Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: Лабораторная работа №7: «Шаблон классов».

**Вариант 13**

Выполнил студент ИВТ-22-2б:

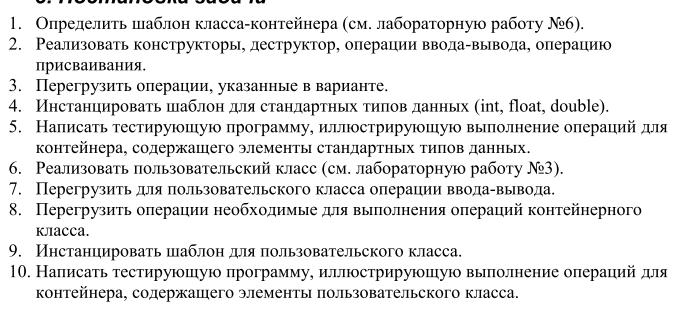
Коняев Александр Сергеевич

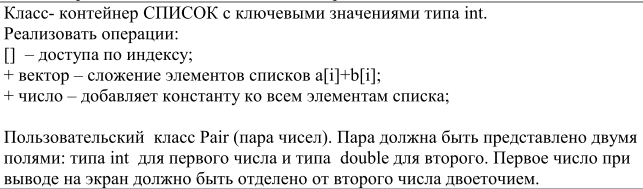
Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**





**Анализ задачи**

Класс Node

Класс LinkedList

* Конструктор с параметрами
* Конструктор без параметров
* Конструктор копирования
* Метод clear
* Метод push\_back
* Метод push\_front
* Метод pop\_back
* Метод pop\_front
* Метод get\_At
* Метод insert
* Метод erase
* Метод print
* Перегруженные операторы: =, [], +, <<, >>

Класс Pair

* Конструктор без параметров
* Конструктор с параметрами
* Конструктор копирования
* Метод get\_first
* Метод set\_first
* Метод get\_second
* Метод set\_second
* Перегруженные операторы: =, <, >, +, --, <<, >>

**UML – диаграмма**

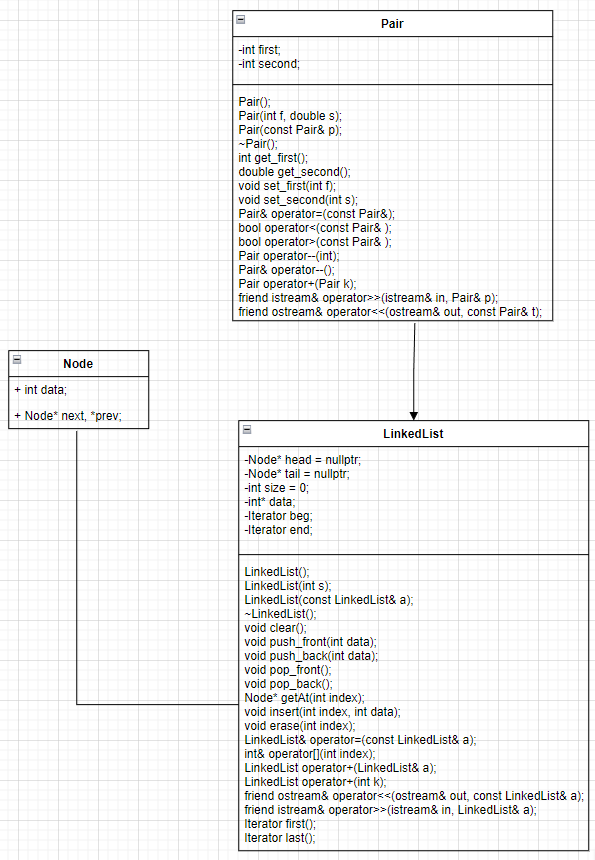


Рисунок 1 – UML-диаграмма.

**Код программы.**

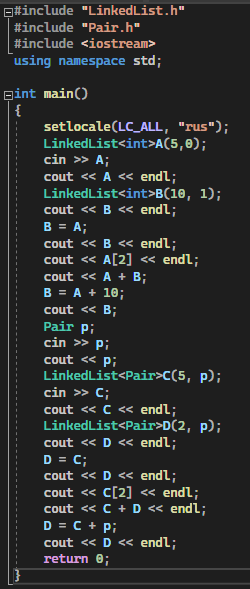


Рисунок 2 – Функция main

**Вывод программы.**

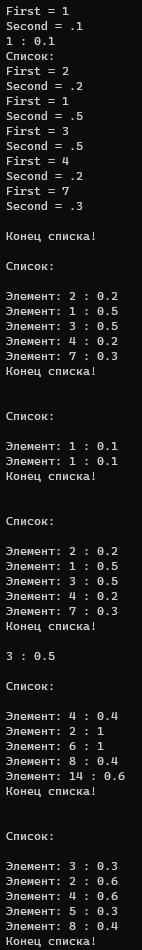
****

Рисунок 3 – Вывод программы

**Ответы на вопросы**

1. Шаблоны в C++ позволяют создавать обобщенные классы и функции, которые могут работать с различными типами данных. Использование шаблонов позволяет сократить количество дублирующегося кода и увеличить переиспользуемость кода. Кроме того, шаблоны позволяют создавать программы, которые могут работать с неизвестными на этапе компиляции типами данных, упрощая процесс разработки и обеспечивая большую гибкость и универсальность. Шаблоны могут применяться как для создания классов, так и для функций, и позволяют создавать контейнеры, алгоритмы и другие средства обработки данных.
2. Шаблоны функций в C++ позволяют создавать обобщенные функции, которые могут работать с различными типами данных. Синтаксис объявления шаблона функции выглядит следующим образом:

template <typename T>

T foo(T arg) {

// реализация функции

}

Где template - ключевое слово для объявления шаблона функции, <typename T> - список параметров шаблона (здесь только один параметр T), который указывает тип данных, используемых в функции, а T - имя типа данных, указанного в списке параметров. foo - имя функции, которая будет использовать шаблон, а arg - параметр функции.

1. Шаблоны классов в C++ позволяют создавать обобщенные классы, которые могут работать с различными типами данных. Синтаксис объявления шаблона класса выглядит следующим образом:

template class MyClass { public: T getValue(); void setValue(T value); private: T value; };

Где template - ключевое слово для объявления шаблона класса, - список параметров шаблона (здесь только один параметр T), который указывает тип данных, используемых в классе, а T - имя типа данных, указанного в списке параметров. MyClass - имя класса, который будет использовать шаблон, а T - параметр класса.

1. Параметры шаблона функции в C++ - это специальные параметры, которые используются в объявлении шаблона функции и определяют типы данных, с которыми данная функция может работать. Это позволяет создавать обобщенные функции, которые могут принимать различные типы данных в качестве аргументов и возвращать значения различных типов.

Синтаксис объявления шаблона функции в C++ выглядит следующим образом:

template <typename T>

T myFunction(T argument);

Здесь template - ключевое слово для объявления шаблона функции, <typename T> - список параметров шаблона (здесь только один параметр T), который указывает тип данных, используемых в функции, а T - имя типа данных, указанного в списке параметров. myFunction - имя функции, которая будет использовать шаблон, а T - параметр функции.

1. Основные свойства параметров шаблона функции в C++:

Параметры шаблона указываются в угловых скобках < > после ключевого слова template.

Параметры шаблона могут быть любого типа данных (в том числе и пользовательского).

Название параметра шаблона можно выбирать произвольно.

Параметры шаблона могут использоваться в объявлении аргументов функции и в ее возвращаемом значении для определения типа данных.

Шаблон функции позволяет создавать обобщенные функции, которые работают с различными типами данных.

Параметры шаблона могут использоваться для задания ограничений на типы данных, с которыми шаблонная функция может работать.

Параметры шаблона могут быть также использованы внутри функции для создания объектов типа параметра шаблона.

1. Параметры шаблона записываются в угловых скобках "< >" после ключевого слова "template". Например, если мы хотим создать шаблон функции, принимающей один параметр шаблона типа "T", то запись в коде будет выглядеть примерно так:

template<typename T>

void myFunction(T parameter) {

// код функции

}

Здесь мы выбрали общепринятый способ записи параметра шаблона "typename T", но вместо "T" можно использовать любое другое имя. Главное, чтобы оно было осмысленным и не конфликтовало с именами других переменных и типов данных в коде.

1. Да, можно перегружать параметризованные функции. Перегрузка функций позволяет определять несколько функций с одним и тем же именем, но различающихся параметрами. Параметризованные функции могут иметь шаблонные параметры, которые могут использоваться в качестве аргументов при перегрузке функций.

Например, вот пример перегруженных функций с одним параметром шаблона T:

template<typename T>

void foo(T t) {

// some code

}

template<typename T>

void foo(T\* ptr) {

// some other code, for pointer types

}

int main() {

int i = 42;

foo(i); // calls foo<int>(int)

foo(&i); // calls foo<int>(int\*)

return 0;

}

1. Основные свойства параметризованных классов в C++:

Параметризация - возможность задавать типы в качестве параметров при создании экземпляров класса.

Можно использовать любой тип данных, включая пользовательские типы данных.

Однажды определенный параметризованный класс может быть использован многократно с различными параметрами типов при создании экземпляров.

Можно использовать параметры типа для создания шаблонов классов и функций.

Параметризованные классы позволяют упрощать и обобщать код, уменьшая необходимость создания отдельных классов для каждого типа данных.

1. Не все компонентные функции параметризованного класса должны быть параметризованными. В зависим от потребности в каждом конкретном случае, какие-то члены класса могут использовать параметры типа, а какие-то нет. Например, параметризованный класс может иметь функцию, которая не зависит от параметра типа, и функция может напрямую зависеть от параметра типа и должна быть параметризованной. Важно помнить, что параметризация используется для универсализации кода, поэтому необходимо рационально и грамотно принимать решения о том, какие части класса будут параметризованными, а какие нет.
2. Да, дружественные функции, описанные в параметризованном классе, могут быть параметризованными. В зависимости от того, нужны ли они для работы с параметром типа, можно решить, будут ли они параметризованными или нет. Если дружественная функция зависит от типа параметра класса, то она должна быть параметризованной. В противном случае – нет. Пример кода для дружественной функции с параметром типа:

template <typename T>

class MyClass {

friend void myFriendFunction(const T& obj);

// ...

};

template <typename T>

void myFriendFunction(const T& obj) {

// Функция, которая работает с объектом типа T

}

1. Да, шаблоны классов могут содержать виртуальные методы. Виртуальные методы обеспечивают возможность переопределения функций в производных классах. Однако, как и в случае с обычными классами, только базовый класс может содержать определение виртуальной функции. В класс-потомок должно быть только объявление переопределенной виртуальной функции.

Например:

template <typename T>

class MyBaseClass {

public:

virtual void MyVirtualFunction(T data) = 0; // чисто виртуальная функция в базовом классе

};

class MyDerivedClass : public MyBaseClass<int> {

public:

void MyVirtualFunction(int data) override { // переопределенная функция в классе-потомке

// реализация

}

};

1. Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса, если они не зависят от параметра шаблона. Для определения таких функций необходимо использовать синтаксис шаблонного параметра в качестве префикса для имени функции и передавать параметр шаблона явно как аргумент функции. Например:

template <typename T>

class MyClass {

public:

void function1(); // компонентная функция, которая не зависит от параметра шаблона

};

// Определение компонентной функции

template <typename T>

void MyClass<T>::function1() {

// ...

}

// Определение компонентной функции, которая зависит от параметра шаблона

template <typename T>

void MyClass<T>::function2(T arg) {

// ...

}

1. Инстанцирование шаблона - это процесс создания конкретной реализации шаблона класса или функции на основе заданных параметров шаблона. Во время компиляции компилятор создает новый объект или функцию на основе шаблона с заданными значениями параметров шаблона. Эти конкретные объекты или функции называются инстансами шаблона и могут быть использованы так же, как и обычные объекты или функции в программе. Инстанцирование шаблона происходит только тогда, когда компилятор встречает ссыль на шаблон в коде программы.
2. Генерирование определения класса по шаблону происходит на стадии компиляции, когда компилятор обрабатывает код программы и создает конкретный объект или функцию на основе шаблона с заданными значениями параметров шаблона. Инстанциация шаблона позволяет генерировать множество определений классов из одного и того же шаблона с разными параметрами шаблона. Таким образом, инстанцирование шаблона позволяет создавать множество конкретных объектов или функций на основе одного и того же шаблона во время компиляции.