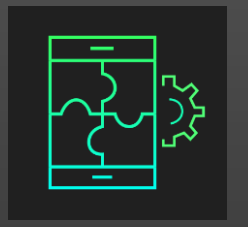
If else



Конструкция if else в C# — это условная конструкция, позволяющая выполнять разные блоки кода в зависимости от заданного условия.

if - выполняет код, если условие истинно (true).

else if: Позволяет проверить дополнительные условия, если предшествующий if не сработал.

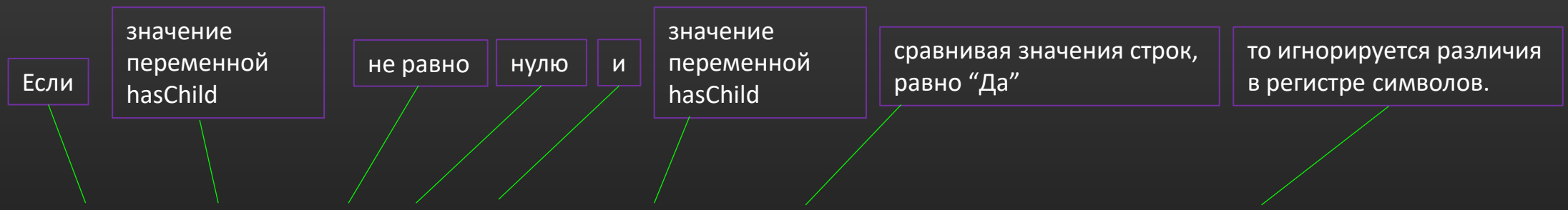
else - выполняет код, если условие ложно (false).

```
if (hasChild != null && hasChild.Equals("Да", StringComparison.OrdinalIgnoreCase))//
Проверяет, равен ли ответ "Да"
{
    Console.WriteLine("Мальчик или девочка?"); // Запрашивает пол ребенка
    childGender = Console.ReadLine(); // Читает введённый пол ребенка
}

else if (hasChild != null && hasChild.Equals("Нет", StringComparison.OrdinalIgnoreCase))
// ну нет и нет))
{
    Console.WriteLine("Понятно, детей нет."); // Уведомляет, если детей нет
}

else
{
    Console.WriteLine("Некорректный ввод. Ответ будет считаться как 'Нет'."); // Сообщает
о некорректном вводе
    hasChild = "Нет"; // Устанавливает значение по умолчанию
}
```

```
if (age < 18) // Проверяем, меньше ли возраст 18
{
    Console.WriteLine("Вы в группе 1"); // Сообщает,
что пользователь в группе 1
}

else if (age >= 18 && age < 65) // Проверяем,
находится ли возраст от 18 до 64
{
    Console.WriteLine("Вы в группе 2"); // Сообщает,
что пользователь в группе 2
}

else // Если ни одно из предыдущих условий не
выполнено, то...
{
    Console.WriteLine("Вы в группе 3"); // Сообщает,
что пользователь в группе 3
}
```

If else

Конструкция if else в C# — это условная конструкция, позволяющая выполнять разные блоки кода в зависимости от заданного условия.




```
if (hasChild != null && hasChild.Equals("Да", StringComparison.OrdinalIgnoreCase))
```

hasChild.Equals — это вызов метода Equals для переменной hasChild. Словами это можно пояснить так:

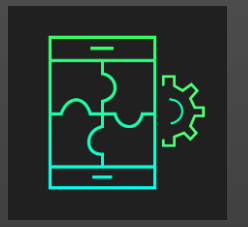
hasChild — это переменная, которая содержит значение.
. — оператор (точка) доступа к методу или свойству объекта.
Equals — это метод, который используется для сравнения значения переменной hasChild с другим значением (в данном случае со строкой "Да").

Получается так, hasChild.Equals("Да", StringComparison.OrdinalIgnoreCase) означает, что происходит сравнение значения переменной hasChild со строкой "Да", игнорируя регистр символов (ДА, Да, дА, да, любой вариант сработает)

StringComparison — это перечисление, которое содержит различные способы сравнения строк.
OrdinalIgnoreCase — это конкретный вариант сравнения, который означает следующее:

- Ordinal — строки сравниваются на основе их бинарного представления (то есть на основе ASCII/UTF-16 кодов символов).
- IgnoreCase — при сравнении регистр символов не учитывается (то есть "Да" и "да« и прочие варианты считаются равными).

try catch



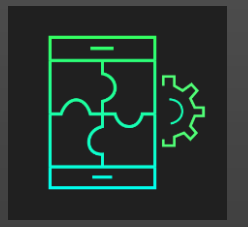
try catch — это конструкция в языке программирования C#, позволяющая обрабатывать исключения (ошибки) в коде.

try - В этом блоке находится код, который может вызвать исключение (ошибку). Если внутри блока try происходит ошибка, выполнение программы переходит в соответствующий блок catch.

catch - Этот блок следит за исключениями, которые могут произойти в блоке try. Если ошибка произошла, в catch можно обработать эту ошибку (например, вывести сообщение, записать в журнал и т.д.).

```
try
{
    age = Convert.ToInt32(ageInput); // Пробует преобразовать введённый возраст в целое число

    if (age < 0) // Проверка: если возраст отрицательный
    {
        Console.WriteLine("Некорректный ввод возраста. Установим возраст в 0."); // Уведомляет пользователя о некорректном вводе
        age = 0; // Устанавливает возраст в 0
    }
}

catch // Блок catch срабатывает, если было выброшено исключение
{
    Console.WriteLine("Некорректный ввод возраста. Установим возраст в 0."); // Уведомляет пользователя о некорректном вводе
}
```

Оператор Равенства (==)



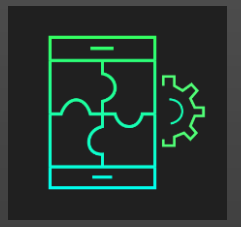
```
int a = 5;
int b = 5;
bool равно = (a == b);

Console.WriteLine(равно);  // Вывод: True
Console.ReadKey();
```

Оператор равенства (==) в C# — логический оператор, который проверяет равенство двух значений. Он используется для сравнения примитивных типов и объектов, что позволяет принимать решения в программировании.

Основное назначение оператора — проверка совпадения значений, что важно в условиях, циклах и обработке данных. Например, он может использоваться для проверки пользовательского ввода или сравнения состояний объектов в играх.

Однако есть риски, связанные с использованием оператора при сравнении объектов. Он может проверять **только ссылки, а не значения**, что может привести к неожиданным результатам, если программист не учитывает это.

Оператор Неравенства (!=)



```
int a = 5;
int b = 3;
bool неравны = (a != b);

Console.WriteLine(неравны);  // Вывод: True
```

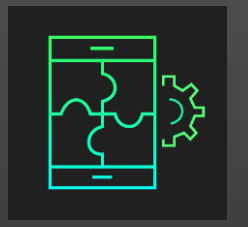
Оператор неравенства (!=) в C# — логический оператор, который проверяет, не равны ли два значения. Он важен для проведения сравнений между примитивными типами и объектами, что позволяет принимать решения в программировании.

Основное назначение оператора — выявление различий, что полезно в условиях и циклах. Например, его используют для проверки пользовательского ввода или анализа данных в алгоритмах.

Сценарии использования разнообразны: от валидации ввода до логики игр (Unity) для определения изменений состояния объектов и просто сопоставления значений.

Риски использования связаны с тем, что оператор может проверять только ссылки при сравнении объектов, а не их значения, что может привести к неожиданным результатам.

Оператор Меньше (<)



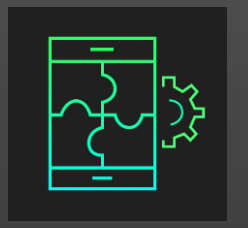
```
int a = 5;
int b = 10;
bool меньше = (a < b);

Console.WriteLine(меньше);  // Вывод: True
```

Оператор "меньше" (<) в C# — сравнительный оператор, определяющий, является ли одно значение меньшим другого. Он возвращает **true**, если первое значение меньше, и **false** в противном случае.

Оператор используется в условиях и циклах, позволяя контролировать поток выполнения и валидацию данных. Например, его можно применять для проверки возраста пользователя или ограничения движения объектов в играх. Можно применять в сценариях от математических вычислений до анализа данных и сортировки. В графических приложениях оператор помогает управлять поведением персонажей.

Однако существуют риски, связанные с использованием оператора. Например, неверное сравнение разных типов данных, таких как целые числа с числами с плавающей запятой, может привести к ошибкам. Также важно учитывать возможность неопределенных или нулевых значений, что может вызвать ошибки выполнения.

Оператор Больше (>)



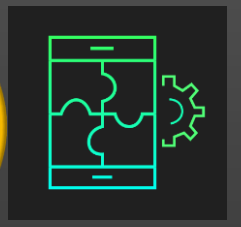
```
int a = 10;
int b = 5;
bool больше = (a > b);

Console.WriteLine(больше);  // Вывод: True
```

Оператор "больше" (>) в C# определяет, является ли первое значение больше второго, возвращая **true** или **false**. Он используется в условиях и циклах для управления логикой, например, проверяя превышение порога.

Сценарии применения оператора включают математические вычисления, анализ данных и поведение персонажей в играх. Он позволяет сортировать и фильтровать данные.

Однако, риски связаны с неверным сравнением типов данных, что может вызвать неожиданные результаты. Работы с нулевыми значениями также могут привести к ошибкам.

Оператор Меньше или Равно (<=)



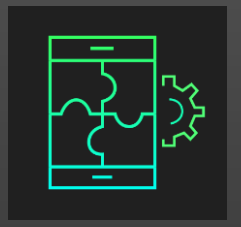
```
int a = 5;
int b = 5;
bool меньшеИлиРавно = (a <= b);

Console.WriteLine(меньшеИлиРавно);  // Вывод: True
```

Оператор "меньше или равно" (<=) в C# относится к сравнительным операторам и определяет, меньше или равно первое значение второму. Он возвращает **true** или **false**, что позволяет использовать его в условиях и циклах.

Применения оператора включают валидацию данных, проверку возрастных ограничений и управление логикой в играх (снова любимый Unity).

Риски использования связаны с неправильным сравнением типов данных и работой с нулевыми значениями, что может вызвать ошибки. Важно соблюдать осторожность с типами данных и условиями при использовании оператора.

Оператор Больше или Равно (>=)



```
int a = 10;
int b = 10;
bool большеИлиРавно = (a >= b);

Console.WriteLine(большеИлиРавно);  // Вывод: True
```

Оператор "больше или равно" (>=) в C# относится к сравнительным операторам и определяет, превышает ли первое значение второе или равно ему. Он возвращает **true** или **false**, что позволяет его использовать в условиях и циклах.

Применения оператора включают валидацию данных, управление играми и фильтрацию записей в базах данных.

Риски использования связаны с неправильным сравнением типов данных и работой с нулевыми значениями, что может вызвать ошибки. Важно быть внимательным к типам данных и условиям при его использовании.

Оператор Логического И (&&)

```
bool условие1 = true;
bool условие2 = false;
bool результат = условие1 && условие2;

Console.WriteLine(результат); // Вывод: False
```

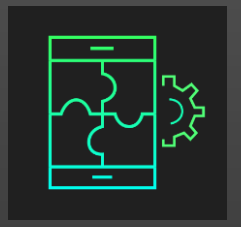
Оператор логического И (&&) в C# относится к логическим операторам и объединяет два или более логических выражений, возвращая true, только если все выражения истинны.

Он широко используется в условиях и циклах для проверки нескольких условий одновременно, например, авторизации пользователя.

Сценарии применения включают валидацию данных и управление доступом в приложениях.

Риски использования могут привести к игнорированию условий с ложными выражениями, что вызывает непредсказуемость кода. Важно следить за читаемостью и структурой логических выражений для предотвращения ошибок.

Оператор Присваивания (=)



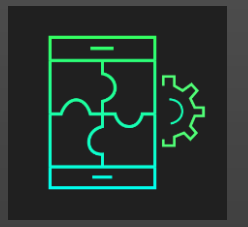
```
            int a = 5;  // Присваиваем переменной a значение 5
            int b = 10; // Присваиваем переменной b значение 10
            int сумма = a + b; // Присваиваем переменной сумма
результат сложения a и b

            Console.WriteLine(сумма); // Вывод: 15
```

Оператор присваивания (=) в C# относится к операторам присваивания и устанавливает значение переменной **слева** от оператора.

Он используется для инициализации переменных, обновления их значений и передачи данных. Применяется в арифметических расчетах и управлении состоянием объектов.

Риски включают ошибки при использовании присваивания вместо сравнения (==), что может вызвать логические ошибки. Важно быть внимательным, чтобы избежать таких проблем.

Оператор Добавление с Присваиванием (+=)



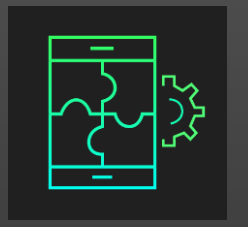
```
int число = 10;  // Инициализируем переменную
число += 5;      // Увеличиваем число на 5

Console.WriteLine(число);  // Вывод: 15
```

Оператор добавления с присваиванием (+=) в C# относится к комбинированным операторам присваивания и позволяет добавлять значение к переменной, одновременно присваивая результат обратно.

Он используется для увеличения чисел, добавления элементов в коллекции и конкатенации строк, что делает код более читабельным и лаконичным. Сценарии включают счётчики, накопление сумм и формирование строк.

Риски включают исключения при неинициализированных переменных и возможные ошибки из-за несовместимости типов. Оператор требует внимательности для предотвращения подобных проблем.

Оператор Вычитание с Присваиванием (-=)



```
int число = 20;  // Инициализируем переменную
число -= 5;      // Уменьшаем число на 5

Console.WriteLine(число); // Вывод: 15
```

Оператор вычитания с присваиванием (-=) в C# относится к комбинированным операторам присваивания и позволяет вычитать значение из переменной, присваивая результат обратно.

Он также используется для уменьшения чисел и вычитания элементов в коллекциях, упрощая код за счет сокращения повторений. Сценарии включают счетчики, накопление отрицательных значений и вычисления.

Риски включают исключения при неинициализированных переменных и возможные ошибки из-за несовместимости типов. Оператор требует внимательности, чтобы избежать таких проблем.