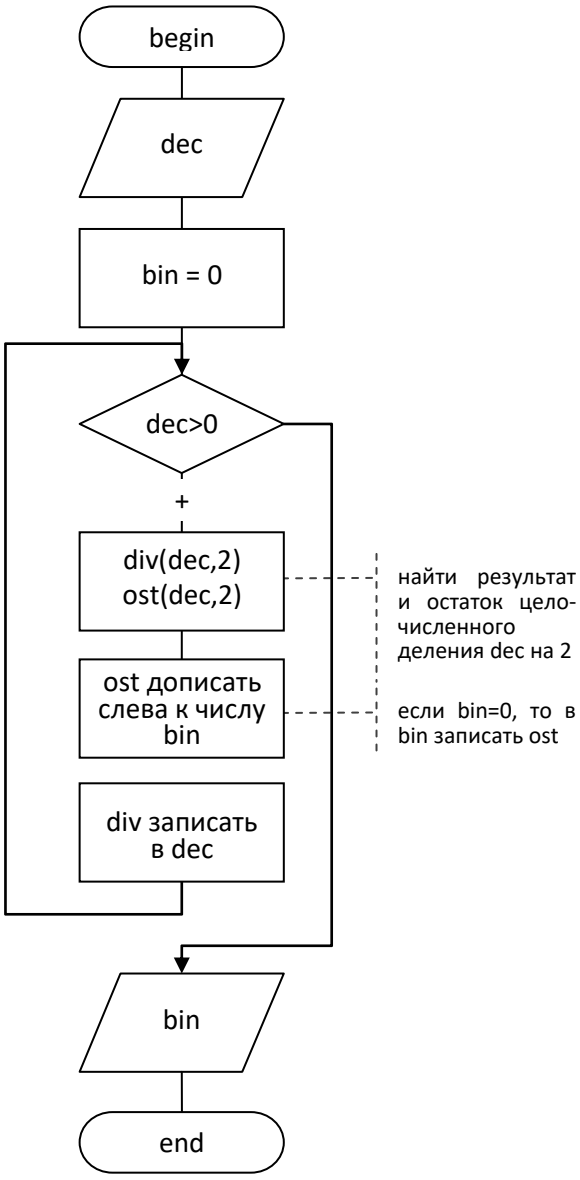
АЛГОРИТМЫ И ПРОГРАММНЫЙ КОД

Часть 0. Введение.

Алгоритмом определяется порядок действий для решения конкретной задачи. Для инициализации работы алгоритма нужны входные данные. Для алгоритма также заранее должны быть указаны и выходные переменные, приобретающие своё значение в процессе работы алгоритма. Схема алгоритма, обычно, представляет собой достаточно общее описание последовательности действий, не уточняющее мелкие детали реализации, и этого перечисления вполне достаточно для такого исполнителя как человек, так как его интеллект вполне позволяет додумать ему очевидные недосказанности. Однако для компьютера такая недосказанность может стать «камнем преткновения», поэтому компьютерная программа, даже написанная на языке высокого уровня, не вполне соответствует исходному словесному описанию, более детализирована частностями и содержит недостающие детали, выраженные синтаксисом языка и семантикой операторов.

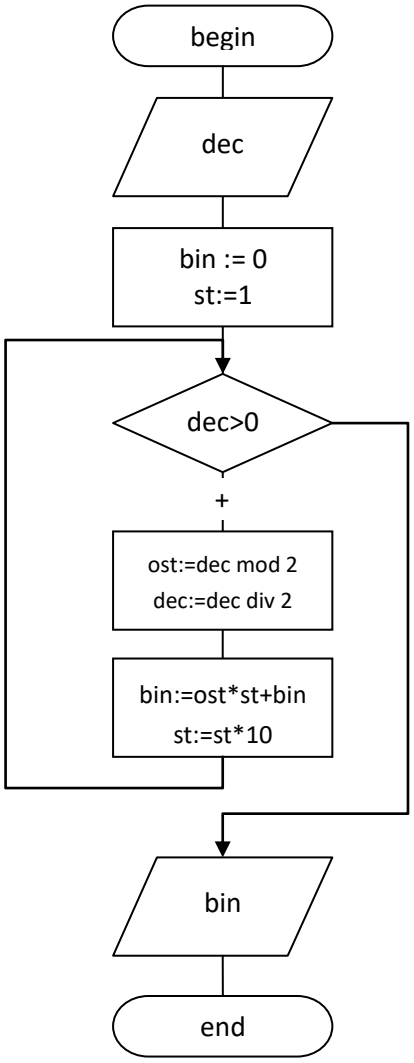


Для определенности рассмотрим пример перевода десятичного числа 13 в двоичную систему счисления в виде таблицы, демонстрирующей итерации решения и последовательный сбор ответа в обратном порядке из пошаговых результатов, и схему алгоритм по переводу целого десятичного числа в двоичную систему счисления:

исходное значение = **13**			
шаг	значение числа	целочисленное деление на 2	остаток от деления
1	13	6	1
2	6	3	0
3	3	1	1
4	1	0	1
ответ = **1101**			

порядок сбора

Приведенная схема алгоритма имеет достаточно общий характер: не уточняются типы входных и выходных данных, не указаны методы работы с данными, не навязывается язык программирования синтаксическими вкраплениями. Тем не менее, зачастую схема алгоритма составляется с детализацией синтаксиса, методов и типов данных:



Вид надписей в блоках данной схемы явно указывает на язык Pascal, а содержание определенно отражает метод работы с данными и подчеркивает, что ответ формируется в целочисленном, а не в строковом виде.

Для большинства случаев это удобно, так как алгоритм практически представляет программу. Сравните сами:

```
var dec,bin,st,ost: word;
begin
  writeln('Введите число: ');
  read(dec);
  bin:=0; st:=1;
  while dec>0 do
  begin
    ost:=dec mod 2;
    dec:=dec div 2;
    bin:=ost*st+bin;
    st:=st*10;
  end;
  write(bin);
end.
```

Однако следует понимать, что алгоритм может быть использован для реализации на любом языке программирования, не только отличающегося синтаксисом, но и обладающего иными, возможно лучшими, методами работы с данными.

Для примера приведу реализацию на языке Python:

```
def dec_bin(dec):
    bin=0                   # тут будем накапливать результат
    st=1                    # степень, чтобы цифру писать в определенный разряд
    while dec>0:
        bin+=dec%2*st       # остаток от целочисленного деления на 2
                            # умножить на степень и добавить к результату
        dec//=2             # dec целочисленно делить на 2 и записать в dec
        st*=10              # степень st домножить на 10 и записать результат в st
    return bin

dec = int(input('Введите число: '))
print(dec_bin(dec))
```

Синтаксис `Python` ближе к `C++` или `JavaScript`, но, несмотря на это, если у вас есть опыт написания программ на языках структурного программирования, то код Python'а вы тоже поймете и сможете адаптировать к своей среде разработки. Именно поэтому, в большинстве случаев, программисты для передачи логики работы обмениваются не схемами алгоритмов, а кусками программного кода. Тем не менее, схемы алгоритмов используют для документирования коммерческой работы, для наглядной передачи логики решения задачи и при обучении.

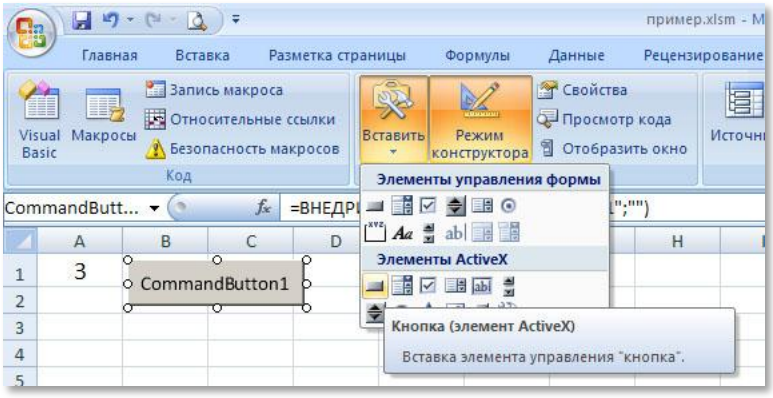
Обратите внимание, что основную работу без изменений логики обработки данных я вынес в отдельную функцию. Так делают для систематизации программы и удобства использования разработанной части кода во всех языках программирования. Подобного рода функции очень удобно использовать для расширения функциональных возможностей Excel средствами встроенного объектно-ориентированного языка программирования VBA (Visual Basic for Applications).

Далее вам предстоит выполнить несколько заданий, в некоторых из них требуется разработать только программный код на языке VBA, а в некоторых – ещё и схему алгоритма.

Чтобы вы смогли самостоятельно перейти к использованию возможностей офисного программирования, проделайте несколько примеров, изложенных ниже.

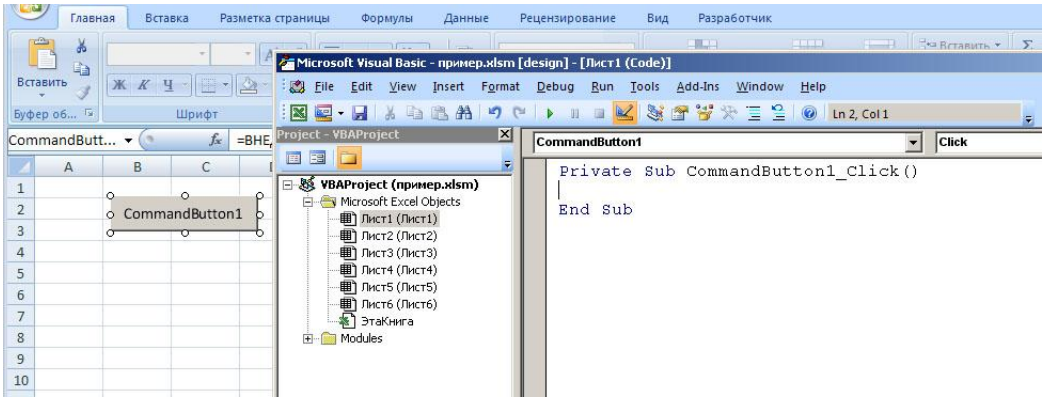
Часть 1. Подпрограммы VBA.

Подпрограмма это программный код, вынесенный из тела основной программы и имеющий собственное имя (идентификатор). Подпрограммы создают, прежде всего, в тех случаях, когда часть программного кода будет неоднократно использоваться при выполнении основной программы. Это позволяет минимизировать длину кода. Подпрограммы бывают двух видов: процедуры и функции. Подпрограммы могут иметь аргументы, через которые в подпрограмму передаются значения (входные аргументы) и возвращаются после вычислений значения (выходные аргументы) в основную программу. Подпрограмма, оформленная как функция, позволяет также возвращать в тело основной программы значение через имя функции. Идентификатор функции в этом случае можно вать как переменную, в которую размещается значение по итогам вычисления функции.



Запустите Excel и перейдите на ленте на вкладку «Разработчик» (если она не видна, то её можно подключить через кнопку Office / круглая в верхнем левом углу / далее «Параметры Excel» и далее «Показывать вкладку Разработчик на ленте»). На вкладке Разработчик включите режим конструктора. С помощью клавиши вставить выберите из элементов ActiveX объект CommandButton и разместите его на листе Excel кликом мыши.

Создадим обработчик события для этой клавиши, кликнув дважды по ней мышкой.



В ответ на двойной клик редактор кода VBE сгенерирует пустую процедуру для обботки этого события. В режиме работы, когда выключена клавиша «Режим конструктора», одиночный клик мышкой по клавише `mandButton1` будет запускать на исполнение процедуру `Sub CommandButton1_Click`.

Напишем программный код, решающий задачу возведения числа из ячейки A1 в степень, указанную в ячейке A2. Ответ будем размещать в ячейке A3.

```
Private Sub CommandButton1_Click()
  a = Cells(1, 1)
  b = Cells(2, 1)
  c = a ^ b
  Cells(3, 1) = c
End Sub
```

Разместите в ячейках A1 и A2 некоторые числа для контроля работоспособности процедуры. Отключите жим конструктора и запустите процедуру, кликнув по `CommandButton1`. Получится примерно такой результат:

Этот простой пример используем для ознакомления с организацией работы с функциями и процедурами в VBA. В качестве задачи для подпрограммы будет вычисление степени числа. Из обработчика события `CommandButton1_Click` на исполнение будет вызываться процедура или функция, реализующая возведение в степень числа. Для чего нужно будет использовать аргументы подпрограммы для передачи в неё значений и возвращение обратно результатов вычислений.

```
Sub p_st(x, y, z)
  z = x ^ y
End Sub
Private Sub CommandButton1_Click()
  a = Cells(1, 1)
  b = Cells(2, 1)
  p_st a, b, c
  Cells(3, 1) = c
End Sub
```

Начнем с разработки подпрограммы вида процедура. Измените программный код, написанный ранее следующим образом, предваряя обработчик события новой процедурой p_st:

В процедуре p_st первые два аргумента входные и третий выходной.

> Внимание! Все изменения, производимые с программным кодом, обязательно апробируйте изменением входных данных на листе Excel и запуском кода на исполнение нажатием на клавишу CommandButton.

Обратите внимание, что VBA не является строготипизированным языком программирования и использование переменных без указания типа данных, которые в них будут храниться. На самом деле под переменные, как и при формальном указании типа данных, выделяется определенное фиксированное пространство в оперативной памяти. В VBA для этих целей используется тип данных Variant, одна переменная данного типа занимает 16 байт. Такое неявное объявление типа данных для переменных универсально (в некотором смысле облегчает работу, в том числе, для начинающих программистов), но требует больше места в памяти и больше времени при циклической обработке данных.

При явном объявлении типов данных для переменных программный код можно было бы выразить следующим образом:

```
Sub p_st(x As Integer, y As Integer, z As Double)
  z = x ^ y
End Sub
Private Sub CommandButton1_Click()
  Dim a As Integer
  Dim b As Integer
  Dim c As Double
  a = Cells(1, 1)
  b = Cells(2, 1)
  p_st a, b, c
  Cells(3, 1) = c
End Sub
```

Теперь исследуем организацию работы VBA с функциями. Преобразуем написанный ранее код до следующего вида:

```
Function p_st(x As Integer, y As Integer) As Double
  p_st = x ^ y
End Function
Private Sub CommandButton1_Click()
  Dim a As Integer
  Dim b As Integer
  Dim c As Double
  a = Cells(1, 1)
  b = Cells(2, 1)
  Cells(3, 1) = p_st(a, b)
End Sub
```

Обратите внимание, что из подпрограммы p_st мы убрали третий аргумент, а результат вычисления передаем через само имя подпрограммы p_st. Так же не забудьте изменить объявление подпрограммы – `Function`.

Именно данная возможность позволяет разрабатывать так называемые рекурсивные функции, то есть такие которые вызывают сами себя. Они используются для итеративной обработки данных без применения операторов цикла.

Реализуем функцию p_st вычисления неотрицательной степени числа, но без использования оператора возведения в степень и с обыкновенным оператором параметрического цикла:

```
Function p_st(x As Integer, y As Integer) As Double
  p_st = 1
  For i = 1 To y
    p_st = p_st * x
  Next
End Function
```

До цикла инициализируем p_st значением 1, чтобы при вычислении нулевой степени любого числа получить результат 0.

Теперь перепишем код для реализации рекурсивной обработки данных:

```
Function p_st(x As Integer, y As Integer) As Double
  If y > 0 Then
    p_st = p_st(x, y - 1) * x
  Else
    p_st = 1
  End If
End Function
```

Обратите внимание, что в функции присутствует оператор ветвления, который реализует два пути перебора данных. Первый путь это шаг рекурсии (p_st = p_st(x, y – 1) * x), на котором производится изменение данных и рекурсивный вызов самой функции. Изменения такие: до этого был вызов функции p_st(x, y) для значения y, а теперь для измененного значения y-1. Если бы данные не изменялись, то рекурсия бы зациклилась. Второй путь это базис рекурсии, который предназначен для останова перебора при наступлении определенного условия.

Обработка начинается со степени обозначенной пользователем, допустим нужно возвести число два в третью степень (2^3=2*2*2). Интерпретируем смысл кода следующим образом. Чтобы найти два в третьей, нужно сначала найти два во второй и только потом (когда будет найдено решения для «два во второй») домножить на два. В свою очередь рекурсивный вызов функции два во второй работает аналогично: чтобы найти два во второй нужно найти два в первой и домножить на два. Рекурсивный вызов функции два в первой: это два в нулевой домножить на два. Наконец, при запросе значения два в нулевой оператором if производится выбор второго пути реализации p_st=1 и рекурсия прерывается. На обратном ходе рекурсии один домножается на два положенные три раза и мы получаем искомый результат.

Задания для самостоятельного исполнения.

Разработайте функции:

1) рекурсивную функцию для вычисления факториала числа N.

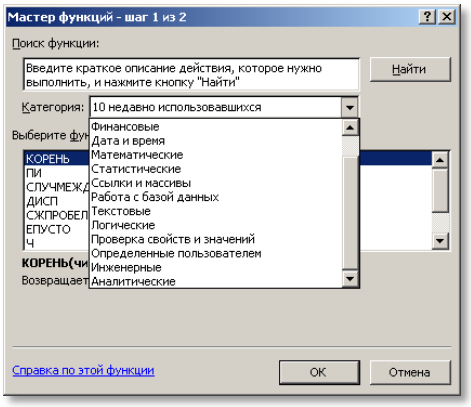
Факториал числа N это произведение чисел от 1 до N.
N!=1*2*3*…*N–1*N.

2) рекурсивную функцию для вычисления суммы нечетных чисел от 1 до N.

3) разработайте алгоритм и реализуйте функцию (не рекурсивную) поиска суммы N членов ряда такой последовательности:

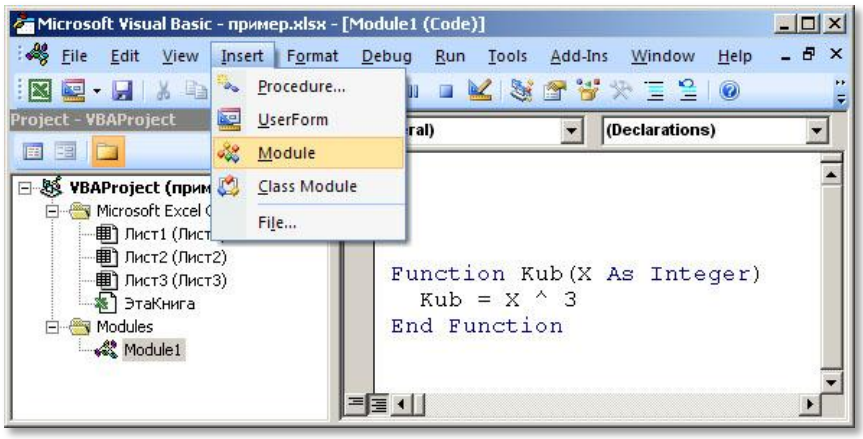
2/1 + 2/3 + 4/3 + 4/5 + 6/5 + 6/7 ...

Часть 2. Разработка пользовательских функций для Excel.

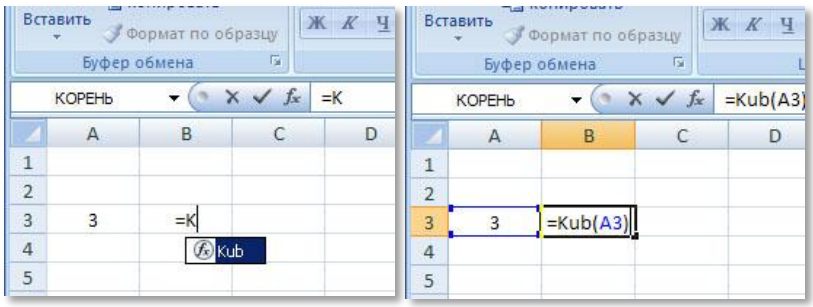


На листе Excel в ячейках вы можете размещать формулы, построенные на основе значительного количества встроенных функций (см. рис). Однако есть возможность использовать таким же образом и собственные функции. Для этого нужно сначала их разработать в редакторе VBE и затем подключить в формулу в ячейке в необходимом месте точно так же как вы подключаете стандартные функции Excel, то есть в ячейке начинаем набор формулы со знака «=» и далее пишем необходимый набор функций с адресами аргументов.

Начнем с простого примера. Пусть функция ищет значение куба числа – $y = x^3$. Назовем функцию Kub. Для входа в редактор кода нажмите сочетание клавиш Alt+F11. Если вы ранее не создавали модуль, то через меню Insert вставьте модуль и в редакторе кода опишите функцию (см. рис). Она без дополнительных манипуляций становится доступной для набора в формулах на листах Excel.



Начните набирать формулу в ячейке листа Excel обычным образом со знака «=», затем вводите нашу функцию и после первого же символа появится контекстная подсказка (см. рис), можно кликнуть по ней мышкой и, в качестве аргумента, указать адрес, откуда его брать. Нажмите Enter, закончив ввод формулы, и вы увидите, что она уже работает. Очень удобно и быстро. Естественно формулу можно копировать, редактировать и удалять, как и при использовании стандартных функций. Таким образом, можно создавать любые необходимые функции.

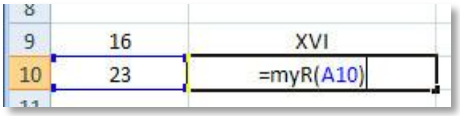


Задания для самостоятельного исполнения.

Создайте пользовательские функции для перевода:

4) десятичного числа в римскую систему счисления (см. рис), используя этот код:

```
myR = Application.WorksheetFunction.Roman(D)
```

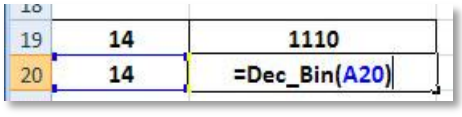


5) градусов в радианы, используя этот код:

```
R = G * Application.Pi / 180
```

6) целого десятичного числа в двоичное Dec_Bin (см. рис);

7) двоичного числа в десятичное Bin_Dec.



В заданиях №3 и №4 вам требуется самостоятельно написать программный код функций в модуле с использованием операторов цикла.