Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа №7**

**«Шаблоны классов»**

**14 вариант**

Выполнил:

Студент группы РИС-23-1б

Шароглазов Егор Алексеевич

Проверила:

Доцент кафедры ИТАС

О.А. Полякова

1. Г

Постановка задачи:

1. Определить шаблон класса-контейнера.

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.

7. Перегрузить для пользовательского класса операции ввода-вывода. 8. Перегрузить операции необходимые для выполнения операций контейнерного класса.

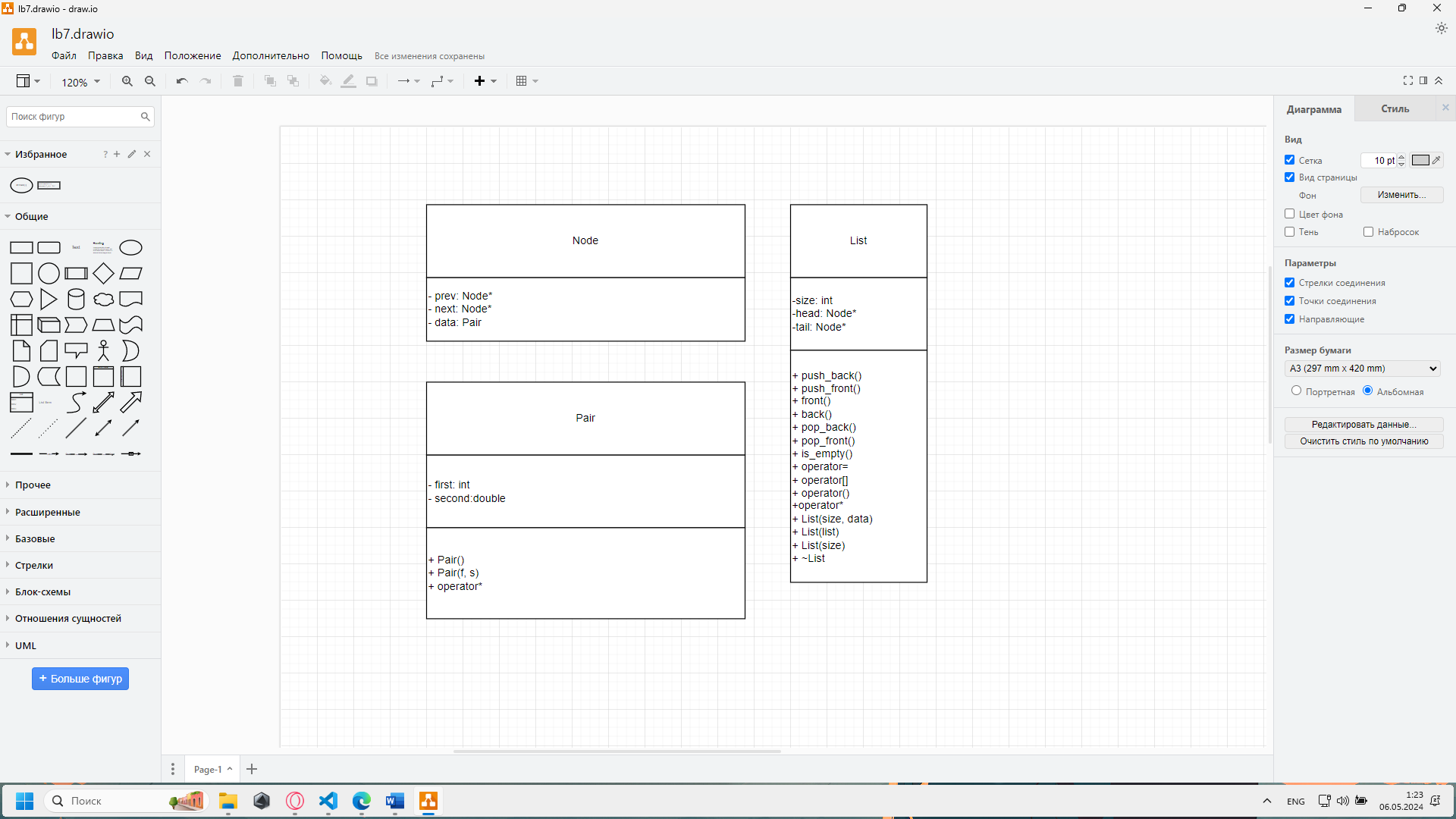
9. Инстанцировать шаблон для пользовательского класса.

10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса

**Анализ задачи:**

1. **Определение шаблонного класса-контейнера:**Создать шаблонный класс Container с параметром типа T.  
   Определить структуру узла Node с данными типа T и указателями на следующий/предыдущий узел.  
   **2. Реализация конструкторов, деструктора, операций ввода-вывода, присваивания:**Конструкторы:  
   Конструктор по умолчанию (пустой контейнер).  
   Конструктор с размером и значением по умолчанию для заполнения контейнера.  
   Конструктор копирования.  
   Деструктор: Освобождение памяти, занимаемой узлами.  
   Операции ввода-вывода:  
   Перегрузка операторов « и » для вывода и ввода элементов контейнера (шаблоны функций).  
   Оператор присваивания:  
   Перегрузка оператора = для копирования одного контейнера в другой.  
   **3. Перегрузка операторов:**Реализовать перегрузку операторов, указанных в варианте задания (шаблоны функций).  
   **4. Инстанцирование шаблона для стандартных типов данных**  
   **5. Тестирующая программа для стандартных типов данных:**Выполнить различные операции с контейнерами стандартных типов:  
   Добавление/удаление элементов  
   Доступ к элементам по индексу  
   Использование перегруженных операторов  
   Вывести результаты операций на консоль.  
   **6. Реализация пользовательского класса:**Создать класс с необходимыми полями и методами.  
   Перегрузить операторы « и » для ввода-вывода объектов.  
   **7-8. Перегрузка операций для пользовательского класса:**Перегрузить операторы, необходимые для работы с контейнером (например, операторы сравнения).  
   **9. Инстанцирование шаблона для пользовательского класса**  
   10. Тестирующая программа для пользовательского класса:  
   Выполнить операции с контейнером объектов UserClass.  
   Вывести результаты на консоль.

**UML – Диаграмма**



**Код**

#include <iostream>

#include <istream>

#include <vector>

using namespace std;

class Pair {

public:

    int first;

    double second;

private:

    friend class List;

public:

    Pair() : first(0), second(0.0) {}

    Pair(int f, double s) : first(f), second(s) {}

    Pair operator\*(const Pair& p) const {

    Pair result;

    result.first = this->first \* p.first;

    result.second = this->second \* p.second;

    return result;

}

friend ostream& operator<<(ostream& out, const Pair& p) {

        out << p.first << " : " << p.second;

        return out;

    }

};

struct Node {

public:

    Pair data;

    Node\* next;

    Node\* prev;

};

class List {

private:

    int size;

    Node\* head;

    Node\* tail;

public:

    void push\_back(const Pair& data){

        Node\* new\_node = new Node;

        new\_node->data=data;

        new\_node->next=nullptr;

        if(this->head==nullptr)

        {

            this->head=new\_node;

            this->tail=new\_node;

        }

        else

        {

            tail->next = new\_node;

            new\_node->prev = tail;

            tail=new\_node;

        }

        this->size++;

    }

    void push\_front(const Pair& data){

        Node\* new\_node = new Node;

        new\_node->data=data;

        if(this->head==nullptr)

        {

            this->head=new\_node;

            this->tail=new\_node;

            this->size++;

        }

        else

        {

            head->prev = new\_node;

            new\_node->next = head;

            head=new\_node;

            this->size++;

        }

    }

    Pair front(){

        return this->head->data;

    }

    Pair back(){

        return this->tail->data;

    }

    Pair pop\_back(){

    Pair temp;

    if(this->tail != nullptr) {

        Node\* current\_node = this->tail;

        tail = current\_node->prev;

        temp = current\_node->data;

        tail->next = nullptr;

        delete current\_node;

        this->size--;

    }

    return temp;

}

    Pair pop\_front(){

        Pair temp;

        if(this->head!=nullptr){

            Node \* current\_node= this->head;

            head=current\_node->next;

            temp=current\_node->data;

            head->prev=nullptr;

            this->size--;

        }

        return temp;

    }

    bool is\_empty(){

        return this->size==0;

    }

    List& operator = (const List& list){

        cout<<"Operator ="<<endl;

        if(this == &list){

            return \*this;

        }

        while (head!=nullptr){

            Node\* temp=head;

            head=head->next;

            delete temp;

        }

        size =0;

        Node\* current\_node=list.head;

        while(current\_node!=nullptr){

            push\_back(current\_node->data);

            current\_node=current\_node->next;

        }

        return \*this;

    }

    Pair& operator [](int index){

        if(index <this->size&&index>=0){

            Node\* current\_node = this->head;

            for(int i=0;i!=index;i++){

                current\_node=current\_node->next;

            }

            return current\_node->data;

        }

        else{

            cerr<<"index out of range";

            exit(0);

        }

    }

    int operator () (){

        return this->size;

    }

    List operator \* (List& list){

        int temp\_size;

        if(this->size>list.size){

            temp\_size=list.size;

        }

        else

        {

            temp\_size=this->size;

        }

        List temp(temp\_size, Pair(0, 0));

        for(int i=0; i<temp\_size; i++){

            temp[i]=(\*this)[i]\*list[i];

        }

        return temp;

    }

    List(int size, const Pair& data) {

    this->size = size;

    if (size > 0) {

        head = new Node;

        head->data = data;

        tail = head;

        for (int i = 1; i < size; ++i) {

            Node\* newNode = new Node;

            newNode->data = data;

            tail->next = newNode;

            newNode->prev = tail;

            tail = newNode;

        }

        tail->next = nullptr;

    } else {

        head = nullptr;

        tail = nullptr;

    }

}

    List(const List& list)

    {

        this->head=nullptr;

        this->tail=nullptr;

        this->size=0;

        Node\* current\_node=list.head;

        while(current\_node!=nullptr){

            push\_back(current\_node->data);

            current\_node=current\_node->next;

        }

    };

    List(int size){

        this->size=size;

        if(size>0){

            Node\* node = new Node;

            this->head=node;

            this->tail=node;

            for(int i=1; i<size; i++){

                Node\* newNode = new Node;

                tail->next = newNode;

                newNode->prev = tail;

                tail=newNode;

            }

            tail->next=nullptr;

        }

        else

        {

            this->head=nullptr;

            this->tail=nullptr;

        }

    };

    ~List()

    {

        Node\* current\_node=head;

        while(current\_node!=nullptr){

            Node\* next=current\_node->next;

            delete current\_node;

            current\_node=next;

            }

        head=nullptr;

    };

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

        out << "Элементы списка: ";

        Node\* current = list.head;

        while (current != nullptr) {

            out << current->data << " ";

            current = current->next;

        }

        return out;

    }

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

        Node\* current = list.head;

        cout << "Введите элементы списка: ";

        while (current != nullptr) {

            in >> current->data.first >> current->data.second;

            current = current->next;

        }

        return in;

    }

};

int main() {

    Pair pair1(1, 2.5);

    Pair pair2(3, 2.7);

    List list1(2, pair1);

    cout<<list1<<endl;

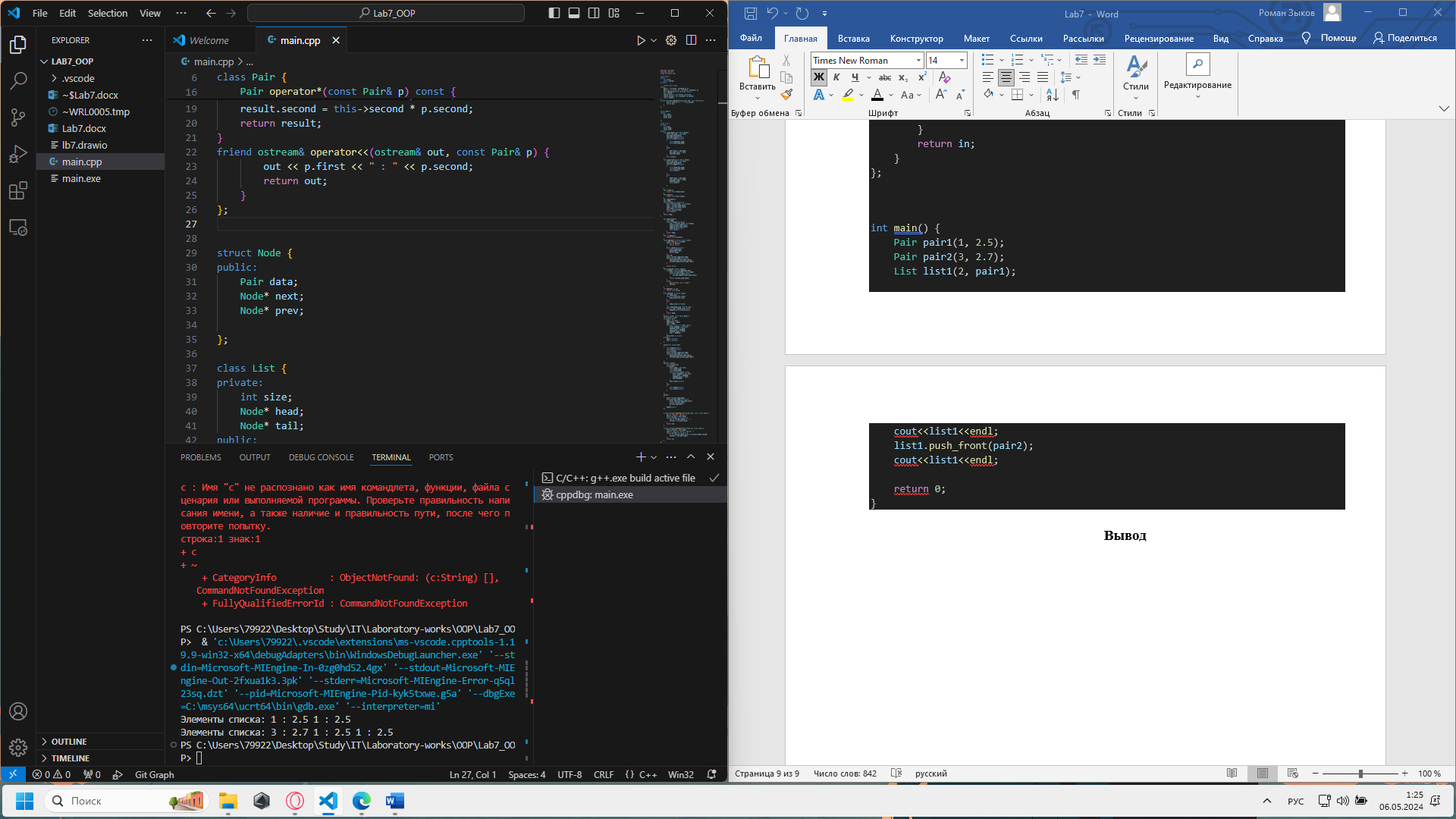
    list1.push\_front(pair2);

    cout<<list1<<endl;

    return 0;

}

**Вывод**



**Вопросы**

· **Смысл использования шаблонов:** Шаблоны позволяют писать обобщенный код, который может работать с различными типами данных, без необходимости дублирования кода для каждого конкретного типа. Это увеличивает гибкость, повышает переиспользование кода и облегчает разработку, так как позволяет писать универсальный код, который может быть применен к разным типам данных.

· **Синтаксис/семантика шаблонов функций:**

* **Синтаксис:** Объявление шаблона функции начинается с ключевого слова template, за которым следует список параметров шаблона в угловых скобках < >, а затем обычное объявление функции.
* **Семантика:** Шаблон функции позволяет определить функцию, которая может работать с различными типами данных. Параметры шаблона функции указываются в угловых скобках и могут быть использованы в теле функции для описания алгоритма.

· **Синтаксис/семантика шаблонов классов:**

* **Синтаксис:** Объявление шаблона класса аналогично шаблону функции, но с ключевым словом template перед объявлением класса.
* **Семантика:** Шаблон класса позволяет определить обобщенный тип данных, который может работать с различными типами данных внутри класса. Параметры шаблона класса могут использоваться для определения типов членов класса, методов и конструкторов.

· **Параметры шаблона функции:** Параметры шаблона функции - это обобщенные типы данных или значения, которые передаются в шаблон функции во время его использования для конкретизации типов данных или значений внутри функции.

· **Основные свойства параметров шаблона функции:**

* Могут быть типами данных или значениями.
* Могут быть константными или неконстантными.
* Могут быть обязательными или необязательными.
* Могут быть одиночными или множественными.

· **Запись параметра шаблона:** Параметры шаблона функции записываются в угловых скобках < > после ключевого слова template и могут быть любыми допустимыми идентификаторами. Например:

· template <typename T>

void myFunction(T parameter) {

// код функции

}

· **Перегрузка параметризованных функций:** Да, шаблонные функции могут быть перегружены. Можно иметь несколько шаблонных функций с различными параметрами шаблона или с различным числом параметров.

· **Основные свойства параметризованных классов:**

* Могут содержать шаблонные методы и/или шаблонные данные.
* Могут быть параметризованы типами данных или значениями.
* Могут быть полностью или частично специализированы.

· **Все ли компонентные функции параметризованного класса являются параметризованными:** Нет, не все. В зависимости от реализации, часть функций может быть обычными, не зависящими от параметров шаблона.

· **Дружественные функции в параметризованных классах:** Да, дружественные функции, описанные в параметризованных классах, также являются параметризованными и могут использовать параметры шаблона.

· **Виртуальные компонентные функции в шаблонах классов:** Да, шаблоны классов могут содержать виртуальные компонентные функции.

· **Определение компонентных функций параметризованных классов вне определения шаблона класса:** Компонентные функции параметризованных классов могут быть определены как внутри, так и вне определения шаблона класса. Определение функций за пределами шаблона класса может быть полезно для улучшения читаемости кода и разделения интерфейса и реализации.

· **Инстанцирование шаблона:** Инстанцирование шаблона - это процесс создания конкретной реализации шаблона функции или класса для определенного набора параметров.

· **Генерирование определения класса по шаблону:** Генерация определения класса по шаблону происходит во время компиляции, когда шаблон используется с конкретными параметрами. Компилятор создает экземпляр класса, заменяя параметры шаблона на конкретные типы данных или значения.