Number to Words

The Ring is an innovative and practical general-purpose multi-paradigm language. The supported programming paradigms are imperative, procedural, object-oriented, declarative using nested structures, functional, meta programming and natural programming. The language is portable (Windows, Linux, macOS, Android, etc.) and can be used to create Console, GUI, Web, Games and Mobile applications. The language is designed to be simple, small, flexible and fast.

In addition, the fundamental principle of RING are :

• Applications programming language. Examples?
• Productivity and developing high quality solutions that can scale.
• Small and fast language that can be embedded in C/C++ projects. How?
• Simple language that can be used in education and introducing Compiler/VM concepts. How?
• General-Purpose language that can be used for creating domain-specific libraries, frameworks and tools. How?
• Practical language designed for creating the next version of the Programming Without Coding Technology software. What?

Homepage:
http://ring-lang.sourceforge.net

Keywords:
https://ringlang.wordpress.com/2016/01/29/list-of-the-ring-programming-language-keywords-and-functions/

Quell: Ring Home Site.

Preamble

Programming is no different than splitting a problem into several smaller ones and then, one after the other, solving them. let's assume we want to represent any number in words.

Writing a small program that gives the results you want is then not that difficult. One that can be created properly and easily expanded takes several aspects into account.

Suppose we want to write an app that converts a number to words.
For example the number: 1,234,567 as:

> one million two hunderd thousand five hunderd sixty seven

Let's address a problem in the following steps:

1. Think about the problem.
2. Break up the problem into parts and write it down formally.
3. Convert the parts into the computer language you are using (pseudocode?).
4. Test the individual parts for functionality.
5. Put the parts together.

How do we start?

Step by step, one after the other..

Introduction

Hello to all who want better or want to Program at all.

RING as a programming language seems to me to be well suited for everyone who wants to learn programming. In the following text I try to shed some light on the actions of a programmer (or one who wants to become one).

The Texts here are fairly concise. The focus here is on the procedure.

Number to words - How do i proceed?

Analysis

The beginning is a consideration:
What do we want?
How do we tackle the problem?
What is a longer number?

We will tackle the task in each step and check them continuously:

Step 1 - Digits from 1 to 19

Topics: Array, input, input control (if, then, else), output.
We start off casually and write a piece of code with the task of spending a number between 1-19 in word. Inspired by the success, we continue.

Step 2 - Digits from 20 to 90

Topics: The same as in part 1
Extension of the code from part-1 by whole numbers - the tens groups: 20-90.

Step 3 - Digits from 1 to 999 (Hundreds)

Topics: the same as before plus function with recursion.
let's deal with the representation of the hundreds (the first of a group of three) and the solution with a recursive function (it's always the same!).

Step 4 - Thousands / Millions Groups

Topics: topics: as before plus division of input into groups.
breaking down the number into individual groups (three numbers each) and dividing them into thousands, millions and so on.

Step 5 - Output of the whole Number as Word

Topics: assembly and testing
assembly of all program parts created so far.

Step 6 - Program with graphical Interface

Topics: same as part-5 but with graphical UI

Analysis

How do we proceed that?

At closer inspection the number: 1234567, we see three groups. The number “1” refers to the group million. the number “234” for the group of thousands and in the last group the number “567”. This however has no name of groups. But it is, as the previous group, also composed of different elements. the elements (or items) are:

5 hundred
6ty 7

So hundreds, tens and the ones.

The logic is visible. we must consider the entire input (1,234,567) to distribute triplets, give them the right groupname (mio, thousand .. and so on) and the (triplets) to disassemble your items (hundreds, tens and ones).
But we will do this from behind (bottom up .. because of the small numbers) and then read out all in reverse order.

It is important to look first at the structure of the number. eg: 1234567

```
          hto
          |||
      1 234 567
```

```
|    |
g    g
```

o: ones (1-19)
t: tens (20-90)
h: "hundred"
g: groups (tsd, mio, mrd ...)

Let's start from the back. 7,6 and 5 consists of: seven (ones) and sixty (tens) and a 5 (representing the hundreds "h"). this is the first group (seen from the back) that is the first to be evaluated and converted.

We want to call the numbers 1-19 all ones. the whole numbers 20-90 tens.
With that, every number from 1 to 999 can be converted into words. first of all the last two, so: 67 (sixty seven) and with the same group (ones) and a word "hundred" the first as: five hundred.

The group straightened out: five hundred sixty seven.

So once again, in order:

1 million
234 thousand
567 (no group name)

Each of the groups may consist of one, two or three charged. This repeats the same. we look at the last 3 Group: six hundred seventy-eight. The first number, if any, is always 1-9 hundred. so there is still a 2er group 78. The first number, tens and the rest ones. Now the task is practically solved.

We need a procedure which searches the input for groups of 3, this an appropriate name (mio, thousand. etc.) allocates, then the individual groups of 3 decomposed, the hundreds and tens determined composed with ones.

All right? Then we go further.

Step 1

1. Digits from 1 to 19

We start small and have a look at how we pay simple 2s starting with one (1-19) to convert into words.
we need the number names for this. We capture these in an array (or list). We call these "ones".

Units or (better say) ones:

```
// units:
ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven", " ... ", "nineteen"]
```

We make it a little motivational test:

```
// *** step 1
// number to words (units:)

ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven",
 "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"]

n = "12"                 // input as string (the number can be any length)
? "input: "+ n           // for control only

if number(n) > 19       // test input
 ? "Incorrect input!"   // message
else
 ? ones[number(n)]      // ones-element nr: 12 output: twelve
ok
```

Well.. the beginning is half of the whole.

The program works.. Well served by 1 to 19,
Fine.. we are now a step further.

Lets move on.

Step 2

2. Digits from 20 to 90

What is needed to pay more than 19 to convert?
We need another name list, the "tens" (whole numbers) that we will combine with the "ones".

Now we add the app to integers (20-90).
The tens we collect in also in an array:

```
// tens:     (whole numbers)
tens =["" , "twenty",  "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety"]
```

```
// *** step 2
// number to words (tens)

load "stdlib.ring"                              // need for string func.
    load "stdlib.ring"          // need for string func.

    // units:
    ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven",
    "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"]

// tens: (whole numbers)
tens=["" , "twenty",  "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety"]

n = "29"
? "len(n): "+len(n)         // n: is string
? "first digit: "+n[1]      // first digit
? "second digit: "+n[2]     // second digit
? "input: "+ n

if number(n) > 99           // test
 ? "Incorrect input!"       // message
end

if number(n[1]) > 1 and number(n[2]) > 0
  ? tens[number(n[1])] +"-"+ ones[number(n[2])]
elseif number(n[2]) = 0
  ? tens[number(n[1])]
end
```

Great.. that works..
With the two parts developed here, the whole logic is almost solved.
In the next step we will need a function for group formation:
The "Hundreds" term.

The program now works from 1 to 99. Well .. we go further.

Step 3

3. Digits from 1 to 999 (Hundreds)

Now we can output numbers from 1 to 99 in words. In order to evaluate a complete group of 3 digits, we need a small extension - the hundreds. How does this work? Simple. The first number of a triplet is always a number of hundreds:

```
  ...
    elseif len (n) = 3              // first place: hundreds
    z = right (n, 2)                // tens / ones
    h = (Floor (n / 100)) * 1       // Find the hundreds with Math-Func.: Floor ()
    rtn = ones [h] + "hunderd"      // pe: 6 hunderd
    rtn = rtn + doWord (z)          // Recursion
  ...
```

The "hundreds" are needed in each group. there are not only a hundred as such, but also a hundred thousand or a hundred thousand million, but also a hundred thousand billion. Our program should be able to convert as long as you want. For this we need a recursive (self-calling) function.

Here it is: doWord ()

```
// *** step 3
// number to words (hunderds)

load "stdlib.ring"   // need for string func.

n = "101"            // input
? "input: "+ n       // show input
? doWord(n)          // print the result in words

// -----------------------------------------------------------------------
func doWord(n)     // recursive (self-calling) function.
 ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven",
"twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"]

 tens=["" , "twenty",  "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety"]

// the zeros in the groups or whole null groups are not treated.

if n="000" see " Zero-Grp "+nl return end // a null group
if n="00" return end
if number(n)< 20         // 1-19
  n = number(n)          // convert to num
  rtn = ones[n]          // ret value
 elseif n < 100          // 20-99
  n=n*1                  // convert to num (another possib.)
  x = n % 10             // for ones
  y = floor(n / 10)      // for tens
  if x = 0 rtn = tens[y]
  else rtn = tens[y] +" "+ ones[x]  // whole tens
  end
 elseif len(n) = 3       // first place: hundreds
  z = right(n,2)
  h = (floor(n / 100)) *1   // Find out the hundreds
  rtn = ones[h] +" hunderd "
  rtn = rtn + doWord(z)     // And.. (Recursion)
 end //if
return rtn                  // pick up
//? "doWord-rtn: "+rtn      // debug
//end-doWord()
```

We are now so far that a complete threegroup is output correctly. The app is capable of an input from 1 to 999 to properly resolve and output.

The program now works from 1 to 999.. well .. we go further.

Step 4

4. Groups: Thousands, Millions ...

In order to process a multi threegroup input we require one addition. We need another list. the list of names for the three groups - the "thousands" (tsds).

Here are the group names:

tsds = ["","thousand ", "million ", "billion ", "trillion ", ...

We make again a little test:

```
// *** step 4
// number to words (illions)

n = "1234567898"
//n = "1000067"


doGruppen(n)

func doGruppen(n)
        rtn = " "
        anz_gr = floor((len(n)+2)/3)            // anzahl gruppen

        // group names - the "thousands"
        tsds = ["","thousand ", "million ", "billion ", "trillion ", "quadrillion ", "quintillion ",
"sextillion",
    "septillion", "octillion", "nonillion", "decillion", "undecillion", "duodecillion", "tredecillion",
    "quattuordecillion", "sexdecillion", "septendecillion", "octodecillion", "novemdecillion",
"vigintillion "]

        for x = 1 to anz_gr-1                           // eingabe abarbeiten
                gr = right(n,3)                                 // gruppe ermittelln
                n = left( n,len(n)-3 )                  // neue nr generieren
                if gr = "" gr = n end                   // wenn gr leer "n" nehmen
                rtn = gr+" "+ tsds[x] + rtn
        next  //end-for

        rtn = n +" "+ tsds[x] + rtn
        ? "In Worten: "+ rtn                            // Ausgabe
return rtn
//end   doGruppen
```

And .. as we can see - it works.

The thing seems to have gone. now we only have to assemble all the parts that we have created and successfully tested individually. We do this in the next step.

Step 5

5. Output of the whole string.

A The completely assembled program:

```
// *** Step-5
// number to words (whole string)

// efective (netto) approximately 30 lines of code for an arbitrary number of words.
// currently limited by tsds ["vigintillion"] (can be traced)
// author: nestor kuka
// datum: 20.02.2018

// Load Ring Libraries
    Load "stdlib.ring"

// example input:
pConvert("1234567")
pConvert("10000009")    // zero-group: "000"
pConvert("103010009")
pConvert("103000109")   // zero-group: "000"
pConvert("103406789")
pConvert("103400789")
pConvert("103400009")
pConvert("123456789")
pConvert("123456789123456789123456")
pConvert("123004567890034567801230 56")
pConvert("12345678912345678912345678912345678912345678912345678912345678")

//-----------------------------------------------------------------------
Func pConvert(n)
if len(n) < 63
  ? "Input: "+n
  doGruppen(n)
else
  ? "Do you really have that much money? ;)"
 end
//end-pConvert

//-----------------------------------------------------------------------
func doWord(n)
 ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven",
 "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"]

 tens=["" , "twenty",  "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety"]

// input test:                  // possibly not kommplett
// the zeros in the groups or whole null groups are not treated.
//if n="000" see " Zero-Grp "+nl return end // a null group
if n="00" return end
if n="000" rtn = "zero" ok
if number(n)< 20 and n!="000"   // 1-19
  n=n*1                         // convert to num
  rtn = ones[n]                 // ret value
 elseif n < 100 and n!="000"    // 20-99
  n=n*1                         // convert to num
  x = n % 10                    // for ones
  y = floor(n / 10)             // for tens
  if x = 0 rtn = tens[y]
  else rtn = tens[y] +" "+ ones[x]  // whole tens
  end
 elseif len(n) = 3 and n!="000"  // first place: hundreds
  z = right(n,2)
  h = (floor(n / 100)) *1        // Find out the hundreds
  rtn = ones[h] +" hundert "
  rtn = rtn + doWord(z)          // And.. (recursive deal)
 end //if
return rtn                       // ..pick up
//? "doWord-rtn: "+rtn           // for debug only
//end-doWord()

//-----------------------------------------------------------------------
func doGruppen(n)
 rtn = " "
 anz_gr = floor((len(n)+2)/3)    // number of groups
 tsds = ["","thousand ", "million ", "billion ", "trillion ", "quadrillion ", "quintillion ",
"sextillion",
 "septillion", "octillion", "nonillion", "decillion", "undecillion", "duodecillion", "tredecillion",
 "quattuordecillion", "sexdecillion", "septendecillion", "octodecillion", "novemdecillion",
"vigintillion "]
for x = 1 to anz_gr-1           // execute input
  gr = right(n,3)               // determine group
  n = left( n,len(n)-3 )        // generate new nr
```

```
  if gr = "" gr = n end          // if gr blank take: "n"
    rtn = doWord(gr) +" "+ tsds[x] + rtn
 next  //end-for
 rtn = doWord(n) +" "+ tsds[x] + rtn
 ? "In Worten: "+ rtn            // output
return rtn
//end doGruppen
```

Of course, the whole thing could be described in much more detail. It has it as food for thought
Hopefully goal achieved. The initiative is now yours...
Do you accept this challenge?

Step 6

6. Graphical user interface

A Ring program (with a graphical UI) could look like this..

```
// *** Step-5
// number to words  (graphical user interface)

// currently limited by tsds ["vigintillion"] (can be traced)
// file: C:\ring\Ring17\RingSamples\src\n2w03c.ring
// author: nestor kuka
// datum: 20.02.2018

import System.GUI
// Load Ring Libraries
    Load "guilib.ring"
    Load "stdlib.ring"

MyApp = New qApp {
        win1 = new qWidget() {
                // window titel
                setwindowtitle("Convert Digits to Words.")
                setGeometry(200,100,270,420) // Pos Left/top - Right/bottom
                label1 = new qLabel(win1) {
                        settext("Eingabe:")
                        setGeometry(10,10,350,30)
                        setalignment(Qt_AlignLeft)
                }
                label1 = new qLabel(win1) {
                        settext("Ausgabe in Worten:")
                        setGeometry(10,80,350,30)
                        setalignment(Qt_AlignLeft)
                }
  label2 = new textedit(win1) {
   setstylesheet("color:#0055ff;background-color:white;")
   oFont = new qfont("",0,0,0)
   oFont.fromstring("Arial")
   setfont(oFont)
   oFont.delete()
   setText("Ausgabe...")
   setGeometry(10,100,250,250)
  }
                btn1 = new qpushbutton(win1) {
                        setGeometry(10,360,110,37)
                        settext("Convert")
                        setclickevent("pConvert()")
                }
                btn1 = new qpushbutton(win1) {
                        setGeometry(150,360,110,37)
                        settext("Close")
                        setclickevent("pClose()")
                }
                lineedit1 = new qlineedit(win1) {
                        setGeometry(10,30,250,30)
                }
                show()
        }
        exec()
}

//-----------------------------------------------------------------------------------------
//                                  F u n c t i o n s:
//-----------------------------------------------------------------------------------------
Func pConvert
 n = lineedit1.text()
if len(n) < 63
  label2.settext(doGruppen(n) )
else label2.settext("Haben Sie wirklich so viel Geld?") end

//-----------------------------------------------------------------------------------------
Func pClose
        MyApp.quit()

//-----------------------------------------------------------------------------------------
func doWord(n)
 ones=["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven",
 "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"]
 tens=["" , "twenty",  "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety"]
if n="000" see " Zero-Grp "+nl return end
if n="00" return end
if number(n)< 20                // 1-19
  n=n*1                         // convert to num
  rtn = ones[n]                 // ret value
```

```
 elseif n < 100                  // 20-99
  n=n*1                          // convert to num
  x = n % 10                     // for ones
  y = floor(n / 10)              // for tens
  if x = 0 rtn = tens[y]
  else rtn = tens[y] +" "+ ones[x]  // whole tens
  end
 elseif len(n) = 3               // first place: hundreds
  z = right(n,2)
  h = (floor(n / 100)) *1        // find out the hundreds
  rtn = ones[h] +" hundert "
  rtn = rtn + doWord(z)          // and ..
 ok //if
return rtn                       // take it
//end doWord()

//-------------------------------------------------------------------------------------
func doGruppen(n)                // groups
 rtn = ""
 anz_gr = floor((len(n)+2)/3)    // number of groups
 tsds = ["","thousand ", "million ", "billion ", "trillion ", "quadrillion ", "quintillion ",
"sextillion",
 "septillion", "octillion", "nonillion", "decillion", "undecillion", "duodecillion", "tredecillion",
 "quattuordecillion", "sexdecillion", "septendecillion", "octodecillion", "novemdecillion",
"vigintillion "]

for x = 1 to anz_gr-1            // Process input
  gr = right(n,3)                // investigate group
  n = left( n,len(n)-3 )         // generate new no
  if gr = "" gr = n end          // if gr empty take "n"
   rtn = doWord(gr) +" "+ tsds[x] + rtn
 next  //end-for
 rtn = doWord(n) +" "+ tsds[x] + rtn
return rtn
//end doGruppen()
//-------------------------------------------------------------------------  end pgm
```

Recapitulation

In this small tutorial we have learned the following areas:

- **Problem Analysis.**
- **Decomposition of the solution on items.**
- **Create the individual (individually testable) procedures and functions**.

Applied elements:

Arrays
Variables
Strings
String to Num: number()
String-length: len()
String-Pos.: right(), left()
Program controll with if, for loop, Input check
Function call (with Recursion and Rerurn Value)
Math Func: floor()
Ring Libraries: "guilib.ring", "stdlib.ring"

About

It's about converting an arbitrarily long number into words. That means: input as a string (because of any length). Break down into groups of three and then loop until the groups and the individual numbers are evaluated and output.

I tried to keep it as simple as possible and hope that it doesn't look too poor.
The idea was to win the BEGINNERS for a simple and very flexible programming language.

If this post could be useful, I would be happy. And should you feel like more, the book by Mansour Ayouni (which is apparently nearing completion) will help you breastfeed.
Thank you in advance for any comments or improvements. Well-founded criticism is always welcome (it is THE best opportunity to get better).

Greetings Nestor

Es geht darum eine beliebig lange zahl in Worte umzuwandeln. Das bedeutet: Eingabe als String (weil beliebig lang). Zerlegung auf Dreiergruppen und dann so lange loopen bis die Gruppen und die einzelnen Zahlen ausgewertet und ausgegeben sind.

Ich habe versucht es so einfach wie möglich zu halten und hoffe, dass es nicht allzu dürftig wirkt. Die Idee dahinter war, den EINSTEIGER für eine einfache und sehr flexible Programmiersprache zu gewinnen.

Wenn dieser Beitrag nützlich sein könnte, würde es mich freuen. Und sollte dabei Lust auf mehr kommen, im Buch von Mansour Ayouni (welches offenbar kurz vor der Vollendung steht) wird man es reichlich stillen können.

Für jeweilige Anmerkungen, Verbesserungen danke ich im Voraus. Begründete Kritik ist immer willkommen (es ist die beste Gelegenheit besser zu werden).

Gruss Nestor