Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.6**

Дисциплина: «Информатика»

Тема Объектно-ориентированное программирование. АТД. Контейнеры Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

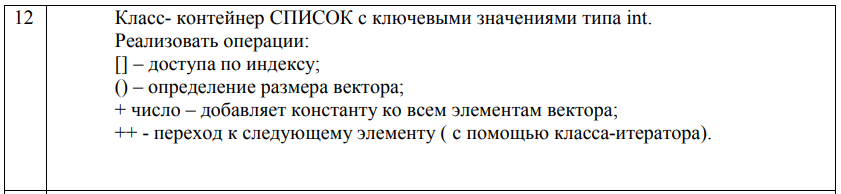
**Пермь 2023**

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация класса-контейнера.

**Постановка задачи**

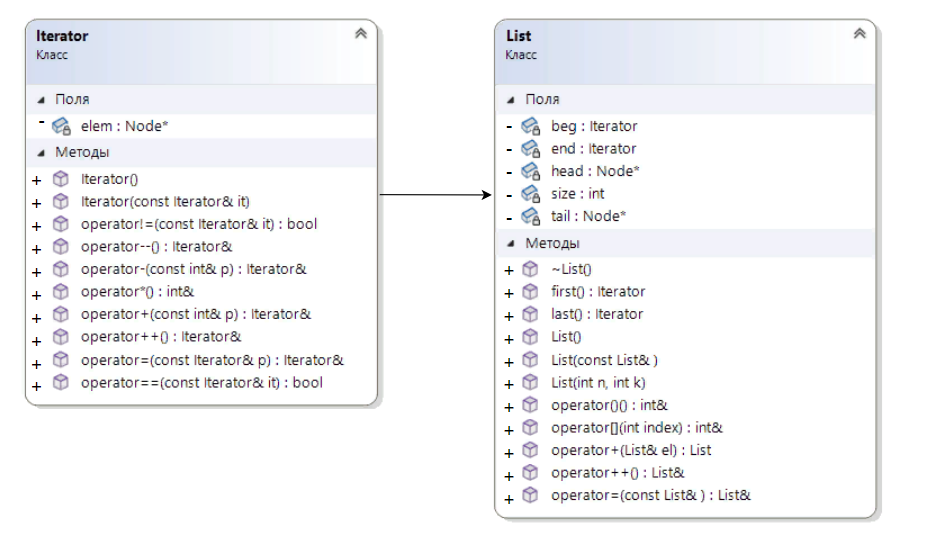
1. Определить класс-контейнер.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
5. Написать тестирующую программу, иллюстрирующую выполнение операций.



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение шаблонного класса List и определение необходимых методов и перегруженных операторов данного класса.
   2. Реализовать определение класса Iterator и определение необходимых методов и операторов данного класса.
   3. Реализовать применение этих функций в главной функции.
2. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
3. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
4. Поставленные задачи решены следующими действиями:
   1. В классе Iterator находится реализация итератора списка, операторов сравнения, инкремента и декремента и разыменовывания итератора.
   2. В классе List находится реализация списка для данных типа int, определение размера списка, доступ элемента списка по индексу, сложение векторов.

**UML-Диаграмма**



**Реализация задачи на языке С++**

**Заголовочный файл List.h**

#pragma once

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

Node\* prev;

Node()

{

data = 0;

next = nullptr;

prev = nullptr;

}

};

class Iterator

{

private:

friend class List;

Node\* elem;

public:

Iterator()

{

elem = nullptr;

}

Iterator(const Iterator& it)

{

elem = it.elem;

}

Iterator& operator=(const Iterator& p)

{

elem = p.elem;

return \*this;

}

bool operator==(const Iterator& it)

{

return elem == it.elem;

}

bool operator!=(const Iterator& it)

{

return elem != it.elem;

}

Iterator& operator++()

{

elem = elem->next;

return \*this;

}

Iterator& operator--()

{

elem = elem->prev;

return \*this;

}

Iterator& operator+(const int& p)

{

for (int i = 0; i < p; i++) elem = elem->next;

return \*this;

}

Iterator& operator-(const int& p)

{

for (int i = 0; i < p; i++) elem = elem->prev;

return \*this;

}

int& operator \*() const

{

return elem->data;

}

};

class List

{

private:

int size;

Node\* head;

Node\* tail;

Iterator beg;

Iterator end;

public:

List();

List(int n, int k = 0); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

List(const List&); // Конструктор копирования

~List(); // Деструктор

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

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

List& operator=(const List&);

int& operator[](int index); // Операция доступа по индексу

int& operator()(); // Длина списка

List operator+(List& el); // Операция сложения

List& operator++(); // префиксная форма. Переход к другому элемента с помощью итератора

Iterator first()

{

return beg;

}

Iterator last()

{

return end;

}

void pushback(int); // добавление в конец

};

**Файл с описанием методов класса List.cpp**

#include "List.h"

List::List()

{

head = nullptr;

tail = nullptr;

size = 0;

}

List::List(int n, int k)

{

size = n;

head = nullptr; // пока нет элементов, помимо первого, нет и ссылки на следующий элемент

tail = nullptr;

Node\* node = new Node; // выделение памяти под узел

node->data = k; // заполнение первого узла данными

node->next = nullptr;

node->prev = nullptr;

head = node; //присваиваем данные первому узлу

tail = node;

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

{

Node\* NewNode = new Node; // Выделение димнамической памяти под новый элемент

Node\* PrevNode = node; //Указатель на "предыдущий" элемент(изначально тот, на котором стоит указатель на следующий)

node->next = NewNode;

node = node->next; // Переход указателем на следующий элемент

node->data = k; // Запись нового элемента

node->next = nullptr; // Указатель на следующий элемент равен нулю, т. к. был записан последний элемент

node->prev = PrevNode;

}

tail = node;

beg.elem = head;

end.elem = tail->next;

}

List::List(const List&p)

{

Node\* node = p.head;

while (node != nullptr)

{

pushback(node->data);

node = node->next;

}

beg.elem = head;

end.elem = tail->next;

}

void List::pushback(int k)

{

Node\* newNode = new Node;

newNode->data = k;

if (head == nullptr)

head = newNode;

else

{

tail->next = newNode;

newNode->prev = tail;

}

tail = newNode;

end.elem = tail;

size++;

}

List::~List()

{

Node\* tmp = head;

while (tmp != nullptr)

{

head = tmp->next;

delete tmp;

tmp = head;

}

}

ostream& operator<<(ostream& out, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

out << tmp->data << " ";

tmp = tmp->next;

}

out << endl;

return out;

}

istream& operator>>(istream& in, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

in >> tmp->data;

tmp = tmp->next;

}

return in;

}

List List::operator+(List& p)

{

int tmpSize;

if (size > p.size)

tmpSize = p.size;

else

tmpSize = size;

List tmpList(tmpSize);

for (int i = 0; i < tmpSize; i++)

tmpList[i] = (\*this)[i] + p[i];

return tmpList;

}

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

{

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

{

Node\* tmp = head;

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

return tmp->data;

}

else

{

cout << "Index outside!" << endl;

exit(7);

}

}

int& List::operator()()

{

return size;

}

List& List::operator=(const List&p)

{

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

Node\* node = head;

while (node != nullptr) {

head = node->next;

delete node;

node = head;

size--;

}

Node\* curr = p.head;

while (curr != nullptr) {

pushback(curr->data);

curr = curr->next;

}

beg.elem = head;

end.elem = tail->next;

return \*this;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include <ctime>

#include "List.h"

using namespace std;

int main()

{

List A(10, 8);

cout << A << endl;

srand(time(NULL));

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

cout << A << endl;

List B(A);

cout << "Copy:\n";

cout << B << endl;

A = A + B;

cout << A << endl;

List C(10, 7);

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

cout << \*iter << " ";

return 0;}

**Ответы на вопросы**

1. Множество элементов, тип элементов не важен
2. Сортировка массива, одной функцией можно сортировать разные массивы и один массив можно сортировать разными функциями
3. Добавление элемента в конец списка
4. Набор однотипных элементов
5. Доступ, удаление, добавление, изменение
6. Последовательный, прямой, ассоциативный
7. Указатель на элемент контейнера
8. Через класс с перегрузкой разыменования
9. Слияние, склеивание
10. Ассоциативный
11. Стек
12. D
13. D
14. Прямой
15. Последовательный