Contents

Recursion

The code for this lecture is available in this archive[1] (first parts) and this one[2] (listing files and folders recursively).

Introduction

Recursion is a central notion in programming, simple to state but difficult to master: a method is *recursive* if it calls itself. This concept is related to the idea of repetition, or looping, of program parts, and come with the same danger of not terminating. Below, we present some simple recursive programs: while some could be written without recursion, some would be very hard, if possible at all, to write without using recursion.

First Examples

Consider the following:

```
static void displayAll(int n)
{
  if (n > 0)
  {
    Console.Write($"{n} ");
    displayAll(n - 1);
  }
}
```

If we call `displayAll(3);`, then the following will happen:

1. `displayAll(3)` will test that `3>0`,
2. `displayAll(3)` will display "3 ",
3. `displayAll(3)` will call `displayAll(2)`,
   (a) `displayAll(2)` will test that `2>0`,

[1]https:/princomp.github.io/code/projects/RecursionHelloWorld.zip
[2]https:/princomp.github.io/code/projects/ListDirectoriesFileRecursively.zip

    (b) `displayAll(2)` will display "2 ",
    (c) `displayAll(2)` will call `displayAll(1)`,
        i. `displayAll(1)` will test that `1>0`,
        ii. `displayAll(1)` will display "1 ",
        iii. `displayAll(1)` will call `displayAll(0)`,
            A. `displayAll(0)` will test that `0>0`,
            B. `displayAll(0)` will terminate.
        iv. `displayAll(1)` will terminate.
    (d) `displayAll(2)` will terminate.
4. `displayAll(3)` will terminate.

Hence, `displayAll` calls itself with a smaller number, unless that number is 0, in which case it simply terminates. In our example, it would display "3 2 1 ".

When the function calls itself matters a lot. Indeed, consider `displayRAll`, which calls itself *before* executing the `Console.WriteLine` instruction:

```
static void displayRAll(int n)
{
  if (n > 0)
  {
    displayRAll(n - 1);
    Console.Write($"{n} ");
  }
}
```

If we call `displayRall(3);`, then the following will happen:

1. `displayRall(3)` will test that `3>0`,
2. `displayRall(3)` will call `displayRall(2)`,
    (a) `displayRall(2)` will test that `2>0`,
    (b) `displayRall(2)` will call `displayRall(1)`,
        i. `displayRall(1)` will test that `1>0`,
        ii. `displayRall(1)` will call `displayRall(0)`,
            A. `displayRall(0)` will test that `0>0`,
            B. `displayRall(0)` will terminate.
        iii. `displayRall(1)` will display "1 ",
        iv. `displayRall(1)` will terminate.
    (c) `displayRall(2)` will display "2 ",
    (d) `displayRall(2)` will terminate.
3. `displayRall(3)` will display "3 ",
4. `displayRall(3)` will terminate.

In this example, "1 2 3 " would be displayed: the order is reversed with respect to `displayAll`!

! Caution

Recursion can be very powerful and can very easily make your program crash or misbehave. To see it for yourself, *after saving all important documents*, replace - with + in the previous examples and run the programs again.

`displayAll` is an example of *tail recursion*: the recursive call is the **last** statement in the method. `displayRAll` is an example of *head recursion*: the recursive call is the **first** statement in the method. They are furthormore both examples of *linear recursion*, as they call themselves only once.

Recursive Methods Returning a Value

Recursive methods can also return a value, used by previous calls to compute some other value.

Multiplication

For example, consider that multiplication can be defined by addition: indeed, $x \times y$ is $y + y + y + \ldots + y$ where y is summed x times. Stated differently (read: recursively), $x \times y$ is $y + ((x - 1) \times y)$. We can implement such a program easily:

```
static int mult(int x, int y)
{
  if (x == 0)
  {
    return 0;
  }
  else if (x == 1)
  {
    return y;
  }
  else
  {
    return y + mult(x - 1, y);
  }
}
```

For example, `mult(2, 10)` tests that `2` is neither 0 nor 1, and adds 10 with the result of `mult(1, 10)`, which is 10 since the first argument is 1.

Observe that `mult(10000000, 0)` would call `mult` 10000001 times and add 0 to itself 10000001 times: this algorithm is not very efficient!

Factorial

The factorial of n is $n! = n \times (n-1) \times (n-2) \times (n-3) \times ... \times 1$. This function can easily be implemented using recursion:

```
  static int factorial(int n)
  {
    if (n == 0)
      return 1;
    else
      return (factorial(n - 1) * n);
  }
```

Note that this code actually compute e.g., $5! = 5 \times 4 \times 3 \times 2 \times 1 \times 1$ (with one superfluous $\times 1$): can you see why?

Listing Files and Directories – Recursively

While multiplication and factorial can be implemented without recursion, some structures makes it natural, or even required, to use recursion. Going through folders and files is an example of such situation.

```
using System;
using System.IO;

class Program
{
  static void Main()
  {
    // We first locate where we currently are.
    DirectoryInfo currentDir = new DirectoryInfo(
      Directory.GetCurrentDirectory()
    );
    Console.WriteLine("Starting from " + currentDir +
↪   ".");
    int count = 5;
    // We go up 5 folders or until we reach the
    // root folder, whichever comes first.
    while (currentDir.Parent != null && count > 0)
    {
      currentDir = currentDir.Parent;
      count--;
      Console.WriteLine("Going up to " + currentDir +
↪   ".");
    }
    Console.WriteLine(
      "Now listing files and folders from here:"
    );
    ListDir(currentDir.ToString());
  }

  // Code in part inspired from
  // https://stackoverflow.com/a/929277
  static void ListDir(string sourceDir)
  {
    try
    {
      Console.WriteLine(sourceDir);

      foreach (string file in
       ↪  Directory.GetFiles(sourceDir))
        Console.WriteLine(file);

      foreach (
        string directory in Directory.GetDirectories(
          sourceDir
        )
      )
        ListDir(directory);
    }
    catch (Exception e)
    {
      Console.WriteLine(e.Message);
    }
  }
}
```

Note that our previous examples were calling themselves only once per method call, but that `ListDir` calls itself as many times as there are folders in the folder currently examined.