

Recursion

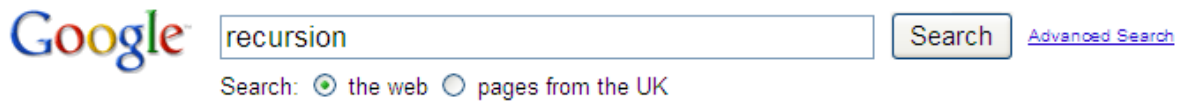


Let us begin with an old joke about *recursion*¹. The clue is in that the footnote points back to this document. This joke was originally included in *The C Programming Language* by Kernighan and Ritchie, wherein on one of the index pages – page 269 in this version – the last page reference for “recursion”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

| <i>THE C PROGRAMMING LANGUAGE</i> | INDEX 269 |
|---|--|
| pointer initialization 102, 138 | ptrdiff_t typedef 103, 147, 206 |
| pointer, null 102, 198 | push function 77 |
| pointer subtraction 103, 138, 198 | pushback, input 78 |
| pointer to function 118, 147, 201 | putc library function 161, 247 |
| pointer to structure 136 | putc macro 176 |
| pointer, void * 93, 103, 120, 199 | putchar library function 15, 152, 161, 247 |
| pointer vs. array 97, 99-100, 104, 113 | puts library function 164, 247 |
| pointer-integer conversion 198-199, 205 | |
| pointers and subscripts 97, 99, 217 | qsort function 87, 110, 120 |
| pointers, array of 107 | qsort library function 253 |
| pointers, operations permitted on 103 | qualifier, type 208, 211 |
| Polish notation 74 | quicksort 87, 110 |
| pop function 77 | quote character, 19, 37-38, 193 |
| portability 3, 37, 43, 49, 147, 151, 153, 185 | quote character, 8, 20, 38, 194 |
| position of braces 10 | |
| postfix ++ and -- 46, 105 | \r carriage return character 38, 193 |
| pow library function 24, 251 | raise library function 255 |
| power function 25, 27 | |
| #pragma 233 | rand function 46 |
| precedence of operators 17, 52, 95, 131-132, | rand library function 252 |
| 200 | RAND_MAX 252 |
| prefix ++ and -- 46, 106 | read system call 170 |
| preprocessor, macro 88, 228-233 | readdir function 184 |
| preprocessor name, __FILE__ 254 | readlines function 109 |
| preprocessor name, __LINE__ 254 | realloc library function 252 |
| preprocessor names, predefined 233 | recursion 86, 139, 141, 182, 202, 269 |
| preprocessor operator, # 90, 230 | recursive-descent parser 123 |
| preprocessor operator, * 90, 230 | redirection see input/output redirection |
| preprocessor operator, defined 91, 232 | register address of 210 |
| primary expression 200 | |

¹ [Recursion](#)

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



Web [Show options...](#)

Did you mean: [recursion](#)

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 algorithm² 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 factorial³ of a number is determined using a recursive method.

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 $1! = 1$.

An implementation of this is given in the file [Factorial.java](#).

As a third example, we consider a recursive algorithm for computing a term of the Fibonacci sequence⁴.

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.

An implementation of this is given in the file [Fibonacci.java](#).

⁴ [Sequences](#)