PROGRAM LexError1;

VAR

a : INTEGER;

b# : REAL; { Недопустимый символ '#' в идентификаторе }

BEGIN

a := 5;

WRITE(a);

END.

PROGRAM LexError2;

VAR

x : INTEGER; { Это комментарий, который не закрыт...

BEGIN

x := 10;

WRITE(x);

END.

PROGRAM LexError3;

BEGIN

WRITE('Hello, world! ; { Строка не закрыта кавычкой }

END.

PROGRAM LexError4;

VAR

r : REAL;

BEGIN

r := 123.; { Точка без цифр после нее }

WRITE(r);

END.

PROGRAM SyntaxError1;

VAR

a : INTEGER

b : INTEGER; { Отсутствует ; после объявления 'a' }

BEGIN

a := 1;

b := 2

WRITE(a + b); { Отсутствует ; после присваивания 'b' }

END.

PROGRAM SyntaxError2;

VAR i : INTEGER;

BEGIN

i = 5; { Использован '=', а не ':=' }

WRITE(i);

END.

PROGRAM SyntaxError3;

BEGIN { Отсутствует соответствующий END. }

WRITE('Hello');

PROGRAM SyntaxError4;

VAR flag : INTEGER; { Используем INTEGER как BOOLEAN для простоты }

BEGIN

flag := 1;

IF flag = 1

WRITE('True'); { Отсутствует THEN }

END.

PROGRAM SyntaxError5;

VAR i : INTEGER;

BEGIN

i := 0;

WHILE i < 5

BEGIN { Отсутствует DO }

i := i + 1;

END;

WRITE(i);

END.

PROGRAM SyntaxError6;

VAR

a : ; { Отсутствует тип }

CONST

Pi = ; { Отсутствует значение }

BEGIN

WRITE('Error');

END.

PROGRAM SyntaxError7;

VAR x, y : INTEGER;

BEGIN

x := (5 + 3 \* (2 + 1 ); { Лишняя/непарная скобка }

y := 5 + (3;

WRITE(x);

END.

PROGRAM SyntaxError8;

VAR a : INTEGER;

BEGIN

a := 10;

IF a > 5 THEN

WRITE('>');

ELSE { Ошибка: перед ELSE не должно быть ';' если THEN - один оператор }

WRITE('<=');

END.

PROGRAM SemanticError1;

VAR a : INTEGER;

BEGIN

a := 5;

b := a + 1; { Переменная 'b' не объявлена }

WRITE(b);

END.

PROGRAM SemanticError2;

VAR

i : INTEGER;

r : REAL;

BEGIN

r := 3.14;

i := r; { Нельзя присвоить REAL переменной INTEGER без явного преобразования }

WRITE(i);

END.

PROGRAM SemanticError3;

VAR

i : INTEGER;

s : INTEGER; { Представим, что строки поддерживаются типом STRING }

BEGIN

i := 5;

s := 'hello'; { Парсер может пропустить, если STRING есть в грамматике }

i := i + s; { Сложение числа и "строки" }

WRITE(i);

END.

PROGRAM SemanticError3b;

VAR i : INTEGER;

BEGIN

i := 5;

i := i + 'hello'; { Сложение числа и строкового литерала }

WRITE(i);

END.

PROGRAM SemanticError4;

PROCEDURE PrintTwo(a : INTEGER; b : INTEGER);

BEGIN

WRITE(a, b);

END;

BEGIN { Main }

PrintTwo(10); { Вызвано с одним аргументом вместо двух }

END.

PROGRAM SemanticError5;

PROCEDURE PrintInt(val : INTEGER);

BEGIN

WRITE(val);

END;

BEGIN { Main }

PrintInt(3.14); { Вызвано с REAL вместо INTEGER }

END.

PROGRAM SemanticError6;

VAR

a : INTEGER;

a : REAL; { Повторное объявление 'a' в том же блоке }

BEGIN

a := 1;

WRITE(a);

END.

PROGRAM IRGenError1;

{ ... объявления ... }

BEGIN

{ ... какой-то код ... }

CASE some\_var OF { Предполагаем, что парсер это построил }

1: WRITE('One');

2: WRITE('Two');

ELSE

WRITE('Other');

END; { CASE }

END.

PROGRAM RuntimeError2;

VAR

x : INTEGER;

y : INTEGER;

BEGIN

y := x + 5; { 'x' объявлена, но ей не присвоено значение }

WRITE(y);

END.

PROGRAM RuntimeError3;

VAR i : INTEGER;

BEGIN

i := 1;

{ Предположим, что IR-генератор позволил сложить число и строку }

i := i + 'oops';

WRITE(i);

END.

PROGRAM RuntimeError4;

VAR age : INTEGER;

BEGIN

WRITE('Enter your age: ');

READ(age); { Пользователь вводит "abc" вместо числа }

WRITE('Age is: ', age);

END.

PROGRAM FeatureTestNoLessGreater; // Изменил имя программы

CONST

Limit = 10;

Epsilon = 0.001;

WelcomeMsg = '--- Pascal-Like Compiler Test (No < or >) ---'; // Изменил сообщение

VAR

counter, userValue, tempInt : INTEGER;

ratio, calculatedValue, userReal : REAL;

isDone : INTEGER; // 0 for false, 1 for true

// Процедура для демонстрации арифметики и параметров

PROCEDURE CalculateValues(inputInt : INTEGER; inputReal : REAL);

VAR

localResultInt : INTEGER;

localResultReal : REAL;

BEGIN

WRITE('\n--- Inside CalculateValues ---', '\n');

WRITE('Received: inputInt = ', inputInt, ', inputReal = ', inputReal, '\n');

localResultInt := inputInt \* Limit + 5;

WRITE('inputInt \* Limit + 5 = ', localResultInt, '\n');

localResultReal := inputReal / 2.0 - Epsilon;

WRITE('inputReal / 2.0 - Epsilon = ', localResultReal, '\n');

// Целочисленное деление

tempInt := inputInt + 7;

localResultInt := tempInt DIV 3;

WRITE('(inputInt + 7) DIV 3 = ', localResultInt, '\n');

// Унарный минус

localResultInt := -inputInt;

WRITE('-inputInt = ', localResultInt, '\n');

calculatedValue := localResultReal;

WRITE('Stored ', calculatedValue, ' into global calculatedValue', '\n');

WRITE('--- Leaving CalculateValues ---', '\n');

END; // Конец CalculateValues

// Рекурсивная процедура для демонстрации рекурсии

PROCEDURE RecursiveCounter(currentVal : INTEGER);

BEGIN

WRITE('Recursive step: ', currentVal, '\n');

// Замена: IF currentVal > 0 THEN на IF NOT (currentVal <= 0) THEN

IF NOT (currentVal <= 0) THEN

BEGIN

RecursiveCounter(currentVal - 1);

END

ELSE

BEGIN

WRITE('Recursion base case reached.', '\n');

END;

END; // Конец RecursiveCounter

// Основной блок программы

BEGIN

WRITE(WelcomeMsg, '\n\n');

// Ввод данных

WRITE('Enter an integer value: ');

READ(userValue);

WRITE('Enter a real value: ');

READ(userReal);

// Вызов процедуры

CalculateValues(userValue, userReal);

// Использование IF/THEN/ELSE и логических операций

WRITE('\n--- Testing Conditions ---', '\n');

counter := userValue;

ratio := userReal;

isDone := 0; // false

// Замена: IF (counter > Limit) AND ... на IF NOT (counter <= Limit) AND ...

IF NOT (counter <= Limit) AND (ratio <> 0.0) THEN

BEGIN

WRITE('Condition NOT (counter <= Limit) AND (ratio <> 0.0) is TRUE.', '\n'); // Сообщение изменено

calculatedValue := ratio / Limit;

isDone := 1;

END

// Замена: ... OR (ratio < Epsilon) на ... OR NOT (ratio >= Epsilon)

ELSE IF (counter = Limit) OR NOT (ratio >= Epsilon) THEN

BEGIN

WRITE('Condition (counter = Limit) OR NOT (ratio >= Epsilon) is TRUE.', '\n'); // Сообщение изменено

calculatedValue := counter + ratio;

isDone := 1;

END

ELSE

BEGIN

WRITE('Neither complex condition was met.', '\n');

IF NOT (counter = 0) THEN

BEGIN

WRITE('However, counter is NOT zero.', '\n');

calculatedValue := -1.0;

END;

END;

WRITE('Value of calculatedValue after IFs: ', calculatedValue, '\n');

WRITE('Value of isDone: ', isDone, '\n');

// Использование WHILE

WRITE('\n--- Testing WHILE loop ---', '\n');

counter := 0;

// Замена: WHILE counter < (userValue DIV 2) DO на WHILE NOT (counter >= (userValue DIV 2)) DO

WHILE NOT (counter >= (userValue DIV 2)) DO

BEGIN

WRITE(' Loop iteration, counter = ', counter, '\n');

counter := counter + 1;

IF counter = 2 THEN

BEGIN

WRITE(' Counter just reached 2!', '\n');

END;

END;

WRITE('WHILE loop finished. Final counter = ', counter, '\n');

// Вызов рекурсивной процедуры

WRITE('\n--- Testing Recursion ---', '\n');

WRITE('Starting recursive countdown from 3:', '\n');

RecursiveCounter(3); // Рекурсивная процедура теперь тоже использует NOT (...)

WRITE('\n--- End of Test Program ---', '\n');

END. // Конец основной программы

I. Базовый синтаксис и объявления:

Минимальная программа:

PROGRAM Minimal;

BEGIN

END.

Объявление переменных разных типов:

PROGRAM VarTest;

VAR

i, j, k : INTEGER;

x, y, z : REAL;

count : INTEGER;

value : REAL;

BEGIN

i := 1;

x := 3.14;

WRITE('Done.');

END.

Объявление констант:

PROGRAM ConstTest;

CONST

MaxVal = 100;

Pi = 3.14159;

Greeting = 'Hello';

VAR

num : INTEGER;

BEGIN

num := MaxVal - 10;

WRITE(Greeting, ', Pi is approx ', Pi, ', num is ', num);

END.

Пустые блоки объявлений и операторов:

PROGRAM EmptyBlocks;

CONST

VAR

BEGIN

END.

Комментарии:

PROGRAM CommentsTest;

{ Это многострочный

комментарий }

VAR // Однострочный комментарий

a : INTEGER; // Еще один

BEGIN

a := 10; { Присваивание }

// WRITE(a); // Закомментированный вывод

WRITE('Test');

END. // Конец программы

II. Арифметические операции:

Все арифметические операции с целыми:

PROGRAM IntArithmetic;

VAR

a, b, sum, diff, prod, quot, rem : INTEGER;

BEGIN

a := 17;

b := 5;

sum := a + b;

diff := a - b;

prod := a \* b;

quot := a DIV b; // Целочисленное деление

rem := 0; // В Паскале есть MOD, у вас его нет, DIV достаточно

WRITE('a=',a,', b=',b,'\n');

WRITE('Sum: ', sum, '\n'); // 22

WRITE('Diff: ', diff, '\n'); // 12

WRITE('Prod: ', prod, '\n'); // 85

WRITE('Quot (DIV): ', quot, '\n'); // 3

END.

Все арифметические операции с вещественными:

PROGRAM RealArithmetic;

VAR

x, y, rSum, rDiff, rProd, rQuot : REAL;

BEGIN

x := 17.0;

y := 5.0;

rSum := x + y;

rDiff := x - y;

rProd := x \* y;

rQuot := x / y; // Вещественное деление

WRITE('x=',x,', y=',y,'\n');

WRITE('rSum: ', rSum, '\n'); // 22.0

WRITE('rDiff: ', rDiff, '\n'); // 12.0

WRITE('rProd: ', rProd, '\n'); // 85.0

WRITE('rQuot: ', rQuot, '\n'); // 3.4

END.

Смешанная арифметика (если поддерживается или как должны обрабатываться типы):

PROGRAM MixedArithmetic;

VAR

i : INTEGER;

r : REAL;

resultReal : REAL;

resultInt : INTEGER;

BEGIN

i := 10;

r := 2.5;

resultReal := i + r; // Должно быть 12.5

WRITE('i + r = ', resultReal, '\n');

resultReal := i \* r; // Должно быть 25.0

WRITE('i \* r = ', resultReal, '\n');

resultReal := i / r; // Должно быть 4.0

WRITE('i / r = ', resultReal, '\n');

// resultInt := i + r; // Это должно вызывать ошибку типа или неявное преобразование

// WRITE(resultInt);

END.

Приоритет операций и скобки:

PROGRAM Precedence;

VAR

res : REAL;

BEGIN

res := 2 + 3 \* 4; // 2 + 12 = 14

WRITE('2 + 3 \* 4 = ', res, '\n');

res := (2 + 3) \* 4; // 5 \* 4 = 20

WRITE('(2 + 3) \* 4 = ', res, '\n');

res := 10 / 2 \* 5; // (10 / 2) \* 5 = 5 \* 5 = 25

WRITE('10 / 2 \* 5 = ', res, '\n');

END.

Унарный минус и плюс:

PROGRAM UnaryOps;

VAR

a, b : INTEGER;

x, y : REAL;

BEGIN

a := 5;

b := -a; // b = -5

WRITE('b = -a: ', b, '\n');

x := -3.14;

y := +x; // y = -3.14 (унарный плюс обычно ничего не меняет)

WRITE('y = +x: ', y, '\n');

b := -(a + 2); // b = -(7) = -7

WRITE('b = -(a+2): ', b, '\n');

END.

III. Ввод-вывод:

Простой ввод и вывод:

PROGRAM InputOutput;

VAR

num : INTEGER;

val : REAL;

BEGIN

WRITE('Enter an integer: ');

READ(num);

WRITE('Enter a real number: ');

READ(val);

WRITE('You entered integer: ', num, ' and real: ', val, '\n');

END.

Вывод нескольких значений, включая строки и константы:

PROGRAM MultiWrite;

CONST Five = 5;

VAR i : INTEGER;

BEGIN

i := 10;

WRITE('Constants: ', Five, ', Variables: ', i, ', Literal string.', '\n');

END.

Чтение нескольких переменных:

PROGRAM MultiRead;

VAR a, b : INTEGER;

BEGIN

WRITE('Enter two integers separated by space/enter: ');

READ(a, b);

WRITE('A = ', a, ', B = ', b, '\n');

END.

IV. Управляющие конструкции:

Простой IF-THEN:

PROGRAM IfThen;

VAR a : INTEGER;

BEGIN

a := 10;

IF a = 10 THEN

WRITE('a is 10', '\n');

IF NOT (a = 5) THEN

WRITE('a is not 5', '\n');

END.

IF-THEN-ELSE:

PROGRAM IfThenElse;

VAR a : INTEGER;

BEGIN

WRITE('Enter a number: ');

READ(a);

IF a = 0 THEN

WRITE('a is zero', '\n')

ELSE

WRITE('a is not zero', '\n');

END.

Вложенные IF:

PROGRAM NestedIf;

VAR a, b : INTEGER;

BEGIN

a := 5;

b := 10;

IF a = 5 THEN

BEGIN

WRITE('a is 5. ', '\n');

IF b = 10 THEN

WRITE('And b is 10.', '\n')

ELSE

WRITE('But b is not 10.', '\n');

END

ELSE

WRITE('a is not 5.', '\n');

END.

Простой WHILE:

PROGRAM WhileLoop;

VAR i : INTEGER;

BEGIN

i := 0;

WHILE i < 5 DO // Замените < на NOT (i >= 5) если < нет

BEGIN

WRITE('i = ', i, '\n');

i := i + 1;

END;

WRITE('Loop finished.', '\n');

END.

WHILE с условием, которое изначально ложно:

PROGRAM WhileFalse;

VAR i : INTEGER;

BEGIN

i := 10;

// Условие i < 5 (или NOT (i >= 5)) ложно

WHILE NOT (i >= 5) DO

BEGIN

WRITE('This should not print.', '\n');

i := i + 1;

END;

WRITE('Loop correctly skipped.', '\n');

END.

BEGIN-END внутри управляющих конструкций:

PROGRAM BeginEndScope;

VAR x : INTEGER;

BEGIN

x := 1;

IF x = 1 THEN

BEGIN // Блок для THEN

WRITE('x is one.', '\n');

x := x + 1;

WRITE('Now x is ', x, '\n');

END;

END.

V. Логические операции:

AND, OR, NOT:

PROGRAM LogicalOps;

VAR

a, b, c : INTEGER; // Используем INTEGER как bool (0=false, <>0=true)

res : INTEGER;

BEGIN

a := 1; { True }

b := 0; { False }

WRITE('--- AND ---', '\n');

res := a AND a; WRITE('T AND T = ', res, '\n'); // True (1)

res := a AND b; WRITE('T AND F = ', res, '\n'); // False (0)

res := b AND a; WRITE('F AND T = ', res, '\n'); // False (0)

res := b AND b; WRITE('F AND F = ', res, '\n'); // False (0)

WRITE('--- OR ---', '\n');

res := a OR a; WRITE('T OR T = ', res, '\n'); // True (1)

res := a OR b; WRITE('T OR F = ', res, '\n'); // True (1)

res := b OR a; WRITE('F OR T = ', res, '\n'); // True (1)

res := b OR b; WRITE('F OR F = ', res, '\n'); // False (0)

WRITE('--- NOT ---', '\n');

res := NOT a; WRITE('NOT T = ', res, '\n'); // False (0)

res := NOT b; WRITE('NOT F = ', res, '\n'); // True (1)

c := 1;

IF (a AND (NOT b)) OR (b AND c) THEN // (T AND T) OR (F AND T) -> T OR F -> T

WRITE('Complex logic is TRUE', '\n')

ELSE

WRITE('Complex logic is FALSE', '\n');

END.

Примечание: Как ваш интерпретатор обрабатывает AND/OR с числами? Python and/or возвращают один из операндов. Если вам нужно строго булево значение (0/1), это нужно учесть в BinOpIR для AND/OR в интерпретаторе.

Операции сравнения:

PROGRAM Comparisons;

VAR i1, i2 : INTEGER;

r1, r2 : REAL;

check : INTEGER;

BEGIN

i1 := 5; i2 := 10; r1 := 5.0; r2 := 4.99;

IF i1 = (i2 DIV 2) THEN WRITE('5 = (10 DIV 2) is TRUE', '\n');

IF i1 <> i2 THEN WRITE('5 <> 10 is TRUE', '\n');

IF NOT (i1 >= i2) THEN WRITE('NOT (5 >= 10) is TRUE (i.e. 5 < 10)', '\n');

IF NOT (r1 <= r2) THEN WRITE('NOT (5.0 <= 4.99) is TRUE (i.e. 5.0 > 4.99)', '\n');

check := 0;

IF i1 = 5 THEN check := 1;

WRITE('check is 1: ', check, '\n');

IF r1 = 5.0 THEN check := 2;

WRITE('check is 2: ', check, '\n');

END.

VI. Процедуры:

Простая процедура без параметров:

PROGRAM ProcNoParams;

PROCEDURE Greet;

BEGIN

WRITE('Hello from procedure!', '\n');

END;

BEGIN

Greet;

Greet;

END.

Процедура с параметрами:

PROGRAM ProcWithParams;

VAR global\_a : INTEGER;

PROCEDURE AddAndPrint(a : INTEGER; b : INTEGER);

VAR sum : INTEGER;

BEGIN

sum := a + b;

WRITE(a, ' + ', b, ' = ', sum, '\n');

global\_a := a; // Демонстрация доступа к параметру

END;

BEGIN

global\_a := 0;

AddAndPrint(5, 7); // 12

AddAndPrint(10, -3); // 7

WRITE('Global\_a after call: ', global\_a, '\n'); // Должно быть 10

END.

Рекурсивная процедура (факториал или Фибоначчи):

PROGRAM RecursionFactorial;

VAR result : INTEGER; // Глобальная для хранения результата, т.к. процедуры не возвращают значения

PROCEDURE Factorial(n : INTEGER; accumulator : INTEGER); // Используем параметр-аккумулятор

BEGIN

IF n = 0 THEN

BEGIN

result := accumulator;

END

ELSE IF NOT (n <= 0) THEN // Защита от отрицательных, эквивалент n > 0

BEGIN

Factorial(n - 1, accumulator \* n);

END

ELSE

BEGIN

result := -1; // Ошибка

END;

END;

BEGIN

result := 1; // Инициализация

Factorial(5, 1); // 5! = 120

WRITE('Factorial of 5 is ', result, '\n');

result := 1;

Factorial(0, 1); // 0! = 1

WRITE('Factorial of 0 is ', result, '\n');

result := 1;

Factorial(1, 1); // 1! = 1

WRITE('Factorial of 1 is ', result, '\n');

END.

Примечание: Поскольку ваши процедуры не возвращают значения, для рекурсии, вычисляющей что-то, часто приходится использовать глобальные переменные или параметры-аккумуляторы.

Взаимная рекурсия (если планируете поддерживать - требует предварительного объявления):

Паскаль требует FORWARD для этого, ваша грамматика его не содержит. Если нет FORWARD, то это не протестировать.

Процедура с локальными переменными, скрывающими глобальные (если области видимости реализованы):

PROGRAM ScopeTest;

VAR x : INTEGER;

PROCEDURE P1;

VAR x : INTEGER; // Локальная x

BEGIN

x := 20;

WRITE('Inside P1, local x = ', x, '\n');

END;

BEGIN

x := 10; // Глобальная x

WRITE('Before P1, global x = ', x, '\n');

P1;

WRITE('After P1, global x = ', x, '\n'); // Должно быть 10

END.

VIII. Краевые случаи:

Очень большие/маленькие числа (проверка на переполнение, если есть ограничения):

PROGRAM LargeNumbers;

VAR big : INTEGER; rbig: REAL;

BEGIN

big := 2000000000; // Близко к лимиту 32-bit int

rbig := 1.0e38;

WRITE(big, ' ', rbig, '\n');

big := big + big; // Может вызвать переполнение, если не longint

rbig := rbig \* rbig; // Может быть inf

WRITE(big, ' ', rbig, '\n');

END.

Деление на ноль (проверка обработки исключений в интерпретаторе):

PROGRAM DivByZero;

VAR a : INTEGER; r: REAL;

BEGIN

a := 5;

// r := a / 0; // Должно вызвать ошибку

// a := a DIV 0; // Должно вызвать ошибку

WRITE('Test will fail if uncommented', '\n');

END.