Калькулятор

Разбор задания

Условия задачи

На вход программа принимает последовательность строк, последним элементом которой является знак равенства. Если введённое выражение вычислимо, то программа выведет результат в новой строке. В противном случае сообщение об ошибке – ERROR.
Калькулятор вычисляет результат без приоритета операций. Он не поддерживает отрицательные числа.
Ограничение на вычисляемый результат: [-10000; +10000].
Калькулятор имеет 4 доступных операции:

- “+” – операция сложения
- “-” – операция вычитания
- “*” – операция умножения
- “/” – операция целочисленного деления (без остатка)
- “=” — выводит результат выражения и завершает работу программы

Дано

```
PROGRAM Calculator;
// переменные и функции
BEGIN
  Res := Calculate(Error);

  IF Error
  THEN
    WRITELN('ERROR')
  ELSE
    WRITELN(Res)
END.
```

Функция вычисления выражения

```
FUNCTION Calculate(VAR WasError: BOOLEAN): INTEGER;
VAR
  CurrentResult: INTEGER;
  NextOperator: STRING;
  ShowResult: BOOLEAN;
BEGIN
  ShowResult := FALSE;

  WHILE NOT WasError AND NOT ShowResult
  DO
    BEGIN
      READLN(CurrentResult);
      ShowResult := TRUE
    END;

  RESULT := CurrentResult
END;
```

Функция вычисления выражения

```
FUNCTION Calculate(VAR WasError: BOOLEAN): INTEGER;
VAR
  CurrentResult: INTEGER;  // явно не хватает переменных
  NextOperator: STRING;
  ShowResult: BOOLEAN;
BEGIN
  ShowResult := FALSE;

  WHILE NOT WasError AND NOT ShowResult
  DO
    BEGIN // отсутствует логика расчёта
      READLN(CurrentResult);
      ShowResult := TRUE
    END;

  RESULT := CurrentResult
END;
```

Функция вычисления выражения

```
FUNCTION Calculate(VAR WasError: BOOLEAN): INTEGER;
VAR
  ExpectOperator, ShowResult: BOOLEAN;
  NextOperator, NextNumberStr: STRING;
  CurrentResult, NextNumber: INTEGER;
BEGIN
  NextOperator := '+';
  WasError := FALSE;
  ExpectOperator := FALSE;
  ShowResult := FALSE;
  NextNumber := 0;
  CurrentResult := 0;

  WHILE NOT WasError AND NOT ShowResult
  DO
    BEGIN
      READLN(CurrentResult);
      ShowResult := TRUE
    END;
```

Основной цикл

```
FUNCTION Calculate(VAR WasError: BOOLEAN): INTEGER;
// описание переменных
BEGIN
  // инициализация переменных
  WHILE NOT WasError AND NOT ShowResult
  DO
    BEGIN
      IF ExpectOperator
      THEN
        BEGIN
          READLN(NextOperator);
          WasError := NOT IsValidOperator(NextOperator);
          ShowResult := NOT WasError AND (NextOperator = ResultOperator)
        END
      ELSE
        // ...
     ExpectOperator := NOT ExpectOperator
     // ...
   END;
  RESULT := CurrentResult
END;
```

Проверка на корректность оператора

```
FUNCTION IsValidOperator(Operator: STRING): BOOLEAN;
BEGIN
 RESULT := FALSE;
 IF (Length(Operator) = 1)
 THEN
   CASE (Operator[1]) OF
     PlusOperator,
     MinusOperator,
     MultiplyOperator,
     DivOperator,
     ResultOperator:
       RESULT := TRUE
   END
END;
```

Основной цикл

```
WHILE NOT WasError AND NOT ShowResult
DO
  BEGIN
    IF ExpectOperator
    THEN
    // считывание и проверка если ожидается оператор
    ELSE
      BEGIN
        READLN(NextNumberStr);
        NextNumber := ParseInteger(NextNumberStr, WasError);
        IF NOT WasError
        THEN
          CurrentResult := ExecuteOperation(
            CurrentResult, NextNumber, NextOperator, WasError
          )
      END;
   ExpectOperator := NOT ExpectOperator
   IF ABS(CurrentResult) > MaxResult
   THEN
     BEGIN
       WasError := TRUE;
       EXIT
     END
 END;
```

ParseInteger

```
FUNCTION ParseInteger(NumStr: STRING; VAR WasError: BOOLEAN): INTEGER;
VAR
  ErrorFlag: INTEGER;
BEGIN
  VAL(NumStr, RESULT, ErrorFlag);
  WasError := ErrorFlag <> 0
END;
```

ExecuteOperation

```
FUNCTION ExecuteOperation(FirstNum, SecondNum: INTEGER;  Operator: CHAR; VAR WasError: BOOLEAN):
INTEGER;
BEGIN
  RESULT := FirstNum;
  WasError := FALSE;

  IF (Operator = DivOperator) AND ( SecondNum = 0)
  THEN
    WasError := TRUE;

  IF NOT WasError
  THEN
    CASE Operator[1] OF
      PlusOperator:
        RESULT := FirstNum + SecondNum;
      MinusOperator:
        RESULT := FirstNum - SecondNum;
      MultiplyOperator:
        RESULT := FirstNum * SecondNum;
      DivOperator:
        RESULT := FirstNum DIV SecondNum
   END
END;
```

Спасибо