Tutorial Link https://codequotient.com/tutorials/Recursion - An Introduction/5a014167cbb2fe34b7775018

TUTORIAL

Recursion - An Introduction

Chapter

Recursion is the process of defining something in terms of itself, and is sometimes called *circular definition*. Recursion, is defined as any solution technique in which large problems are solved by reducing them to smaller problems of the ***same form***. The “same form” is crucial to the definition, which otherwise describes the basic strategy of stepwise refinement. Both strategies involve decomposition. What makes recursion special is that the subproblems in a recursive solution have the same form as the original problem. As a problem-solving tool, recursion is so powerful that it at times seems almost magical. In addition, using recursion often makes it possible to write complex programs in simple and profoundly elegant ways.

To gain a better sense of what recursion is, let’s imagine top boss of a company tells top executives to collect feedback from everyone in the company. Then each of these executive gathers his/her direct reports and tells them to gather feedback from their direct reports. And they will call their direct reports to do the same so on till the last person having no direct supervision.People with no direct reports -- the leaf nodes in the tree -- give their feedback. The feedback travels back up the tree with each manager adding his/her own feedback. Eventually all the feedback makes it back up to the top boss.

Secondly, If you have to raise a fund for some political party (say BJP) during elections time. Then either you will give all the money which may or maynot be possible. As is often the case when you are faced with a task that exceeds your own capacity, the answer lies in delegating part of the work to others. So you may call other people in your circle to do the same for some smaller amounts. They will also do the same for more smaller amounts from their circles. And eventually, at last you may be able to collect the money in a recursive way.

To see a real example of recursion, search for the word “recursion” on www.google.com. At top of result it will ask you “Did you mean : recursion”. if you click on that link, Google will lend you on the same page back each time you click on it. This is an example of infinite recursion. Also many times we are in a room or lift with mirrors on side walls, there also the endless recursion can be seen.

Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to iteration). A common computer programming tactic is to divide a problem into sub - problems of the same type as the original, solve those sub - problems, and combine the results. This can be achieved by recursion.

A recursive function definition has

1. one or more base cases, meaning input(s) for which the function produces a result trivially (without recurring), and
2. one or more recursive cases, meaning input(s) for which the program recurs (calls itself)

For example, the factorial function can be defined recursively by the equations $0! = 1$ and, for all $n > 0$, $n! = n*(n − 1)!$. Neither equation by itself constitutes a complete definition; the first is the base case, and the second is the recursive case. Because the base case breaks the chain of recursion, it is sometimes also called the "terminating case".

The job of the recursive cases can be seen as breaking down complex inputs into simpler ones. In a properly designed recursive function, with each recursive call, the input problem must be simplified in such a way that eventually the base case must be reached. Because if the problem breakdown is such that base case is never hit, can cause an infinite loop.

In programming languages, if a function is calling itself, it is called recursive function(or circular functions). Recursive functions are very helpful in programming. They will make the program compact and easy. Also there are several problems which solving with recursion is easy than the iterative approach. Following is a general structure of recursion: -

```
int recursive_function()
{
    ... .. …
    if(some condition)
        return value;
    else
        recursive_function();
    ... .. ...
}

int main()
{
    ... .. ...
    recursive_function();
    ... .. ...
}
```

The following program will calculate the factorial of a number recursively: -

Recursion Implementation

Javascript

```
function fact_recursive(num){
    let result;
    if (num == 0)                  // Base case for
recursion.
        return 1;      // fact() return 1 if argument is
0
    else{
        result = num * fact_recursive(num-1);      //
Call recursively with lesser number.
        return result;
    }
}

```

C

```
int fact_recursive(int num)
{
  int result;
  if (num == 0)          // Base case for recursion.
    return 1;      // fact() return 1 if argument is 0
  else
  {
    result = num * fact_recursive(num-1);      // Call
recursively with lesser number.
    return result;
  }
}



```

Java

```
static int fact_recursive(int num)
{
  int result;
  if (num == 0)          // Base case for recursion.
    return 1;      // fact() return 1 if argument is 0
  else
  {
    result = num * fact_recursive(num-1);      // Call
recursively with lesser number.
    return result;
  }
}

```

Python 3

```
def fact_recursive(num):
  if (num == 0):                # Base case for recursion.
    return 1                    # fact() return 1 if argument is 0
  else:
    result = num * fact_recursive(num-1);      # Call recursively with lesser number.
    return result

```

```
int fact_recursive(int num)
{
  int result;
  if (num == 0)          // Base case for recursion.
    return 1;     // fact() return 1 if argument is 0
  else
  {
    result = num * fact_recursive(num-1);      // Call recursively with lesser number.
    return result;
  }
}
```

If you express the fund collecting example, described above in pseudocode, it has the following structure:

```
void CollectContributions(int money)
{
  if (money can be collected easily from one source)
  {
    Collect the money.
  }
  else
  {
    Find X people in circle.
    Tell each person to collect money/X rupees.
    Combine the money raised by all persons.
  }
}
```

The key in this pseudocode translation is the line saying

```
“Tell each person to collect money/X rupees.”
```

It is simply the original problem reproduced at a smaller scale. The basic character of the task — raise money — remains exactly the same; the only difference is that money has a smaller value. Moreover, because the problem is the same, you can solve it by calling the original function.

This structure provides a template for writing recursive functions and is therefore called the recursive paradigm. You can apply this technique to programming problems as long as they meet the following conditions:

1. You must be able to identify simple cases for which the answer is easily determined.

2. You must be able to identify a recursive decomposition that allows you to break any complex instance of the problem into simpler problems of the same form.

Video Explanation

<iframe width="560" height="315" src="https://www.youtube.com/embed/slRWgAZUjxs" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

codequotient.com

Tutorial by codequotient.com | All rights reserved, CodeQuotient 2020