

Дробные трансформации

Разбор первого задания для 10-11 классов

Условие

- Дано: 2 правильные несократимые дроби

Условие

- Дано: 2 правильные несократимые дроби
- Действия:
 - Увеличение числителя и знаменателя первой дроби на единицу
 - Сокращение первой дроби если это возможно

Условие

- Дано: 2 правильные несократимые дроби
- Действия:
 - Увеличение числителя и знаменателя первой дроби на единицу
 - Сокращение первой дроби если это возможно
- Результат:
 - строка ERROR — знаменатель хотя бы одной дроби равен нулю
 - число -1 — изначальные дроби равны
 - число n — количество шагов для достижения второй дроби
 - число 0 — невозможно достичь второй дроби

Дробь

```
FUNCTION ReadFraction(VAR F: TEXT; VAR Frac: Fraction): BOOLEAN;  
BEGIN {ReadFraction}  
    READ(F, Frac.Numerator);  
    READ(F, Frac.Denominator);  
  
END; {ReadFraction}
```

Дробь

```
FUNCTION ReadFraction(VAR F: TEXT; VAR Frac: Fraction): BOOLEAN;  
BEGIN {ReadFraction}  
    READ(F, Frac.Numerator);  
    READ(F, Frac.Denominator);  
    ReadFraction := Frac.Denominator <> 0  
END; {ReadFraction}
```

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
2. Сокращение дроби

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
2. Сокращение дроби

Что происходит со значением дроби
в результате каждого из действий?

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
2. Сокращение дроби

	Числитель	Знаменатель	Значение
Изначально	1	5	0.2
Выполняем действие 1			
Выполняем действие 2			

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
 - Значение дроби увеличивается
2. Сокращение дроби

	Числитель	Знаменатель	Значение
Изначально	1	5	0.2
Выполняем действие 1	2	6	0.333
Выполняем действие 2			

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
 - Значение дроби увеличивается
2. Сокращение дроби
 - Значение дроби не изменяется

	Числитель	Знаменатель	Значение
Изначально	1	5	0.2
Выполняем действие 1	2	6	0.333
Выполняем действие 2	1	3	0.333

Действия с дробью (трансформации)

1. Увеличение числителя и знаменателя на единицу
 - Значение дроби увеличивается
2. Сокращение дроби
 - Значение дроби не изменяется

Вывод

Вторую дробь можно получить только тогда, когда она будет меньше первой

Сокращение дроби

```
{ Наибольший общий делитель }  
{ Greatest common divisor }  
FUNCTION GCD(A, B: INTEGER): INTEGER;  
BEGIN {GCD}  
    WHILE A <> B DO  
        IF A > B THEN  
            A := A - B  
        ELSE  
            B := B - A;  
        GCD := A  
    END; {GCD}
```

Сокращение дроби

```
{ Наибольший общий делитель }  
{ Greatest common divisor }  
FUNCTION GCD(A, B: INTEGER): INTEGER;  
VAR  
    Temp: INTEGER;  
BEGIN {GCD}  
    WHILE B <> 0  
    DO  
        BEGIN  
            Temp := B;  
            B := A MOD B;  
            A := Temp  
        END;  
    GCD := A  
END; {GCD}
```

Сокращение дроби

```
{ Наибольший общий делитель }
{ Greatest common divisor }
FUNCTION GCD(A, B: INTEGER): INTEGER;
VAR
    Temp: INTEGER;
BEGIN {GCD}
    WHILE B <> 0
    DO
        BEGIN
            Temp := B;
            B := A MOD B;
            A := Temp;
        END;
    GCD := A
END; {GCD}
```

[illegible]

Сокращение дроби

```
{ Наибольший общий делитель }
{ Greatest common divisor }
FUNCTION GCD(A, B: INTEGER): INTEGER;
VAR
    Temp: INTEGER;
BEGIN {GCD}
    WHILE B <> 0
    DO
        BEGIN
            Temp := B;
            B := A MOD B;
            A := Temp;
        END;
    GCD := A
END; {GCD}
```

Действие	A	B	Temp
Дано	36	24	
Temp := B			24
B := A MOD B		12	
A := Temp	24		
B <> 0	TRUE		
Temp := B			12
B := A MOD B		0	
A := Temp	12		
B <> 0	FALSE		

Сокращение дроби

```
PROCEDURE SimplifyFraction(VAR Frac: Fraction);  
VAR  
    CommonDivisor: INTEGER;  
BEGIN {SimplifyFraction}  
    CommonDivisor := GCD(Frac.Numerator, Frac.Denominator);  
    Frac.Numerator := Frac.Numerator DIV CommonDivisor;  
    Frac.Denominator := Frac.Denominator DIV CommonDivisor  
END; {SimplifyFraction}
```

Вспомогательные функции

```
FUNCTION AreFractionsEqual(Frac1, Frac2: Fraction): BOOLEAN;  
BEGIN {AreFractionsEqual}  
    AreFractionsEqual :=  
        Frac1.Numerator * Frac2.Denominator = Frac2.Numerator * Frac1.Denominator  
END; {AreFractionsEqual}
```

```
FUNCTION IsFractionLess(Frac1, Frac2: Fraction): BOOLEAN;  
BEGIN {IsFractionLess}  
    IsFractionLess :=  
        Frac1.Numerator * Frac2.Denominator < Frac2.Numerator * Frac1.Denominator  
END; {IsFractionLess}
```

Подсчет шагов

```

FUNCTION CalculateFractionTransformationCount(VAR Frac1, Frac2: Fraction): INTEGER;
VAR
    NumSteps: INTEGER;
BEGIN {CalculateFractionTransformationCount}
    CalculateFractionTransformationCount := -1;
    IF AreFractionsEqual(Frac1, Frac2)
    THEN
        EXIT;
    END;
END; {CalculateFractionTransformationCount}

```

Подсчет шагов

```
FUNCTION CalculateFractionTransformationCount(VAR Frac1, Frac2: Fraction): INTEGER;
VAR
    NumSteps: INTEGER;
BEGIN {CalculateFractionTransformationCount}
    CalculateFractionTransformationCount := -1;
    IF AreFractionsEqual(Frac1, Frac2)
    THEN
        EXIT;
    NumSteps := 0;
    WHILE IsFractionLess(Frac1, Frac2)
    DO
        BEGIN
            Frac1.Numerator := Frac1.Numerator + 1;
            Frac1.Denominator := Frac1.Denominator + 1;
            SimplifyFraction(Frac1);
            NumSteps := NumSteps + 1
        END;
    END;
END; {CalculateFractionTransformationCount}
```

Подсчет шагов

```
FUNCTION CalculateFractionTransformationCount(VAR Frac1, Frac2: Fraction): INTEGER;
VAR
    NumSteps: INTEGER;
BEGIN {CalculateFractionTransformationCount}
    CalculateFractionTransformationCount := -1;
    IF AreFractionsEqual(Frac1, Frac2)
    THEN
        EXIT;
    NumSteps := 0;
    WHILE IsFractionLess(Frac1, Frac2)
    DO
        BEGIN
            Frac1.Numerator := Frac1.Numerator + 1;
            Frac1.Denominator := Frac1.Denominator + 1;
            SimplifyFraction(Frac1);
            NumSteps := NumSteps + 1
        END;
    IF AreFractionsEqual(Frac1, Frac2)
    THEN
        CalculateFractionTransformationCount := NumSteps
    ELSE
        CalculateFractionTransformationCount := 0
    END; {CalculateFractionTransformationCount}
```

Основная программа

```
VAR
  Fraction1, Fraction2: Fraction;
  NumSteps: INTEGER;

BEGIN {FractionTransformation}
  IF NOT ReadFraction(INPUT, Fraction1) OR NOT ReadFraction(INPUT, Fraction2)
  THEN
    WRITELN('ERROR')
  ELSE
    WRITELN(CalculateFractionTransformationCount(Fraction1, Fraction2))
END. {FractionTransformation}
```

Вопросы

Разбор задания №2

Анализ ДНК

Постановка задачи

Вы работаете в биотехнологической компании, которая разрабатывает алгоритмы для анализа цепочек ДНК. В процессе исследований биоинженеры представляют последовательности фрагментов ДНК в виде квадратных матриц, где каждая клетка содержит символ нуклеотида (A, T, C, G).

Было замечено, что некоторые виды мутаций проявляются как палиндромные последовательности нуклеотидов вдоль строки, столбца или диагонали. Палиндром — это биологический маркер симметричной мутации, и его наличие может указывать на потенциально нестабильные участки ДНК. Необходимо найти все палиндромы.

Постановка задачи

Формат входных данных:

На вход программа принимает размер матрицы N ($2 \leq N \leq 10$), потом на N строках по N символов нуклеотидов (**A, T, C, G**). Необходимо найти все палиндромы (слова, читающиеся одинаково слева направо и справа налево), которые можно получить:

- В строках
- В столбцах
- На главных диагоналях (из левого верхнего в правый нижний угол и наоборот)

Гарантируется, что матрица вводится нужного размера.

Постановка задачи

Формат выходных данных:

- После нахождения всех уникальных палиндромов, программа выводит их в алфавитном порядке.
- Если палиндромов нет, вывести **'NONE'**.
- Если в матрице попадетсся символ, не являющийся нуклеотидом (**A, T, C, G**), вывести **'ERROR'** и завершить программу.
- Если N не входит в допустимый диапазон, вывести **'ERROR'** и завершить программу.

Примеры входных и выходных данных

Ввод	Вывод	Объяснение
3 ACA CGC TTT	ACA CGC TTT	В этой матрице присутствуют только палиндромы в строках
3 ACA CAC ACA	AAA ACA CAC	В этой матрице присутствуют палиндромы в строках (ACA, CAC), в столбцах (ACA, CAC) и на главных диагоналях (AAA), после проверки на уникальность и сортировке остаются AAA, ACA, CAC
4 ATCG AAAC GTAA TTTC	NONE	В этой матрице отсутствуют палиндромы.
4 ABCD EFGH IJKL MNOP	ERROR	В матрице присутствуют символы, не являющиеся нуклеотидами (A, T, C, G)
52 ...	ERROR	Размер матрицы не может превышать 10.

Заготовка

```
PROCEDURE SortPalindromesLexicographically(VAR PalindromeCount: INTEGER; VAR Palindromes: PalindromeArray);
VAR
  I, J: INTEGER;
  Temp: STRING;
BEGIN
  FOR I := 1 TO PalindromeCount - 1
  DO
    FOR J := I + 1 TO PalindromeCount
    DO
      IF Palindromes[I] > Palindromes[J]
      THEN
        BEGIN
          Temp := Palindromes[I];
          Palindromes[I] := Palindromes[J];
          Palindromes[J] := Temp;
        END
      END;
    END;
  END;
```

Заготовка

```
PROCEDURE ReadMatrix(VAR MatrixSize: INTEGER; VAR Matrix: MatrixType);  
VAR  
    I, J: INTEGER;  
BEGIN  
    READLN(MatrixSize);  
    FOR I := 1 TO MatrixSize  
    DO  
        BEGIN  
            FOR J := 1 TO MatrixSize  
            DO  
                READ(Matrix[I, J]);  
            READLN  
        END  
    END;  
END;
```

Заготовка

```
PROCEDURE PrintPalindromes(VAR PalindromeCount: INTEGER; VAR Palindromes: PalindromeArray);  
VAR  
    I: INTEGER;  
BEGIN  
    IF PalindromeCount = 0  
    THEN  
        WRITELN('NONE')  
    ELSE  
        FOR I := 1 TO PalindromeCount  
        DO  
            WRITELN(Palindromes[I])  
        END;  
END;
```


Анализ заготовки

Что мы можем:

- Считывать нашу матрицу
- Выводить нашу матрицу
- Сортировать палиндромы

Но:

- Нет обработки ошибок
- Нет палиндромов(

Можно сделать лучше

Добавляем константы, типы и глобальные переменные

CONST

```
Nucleotides = ['A', 'C', 'T', 'G'];  
MaxN = 10;  
MinN = 2;
```

TYPE

```
MatrixType = ARRAY[1 .. MaxN, 1 .. MaxN] OF CHAR;  
PalindromeArray = ARRAY[1 .. MaxN * 2 + 2] OF STRING;
```

VAR

```
Matrix: MatrixType;  
Palindromes: PalindromeArray;  
PalindromeCount, MatrixSize: INTEGER;  
Error: BOOLEAN;
```

Добавляем обработку неправильной записи

```
PROCEDURE ReadMatrix(VAR MatrixSize: INTEGER; VAR Matrix: MatrixType; VAR Error: BOOLEAN);
VAR
  I, J: INTEGER;
BEGIN
  READLN(MatrixSize);
  IF NOT (MatrixSize IN [MinN .. MaxN])
  THEN
    BEGIN
      Error := TRUE;
      EXIT
    END;
  FOR I := 1 TO MatrixSize
  DO
    BEGIN
      FOR J := 1 TO MatrixSize
      DO
        BEGIN
          READ(Matrix[I, J]);
          IF NOT (Matrix[I, J] IN Nucleotides)
          THEN
            BEGIN
              Error := TRUE;
              EXIT
            END
          END;
        READLN
      END
    END;
  END;
END;
```

Добавляем основной блок программы

```
BEGIN
  ReadMatrix(MatrixSize, Matrix, Error);
  IF Error
  THEN
    BEGIN
      WRITELN('ERROR');
      EXIT
    END;
  SearchPalindromesInMatrix(Palindromes, PalindromeCount, MatrixSize, Matrix);
  SortPalindromesLexicographically(PalindromeCount, Palindromes);
  PrintPalindromes(PalindromeCount, Palindromes)
END.
```

Добавляем функцию поиска палиндромов

```
PROCEDURE SearchPalindromesInMatrix(VAR Palindromes: PalindromeArray; VAR PalindromeCount, MatrixSize: INTEGER; Matrix: MatrixType);
VAR
    RowWord, ColWord, MainDiagonal, SecondaryDiagonal: STRING;
    J, I: INTEGER;
BEGIN
    PalindromeCount := 0;
    FOR I := 1 TO MatrixSize
    DO
        BEGIN
            RowWord := '';
            ColWord := '';
            FOR J := 1 TO MatrixSize
            DO
                BEGIN
                    RowWord := RowWord + Matrix[I, J];
                    ColWord := ColWord + Matrix[J, I]
                END;
                AddIfUniquePalindrome(PalindromeCount, Palindromes, RowWord);
                AddIfUniquePalindrome(PalindromeCount, Palindromes, ColWord)
            END;
            MainDiagonal := '';
            SecondaryDiagonal := '';
            FOR I := 1 TO MatrixSize
            DO
                BEGIN
                    MainDiagonal := MainDiagonal + Matrix[I, I];
                    SecondaryDiagonal := SecondaryDiagonal + Matrix[I, MatrixSize - I + 1];
                END;
                AddIfUniquePalindrome(PalindromeCount, Palindromes, MainDiagonal);
                AddIfUniquePalindrome(PalindromeCount, Palindromes, SecondaryDiagonal)
            END;
        END;
    END;
```

Добавляем функцию проверки на уникальность

```
PROCEDURE AddIfUniquePalindrome(VAR PalindromeCount: INTEGER; VAR Palindromes: PalindromeArray; Str: STRING);
VAR
    I: INTEGER;
BEGIN
    IF NOT IsPalindrome(Str)
    THEN
        EXIT;
    FOR I := 1 TO PalindromeCount
    DO
        IF Palindromes[I] = Str
        THEN
            EXIT;
        PalindromeCount := PalindromeCount + 1;
        Palindromes[PalindromeCount] := Str
    END;
```

Добавляем функцию проверки на палиндром

```
FUNCTION IsPalindrome(Str: STRING): BOOLEAN;  
VAR  
    Left, Right: INTEGER;  
BEGIN  
    Left := 1;  
    Right := LENGTH(Str);  
    WHILE (Left < Right) AND (Str[Left] = Str[Right])  
    DO  
        BEGIN  
            Left := Left + 1;  
            Right := Right - 1  
        END;  
    IsPalindrome := Left >= Right  
END;
```


Что изменилось

Теперь мы можем:

- Проверить является ли строка палиндромом
- Проверить уникальность палиндрома
- Обработываем ошибки

Вопросы

Разбор задания №3

Система обнаружения

Постановка задачи

На складе установлены датчики, ориентированные в одном из четырёх направлений (вверх, вниз, влево, вправо). Эти датчики испускают сигналы в выбранном направлении для сканирования коридоров. Сигнал продвигается по прямой, пока не встретит препятствие в виде стены или не выйдет за границы склада. Если на пути сигнала находится другой датчик, система фиксирует его и помечает как "обнаруженный". Задача — смоделировать прохождение сигнала от каждого датчика и определить, какие объекты будут зафиксированы системой.

Необходимо промоделировать испускание сигналов датчиками:

1. Если сигнал попадает в другой датчик, этот датчик помечается как обнаруженный (латинский символ 'O')
2. Если сигнал попадает в стену ('I' или '-') или выходит за границы склада, ничего не происходит
3. Сигналы в конечную точку прилетают мгновенно

Постановка задачи

Формат входных данных:

На вход программе подаются два числа N и M — длина и ширина склада. N и M находятся в диапазоне от 2 до 10. Затем вводится план склада размером $N \times M$, состоящее из символов:

- Стены:
 - 'I' (вертикальная стена)
 - '-' (горизонтальная стена)
- Пустое пространство:
 - '.'
- Датчики:
 - '>' (Датчик, ориентированный вправо)
 - '<' (Датчик, ориентированный влево)
 - '^' (Датчик, ориентированный вверх)
 - 'v' (Датчик, ориентированный вниз)

Гарантируется, что план склада вводится нужного размера.

Постановка задачи

Формат выходных данных:

- Если код выполнен успешно, то
 - На первой строчке пишется количество обнаруженных датчиков, если их нет, то написать **'NONE'**.
 - Показать склад после моделирования сигналов (с отображением обнаруженных датчиков).
- Если встречен неопознанный символ, вывести **'ERROR'** и завершить программу.
- Если N или M не входит в допустимый диапазон, вывести **'ERROR'** и завершить программу.

Примеры входных и выходных данных

Ввод	Вывод	Объяснение
5 7>>...< .-...-. .^...<.	3 2 3 2 7 4 2>0...0 .-...-. .0...<.	Датчики с позиций (2;3) и (2;7) попали друг в друга. Датчик с позиции (2;2) попал в (2;3). Датчик с позиции (4;2) попал в стену. Датчик с позиции (6;6) попал в (4;2)
2 2 <> ..	NONE <> ..	Ни один из датчиков не попал в другие датчики
2 2 >! ..	ERROR	Неопознанный символ
52 ...	ERROR	Размер склада не должен превышать 10

Заготовка

```
PROCEDURE ReadField(VAR M, N: INTEGER; VAR Field: FieldType);
VAR
    X, Y: INTEGER;
    Ch: CHAR;
BEGIN
    READLN(M, N);
    FOR Y := 1 TO M
    DO
        BEGIN
            FOR X := 1 TO N
            DO
                BEGIN
                    READ(Ch);
                    Field[X, Y] := Ch
                END;
            READLN
        END
    END;
END;
```


Заготовка

```
PROCEDURE PrintField(Field: FieldType; M, N: INTEGER);  
VAR  
    X, Y: INTEGER;  
BEGIN  
    FOR Y := 1 TO M  
    DO  
        BEGIN  
            FOR X := 1 TO N  
            DO  
                WRITE(Field[X, Y]);  
            WRITELN  
        END  
    END  
END;
```

Анализ заготовки

Что мы имеем:

- Процедура, считывающая поле
- Процедура, печатающая поле

Но:

- Нет обработки ошибок
- Не реализована работа датчиков

Можно сделать лучше

Добавляем константы, типы и глобальные переменные

```
CONST
    MAX_SIZE = 10;
    MIN_SIZE = 2;
    SENSOR_SYMBOLS = ['^', 'v', '<', '>'];
    WALL_SYMBOLS = ['|', '-'];
    SPACE_SYMBOLS = ['.'];
    ALLOWED_SYMBOLS = SENSOR_SYMBOLS + WALL_SYMBOLS + SPACE_SYMBOLS;
    DETECTED_SENSOR = '0';
```

```
TYPE
    FieldType = ARRAY[1 .. MAX_SIZE, 1 .. MAX_SIZE] OF CHAR;
```

```
VAR
    InputField, DetectedField: FieldType;
    N, M, DetectedSensorsCount: INTEGER;
    Error: BOOLEAN;
```

Добавляем обработку неправильной записи

```
PROCEDURE ReadField(VAR Error: BOOLEAN);
VAR
  X, Y: INTEGER;
  Ch: CHAR;
BEGIN
  Error := FALSE;
  READLN(M, N);
  IF NOT (N IN [MIN_SIZE .. MAX_SIZE]) OR NOT (M IN [MIN_SIZE .. MAX_SIZE])
  THEN
    BEGIN
      Error := TRUE;
      EXIT
    END;
  FOR Y := 1 TO M
  DO
    BEGIN
      FOR X := 1 TO N
      DO
        BEGIN
          READ(Ch);
          IF NOT (Ch IN ALLOWED_SYMBOLS)
          THEN
            BEGIN
              Error := TRUE;
              EXIT
            END;
          InputField[X, Y] := Ch;
          DetectedField[X, Y] := Ch
        END;
      READLN
    END
  END
END;
```

Добавляем основной блок программы

```
BEGIN
  ReadField(Error);
  IF Error
  THEN
    BEGIN
      WRITELN('ERROR');
      EXIT
    END;
  SimulateAllEmissions;
  GetDetectedSensorsCount(DetectedSensorsCount);
  IF DetectedSensorsCount = 0
  THEN
    WRITELN('NONE')
  ELSE
    WRITELN(DetectedSensorsCount);
  PrintField(DetectedField)
END.
```

Добавляем функцию симуляции работы датчиков

```
PROCEDURE SimulateAllEmissions;  
VAR  
  X, Y: INTEGER;  
BEGIN  
  FOR Y := 1 TO M  
  DO  
    FOR X := 1 TO N  
    DO  
      IF InputField[X, Y] IN SENSOR_SYMBOLS  
      THEN  
        SimulateEmission(X, Y);  
      END IF;  
    END DO;  
  END DO;  
END;
```

```

PROCEDURE SimulateEmission(StartX, StartY: INTEGER);
VAR
  X, Y: INTEGER;
  DeltaX, DeltaY: INTEGER;
BEGIN
  CASE InputField[StartX, StartY] OF
    '>': BEGIN DeltaY := 0; DeltaX := 1 END;
    '<': BEGIN DeltaY := 0; DeltaX := -1 END;
    '^': BEGIN DeltaY := -1; DeltaX := 0 END;
    'v': BEGIN DeltaY := 1; DeltaX := 0 END
  END;
  X := StartX + DeltaX;
  Y := StartY + DeltaY;
  WHILE (Y IN [1 .. M]) AND (X IN [1 .. N])
  DO
    BEGIN
      IF (InputField[X, Y] IN WALL_SYMBOLS)
      THEN
        EXIT;
      IF (InputField[X, Y] IN SENSOR_SYMBOLS)
      THEN
        BEGIN
          DetectedField[X, Y] := DETECTED_SENSOR;
          EXIT
        END;
      Y := Y + DeltaY;
      X := X + DeltaX
    END
  END;
END;

```

Добавляем функцию симуляции
работы конкретного датчика

Добавляем процедуру, подсчитывающую количество обнаруженных датчиков

```
PROCEDURE GetDetectedSensorsCount(VAR DetectedSensorsCount: INTEGER);
VAR
    Y, X: INTEGER;
BEGIN
    DetectedSensorsCount := 0;
    FOR Y := 1 TO M
    DO
        FOR X := 1 TO N
        DO
            IF DetectedField[X, Y] = DETECTED_SENSOR
            THEN
                DetectedSensorsCount := DetectedSensorsCount + 1
            END IF;
        END DO;
    END DO;
END;
```

Что изменилось

Теперь мы:

- Можем провести симуляцию работы датчиков, не затрагивая введенное поле
- Можем провести симуляцию работы конкретного датчика
- Обрабатываем ошибки

Вопросы