Оглавление

Введение

Повторение – это многократное выполнение одного или нескольких действий алгоритма. Повторение – это еще одно проявление нелинейности алгоритма. Оно может быть реализовано явно (с помощью операторов цикла) или неявно (с помощью оператора безусловного перехода).

Циклы делятся на две категории: циклы с известным числом повторений (циклы со счетчиком) и циклы с неизвестным числом повторений (циклы с условием). В первом случае для организации цикла используется специальная переменная (счетчик), значение которой меняется в заданном диапазоне с некоторым шагом. Во втором случае счетчика нет. Цикл продолжается пока выполняется некоторое условие. Его еще называют условием цикла.

Классификация циклов приведена на рис. 1.

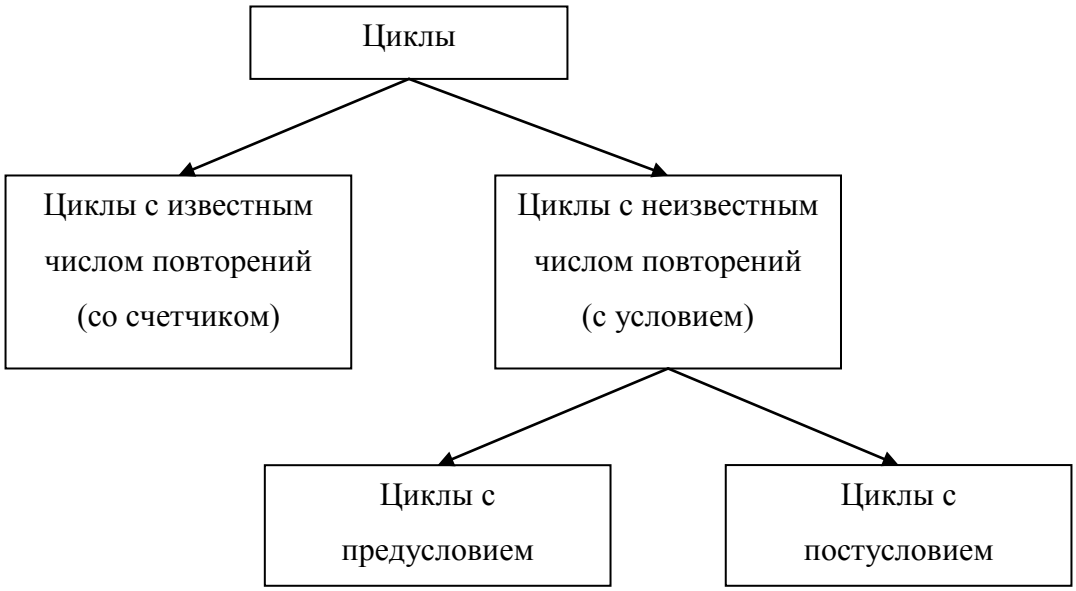

Рис. 1. Классификация циклов

1. Цикл со счетчиком

Цикл со счетчиком применяется в тех случаях, когда можно точно определить, сколько раз должны повториться операторы программы. В Visual Basic 2005 цикл со счетчиком реализуется с помощью оператора For. Рассмотрим его синтаксис.

```
For Счетчик = Нач. значение To Кон. Значение Step Шаг
    Операторы тела цикла
```

Оглавление

Next

В синтаксической структуре принято выделять две части: заголовок цикла (первая строка оператора цикла) и тело цикла (блок операторов, стоящих между строками For и Next). В старых версиях языка Basic после ключевого слова Next необходимо было указывать Счетчик цикла. В Visual Basic 2005 это является необязательным.

Выполнение цикла со счетчиком происходит в несколько этапов.

1. Заголовок цикла проверяется на отсутствие противоречий. Это возможно в двух случаях.
   - Если Начальное значение меньше Конечного Значения, то Шаг цикла должен быть больше нуля.
   - Если Начальное значение больше Конечного Значения, то Шаг цикла должен быть меньше нуля.

   Visual Basic 2005 позволяет не указывать Шаг в заголовке цикла, опуская при этом ключевое слово Step. В таких случаях Шаг цикла считается равным единице. Если заголовок цикла является противоречивым, то цикл выполняться не будет, а работа программы будет продолжена с оператора, стоящего после ключевого слова Next.
2. Если в заголовке цикла нет противоречий, то переменная Счетчик становится равной Начальному значению.
3. При данном значении Счетчика выполняются операторы тела цикла.
4. Значение счетчика изменяется на величину Шага. Если Шаг положительный, то значение Счетчика будет увеличиваться. Если Шаг отрицательный – уменьшаться. Если в заголовке цикла шаг не указан, то значение Счетчика будет увеличиваться на единицу.
5. Проверяется, попадает ли значение Счетчика в диапазон от Начального значения до Конечного значения. Если да, то происходит переход к пункту 3, и цикл выполняется еще раз. В противном случае работа цикла завершается.
6. Если среди операторов тела цикла встречается оператор Exit For, то выполнение цикла после этого оператора сразу прекращается независимо от значения Счетчика.

Оглавление

В качестве примеров работы цикла со счетчиком рассмотрим три задачи: построения таблицы значений функции, вычисление факториала и обработку совокупности числе с известным числом элементов.

1.1. Табуляция функции

Табуляция – это построение таблицы значений заранее заданной функции. Таблица строится на некотором отрезке изменения аргумента функции. Причем таблица строится не во всех точках отрезка, а только в некоторых. Существует два вида этой задачи.

В первом случае известно, что точки на отрезке расположены равномерно и расстояние между ними одинаковое. Первая точка совпадает с началом отрезка. Исходными данными при этом являются: начало и конец отрезка и расстояние между двумя соседними точками (оно называется шаг табуляции). Такая задача называется табуляция с известным шагом.

Во втором случае шаг табуляции неизвестен, но вместо него задано количество точек, расположенных на отрезке. Считается, что точки на отрезке расположены равномерно. Первая точка совпадает с началом отрезка, а последняя – с его концом. Исходными данными при этом являются: начало и конец отрезка и количество точек табуляции (другое название – количество узлов табуляции). Такая задача называется табуляция с известным числом узлов.

Рассмотрим первый случай – табуляция с известным шагом. Построим таблицу для следующей функции.

$$y = \sqrt{x+4} - \frac{1}{x}$$

Возьмем отрезок $[a;b]$, шаг табуляции обозначим h. Для ввода исходных данных будем использовать функцию `InputBox` и предусмотрим проверку правильности исходных данных. Исходные данные будем считать правильными, если левая граница отрезка меньше правой, а шаг h – положительный и не превышает длины отрезка. Результирующую таблицу значений будем выводить в окно списка `lstResult`. Внешний вид разработанной формы представлен на рис. 2. Обратите внимание, что форма предназначена для обоих типов табуляции.

Оглавление

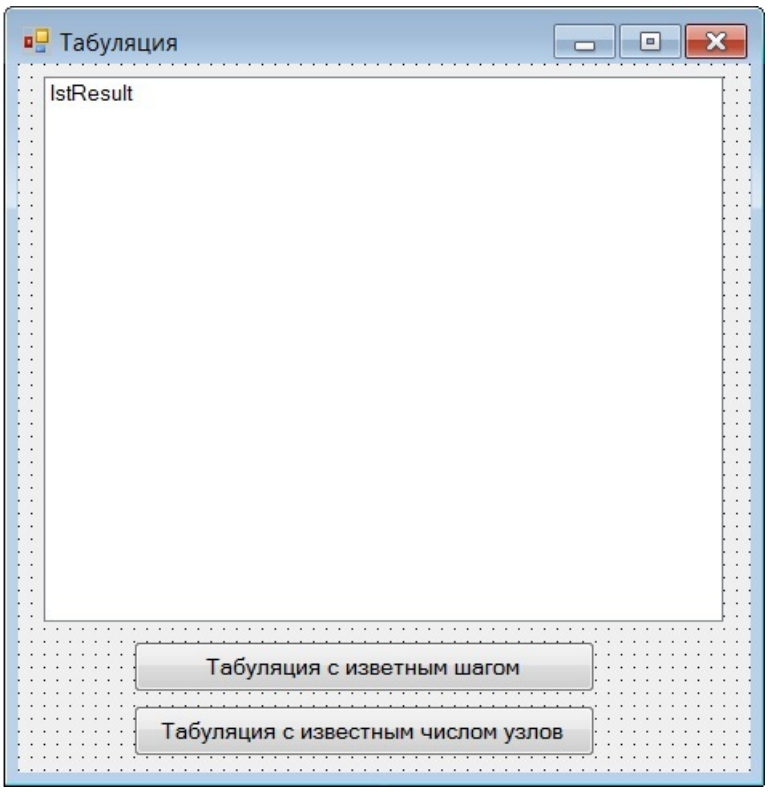

Рис. 2. Экранная форма для задачи табуляции функции

Разработку программы начнем с объявления переменных. Все необходимые нам переменные имеют тип `Single`.

```
Dim a, b, h, x, y As Single
```

Для организации повтора ввода данных в случае обнаружения ошибки пред блоком ввода поставим метку.

```
vvod:
```

Вводим с клавиатуры значения переменных `a`, `b` и `h`. Так как все эти переменные являются числами, то при вводе необходимо использовать преобразование `Val`.

```
a = Val(InputBox("Введите начало отрезка"))
b = Val(InputBox("Введите конец отрезка"))
h = Val(InputBox("Введите шаг табуляции"))
```

Проверяем правильность исходных данных. Ошибочными будут ситуации, когда левая граница отрезка больше, чем правая (`a > b`), когда шаг меньше или равен нулю (`h <= 0`) или когда шаг больше длины отрезка (`h > b - a`).

```
If a > b Or h <= 0 Or h > b - a Then
```

При обнаружении ошибки выводим сообщение и передаем управление на метку, стоящую перед блоком ввода, обеспечивая таким образом повторный ввод исходных данных.

Оглавление

```
            MsgBox("Неправильные данные")
            GoTo vvod
        End If
```

Если исходные данные введены верно, то переходим к построению таблицы значений функции. Сначала очищаем окно списка от предыдущих результатов работы программы.

```
        lstResult.Items.Clear()
```

Затем выводим в окно списка заголовок таблицы. Использование константы vbTab позволяет формировать колонки таблицы.

```
        lstResult.Items.Add("x" + vbTab + "y")
```

Организуем цикл, который будет последовательно вычислять значение функции в каждой точке. В качестве счетчика цикла используем переменную `x`, в которой хранится значение аргумента функции. В соответствии с условием задачи значение аргумента должно меняться в пределах заданного отрезка. Поэтому начальное значение счетчика (аргумента) совпадает с левой границей отрезка, а конечное – с правой. Шаг цикла совпадает с шагом изменения аргумента.

```
        For x = a To b Step h
```

Перед вычислением значения функции необходимо проверить, попадает ли значения аргумента в область допустимых значений. Для нашей функции недопустимыми являются те значения аргумента, при которых становится отрицательным подкоренное выражение или обращается в ноль знаменатель дроби.

```
            If x + 4 < 0 Or x = 0 Then
```

Если значение переменной `x` не принадлежит области допустимых значений, то в результирующей таблице вместо значения функции надо вывести слово «Ошибка». Константа `vbTab` используется для формирования колонок таблицы.

```
                lstResult.Items.Add(Str(x) + vbTab + _
                                    "Ошибка")
            Else
```

В противном случае, когда значение аргумент принадлежит области допустимых значений, вычисляем значение функции.

```
                y = Math.Sqrt(x + 4) + 1 / x
```

В окно списка выводим значение аргумента и значение функции. Константа `vbTab` используется для формирования колонок таблицы.

```
                lstResult.Items.Add(Str(x) + vbTab + Str(y))
```

Оглавление

```
        End If
    Next
```

На этом разработка программы завершена. Пример ее работы при следующих исходных данных (a = -10, b = 10, h = 2) приведен на рис. 3.

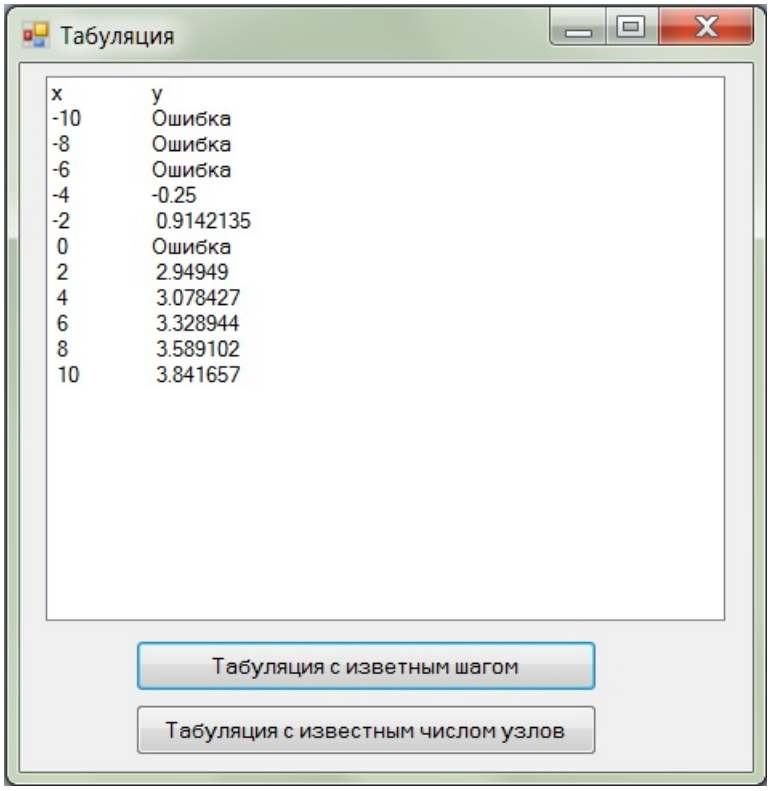

Рис. 3. Пример работы программы табуляции функции

Задача табуляции с известным числом узлов решается практически аналогично. Для этого необходимо вычислить шаг между двумя узлами. Это делается по формуле

$$h = \frac{b-a}{n-1}$$

где n – количество узлов табуляции. Формула реализуется следующим образом.

```
h = (b - a) / (n - 1)
```

Эта строка вставляется непосредственно перед циклом `For`, и задача сводится к предыдущей. Текст программы табуляции функции с известным числом узлов приведен в приложении 1.

1.2. Вычисление факториала

Рассмотрим еще одно применение цикла `For`. Это задача вычисления факториала натурального числа.

Оглавление

Факториалом натурального числа *n* называется произведение всех натуральных чисел от единицы до *n* включительно.

$$n! = 1 \cdot 2 \cdot 3 \times \ldots \times (n-1) \cdot n$$

Программная реализация этого процесса предполагает последовательное вычисление всех произведений. Сначала находится произведение 1 и 2. Полученный результат умножается на 3. Новый результат умножается на 4. И так далее до тех пор, пока не появится множитель *n*. Такой процесс называется вычисление произведения методом накопления. Рассмотрим программную реализацию этого алгоритма.

Исходные данные будем вводить с помощью функции `InputBox`. Для вывода результата будем использовать функцию `MsgBox`. Поэтому на экранной форме будет находиться только кнопка, запускающая программу.

Разработку программы начнем с описания переменных. Факториал числа является быстро растущей функцией. Поэтому для хранения результатов вычислений используют тип данных с максимальной емкостью. В Visual Basic 2005 это тип ULong. Он позволяет вычислять факториалы первых 20 чисел.

```
Dim f As ULong
```

Также нам потребуются две целых переменных: `n` – для хранения числа, факториала которого необходимо вычислить, `i` – для организации цикла. Так как значения этих переменных не будут превышать 20, то для них будем использовать тип `Byte`.

```
Dim i, n As Byte
```

Просим пользователя ввести число, факториал которого необходимо вычислить. Как всегда при вводе чисел используем преобразование `Val`.

```
n = Val(InputBox("введите натуральное число n"))
```

Начальное значение факториала полагаем равным единице, так как 1! = 1.

```
f = 1
```

Организуем цикл для вычисления факториала. Сомножители будут изменяться от 2 до `n`, что отображено в заголовке цикла.

```
For i = 2 To n
```

На каждом шаге цикла очередной сомножитель будет совпадать со значением счетчика `i`. Поэтому умножаем значение переменной `f` на `i`, а результат записываем обратно в переменную `f` вместо старого значения.

```
    f *= i
Next
```

Оглавление

После окончания цикла выводим полученные результаты. Знак плюс используется для соединения частей строки. Сначала выводим значение переменной `n`. Так как `n` – это число, то для его вывода необходимо использовать преобразование `Str`. Затем выводим знаки факториала и равенства. Наконец печатаем значение переменной `f`. Это и есть вычисленный факториал. Так как факториал – это число, то для его вывода используем преобразование `Str`.

```
MsgBox(Str(n) + "! =" + Str(f))
```

Полный текст рассмотренной программы приведен в приложении 2.

1.3. Обработка совокупности чисел с известным числом элементов

Совокупностью будем называть произвольный набор чисел, которые последовательно вводятся с клавиатуры. Совокупности чисел бывают двух видов: с известным числом элементов и с неизвестным числом элементов. В первом случае сначала указывается количество чисел, входящих в совокупность, а затем вводятся сами числа. Для работы с такими совокупностями используется цикл `For`. При обработке совокупности с неизвестным числом элементов количество ее элементов заранее неизвестно. Ввод продолжается до тех пор, пока не выполнится некоторое условие. Такая совокупность обрабатывается при помощи цикла с условием (см. раздел 2.2).

Рассмотрим основные приемы для обработки совокупности с известным количеством элементов. Основное правило при решении задач этого типа можно сформулировать следующим образом.

Совокупность всегда обрабатывается в пределах одного цикла.

Для ввода всех исходных данных будем использовать функцию `InputBox`. Так как все вводимые величины являются числами, то при вводе необходимо использовать преобразование `Val`. Элементы совокупности будем выводить в окно списка. Дадим ему имя `lstA`. Вывод организуем в две колонки. В первой колонке будем печатать порядковый номер элемента, а во второй – значение элемента. В это же окно списка будем выводить результаты вычислений. Для каждого рассматриваемого приема обработки совокупности чисел с известным количеством элементов поместим на форму отдельную кнопку. Внешний вид разработанной экранной формы приведен на рис. 4.

Оглавление

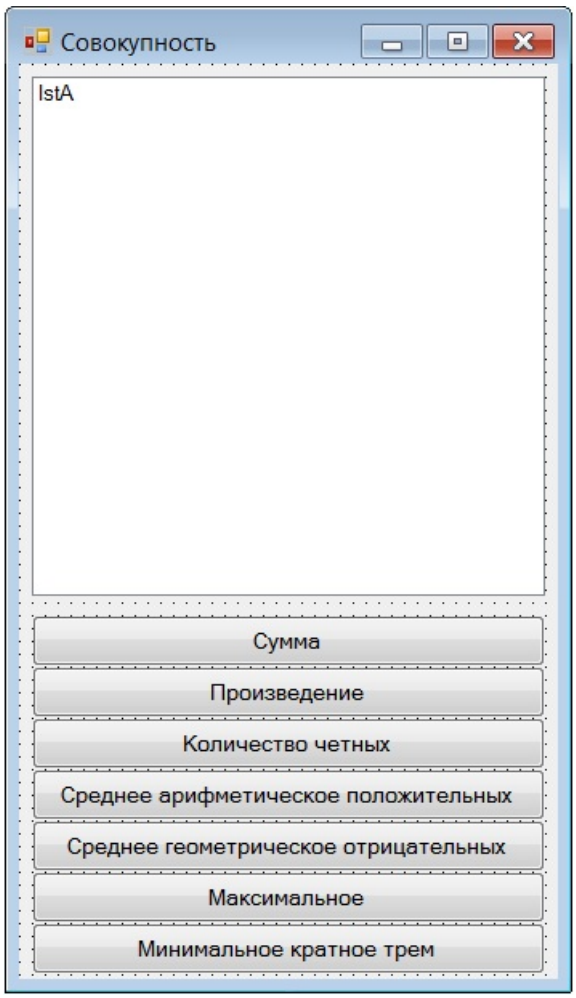

Рис. 4. Экранная форма программы обработки совокупности чисел с известным числом элементов

Рассмотрим основные приемы обработки совокупности с известным числом элементов.

Вычисление суммы чисел.

Сумма элементов совокупности вычисляется методом накопления. Для этого заводится отдельная переменная, в которой постепенно будет накапливаться нужная сумма. Начальное значение суммы задается перед основным циклом. До начала вычислений сумма равна нулю. На каждом шаге цикла к уже накопленной сумме прибавляется значение очередного элемента совокупности. Тогда после завершения основного цикла в этой переменной будет храниться сумма всех элементов совокупности.

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- a – для хранения очередного элемента совокупности;

Оглавление

- i – номер обрабатываемого элемента;
- n – количество элементов совокупности.

```
Dim a, i, n As Integer
```

В переменной summa будет накапливаться результат.

```
Dim summa As Integer
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование Val.

```
n = Val(InputBox("Введите количество элементов"))
```

Задаем начальное значение переменной summa.

```
summa = 0
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит n элементов, то номера этих элементов будут меняться в диапазоне от 1 до n. Поэтому начальное значение счетчика i (номера элемента) равно 1, а конечное – n.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование Val.

```
    a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (i) и его значение (a). Константа vbTab используется для организации вывода в две колонки.

```
    lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Добавляем значение элемента совокупности к переменной summa, накапливая в ней таким образом сумму всех элементов.

```
    summa += a
Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Затем печатаем накопленное значение суммы.

```
lstA.Items.Add("Сумма чисел =" + Str(summa))
```

Оглавление

Вычисление произведения чисел.

Произведение элементов также вычисляется накоплением. Но в отличие от суммы начальное значение произведения равно 1. Обратите внимание, если начальное значение произведения задать равным нулю, то получение результата станет невозможным. Начальное значение, так же как и в предыдущем случае, задается перед основным циклом. В цикле значение произведения умножается на очередной элемент совокупности. Тогда по завершении основного цикла мы получим искомое произведение всех элементов совокупности.

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- `a` – для хранения очередного элемента совокупности;
- `i` – номер обрабатываемого элемента;
- `n` – количество элементов совокупности.

```
Dim a, i, n As Integer
```

В переменной `proiz` будет накапливаться результат.

```
Dim proiz As Integer
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование `Val`.

```
n = Val(InputBox("Введите количество элементов"))
```

Задаем начальное значение произведения.

```
proiz = 1
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит `n` элементов, то номера этих элементов будут меняться в диапазоне от 1 до `n`. Поэтому начальное значение счетчика `i` (номера элемента) равно 1, а конечное – `n`.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование `Val`.

```
a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (`i`) и его значение (`a`). Константа `vbTab` используется для организации вывода в две колонки.

Оглавление

```
            lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Умножаем ранее накопленное произведение на очередной элемент совокупности, формируя таким образом искомый результат.

```
            proiz *= a
        Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
        lstA.Items.Add("-----------------------------------")
```

Выводим вычисленное произведение.

```
        lstA.Items.Add("Произведение чисел =" + Str(proiz))
```

Определение количества четных чисел в совокупности.

Определение количества чисел во многом похоже на вычисление суммы, с той лишь разницей, что к результирующей переменной прибавляется единица, а не очередной элемент совокупности. Перед началом основного цикла количество нужных элементов полагается равным нулю. После ввода каждого элемента совокупности проводится анализ, соответствует ли данный элемент поставленному условию (например, является ли оно четным). Если значение элемента удовлетворяет условию, то искомое количество увеличивается на единицу.

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- `a` – для хранения очередного элемента совокупности;
- `i` – номер обрабатываемого элемента;
- `n` – количество элементов совокупности.

```
        Dim a, i, n As Integer
```

Для подсчета количества четных элементов заведем специальную переменную. Очевидно, что она будет иметь целый тип.

```
        Dim kol As Integer
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
        lstA.Items.Clear()
```

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование `Val`.

Оглавление

```
n = Val(InputBox("Введите количество элементов"))
```

До начала основного цикла количество элементов полагаем равным нулю, так как в нашей совокупности пока нет ни одного четного числа.

```
kol = 0
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит `n` элементов, то номера этих элементов будут меняться в диапазоне от 1 до `n`. Поэтому начальное значение счетчика `i` (номера элемента) равно 1, а конечное – `n`.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование `Val`.

```
    a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (`i`) и его значение (`a`). Константа `vbTab` используется для организации вывода в две колонки.

```
    lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Анализируем значение введенного элемента совокупности, проверяя, является ли оно четным. Если значение переменной `a` четное, то остаток от его деления на два будет равен нулю. Для нечетных чисел остаток от деления на два отличен от нуля. Он равен 1 для положительных чисел и -1 для отрицательных чисел.

```
    If a Mod 2 = 0 Then
```

Если текущий элемент совокупности четный, то количество четных чисел необходимо увеличить на единицу. Новое значение будет записано в ту же переменную `kol` вместо старого значения.

```
        kol += 1
    End If
Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Перед выводом найденного значения необходимо провести его анализ.

```
If kol = 0 Then
```

Если количество элементов равно нулю, значит, в совокупности нет четных чисел. В этом случае выводим поясняющее сообщение.

Оглавление

```
        lstA.Items.Add("В совокупности нет четных чисел")
    Else
```

В противном случае выводим найденное значение.

```
        lstA.Items.Add("Количество четных чисел =" + _
                       Str(kol))
    End If
```

Вычисление среднего арифметического положительных чисел в совокупности.

Среднее арифметическое элементов совокупности, удовлетворяющих некоторому условию (например, положительных) определяется как отношение суммы необходимых элементов к их количеству. Поэтому задача вычисления среднего арифметического решается как комбинация задач вычисления суммы и количества нужных элементов совокупности.

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- `a` – для хранения очередного элемента совокупности;
- `i` – номер обрабатываемого элемента;
- `n` – количество элементов совокупности.

```
    Dim a, i, n As Integer
```

Для вычисления среднего арифметического необходимо найти сумму положительных элементов и их количество. Эти значения будем хранить соответственно в переменных `summa` и `kol`.

```
    Dim summa, kol As Integer
```

Еще нам потребуется переменная для хранения результатов вычислений. Так как среднее арифметическое определяется в результате деления, то переменная `sred` обязательно должна иметь рациональный тип данных.

```
    Dim sred As Single
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
    lstA.Items.Clear()
```

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование `Val`.

```
    n = Val(InputBox("Введите количество элементов"))
```

Оглавление

Перед основным циклом задаем начальные значения для суммы и количества.

```
summa = 0
kol = 0
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит n элементов, то номера этих элементов будут меняться в диапазоне от 1 до n. Поэтому начальное значение счетчика i (номера элемента) равно 1, а конечное – n.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование Val.

```
    a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (i) и его значение (a). Константа vbTab используется для организации вывода в две колонки.

```
    lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Анализируем значение очередного элемента совокупности.

```
    If a > 0 Then
```

Если элемент – положительное число, то его значение необходимо прибавить к сумме, а количество положительных элементов увеличить на единицу.

```
        summa += a
        kol += 1
    End If
Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Так как среднее арифметическое получается в результате деления, то перед его непосредственным вычислением необходимо проверить, обращается ли знаменатель дроби в ноль.

```
If kol = 0 Then
```

Если в совокупности не было положительных чисел, то их количество будет равно нулю. В этом случае невозможно вычислить среднее арифметическое положительных элементов. Поэтому вместо ответа мы выводим поясняющее сообщение.

```
    lstA.Items.Add("Нет положительных чисел")
Else
```

Оглавление

В противном случае (если количество неравно нулю) вычисляем среднее арифметическое.

```
sred = summa / kol
```

И выводим найденное значение.

```
lstA.Items.Add("Сред. арифм. полож. чисел =" + _
                Str(sred))
End If
```

Вычисление среднего геометрического отрицательных чисел в совокупности.

Среднее геометрическое элементов совокупности, удовлетворяющих некоторому условию (например, отрицательных) определяется как корень степени k из произведения необходимых элементов, где k – количество таких элементов. Поэтому задача вычисления среднего геометрического решается как комбинация задач вычисления произведения и количества нужных элементов совокупности. Извлечение корня k-ой степени реализуется как возведение числа в степень $(1 / k)$.

$$x = \sqrt[k]{p} = p^{\left(\frac{1}{k}\right)}$$

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- `a` – для хранения очередного элемента совокупности;
- `i` – номер обрабатываемого элемента;
- `n` – количество элементов совокупности.

```
Dim a, i, n As Integer
```

Для вычисления среднего геометрического необходимо найти произведение и количество отрицательных чисел. Заведем отдельные переменные для хранения этих значений.

```
Dim proiz, kol As Integer
```

Среднее геометрическое определяется в результате извлечения корня некоторой степени. Поэтому итоговая переменная обязательно будет иметь рациональный тип данных.

```
Dim geom As Single
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Оглавление

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование Val.

```
n = Val(InputBox("Введите количество элементов"))
```

Перед основным циклом задаем начальные значения для произведения и количества. Обратите внимание, что начальное значение произведения равно единице.

```
proiz = 1
kol = 0
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит n элементов, то номера этих элементов будут меняться в диапазоне от 1 до n. Поэтому начальное значение счетчика i (номера элемента) равно 1, а конечное – n.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование Val.

```
a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (i) и его значение (a). Константа vbTab используется для организации вывода в две колонки.

```
lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Анализируем значение очередного элемента совокупности.

```
If a < 0 Then
```

Если элемент является отрицательным числом, то его необходимо включить в произведение. При этом количество отрицательных элементов увеличится на единицу.

```
    proiz *= a
    kol += 1
End If
Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Перед вычислением среднего геометрического необходимо провести анализ промежуточных результатов.

```
if kol = 0 Then
```

Оглавление

Если количество отрицательных чисел равно нулю, то задача вычисления среднего геометрического не имеет смысла. В этом случае надо вывести поясняющее сообщение.

```
            lstA.Items.Add("Нет отрицательных чисел")
        Else
```

В противном случае мы должны проверить знак полученного произведения.

```
            If proiz > 0 Then
```

Если произведение чисел получилось положительным, то можно извлекать из него корень любой ненулевой степени.

```
                geom = proiz ^ (1 / kol)
            Else
```

Иначе (когда подкоренное выражение отрицательное) извлечение корня возможно только при нечетной степени корня. Очевидно, что для данной задачи отрицательное произведение будет получаться только при нечетном количестве сомножителей. Поэтому дополнительную проверку степени корня можно не выполнять. Обратите внимание, что в Visual Basic 2005 операция извлечения корня произвольной степени определена только для положительных подкоренных выражений. Поэтому, когда необходимо извлечь корень нечетной степени из отрицательного числа, поступают следующим образом. Корень извлекается из модуля подкоренного выражения, а у полученного результата знак меняется на противоположный.

```
                geom = -Math.Abs(proiz) ^ (1 / kol)
            End If
```

После извлечения корня выводим полученное значение среднего геометрического отрицательных элементов совокупности.

```
            lstA.Items.Add("Сред. геом. отриц. чисел =" + _
                           Str(geom))
        End If
```

Поиск максимального числа в совокупности.

Для того чтобы найти максимальное число в совокупности нам потребуется дополнительная переменная, в которой будет храниться значение максимального из уже введенных элементов. После ввода каждого нового элемента совокупности мы будем проверять, не превышает ли его значение ранее найденного максимума. Если да, то значение максимума необходимо задать равным введенному элементу. Таким образом, после окончания основного цикла значение максимума будет совпадать со значением наибольшего элемента совокупности.

Оглавление

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- a – для хранения очередного элемента совокупности;
- i – номер обрабатываемого элемента;
- n – количество элементов совокупности.

```
Dim a, i, n As Integer
```

Нам потребуется переменная, в которой будет храниться значение максимального элемента совокупности. Обратите внимание, что тип этой переменной всегда будет совпадать с типом элемента совокупности.

```
Dim max As Integer
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование Val.

```
n = Val(InputBox("Введите количество элементов"))
```

Перед основным циклом задаем начальное значение максимума. Так как в процессе анализа элементов совокупности значение максимума будет увеличиваться, то начальное значение этой переменной должно быть достаточно маленьким.

```
max = -100000
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит n элементов, то номера этих элементов будут меняться в диапазоне от 1 до n. Поэтому начальное значение счетчика i (номера элемента) равно 1, а конечное – n.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование Val.

```
a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (i) и его значение (a). Константа vbTab используется для организации вывода в две колонки.

```
lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Анализируем значение элемента совокупности.

```
If a > max Then
```

Оглавление

Если оно превышает ранее найденный максимум, то значение текущего элемента совокупности запоминается как новое значение максимума.

```
            max = a
        End If
    Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Теперь выводим найденное значение максимума.

```
lstA.Items.Add("Максимальное число =" + Str(max))
```

Поиск минимального числа, кратного трем.

Нахождение минимального элемента совокупности практически аналогичного поиску максимума. Изменить надо лишь используемую операцию сравнения и начальное значение. Так как в процессе анализа элементов совокупности значение минимума будет уменьшаться, то его начальное значение должно быть достаточно большим.

При поиске минимального (или максимального) числа, удовлетворяющего некоторому условию (например, кратного трем), в алгоритме возникает дополнительное условие. Теперь в сравнении будут участвовать только те элементы, которые удовлетворяет поставленному условию.

Рассмотрим особенности программной реализации этого алгоритма. Для обработки совокупности нам потребуются три целочисленных переменных:

- `a` – для хранения очередного элемента совокупности;
- `i` – номер обрабатываемого элемента;
- `n` – количество элементов совокупности.

```
Dim a, i, n As Integer
```

Нам потребуется переменная, в которой будет храниться значение минимального элемента совокупности. Обратите внимание, что тип этой переменной всегда будет совпадать с типом элемента совокупности.

```
Dim min As Integer
```

Работа программы начинается с очистки окна списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Оглавление

Просим пользователя ввести количество элементов совокупности. Так как эта информация является числовой, то при ее вводе необходимо использовать преобразование Val.

```
n = Val(InputBox("Введите количество элементов"))
```

Перед основным циклом задаем начальное значение минимума. Так как в процессе анализа элементов совокупности значение минимума будет уменьшаться, то начальное значение этой переменной должно быть достаточно большим.

```
min = 100000
```

Организуем основной цикл для обработки совокупности чисел. Так как совокупность содержит n элементов, то номера этих элементов будут меняться в диапазоне от 1 до n. Поэтому начальное значение счетчика i (номера элемента) равно 1, а конечное – n.

```
For i = 1 To n
```

Вводим очередной элемент совокупности. Так как элемент совокупности является числом, то для его ввода необходимо использовать преобразование Val.

```
a = Val(InputBox("Введите элемент совокупности"))
```

В окно списка выводим номер элемента совокупности (i) и его значение (a). Константа vbTab используется для организации вывода в две колонки.

```
lstA.Items.Add(Str(i) + vbTab + Str(a))
```

Анализируем значение очередного элемента совокупности.

```
If a Mod 3 = 0 And a < min Then
```

Если оно кратно трем (то есть остаток от его деления на 3 равен нулю) и само значение меньше ранее найденного минимума, то значение текущего элемента совокупности запоминается как новое значение минимума.

```
        min = a
    End If
Next
```

После завершения основного цикла выводим полученные результаты. Сначала печатаем горизонтальную черту из нескольких знаков минус. Она зрительно отделит исходные данные от результатов вычислений.

```
lstA.Items.Add("-----------------------------------")
```

Так как в совокупности может не быть чисел, кратных трем, то перед выводом результата, необходимо провести его проверку.

```
If min = 100000 Then
```

Оглавление

Если за время выполнения основного цикла значение минимума не изменилось и совпадает с начальным значением, значит, что в совокупности не было элементов, кратных трем. В таком случае необходимо вместо значения минимума вывести поясняющее сообщение.

```
            lstA.Items.Add("Нет чисел, кратных 3")
        Else
```

Иначе мы можем вывести найденное значение.

```
            lstA.Items.Add("Минимальное кратное трем = " + _
                           Str(min))
        End If
```

Два примера работы программы в режиме вычисления среднего арифметического положительных чисел приведены на рис. 5.

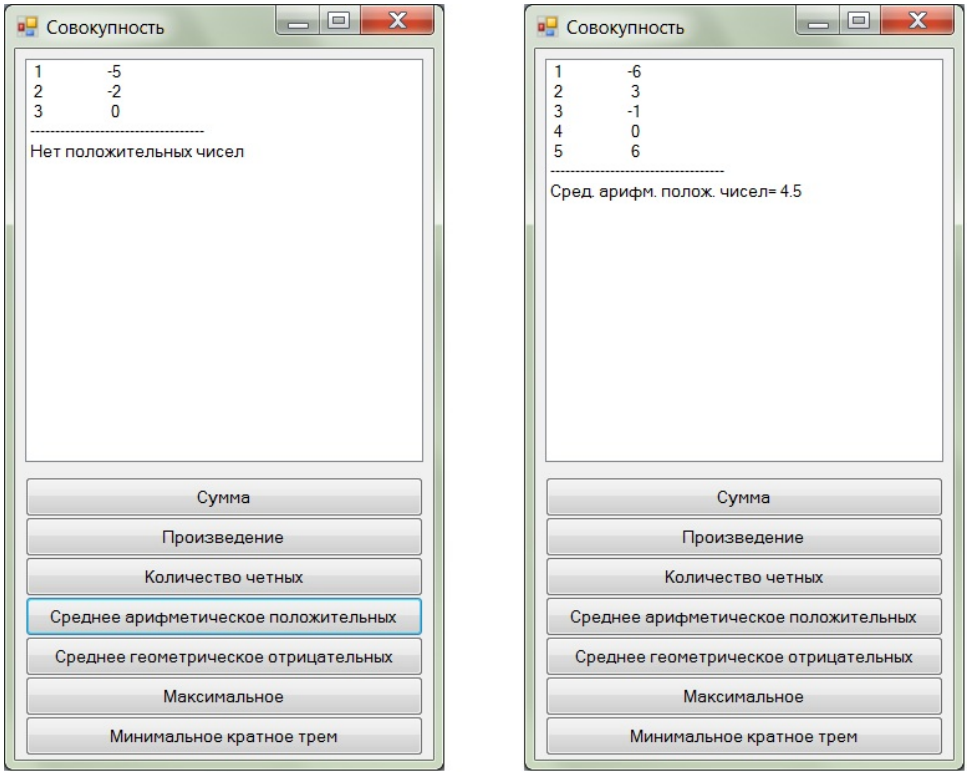

Рис. 5 Пример работы программы обработки совокупности с известным числом элементов

2. Цикл с условием

Цикл с условием используется в тех случаях, когда число повторений цикла заранее неизвестно. Например, при обработке совокупности чисел, ввод которой прекращается при появлении первого нуля.

Оглавление

Цикл с условием – это многострочный оператор Visual Basic 2005, первая строка которого начинается со слова Do, а последняя строка начинается со слова Loop.

Выделяют две разновидности цикла с условием: цикл с предусловием и цикл с постусловием. В цикле с предусловием условие цикла располагается перед телом цикла.

```
Do Условие Цикла
   Тело цикла
Loop
```

При этом возможна ситуация, когда операторы тела цикла не выполнятся ни разу. Другими словами условие цикла можно сформулировать таким образом, что управление никогда не попадет внутрь цикла.

В цикле с постусловием условие расположено после тела цикла. Поэтому операторы тела цикла обязательно выполнятся хотя бы один раз, чтобы выполнение программы могло дойти до проверки условия.

```
Do
   Тело цикла
Loop Условие цикла
```

Вне зависимости от вида цикла его условие может быть записано в одной из двух форм.

- Условие продолжения цикла (условие While).

  ```
  While Условное выражение
  ```

  В этом случае операторы тела цикла выполняются, пока Условное выражение имеет значение Истина (True). Цикл завершается, когда Условное выражение принимает значение Ложь (False).

- Условие завершения цикла (условие Until).

  ```
  Until Условное Выражение
  ```

  Цикл с таким условием завершается, когда Условное выражение принимает значение Истина (True). Если Условное выражение ложно, то выполнение цикла продолжается.

Таким образом, в Visual Basic 2005 возможны четыре различных варианта цикла с условием.

- ```
  Do While Условное выражение
      Тело цикла
  Loop
  ```

Оглавление
  ```

- Do Until Условное Выражение
  Тело цикла
  Loop
- Do
  Тело цикла
  Loop While Условное выражение
- Do
  Тело цикла
  Loop Until Условное Выражение

Меняя Условные выражения, каждый вид цикла можно заменить на любой другой без потери работоспособности программы. Если Условие цикла сформулировано с ошибкой, то программа может попасть в бесконечный цикл. Бесконечный цикл – это цикл, в котором количество повторов ничем не ограничено. В таких случаях говорят, что программа «зациклилась» или «зависла». Чаще всего бесконечный цикл возникает из-за ошибок в условии цикла: условие продолжения всегда имеет значение Истина (True) или условие завершения цикла всегда имеет значение Ложь (False). Прервать программу, попавшую в бесконечный цикл, можно только из среды Visual Basic 2005. Для этого необходимо перейти из окна работающей программы в окно среды Visual Basic 2005 и выбрать команду **Stop Debugging** из пункта меню **Debug**.

Среди операторов тела цикла может встречаться особый оператор Exit Do. Он прекращает выполнение цикла при любом значении его условия. Выполнение программы продолжается с оператора, стоящего сразу после ключевого слова Loop.

2.1. Ввод с проверкой

В качестве примера использования цикла с условием рассмотрим задачу ввода значения переменной с проверкой. Будем вводить с клавиатуры значение переменной n, которое должно находиться в диапазон [3; 20]. Эту задачу можно решить двумя способами: с помощью оператора безусловного перехода GoTo или при помощи цикла с условием. Первый способ выглядит следующим образом.

```
Dim n As Integer
vvod:
n = Val(InputBox("Введите число n от 3 до 20"))
If n < 3 Or n > 20 Then
    MsgBox("Неправильное значение")
```

Оглавление

```
        GoTo Vvod
    End If
```

Второй способ решения предполагает использование цикла с условием. Решим эту же задачу, используя цикл с постусловием. Сначала воспользуемся циклом с условием `Until`. В этом случае условие цикла надо сформулировать таким образом, чтобы он прекращал свою работу при правильном значении переменной `n`. То есть условие цикла – это условие правильности значения переменной.

```
Dim n As Integer
Do
    n = Val(InputBox("Введите число n от 3 до 20"))
Loop Until n >= 3 And n <= 20
```

Теперь рассмотрим использование цикла с условием `While`. Это условие продолжения цикла, поэтому оно должно быть сформулировано таким образом, чтобы при неправильном значении `n` цикл выполнялся еще раз.

```
Dim n As Integer
Do
    n = Val(InputBox("Введите число n от 3 до 20"))
Loop While n < 3 Or n > 20
```

Эту же задачу можно решать, используя циклы с предусловием. Но решение получается громоздким и на практике не применяется.

2.2. Обработка совокупности чисел с неизвестным числом элементов

Другая область использования циклов с условием – это задачи обработки совокупности чисел, в которых количество элементов заранее неизвестно. Существует два вида таких совокупностей. В первом случае ввод чисел прекращается с появлением элемента, имеющего некоторое заранее заданное значение, например, ноль. Во втором случае ограничение на значение элементов отсутствует, но после ввода каждого элемента пользователь отвечает на вопрос, хочет ли он дальше вводить элементы совокупности. Иногда этот вопрос может задаваться не после ввода очередного элемента, а перед ним. Это определяет вид используемого цикла. Если вопрос задается перед вводом элемента, применяется цикл с предусловием. Если вопрос задается после ввода элемента, применяется цикл с постусловием. Алгоритмы обработки совокупности, описанные в разделе 1.3, не зависят от ее вида и способа ввода. Поэтому рассмотрим два частных случая обработки совокупностей с неизвестным числом элементов.

Оглавление

Найти произведение всех элементов совокупности. Ввод чисел прекращается при появлении элемента, равного нулю.

Данную задачу целесообразно решать, используя цикл с предусловием. Так как совокупность заканчивается нулевым элементом, то, очевидно, что последний введенный элемент (ноль) в произведение включать не следует. Поэтому проверка введенного значения должна предшествовать его включению в общее произведение. Элементы совокупности будем вводить, используя функцию `InputBox`. Элементы совокупности и результаты вычислений будем выводить в окно списка с именем `lstA`.

Для решения задачи нам потребуются следующие переменные: `a` – элемент совокупности, `proiz` – искомое произведение, `kol` – количество элементов в совокупности.

```
Dim a, proiz, kol As Integer
```

Очищаем окно списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Задаем начальные значения для произведения и количества.

```
proiz = 1
kol = 0
```

Вводим первый элемент совокупности.

```
a = Val(InputBox("Введите элемент совокупности"))
```

Проверяем, равен ли введенный элемент нулю. Если да, то выполнение цикла прекращается.

```
Do Until a = 0
```

Если элемент совокупности не равен нулю, то выводим его значение в окно списка.

```
    lstA.Items.Add(Str(a))
```

Увеличиваем количество элементов совокупности на единицу.

```
    kol += 1
```

И включаем элемент совокупности в общее произведение.

```
    proiz *= a
```

Просим пользователя ввести очередной элемент совокупности.

```
    a = Val(InputBox("Введите элемент совокупности"))
```

Этот элемент будет обработан при следующем повторе тела цикла.

```
Loop
```

Оглавление

После завершения основного цикла, выводим в окно списка горизонтальную черту, которая позволит зрительно отделить элементы совокупности от результатов вычислений.

```
lstA.Items.Add("----------------------------")
```

Проверяем, есть ли в совокупности ненулевые элементы.

```
If kol = 0 Then
```

Если количество элементов в совокупности равно нулю, значит, первый же введенный элемент был равен нулю. В этом случае мы не можем вычислить произведение элементов. Поэтому вместо значения произведения выводим поясняющее сообщение.

```
    lstA.Items.Add("Нет ненулевых элементов")
Else
```

В противном случае, выводим значение вычисленного произведения.

```
    lstA.Items.Add("Произведение = " + Str(proiz))
End If
```

Примеры работы программы представлены на рис. 6.

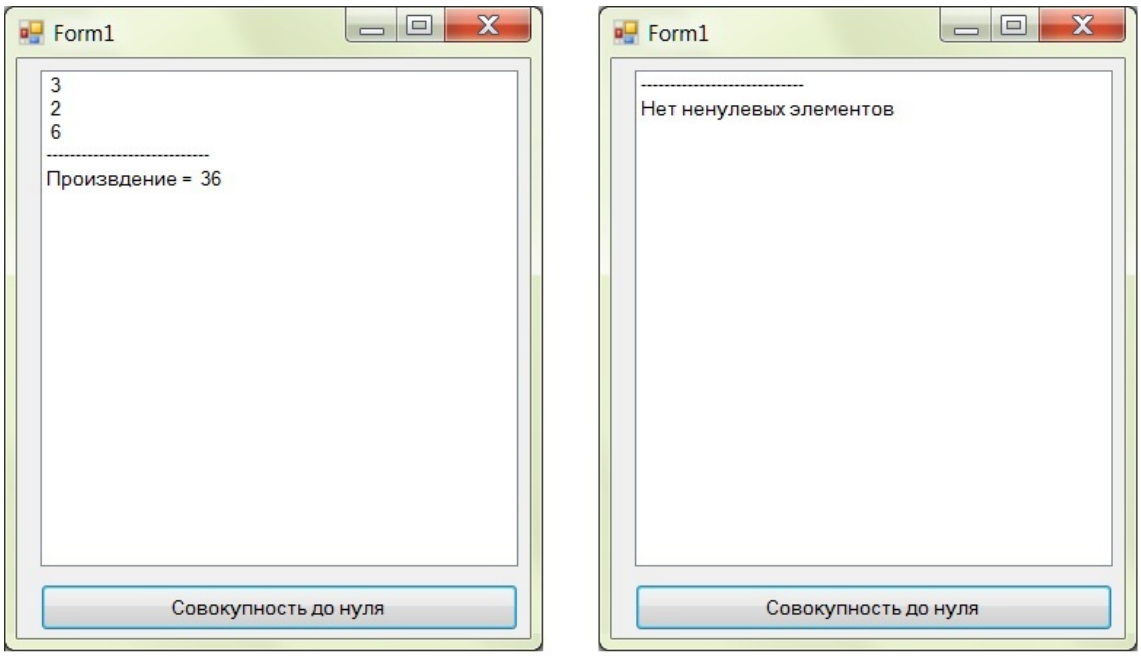

Рис. 6. Примеры работы программы обработки совокупности, ограниченной нулем

Найти сумму все элементов последовательности. После ввода каждого числа пользователю задается вопрос, хочет ли он продолжить ввод чисел.

Для решения этой задачи будем использовать цикл с постусловием, так как вопрос пользователю задается после ввода очередного элемента. Сам алгоритм

Оглавление

вычисления суммы элементов совокупности ничем не отличатся от аналогичного алгоритма, разработанного для совокупности с известным числом элементов. Элементы совокупности будем вводить, используя функцию `InputBox`. Элементы совокупности и результаты вычислений будем выводить в окно списка с именем `lstA`.

Для решения задачи нам потребуются три целочисленных переменных: `a` – элемент совокупности, `summa` – искомая сумма элементов, `otvet` – для хранения и обработки ответа пользователя на запрос о повторном вводе.

```
Dim a, summa, otvet As Integer
```

Очищаем окно списка от результатов предыдущих запусков программы.

```
lstA.Items.Clear()
```

Задаем начальное значение суммы.

```
summa = 0
```

Организуем основной цикл.

```
Do
```

Вводим очередной элемент совокупности.

```
    a = Val(InputBox("Введите элемент совокупности"))
```

Выводим его значение в окно списка.

```
    lstA.Items.Add(Str(a))
```

Добавляем значение элемента совокупности к ранее накопленной сумме.

```
    summa += a
```

Спрашиваем у пользователя, хочет ли он еще вводить элементы совокупности.

```
    otvet = MsgBox("Еще вводить числа?", 32 + 4)
```

Анализируем ответ пользователя. Если пользователь ответил «Нет», то в переменную `otvet` будет записано число 7 (см. раздел 4.10). В этом случае основной цикл завершает свою работу.

```
Loop Until otvet = 7
```

После завершения основного цикла, выводим в окно списка горизонтальную черту, которая позволит зрительно отделить элементы совокупности от результатов вычислений.

```
lstA.Items.Add("----------------------------")
```

В окно списка выводим вычисленную сумму всех элементов совокупности.

```
lstA.Items.Add("Сумма = " + Str(summa))
```

Пример работы программы приведен на рис. 7.

Оглавление

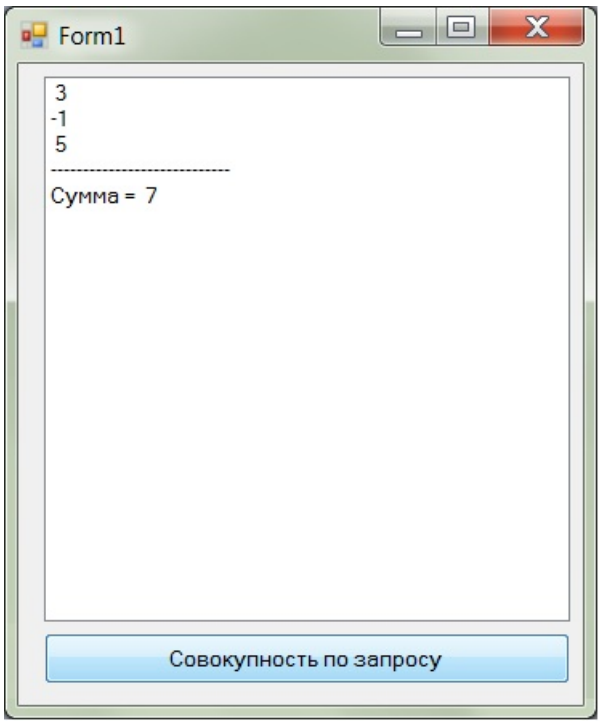

Рис. 7. Пример обработки совокупности с неизвестным числом элементов

2.3. Вычисление суммы ряда по общей формуле

Одной из основных областей применения циклов с условием являются приближенные математические вычисления, например, вычисление суммы ряда и решение нелинейных уравнений. Изучим особенности программной реализации некоторых из этих задач и начнем с задачи вычисления суммы бесконечного сходящегося ряда.

Рассмотрим ряд

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!} + \ldots ,$$

где n = 1, 2, 3,… Требуется найти сумму ряда в некоторой заранее заданной точке x с определенной точностью ε.

Исходными данными для этой задачи являются значение x и необходимая точность вычислений ε. Параметр «точность вычислений» означает, что слагаемые, которые по модулю меньше точности, считаются несущественными и в общую сумму не включаются. Для ввода исходных данных будем использовать функцию `InputBox`. Результатом программы является значение накопленной суммы. Но для проверки правильности вычислений мы будем еще выводить значение левой части выражения, номер и значение последнего слагаемого, вошедшего в сумму. Для вывода результатов будем использовать окно списка с именем `lstA`.

Оглавление

Для решения этой задачи нам потребуется организовать цикл с условием. На каждом шаге цикла мы будем по общей формуле вычислять значение очередного слагаемого и прибавлять его к общей сумме. Как только очередное слагаемое станет меньше заданной точности, мы закончим выполнение цикла.

Рассмотрим особенности программной реализации этого алгоритма. Для решения задачи нам потребуются следующие переменные: `x` – точка, в которой вычисляется сумма ряда, `eps` – требуемая точность вычислений, `summa` – искомая сумма ряда, `slag` – очередное слагаемое. Все эти переменные имеют рациональный тип данных. Для повышения точности наших вычислений будем использовать тип `Double`.

```
Dim x, summa, slag, eps As Double
```

Для вычисления факториала, стоящего в знаменателе дроби, заведем переменную `f`, которая будет иметь целый тип с максимально возможной емкостью – `ULong`.

```
Dim f As ULong
```

Так как для вычисления факториала необходимо организовать цикл, то нам потребуется специальная переменная – счетчик `i`. Заметим, что формула общего члена ряда зависит от номера слагаемого – *n*. Поэтому при решении задачи нам потребуется переменная для хранения номера очередного слагаемого. Назовем ее `n`. Она будем иметь целый тип данных.

```
Dim n, i As Integer
```

Работа программы начинается с очистки окна списка от ее предыдущих результатов.

```
lstA.Items.Clear()
```

Вводим исходные данные.

```
x = Val(InputBox("Введите точку"))
eps = Val(InputBox("Введите точность"))
```

Задаем начальные значения. Начальное значение суммы равно нулю. Номер текущего слагаемого тоже равен нулю, так как никакого слагаемого на данный момент времени мы не вычислили.

```
summa = 0
n = 0
```

Организуем основной цикл.

```
Do
```

На каждом шаге цикла будем вычислять очередного слагаемое. При этом его номер будет на единицу больше, чем на предыдущем шаге.

Оглавление

```
n += 1
```

Начинаем вычислять значение очередного слагаемого. Первым шагом будет вычисление факториала, стоящего в знаменателе дроби.

```
f = 1
For i = 2 To 2 * n - 1
    f *= i
Next
```

Затем по общей формуле вычисляем само слагаемое, заменяя факториал на уже найденное значение.

```
slag = (-1) ^ (n + 1) * x ^ (2 * n - 1) / f
```

Полученное слагаемое добавляем к общей сумме.

```
summa += slag
```

Проверяем, если модуль слагаемого меньше заданной точности, то дальнейшие вычисления не приведут к заметным изменениям результата, и выполнение цикла можно завершить.

```
Loop Until Math.Abs(slag) <= eps
```

Выводим в окно списка полученную сумму, значение выражения, стоящего в левой части равенства, номер и значение последнего слагаемого.

```
lstA.Items.Add("summa=" + Str(summa))
lstA.Items.Add("sin(x)=" + Str(Math.Sin(x)))
lstA.Items.Add("n=" + Str(n))
lstA.Items.Add("Последнее слагаемое =" + Str(slag))
```

Пример работы программы приведен на рис. 8. Исходные данные для этого случая: `x` = 1, `eps` = 1e-4 = 10^{-4}.

Оглавление

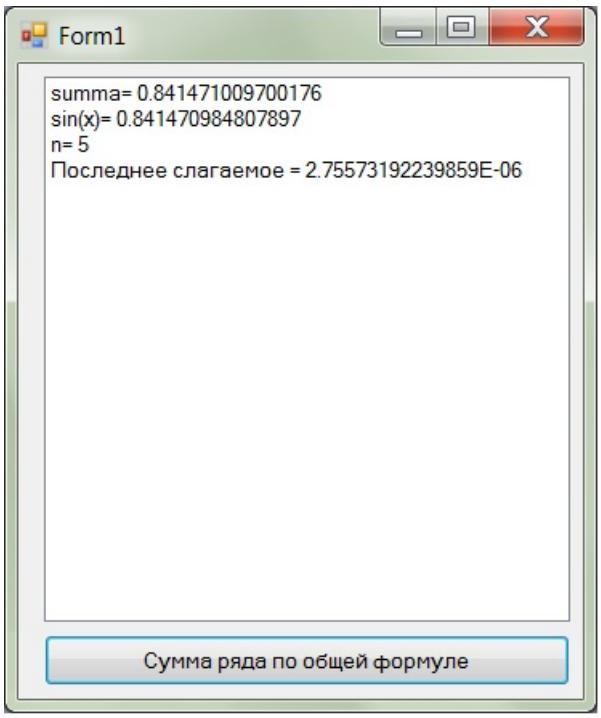

Рис. 8. Пример работы программы вычисления суммы ряда по общей формуле

2.4. Вычисление суммы ряда с использованием рекуррентного соотношения

Вычисление суммы ряда по общей формуле, рассмотренное в предыдущем разделе, безусловно, является общим способом решения подобных задач. Но иногда процесс вычислений можно существенно упростить, построив рекуррентное соотношение для вычисления очередного слагаемого. В этом случае каждое слагаемое вычисляется как некоторая функция, зависящая от одного или нескольких предыдущих членов ряда. Построим рекуррентное соотношение для следующего ряда.

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^{n+1}\frac{x^{2n-1}}{(2n-1)!} + \ldots \text{ , где } n = 1, 2, 3, \ldots$$

Сначала запишем формулы для слагаемых с номерами n и n - 1.

$$s_n = (-1)^{n+1}\frac{x^{2n-1}}{(2n-1)!}$$

$$s_{n-1} = (-1)^{(n-1)+1}\frac{x^{2(n-1)-1}}{(2(n-1)-1)!} = (-1)^{n-1+1}\frac{x^{2n-2-1}}{(2n-2-1)!} = (-1)^n\frac{x^{2n-3}}{(2n-3)!}$$

Вычислим отношение s_n / s_{n-1}.

Оглавление

$$\frac{s_n}{s_{n-1}} = \frac{(-1)^{n+1}\dfrac{x^{2n-1}}{(2n-1)!}}{(-1)^{n}\dfrac{x^{2n-3}}{(2n-3)!}} = \frac{(-1)^{n+1}x^{2n-1}(2n-3)!}{(-1)^{n}x^{2n-3}(2n-1)!} = -\frac{x^2(2n-3)!}{(2n-1)!} =$$

$$= -x^2\frac{1\cdot 2\cdot 3\times\cdots\times(2n-3)}{1\cdot 2\cdot 3\times\cdots\times(2n-3)(2n-2)(2n-1)} = -\frac{x^2}{(2n-2)(2n-1)}$$

Таким образом, мы получили, что

$$\frac{s_n}{s_{n-1}} = -\frac{x^2}{(2n-2)(2n-1)}$$

Отсюда можно легко получить рекуррентное соотношение для вычисления очередного слагаемого.

$$s_n = -\frac{x^2\, s_{n-1}}{(2n-2)(2n-1)}$$

В этой формуле s_n – очередное слагаемое, s_{n-1} – предыдущее слагаемое, x – точка, в которой вычисляется сумма ряда, n – номер очередного слагаемого.

Используя полученную рекуррентную формулу, найдем сумму ряда в некоторой заранее заданной точке x с определенной точностью ε. Так же как и предыдущем случае, исходными данными будут значение x и необходимая точность вычислений ε. Для ввода исходных данных будем использовать функцию `InputBox`. Результатом программы является значение накопленной суммы. Но для проверки правильности вычислений мы будем еще выводить значение левой части выражения, номер и значение последнего слагаемого, вошедшего в сумму. Для вывода результатов будем использовать окно списка с именем `lstA`.

Для решения этой задачи нам потребуется организовать цикл с условием. На каждом шаге цикла мы будем по рекуррентной формуле вычислять значение очередного слагаемого и прибавлять его к общей сумме. Как только очередное слагаемое станет меньше заданной точности, мы закончим выполнение цикла.

Рассмотрим особенности программной реализации этого алгоритма. Для решения задачи нам потребуются следующие переменные: `x` – точка, в которой вычисляется сумма ряда, `eps` – требуемая точность вычислений, `summa` – искомая сумма ряда, `slag` – очередное слагаемое. Все эти переменные имеют рациональный тип данных. Для повышения точности наших вычислений будем использовать тип `Double`.

```
Dim x, summa, slag, eps As Double
```

Оглавление

Так как рекуррентная формула зависит от номера очередного слагаемого, то для его хранения заведем переменную n.

```
Dim n As Integer
```

Очищаем окно списка.

```
lstA.Items.Clear()
```

Вводим исходные данные.

```
x = Val(InputBox("Введите точку"))
eps = Val(InputBox("Введите точность"))
```

Задаем начальные значения переменных. Номер первого слагаемого равен единице.

```
n = 1
```

По формуле ряда определяем первое слагаемое.

```
slag = x
```

Итоговую сумму полагаем равной первому слагаемому.

```
summa = slag
```

Организуем основной цикл.

```
Do
```

На каждом шаге мы вычисляем очередное слагаемое. При этом его номер будет на единицу больше, чем на предыдущем шаге.

```
    n += 1
```

Используя рекуррентное соотношение, вычисляем очередное слагаемое.

```
    slag = -slag * x ^ 2 / ((2 * n - 2) * (2 * n - 1))
```

И добавляем его к итоговой сумме.

```
    summa += slag
```

Проверяем, если модуль слагаемого меньше заданной точности, то дальнейшие вычисления не приведут к заметным изменениям результата, и выполнение цикла можно завершить.

```
Loop Until Math.Abs(slag) <= eps
```

Выводим в окно списка полученную сумму, значение выражения, стоящего в левой части равенства, номер и значение последнего слагаемого.

```
lstA.Items.Add("summa=" + Str(summa))
lstA.Items.Add("sin(x)=" + Str(Math.Sin(x)))
lstA.Items.Add("n=" + Str(n))
lstA.Items.Add("Последнее слагаемое =" + Str(slag))
```

Оглавление

Пример работы программы приведен на рис. 9. Исходные данные для этого случая: `x` = 1, `eps` = 1e-4 = 10^{-4}.

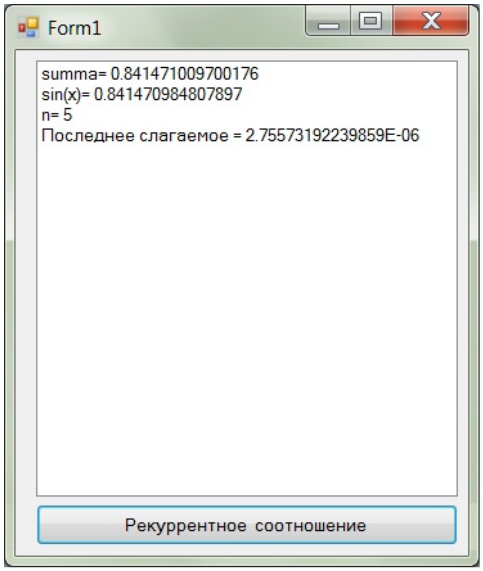

Рис. 9. Пример работы программы вычисления суммы ряда с использованием рекуррентного соотношения

2.5. Вычисление произведения ряда

Вычисление произведения ряда во многом похоже на вычисление ряда по общей формуле. Только вместо накопления суммы мы будем использовать прием накопления произведения. Соответственно изменятся начальное значение итоговой переменной (оно станет равным единице) и условие завершения цикла. Основной цикл должен завершиться тогда, когда очередной сомножитель будет отличаться от единицы не более чем на заданную величину, которая и является точностью вычислений ε.

Рассмотрим задачу вычисления произведения ряда

$$\cos x = \left(1-\frac{4x^2}{\pi^2}\right)\left(1-\frac{4x^2}{9\pi^2}\right)\cdots\left(1-\frac{4x^2}{(2n-1)^2\pi^2}\right)\cdots, \quad n = 1, 2, 3...$$

в некоторой заранее заданной точке *x* с определенной точностью ε. Для ввода исходных данных будем использовать функцию `InputBox`. В качестве результатов будем выводить итоговое произведение, значение левой части выражения, номер и значение последнего сомножителя. Для проверки правильности работы программы на каждом шаге цикла будем выводить его номер и накопленное значение произведения.

Для решения задачи нам потребуются следующие переменные: `x` – точка, в которой вычисляется сумма ряда, `eps` – требуемая точность вычислений, `proiz` –

Оглавление

искомое произведение, p – очередной сомножитель. Все эти переменные имеют рациональный тип данных. Для повышения точности наших вычислений будем использовать тип Double.

```
Dim x, proiz, p, eps As Double
```

Так как общая формула члена ряда зависит от номера сомножителя, то для его хранения заведем переменную n.

```
Dim n As Integer
```

Очищаем окно списка.

```
lstA.Items.Clear()
```

Вводим исходные данные.

```
x = Val(InputBox("Введите точку"))
eps = Val(InputBox("Введите точность"))
```

Задаем начальные значения переменных. Обратите внимание, что начальное значение для произведения – единица.

```
proiz = 1
```

Номер текущего сомножителя равен нулю, так как никакого сомножителя на данный момент времени мы не вычислили.

```
n = 0
```

Организуем основной цикл.

```
Do
```

На каждом шаге цикла номер очередного сомножителя увеличивается на единицу.

```
  n += 1
```

По общей формуле вычисляем очередной сомножитель.

```
  p = 1 - 4 * x ^ 2 / ((2 * n - 1) ^ 2 * Math.PI ^ 2)
```

Включаем его в итоговое произведение.

```
  proiz *= p
```

Для проверки правильности работы программы выводим в окно списка номер шага цикла (он совпадает с номером текущего сомножителя) и накопленное значение произведения. Использование константы vbTab позволяет организовать вывод информации в две колонки.

```
  lstA.Items.Add(Str(n) + vbTab + Str(proiz))
```

Проверяем, если текущий сомножитель отличается от единицы меньше, чем на величину заданной точности, то дальнейшие вычисления не приведут к заметным изменениям результата, и выполнение цикла можно завершить.

Оглавление

```
Loop Until Math.Abs(1 - p) <= eps
```

Выводим горизонтальную черту, чтобы зрительно отделить результаты программы от промежуточных значений.

```
lstA.Items.Add("-----------------------------------")
```

В окно списка выводим итоговое значение произведения, значение левой части выражения, номер и значение последнего сомножителя.

```
lstA.Items.Add("Произведение =" + Str(proiz))
lstA.Items.Add("cos(x)=" + Str(Math.Cos(x)))
lstA.Items.Add("n=" + Str(n))
lstA.Items.Add("Последний сомножитель =" + Str(p))
```

Пример работы программы приведен на рис. 10. Исходные данные для этого случая: `x` = 1, `eps` = 1e-3 = 10^{-3}.

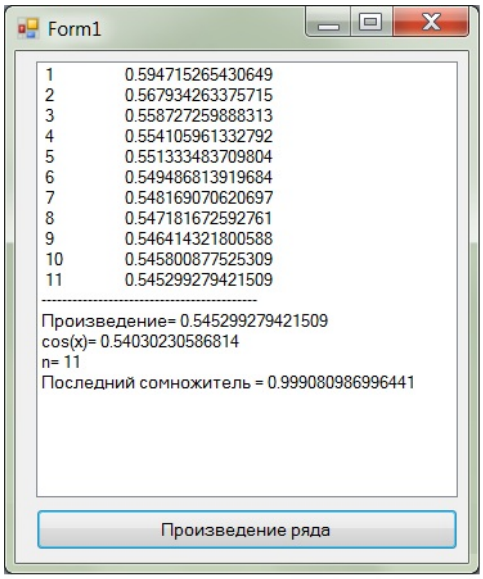

Рис. 10. Пример работы программы вычисления произведения ряда

2.6. Решение нелинейных уравнений методом простой итерации

Рассмотрим еще один пример использования цикла с условием. Это один из наиболее распространенных способов решения нелинейных уравнений вида

$$x = f(x).$$

Он называется метод простой итерации. Исходя из особенностей решаемой задачи, выбирается некоторое начальное приближение к решению уравнения. Как правило, оно обозначается x_0. Все последующие приближения к корню уравнения вычисляются по формуле

Оглавление

$$x_k = f(x_{k-1}),$$

где x_k – очередное приближение к корню уравнения, x_{k-1} – предыдущее приближение к корню уравнения. Процесс продолжается до тех пор, пока расстояние между точками x_k и x_{k-1} не станет меньше некоторого заранее заданного числа, которое является требуемой точностью вычислений ε. Схема работы метода простой итерации приведена на рис. 11.

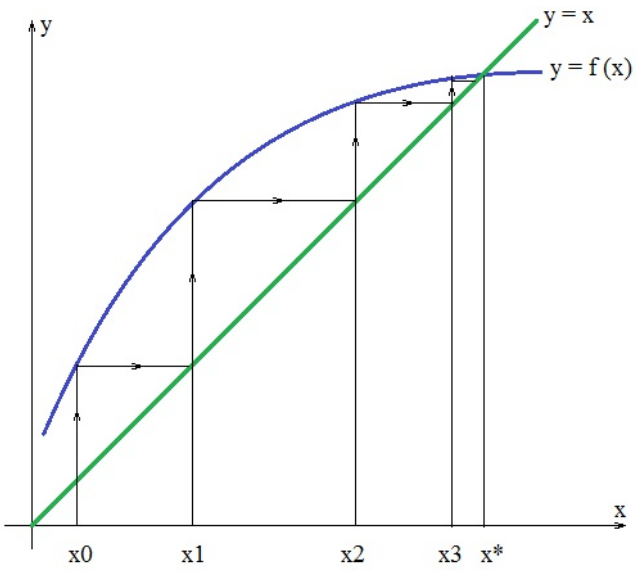

Рис. 11. Метод простой итерации

Рассмотрим особенности программой реализации данного алгоритма. Требуется решить уравнение

$$x = \cos x$$

методом простой итерации с заданной точностью. Начальное значение x_0 = 0,5. Единственным исходным данным для этой задачи является требуемая точность вычислений ε. Для ее ввода будем использовать функцию `InputBox`. Результатами программы будут значение корня уравнения, значение его правой части и количество потребовавшихся шагов цикла. Для проверки правильности работы программы на каждой итерации будем выводить номер шага и текущее значение корня уравнения. Весь вывод информации будем осуществлять с помощью окна списка с именем `lstA`.

Для решения задачи нам потребуются следующие переменные: `xTek` – очередное приближение к корню уравнения, `xPred` – предыдущее приближение к корню уравнения, `eps` – требуемая точность вычислений. Все эти переменные имеют

Оглавление

рациональный тип данных. Для повышения точности наших вычислений будем использовать тип Double.

```
Dim xTek, xPred, eps As Double
```

Так как на каждой итерации требуется выводить ее номер, то для его хранения потребуется целочисленная переменная. Назовем ее n.

```
Dim n As Integer
```

Очищаем окно списка.

```
lstA.Items.Clear()
```

Вводим требуемую точность вычислений.

```
eps = Val(InputBox("Введите точность"))
```

Задаем начальное приближение к корню уравнения.

```
xTek = 0.5
```

Начальное приближение также называют нулевым. Поэтому номер итерации равен нулю.

```
n = 0
```

Организуем основной цикл.

```
Do
```

В окно списка выводим номер итерации и значение текущего приближения к корню уравнения. Использование константы vbTab позволяет организовать вывод информации в две колонки.

```
lstA.Items.Add(Str(n) + vbTab + Str(xTek))
```

Вычисляем очередное приближение к корню уравнения. Запоминаем значение текущего приближения как предыдущее приближение к корню уравнения.

```
xPred = xTek
```

При переходе на следующую итерацию ее номер увеличивается на единицу.

```
n += 1
```

По методу простой итерации вычисляем новое приближение к корню уравнения. Для этого подставляем значение предыдущего приближения в выражение, стоящее в правой части уравнения.

```
xTek = Math.Cos(xPred)
```

Проверяем, насколько мало расстояние между текущим и предыдущим приближениями. Если это расстояние меньше требуемой точности вычислений, значит, мы нашли корень уравнения с нужной точностью, и процесс вычислений можно остановить.

Оглавление

```
Loop Until Math.Abs(xTek - xPred) < eps
```

Выводим горизонтальную черту, чтобы зрительно отделить результаты работы программы от промежуточных значений.

```
lstA.Items.Add("-----------------------------------")
```

В окно списка выводим полученные результаты: значение последнего приближения к корню уравнения, значение правой части уравнения и количество выполненных итераций (оно совпадает с количеством повторов цикла). Если программа написана правильно, то разница между первыми двумя числами будет незначительной.

```
lstA.Items.Add("x =" + Str(xTek))
lstA.Items.Add("cos(x) =" + Str(Math.Cos(xTek)))
lstA.Items.Add("Количество шагов =" + Str(n))
```

Пример работы программы приведен на рис. 12. Исходные данные для этого случая: `eps` $= 10^{-3}$.

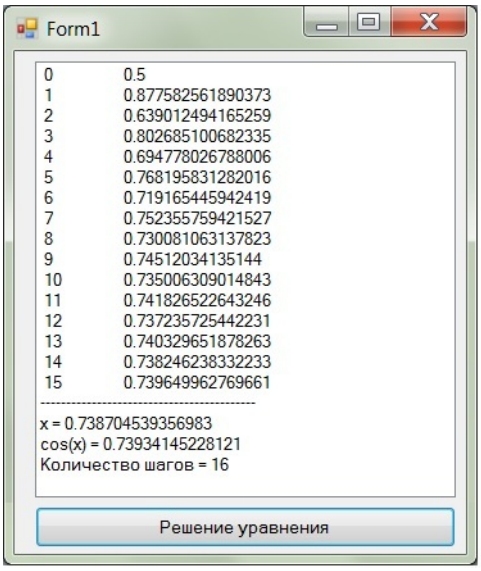

Рис. 12. Пример работы программы решения нелинейного уравнения

Приложение 1

Табуляция функции $y = \sqrt{x+4} - \frac{1}{x}$ с известным количеством узлов.

```
Dim a, b, h, x, y As Single
Dim n As Integer
vvod:
a = Val(InputBox("Введите начало отрезка"))
b = Val(InputBox("Введите конец отрезка"))
```

Оглавление

```
n = Val(InputBox("Введите количество узлов табуляции"))
If a > b Or n < 2 Then
    MsgBox("Неправильные данные")
    GoTo vvod
End If
lstResult.Items.Clear()
lstResult.Items.Add("x" + vbTab + "y")
h = (b - a) / (n - 1)
For x = a To b Step h
    If x + 4 < 0 Or x = 0 Then
        lstResult.Items.Add(Str(x) + vbTab + "Ошибка")
    Else
        y = Math.Sqrt(x + 4) + 1 / x
        lstResult.Items.Add(Str(x) + vbTab + Str(y))
    End If
Next
```

Приложение 2

Вычисление факториала заданного натурального числа.

```
Dim f As ULong
Dim i, n As Byte
n = Val(InputBox("введите натуральное число n"))
f = 1
For i = 2 To n
    f *= i
Next
MsgBox(Str(n) + "! =" + Str(f))
```

Список литературы

1. Волчёнков Н.Г. Программирование на Visual Basic 6: В 3-х ч. Часть 1. – М.: ИНФРА-М, 2000. – 286 с.
2. Шевякова Д.А., Степанов А.М., Карпов Р.Г. Самоучитель Visual Basic 2005 / под общ. ред. А.Ф. Тихонова. – СПб.: БХВ-Петербург, 2007. – 576 с.
3. Богданов М.Р. Visual Basic 2005 на примерах. – СПб.: БХВ-Петербург, 2007. – 592 с.

Оглавление