

Программирование ветвлений на Паскале

Ветвления позволяют программе выбирать действия по условию. В Pascal это if, then, else и case.





Где в реальной программе встречаются ветвления

Ветвления живут в играх, тестах, калькуляторах, меню и справочниках: программа сравнивает значения, проверяет условие и выбирает один из вариантов. Для 9 класса Pascal особенно удобен: синтаксис строгий, а условия читаются наглядно.

Базовая конструкция if

Основная запись условия в Pascal выглядит так: `if условие then действие`. Если нужно, чтобы результат был разным, добавляется ветка `else`, и тогда выполняется подходящая команда в зависимости от проверки.

Условие собирают из операторов сравнения: `=`, `<`, `>`, `<=`, `>=`, `<>`. Например, можно проверять, число положительное или нет, и печатать соответствующий текст в результате выполнения программы.



Операторы сравнения в Паскале

Таблица показывает, как записываются операторы сравнения и какие значения они возвращают в условиях: True или False.

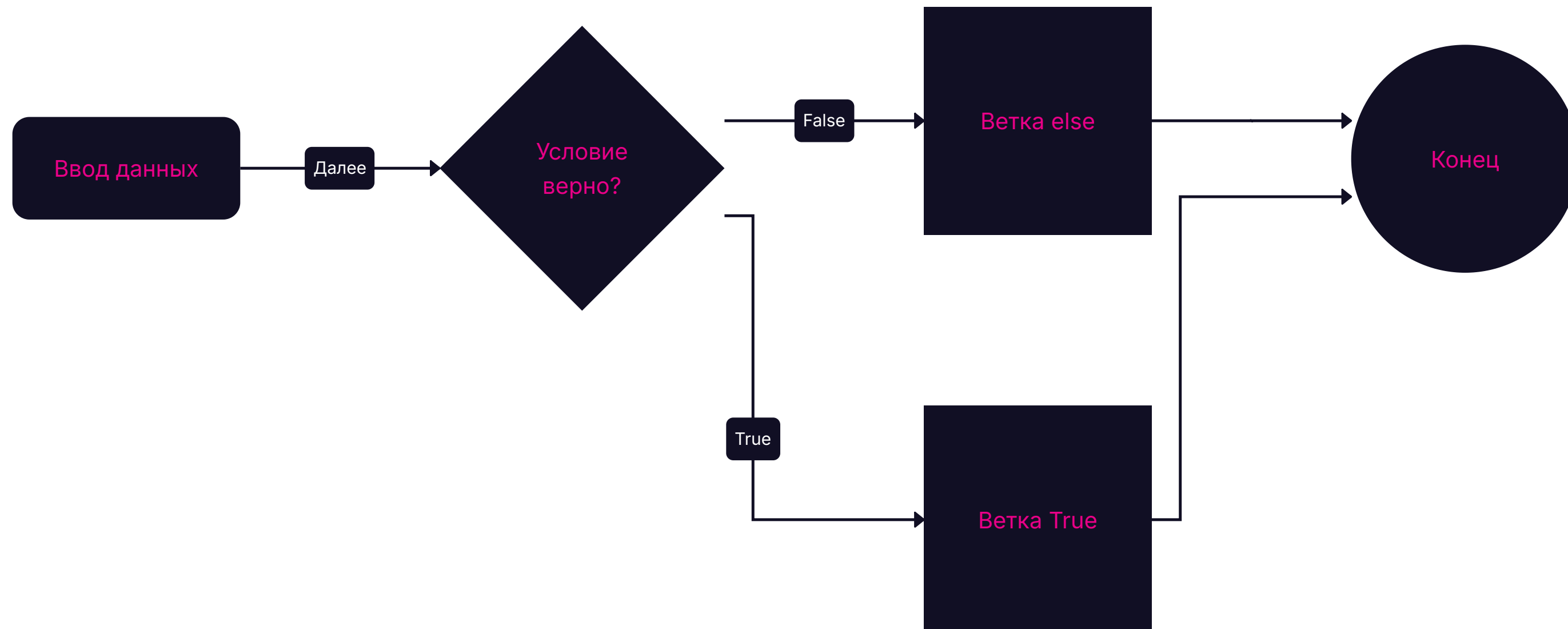
Любое сравнение в условии даёт два возможных исхода: True (условие верно) или False (условие неверно).

Правила записи условий в языке Pascal.

Оператор	Смысл	Пример
=	равно	a = 5
<>	не равно	a <> 0
<	меньше	x < 10
>	больше	x > 3
<=	меньше или равно	x <= 7
>=	больше или равно	x >= 2

Как работает ветвление if-then-else

Поток выполнения: ввод → проверка → выбор ветки True/False → выполнение команды → завершение.



Составные операторы в ветвлениях

01

Когда в одной ветке нужно выполнить больше одной команды, Pascal позволяет объединить их в блок `begin...end`. Тогда программа не «перескакивает» между командами по одному действию, а выполняет всю группу.

02

Составной оператор удобен, если после проверки условия требуется вывести сообщение, изменить несколько переменных или сделать несколько вычислений подряд. Весь этот набор становится одним логическим шагом.

03

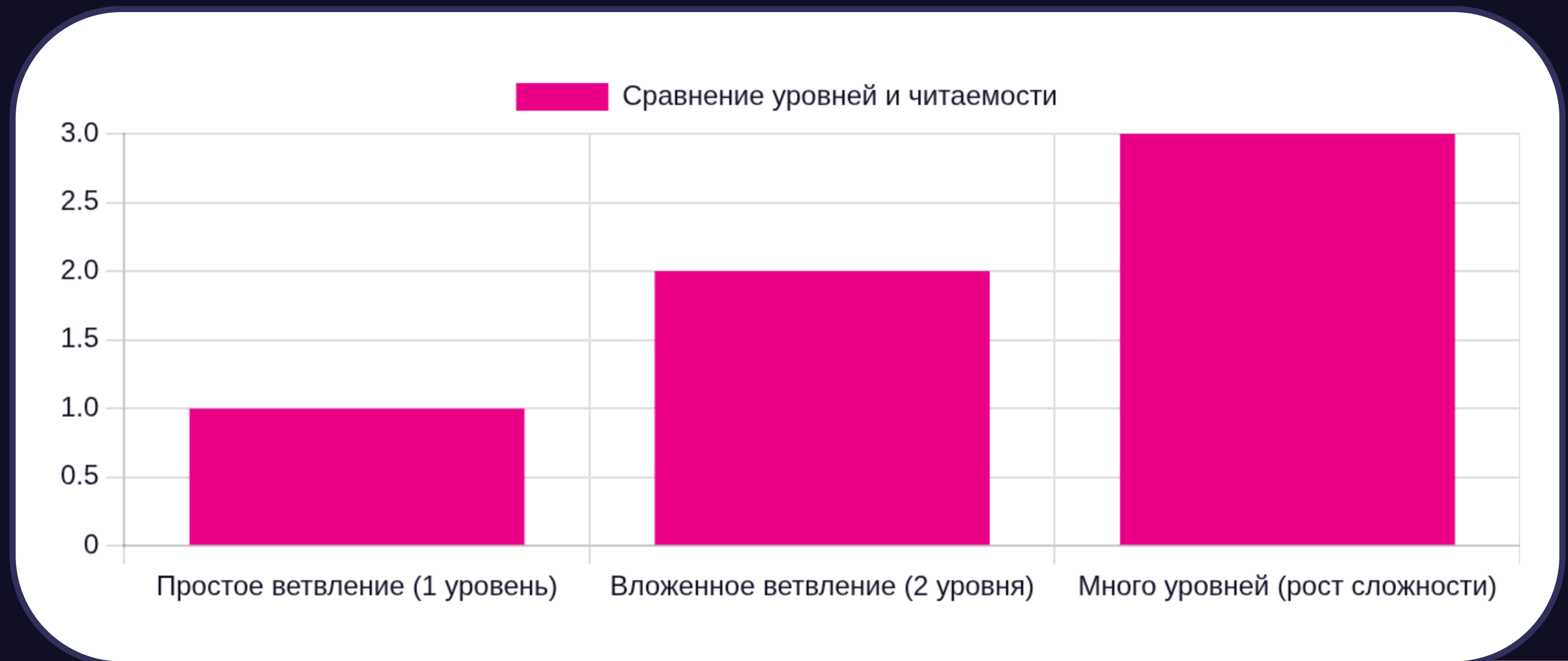
Блок `begin...end` особенно полезен, когда ветка длинная: так проще проверять, где начинается и заканчивается выполнение, и меньше вероятность ошибиться с «лишними» командами внутри `if`.

Простое и вложенное ветвление

Простое ветвление обычно короткое и легко читается. Вложенное добавляет новые проверки внутри условий, поэтому требуется больше внимания к структуре и отступам.

Чем глубже вложенность, тем сложнее сопровождать программу. Для обучения проще начинать с одного `if`, а затем постепенно переходить к нескольким уровням.

Обобщённое сравнение по структуре программ на Pascal (логическая оценка).



Вложенные ветвления

01

Вложенное ветвление — это `if`, который находится внутри другой проверки. Сначала программа решает первую задачу (например, верхний диапазон), а затем уточняет решение второй проверкой.

02

Такой подход подходит, когда условия идут последовательно: сначала отделяем одну группу значений, а затем внутри неё выбираем более точный ответ. Например, это удобно для шкалы оценок по диапазонам баллов.

03

В Pascal важно аккуратно оформлять вложенные ветки: используйте отступы и при длинных ветках добавляйте `begin...end`. Тогда видно, какие команды относятся к какому именно `if`.

04

Если пропустить `begin...end` или «съедут» границы блока, программа может выполнить не те команды. Поэтому при вложениях особенно полезна тщательная проверка кода перед запуском.

Логические операторы в условиях

Оператор	Что означает	Пример
and	оба условия истинны	<code>a > 0 and a < 10</code>
or	достаточно одного истинного	<code>x = 1 or x = 2</code>
not	делает противоположное	<code>not (b = 0)</code>

Таблица показывает три логических оператора, с помощью которых собирают сложные условия в Pascal.

Логические операторы позволяют объединять проверки, чтобы получить одно итоговое значение условия: True или False.

Основы логики и условия в языке Pascal.

Проверка чётности числа

Чётное или нечётное: быстрый ответ

Представь число как игрока на входе в «клуб». Мы делим его на 2 и смотрим остаток: если остаток 0, то число проходит как чётное. В противном случае программа объявляет, что число нечётное.



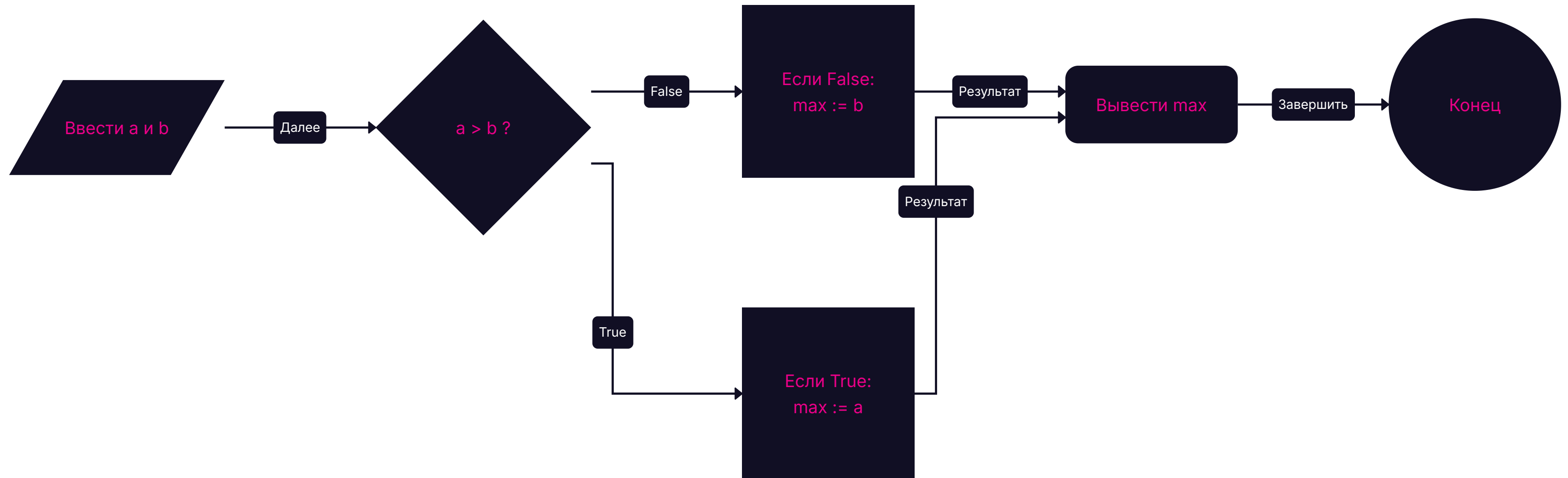
Остаток от деления как подсказка

Иногда достаточно одного остатка, чтобы понять результат. Оператор `mod` вычисляет остаток при делении. Поэтому запись вида `if n mod 2 = 0 then ... else ...` превращает математику в понятный алгоритм с выводом сообщения для любого `n`.



Алгоритм поиска наибольшего из двух чисел

Ввод a и b → сравнение $a > b$ → выбор результата (ветвление) → завершение.



Оператор case: когда вариантов много



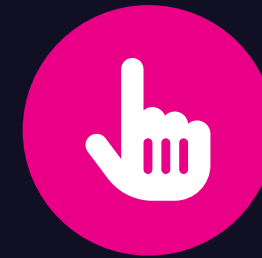
Выбор по значению

Оператор case подходит, когда одну переменную сравнивают сразу с несколькими точными вариантами. Например, при значениях `n` выводят разный пункт меню. Код становится короче и читается легче.



Дни недели и справочники

case удобно использовать для выбора дня недели, месяца или номера правила. Вместо длинной цепочки `if` программа делает один явный выбор: при совпадении значения выполняется соответствующая ветка.



Меню и номера ответов

Если пользователь вводит номер варианта, case позволяет сразу перейти к нужному действию. Для предусмотренных значений пишут отдельные ветки, а для остальных — блок `else`, чтобы обработать ошибку.

Когда использовать if, а когда case

Выберите конструкцию по характеру условия: диапазоны и сложная логика — для if, фиксированные точные варианты — для case.

if гибче: подходит для диапазонов и сложных условий. case компактнее и нагляднее, когда выбирается один вариант из нескольких точно заданных значений одной переменной.

Правила построения ветвлений в Pascal.

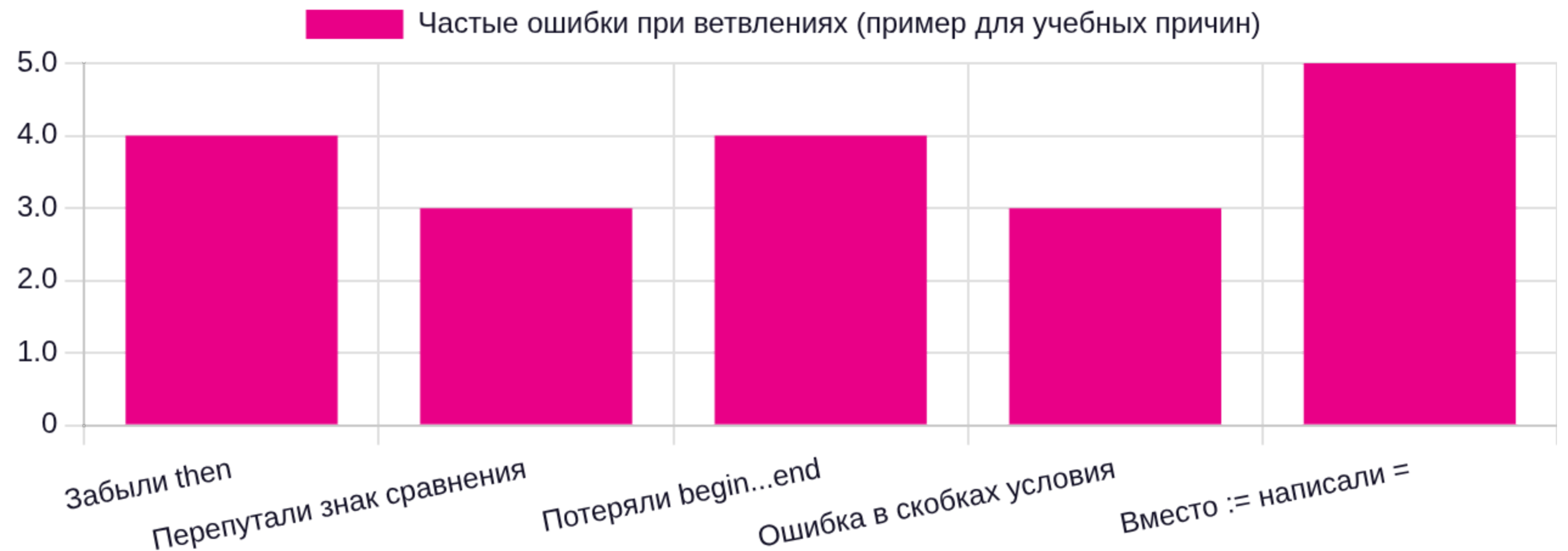
Признак задачи	Подходящая конструкция	Итог
Диапазоны, неравенства	if	Нужны условия вида <, >, <=, >=
Неравенства и сложные связи	if	Подходят and/or для нескольких проверок
Много равных вариантов	case	Наглядно выбирается нужная ветка по значению

Типичные ошибки в ветвлениях

Часть ошибок видна только при тестировании: ветка выполняется «не там», потому что условие составлено неверно или команда не вошла в блок.

Чтобы снизить число сбоев, внимательно проверяйте скобки, парность операторов и состав веток. Отдельно контролируйте различие «сравнить» и «присвоить».

Учебная практика по синтаксису и логике ветвлений в Pascal; примерная шкала для демонстрации распространённости ошибок.



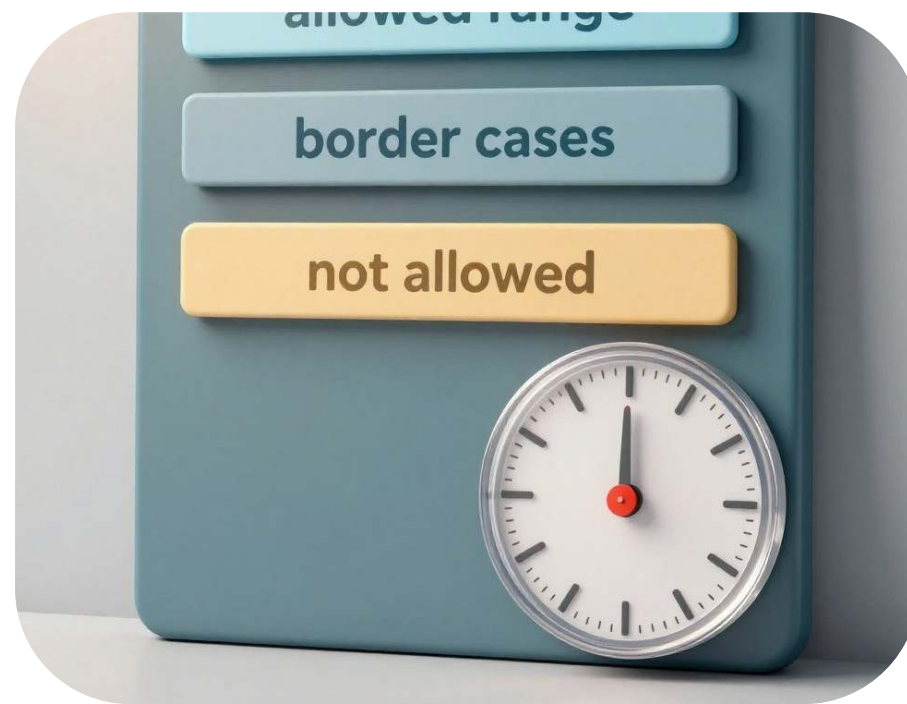
2

Важно не перепутать два разных оператора в Pascal, иначе условие перестанет работать.

Учебные материалы по языку Pascal: синтаксис операторов = и :=.

Ветвления ломаются, когда вместо присваивания делают сравнение или наоборот. Это особенно опасно в счётчиках, флагах и промежуточных результатах.

Паскаль в школьных задачах



Возраст и школьные правила

Представьте задачу: пользователь вводит возраст, а программа отвечает, можно ли идти дальше по условию. Обычно требуется 2–3 ветки, где сравнение «попадает в диапазон» и выводится нужное сообщение.



Тест с вариантами ответов

В тестах часто нужно проверить, какой вариант ввели, и вывести результат. Здесь удобно использовать `if` для логики проверки или `case`, когда вариантов немного и они равны по смыслу. Код получается коротким и понятным.

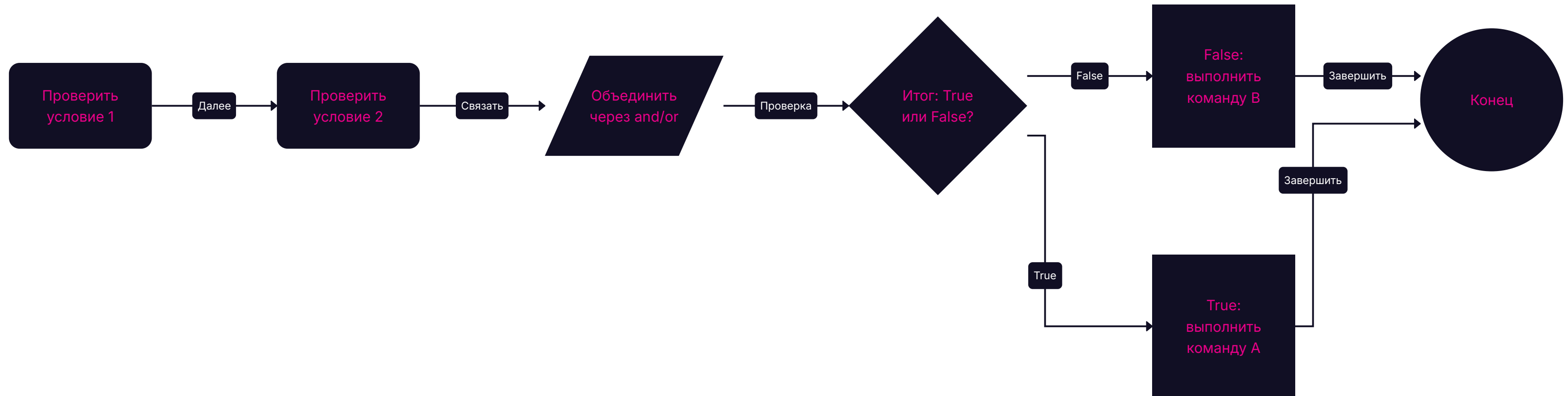


Баллы: «оценка по диапазону»

Самый частый школьный сюжет — перевести баллы в отметку. Сначала проверяют верхние границы, затем уточняют нижние диапазоны. Вложенные `if` помогают последовательно учесть все интервалы и не перепутать границы.

Как читать сложное условие

Проверки по шагам → объединение and/or → получение True/False → выбор ветки.



Краткая памятка по синтаксису ветвлений

Форма	Когда применяется	Смысл
if условие then действие	Проверка одного условия	Если условие верно — выполняется действие
if...then...else...	Два варианта	else выполняется, когда условие ложно
begin...end	Несколько команд	Собирает команды в одну ветку
case переменная of	Много равных вариантов	Выбирает ветку по значению

Записывайте конструкцию точно: пропущенный ключевой элемент или неверная форма записи приведут к ошибке компиляции.

Каждая часть конструкции должна стоять в нужном порядке. Если условие или блок записаны неточно, компилятор укажет ошибку, а логика может нарушиться.

Синтаксис ветвлений в языке Pascal.

Пошаговая проверка программы с ветвлением

1. Введите тестовые значения

Подготовьте набор: ноль, отрицательное число, маленькое положительное и большое число. Это помогает проверить, что условие срабатывает в разных ситуациях.

2. Проверьте ключевое условие

Для каждого значения подумайте, какой веткой должна управлять проверка. Сравните ожидаемый True/False с тем, что делает программа при вводе.

3. Убедитесь, что выполнен нужный блок

Проверьте, что команды попадают именно в ту ветку. Особое внимание к `begin...end`: без него часть действий может оказаться «снаружи».

4. Сверьте вывод с логикой задачи

Сравните полученный результат с примером из условия. Если хотя бы один вариант не совпадает, значит, условие или ветвление записаны неверно.

Ветвления делают программу «умной»

Ветвления делают программу гибкой: она выбирает действия по условию. Основа — if, else, case, а также and, or, not и блоки begin...end. Проверяйте знаки = и := и тестируйте решения разными значениями.