loop control and repetitions

...with a glimp on performance

by Blazko, mail@neveprise.de

Version: 1.01, 2000-12-19

This article discusses loop iterations with an issue on performance.

Table of contents

- 1.0 introduction
- 2.0 first of all: what is a loop?
- 3.0 kinds of loops
  - 3.1 repeat...until
  - 3.2 while ? do...
  - 3.3 for ? do...
  - 3.4 some others
- 4.0 and what does it mean?
  - 4.1 Example #1
  - 4.2 Example #2 (or: slightly more performance)

1.0 introduction

> top of page

This is no new topic at all: what kinds of loops exist and how do they work. But when creating apps that tend to process a lot of repetitions, this stuff becomes more and more important. This document tries to explain a few loop techs and show their (dis-) advantages.

2.0 first of all: what is a loop?

> top of page

Stupid question - eh? Okay, here a silly and state-of-the-art (gee:-) answer. A loop is nothing more than a repetition that is beeing processed by your application. Often, loops are used to step thru lists of any kind and evaluate their values or properties. On the other hand, there are more basic loops: they simply repeat some methods a several times. For example, this text output has been created simply by a loop that is processed 42 times:

```
...
Deepthought sez:
Your answer is "40"...question forgotten.
Your answer is "41"...question forgotten.
Your answer is "42"...question forgotten.
```

You see, there has just been called some write method that outputs some characters and incremented a value. Maybe you know the simpliest loop from your C64 Basic prompt:

```
10 ? "Hello world!"
20 Goto 10
30 Run
```

This code segment forms an infinite loop that just displays "Hello world!" unless it is stopped externally by the user (reset button).

A vital job for loops is needed in databases. There you have tables with equally formatted values - an array. This can look like dis:

ID	Name	URL	eMail
1	The Trash Inc.	http://www.the-trash-inc.de	mail@the-trash-inc.de
2	Dream Dimensions	http://www.dream-dimensions.de	mail@dream-dimensions.de
3	AlGore Artworks	http://www.saufcouch.de	mail@saufcouch.de
4	neveprise.net	http://www.neveprise.net	mail@neveprise.de
5	Simple Disease II	http://www.simpledisease.de	mail@simpledisease.de

>

If I want to know the email adress of AlGore Artworks, I have to search the table for a special criteria, e.g. it’s ID or name. And that is what a loop does: it walks thru each row of the table and compares an actual field (Name or ID) with the value I fed the loop with. If found, the loop exits that time thus leaving me in the right row. Now I can query all fields of that row; such as URL or eMail. <p>

You now might think: okay, why is performance thus important? - And you might be right, your local database with some CD texts etc. might do well, but large commercial databases are holding some millions of datasets (rows). To search’em, an efficient coding is very mission critical - so to speak.

3.0 kinds of loops

> top of page

Of course, there exist some different types of loops with their very own pros and contras. But before doing lots of theory, let us just take a sneak on some coding, all doing the same.

3.1 repeat...until

> top of page

One of the most popular loop is the "repeat...until" statement. It looks like dis:

```
i:= 0;
repeat
    write("Hello!" + IntToStr(i+1));
    Inc(i);
until i= 41;
```

An integer variable called "i" is init'ed to the value of zero. The reserved word "repeat" introduces the block of code that has to be repeated. In that block, we just write "Hello!", the actual value of "i" and increment the value of "i" by one (is the same as "i:= i + 1;" or "i++" in C and Java). The word "until" defines the condition that will exit the loop gracefully. In this case, the loop is repeated as long as "i" is no larger than 41. Cos we start with an value of zero instead of one, the loop executes 42 times.

No matter how you turn the fish, this loop is executed at least one pass, cos the check of "i" is performed at the bottom of our loop. You can test it by initialzing "i" to 41 - you'll see that write()" is called once.

3.2 while ? do...

> top of page

"While do" loops turn it the other way, they perform the check prior each block execution:

```
i:= 0;
while i < 42 do begin
    write("Hello!" + IntToStr(i+1));
    Inc(i);
end;
```

Again, we set the int var to zero. Then, "while" checks if "i" is smaller than 42 and - if so - execs the block until "end" and continues to check the condition again.... For all values from 0 to 41, the "write()" block is called; that sums 42 passes. To see that this loop checks the condition at its top, you might set "i" to 42 and see how often write is done: 0 times. In contrast to the "repeat...until" loop that is performed at least once, a "while...do" loop might not be executed at all.

3.3 for ? to ? loops

> top of page

The for...to loops are used very often. In contrast to the loop kinds we spoke before, for loops just execute the block predefined times:

```
for i:= 0 to 99 do write("Hello!" + IntToStr(i+1));
```

The "for" reserved word inits i to zero and tells the following block to be called 100 times. Inside the loop, the i variable can be queried as well. The only way to end the loop at less than 100 passes is to quit the loop from its inside block:

```
for i:= 0 to 99 do begin
    write("Hello!" + IntToStr(i));
    if SomeStringList[i] = 'Doh!' then Exit;
end;
```

or

```
for i:= 99 downto 0 do begin
    write("Hello!" + IntToStr(i));
    if SomeStringList[i] = 'Doh!' then Exit;
end;
```

In C/C++/C# it'll look like dis:

```
for (i = 0, 99, i++)
{
    printf "Hello! i\n";
    if (SomeStringList[i] == "Doh!") {Exit;}
}
```

or

```
for (i = 99, 0, i--)
{
    printf "Hello! i\n";
    if (SomeStringList[i] == "Doh!") {Exit;}
}
```

In this example, you can additionally see what the "i" var is useful for. We use the current value of "i" to specify an indexed value in a string list ("StringList[5]" is the sixth string in such a list, starting from index 0) and check if that value is "Doh!". If so, the loop is aborted by calling "Exit". Thus, we may leave the loop although it has not passed a hundred times.

The first example (using "to" or "i++") loops low-high - means it starts with zero and counts up to 99 while the second example ("downto" or "i--") loops high-low; starting with 99 and decrementing until reaching zero.

3.4 some others

> top of page

Some languages such as COBOL (ugh!) and ABAP/4 know even simplier kinds of fixed-pass loops that do not offer something like an i param to query (although ABAP has the global sy-index, okay...). A block is exec'd dumb fifteen times:

```
perform 15 times.
    write: / 'I am line', sy-index, '!' no-gap.
endperform.
```

ABAP also can auto-set the number of iterations required for a loop. For example, when using internal tables (imagine them as in-mem database tables or as an array of records / structures):

```
loop at some_itab.
    if some_itab-otype eq 'P'.
        write: / some_itab-stext.
    endif.
endloop.
```

This loop examines the internal table some_itab and does the loop as long as the table has rows. Very useful if you're lazy :-).

4.0 and what does all that mean?

> top of page

First of all, you have to decide what loop kind will solve your problem best. Do I know exactly how many times I need to iterate? Will it be under almost any circumstances be xy times? Or do I have to run that loop unknown times, depending on special conditions? Must that loop perform at least once to accomplish my goal?
> Well, here some practical examples. At the end we will regard some performance aspects. <p>

4.1 Example #1

> top of page

Imagine, you have created a simple console app that should list the files of your root directory and print'em onto screen. A code segment might look like below:

```
 function TMyApp.ReadRoot(): boolean;
var FileList:   TStringList;
    iCnt:       integer;
begin
    Result:= false;
    FileList:= TStringList.Create();
    MyUtils.FileSeek('/usr/local/src/*', FileList);
    writeln("Contents of source directory:" + ^j);
    iCnt:= 0;
    while iCnt < Length(FileList) do begin
        writeln(FileList[iCnt]);
        Inc(iCnt);
    end;
    FileList.Free();
    Result:= true;
end;
```

This routine defines two variables: an integer used as a simple counter ("iCnt") and a string list that will hold the file names. A string list is nothing more than a dynamic array of strings (or more precisely an array of an array of characters). After having created the string list object "FileList" with its base class constructor (it just allocates memory and inits values and methods), we call a custom method that reads a given path and returns all file entries of that directory in a string list. Note that our function "FileSeek()" has been invented for this demonstration.
> For simplicity we do no error checking, we assume that "FileList" holds a list of strings containing each file. What we want to do is to write those file names out to a console box. To accomplish dis, we initialize our counter "iCnt" to a value of zero.

>For each file in the list, we want to do a call to the "write()" procedure that simply sends a string to the standard output (STDOUT), thus we can see it in the console box. We run a loop that performs as much times as we have items in the list.

But how to get that number? Well, our "FileList" has been filled and by calling the "Length()" method, we can get the number of strings in that list. E.g. when having 3 files in FileList, "Length(FileList)" will return 3 - quite simple. Having set "iCnt" to zero, we just have to run the loop until "iCnt" has reached the value of "Length()" when increasing "iCnt" by the value of 1 each loop iteration.
> Cos a while loop checks it’s condition prior running its dedicated block, the loop exits if "iCnt" has a value "Length(FileList) - 1" (cos counting starts from zero and not from plus one). <p>

4.2 Example #2 (or: slightly more perofrmance)

> top of page

Let us take a closer look to our loop from the previous example:

```
while iCnt < Length(FileList) do begin
    writeln(FileList[iCnt]);
    Inc(iCnt);
end;
```

We have a leak of performance there: each time the loop is performed, we check if "iCnt" is smaller than the amount of items in the string list obtained by "Length()". The problem is that the "Length()" routine is also executed each iteration. Depending on the size of the string list, this could take some time and might cause some heavy system load if the result value is not cached somehow by the system.
> Cos the value of "Length(FileList)" will not change within loop’s execution, we will swap that value into a temporary integer variable: <p>

```
var iCnt,
    iTmp: integer;
...
    iTmp:= Length(FileList);
    while iCnt < iTmp do begin
        ...
    end;
```

We only waste four bytes more to buffer the list’s size into a temporary variable (integer = 32bit signed = four bytes) - but we gain much performance cos "Length()" is only called once. But there is just another leak: we manually increase our counter "iCnt" by the value of one. This is not neccessary, cos we know the string list’s size and just need an actual iteration counter to specify the actual index in the string list.
> Thus, why using the while loop at all? There is a better solution for this case, as we learned from above: <p>

```
for iCnt:= 0 to Length(FileList) do begin
    writeln(FileList[iCnt]);
end;
```

Our advantage: we saved to use a temporary buffer integer and have no need to manually shift our counter. As you see, the code has been shortened a bit. Think of larger projects - this can safe valuable space in your editor :-).

But at all: we improoved performance, cos a "for...to" loop does an internal increment for the counter and is thus faster. Also, the internal checking if the counter is less than "Length()" is quicker. It merely executes xy times instead of doing a more complex and variant condition check (variant? well, the "for...to" loop does only work with numbers while "while" really might check any condition, not only if a number has a special value).

To be continued...

> top of page

Copyleft (C)2000-2001 by Blazko. This document is licensed under the terms of the GPL.