УО «Белорусский государственный технологический университет»

Факультет информационных технологий

Кафедра «Информационных систем и технологий»

Специальность 1-40 05 01 «Информационные Системы и Технологии»

**Реферат**

**На тему «Шаблоны С++»**

|  |  |  |  |
| --- | --- | --- | --- |
| **Выполнил** |  |  |  |
| Студент 1 курса группы 1 |  |  | Д.И. Велютич |
|  | подпись, дата |  | инициалы и фамилия |
| **Проверил** |  |  | Н.И. Белодед |
|  | подпись, дата |  | инициалы и фамилия |

Минск 2023

Оглавление

[Введение и теория 2](#_Toc134322939)

[Объяснения работы программы 4](#_Toc134322940)

[Программный код 8](#_Toc134322941)

[Скриншоты и объяснения работы программы 10](#_Toc134322942)

[Выводы 11](#_Toc134322943)

# Введение и теория

Шаблоны в C++ являются мощным инструментом метапрограммирования, позволяющим разработчикам создавать функции или классы, которые могут работать с любым типом данных. Они были введены в стандарт языка C++ в 1998 году с выпуском стандарта ISO/IEC 14882:1998.

**Теория**

Базовой идеей шаблонов в C++ является создание "шаблона" или "рецепта" для кода, который может быть адаптирован для работы с любым типом данных. Самый простой пример этого - это функция сортировки. Вместо того чтобы писать отдельные функции сортировки для массивов целых чисел, действительных чисел, строк и т.д., вы можете написать одну функцию сортировки, которая работает со всеми этими типами.

Шаблоны в C++ могут быть классифицированы как функциональные шаблоны и шаблоны классов.

Функциональные шаблоны представляют собой функции, которые могут быть параметризованы типом. Это означает, что вы можете использовать одну и ту же функцию для различных типов данных.

Шаблоны классов позволяют создавать классы, которые могут работать с различными типами данных. Примером может служить стандартная библиотека шаблонов (STL) в C++, которая включает в себя ряд полезных шаблонов классов, таких как векторы, списки, очереди и т.д.

**Преимущества**

1. Универсальность: Шаблоны позволяют писать код, который работает с любым типом данных, уменьшая дублирование кода.

2. Эффективность: Поскольку шаблоны компилируются во время компиляции, они обычно не вносят никаких дополнительных накладных расходов по сравнению с кодом, написанным специально для каждого типа данных.

3. Гибкость: Шаблоны позволяют создавать более обобщенные и гибкие алгоритмы и структуры данных.

**Недостатки**

1. Сложность: Шаблоны могут быть сложными для понимания и использования, особенно в сложных и современных техниках метапрограммирования.

2. Сообщения об ошибках: Ошибки, связанные с шаблонами, часто сложны для понимания и разрешения, поскольку они могут включать много деталей о внутренней работе шаблонов.

3. Время компиляции: Использование шаблонов может увеличить время компиляции, особенно при использовании сложных шаблонов или техник метапрограммирования.

4. Размер бинарных файлов: Так как каждая специализация шаблона порождает свой набор инструкций при компиляции, это может привести к увеличению размера итогового исполняемого файла.

**Область применения**

Шаблоны наиболее полезны в ситуациях, где требуется работа с различными типами данных, но алгоритмы или структуры данных остаются неизменными. Например, в библиотеках, стандартных алгоритмах, контейнерах, итераторах, функциональных объектах, и т.д. Они также полезны для реализации обобщенного программирования и метапрограммирования.

**История**

Шаблоны были впервые введены в C++ в 1998 году и были частью стандарта C++98. Они были созданы в ответ на потребность в более обобщенных алгоритмах и структурах данных. Несмотря на их сложность, они быстро стали одной из самых мощных и популярных возможностей C++. С тех пор было много улучшений и расширений, включая введение шаблонов с переменным числом аргументов в C++11, и концепций в C++20, которые позволяют выражать ожидания от типов шаблонов.

# Объяснения работы программы

Эта программа реализует шаблонный класс LinkedList (связанный список) в языке программирования C++. Шаблонный класс позволяет создавать списки элементов различных типов.

Программа начинается с включения заголовочного файла iostream, который содержит определения стандартных потоков ввода-вывода. Затем определены две структуры: структура Node (узел списка) и шаблонный класс LinkedList.

Структура Node содержит два поля: data для хранения данных и next для указания на следующий узел списка.

Шаблонный класс LinkedList имеет приватные члены данных head (указатель на начало списка) и tail (указатель на конец списка). Он содержит методы для добавления элементов в конец списка (add) и вывода списка на экран (print).

В функции main() создаются три экземпляра LinkedList для разных типов данных: intList для целых чисел, doubleList для вещественных чисел и stringList для строк. В каждый из списков добавляются несколько элементов с помощью метода add(), а затем содержимое каждого списка выводится на экран с помощью метода print().

В итоге, программа создает три разных связанных списка (целых чисел, вещественных чисел и строк), заполняет их данными и выводит содержимое каждого списка на экран.

1. Сначала мы определяем структуру `Node`, которая является элементом нашего связанного списка. Это шаблонная структура, поэтому `Node` может хранить данные любого типа `T`. Каждый узел содержит данные и указатель на следующий узел списка.

template <typename T>

struct Node {

T data;

Node\* next;

};

2. Затем мы определяем класс `LinkedList`, который является нашим связанным списком. Это также шаблонный класс, поэтому `LinkedList` может хранить узлы с данными любого типа `T`.

template <typename T>

class LinkedList {

3. Внутри класса `LinkedList` у нас есть два частных члена: `head` и `tail`, которые являются указателями на начало и конец списка соответственно.

private:

Node<T>\* head;

Node<T>\* tail;

4. Конструктор класса инициализирует `head` и `tail` нулевыми указателями, что означает, что список изначально пуст.

public:

LinkedList() : head(nullptr), tail(nullptr) {}

5. Деструктор класса удаляет все узлы списка. Это необходимо, чтобы предотвратить утечки памяти.

~LinkedList() {

Node<T>\* current = head;

while (current != nullptr) {

Node<T>\* next = current->next;

delete current;

current = next;

}

}

6. Функция `add` добавляет новый узел в конец списка. Если список пуст, `head` и `tail` указывают на новый узел. В противном случае новый узел добавляется после `tail`, и `tail` начинает указывать на новый узел.

void add(T data) {

Node<T>\* node = new Node<T>{ data, nullptr };

if (head == nullptr) {

head = tail = node;

} else {

tail->next = node;

tail = node;

}

}

7. Функция `print` выводит на экран все элементы списка. Она проходит по списку, начиная с `head`, и печатает данные каждого узла.

void print() {

Node<T>\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

8. В функции `main` мы создаем и используем несколько связанных списков разных типов данных. Каждый список заполняется различными данными с помощью функции `add`, а затем выводится на экран с помощью функции `print`.

int main() {

// Создаем список целых чисел

LinkedList<int> intList;

intList.add(5);

intList.add(10);

intList.add(15);

std::cout << "Список целых чисел: ";

intList.print();

// Создаем список вещественных чисел

LinkedList<double> doubleList;

doubleList.add(5.5);

doubleList.add(10.1);

doubleList.add(15.2);

std::cout << "Список вещественных чисел: ";

doubleList.print();

// Создаем список строк

LinkedList<std::string> stringList;

stringList.add("Hello");

stringList.add("World");

stringList.add("!");

std::cout << "Список строк: ";

stringList.print();

return 0;

}

9. В функции `main` мы создаем три списка разных типов данных (`int`, `double`, и `std::string`). Мы используем функцию `add` для добавления элементов в каждый список, а затем используем функцию `print` для вывода содержимого списков на экран.

В итоге, эта программа демонстрирует использование шаблонов для создания обобщенных структур данных, таких как связанный список, который может хранить элементы различных типов данных без изменения кода класса.

# Программный код

#include <iostream>

// Определение структуры "Node", которая представляет узел в связном списке.

// Это шаблонная структура, поэтому она может хранить данные любого типа T.

template <typename T>

struct Node {

T data; // Данные, хранящиеся в узле

Node\* next; // Указатель на следующий узел в списке

};

// Определение шаблонного класса "LinkedList".

// Это связанный список, который может хранить элементы любого типа T.

template <typename T>

class LinkedList {

private:

Node<T>\* head; // Указатель на начало списка

Node<T>\* tail; // Указатель на конец списка

public:

// Конструктор класса. Инициализирует head и tail как nullptr, т.е. список пуст.

LinkedList() : head(nullptr), tail(nullptr) {}

// Деструктор класса. Удаляет все узлы в списке, чтобы избежать утечки памяти.

~LinkedList() {

Node<T>\* current = head; // Начинаем с начала списка

while (current != nullptr) { // Пока не пройдем весь список

Node<T>\* next = current->next; // Сохраняем указатель на следующий узел

delete current; // Удаляем текущий узел

current = next; // Переходим к следующему узлу

}

}

// Функция для добавления нового элемента в конец списка

void add(T data) {

Node<T>\* node = new Node<T>{ data, nullptr }; // Создаем новый узел с данными

if (head == nullptr) { // Если список пуст

head = tail = node; // Новый узел становится единственным в списке

}

else { // Если в списке уже есть элементы

tail->next = node; // Добавляем новый узел в конец списка

tail = node; // Обновляем указатель на конец списка

}

}

// Функция для печати всех элементов списка

void print() {

Node<T>\* current = head; // Начинаем с начала списка

while (current != nullptr) { // Пока не пройдем весь список

std::cout << current->data << " "; // Печатаем данные текущего узла

current = current->next; // Переходим к следующему узлу

}

std::cout << std::endl; // Печатаем новую строку после всех элементов списка

}

};

int main() {

setlocale(0, "rus"); // Установка локали для поддержки русского языка

// Создаем список целых чисел

LinkedList<int> intList; // Создаем новый список для целых чисел

intList.add(5); // Добавляем числа в список

intList.add(10);

intList.add(15);

std::cout << "Список целых чисел: ";

intList.print(); // Печатаем все элементы списка

// Создаем список вещественных чисел

LinkedList<double> doubleList; // Создаем новый список для вещественных чисел

doubleList.add(5.5); // Добавляем числа в список

doubleList.add(10.1);

doubleList.add(15.2);

std::cout << "Список вещественных чисел: ";

doubleList.print(); // Печатаем все элементы списка

// Создаем список строк

LinkedList<std::string> stringList; // Создаем новый список для строк

stringList.add("Hello"); // Добавляем строки в список

stringList.add("World");

stringList.add("!");

std::cout << "Список строк: ";

stringList.print(); // Печатаем все элементы списка

return 0; // Конец функции main. Программа успешно завершена.

}

# Скриншоты и объяснения работы программы

Эта программа демонстрирует использование шаблонного класса связного списка (LinkedList) в C++.

LinkedList - это тип данных, где каждый элемент содержит ссылку на следующий элемент. Это обеспечивает эффективное добавление и удаление элементов из любого места в списке. В данной программе связный список реализован как шаблон, что позволяет хранить в нем объекты различных типов.

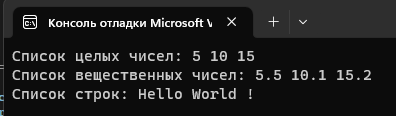
В главной функции main() программа создает три разных связных списка:

1. Список целых чисел (LinkedList<int>).

2. Список вещественных чисел (LinkedList<double>).

3. Список строк (LinkedList<std::string>).

В каждый из этих списков добавляются три значения соответствующего типа, после чего они выводятся на экран. Это демонстрирует универсальность и гибкость использования шаблонов в C++.



# Выводы

Шаблоны в C++ представляют собой мощный инструмент, который позволяет разработчикам создавать обобщенные функции и классы, способные работать с любым типом данных. Они играют центральную роль в обобщенном программировании и метапрограммировании в C++, и являются ключевым компонентом многих стандартных библиотек и алгоритмов.

Хотя шаблоны могут быть сложными в использовании и понимании, особенно для новых программистов, их преимущества в гибкости, универсальности и эффективности обычно превосходят эти сложности. Однако разработчикам следует быть внимательными к потенциальным проблемам, таким как увеличение времени компиляции и сложность отладки.

В итоге, понимание и использование шаблонов в C++ является важным навыком для каждого программиста на C++, и они будут продолжать играть важную роль в развитии языка в будущем.