Loops and their control variables

Discussion and proposed guidelines

Derek M. Jones
derek@knosof.co.uk

1 Introduction

The implementation of many algorithms requires some set of operations to be performed a specified number of times and also a way of maintaining and using a counter for each iteration. To support this commonly occurring requirement support for a special kind of loop, known as a *for loop*, has been included in many programming languages. The variable used to maintain the iteration count is also used to control the number of iterations (or, from another point of view when it terminates) and is often given the name *loop control variable*.

The loop control variable is often given a special status by coding guidelines, a practice that partly derives from the behavior specified in some or the early, and widely used, programming languages (e.g., Fortran prior to the 1977 standard). It was often the case that modifying the value of a loop control variable inside the body of a loop resulted in undefined behavior and accessing the value of a loop control variable outside of the loop was not guaranteed to return a particular value.

This proposal discuses for loop usage and possible recommendations that restrict the use of loop control variables for reasons based on programming language semantics and/or developer expectations.

The material in this proposal has been distilled from that appearing in sentence 1760 of "The New C Standard: An economic and cultural commentary" by Derek M. Jones (available via http://www.knosof.co.uk/cbook/cbook.html).

2 for loop constructs

The functionality of loop statements in different languages varies from the spartan to the everything plus kitchen sink.

Ada 83 requires the lower and upper bounds to be known at compile time and the loop header acts as the definition of the variable appearing in it (this requirement was relaxed in Ada 95).

```
1  for v in 1 .. 10 loop  -- declare v to have the discrete range 1 to 10
```

Pascal (and Fortran since 1977) allows the value of the increment to be specified, which along with the start and end value may be an expression calculated at runtime (this calculation occurs prior to every first iteration of the loop). The loop control variable must have previously been declared.

```
1  for v:=start_var to end_var step inc_var do  (* Extended Pascal *)
```

Support for a comma operator in C, C++, C#, and Java makes it possible for a loop header to contain what is effectively more than one independent expression. Both the termination *increment* expression and end expression are evaluated on every iteration (ie, not just prior to the first iteration).

```
1  for (lcv_1=0, lcv_2=0; a1[lcv_1] < a2[lcv_2]; lcv_1++, lcv_2++) // C, C++, C#, Java
```

The common theme running through all of these for loop headers is the idea of a (loop control) variable that is given an initial value which is modified by a constant amount until it reaches a value that terminates the loop iteration.

In nearly all cases all modifications to this loop control variable occur in expressions that appear in the loop header.

2.1 Developer usage

Developers may chose to use one kind of loop rather than another purely out of force of habit. If enough developers follow the same usage patterns then these become idiomatic and are viewed as being part of the culture of that language's usage. Unless it can be shown that the cost of using a particular idiom is greater than the benefit, there is nothing to be gained from trying to change existing, common, usage idioms.

If there is an idiom that is commonly used, is it worthwhile requiring that all developers use it? What if there are two idioms for the achieving the same effect (e.g., in C, C++ and Java, both `for (; ;)` and `while (1)`

implement a so called infinite loop and are commonly seen in code (based on measurements of C source, 2.5% of for loops and 6.2% of while loops), should the more common one be recommended over the other???

Developers who regularly use one language may regard the different ways in which loops are used in other languages as being inappropriate. For instance, Ada developers might argue that the following loop should be replaced by a while loop because two different variables are involved both in the termination condition and being incremented on each iteration.

```
1   for (lcv_1=0, lcv_2=0; a1[lcv_1] < a2[lcv_2]; lcv_1++, lcv_2+=2) // C, C++, C#, Java
```

However, placing them in a for loop header has the benefit of enabling the reader to easily locate all of the information involved in controlling the loop iteration.

The purpose of these guidelines is not to redesign a language to make it fit the design model of another language.

2.2 Which variable is the loop control variable?

The specification of some languages makes it easy to deduce which variable should be classified as the loop control variable. For instance, Ada defines something called a *loop parameter*, while Pascal actually calls the likely candidate a *control-variable*.

The definition of other languages (e.g., C, C++, Java) do not include the specification of any variable that might be suggest it is a candidate for the status of loop control variable. Also these languages have very generalised looping constructs which can make it difficult to exactly specify which, if any, variable should be considered to be a loop control variable (the first edition of the C Standard actually defined the **`for`** statement in terms of the **`while`** statement).

2.3 C, C++, C# and Java

These, and other, languages support a for loop construct that is essentially a while loop with the three loop components (initial code, termination test, and increment) appearing between a matching pair of parenthesis. Because expressions that can occur in a for loop are not as restrictive as they are in many other languages, it can be difficult to identify which variable, if any, plays the role of the loop control variable. In the following example:

```
1   for (lcv_1=0; lcv_2 < lcv_3; lcv_4++)
```

any one of four different variables has some claim to being the loop control variable. It is also possible to claim that there is more than one loop control variable. In the following example:

```
1   for (f_1(); /* empty */ ; f_2())
```

there are no obvious candidates for the status of loop control variable.

Table .1: Occurrence of various kinds of **for** statement controlling expressions (as a percentage of all such expressions), based on the visible form of the `.c` files. Where *object* is a reference to a single object, which may be an identifier, a member (e.g., `s.m`, `s->m->n`, or `a[expr]`); *assignment* is an assignment expression, *integer-constant* is an integer constant expression, and *expression* represents all other expressions.

Abstract Form of **for** loop header	%
assignment ; object < object ; object **v++**	33.2
assignment ; object < *integer-constant* ; object **v++**	11.3
assignment ; object ; assignment	7.0
assignment ; object < expression ; object **v++**	3.3
assignment ; object < object ; **++v** object	2.7
; ;	2.5
assignment ; object != object ; assignment	2.5
assignment ; object <= object ; object **v++**	2.2
assignment ; object >= *integer-constant* ; object **v--**	1.6
assignment ; object < function-call ; object **v++**	1.4
assignment ; object < object ; object **v++** , object **v++**	1.4
others	31.1

2.3.1 Deducing the loop control variable

Given a loop header the following algorithm frequently returns an answer that has been found to be acceptable to developers. Note that the algorithm may return zero, or multiple answers; a union or structure member selection operator and its two operands is treated as a single object, but both an array and any objects in its subscript are treated as separate objects and therefore possible loop control variables:

1. list all objects appearing in *expression-2* (the controlling expression). If this contains a single object, it is the loop control variable (33.2% of cases in the `.c` files),
2. remove all objects that do not appear in *expression-3* (which is evaluated on every loop iteration). If a single object remains, that is the loop control variable (91.8% of cases in the `.c` files),
3. remove all objects that do not appear in *clause-1* (which is only evaluated once, prior to loop iteration). If a single object remains, that is the loop control variable (86.2% of cases in the `.c` files).

Unlike the example given above, in practice the same object often appears as an operand somewhere within all three components (see
Figure .1 was obtained by measuring the visible form of a large amount of C source code. It plots the number of possible loop control variables appearing in *expression-2* (square-box) after filtering against the objects in *expression-3* (cross) and after filtering against the objects in *clause-1*.

3 Possible recommendations

Experience shows that developers often assume that, in a **for** statement, modification of a loop control variables only occurs within the loop header (i.e., not the loop body). This leads to them forming beliefs about properties of the loop, for instance, *it loops 10 times*. There tend to be fewer assumptions made about the use of **while** statements (which might not even be thought to have a loop control variable associated with them) and the following guideline is likely to cause developers to use this form of looping construct.

Cg .1 A loop control variable shall not be modified during the execution of the body of a **for** statement.

Some coding guideline documents recommend that loop control variables not have floating-point type. It might be thought that such a recommendation only makes sense in languages where the loop termination condition involves an equality test. However, in languages such as C, C++, and Java the controlling expression of a for loop can contain relational operators, which can also have a dependence on the accuracy of floating-point operations. For instance, it is likely that the author of the following fragment expects the loop to

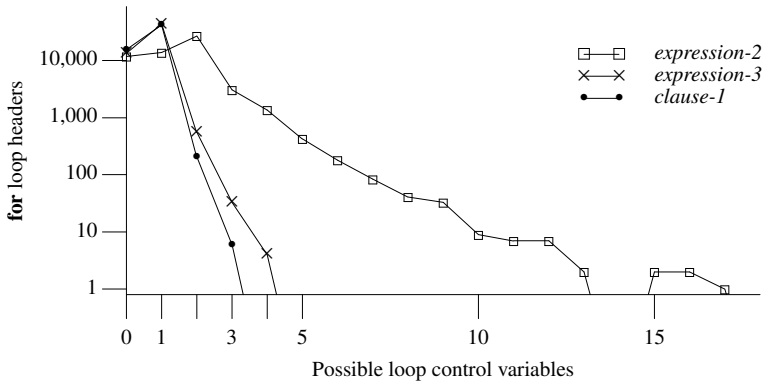

Figure .1: Number of possible loop control variables appearing in *expression-2* (square-box) after filtering against the objects in *expression-3* (cross) and after filtering against the objects in *clause-1* (bullet). Based on the visible form of the `.c` files.

iterate 10 times. However, it is possible that 10 increments of `i` result in it having the value 9.9999, and loop termination not occurring until after the eleventh iteration.

```
1   for (float i=0.0; i < 10.0; i++)
```

A possible developer response to a guideline recommendation that loop control variables not have floating point type is to use a while loop (which are not covered by the algorithm for deducing loop control variables). Some of the issues associated with the finite accuracy of operations on floating-point values can be addressed with guideline recommendations. However, the difficulty of creating wording for a recommendation dealing with the use of floating-point values to control the number of loop iterations is such that none is attempted here. Future work....

Should there be exactly one loop control variable per **`for`** statement?

- Why has the developer use a for loop without a loop control variable when a while loop might be more applicable?
- Multiple loop control variables. The following example may look complicated, but is it more complicated than all of the alternative ways of expressing this loop? There does not appear to be any cost/benefit reason to limit loops to only having one loop control variable.

```
1   for (lcv_1=0, lcv_2=0; a1[lcv_1] < a2[lcv_2]; lcv_1++, lcv_2++) // C, C++, C#, Java
```

In those languages where modifying a loop control variable in the body of the loop results in undefined behavior a recommendation needs to be specified.

LangSpec .2 In those languages where modifying a loop control variable in the body of the loop results in undefined behavior a loop control variable shall not be modified in the body of the loop.

In those languages where accessing a loop control variable outside of the body of the loop returns an undefined value a recommendation needs to be specified.

LangSpec .3 In those languages where accessing a loop control variable outside of the body of the loop returns an undefined value, the value of a loop control variable shall not be accessed outside of the loop body.

References

Citations added in version 1.0b start at 1449.