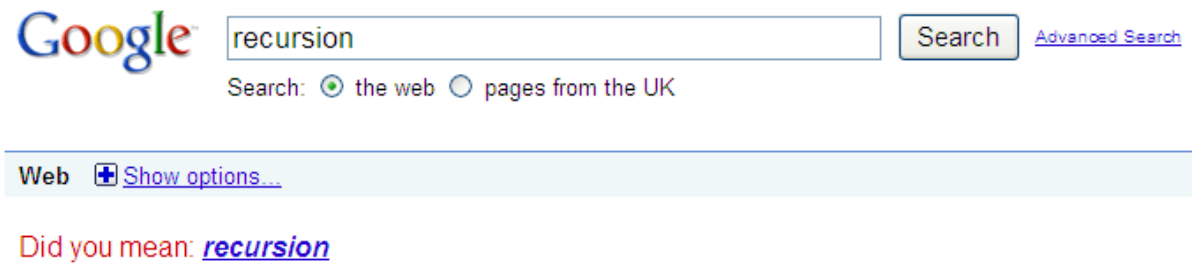
Recursion

Let us begin with an old joke about *recursion*[1]. The clue is in that the footnote points back to this document. This joke was originally included in *The C Programming Language* by Kernighan and Ritchie, wherein on one of the index pages – page 269 in this version – the last page reference for "recursion" is also page 269.

THE C PROGRAMMING LANGUAGE INDEX 269

[1] Recursion

The same joke appears today; when we Google "recursion" we get the following:



But the aforementioned "jokes" are not just jokes, they also communicate the meaning of recursion in a very simple way; a recursive function in one which calls itself. On the face of it this doesn't seem to be helpful; in fact recursion is one of the most useful facilities in algorithm[2] design and most powerful construct of computer programming.

Example 1: hello world

To get started we will look at an example which is written in a java-like or c-like code:

```
void hello()
{
System.out.println("hello world")
hello()
}
```

If this program is run then "hello" would be printed out continuously.

As illustrated by the example above, in practice we find that a recursive function should have a stopping condition (if the program is required to complete a process).

Effectively, if a problem can be reduced to a smaller (or simpler) problem(s) of the same type, and this is carried out repetitively then this eventually results in problems that are straightforward, and hence the whole problem can be solved. Such problems are ideal for solution by a recursive algorithm.

In the following example, the factorial[3] of a number is determined using a recursive method. An implementation of this is given in the file Factorial.java. An implementation of this is given in the file Fibonacci.java.

[2] Algorithms
[3] Factorial

Example 2: factorial function

The code is as follows:

```
int factorial(int n)
{
    if (n==1) return(1);
    else return (n*factorial(n-1));
}
```

The method uses the useful relationship:

$$n! = n(n-1)!\,;$$

the factorial is written in terms of the factorial of a smaller number. And the stopping condition $1! = 1$.

As a third example, we consider a recursive algorithm for computing a term of the Fibonacci sequence[4].

Example 3: Fibonacci sequence

The Fibonacci sequence is as follows

1, 1, 2, 3, 5, 8, 13, 21, ...;

It starts 1, 1 then each of the following terms is determined as the sum of the previous two terms. The code is as follows:

```
static int fibonacci(int n)
{
    if (n<=2) return(1);
    else return (fibonacci(n-1)+fibonacci(n-2));
}
```

The method uses the recurrence relationship:

$$f_n = f_{n-1} + f_{n-2}\,;$$

the Fibonacci number of index *n* is written in terms of two earlier Fibonacci terms.

[4] Sequences