**Глава 1: Введение в C++ и его история**

**1.1. Что такое C++?**

Представь себе, что C++ — это как конструктор LEGO, но для программ. Он позволяет создавать из простых "кубиков" (инструкций) сложные и мощные приложения. Но, в отличие от LEGO, C++ дает тебе полный контроль над "деталями", что позволяет создавать очень эффективные программы.

В более формальном определении, C++ - это *компилируемый, статически типизированный язык программирования общего назначения*. Давай разберем это определение по частям:

* **Компилируемый:** Это значит, что написанный тобой код на C++ (называемый "исходным кодом") сначала преобразуется специальной программой – компилятором – в машинный код (последовательность инструкций, которые понимает компьютер), а затем этот машинный код выполняется компьютером. Это делает программы на C++ очень быстрыми.
  + **Аналогия:** Представь, что ты пишешь письмо на русском, а компилятор — это переводчик, который переводит это письмо на английский, чтобы его мог прочитать англоговорящий получатель.
* **Статически типизированный:** Это значит, что ты должен явно указать тип данных для каждой переменной (например, число, символ, строка) при ее объявлении. Компилятор проверяет типы данных во время компиляции, чтобы убедиться, что все операции выполняются правильно.
  + **Аналогия:** Представь, что ты сортируешь разные типы игрушек в отдельные коробки. Статическая типизация заставляет тебя четко указать, что эта коробка для машинок, а эта для кукол, чтобы не было путаницы.
* **Язык программирования общего назначения:** Это значит, что C++ можно использовать для создания широкого спектра программ: от игр и операционных систем до научных симуляций и встраиваемых систем. Это как универсальный инструмент, который может пригодиться для многих задач.

**1.2. Краткая история C++**

C++ не возник из ниоткуда. Он имеет свою историю, которая тесно связана с языком C.

* **1970-е: Рождение языка C**
  + Язык C был разработан Деннисом Ритчи в Bell Labs для создания операционной системы Unix.
  + C стал очень популярным благодаря своей простоте, гибкости и высокой производительности.
  + Он позволял программистам иметь прямой доступ к памяти, что было очень важно в то время.
* **1979: Bjarne Stroustrup и "C with Classes"**
  + Бьярн Страуструп, работая в Bell Labs, начал разрабатывать расширение языка C, которое добавило поддержку объектно-ориентированного программирования (ООП).
  + Он назвал свой язык "C with Classes" (С с классами). Классы позволяют создавать "объекты", которые могут иметь свои свойства и функции, что упрощает разработку сложных программ.
* **1983: C++**
  + Язык "C with Classes" был переименован в C++. Оператор ++ в C обозначает инкремент (увеличение на 1). Название C++ намекает на то, что это "улучшенная" версия языка C.
  + В C++ были добавлены новые функции, такие как виртуальные функции, перегрузка операторов, шаблоны и обработка исключений.
  + C++ сохранил обратную совместимость с C, это означало, что можно было использовать старый код на C в C++ программе.
* **1998: Стандартизация C++ (C++98)**
  + Был выпущен первый международный стандарт языка C++ (ISO/IEC 14882:1998), который определил основные правила языка и его стандартную библиотеку.
  + Это гарантировало, что программы на C++ будут работать одинаково на разных платформах.
* **2003, 2011, 2014, 2017, 2020, 2023: Новые стандарты**
  + Позже были выпущены новые стандарты языка (C++03, C++11, C++14, C++17, C++20, C++23), которые добавляли новые возможности и улучшали существующие.
  + Каждый новый стандарт приносил улучшения в производительности, безопасности и удобстве языка.
  + Стандарт C++20, например, добавил такие мощные функции, как модули и корутины.

**1.3. Почему C++ по-прежнему популярен?**

Несмотря на появление новых языков, C++ остается одним из самых популярных языков программирования по нескольким причинам:

* **Производительность:** Он идеально подходит для задач, где важна максимальная скорость и минимальное потребление ресурсов (например, игры, операционные системы, высокопроизводительные вычисления).
* **Контроль:** Он дает разработчику максимальный контроль над памятью и ресурсами. Это позволяет создавать высокооптимизированный код.
* **Универсальность:** Он подходит для самых разных проектов.
* **Большое сообщество:** C++ имеет огромное сообщество разработчиков, что означает, что есть много готовых библиотек, инструментов и ресурсов для обучения.
* **Наследие:** Многие существующие проекты и технологии используют C++, поэтому знание языка является ценным навыком.

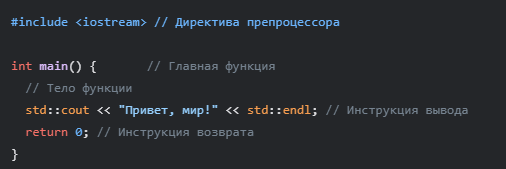
**Глава 2: Структура программы на C++**

В этой главе мы изучим, из чего состоит программа на C++, как она строится и как ее правильно организовывать. Мы рассмотрим следующие ключевые понятия:

* Директивы препроцессора
* Функция main()
* Объявление переменных
* Инструкции и операторы
* Комментарии

**2.1. Базовая структура программы**

Самая простая программа на C++ выглядит примерно так:



Давай разберем эту программу построчно:

* **#include <iostream>**: Это директива препроцессора.
  + **Препроцессор** — это специальная программа, которая обрабатывает исходный код перед тем, как его компилирует компилятор.
  + #include говорит препроцессору вставить содержимое файла iostream в текущий файл.
  + Файл iostream (input-output stream) содержит объявления для ввода и вывода данных, такие как std::cout и std::endl. Мы поговорим о потоках ввода-вывода подробнее в будущем.
* **int main() { ... }**: Это объявление главной функции программы.
  + **Функция** — это блок кода, который выполняет определенную задачу.
  + main — это специальная функция, с которой начинается выполнение любой программы на C++.
  + int — это тип возвращаемого значения функции main. В данном случае это целое число, которое обычно используется для обозначения статуса завершения программы (0 - успешное завершение).
  + { ... } — это тело функции, где помещается весь код, который нужно выполнить.
* **std::cout << "Привет, мир!" << std::endl;**: Это инструкция вывода на экран.
  + std::cout (character output) — это объект, который представляет стандартный поток вывода (обычно консоль).
  + << — это оператор вывода, который помещает данные в поток вывода.
  + "Привет, мир!" — это строковая константа, которую мы хотим вывести на экран.
  + std::endl — это манипулятор, который добавляет перевод строки в поток вывода (переводит курсор на следующую строку).
* **return 0;**: Это инструкция возврата значения из функции.
  + return — это оператор возврата, который возвращает значение из функции.
  + 0 — это целое число, которое означает, что программа завершилась успешно.

**2.2. Директивы препроцессора**

Как мы уже видели, директивы препроцессора начинаются с символа #. Они используются для управления процессом компиляции. Наиболее распространенные директивы:

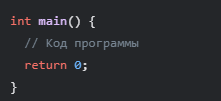
* #include: Включает содержимое указанного файла в текущий исходный файл. Используется для подключения библиотек и других файлов.
* #define: Определяет макрос, который представляет собой замену текста.



* #ifdef, #ifndef, #else, #endif: Используются для условной компиляции, когда некоторые части кода компилируются в зависимости от определенного условия.

**2.3. Функция main()**

Функция main() — это сердце любой программы на C++. Она имеет следующую структуру:

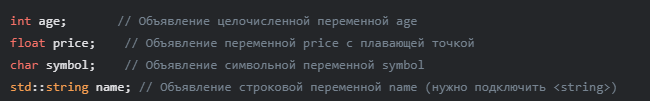


* int указывает на то, что функция возвращает целое число (обычно 0 при успешном завершении).
* main — это имя функции.
* () — это список параметров (в данном случае список пуст).
* { ... } — это тело функции, которое содержит код, который нужно выполнить.

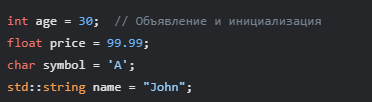
Функция main() может также принимать аргументы командной строки, но об этом мы поговорим позже.

**2.4. Объявление переменных**

Переменные используются для хранения данных. В C++ ты должен объявить переменную, прежде чем ее использовать. Объявление переменной включает в себя указание ее типа и имени.



Ты также можешь инициализировать переменную при объявлении:



**2.5. Инструкции и операторы**

Инструкция — это команда, которая заставляет программу выполнить какое-либо действие. Обычно инструкция заканчивается точкой с запятой ;.

Примеры инструкций:

* Объявление переменной: int age;
* Присваивание значения: age = 30;
* Вывод на экран: std::cout << "Привет!";
* Вызов функции: return 0;

Операторы — это символы, которые выполняют определенные действия над данными (операндами).

Примеры операторов:

* Арифметические операторы: +, -, \*, /, %
* Оператор присваивания: =
* Операторы сравнения: ==, !=, >, <, >=, <=
* Операторы ввода-вывода: <<, >>

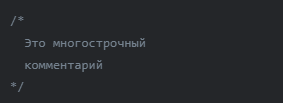
**2.6. Комментарии**

Комментарии — это текст в коде, который игнорируется компилятором. Они используются для пояснения кода и делают его более понятным. В C++ есть два типа комментариев:

* Однострочные комментарии начинаются с //:



* Многострочные комментарии начинаются с /\* и заканчиваются \*/:



**2.7. Пример полной программы**

Вот пример программы, которая демонстрирует большинство элементов, которые мы рассмотрели:



**2.8. Закрепление материала**

1. Что такое директивы препроцессора и зачем они нужны?
2. Какова роль функции main() в программе?
3. Как объявляются переменные в C++?
4. Что такое инструкция и оператор?
5. Для чего нужны комментарии?

Попробуй самостоятельно написать простую программу, которая:

1. Запрашивает имя пользователя.
2. Запрашивает возраст пользователя.
3. Выводит на экран сообщение с приветствием и возрастом пользователя.

**Глава 3: Переменные в C++**

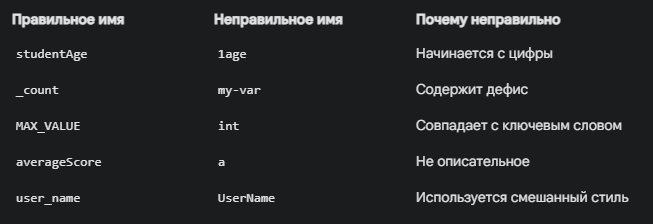
Переменная — это именованная область памяти, в которой можно хранить данные. Представь себе переменную как контейнер, в котором можно положить значение определенного типа (число, символ, строка и т.д.). В C++, как и в других языках программирования, переменные играют ключевую роль в хранении и обработке информации.

**3.1. Именование переменных**

Имена переменных (идентификаторы) в C++ должны соответствовать определенным правилам. Правила хорошего стиля и читаемого кода также играют важную роль.

* **Правила именования:**
  + Имя переменной может состоять из букв (a-z, A-Z), цифр (0-9) и символа подчеркивания \_.
  + Имя переменной должно начинаться с буквы или символа подчеркивания. Цифра в начале недопустима.
  + C++ — регистрозависимый язык, то есть age, Age и AGE — это три разные переменные.
  + Имя переменной не может совпадать с ключевыми словами C++ (например, int, float, return, if и т.д.).
* **Хороший стиль именования:**
  + Имена переменных должны быть *описательными*. Это означает, что имя должно давать представление о том, какая информация хранится в переменной. Например, studentAge лучше, чем sa.
  + Используй *camelCase* или *snake\_case* для составных имен:
    - **camelCase**: firstName, studentGrade, calculateArea
    - **snake\_case**: first\_name, student\_grade, calculate\_area
    - Выбери один стиль и придерживайся его в своем коде.
  + Имена констант (значений, которые не изменяются) обычно пишут заглавными буквами с подчеркиваниями: MAX\_VALUE, PI.
  + Избегай слишком коротких имен (кроме счетчиков в циклах, например, i, j) или слишком длинных имен.
  + Избегай использования имен, которые могут быть легко перепутаны, например, l (маленькая L) и 1 (единица).

**Примеры правильных и неправильных имен переменных:**

****

**3.2. Определение переменной**

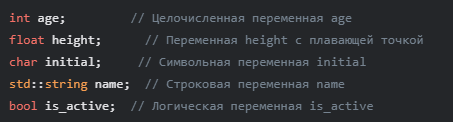
Определение переменной — это процесс, в результате которого выделяется память для хранения данных определенного типа и переменной присваивается имя. Определение переменной включает в себя:

* **Указание типа данных:** Тип данных определяет, какие значения можно хранить в переменной (целые числа, числа с плавающей точкой, символы, строки и т.д.).
* **Имя переменной:** Имя, которое будет использоваться для доступа к переменной.

Синтаксис определения переменной:



Примеры определения переменных:



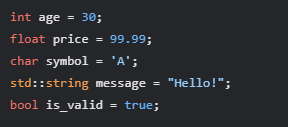
В C++, **каждая переменная должна быть определена до ее использования**. Попытка использования неопределенной переменной приведет к ошибке компиляции.

**3.3. Инициализация переменных**

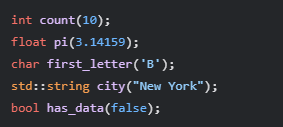
Инициализация переменной — это процесс присваивания ей начального значения при объявлении. Это очень важный шаг, так как использование неинициализированной переменной может привести к непредсказуемому поведению программы.

В C++ есть несколько способов инициализации переменных:

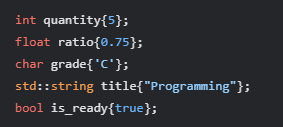
1. **Инициализация с использованием оператора =:**



1. **Инициализация в скобках:**



1. **Инициализация с использованием фигурных скобок (списковая инициализация):**



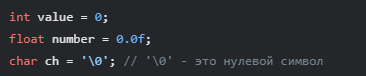
Фигурные скобки имеют дополнительное преимущество: они предотвращают неявные преобразования типов (сужение). Например, если ты попытаешься инициализировать int переменной типа float с использованием фигурных скобок, компилятор выдаст ошибку. Это помогает избежать потенциальных проблем, связанных с потерей данных.



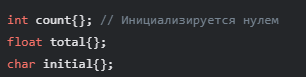
**3.4. Инициализация нулем**

Иногда полезно инициализировать переменные нулевым значением при объявлении. Это можно сделать следующими способами:

1. **Явная инициализация нулем:**



1. **Инициализация в фигурных скобках:**



Если ты используешь фигурные скобки без начального значения, переменная будет инициализирована нулем (или значением по умолчанию для соответствующего типа).

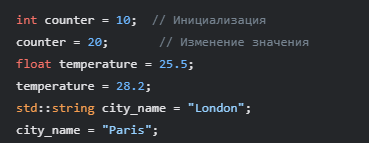
**3.5. Инициализация по умолчанию**

Если переменная не инициализируется явно, то ее значение будет *неопределенным*. Это значит, что переменная может содержать случайное значение из области памяти, где она была выделена.

**Важно:** Всегда явно инициализируй переменные при объявлении, чтобы избежать непредсказуемого поведения программы.

**3.6. Изменение значения переменной**

После инициализации значение переменной можно изменять с помощью оператора присваивания =:



Оператор присваивания копирует значение справа от него в переменную, находящуюся слева.

**3.7. Закрепление материала**

1. Перечисли правила именования переменных в C++.
2. Почему важно выбирать описательные имена для переменных?
3. Что такое определение переменной?
4. Почему важно инициализировать переменные?
5. Какие способы инициализации переменных ты знаешь?
6. В чем отличие инициализации с помощью фигурных скобок от инициализации с помощью оператора =?
7. Что происходит, если не инициализировать переменную?
8. Как изменить значение переменной после ее инициализации?

**Практическое задание:**

1. Создай несколько переменных разных типов (int, float, char, string, bool).
2. Присвой им начальные значения различными способами (с помощью =, в скобках, в фигурных скобках).
3. Измени значения этих переменных.
4. Выведи значения переменных на экран, чтобы проверить, что они действительно изменились.
5. Попробуй использовать разные стили именования переменных (camelCase, snake\_case) и выбери тот, который тебе больше нравится.

**Глава 4: Константы в C++**

Константа – это именованная область памяти, которая содержит значение, которое не может быть изменено во время выполнения программы. Константы играют важную роль в обеспечении безопасности, надежности и читаемости кода.

**4.1. Зачем нужны константы?**

Константы используются для:

* **Хранения неизменяемых значений:** Например, математические константы (число π), физические константы (скорость света), значения конфигурации (максимальный размер массива) и т.д.
* **Улучшения читаемости кода:** Использование констант с описательными именами делает код более понятным и легким для восприятия.
* **Предотвращения ошибок:** Компилятор гарантирует, что значение константы не будет случайно изменено. Это уменьшает вероятность ошибок в программе.
* **Упрощения сопровождения кода:** Если необходимо изменить значение, которое используется в нескольких местах программы, то достаточно изменить значение константы в одном месте, а не искать и менять его по всему коду.

**4.2. Способы объявления констант**

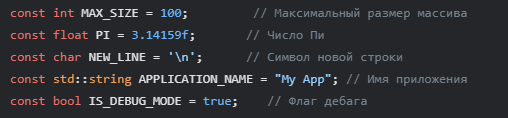
В C++ есть два основных способа объявления констант:

1. **С использованием ключевого слова const:**  
   Этот способ является наиболее распространенным и рекомендуется для большинства случаев.



* const — ключевое слово, которое указывает, что это константа.
* тип\_данных — тип данных константы (например, int, float, char).
* имя\_константы — имя константы, которое должно соответствовать правилам именования переменных (но обычно используется UPPER\_SNAKE\_CASE).
* значение — начальное значение константы.

**Примеры:**

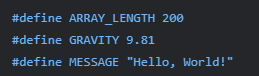


1. **С использованием директивы препроцессора #define:**  
   Этот способ, унаследованный от C, является менее предпочтительным в C++ и лучше избегать его использования, если есть возможность использовать const.



* #define — директива препроцессора, которая определяет макрос.
* имя\_константы — имя константы (обычно пишется заглавными буквами).
* значение — значение константы.

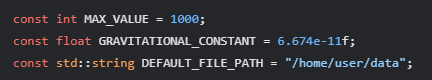
**Примеры:**



1. **Почему #define не рекомендуется?**
   * #define работает на уровне препроцессора, а не на уровне компилятора, поэтому у него нет типовой безопасности.
   * Ошибки, связанные с #define, могут быть сложнее в отладке.
   * Константы, объявленные с помощью const, могут иметь тип данных, что позволяет компилятору проверять правильность их использования.

**4.3. Правила именования констант**

По соглашению, имена констант обычно пишутся заглавными буквами, а слова разделяются подчеркиванием (UPPER\_SNAKE\_CASE). Это делает их легко узнаваемыми в коде.



**4.4. Инициализация констант**

Константы должны быть инициализированы при объявлении. После инициализации значение константы не может быть изменено. Попытка изменить значение константы приведет к ошибке компиляции.

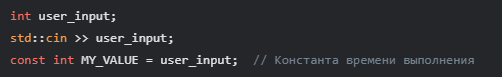


**4.5. Константы времени компиляции и времени выполнения**

* **Константы времени компиляции:** Значения констант const, объявленных с использованием литералов, вычисляются во время компиляции. Это означает, что их значения известны на этапе компиляции.



* **Константы времени выполнения:** Значение константы const может быть получено во время выполнения программы, например, из аргументов командной строки, ввода пользователя или из других переменных.



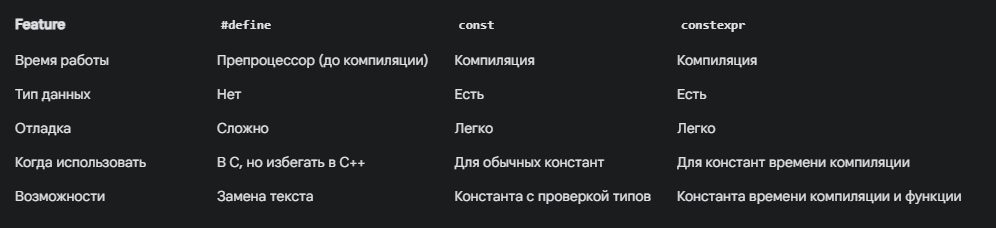
**4.6. constexpr (константы, вычисляемые на этапе компиляции)**

В C++11 появился ключевое слово constexpr, которое позволяет объявлять константы, значения которых должны быть вычислены на этапе компиляции. constexpr константы могут использоваться там, где требуются константы времени компиляции (например, при определении размеров массивов).



constexpr может использоваться не только для простых значений, но и для функций, которые возвращают константное значение на этапе компиляции.

**4.7. Различия между #define, const и constexpr**



**4.8. Закрепление материала**

1. Зачем нужны константы в программировании?
2. Какие способы объявления констант в C++ ты знаешь?
3. В чем разница между const и #define?
4. Почему const предпочтительнее, чем #define в C++?
5. Как и когда использовать constexpr?
6. Что такое константы времени компиляции и времени выполнения?

**Практическое задание:**

1. Объяви несколько констант разных типов (int, float, char, string, bool), используя const.
2. Попробуй объявить константы с помощью #define (но помни, что это не рекомендуется).
3. Попытайся изменить значение константы и посмотри, как отреагирует компилятор.
4. Попробуй использовать constexpr для константы и для функции.

**Глава 5: Ввод и вывод в консоль в C++**

Ввод и вывод — это способы, которыми программа может получать данные от пользователя (ввод) и отображать результаты своей работы (вывод). Консоль — это стандартное текстовое окно, через которое осуществляется это взаимодействие.

В C++ для ввода и вывода используются объекты из библиотеки iostream, в частности, std::cin (стандартный ввод) и std::cout (стандартный вывод).

**5.1. Вывод в консоль (std::cout)**

std::cout — это объект, представляющий стандартный поток вывода (обычно это консоль). Для вывода данных на экран используется оператор << (оператор вставки в поток).



**Основные моменты:**

* std::endl: добавляет символ новой строки, перемещая курсор на следующую строку.
* Можно выводить текст (строковые литералы), значения переменных и выражения.
* Можно объединять несколько выводов в одну инструкцию, используя оператор <<.

**5.2. Ввод из консоли (std::cin)**

std::cin — это объект, представляющий стандартный поток ввода (обычно это клавиатура). Для получения данных от пользователя используется оператор >> (оператор извлечения из потока).

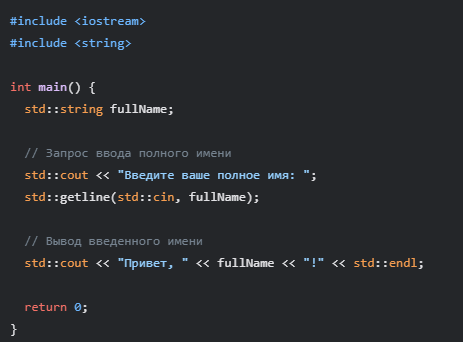


**Основные моменты:**

* std::cin >> variable; извлекает значение из потока ввода и сохраняет его в переменную variable.
* Тип переменной variable должен соответствовать типу вводимых данных.
* Оператор >> пропускает пробелы и переводы строк, и считывает данные до ближайшего пробела или перевода строки.
* Для ввода строк с пробелами нужно использовать std::getline().

**5.3. Ввод строк с пробелами (std::getline())**

Для ввода строк, которые могут содержать пробелы, используется функция std::getline(). Она считывает всю строку до символа новой строки (\n).



**5.4. Комбинирование ввода и вывода**

Можно комбинировать ввод и вывод в одной программе, чтобы создавать интерактивные приложения.



**Обрати внимание:**

* std::ws используется для пропуска начальных пробелов перед считыванием строки через std::getline. Без него предыдущий ввод (std::cin >> num1 >> num2;) может оставить перенос строки, который затем будет считан getline.

**5.5. Закрепление материала**

1. Для чего используется std::cout?
2. Для чего используется std::cin?
3. Как вывести на экран текст и значение переменной?
4. Как ввести значение с клавиатуры и сохранить его в переменную?
5. Как ввести строку с пробелами?
6. Напиши программу, которая запрашивает у пользователя два числа, а затем выводит их сумму.
7. Напиши программу, которая запрашивает у пользователя его имя и фамилию, а затем выводит их на экран в формате "Фамилия, Имя".

**Глава 6: Пространства имен и псевдонимы (using) в C++**

**6.1. Пространства имен (Namespaces)**

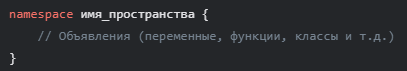
Пространства имен (namespaces) — это механизм для организации имен в C++, предотвращая конфликты имен. Представь, что пространство имен — это как папка, в которой хранятся имена переменных, функций, классов и других элементов программы. В разных папках могут быть файлы с одинаковыми именами, и это не вызовет конфликта.

**Зачем нужны пространства имен?**

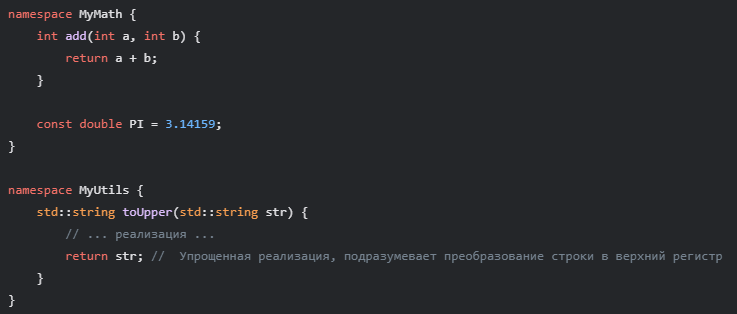
* **Предотвращение конфликтов имен:** Когда два разных разработчика или библиотеки используют одно и то же имя для разных целей, это может привести к конфликту. Пространства имен позволяют разделить эти имена, помещая их в разные области видимости.
* **Организация кода:** Пространства имен позволяют логически группировать связанные имена, что делает код более читаемым и структурированным.

**6.2. Объявление пространства имен**

Для объявления пространства имен используется ключевое слово namespace:



**Пример:**



**6.3. Доступ к элементам пространства имен**

Для доступа к элементам (переменным, функциям, классам) из пространства имен используется оператор разрешения области видимости :: (scope resolution operator).



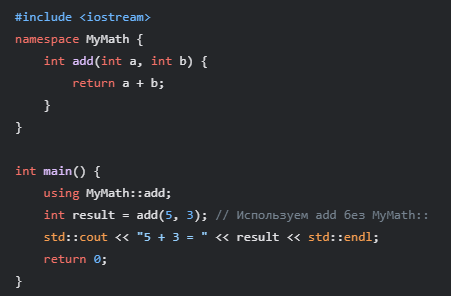
**6.4. Директива using**

Использование оператора :: для доступа к элементам пространства имен может сделать код громоздким. Для упрощения доступа можно использовать директиву using. Директива using бывает двух видов:

1. **using для конкретного элемента:**



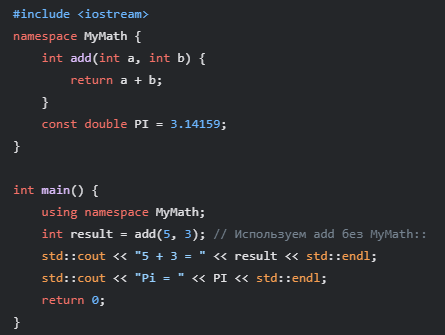
Это позволяет использовать элемент непосредственно, без указания имени пространства имен.



1. **using для всего пространства имен:**



Это позволяет использовать все элементы из указанного пространства имен без явного указания имени пространства.



1. **Важно:** Использование using namespace для всего пространства имен может привести к конфликтам имен, особенно в больших проектах. Поэтому, предпочтительно использовать using для конкретных элементов, или использовать using namespace в небольших файлах.

**6.5. Псевдонимы (Aliases)**

Иногда имена пространств имен или типов данных могут быть очень длинными. Для упрощения их использования можно определить псевдонимы (alias). Для этого используется ключевое слово using.



**Пример:**



**6.6. using namespace std;**

Ты наверняка заметил строку using namespace std; во многих примерах кода. std — это пространство имен стандартной библиотеки C++, в которой определены такие объекты как cout, cin, endl, string и многие другие.

Использование using namespace std; упрощает доступ ко всем элементам из std без явного указания std::. **В небольших программах это удобно, но в больших проектах это может привести к проблемам**, поэтому лучше избегать использование using namespace std; в глобальной области видимости.

**6.7. Закрепление материала**

1. Что такое пространства имен и для чего они нужны?
2. Как объявить пространство имен?
3. Как получить доступ к элементам пространства имен?
4. Для чего используется директива using?
5. В чем разница между using для конкретного элемента и using namespace?
6. Когда лучше использовать псевдонимы?
7. Почему не рекомендуется использовать using namespace std; в глобальной области видимости?

**Практическое задание:**

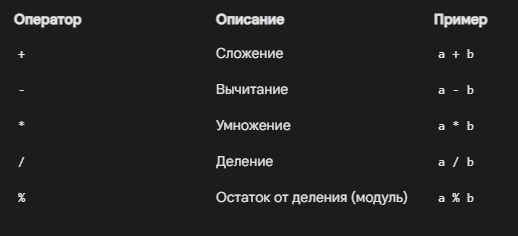
1. Создай свое собственное пространство имен MyProject.
2. Помести в него несколько функций и переменных.
3. Используй элементы из твоего пространства имен в функции main(), как с помощью ::, так и с помощью директивы using.
4. Создай псевдонимы для нескольких типов данных и используй их в своей программе.

**Глава 7: Арифметические операции в C++**

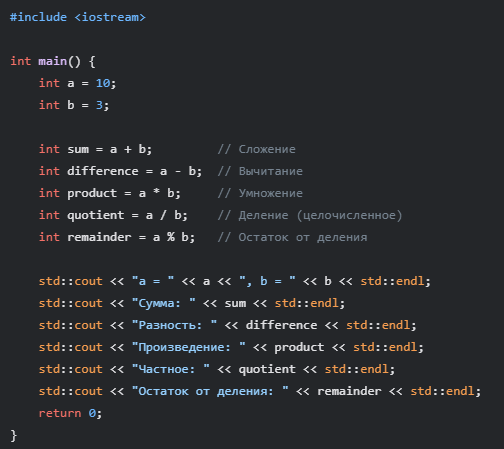
Арифметические операции позволяют выполнять математические вычисления над числовыми значениями. C++ предоставляет широкий набор арифметических операторов, которые позволяют выполнять сложение, вычитание, умножение, деление и другие операции.

**7.1. Основные арифметические операторы**

C++ поддерживает следующие основные арифметические операторы:



**Пример использования:**



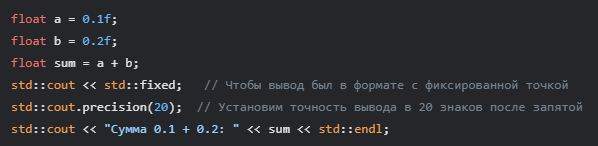
**Обратите внимание:**

* Деление целых чисел (int) в C++ является целочисленным. Это означает, что дробная часть результата отбрасывается. Например, 10 / 3 равно 3, а не 3.33.
* Оператор % (модуль) возвращает остаток от деления. Он работает только с целыми числами.

**7.2. Некоторые особенности при работе с числами с плавающей точкой**

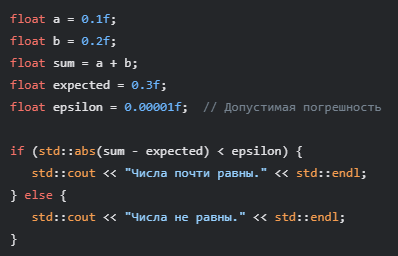
Числа с плавающей точкой (float и double) имеют особенности, которые нужно учитывать при выполнении арифметических операций:

* **Неточность представления:** Числа с плавающей точкой представляются в памяти компьютера с ограниченной точностью. Это может привести к небольшим ошибкам округления при выполнении вычислений.



Результат может быть не точно 0.3, а, например, 0.30000001192092896.

* **Сравнение чисел с плавающей точкой:** Из-за неточности представления, не рекомендуется сравнивать числа с плавающей точкой на равенство (==). Вместо этого лучше использовать сравнение с некоторой погрешностью (эпсилон).



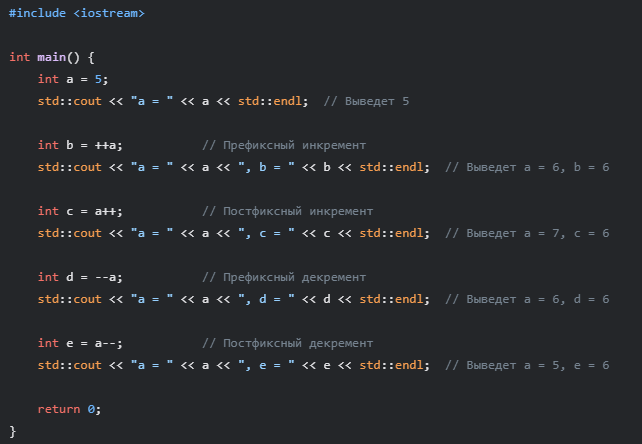
std::abs() - функция для вычисления модуля числа.

* **Деление на ноль:** Деление на ноль (/ 0.0f или / 0.0) для чисел с плавающей точкой не вызывает ошибку во время выполнения (как при целочисленном делении). В результате деления на ноль получается специальное значение "бесконечность" (inf) или "не число" (NaN).

**7.3. Инкремент и декремент**

Операторы инкремента (++) и декремента (--) используются для увеличения и уменьшения значения переменной на 1, соответственно. Они бывают двух видов:

* **Префиксный (++a, --a):** Сначала изменяет значение переменной, затем возвращает новое значение.
* **Постфиксный (a++, a--):** Сначала возвращает старое значение, затем изменяет значение переменной.

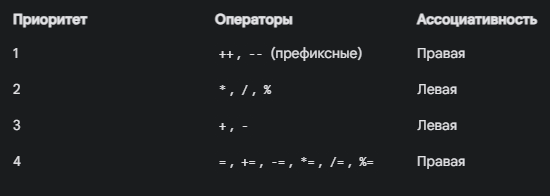


**7.4. Приоритет и ассоциативность операторов**

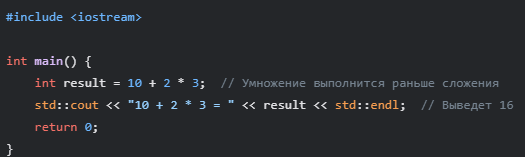
Приоритет и ассоциативность операторов определяют порядок выполнения операций в выражении.

* **Приоритет:** Операторы с более высоким приоритетом выполняются раньше операторов с более низким приоритетом. Например, умножение и деление имеют более высокий приоритет, чем сложение и вычитание.
* **Ассоциативность:** Определяет порядок выполнения операторов с одинаковым приоритетом. Операторы могут быть левоассоциативными (выполняются слева направо) или правоассоциативными (выполняются справа налево).

Вот таблица приоритетов и ассоциативности некоторых операторов (от высшего к низшему):

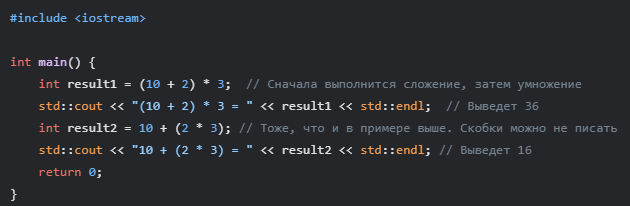


**Пример:**



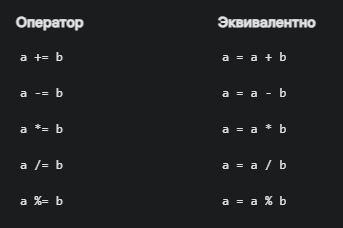
**7.5. Переопределение порядка операций**

Для переопределения порядка операций можно использовать круглые скобки (). Выражения в скобках всегда выполняются раньше операций вне скобок.

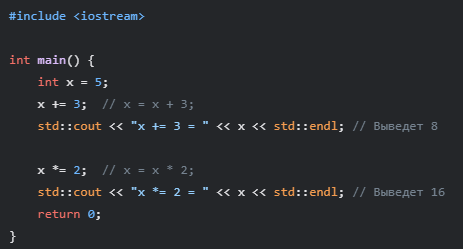


**7.6. Составные операторы присваивания**

Для удобства и краткости можно использовать составные операторы присваивания, которые выполняют операцию над переменной и присваивают ей результат:



**Пример:**



**7.7. Закрепление материала**

1. Перечисли основные арифметические операторы в C++.
2. В чем отличие целочисленного деления от деления чисел с плавающей точкой?
3. Почему при сравнении чисел с плавающей точкой не рекомендуется использовать оператор ==?
4. Чем отличается префиксный инкремент от постфиксного?
5. Что такое приоритет и ассоциативность операторов?
6. Как переопределить порядок выполнения операций?
7. Для чего используются составные операторы присваивания?

**Практическое задание:**

1. Напиши программу, которая запрашивает у пользователя два числа и выполняет все основные арифметические операции.
2. Напиши программу, которая вычисляет площадь и периметр круга, используя константу PI.
3. Поэкспериментируй с числами с плавающей точкой и посмотри, как работают особенности их представления.
4. Используй инкремент и декремент в разных ситуациях и посмотри, как это влияет на результат.
5. Напиши программу, где нужно переопределить порядок выполнения операций, используя скобки.

**Глава 8: Статическая типизация и преобразования типов в C++**

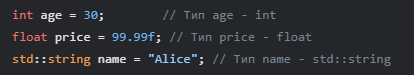
C++ — это язык со статической типизацией. Это означает, что тип каждой переменной должен быть известен на этапе компиляции. Компилятор проверяет типы данных во время компиляции и выдает ошибку, если обнаруживает несоответствие типов.

**8.1. Что такое статическая типизация?**

Статическая типизация обеспечивает следующие преимущества:

* **Безопасность:** Компилятор проверяет типы данных, что позволяет обнаружить ошибки, связанные с неправильным использованием типов, на этапе компиляции, а не во время выполнения программы.
* **Производительность:** Компилятор может оптимизировать код, основываясь на информации о типах данных.
* **Читаемость и понятность:** Явно указанные типы данных делают код более понятным и легким для восприятия.

В отличие от языков с динамической типизацией (например, Python), где тип переменной определяется во время выполнения, в C++ тип переменной объявляется явно при ее создании.



**8.2. Преобразования типов (Type Conversions)**

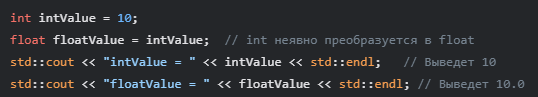
Преобразование типов — это процесс изменения типа данных значения. Иногда возникает необходимость преобразовать значение из одного типа в другой. В C++ существуют два основных вида преобразования типов:

1. **Неявные преобразования (Implicit conversions)**:  
   Это преобразования, которые компилятор выполняет автоматически без явного указания со стороны программиста. Неявные преобразования обычно выполняются, когда значение одного типа используется в контексте, где ожидается значение другого типа.
2. **Явные преобразования (Explicit conversions)**:  
   Это преобразования, которые программист должен явно указать в коде. Явные преобразования используются, когда компилятор не может автоматически выполнить преобразование или когда нужно явно контролировать процесс преобразования.

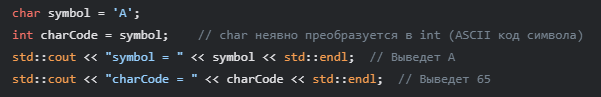
**8.3. Неявные преобразования**

Неявные преобразования выполняются автоматически, когда компилятор считает, что это безопасно. Примеры неявных преобразований:

* **Преобразование целых чисел в числа с плавающей точкой:**



* **Преобразование символов в целые числа:**

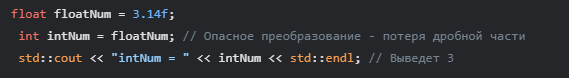


* **Преобразования в арифметических операциях:**  
  При выполнении арифметических операций с операндами разных типов, компилятор автоматически выполняет преобразование типа с меньшей точностью в тип с большей точностью.
* Правило преобразования типов при арифметических операциях: bool -> char -> short -> int -> long -> long long -> float -> double -> long double

**8.4. Опасные и безопасные преобразования**

Некоторые преобразования типов являются безопасными (то есть не приводят к потере данных), а другие — опасными (могут привести к потере данных или непредсказуемым результатам).

* **Безопасные преобразования:**
  + Преобразование целого числа в число с плавающей точкой.
  + Преобразование типа с меньшей точностью в тип с большей точностью (например, int в double).
* **Опасные преобразования:**
  + Преобразование числа с плавающей точкой в целое число (дробная часть отбрасывается).



* + Преобразование типа с большей точностью в тип с меньшей точностью (например, double в int).
  + Преобразование целых чисел в символы, когда значение целого числа выходит за пределы диапазона символов.

**8.5. Явные преобразования типов (Type Casting)**

Явные преобразования типов позволяют программисту явно контролировать процесс преобразования, в тех случаях когда неявные преобразования не подходят. В C++ существует несколько способов выполнить явное преобразование типов:

1. **C-style cast (преобразование в стиле C):**  
   Этот способ унаследован от языка C, его лучше избегать из-за его опасности.



Пример:



1. **static\_cast:**  
   Этот оператор следует использовать для безопасных и логичных преобразований.



Пример:



1. **dynamic\_cast:**  
   Этот оператор используется для преобразования указателей и ссылок на объекты классов в иерархии наследования. Он проверяет корректность преобразования во время выполнения. Мы обсудим dynamic\_cast позже при изучении ООП.
2. **const\_cast:**  
   Этот оператор используется для изменения const или volatile квалификаторов. Необходимо применять с осторожностью.
3. **reinterpret\_cast:**  
   Этот оператор используется для низкоуровневых преобразований, которые могут привести к непредсказуемым результатам. Необходимо использовать с осторожностью.

**Примеры явных преобразований:**



**Важно:**

* Используйте static\_cast для большинства преобразований типов, так как он безопаснее C-style cast.
* dynamic\_cast используется для преобразования типов в иерархии наследования.
* const\_cast используется для изменения const и volatile квалификаторов (используйте с осторожностью).
* reinterpret\_cast используется для низкоуровневых преобразований (используйте с осторожностью).

**8.6. Закрепление материала**

1. Что такое статическая типизация?
2. В чем ее преимущества?
3. Какие типы преобразований ты знаешь?
4. Что такое неявные преобразования и когда они выполняются?
5. Какие преобразования типов считаются безопасными, а какие — опасными?
6. Какие явные преобразования типов в C++ ты знаешь и когда их нужно применять?
7. Почему рекомендуется использовать static\_cast вместо C-style cast?

**Практическое задание:**

1. Напиши программу, которая демонстрирует неявные преобразования при арифметических операциях.
2. Попробуй разные явные преобразования типов (безопасные и опасные).
3. Исследуй, как изменяются значения при разных преобразованиях.
4. Напиши программу, которая запрашивает у пользователя число с плавающей точкой, а затем выводит его целочисленное значение (с отбрасыванием дробной части) и округленное значение (используя функцию std::round).

**Глава 9: Поразрядные операции в C++**

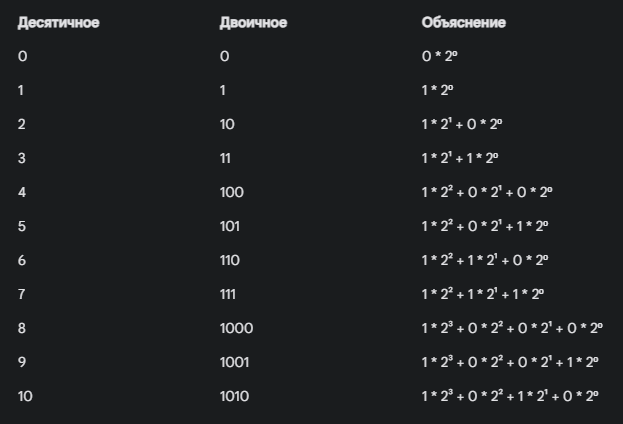
Поразрядные операции позволяют выполнять манипуляции с битами данных, то есть с отдельными нулями и единицами, составляющими двоичное представление чисел.

**9.1. Двоичное представление чисел**

В компьютере вся информация, включая числа, хранится в двоичной системе счисления, которая использует только два символа: 0 и 1. Каждый такой символ называется *битом*.

* **Десятичная система:** Основание 10, использует цифры от 0 до 9.
* **Двоичная система:** Основание 2, использует цифры 0 и 1.

Чтобы понять, как числа представляются в двоичной системе, рассмотрим примеры:



Для того чтобы представить более крупные числа в двоичном виде, добавляются новые разряды (биты).

**9.2. Представление отрицательных чисел**

Для представления отрицательных чисел обычно используется *дополнительный код* (two's complement).

1. **Инвертирование битов:** Сначала все биты положительного числа инвертируются (0 заменяются на 1, а 1 на 0).
2. **Прибавление 1:** Затем к результату прибавляется 1.

Например, представим число -5 в 8-битном формате:

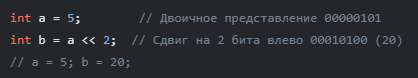
1. Положительное число 5: 00000101
2. Инвертирование битов: 11111010
3. Прибавление 1: 11111011

Таким образом, -5 представляется как 11111011.

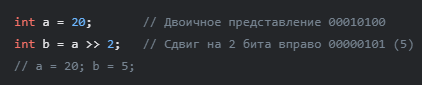
**9.3. Операции сдвига**

Операции сдвига перемещают биты числа влево или вправо:

* **Сдвиг влево (<<):** Сдвигает биты числа влево на заданное количество позиций. В освободившиеся биты справа добавляются нули. Эквивалентно умножению на 2 в степени сдвига.

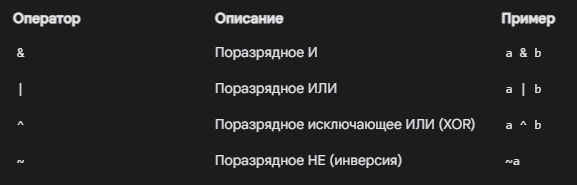


* **Сдвиг вправо (>>):** Сдвигает биты числа вправо на заданное количество позиций. В освободившиеся биты слева добавляются нули (для беззнаковых чисел) или сохраняется знаковый бит (для знаковых чисел). Эквивалентно целочисленному делению на 2 в степени сдвига.

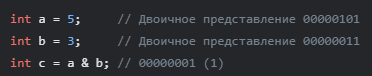


**9.4. Поразрядные операции**

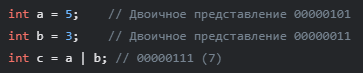
C++ поддерживает следующие поразрядные операторы:



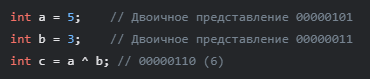
* **Поразрядное И (&):** Возвращает 1 в том бите, если оба соответствующих бита операндов равны 1, иначе 0.



* **Поразрядное ИЛИ (|):** Возвращает 1 в том бите, если хотя бы один из соответствующих битов операндов равен 1, иначе 0.



* **Поразрядное исключающее ИЛИ (^):** Возвращает 1 в том бите, если соответствующие биты операндов отличаются, иначе 0.



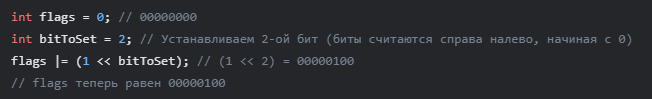
* **Поразрядное НЕ (~):** Инвертирует все биты числа (0 становится 1, а 1 становится 0).



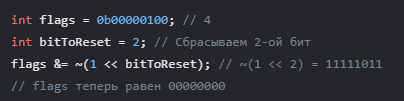
**9.5. Пример практического применения операций**

Давай рассмотрим несколько примеров, где поразрядные операции могут быть полезными:

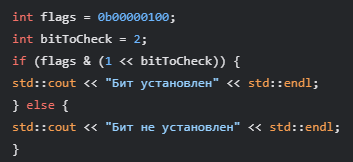
* **Установка бита:** Чтобы установить определённый бит в 1, можно использовать поразрядное ИЛИ.



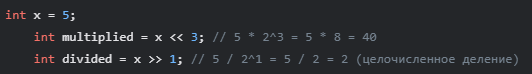
* **Сброс бита:** Чтобы сбросить определённый бит в 0, можно использовать поразрядное И с инвертированным значением.



* **Проверка состояния бита:** Чтобы проверить, установлен ли бит в 1, можно использовать поразрядное И.



* **Быстрое умножение и деление на степени 2:** Сдвиг влево и вправо могут быть использованы для быстрого умножения и деления на степени 2, но используйте это с осторожностью, так как при сдвиге может произойти переполнение.



**9.6. Закрепление материала**

1. Как числа представляются в двоичной системе?
2. Как представляются отрицательные числа с помощью дополнительного кода?
3. Что делают операторы сдвига влево и вправо?
4. Перечисли поразрядные операторы и опиши, что они делают.
5. Приведи примеры практического применения поразрядных операций.
6. Напиши функцию, которая устанавливает заданный бит в числе.
7. Напиши функцию, которая проверяет, установлен ли заданный бит в числе.

**Практическое задание:**

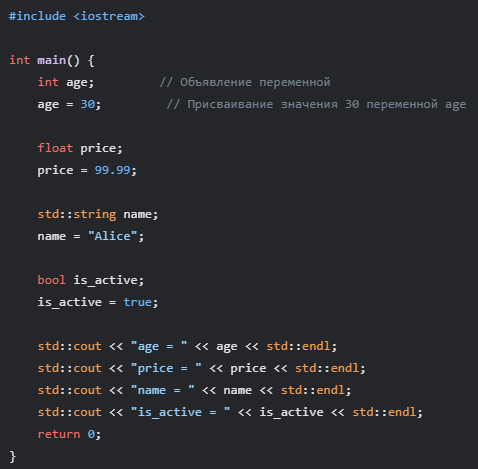
1. Напиши программу, которая преобразует десятичное число в двоичное представление.
2. Напиши программу, которая выполняет поразрядные операции (&, |, ^, ~) над двумя введенными числами.
3. Напиши программу, которая меняет местами значения двух целых чисел, используя поразрядное исключающее ИЛИ.
4. Попробуй использовать поразрядные операции для решения других задач, например, проверки чётности числа или выделения определенных битов из числа.

**Глава 10: Операции присваивания в C++**

Операция присваивания – это способ присвоить значение переменной. В C++ используется оператор =, который копирует значение, расположенное справа от него, в переменную, расположенную слева.

**10.1. Простой оператор присваивания =**

Простой оператор присваивания = является основным инструментом для присваивания значений переменным.



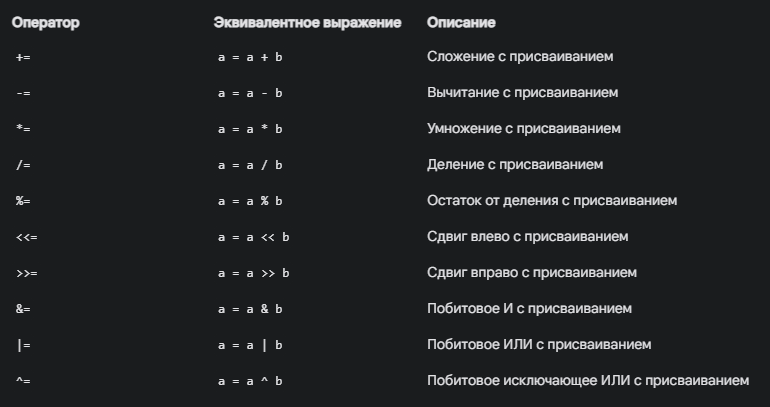
**Основные моменты:**

* Оператор присваивания = копирует значение из правой части в левую.
* Тип значения в правой части должен быть совместим с типом переменной в левой части.
* Оператор присваивания можно использовать для инициализации переменной при объявлении:



**10.2. Составные операторы присваивания**

C++ предоставляет составные операторы присваивания, которые сочетают в себе арифметическую операцию и присваивание. Это позволяет сократить запись кода и сделать его более читаемым.



**Примеры использования:**

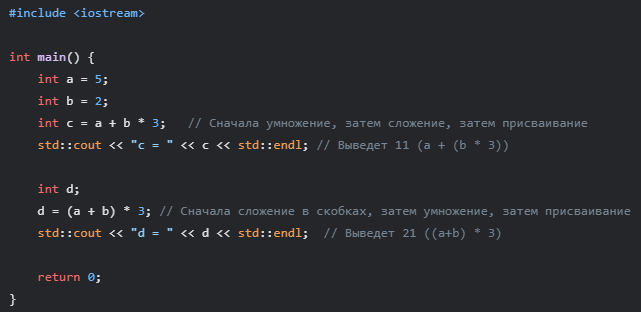


**Основные моменты:**

* Составные операторы присваивания позволяют сократить запись кода, выполняя операцию над переменной и присваивая результат ей же.
* Они делают код более читаемым и понятным.

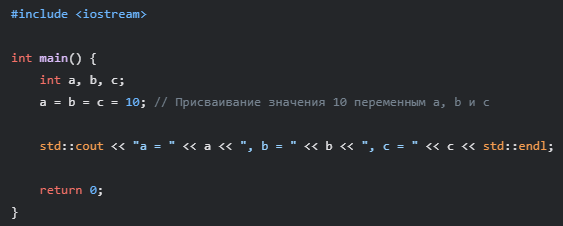
**10.3. Присваивание и порядок выполнения операций**

Оператор присваивания имеет самый низкий приоритет среди всех операторов. Это означает, что все вычисления в правой части выполняются до того, как значение будет присвоено переменной в левой части.



**10.4. Присваивание значений в цепочке**

В C++ возможно присваивать значения в цепочке:



В этом случае присваивание выполняется справа налево. Сначала 10 присваивается переменной c, затем значение c присваивается переменной b, затем значение b присваивается переменной a.

**10.5. Инициализация и присваивание**

Важно отличать инициализацию от присваивания.

* **Инициализация** выполняется при объявлении переменной и задает ей начальное значение.



* **Присваивание** выполняется после объявления переменной и изменяет ее значение.



**10.6. Закрепление материала**

1. Что делает оператор присваивания =?
2. Перечисли составные операторы присваивания.
3. В чем преимущество составных операторов присваивания?
4. Какой приоритет имеет оператор присваивания?
5. Как можно присвоить значения в цепочке?
6. В чем разница между инициализацией и присваиванием?

**Практическое задание:**

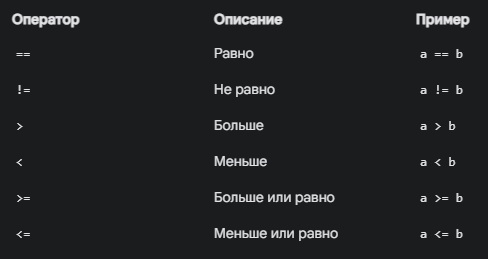
1. Напиши программу, которая использует все составные операторы присваивания.
2. Напиши программу, которая запрашивает у пользователя начальное значение и затем несколько раз изменяет его с помощью разных составных операторов присваивания.
3. Напиши программу, которая показывает порядок выполнения операций при присваивании.
4. Разберись с примерами с побитовыми операциями, как они работают с присваиваниями.

**Глава 11: Условные выражения в C++**

Условные выражения используются для сравнения значений и принятия решений на основе результатов этих сравнений. Они являются основой для управления потоком выполнения программы, позволяя ей выполнять различные действия в зависимости от условий.

**11.1. Операции сравнения**

Операции сравнения используются для сравнения двух значений и возвращают логическое значение true или false в зависимости от результата сравнения.



**Примеры использования:**

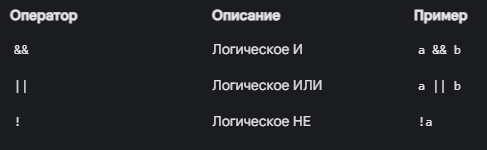


**Основные моменты:**

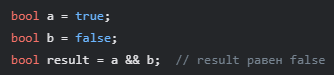
* Операции сравнения возвращают логические значения (true или false).
* Необходимо отличать оператор присваивания = от оператора сравнения ==.
* При сравнении чисел с плавающей точкой нужно учитывать их неточность представления. Лучше сравнивать с погрешностью.
* std::boolalpha — это манипулятор вывода, который выводит логические значения как true и false вместо 1 и 0.

**11.2. Логические операции**

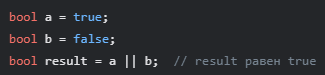
Логические операторы используются для объединения нескольких условий в более сложные выражения. C++ поддерживает следующие логические операторы:



* **Логическое И (&&):** Возвращает true только в том случае, если оба операнда true, иначе false.



* **Логическое ИЛИ (||):** Возвращает true, если хотя бы один из операндов true, иначе false.



* **Логическое НЕ (!):** Инвертирует значение операнда. Если операнд true, то результат false, и наоборот.



**Примеры использования:**



**11.3. Приоритет и ассоциативность операторов**

При выполнении условных выражений необходимо учитывать приоритет и ассоциативность операторов:



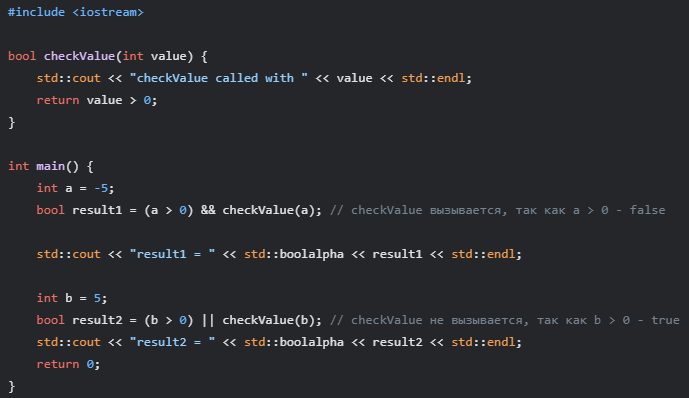
Оператор присваивания имеет самый низкий приоритет, поэтому сначала вычисляются условные выражения.

**11.4. Короткое вычисление**

Логические операторы && и \|\| используют *короткое вычисление* (short-circuit evaluation). Это означает, что второй операнд вычисляется только в том случае, если это необходимо для определения результата всего выражения.

* При использовании && если первый операнд false, то второй операнд не вычисляется, так как результат выражения будет false независимо от значения второго операнда.
* При использовании || если первый операнд true, то второй операнд не вычисляется, так как результат выражения будет true независимо от значения второго операнда.

Это может быть полезно для оптимизации кода и предотвращения ошибок.



**11.5. Закрепление материала**

1. Перечисли операторы сравнения и опиши, что они делают.
2. Какие логические операторы есть в C++?
3. Что такое короткое вычисление и как оно работает?
4. Каков приоритет логических операторов?
5. В чем разница между операторами = и ==?

**Практическое задание:**

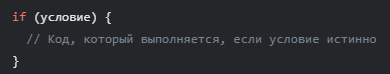
1. Напиши программу, которая запрашивает у пользователя два числа и проверяет, равны ли они, и какое из них больше.
2. Напиши программу, которая определяет, является ли введенное число положительным, отрицательным или нулем.
3. Напиши программу, которая определяет, является ли введенный год високосным. (Год является високосным, если он делится на 4, но не делится на 100, либо делится на 400.)
4. Напиши программу, которая определяет, попадает ли введенное число в заданный диапазон (например, от 10 до 100).
5. Используй логические операторы для создания сложных условий.

**Глава 12: Конструкция if-else и тернарный оператор в C++**

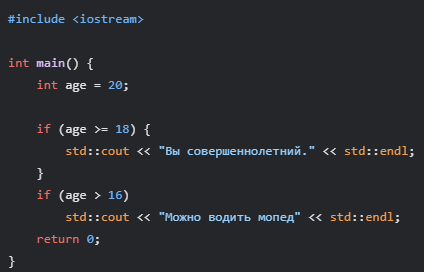
В этой главе мы подробно рассмотрим оператор if, оператор if-else, вложенные конструкции if-else и тернарный оператор, и узнаем, как их правильно использовать в разных ситуациях.

**12.1. Оператор if**

Оператор if позволяет выполнить блок кода только в том случае, если заданное условие истинно (true).

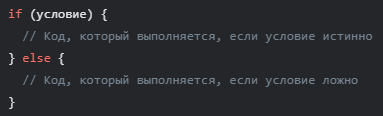


* условие: Выражение, которое возвращает логическое значение (true или false).
* { ... }: Блок кода, который выполняется, если условие true. Если блок состоит из одной инструкции, то фигурные скобки можно опустить.

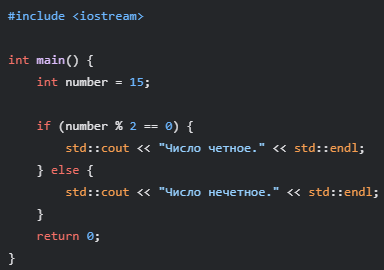


**12.2. Оператор if-else**

Оператор if-else позволяет выполнить один блок кода, если условие истинно (true), и другой блок кода, если условие ложно (false).

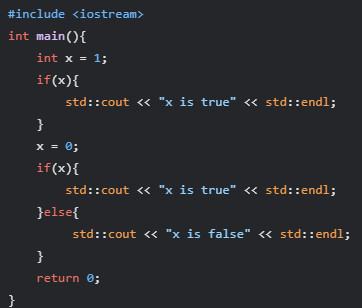


* условие: Выражение, которое возвращает логическое значение (true или false).
* { ... }: Блок кода, который выполняется, если условие true (после if).
* { ... }: Блок кода, который выполняется, если условие false (после else).



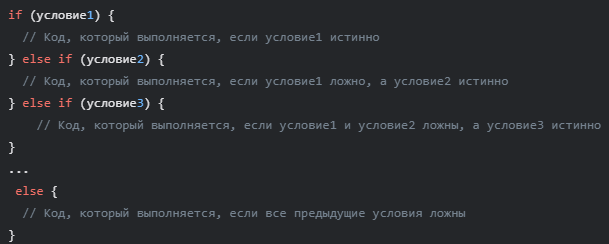
**12.3. Целочисленные условия**

В С++ любое целое число можно использовать в качестве условия. При этом 0 рассматривается как false, а любое другое целое число рассматривается как true.

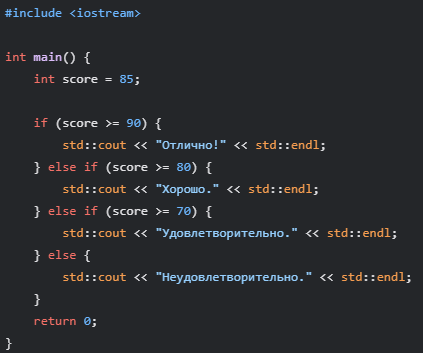


**12.4. Конструкция if-else if-else**

Конструкция if-else if-else позволяет проверять несколько условий последовательно. Если одно из условий выполняется, то соответствующий блок кода выполняется, а остальные условия не проверяются.

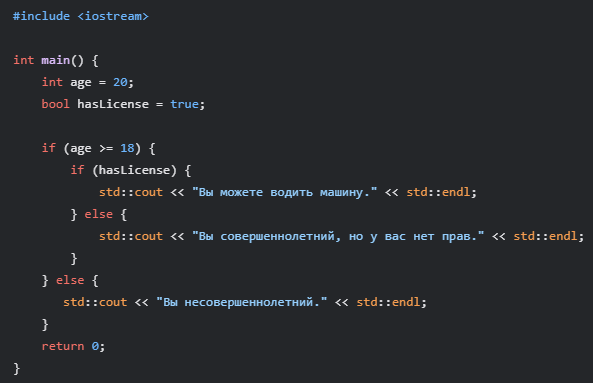


* Можно использовать любое количество блоков else if.
* Блок else является необязательным.



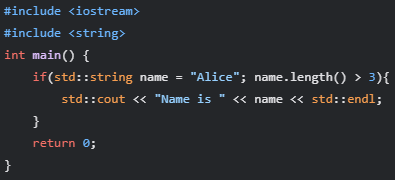
**12.5. Вложенные конструкции if-else**

Можно вкладывать операторы if-else друг в друга для создания более сложных условий.



**12.6. Блок if с инициализацией переменной**

Начиная со стандарта C++17, в блоке if можно не только задать условие, но и объявить переменную, видимую только внутри этого блока.

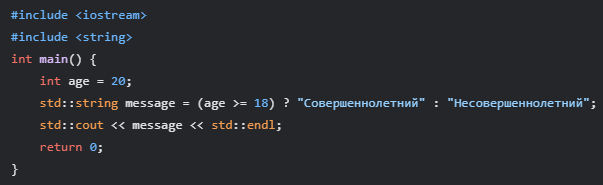


**12.7. Тернарный оператор**

Тернарный оператор ? : представляет собой сокращенную форму оператора if-else для простых условных выражений.



* Если условие истинно, то возвращается значение выражение1, иначе возвращается значение выражение2.



**Основные моменты:**

* Тернарный оператор подходит только для простых выражений.
* Значения выражение1 и выражение2 должны быть совместимы по типам.

**12.8. Закрепление материала**

1. Как работает оператор if?
2. Как работает оператор if-else?
3. Для чего нужна конструкция if-else if-else?
4. Можно ли вкладывать операторы if-else друг в друга?
5. Для чего используется тернарный оператор?
6. Когда лучше использовать тернарный оператор, а когда лучше использовать if-else?

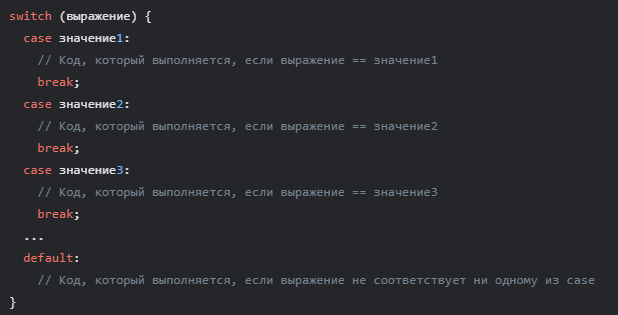
**Практическое задание:**

1. Напиши программу, которая запрашивает у пользователя число и проверяет, является ли оно положительным, отрицательным или нулем. Используй операторы if-else if-else.
2. Напиши программу, которая определяет максимальное из трех введенных чисел. Используй вложенные if-else.
3. Напиши программу, которая определяет, является ли введенный символ гласной или согласной буквой (в английском алфавите).
4. Напиши программу, которая использует тернарный оператор для определения четности введенного числа.
5. Напиши программу, которая имитирует простую систему авторизации.
6. Попробуй использование блока if с инициализацией переменной.

**Глава 13: Конструкция switch-case в C++**

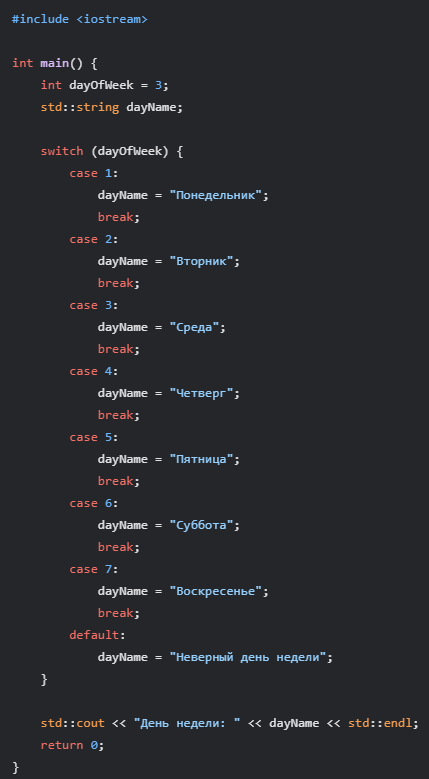
Оператор switch-case используется для выбора одного из нескольких блоков кода для выполнения в зависимости от значения целочисленного выражения или выражения, приводимого к целочисленному типу.

**13.1. Базовая структура оператора switch-case**



* switch (выражение): Выражение должно быть целочисленным типом (например, int, char, enum), или типом, приводимым к целочисленному.
* case значение: Константа или константное выражение целочисленного типа.
* break;: Оператор break завершает выполнение switch и передает управление следующей инструкции после switch.
* default:: Необязательный блок кода, который выполняется, если ни один из case не соответствует значению выражения.

**Пример использования:**



**Основные моменты:**

* Выполнение начинается с case, значение которого совпадает со значением выражения в switch.
* Если break не используется, выполнение "проваливается" к следующему case (или default), что может привести к непредсказуемому поведению.
* Блок default выполняется, если ни один из case не подходит.

**13.2. Совмещение условий**

Можно объединить несколько case для выполнения одного и того же блока кода, убрав оператор break между ними.



**Основные моменты:**

* break нужно ставить только после того, как мы хотим завершить выполнение блока кода.

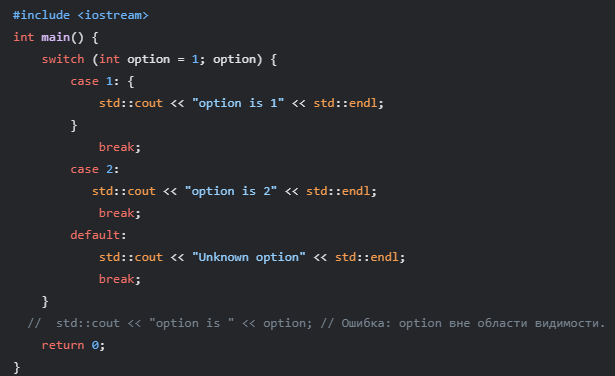
**13.3. Переменные в блоках case**

Внутри каждого блока case можно объявлять переменные, но их область видимости ограничена только этим блоком. Если вам нужна переменная, доступная во всех блоках case, объявите ее перед switch.

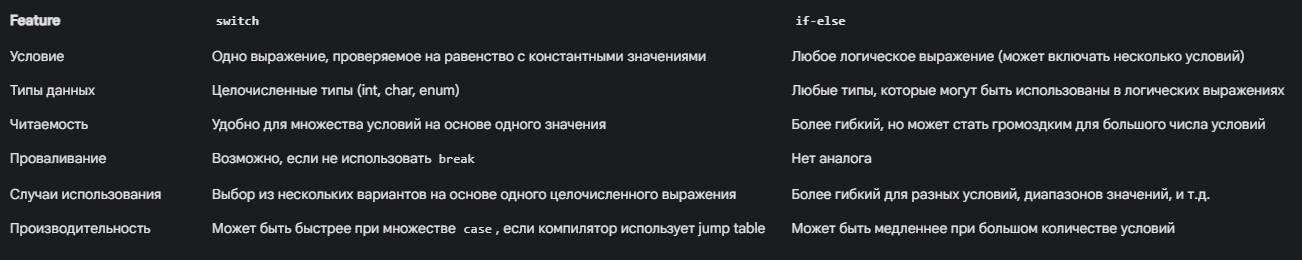


**13.4. switch с инициализацией переменной**

Начиная со стандарта C++17, в операторе switch можно объявлять переменную, которая будет видна только внутри этого оператора.



**13.5. Сравнение switch и if-else**



**13.6. Закрепление материала**

1. Как работает оператор switch-case?
2. Для чего нужен оператор break в switch-case?
3. Можно ли объединять несколько case в switch-case?
4. Где можно объявлять переменные в блоках case и в чем особенность их области видимости?
5. Что такое jump table?
6. В чем отличие switch-case от if-else if-else?
7. Когда лучше использовать switch-case, а когда if-else?

**Практическое задание:**

1. Напиши программу, которая запрашивает у пользователя номер месяца и выводит название месяца.
2. Напиши программу, которая имитирует простой калькулятор. Пользователь вводит два числа и символ операции (например, +, -, \*, /), а программа выводит результат.
3. Напиши программу, которая преобразует оценку в виде буквы (A, B, C, D, F) в ее словесное описание (например, "отлично", "хорошо" и т.д.).
4. Попробуй создать блок switch с инициализацией переменной.

**Глава 14: Циклы в C++**

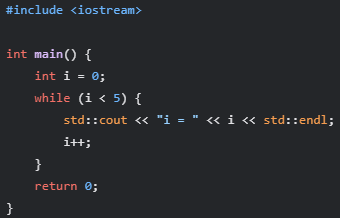
Циклы используются для многократного выполнения одного и того же блока кода. В C++ есть три основных вида циклов: while, for и do...while.

**14.1. Цикл while**

Цикл while выполняет блок кода до тех пор, пока заданное условие истинно.



* условие: Выражение, которое возвращает логическое значение (true или false).
* { ... }: Блок кода, который выполняется, пока условие true.

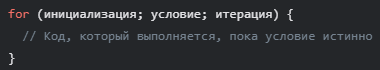


**Основные моменты:**

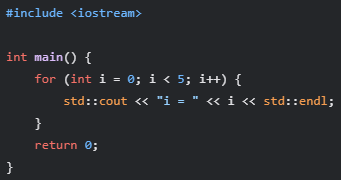
* Перед выполнением тела цикла всегда проверяется условие.
* Если условие ложно (false) с самого начала, тело цикла не выполнится ни разу.
* Внутри тела цикла необходимо изменять переменные, от которых зависит условие, чтобы цикл в конечном итоге завершился.

**14.2. Цикл for**

Цикл for используется для повторяющегося выполнения блока кода определенное количество раз.



* инициализация: Выполняется один раз перед началом цикла. Обычно используется для объявления и инициализации счетчика.
* условие: Выражение, которое возвращает логическое значение (true или false). Цикл выполняется, пока условие true.
* итерация: Выполняется после каждой итерации цикла. Обычно используется для изменения счетчика.
* { ... }: Блок кода, который выполняется, пока условие true.

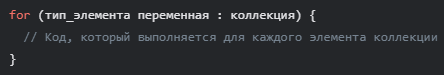


**Основные моменты:**

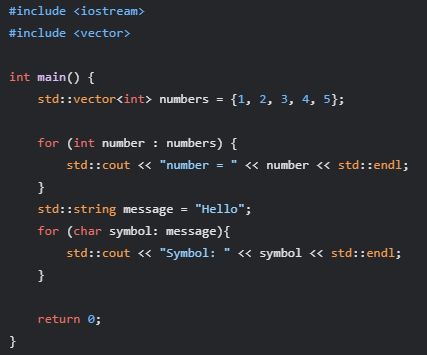
* Цикл for удобен, когда известно количество итераций.
* Все три части цикла (инициализация, условие, итерация) могут быть пустыми, но точки с запятой (;) остаются.
* Область видимости переменной, объявленной в блоке инициализации цикла for, ограничена телом цикла.

**14.3. Перебор значений в стиле for-each (цикл на основе диапазона)**

Начиная с C++11, появился цикл for, который позволяет перебирать элементы коллекций, массивов и строк.

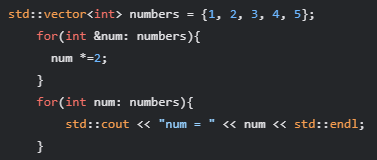


* тип\_элемента: Тип элементов в коллекции. Можно использовать auto для автоматического определения типа.
* переменная: Переменная, которая будет содержать текущий элемент коллекции.
* коллекция: Коллекция (например, массив, std::vector, строка), элементы которой нужно перебрать.



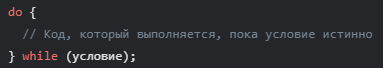
**Основные моменты:**

* Цикл for-each удобен для перебора всех элементов коллекции.
* Он более лаконичен и читаемый, чем обычный цикл for для перебора элементов коллекций.
* Можно использовать ссылку & для изменения значений элементов:

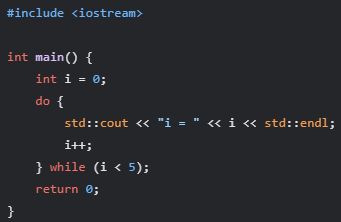


**14.4. Цикл do...while**

Цикл do...while похож на цикл while, но проверяет условие после выполнения блока кода. Это гарантирует, что блок кода будет выполнен хотя бы один раз.



* { ... }: Блок кода, который выполняется.
* условие: Выражение, которое возвращает логическое значение (true или false).

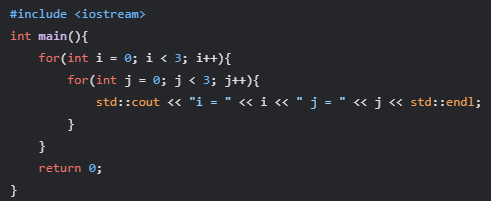


**Основные моменты:**

* Тело цикла do...while выполняется хотя бы один раз.
* Условие проверяется после выполнения тела цикла.

**14.5. Вложенные циклы for**

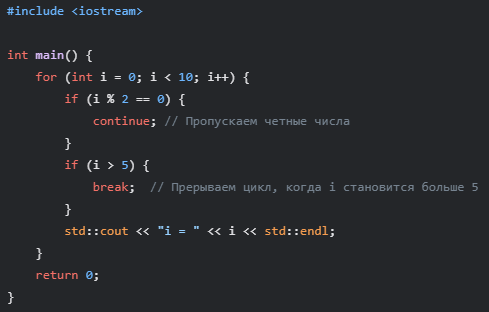
Можно вкладывать циклы друг в друга. Это позволяет, например, перебирать элементы двумерных массивов.



**14.6. Операторы continue и break**

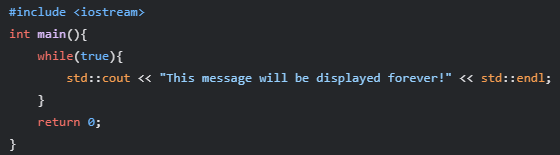
Операторы continue и break позволяют управлять выполнением циклов:

* continue: Завершает текущую итерацию цикла и переходит к следующей итерации.
* break: Прерывает выполнение цикла и передает управление следующей инструкции после цикла.



**14.7. Бесконечные циклы**

Цикл, условие которого всегда истинно, называется *бесконечным*. Бесконечные циклы часто используются в играх, операционных системах и других программах, где требуется непрерывное выполнение.



* Для того, что прервать работу бесконечного цикла обычно нужно использовать break внутри тела цикла.

**14.8. Закрепление материала**

1. Какие виды циклов есть в C++?
2. В чем отличие цикла while от цикла do...while?
3. Когда лучше использовать цикл for, а когда while или do...while?
4. Что делает цикл for-each?
5. Для чего нужны операторы continue и break?
6. Что такое бесконечный цикл?

**Практическое задание:**

1. Напиши программу, которая выводит все числа от 1 до 100, используя циклы while и for.
2. Напиши программу, которая запрашивает у пользователя число и выводит таблицу умножения для этого числа.
3. Напиши программу, которая вычисляет факториал введенного числа с помощью цикла for.
4. Напиши программу, которая запрашивает у пользователя числа и суммирует их до тех пор, пока пользователь не введет 0. Используй цикл do...while.
5. Напиши программу, которая перебирает элементы массива и выводит только четные числа, используя цикл for-each.
6. Напиши программу, которая выводит на экран "ёлочку" из символов, используя вложенные циклы for.
7. Напиши бесконечный цикл с выходом по условию.

**Глава 15: Ссылки в C++**

Ссылка — это альтернативное имя для существующей переменной. Представьте себе ссылку как псевдоним или другое имя, которое указывает на ту же область памяти, что и исходная переменная.

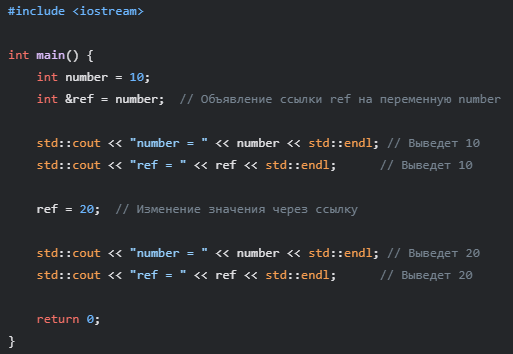
**15.1. Объявление ссылки**

Для объявления ссылки используется символ & после типа данных:



* тип\_данных: Тип данных переменной, на которую ссылается ссылка.
* &: Символ, который указывает, что это ссылка.
* имя\_ссылки: Имя ссылки.
* имя\_переменной: Имя существующей переменной, на которую ссылается ссылка.

**Пример:**



**Основные моменты:**

* Ссылка должна быть инициализирована при объявлении.
* Ссылка не является отдельной переменной. Она является другим именем для существующей переменной.
* Изменение значения через ссылку влияет на значение исходной переменной.
* После инициализации ссылку нельзя перенаправить на другую переменную.
* Ссылки не могут быть nullptr (null pointer).
* Ссылки могут быть только на левосторонние значения (lvalue).

**15.2. Ссылки как параметры функций**

Ссылки часто используются в качестве параметров функций для передачи аргументов по ссылке, а не по значению. Это позволяет функции изменять значения переменных, переданных в качестве аргументов.

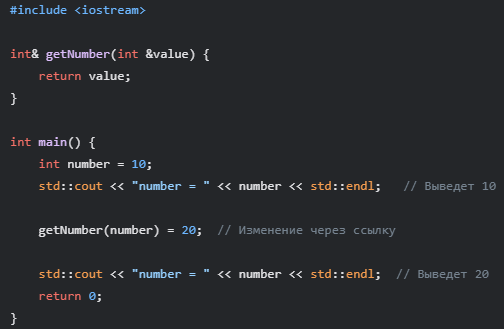


**Основные моменты:**

* Когда функция принимает аргумент по ссылке, функция работает с исходной переменной, а не с ее копией.
* Изменения параметра-ссылки в функции повлияют на исходную переменную.
* Это позволяет функциям возвращать более одного значения (через изменение параметров, переданных по ссылке).

**15.3. Ссылки как возвращаемое значение функции**

Функция может возвращать ссылку. Это позволяет избежать создания копии возвращаемого значения, но это нужно использовать с осторожностью.

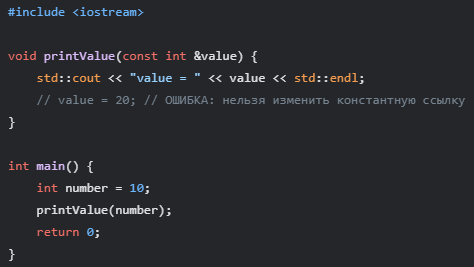


**Основные моменты:**

* Функция возвращает ссылку на исходную переменную, а не ее копию.
* Это позволяет изменять исходную переменную через возвращаемую ссылку.
* Следует избегать возвращения ссылок на локальные переменные из функции, так как они перестанут существовать после выхода из функции, что приведет к неопределенному поведению.

**15.4. Константные ссылки**

Константная ссылка (const &) позволяет ссылаться на переменную, не изменяя ее значение. Это полезно для передачи аргументов в функции, если функция не должна изменять их значения.

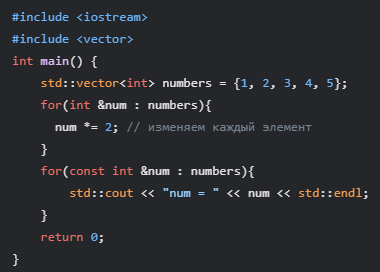


**Основные моменты:**

* Константная ссылка может ссылаться как на константную, так и на не константную переменную.
* Значение переменной, на которую ссылается константная ссылка, не может быть изменено через эту ссылку.

**15.5. Ссылки в цикле for**

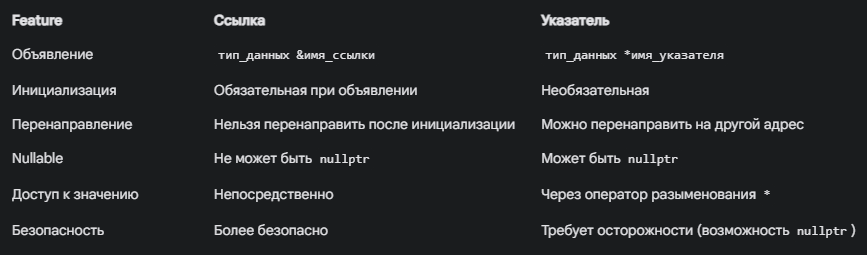
Ссылки особенно полезны в циклах for-each при переборе элементов коллекций или массивов. Использование ссылок позволяет изменять элементы коллекции, а также избежать лишнего копирования элементов при переборе.



* Если нет необходимости изменять значение элемента в цикле, лучше использовать const & для предотвращения случайного изменения.

**15.6. Различия между ссылками и указателями**

Ссылки и указатели похожи в том, что они оба позволяют косвенно обращаться к данным. Однако между ними есть важные отличия:



**15.7. Закрепление материала**

1. Что такое ссылка в C++?
2. Как объявить ссылку?
3. Почему ссылка должна быть инициализирована при объявлении?
4. Чем ссылка отличается от переменной?
5. Как использовать ссылки в качестве параметров функций?
6. Как использовать ссылки в качестве возвращаемых значений функций?
7. Что такое константная ссылка и когда ее нужно использовать?
8. Чем отличаются ссылки от указателей?
9. Как использовать ссылки в цикле for-each?

**Практическое задание:**

1. Напиши программу, которая демонстрирует работу с ссылками и их влияние на исходные переменные.
2. Напиши функцию, которая меняет местами значения двух целых чисел, используя ссылки.
3. Напиши функцию, которая принимает массив чисел и удваивает их значения, используя ссылки.
4. Напиши функцию, которая принимает строку и выводит ее на экран, используя константную ссылку.
5. Напиши программу, которая использует ссылки в цикле for-each для перебора элементов вектора.