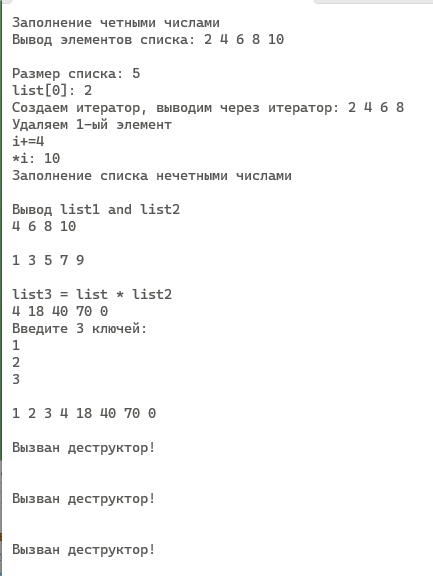
**UML таблица**



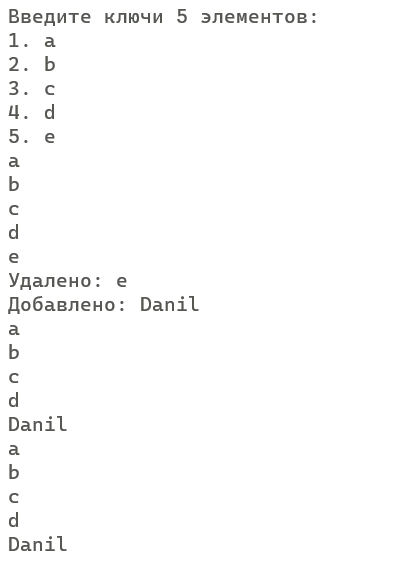
**Скриншоты работы программы**

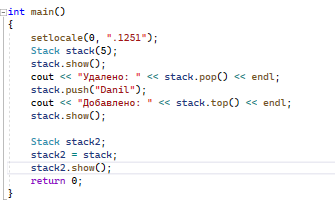
**Список:**

****

****

**Cтек:**

****

****

**Код программы**

**Main.cpp:**

#include "LinkedList.h"

int **main**()

{

setlocale(0, ".1251");

LinkedList list;

cout << "?????????? ??????? ???????\n";

for (int i = 2; i <= 10; i += 2) list.push(i);

cout << "????? ????????? ??????: ";

cout << list << endl;

cout << "\n?????? ??????: ";

cout << list() << endl;

cout << "list[0]: ";

cout << list[0] << endl;

cout << "??????? ????????, ??????? ????? ????????: ";

Iterator i = list.first();

for (; i != list.last(); i++)

cout << \*i << " ";

cout << endl;

cout << "??????? 1-?? ???????\n";

list.pop();

i = list.first();

cout << "i+=4\n";

i = i + 4;

cout << "\*i: ";

cout << \*i << endl;

cout << "?????????? ?????? ????????? ???????\n";

LinkedList list2;

for (int i = 1; i < 10; i += 2) list2.push(i);

cout << endl;

cout << "????? list1 and list2 \n";

cout << list << endl << endl;

cout << list2 << endl << endl;

cout << "list3 = list \* list2\n";

LinkedList list3 = list \* list2;

cout << list3 << endl;

list3.push\_before(3, 1);

cout << endl << list3 << endl;

return 0;

}

**LinkedList.cpp:**

#include "LinkedList.h"

LinkedList::**LinkedList**()

{

head = nullptr;

size = 0;

top = nullptr;

}

LinkedList::**LinkedList**(size\_t s, int k = 0)

{

size = s;

if (size != 0) {

//---------??????? ?????? ? ???????? ?? ???????? k---------//

Point\* obj = new Point;

obj->num = k;

head = obj;

//--------------------------------------------------------//

//-------??????? ????????? size - 1 ????????--------------//

Point\* temp = head;

for (int i = size - 1; i != 0; i--)

{

obj = new Point;

temp->next = obj;

obj->num = k;

temp = obj;

}

// ??????? + ?????????

top = obj;

beg.elem = head;

end.elem = top;

}

}

void LinkedList::**push**(int k)

{

//------------------------------------------//

// ???? ?????? 0, ?? ?????? ?????? = ??????

//------------------------------------------//

if (size == 0)

{

head = new Point;

head->num = k;

top = head;

beg.elem = head;

}

//------------------------------------------//

// ????? ????? ????? ????????? ????? ???????

//------------------------------------------//

else {

Point\* obj = new Point;

top->next = obj;

obj->num = k;

top = obj;

}

// ??????????? ??????, ???????? ?? ?????

end.elem = top;

size++;

}

int LinkedList::**pop**()

{

// ??????? ????????????? ?????, ??????? ??????, ?????????? ??????

int k = head->num;

Point\* temp = head;

head = head->next;

size--;

delete temp;

beg.elem = head;

return k;

}

LinkedList::~**LinkedList**()

{

cout << endl << "?????? ??????????!" << endl << endl;

while (head != nullptr)

{

pop();

}

}

//----------------------------?????----------------------------------//

void LinkedList::**output**(Point\* obj) const

{

cout << obj->num << " ";

if (obj == top) return;

output(*obj->next*);

}

void LinkedList::**show**() const

{

if (size == 0) cout << "?????? ????" << endl;

else output(*head*);

}

ostream& operator <<(ostream& out, const LinkedList& list)

{

list.show();

return out;

}

istream& operator>>(istream& in, LinkedList& list)

{

int k;

in >> k;

list.push(k);

return in;

}

//-------------------------------------------------------------------//

int& LinkedList::operator[](int index) const

{

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

{

Point\* temp = head;

int count = 0;

while (temp != nullptr)

{

if (count == index)

return temp->num;

temp = temp->next;

++count;

}

}

cout << endl << endl;

perror("?????? ??????????????");

cout << endl;

exit(1);

}

LinkedList::**LinkedList**(const LinkedList& list)

{

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

{

this->push(list[i]);

}

}

LinkedList& LinkedList::operator =(const LinkedList& list)

{

// ??????? size\_temp, ?.?. pop ????????? size, ????? ?????????

int size\_temp = size;

// ??????? ??? ????????

for (int i = 0; i < size\_temp; i++)

{

this->pop();

}

// ???????? ????? push()

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

{

this->push(list[i]);

}

return \*this;

}

LinkedList LinkedList::operator \* (LinkedList& list)

{

// ??????? ????? ?????? ?? ???? ???????, ???????? ????? ?????? ????????

LinkedList new\_list(max((int)size, list()), 0);

// ???????? ?? ???? ??????? ???????????? ??????? ??????

for (int i = 0; i < min((int)size, list()); i++)

{

new\_list[i] = (\*this)[i] \* list[i];

}

return new\_list;

}

Point\* LinkedList::**get\_obj**(int num)

{

Point\* obj = head;

for (int i = 1; i < num; ++i)

{

obj = obj->next;

}

return obj;

}

void LinkedList::**push\_before\_obj**(int k, int num, Point\* obj)

{

if (k == 0) return;

Point\* new\_obj = new Point;

cin >> new\_obj->num;

push\_before\_obj(k - 1, num, *obj*);

if (num == 1)

{

new\_obj->next = head;

head = new\_obj;

}

else {

new\_obj->next = obj->next;

obj->next = new\_obj;

}

size++;

}

void LinkedList::**push\_before**(int k, int num)

{

if (size != 0 && num >= 1 && num < size)

{

if (k == 0) return;

Point\* obj = get\_obj(num - 1);

cout << "??????? " << k << " ??????: " << endl;

push\_before\_obj(k, num, *obj*);

}

else cout << "?????? ???? ??? ?????? ???????? ??????" << endl;

}

**Stack.cpp:**

#include "Stack.h"

void Stack::**push\_op**(Point\* obj)

{

if (obj == nullptr) return;

push\_op(*obj->next*);

this->push(obj->data);

}

Stack& Stack::operator=(const Stack& stack)

{

while (this->pop() != "");

Point temp = \*stack.head;

push\_op(*&temp*);

return \*this;

}

Stack::**Stack**(const Stack& stack)

{

Point temp = \*stack.head;

push\_op(*&temp*);

}

Stack::**Stack**(int n) : Stack()

{

if (n > 0)

{

string str;

cout << "??????? ????? " << n << " ?????????:\n";

for (int i = 1; i <= n; i++)

{

cout << i << ". ";

getline(*cin*, *str*);

push(str);

}

}

}

Stack::~**Stack**()

{

while (pop() != "");

}

void Stack::**push**(string str)

{

size++;

Point\* obj = new Point;

obj->data = str;

obj->next = head;

head = obj;

}

string Stack::**pop**()

{

string result = "";

if (!stack\_is\_empty())

{

result = head->data;

Point\* temp = head;

head = head->next;

delete temp;

size--;

}

return result;

}

void Stack::**output**(Point\* obj)

{

if (obj == nullptr) return;

output(*obj->next*);

cout << obj->data << endl;

}

void Stack::**show**()

{

if (stack\_is\_empty()) cout << "???? ????" << endl;

else output(*head*);

}