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 № 15.7**

Дисциплина: «Информатика»

Тема Объектно-ориентированное программирование. Шаблоны классов Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

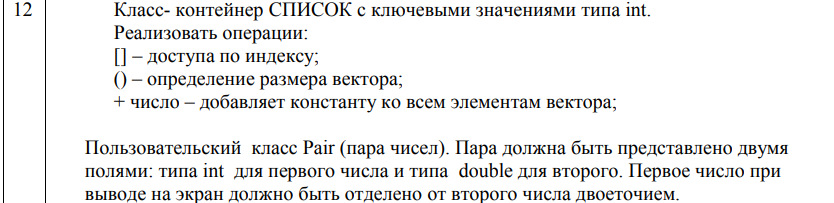
**Пермь 2023**

**Цель работы**

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

**Постановка задачи**

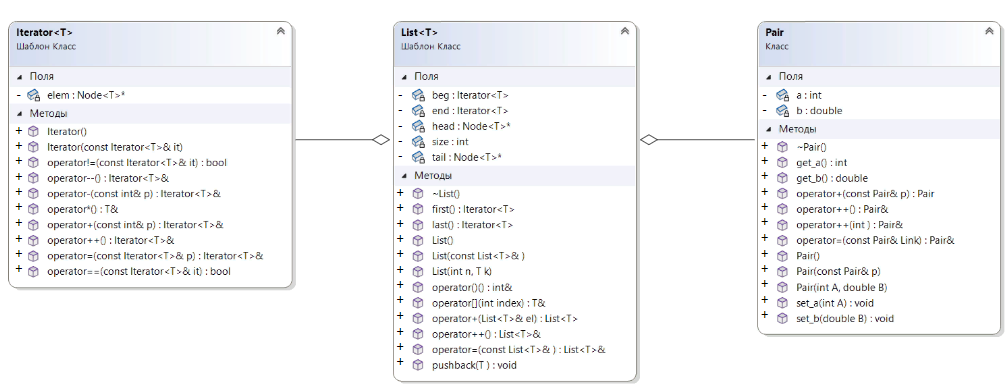
1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
6. Реализовать пользовательский класс (см. лабораторную работу №3).
7. Перегрузить для пользовательского класса операции ввода-вывода.
8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
9. Инстанцировать шаблон для пользовательского класса.
10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение шаблонного класса List и определение необходимых методов данного класса.
   2. Реализовать определение класса Pair 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 классах Pair находится реализация ввода-вывода и сложение чисел
   2. В шаблонном классе List находится реализация списка для различных типов данных, 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;

#ifndef LIST\_H

#define LIST\_H

template <class T>

struct Node

{

T data;

Node<T>\* next;

Node<T>\* prev;

Node()

{

next = nullptr;

prev = nullptr;

}

};

template <class T>

class Iterator

{

private:

template <class T>

friend class List;

Node<T>\* elem;

public:

Iterator()

{

elem = nullptr;

}

Iterator(const Iterator<T>& it)

{

elem = it.elem;

}

Iterator<T>& operator=(const Iterator<T>& p)

{

elem = p.elem;

return \*this;

}

bool operator==(const Iterator<T>& it)

{

return elem == it.elem;

}

bool operator!=(const Iterator<T>& it)

{

return elem != it.elem;

}

Iterator<T>& operator++()

{

elem = elem->next;

return \*this;

}

Iterator<T>& operator--()

{

elem = elem->prev;

return \*this;

}

Iterator<T>& operator+(const int& p)

{

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

return \*this;

}

Iterator<T>& operator-(const int& p)

{

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

return \*this;

}

T& operator \*() const

{

return elem->data;

}

};

template <class T>

class List

{

private:

int size;

Node<T>\* head;

Node<T>\* tail;

Iterator<T> beg;

Iterator<T> end;

public:

List();

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

List(const List<T>&); // Конструктор копирования

~List(); // Деструктор

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

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

List<T>& operator=(const List<T>&);

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

int& operator()(); // Длина списка

List<T> operator+(List<T>& el); // Операция сложения

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

Iterator<T> first()

{

return beg;

}

Iterator<T> last()

{

return end;

}

void pushback(T); // добавление в конец

};

template <class T>

List<T>::List()

{

head = nullptr;

tail = nullptr;

size = 0;

}

template <class T>

List<T>::List(int n, T k)

{

size = n;

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

tail = nullptr;

Node<T>\* node = new Node<T>; // 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<T>\* NewNode = new Node<T>; // Выделение димнамической памяти под новый элемент

Node<T>\* 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;

}

template <class T>

List<T>::List(const List<T>& p)

{

Node<T>\* node = p.head;

while (node != nullptr)

{

pushback(node->data);

node = node->next;

}

beg.elem = head;

end.elem = tail->next;

}

template <class T>

void List<T>::pushback(T k)

{

Node<T>\* newNode = new Node<T>;

newNode->data = k;

if (head == nullptr)

head = newNode;

else

{

tail->next = newNode;

newNode->prev = tail;

}

tail = newNode;

end.elem = tail;

size++;

}

template <class T>

List<T>::~List()

{

Node<T>\* tmp = head;

while (tmp != nullptr)

{

head = tmp->next;

delete tmp;

tmp = head;

}

}

template <class T>

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

{

Node<T>\* tmp = p.head;

while (tmp != nullptr)

{

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

tmp = tmp->next;

}

out << endl;

return out;

}

template <class T>

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

{

Node<T>\* tmp = p.head;

while (tmp != nullptr)

{

in >> tmp->data;

tmp = tmp->next;

}

return in;

}

template <class T>

List<T> List<T>::operator+(List<T>& p)

{

int tmpSize;

if (size > p.size)

tmpSize = p.size;

else

tmpSize = size;

List<T> tmpList(tmpSize);

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

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

return tmpList;

}

template <class T>

T& List<T>::operator[](int index)

{

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

{

Node<T>\* tmp = head;

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

return tmp->data;

}

else

{

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

exit(7);

}

}

template <class T>

int& List<T>::operator()()

{

return size;

}

template <class T>

List<T>& List<T>::operator=(const List<T>& p)

{

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

Node<T>\* node = head;

while (node != nullptr) {

head = node->next;

delete node;

node = head;

size--;

}

Node<T>\* curr = p.head;

while (curr != nullptr) {

pushback(curr->data);

curr = curr->next;

}

beg.elem = head;

end.elem = tail->next;

return \*this;

}

#endif

**Заголовочный файл Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int a;

double b;

public:

Pair();

Pair(int A, double B);

Pair(const Pair& p);

~Pair();

int get\_a();

void set\_a(int A);

double get\_b();

void set\_b(double B);

Pair& operator ++(); // Оператор префиксного инкремента

Pair& operator ++(int); // Оператор постфиксного инкремента

Pair& operator = (const Pair& Link); // Оператор присваивания

friend bool operator < (Pair& p, Pair& q); // Оператор сравнения

friend bool operator > (Pair& p, Pair& q); // Оператор сравнения

friend istream& operator >> (istream& in, Pair& Pointer); // Оператор ввода

friend ostream& operator << (ostream& out, const Pair& Pointer); // Оператор вывода

Pair operator +(const Pair& p) const;

};

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

#include "Pair.h"

Pair::Pair()

{

a = 0;

b = 0.0;

}

Pair::Pair(int A, double B)

{

a = A;

b = B;

}

Pair::Pair(const Pair& p)

{

a = p.a;

b = p.b;

}

Pair::~Pair()

{

}

int Pair::get\_a()

{

return a;

}

void Pair::set\_a(int A)

{

a = A;

}

double Pair::get\_b()

{

return b;

}

void Pair::set\_b(double B)

{

b = B;

}

Pair& Pair::operator ++() // Оператор префиксного инкремента

{

++a;

return \*this;

}

Pair& Pair::operator ++(int) // Оператор постфиксного инкремента

{

b++;

return \*this;

}

Pair& Pair::operator = (const Pair& Link) // Оператор присваивания

{

if (&Link != this)

{

a = Link.a;

b = Link.b;

}

return \*this;

}

Pair Pair::operator+(const Pair& p) const

{

return Pair(this->a + p.a, this->b + p.b);

}

bool operator < (Pair& p, Pair& q) // Оператор сравнения

{

if (p.a < q.a)

{

return true;

}

else

{

if (p.a > q.a)

{

return false;

}

else

{

if (p.b < q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator > (Pair& p, Pair& q) // Оператор сравнения

{

if (p.a > q.a)

{

return true;

}

else

{

if (p.a < q.a)

{

return false;

}

else

{

if (p.b > q.b)

{

return true;

}

else

{

return false;

}

}

}

}

istream& operator >> (istream& in, Pair& Pointer) // Оператор ввода

{

cout << "Введите a: ";

in >> Pointer.a;

cout << "Введите b: ";

in >> Pointer.b;

return in;

}

ostream& operator << (ostream& out, const Pair& Pointer) // Оператор вывода

{

cout << "a:b ";

return (out << Pointer.a << " : " << Pointer.b);

}

**Файл с главной программой main.cpp**

#include <iostream>

#include "List.h"

#include "Pair.h"

using namespace std;

int main() {

setlocale(0, "");

Pair p1(2, 2.5);

List<Pair> a(3, p1);

cout << a << endl;

cin >> a;

cout << a << endl;

List<Pair> b(a);

cout << b << endl;

List<Pair> c = a + b;

cout << "a + b\n" << c << endl;

for (Iterator<Pair> iter = b.first(); iter != b.last(); ++iter)

cout << \*iter << '\n';

return 0;

}

**Ответы на вопросы**

1. Адаптировать класс под использование любых типов данных
2. Ключевое слово template
3. Ключевое слово template
4. Поля, принимающие значения любого типа
5. Не могут быть виртуальными, могут быть дружественными и наследуемыми
6. Template <typename T>
7. Да
8. Могут содержать дружественные, но не виртуальные функции. Код прописывается во время компиляции
9. Нет
10. Да
11. Нет
12. Ключевое слово template

13. Генерация кода по определенному типу данных

14. При компиляции