**ЛАБОРАТОРНАЯ РАБОТА № 1**

РЕКУРРЕНТНЫЕ АЛГОРИТМЫ

1. ТЕОРИЯ И ПОЯСНЕНИЯ

I.1. Рекуррентные последовательности

Пусть задана последовательность числовых значений , для которой выполняются соотношения вида:

 где  – константы,

.

Такие соотношения называются рекуррентными. Они показывают, что значения элементов последовательности могут быть вычислены постепенно одно за другим, если известны  первых значений. В практических задачах чаще всего .

В качестве примера использования рекуррентной последовательности рассмотрим задачу вычисления суммы



Введем обозначение:  – *k*-е слагаемое. Тогда начальный элемент последовательности слагаемых , т.е. для номеров выполняется условие . Найдем зависимость, связывающую между собой два соседних слагаемых. Запишем выражение для (*k*‑1)-го слагаемого . Выразим  через :  Получилось рекуррентное соотношение для последовательности слагаемых:

,



Назовем сумму первых k слагаемых  частичной суммой. Для последовательности частичных сумм тоже записывается рекуррентное соотношение:

,



Оба рекуррентных соотношения можно использовать в алгоритме одновременно, в одном цикле.

Рекуррентные соотношения являются основой рекуррентных или итеративных алгоритмов, смысл которых заключается в постепенном приближении вычисляемого значения к результату. Итерация – это многократно повторяющееся действие. Каждый шаг выполнения итеративного алгоритма соответствует одной итерации.

Можно выделить два варианта задач работы с рекуррентными последовательностями:

1. вычислить значение элемента с заданным номером;
2. выполнить последовательные вычисления до достижения заданной точности.

Основой итеративного алгоритма является цикл. Для первого варианта задач – это цикл FOR с известным числом шагов, для второго – цикл WHILE с условием.

Существуют функции, которые могут быть представлены в виде бесконечной суммы (в виде ряда), причем слагаемые связаны друг с другом рекуррентными соотношениями. В таких рядах выполняется условие сходимости – при добавлении одного слагаемого за другим сумма постепенно приближается к истинному значению функции для заданного значения аргумента. Необходимым условием этого является уменьшение абсолютного значения слагаемых с увеличением их номера. В практических случаях итеративный цикл нужно завершать, когда абсолютное значение очередного слагаемого становится меньше заданной точности.

I.2. Ввод исходных данных из текстового файла и вывод результатов в текстовый файл

Программирование работы с файлом разделяется на 3 этапа: открытие, чтение или запись, закрытие. Для выполнения этих действий в языке Free Pascal имеются специальные встроенные процедуры.

#### ОТКРЫТИЕ

Файл, именованная совокупность данных на внешнем носителе, формируется как последовательность компонентов (записей) определенного типа. Организация работы с файлом из программы начинается с назначения ему переменной. Все обращения к файлу из программы выполняются с помощью этой переменной. Как любая переменная она должна быть объявлена с определенным типом. Для текстовых файлов – это тип TextFile, для двоичных – File of Тип\_Компонента.

Назначение переменной файлу выполняет процедура

***AssignFile (файловая переменная, имя файла)***

Имя файла – параметр типа String. Его значение – имя файла с данными. Если исполняемый файл и файл с данными находятся в разных папках, то в этой строке должен указываться путь к файлу и его имя с расширением.

После того, как файлу назначена переменная, его можно открыть. Открытие выходного файла производится с помощью процедур:

***Rewrite (файловая переменная)***

или

***Append (файловая переменная)***

Открытие файла для создания как нового выполняется процедурой Rewrite. В файл, открытый таким образом, можно записывать информацию. Если применить Rewrite к существующему файлу, то его содержимое будет потеряно. Открытие существующего файла с целью дозаписи новых компонентов в конец файла выполняется процедурой Append.

#### ЧТЕНИЕ

Чтение – копирование информации из файла в указанную область оперативной памяти. Эта операция выполняется с помощью процедур:

Read (файловая переменная, список переменных),

Readln (файловая переменная, список переменных).

При каждом вызове одной из этих процедур производится запись одного текущего компонента файла. После вызова текущим становится следующий, соседний компонент.

#### ЗАПИСЬ

Запись – копирование информации из указанной области оперативной памяти в файл. Эта операция выполняется с помощью процедур:

Write (файловая переменная, список значений или переменных),

Writeln (файловая переменная, список значений или переменных).

При каждом вызове одной из этих процедур производится запись одного текущего компонента файла. После вызова текущим становится следующий, соседний компонент.

#### ЗАКРЫТИЕ

После завершения работы с файлом его следует закрыть, т.е. разорвать связь между файловой переменной и данными на внешнем носителе. Эти действия выполняет процедура:

***CloseFile (файловая переменная)***

Если данная процедура не используется, то файл закроется во время завершения выполнения программы, однако такое некорректное закрытие может привести к потере информации.

ПРИМЕР 1. Чтение значений входных переменных программы из текстового файла.

Var fp : TextFile;

…

AssignFile(fp, ‘InFile.txt’);

Read(fp, список переменных);

…

CloseFile(fp);

ПРИМЕР 2. Запись результатов работы программы в новый текстовый файл.

Var fp : TextFile;

…

AssignFile(fp, ‘OutFile.txt’);

Rewrite(fp);

…

Writeln(fp, список переменных);

…

CloseFile(fp);

ПРИМЕР 3. Добавление результатов работы программы в конец существующего текстового файла.

Var fp : TextFile;

…

AssignFile(fp, ‘OutFile.txt’);

Append(fp);

…

Writeln(fp, список переменных);

…

CloseFile(fp);

I.3. Системы счисления

Система счисления задает правила записи чисел. Из истории известно много примеров использования разных систем счисления или нумерации. Наиболее старые из них возникли несколько тысяч лет назад. Люди применяли специальные значки или буквы для изображения чисел. Все виды нумерации можно разделить на две группы: непозиционные и позиционные. В непозиционной системе счисления обычно каждый новый разряд числа изображается специальным знаком. Например, в римской системе: единица – I, десять – X, сто – С и т.д. В позиционной системе вклад каждой цифры в число зависит от позиции (разряда), в которой эта цифра находится. Количество разных цифр, которые можно использовать для записи чисел в позиционной системе, называется основанием системы счисления.

I.4. Представление чисел в позиционной системе счисления

Величину любого числа, заданного в позиционной системе счисления с основанием *p*, можно представить в виде полинома относительно *p*

, (1)

где  – отдельные цифры заданного числа, для которых выполняется условие  Если , то для записи можно использовать десятичные цифры. Например,  При возрастании p приходится добавлять специальные знаки, обозначающие величины больше 9:  Для систем с большими основаниями можно записывать каждую цифру в виде десятичного числа:



I.5. Рекуррентное вычисление полинома

Если вычислить значение полинома по правилам десятичной арифметики, то получится величина того же числа в десятичной системе счисления. (1) можно преобразовать, используя формулу Горнера:

 (2)

Последовательность частичных сумм  является рекуррентной:



I.6. Перевод числа из одной системы счисления в другую

Пусть в (1) заданы  и цифры . На основе (2) можно записать величину того же числа в системе с другим основанием :

 (3)

где  – цифры, для которых выполняется условие  Задача перевода заключается в вычислении этих цифр и определении их количества .

Поскольку  – целые числа, на основе (3) можно получить рекуррентную последовательность:



I.7. Системы счисления с основаниями, равными .

Целые положительные числа хранятся в памяти компьютера в двоичном формате, полностью совпадающим с представлением числа в двоичной системе счисления. Для перевода числа из двоичной системы в систему с основанием  достаточно разбить последовательность двоичных разрядов на группы по  двоичных цифр и обозначить каждую группу соответствующей цифрой новой системы счисления. Разбиение производится в направлении от младших разрядов к старшим. Перевод из системы с основанием  в двоичную заключается в последовательной записи групп двоичных разрядов для каждой 16-ричной цифры. Ниже приведена таблица соответствия для 8- и 16-ричной систем счисления:

|  |  |  |  |
| --- | --- | --- | --- |
| Группа двоичных разрядов | 8-ричная цифра (число) | 16-ричная цифра | Десятичное число |
| 0000 | 0 | 0 | 0 |
| 0001 | 1 | 1 | 1 |
| 0010 | 2 | 2 | 2 |
| 0011 | 3 | 3 | 3 |
| 0100 | 4 | 4 | 4 |
| 0101 | 5 | 5 | 5 |
| 0110 | 6 | 6 | 6 |
| 0111 | 7 | 7 | 7 |
| 1000 | 10 | 8 | 8 |
| 1001 | 11 | 9 | 9 |
| 1010 | 12 | A | 10 |
| 1011 | 13 | B | 11 |
| 1100 | 14 | C | 12 |
| 1101 | 15 | D | 13 |
| 1110 | 16 | E | 14 |
| 1111 | 17 | F | 15 |

Например, пусть задано число . Его нужно перевести в систему с основанием . Разобьем последовательность двоичных разрядов на группы по 4 и обозначим каждую группу соответствующей цифрой 16-ричной системы счисления:



Получили .

1. ЗАДАНИЯ

II.1. ЧИСЛА ФИБОНАЧЧИ

Напишите программу вычисления N-го числа Фибоначчи и вычисления суммы N первых чисел Фибоначчи. Программу следует реализовать в консольном режиме системы Lazarus.

**На входе** программы задается значение N.

**Результаты вычислений** с пояснительными текстами выведите построчно на экран.

**ИЗ ДВУХ СЛЕДУЮЩИХ ЗАДАНИЙ ДОСТАТОЧНО ВЫПОЛНИТЬ ОДНО ПО ВАШЕМУ ВЫБОРУ.**

II.2. ИТЕРАТИВНОЕ ВЫЧИСЛЕНИЕ БЕСКОНЕЧНЫХ СУММ

Напишите программу вычисления приближенных значений функции, представленной в виде ряда, в десяти равноотстоящих друг от друга точках указанного диапазона возможных значений аргумента. *Конкретную функцию каждый студент берет из приведенного ниже перечня, при этом номер функции равен порядковому номеру студента в списке группы*.

Для каждого значения аргумента выполняйте вычисления до достижения каждой из заданных границ точности и одновременно считайте соответствующее число итераций. Программу следует реализовать в консольном режиме системы Lazarus.

**На входе** программы задаются два числа: начальная и конечная точки указанного диапазона возможных значений аргумента.

**Результаты вычислений** сохраните таким образом, чтобы их можно было вывести на экран и записать в файл в виде таблицы (линии границ в таблице выводить не нужно):

|  |  |  |  |
| --- | --- | --- | --- |
| Значение аргумента  *x* | Точность вычислений  *eps* | Значение функции  *f* | Число итераций  *n* |
| начальное значение диапазона | 10-1 |  |  |
| 10-2 |  |  |
| 10-3 |  |  |
| 10-4 |  |  |
| 10-5 |  |  |
| 10-6 |  |  |
| **. . .** | 10-1 |  |  |
| 10-2 |  |  |
| 10-3 |  |  |
| 10-4 |  |  |
| 10-5 |  |  |
| 10-6 |  |  |
| конечное значение диапазона | 10-1 |  |  |
| 10-2 |  |  |
| 10-3 |  |  |
| 10-4 |  |  |
| 10-5 |  |  |
| 10-6 |  |  |

II.3. ПЕРЕВОД ЧИСЕЛ ИЗ ОДНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДРУГУЮ

Напишите программу, которая **вводит**:

1. основание системы счисления , ;
2. количество цифр в числе с введенным основанием;
3. последовательность цифр этого числа. Каждая цифра – десятичное число, в последовательности эти цифры разделяются пробелами;
4. новое основание системы счисления , .

**Результат работы** программы – последовательность цифр представления числа по новому основанию. Каждая цифра – десятичное число, в последовательности эти цифры разделяются пробелами.

Программу следует реализовать в консольном режиме системы Lazarus.

***Замечание***. Значение вводимого числа не должно превышать максимально допустимого предела для внутреннего представления целочисленных данных. Рекомендуемый способ проверки заключается в сравнении введенной последовательности цифр с цифрами максимального целого, представленного в системе с основанием .

1. ПЕРЕЧЕНЬ ФУНКЦИЙ
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 
17. 
18. 
19. 
20. 
21. 
22. 
23. 
24. 
25. 
26. 
27. 
28. 
29. 
30. 
31. 

