Пермский национальный исследовательский политехнический университет.

Лабораторная работа № 7 по ООП.

«Шаблоны классов».

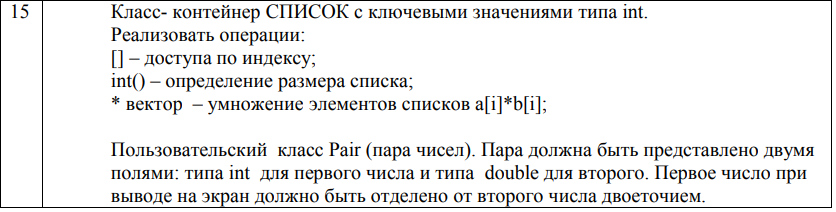
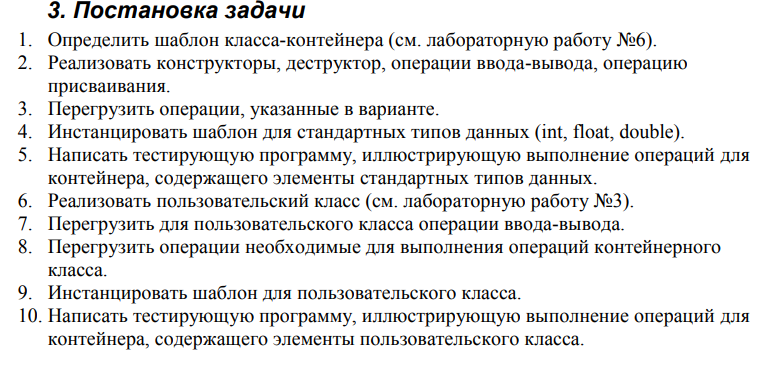
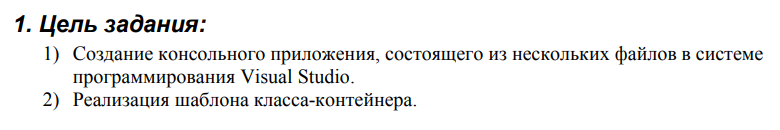
Выполнил: студент группы РИС-23-2б

Вековшинин Иван Николаевич

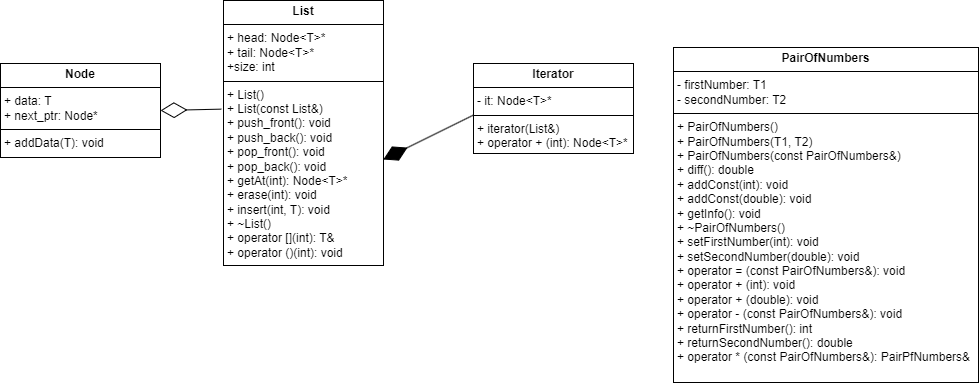
Проверила: доцент кафедры ИТАС

О.А. Полякова.

2024 г.

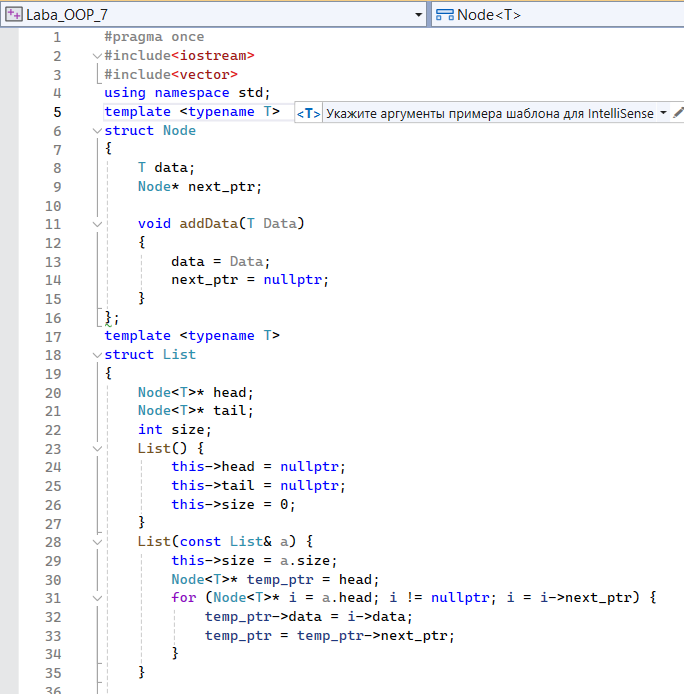


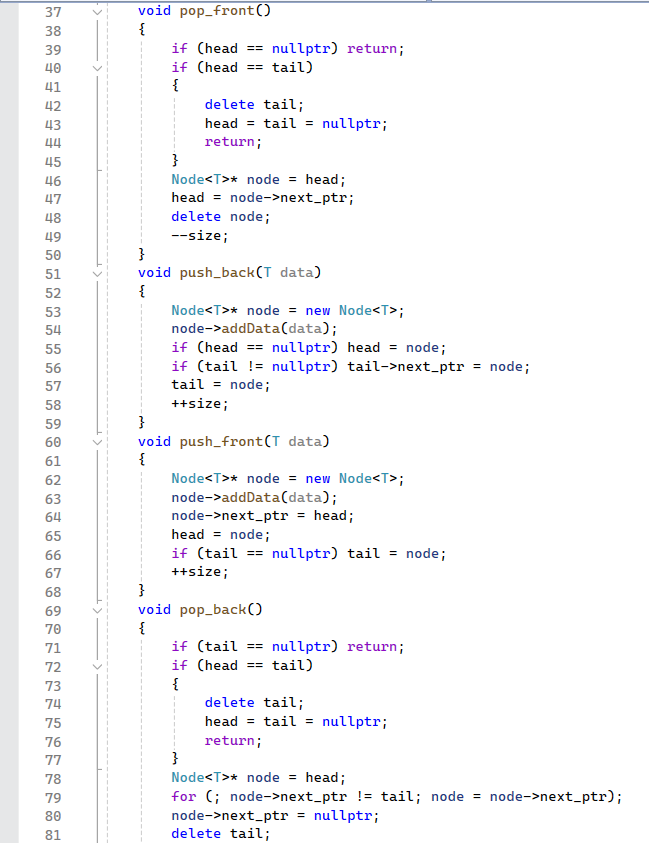
UML Diagram

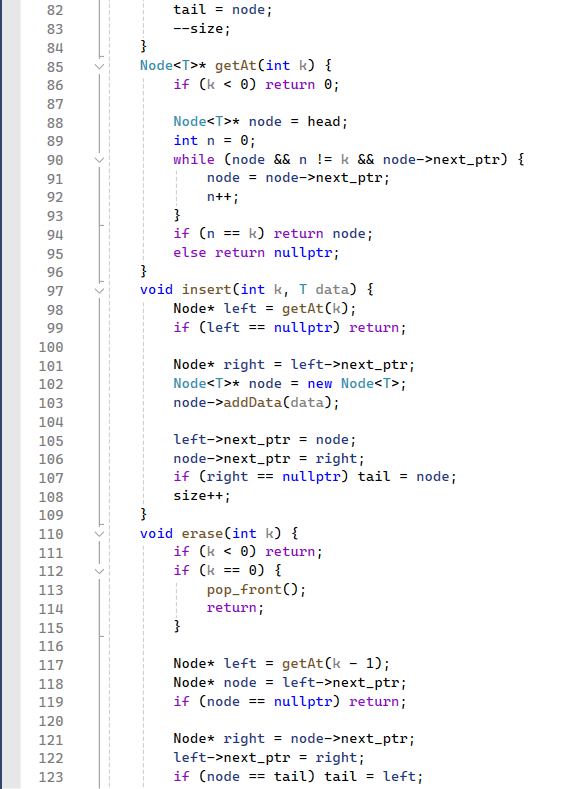


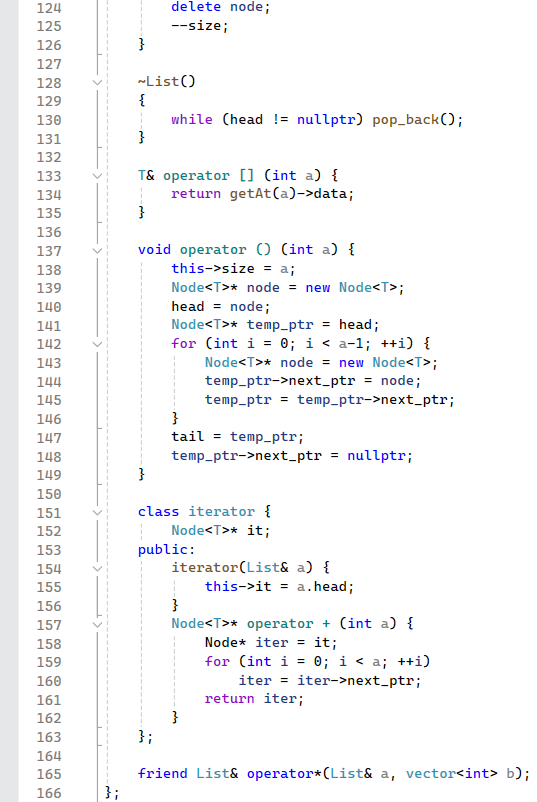
Код:

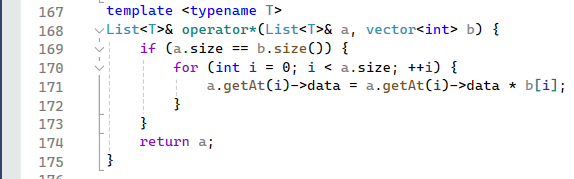
Класс контейнер











Пользовательский шаблонный класс Pair

#pragma once

#include<iostream>

using namespace std;

template <typename T1, typename T2>

class PairOfNumbers {

T1 firstNumber;

T2 secondNumber;

public:

PairOfNumbers();

PairOfNumbers(T1, T2);

PairOfNumbers(const PairOfNumbers&);

double diff();

void addConst(int);

void addConst(double);

void getInfo();

~PairOfNumbers();

void setFirstNumber(int);

void setSecondNumber(double);

void operator = (const PairOfNumbers&);

void operator + (int);

void operator + (double);

void operator - (const PairOfNumbers&);

int returnFirstNumber();

double returnSecondNumber();

private:

T1 FirstNumber; //для запоминания изначальных значений

T2 SecondNumber;

};

template <typename T1, typename T2>

PairOfNumbers<T1, T2>::PairOfNumbers() {

cout << "Вызвался конструктор по умолчанию" << '\t' << this << endl;

this->firstNumber = 0;

this->secondNumber = 0;

this->FirstNumber = 0;

this-> SecondNumber = 0;

}

template <typename T1, typename T2>

PairOfNumbers<T1, T2>::PairOfNumbers(T1 first, T2 second) {

cout << "Вызвался конструктор с параметарми" << '\t' << this << endl;

this->firstNumber = first;

this->secondNumber = second;

this->FirstNumber = first;

this->SecondNumber = second;

}

template <typename T1, typename T2>

PairOfNumbers<T1, T2>::PairOfNumbers(const PairOfNumbers& a) {

cout << "Вызвался конструктор копирования" << '\t' << this << endl;

this->firstNumber = a.FirstNumber;

this->secondNumber = a.SecondNumber;

}

template <typename T1, typename T2>

double PairOfNumbers<T1, T2>::diff() { //вычитание

return this->firstNumber - this->secondNumber;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>::addConst(int number) { //добавление константы к int

this->firstNumber += number;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>::addConst(double number) { //добавление константы к double

this->secondNumber += number;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>::getInfo() { //вывод на экран

cout << this->firstNumber << " : " << this->secondNumber;

}

template <typename T1, typename T2>

PairOfNumbers<T1, T2>::~PairOfNumbers() {

cout << "Вызвался деструктор" << '\t' << this << endl;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>::setFirstNumber(int a) { //инициализация первого числа

this->firstNumber = a;

this->FirstNumber = a;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>::setSecondNumber(double a) { //инициализация второго числа

this->secondNumber = a;

this->SecondNumber = a;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>:: operator = (const PairOfNumbers& a) { //перегразука оператора =

cout << "Вызвался оператор = " << endl;

this->firstNumber = a.FirstNumber;

this->secondNumber = a.SecondNumber;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>:: operator + (int x) { //перегразука оператора +

cout << "Вызвался оператор + " << endl;

firstNumber += x;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>:: operator + (double x) { //перегразука оператора +

cout << "Вызвался оператор + " << endl;

secondNumber += x;

}

template <typename T1, typename T2>

void PairOfNumbers<T1, T2>:: operator - (const PairOfNumbers& a) { //перегразука оператора -

firstNumber -= a.firstNumber;

secondNumber -= a.secondNumber;

}

template <typename T1, typename T2>

int PairOfNumbers<T1, T2>::returnFirstNumber() { //получение значение первого числа

return firstNumber;

}

template <typename T1, typename T2>

double PairOfNumbers<T1, T2>::returnSecondNumber() { //получение значение второго числа

return secondNumber;

}

template <typename T1, typename T2>

ostream& operator<<(ostream& os, PairOfNumbers<T1, T2>& b) //перегрузка вывода

{

cout << "Вызвалась перегрузка оператора <<" << endl;

os << b.returnFirstNumber() << " : " << b.returnSecondNumber();

return os;

}

template <typename T1, typename T2>

istream& operator >> (istream& in, PairOfNumbers<T1, T2>& c) //перегрузка ввода

{

cout << "Вызвалась перегрузка оператора >>" << endl;

int first; double second;

in >> first >> second;

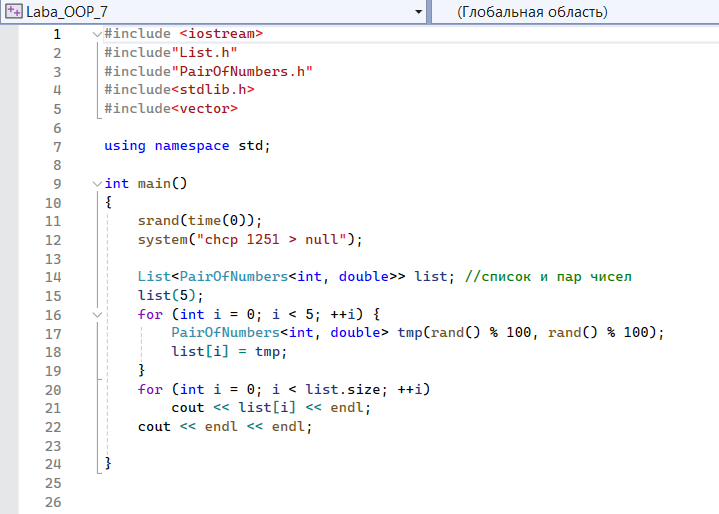
c.setFirstNumber(first);

c.setSecondNumber(second);

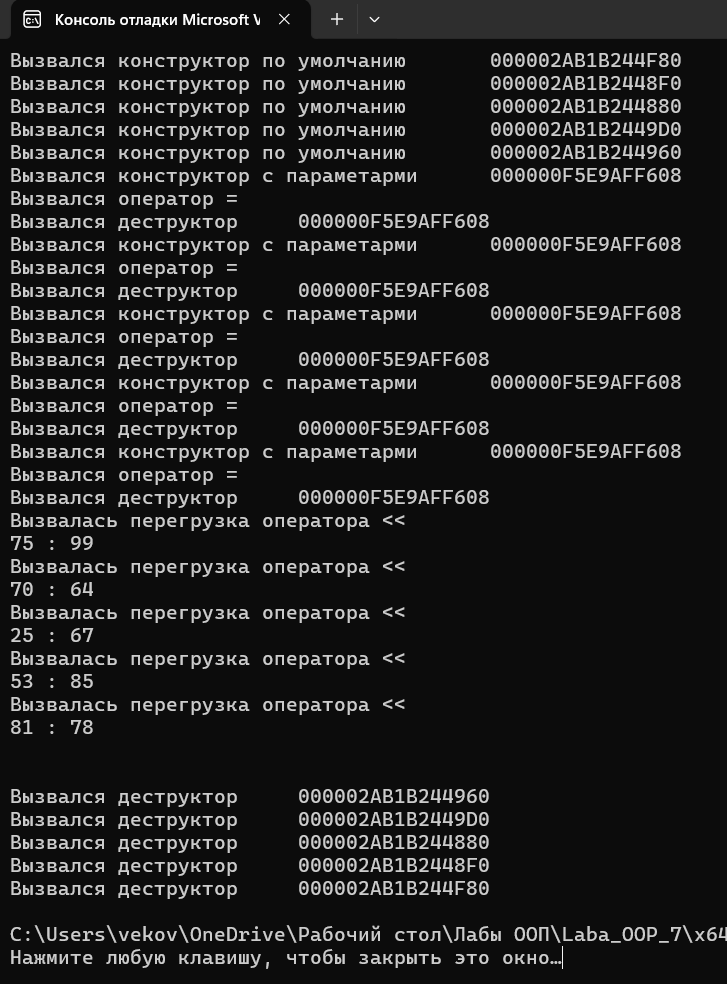
return in;

}

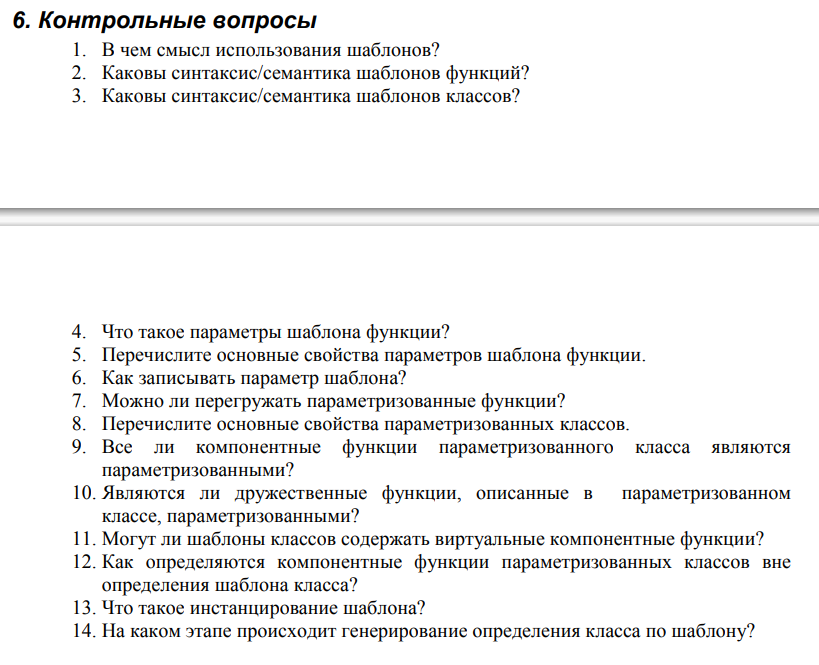
Функция main



Работа программы:



Ответы на вопросы:



1. Шаблоны позволяют сделать программу более универсальной, для использования класса(или методов, или функций) с переменными разных типов, при этом не прописывая каждый раз функцию для работы с новым типом данных. Компилятор сам определит с каким типом данных надо работать.
2. Ключевое слово template, указание в угловых скобках ключевых слов class (или template) для обозначения шаблонной переменной. Пример: template<typename T>.
3. Шаблоны классов в C++ позволяют создавать универсальные классы, которые могут работать с различными типами данных, обеспечивая гибкость и повторное использование кода. Синтаксис описания шаблона аналогичен описанию шаблона для функции. Но для использования шаблонного класса нужно указывать заранее тип данных в угловых скобках. Например: vector<int>, в угловых скобках указали, что работаем с целыми числами.
4. Параметр шаблона представляет произвольный идентификатор.
5. Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.   
   Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.   
   Если friend-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону класса имеется собственная friend-функция.   
   В рамках параметризованного класса нельзя определить friend-шаблоны (дружественные параметризованные классы).  
   Шаблоны функций, которые являются членами классов, нельзя описывать как virtual.   
   Локальные классы не могут содержать шаблоны в качестве своих элементов.
6. В C++ параметры шаблона записываются с помощью угловых скобок < >.
7. Параметризированные функции можно перегружать.
8. Одним из основных преимуществ параметризированных классов в C++ является общий подход к обработке данных различных типов, что повышает гибкость и эффективность программирования.
9. Не все компонетные функции шаблонного класса параметризированные.
10. Дружественные функции описанные в параметризированном классе не являются параметризированными.
11. Не могут содержать виртуальные компонентные функции
12. Тип\_возвращаемых данных Название\_класса<параметризированный\_тип>::название\_функции(тип параметры){};
13. Каждый раз, когда параметризированный класс используется с конкретным типом данных, происходит процесс инстанцирования, при котором компилятор создает версию класса для этого конкретного типа данных.
14. На этапе компиляции.