Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№7»**

Выполнил:

студент группы РИС-23-1б

Кривошеин Александр Антонович

Проверила:

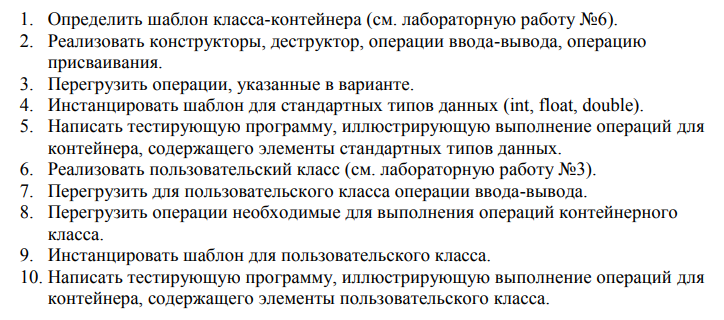
доцент кафедры ИТАС

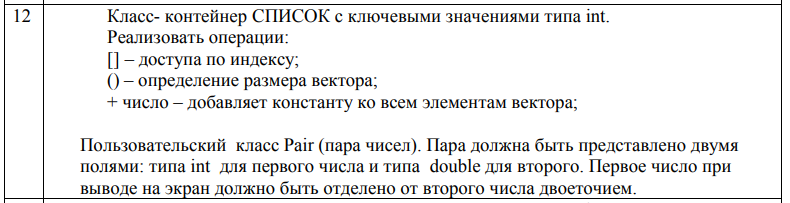
Полякова Ольга Андреевна

2024 г.

**Разработка алгоритма**

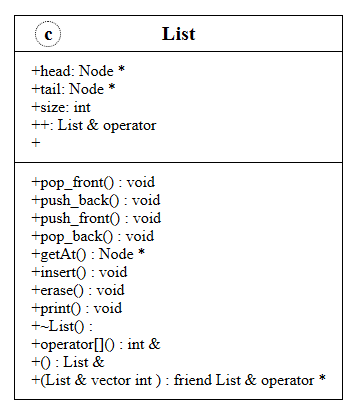
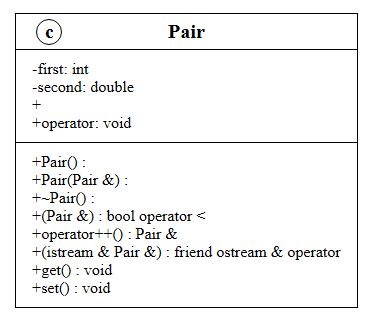
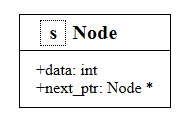
**Постановка задачи:**

****

****

**Анализ задачи:**

1. Создадим несколько заголовочных файлов и файлом с кодом для удобной работы.
2. Используем готовую реализацию класса-контейнера списка из лабораторной работы №6.
3. Также используем класс Pair из лабораторной работы №3.
4. На основе лабораторной работы №6 создадим шаблонный класс List, который позволит создавать список элементов произвольного типа данных.
5. Перегрузим операции [] доступа по индексу, () – определении размера и + – для добавления константы к элементам списка.



Код на С++:

main.cpp:

#include <iostream>

using namespace std;

#include "list.h"

#include "pair.h"

int main()

{

    // system("chcp 1251>null");

    List<int> list1(5);

    List<float> list2(5);

    List<double> list3(5);

    List<Pair> list4(3);

    List<int> list5(5);

    cout << "list1 int\n";

    cin >> list1;

    cout << list1;

    cout << "\nThe output of the element 1: ";

    cout << list1[0] << "\n";

    cout << "\nlist2 float\n";

    cin >> list2;

    cout << list2;

    cout << "\nThe output of the element 3: ";

    cout << list2[2] << "\n";

    cout << "\nlist3 double\n";

    cin >> list3;

    cout << list3 << "\n";

    cout << "\nlist4 Pair\n";

    cin >> list4;

    cout << list4;

    cout << "\nThe output of the element 2:\n";

    cout << list4[1] << "\n";

    cout << "\nlist5 = list1\n";

    list5 = list1;

    cout << list5 << "\n";

    return 0;

}

pair.h:

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

    int first;

    double second;

public:

    Pair();

    Pair(int, double);

    Pair(Pair &);

    ~Pair(){};

    bool operator<(Pair &);

    bool operator>(Pair &);

    Pair &operator++();

    Pair &operator++(int);

    friend ostream &operator<<(ostream &out, Pair i);

    friend istream &operator>>(istream &stream, Pair &pair);

    void operator=(Pair &);

    void get();

    void set(int, double);

};

Pair::Pair()

{

    this->first = 0;

    this->second = 0;

}

Pair::Pair(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

Pair::Pair(Pair &copying)

{

    this->first = copying.first;

    this->second = copying.second;

}

bool Pair::operator<(Pair &temp)

{

    if ((this->first + this->second) < (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator>(Pair &temp)

{

    if ((this->first + this->second) > (temp.first + temp.second))

        return true;

    return false;

}

Pair &Pair::operator++()

{

    ++this->first;

    return \*this;

}

Pair &Pair::operator++(int)

{

    ++this->second;

    return \*this;

}

void Pair::get()

{

    std::cout << "first = " << this->first << ", second = " << this->second;

}

void Pair::set(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

std::ostream &operator<<(std::ostream &pairout, Pair temp)

{

    return (pairout << temp.first << ':' << temp.second);

}

istream &operator>>(istream &stream, Pair &pair)

{

    int first;

    double second;

    stream >> first >> second;

    pair.first = first;

    pair.second = second;

    return stream;

}

void Pair::operator=(Pair &other)

{

    this->first = other.first;

    this->second = other.second;

}

list.h:

#pragma once

#include <iostream>

template <typename T>

struct Node

{

    T data;

    Node<T>\* ptr\_to\_next\_node = nullptr;

};

template <typename T>

class List

{

private:

    int size;

    Node<T>\* head\_node;

    Node<T>\* tail\_node;

public:

    List(int);

    List(List<T>&);

    ~List();

    void push\_back(T);

    void pop\_front();

    int operator () ();

    T& operator[](int);

    List& operator = (List<T>&);

    List operator \*(List<T>&);

    friend std::ostream& operator << (std::ostream& stream, List<T>& list)

    {

        std::cout << "Output of elements: \n";

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

            stream << current\_node->data << ' ';

            current\_node = current\_node->ptr\_to\_next\_node;

        }

        return stream;

    }

    friend std::istream& operator >> (std::istream& stream, List<T>& list)

    {

        std::cout << "Enter elements: ";

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

            stream >> current\_node->data;

            current\_node = current\_node->ptr\_to\_next\_node;

        }

        return stream;

    }

};

template <typename T>

List<T>::List(int size)

{

    if (size > 0)

    {

        Node<T>\* node = new Node<T>;

        this->head\_node = node;

        this->tail\_node = node;

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

        {

            Node<T>\* New\_Node = new Node<T>;

            tail\_node->ptr\_to\_next\_node = New\_Node;

            tail\_node = New\_Node;

        }

        this->size = size;

    }

    else

    {

        this->head\_node = nullptr;

        this->tail\_node = nullptr;

    }

}

template <typename T>

List<T>::List(List& list)

{

    this->size = 0;

    Node<T>\* current\_node = list.head\_node;

    while (current\_node != nullptr)

    {

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

        current\_node = current\_node->ptr\_to\_next\_node;

    }

}

template <typename T>

List<T>::~List()

{

    while (head\_node != nullptr)

    {

        this->pop\_front();

    }

}

template <typename T>

void List<T>::push\_back(T data)

{

    Node<T>\* New\_Node = new Node<T>;

    New\_Node->data = data;

    if (this->head\_node == nullptr)

    {

        this->head\_node = New\_Node;

        this->tail\_node = New\_Node;

    }

    else

    {

        this->tail\_node->ptr\_to\_next\_node = New\_Node;

        this->tail\_node = New\_Node;

    }

    this->size++;

}

template <typename T>

void List<T>::pop\_front()

{

    if (this->tail\_node != nullptr)

    {

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

        head\_node = current\_node->ptr\_to\_next\_node;

        this->size--;

    }

}

template <typename T>

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

{

    return this->size;

}

template <typename T>

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

{

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

    for (int i = 0; i != index; i++)

    {

        current\_node = current\_node->ptr\_to\_next\_node;

    }

    return current\_node->data;

}

template <typename T>

List<T>& List<T>::operator = (List<T>& list)

{

    if (this != &list)

    {

        while (head\_node != nullptr)

        {

            this->pop\_front();

        }

        size = 0;

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

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

            current\_node = current\_node->ptr\_to\_next\_node;

        }

    }

    return \*this;

}

template <typename T>

List<T> List<T>::operator \*(List<T>& list)

{

    int temp\_size;

    if (this->size > list.size)

    {

        temp\_size = list.size;

    }

    else

    {

        temp\_size = this->size;

    }

    List<T> temp\_list(temp\_size);

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

    {

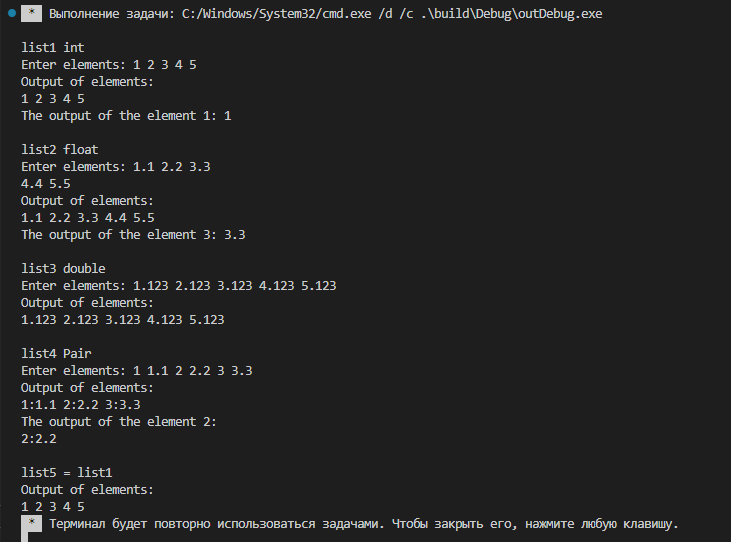
        temp\_list[i] = (\*this)[i] \* list[i];

    }

    return temp\_list;

}

Пример работы программы:

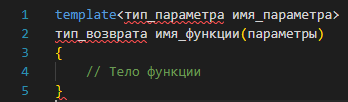


Контрольные вопросы

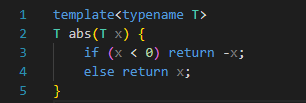
1. В чем смысл использования шаблонов?

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные.

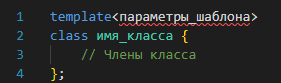
1. Каковы синтаксис/семантика шаблонов функций?

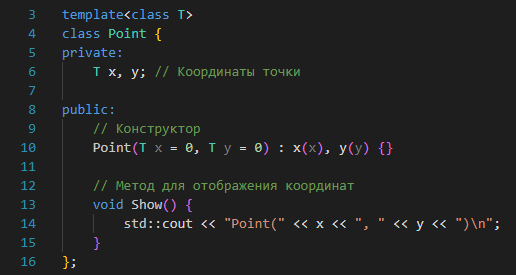


Вот пример обобщенной функции, которая вычисляет абсолютное значение числа:



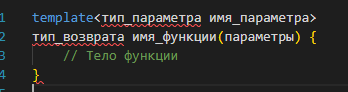
1. Каковы синтаксис/семантика шаблонов классов?

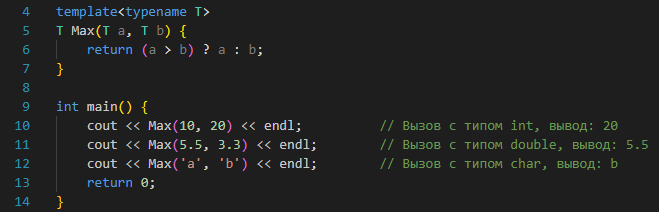




1. Что такое параметры шаблона функции?

Параметры шаблона функции — это типы или значения, которые определяют конкретную версию функции. Они указываются внутри угловых скобок после ключевого слова template перед именем функции.





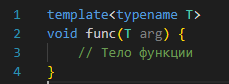
1. Перечислите основные свойства параметров шаблона

функции.

* шаблоны не могут быть виртуальными
* шаблоны могут содержать статические элементы, дружественные функции и классы
* шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

1. Как записывать параметр шаблона?

Параметр шаблона записывается внутри угловых скобок после ключевого слова template, например <int>, <double>, <T>, где T - это псевдоним типа, который будет определен позже.



1. Можно ли перегружать параметризованные функции?

Да, можно перегружать параметризованные функции. Перегрузка

осуществляется так же, как и для обычных функций, с учетом параметров

шаблона.

1. Перечислите основные свойства параметризованных классов.

* Параметры шаблона могут быть как значениями, так и типами.

Например, можно создать шаблон класса с параметром, который задает размер массива.

* Параметры шаблона могут быть любыми типами данных, включая примитивные типы, пользовательские типы и другие шаблоны классов.

-Шаблонные классы могут быть унаследованы от других шаблонных классов и могут иметь шаблонные функции-члены.

-Параметры шаблона могут использоваться внутри класса для объявления переменных, функций и типов.

-Каждый конкретный экземпляр параметризованного класса является отдельным типом, который определяется параметрами шаблона.

1. Все ли компонентные функции параметризованного

Класса являются параметризованными?

Да, все компонентные функции (методы) параметризированного класса

являются параметризованными, то есть они используют те же параметры

шаблона, что и сам класс.

1. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

Да

1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, шаблоны классов могут содержать виртуальные компонентные

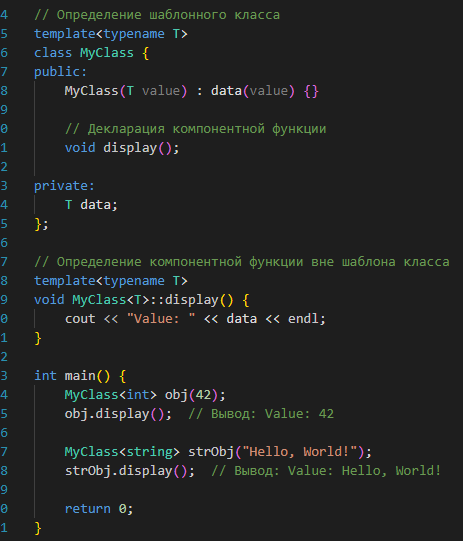
функции. Виртуальность работает так же, как и для обычных классов, с учетом

параметров шаблона.

12. Как определяются компонентные функции

параметризованных классов вне определения шаблона класса?

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса с помощью использования template<class T> перед каждым определением функции, где T - это параметр шаблона.



1. Что такое инстанцирование шаблона?

Инстанцирование шаблона — это процесс генерации конкретной версии шаблона для определенных типов. Оно происходит автоматически, когда компилятор встречает использование шаблона с конкретными типами.

1. На каком этапе происходит генерирование определения класса по шаблону?

Генерирование определения класса по шаблону происходит во время

компиляции, когда компилятор встречает использование шаблона класса с

конкретными типами.