Infinite loop syntax support for a Pascal-family language

http://coderesearchlabs.com/articles/ILP.pdf

Code Research Laboratories
www.coderesearchlabs.com

Javier Santo Domingo
j-a-s-d@coderesearchlabs.com

Buenos Aires, Argentina

started - April 9th, 2010
published - July 9th, 2010
last revision - May 4th, 2011

1. **Introduction**

Infinite loops are a basic logic flow need that is supported by conventional processors and virtual machines (see table A), but there is always a valid solution to avoid them in High Level Languages. In any case, the discussion if it is or not a good practice will never end. There is no code (operative system, service daemon, command line, real-time system, message loop, etc) that can not avoid an infinite loop, that's a fact: you can always write loops with an exit condition.

Besides that, and nowadays that code readability is very important (and that *goto* has been totally removed from good practices), there are many developers that prefer to implement an infinite loop and break it's execution in the middle [1], or that coding a program "designed to never stop" come to solutions with an infinite loop.

x86's assembly	ARM's assembly	.NET's ilasm	Java's bytecode	Llvm's assembly

label:
  ; something
  jmp label
	label
	  ; something
	  B label
		label:
		  // something
		  br label
			label:
			  ; something
			  goto label
				Loop:
				  ; something
				  br label %Loop

Infinite Loops (unconditional jumps/branches) in Some Low/Mid Level Of Abstraction Languages
table A

2. **Different Kind of Loops**

Usually, how you order the logic for the solution you are writing determines where you will be testing the exit condition for a loop: at the top or at the bottom [2]. If you need to loop zero-or-many times, then you should test at the top, and if you have to loop one-time-at-least, then you should test it at the bottom [3]. In any case, you order your logic based on what is the "common case" that the loop will face. In this context is where infinite loops appear.

3. **Infinite Loop Convenience**

As expressed before, the source code readability is one of the keys of good development today. This is specially true today where it is very common to mantain source code that "someone else" wrote. Given that situation, the more accurate or specific the language is, the more easier to understand the code is, and the bugs to find are. And to be both things, a language must be richer enough to offer logic structures that let you express exactly what you mean.

In this field, an infinite loop indicates the code inside of it has a very specific behaviour. So, a language that wants to satisfy that programming need, should give you the right tool to craft your special solution in a clear way.

In languages lacking of the syntactic support for this, coders usually refer to while-true-do or repeat-until-false or other kind of loops with impossible exit conditions like the 0=1 comparison [4] or even placing no condition into structures designed to have one (eg. "for(;;)").

Just to have a sneak preview, search the internet for "while true do" to see how common it is. In any case, there exist also many well known languages that already offer this feature (see table B), indicating how important it is.

Basic	Perl	Rexx
DO/LOOP [5]	LOOP { } [6]	DO FOREVER/END [7]

non Pascal-family well-known High Level Languages with Infinite Loop support
table B

4. **The Suitable Pascal Loop**

In the Pascal language family, where infinite loops are implemented too [8][9], you have many ways to iterate.

- FOR-TO-DO (which is a ↻ loop associated to a Σ summatory)
- FOR-IN-DO (supported by Delphi, FreePascal, etc)
- WHILE-DO

But there is only one that has the exit condition at the end:

- REPEAT/UNTIL

It is fair to say that not only Pascal supports the REPEAT keyword: there are many others like, for example, Icon, Logo and Lua.

So, if you want to apply an infinite loop solution for a language that lacks of it, and has many ways to iterate as is this the case, you should see first if you can convert the logic structure that checks the condition ot the end, just to follow the common nature of reasoning, that means converting the one-time-at-least loop that the language has [10] (since if you have a chance of not looping at all, you should check it before entering the "end-less" loop).

5. **Possible Solutions**

Besides one just can incorporate the other Pascal-family LOOP/END structure like those from Oberon [11], Zonnon [12], Modula-3 [13] and/or Ada [14] (a form to express this structure that is also present in other non-pascal-family languages like Shater [15]), what I propose to satisfy the need of an infinite loop is to replace the REPEAT's UNTIL part with something that expresses that the loop has no ending condition. The reason for this is that as an infinite loop is a one-or-more-time loop by definition, when the pascal programmer is reasoning and writing the code, he comes with the idea to write a REPEAT block (he should discard automatically the other loop constructions since there is not a predefined amount of times to iterate and there is no condition at the beginning).

I conclude that is more natural to associate the infinite loop to the REPEAT keyword than to create a new keyword for that like LOOP (even with the "begin/end" pair as also supported by Oxygene [16]). There are a couple of options available on this line:

A) We can popularize just the same solution used in Pascal-FC [17] (also in BlitzMax [18]):

```
REPEAT
        (* something *)
FOREVER;
```

B) Or move forever as a modifier of repeat (in this case will look more like Obix [19], some versions of HyperTalk [20] or even MultiLogo [21]):

```
REPEAT FOREVER
        (* something *)
END;
```

C) Reuse the well-known "begin/end" block which can not be omitted if is one line long (in this case the syntax would not compatible with Spin [22] but at the same time will remind to Verilog's "forever begin/end" block [23]):

```
REPEAT BEGIN
        (* something *)
END;
```

D) Or we can use an "end" keyword to close the block.

```
REPEAT
        (* somethig *)
END;
```

E) Or may be adopt some other solution:

```
REPEAT
        (* something *)
AGAIN;

REPEAT
        (* something *)
LOOP;

REPEAT LOOP
        (* something *)
END;

REPEAT LOOP
        (* something *)
AGAIN;

REPEAT LOOP
        (* something *)
FOREVER;
```

6. **Conclusion**

Professor Wirth, on his talk at Google's TechTalks of the past year (GTAC 2009) [24], insisted very much in that "programming languages must be rigorously defined" and that "programming languages are for human consumption". By that time I already had in mind all this expressed before but it was pretty interesting to see how that words of the father of the Pascal language fit very well in the intention of this solution I'm proposing.

Finally, I have implemented the REPEAT/FOREVER (A) solution on this topic for the MIDletPascal (3.1 ALPHA and superior) project [25], so if you want to check out the implementation just browse the "MPC.3.1.IDE" compiler code, and/or more important yet: try the REPEAT/FOREVER structure in some MIDletPascal code you may write.

References

[1] http://stackoverflow.com/questions/224204/why-use-infinite-loops/224215#224215
[2] http://stackoverflow.com/questions/224138/infinite-loops-top-or-bottom
[3] http://stackoverflow.com/questions/224059/do-your-loops-test-at-the-top-or-bottom/224075#224075
[4] http://www.mikroe.com/forum/viewtopic.php?p=6352&sid=f6888409a78473b33a5f6299335609fb
[5] http://www.oopic.com/do.htm
[6] http://perl6.wikia.com/wiki/Loop
[7] http://publib.boulder.ibm.com/infocenter/iadthelp/v6r0/index.jsp?topic=/com.ibm.etools.iseries.orxw.doc/orxw_prg35.htm
[8] http://wiki.freepascal.org/Networking/es
[9] http://www.gnu-pascal.de/h-gpcs-en.html
[10] The Programming Language Pascal #9.2.3.2 "Repeat Statements" (1973, Prof. Wirth), Pascal ISO 7185:1990 #6.8.3.7 "Repeat-statements", etc
[11] http://www.mathematik.uni-ulm.de/oberon/reports/report-1992.html
[12] http://www.zonnon.ethz.ch/archive/znnLanguageReportv04y090606draft.pdf, 7.8 “The loop Statement”
[13] http://www.opencm3.net/doc/tutorial/m3/m3_38.html#SEC38
[14] http://en.wikibooks.org/wiki/Ada_Programming/Control#Endless_Loop
[15] http://www.icsi.berkeley.edu/~sather/Publications/article.html
[16] http://en.wikipedia.org/wiki/Oxygene_%28programming_language%29
[17] http://www-users.cs.york.ac.uk/~burns/ex/sem.html, created by Alan Burns and Geoff Davies in 1990 enhancing the Simplified Pascal (Pascal-S, 1976) of the Proffesor Wirth with several types of concurrency.
[18] http://en.wikibooks.org/wiki/BlitzMax/Language/Program_Flow, created by Blitz Research Limited.
[19] http://www.obix.lu/docs/manuals/bk01pt02ch08s04s05.html
[20] http://www.jaedworks.com/hypercard/scripts/hypertalk-bnf.html and http://infomotions.com/musings/tcp-communications/
[21] http://llk.media.mit.edu/papers/MultiLogo.html, created by Mitchel Resnick in 1990.
[22] http://www.instructables.com/id/Programming-the-Propeller-Microcontroller/step3/Spin-Basics/
[23] http://www.asic-world.com/verilog/verifaq2.html
[24] http://www.youtube.com/watch?v=8W5Jd_wzB90, Professor Niklaus Wirth at the 4th Annual Google Test Automation Conference, October 21st-22nd, 2009, Zurich, CH
[25] http://sourceforge.net/projects/midletpascal/