Continuations

Alan Nall

IU, Bloomington, IN 1983

I first encountered the continuation in a course called C511 - Advanced Programming Concepts at Indiana University . The concept of the continuation was one of the basic concepts being taught in that course. It was also a concept that grabbed my mind, ran off with it, and only returned it after substantial renovation and expansion. I spent perhaps an overlong time with the continuation, thinking about its implications and the ways in which one might explain it to someone else. Many times I awoke in the dead of the night with the 'light bulb' of a new idea blinking above my head. It was an interesting and fun process, but it played merry hell with my sleep, study and eating habits.

The process did one thing more; it created this paper. After I stumbled through a seemingly unending series of mires and pitfalls associated with the continuation, I found myself wanting to share what I had learned. I wanted to help others avoid some of the disruption of life I had experienced in trying to comprehend the continuation. I wanted to provide people with a little help so they could more quickly arrive at the point where they could use the continuation in their programming, and thus be able to spend their late nights playing with the continuation instead of staring at it.

The first words I ever heard describing a continuation were these: "the continuation is the rest of the program." This statement is accurate but incomplete. It requires knowledge of continuations in order to be comprehensible. It is, however, tantalilzing. Spend a few moments thinking about this statement now; we'll come back to it later. For now let's consider the basic ideas about continuations needed to make the above statement complete.

First, in order to make sense, a continuation has to have an object. A continuation is always the continuation of something. Whether that something is a trip, a concert, a program or a book, we always speak about the continuation of the book, trip, concert or program. In the statment above, the object would be the program mentioned.

Second, a continuation has to have an instance. This requirement is perhaps not as obvious as the last. But consider. If I am on vacation, driving in my car, the rest of my trip is different at 4:00 p.m. than it is at 8:00 p.m. (especially if, for example, I arrive at my destination at 6:00 p.m.). So in order to specify a particular continuation, we need to specify not only what it is of but where it is at (where the at could be time or space).

With these additions the concept of a continuation becomes a little clearer. When I refer to the continuation of a trip at 4:00 p.m. Dec. 17, 1983, I am referring to all of the trip that follows that point. If I refer to the continuation of the book at page 16, line 4, I am referring to all of the book following that point. If I refer to the continuation of a program at point A, I am referring to all of the program following point A - the rest of the program at point A.

Now that we have made this term more precise, how do we make it useful? A continuation now describes something very specific and when something can be described, it is not too much of a step to store and work with it. Let's start with something fanciful in the real world. Let's say you have this magic lamp which, when rubbed, picks up the continuation of your life at that point. Further, this lamp would give you a trigger word to invoke that continuation and return to that point in your life. Your life would then go on as it had until you spoke one of these keywords. If you never spoke a keyword, your life would be ordinary, free of skipping around.

Before we get into the uses of this lamp, we need to specify one thing - just how specific is a continuation? We say that the continuation of your life at a given point is all of your life that follows that point. The question is how set in iron is this? If I invoke this continuation, I return to this point, if everything were the same as it had been, including me, I won't know I invoked the continuation. I will go on with my life exactly as I did before (making all the same decisions) eventually invoking the continuation and coming back to do it all again. This would not be very useful, either in life or in programming. (Infinite loops are rarely fun.)

How do we make this useful? One way would be to have the lamp capable of sending back a message when the continuation is invoked. Now information can pass from where a continuation is invoked to where it was created, and be acted upon by the person who caused its creation. They would not know all of their life that will transpire, but they will have a little more information.

This is useful. Imagine yourself, as your car is sailing off the bridge, invoking a continuation with the message, "learn to drive on ice". Or, perhaps you might, after a particularly boring date, invoke the continuation just before you called the person, with the message, "Don't call X. It'll be a boring date." How many times have you said, "If only I knew then what I know now" or "If I had it to do over, I'd do X." With continuations with messages, you can do it over and with a careful enough message, you could know then what you know now.

Stop for a few moments now and create a few useful continuations and their invocations. Write down when you would create the continuation, what the situation is when you invoke it, and what message you pass through it. Do this a few times, then put the pen down, put your hands behind your head and stare at the ceiling for five minutes.

Done staring? Mind feel a little stretched? Well, consider the possibilities when we allow the trigger word for a continuation to be part of the message sent through a continuation. In other words, you can tell the receiver to do something, then invoke continuation X and tell you what you wanted to know. Spend about 10 minutes on that one - it has some really interesting twists.

Fortunately, or unfortunately depending upon how you feel about hindsight and the threading of time, continuations do not exist in the real world of human lives. They do, however, exist in some programming languages and they provide those programming languages with an extremely powerful tool. Since I encountered continuations in the language, Scheme, and am most familiar with them in that language, we will discuss continuations in languages within the context of Scheme.

This paper is going to assume from here on that the reader has a working knowledge of Scheme. If this is not the case, the reader may still be all right if (s)he has a good grasp of functional languages (a strong knowledge of LISP might be sufficient) and is willing to muddle a bit.

Scheme implements continuations as a function, CALL-WITH-CURRENT-CONTINUATION, for which a macro call/CC is generally written. This macro is a function of one argument of the form (call/CC (lambda (c) <body>)) if there is no call to c in <body> the function terminates normally with its expected value. However, if there is a call to c in <body>, then the function terminates immediately with the single argument to the invocation of c being the value of the function. This is accomplished by binding c to the continuation of the program at the point of the function call (everything that could follow the (call/CC (lambda (c) <body>)) statement) thus when we invoke c, we are invoking the rest of the program. Note that part of the continuation is the environment at the point at which the continuation was created so that any changes made to the environment by the <body> will be lost when the continuation is invoked. (There is a way they might be recovered. Can you think what it is?) The only communation back from the body is the one argument given to the invocation of the continuation.

Hopefully, all of the above did not cause too much confusion. If it did, two courses of action are open, muddling on through the example that follows or rereading what has come before. (I would do the latter only once before I did the former.) You might also stare at the ceiling for a while, go do something else or read a book on Scheme. Any or all of these may help. Don't worry if things look bleak, continuations, like many things in functional programming, tend to work on the lightning principle - when things are darkest - BANG - enlightenment hits.

The example of programming with continuations which takes up most of the rest of this paper is taken from assignments given in C511 when I took that course. In solving the problems originally, many mistakes were made, a few blind alleys were followed and both the code and my understanding evolved a great deal. Here only a subset of that process will be shown. Only major changes in code or concept are mentioned (I have left out, for instance, the week I spent trying to solve the problem generated when I typed car instead of cdr at one point). Hopefully, that which remains will be sufficient to impart to you the major steps my thinking took in solving the problems, as well as do a little renovation and expansion on your thinking process and apparatus.