1) Сумма ряда. Итерационные алгоритмы.

Итерационные алгоритмы: последующий член член ряда зависит от предыдущего (не вычисляется заново от начального значения)

Вычислить сумму ряда:

Основная программа: ввод исходных данных (параметр Х, точность вычисления, количество членов ряда); вывод двух практически одинаковых результатов

Две подпрограммы:

1. с заданной точностью ε (0 < ε < 1): либо по разнице двух сумм, либо по члену ряда. Вывести на экран вычисленную сумму, количество использованных при ее вычислении членов ряда и каждый член ряда.

2. Вычислить n-ое приближение. Входным n является количество, вычисленное с помощью подпрограммы 1.

При выполнении задания запрещается использовать одномерные массивы и встроенные функции возведения в степень.

Вывод из подпрограммы:

X=0.5 Сумма

1 = 0.25 0.25

2 = -2.25 -2.0

3 = 1.5 -0.5

 4 = -2.33333 -2.83333

Вывод из основной программы:

Х = 0.5

Сумма ряда = -2.8333

Ряд: x - (2\*x^2)/3! + (3\*x^3)/4! - ...

Решение:

1. **Основная программа** (**PROGRAM sum\_series ... END PROGRAM sum\_series**): это основная часть кода, которая выполняется при запуске. В ней происходит ввод данных, вызов подпрограмм и вывод результатов.
2. **Подпрограммы** (**SUBROUTINE ... END SUBROUTINE**): это блоки кода, которые можно вызывать из основной программы или других подпрограмм. В нашем коде есть две подпрограммы:
   * **compute\_sum\_epsilon** вычисляет сумму ряда с заданной точностью.
   * **compute\_sum\_n** вычисляет сумму ряда на основе заданного числа членов ряда *n*.
3. **Функция** (**FUNCTION ... END FUNCTION**): это блок кода, который возвращает значение. В нашем коде есть одна функция:
   * **factorial** вычисляет факториал заданного числа.

Таким образом, в этом коде есть и основная программа, и подпрограммы, и функция.

1. **PROGRAM sum\_series**: Объявление начала основной программы с названием **sum\_series**.
2. **IMPLICIT NONE**: Указывает компилятору, что все переменные должны быть явно объявлены. Это предотвращает ошибки из-за неявных типов переменных.
3. **REAL :: x, epsilon, sum1, sum2**: Объявление вещественных переменных. Эти переменные будут использоваться для хранения значения **x**, точности **epsilon** и двух разных сумм ряда.
4. **INTEGER :: n**: Объявление целочисленной переменной для хранения количества членов ряда.

5-6. **PRINT \*, "Введите значение X:"** и **READ \*, x**: Вывод запроса на ввод значения *x* и чтение этого значения.

7-8. **PRINT \*, "Введите точность вычисления:"** и **READ \*, epsilon**: Вывод запроса на ввод значения точности и чтение этого значения.

1. **CALL compute\_sum\_epsilon(x, epsilon, sum1, n)**: Вызов подпрограммы для вычисления суммы ряда с заданной точностью **epsilon**. Результаты будут сохранены в **sum1** и **n**.
2. **CALL compute\_sum\_n(x, n, sum2)**: Вызов подпрограммы для вычисления суммы ряда на основе *n* членов ряда. Результат сохранится в **sum2**.

11-12. Вывод результатов: значения *x* и вычисленной суммы ряда.

1. **CONTAINS**: Ключевое слово, указывающее начало блока внутренних функций и подпрограмм основной программы.

14-29. **SUBROUTINE compute\_sum\_epsilon(x, epsilon, sum, n) ... END SUBROUTINE compute\_sum\_epsilon**: Подпрограмма, вычисляющая сумму ряда с заданной точностью. Внутри используется цикл **DO WHILE** для итеративного добавления членов ряда к сумме до тех пор, пока абсолютное значение текущего члена ряда не станет меньше **epsilon**.

30-44. **SUBROUTINE compute\_sum\_n(x, n, sum) ... END SUBROUTINE compute\_sum\_n**: Подпрограмма для вычисления суммы ряда на основе *n* членов ряда. Внутри используется цикл **DO** для итеративного добавления членов ряда к сумме.

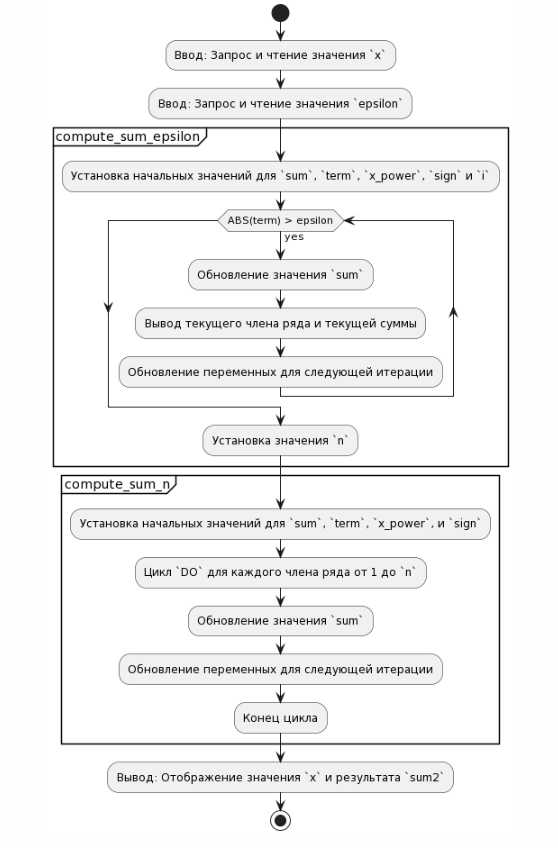
45-53. **INTEGER FUNCTION factorial(n) ... END FUNCTION factorial**: Функция для вычисления факториала числа. Внутри используется цикл **DO** для умножения чисел от 1 до *n*.

1. **END PROGRAM sum\_series**: Объявление конца основной программы.

В целом, этот код представляет собой основную программу, которая использует две подпрограммы для вычисления суммы ряда двумя разными способами, а также функцию для вычисления факториала.

(5,6; 7,8; 11-12 и т.д. – это строки соответствующего кода).

Блок-схема:



2) Формируется одномерный массив. Элементы одномерного массива вводятся из файла. Количество элементов одномерного массива зависит от количества элементов, считываемых из файла, т.е. элементы считываются ПОКА НЕ КОНЕЦ ФАЙЛА.

Требуется:

1 часть:

➢ Вывести сформированный (исходный) массив по «К (вводится)» элементов построчно.

➢ Должен быть удобный для пользователя интерфейс.

2 часть:

➢ Вывести результирующий массив в строку.

Подсказка: вместо значения элемента массива чаще всего необходим его индекс!

Написать подпрограмму замены элементов, стоящих на нечетных местах, любым введенным значением. С помощью подпрограммы заменить соответствующие элементы массива средним арифметическим значением.

Пробовала оператор eof в цикле с условием, не получилось, возможно, надо написать подпрограмму чтения файла, пока он не закончится.

Решение:

1. **PROGRAM main\_program**: Определение начала основной программы с названием **main\_program**.
2. **IMPLICIT NONE**: Указывает компилятору, что все переменные должны быть явно объявлены.
3. **REAL, ALLOCATABLE :: array(:)**: Объявление динамического массива **array** типа **REAL**.
4. **REAL :: avg\_value**: Объявление переменной **avg\_value** для хранения среднего арифметического значения массива.
5. **INTEGER :: k, i**: Объявление целочисленных переменных **k** (для количества элементов в строке при выводе) и **i** (для индексации).
6. **CALL read\_array\_from\_file(array)**: Вызов подпрограммы для чтения массива из файла **input.txt**.

7-9. Вывод запроса на ввод количества элементов для вывода в одной строке и чтение этого значения.

1. **CALL print\_array(array, k)**: Вызов подпрограммы для вывода элементов массива построчно с заданным количеством элементов в строке.
2. **avg\_value = SUM(array) / SIZE(array)**: Вычисление среднего арифметического значения элементов массива.
3. **CALL replace\_odd\_elements(array, avg\_value)**: Вызов подпрограммы для замены элементов массива, стоящих на нечетных местах, на среднее арифметическое значение.

13-16. Вывод результирующего массива.

1. **CONTAINS**: Разделитель, который указывает на начало блока внутренних подпрограмм основной программы.

18-37. **SUBROUTINE read\_array\_from\_file(arr) ... END SUBROUTINE read\_array\_from\_file**: Подпрограмма для чтения массива из файла **input.txt**. - Сначала открывается файл для чтения. - Выполняется подсчет количества элементов в файле. - Выделяется память для массива и производится чтение значений из файла. - Файл закрывается.

38-47. **SUBROUTINE print\_array(arr, k) ... END SUBROUTINE print\_array**: Подпрограмма для вывода элементов массива построчно с заданным количеством элементов в строке.

48-57. **SUBROUTINE replace\_odd\_elements(arr, value) ... END SUBROUTINE replace\_odd\_elements**: Подпрограмма для замены элементов массива, стоящих на нечетных позициях, заданным значением.

1. **END PROGRAM main\_program**: Определение конца основной программы.

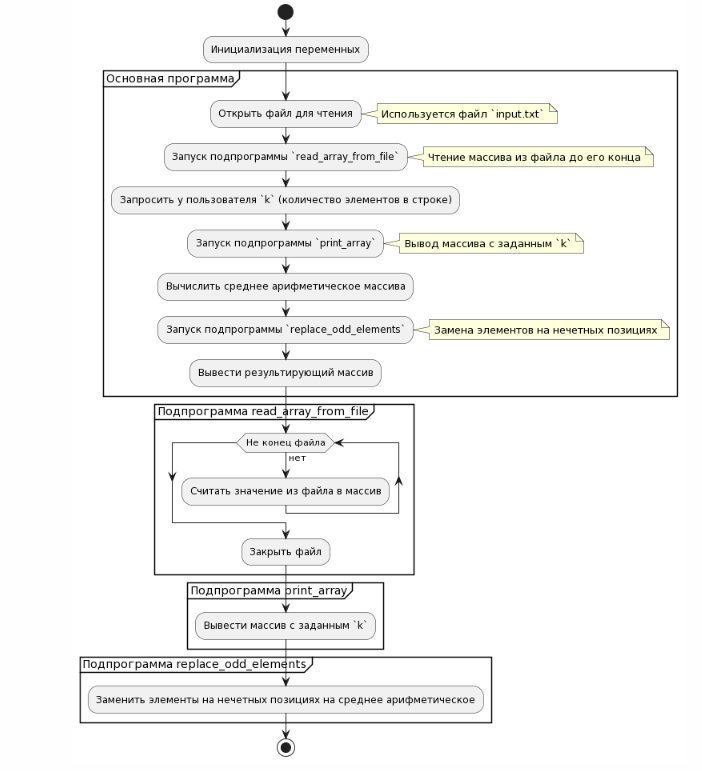
В целом, данный код предназначен для чтения массива из файла, вывода его в удобном формате, и замены элементов массива на нечетных позициях средним арифметическим значением всего массива.

Краткое описание:

1. Программа начинается с чтения массива из файла **input.txt**.
2. Пользователю предлагается ввести количество элементов для вывода в одной строке, и массив выводится в соответствии с этим значением.
3. Элементы массива на нечетных позициях заменяются средним арифметическим значением всего массива.
4. Результирующий массив выводится на экран.

Пожалуйста, убедитесь, что у вас есть файл **input.txt** в той же папке, что и ваша программа, перед тем как запускать этот код.

Блок схема:



3) Дана целочисленная матрица размерностью (NxM).

Общие требования:

→ ввод и вывод элементов матрицы осуществлять по элементам строк – построчно

→ вывод исходной матрицы обязателен

→ интерфейс должен быть понятным для пользователя

Основная программа:

ввод исходных данных, вызов подпрограммы, вывод результата

Подсказка: вместо значения элемента массива чаще всего необходим его индекс!

Составить подпрограмму нахождения индекса минимального элемента кратного заданному целому числу в целочисленном одномерном массиве. Заменить нулями все элементы, соответствующие данному критерию, в исходном двумерном массиве.

Решение:

1. **PROGRAM matrix\_program**: Определение начала основной программы с названием **matrix\_program**.
2. **IMPLICIT NONE**: Указание компилятору, что все переменные должны быть явно объявлены.
3. **INTEGER, ALLOCATABLE :: matrix(:,:)**: Объявление динамической двумерной матрицы **matrix**.
4. **INTEGER :: n, m, i, j, num**: Объявление целочисленных переменных **n** и **m** (размерности матрицы), **i** и **j** (для индексации) и **num** (заданное пользователем число).

6-9. Ввод размерности матрицы и инициализация ее.

11-16. Ввод элементов матрицы пользователем.

18-22. Вывод исходной матрицы на экран.

24-26. Запрос и чтение заданного числа от пользователя.

28-32. Применение подпрограммы **replace\_elements** к каждой строке матрицы для замены элементов, удовлетворяющих условию.

34-38. Вывод измененной матрицы на экран.

1. **CONTAINS**: Разделитель, указывающий на начало блока внутренних подпрограмм основной программы.

42-56. **SUBROUTINE replace\_elements(row, num) ... END SUBROUTINE replace\_elements**: Подпрограмма для замены элементов в строке матрицы на ноль, если они кратны заданному числу и если в этой строке есть минимальный элемент, удовлетворяющий этому условию.

58-71. **INTEGER FUNCTION find\_min\_index(arr, num) ... END FUNCTION find\_min\_index**: Функция, которая возвращает индекс минимального элемента массива, кратного заданному числу. Если такого элемента нет, возвращает -1.

1. **END PROGRAM matrix\_program**: Определение конца основной программы.

**Общий принцип работы программы**: Программа сначала запрашивает у пользователя размерность матрицы и ее элементы. Затем исходная матрица выводится на экран. Далее пользователь вводит заданное число. Производится замена элементов матрицы в соответствии с критерием (если есть минимальный элемент, кратный заданному числу, все элементы строки, кратные этому числу, заменяются нулями). Результирующая матрица выводится на экран.

**Краткое описание кода**:

1. Программа начинается с ввода размерности матрицы и ее элементов.
2. Затем исходная матрица выводится на экран.
3. Пользователь вводит заданное число.
4. Производится замена элементов матрицы в соответствии с критерием (если есть минимальный элемент, кратный заданному числу, все элементы строки, кратные этому числу, заменяются нулями).
5. Результирующая матрица выводится на экран.

В блоке **CONTAINS** находятся две подпрограммы:

1. **replace\_elements**: заменяет элементы одномерного массива (строки матрицы) на ноль, если они кратны заданному числу и если в этой строке есть минимальный элемент, удовлетворяющий критерию.
2. **find\_min\_index**: находит индекс минимального элемента массива, который кратен заданному числу. Если такого элемента нет, возвращает -1.

Блок-схема:

