FREE eBook

LEARNING Recursion

Free unaffiliated eBook created from
Stack Overflow contributors.

#recursion

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: recursion

It is an unofficial and free Recursion ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Recursion.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Recursion

Remarks

Recursion refers to something being defined in terms of itself. In the context of programming it is either the practice of using functions that call themselves or recursive types.

Recursive Functions

There are two parts to a recursive function:

- One or more base cases
- A recursive step

Because a recursive function calls itself the string of function calls could go on forever. In order to make sure the function terminates base cases must be included. These will tell the function when to return a value instead of performing another recursive call. Typically, recursive functions will check their inputs against the base case and return if the base case has been reached. If the base case has not been reached then the function will proceed to the recursive step.

The recursive step is when a function will call itself. Many recursive functions will return some calculation performed on the result of the recursive call. The Fibonacci sequence example demonstrates this. That function recursively calls itself twice and then returns the sum of the results of those calls

One common use for recursion is navigating and performing calculations on large data structures such as trees and graphs. This works well because recursion breaks the problem down to solve smaller problems and builds upon those solutions to solve the larger problem. Another common use for recursion is performing iterative operations (looping) in languages (typically functional languages) that have no built in looping structures.

Recursive Types

A recursive type is one whose values are composed from values of the same type, or phrased differently: a recursive type is defined in terms of itself. For an example, a list is a recursive type as every subset of the list is itself a list. Trees are another example, as nodes are removed from a tree the remaining construct is still a tree.

Formally, the set of values of a recursive type, *T*, will be dfefined by a recursive set equation on the form:

T = ... T ...

A recursive set equation can have many solutions, but such an equation always has a least

solution that is a subset of every other solution.

As a practical example consider this Haskell definition for a list type,

```
data List a = Nil | Cons a (List a)
```

meaning that a list of `a`'s is either an empty list or a cons cell containing an '`a`' (the "head" of the list) and another list (the "tail").

Examples

Fibonacci Sequence Using Recursion in Javascript

```
// Returns the nth number in the Fibonacci sequence
function fibonacci(n) {
    if (n === 1 || n === 0) {
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Checking for Palindromes with Recursion in C++

```
// Checks a string to see if it is a palindrome
bool function IsPalindrome(string input) {
    if (input.size() <= 1) {
        return true;
    }
    else if (input[0] == input[input.size() - 1]) {
        return IsPalindrome(input.substr(1,input.size() - 2));
    }
    else {
        return false;
    }
}
```

Create A Hierarchy Structure Using Recursion [JavaScript]

```
var directories = [
  {name: 'users' , parent : null },
  {name: 'distalx' , parent : 'users' },
  {name: 'guest' , parent : 'users' },
  {name: 'shared' , parent : 'users' },
  {name: 'documents' , parent : 'distalx' },
  {name: 'music' , parent : 'distalx' },
  {name: 'desktop' , parent : 'distalx' },
  {name: 'javascript' , parent : 'documents' },
  {name: 'funjs' , parent : 'documents' },
  {name: 'functions' , parent : 'documents' }
]
```

```
var sortDirectories= function(directories, parent){
  let node = [];
  directories
  .filter(function(d){ return d.parent === parent})
  .forEach(function(d){
    var cd = d;
    cd.child = sortDirectories(directories, d.name);
    return node.push(cd);
  })
  return node;
}

var results = sortDirectories(directories, null);
JSON.stringify(results, null, ' ');
```

output

```
[{
  "name": "users",
  "parent": null,
  "child": [
      { "name": "distalx",
        "parent": "users",
        "child": [
            { "name": "documents",
              "parent": "distalx",
              "child": [
                  { "name": "javascript",
                    "parent": "documents",
                    "child": []
                  },
                  { "name": "funjs",
                    "parent": "documents",
                    "child": []
                  },
                  { "name": "functions",
                    "parent": "documents",
                    "child": []
                  }
              ]
            },
            { "name": "music",
              "parent": "distalx",
              "child": []
            },
            { "name": "desktop",
              "parent": "distalx",
              "child": []
            }
        ]
      },
      {
        "name": "guest",
        "parent": "users",
        "child": []
      },
      {
```

```
        "name": "shared",
        "parent": "users",
        "child": []
      }
  ]
}]
```

Searching For A Value In A Binary Search Tree With Recursion in C/C++

Structure for Binary Search Tree (BST) nodes:

```
struct node {
    int data;
    node * left;
    node * right;
}
```

Search Algorithm

```
// Check if a value exists in the tree
bool BSTSearch(node * current, int value)
{
    if (current->data == value)
    {
        return true;
    }
    else if (current->data < value)
    {
        if (current->left == NULL)
        {
            return false;
        }
        else
        {
            return BSTSearch(current->left, value);
        }
    }
    else if (current->data > value)
    {
        if (current->right == NULL)
        {
            return false;
        }
        else
        {
            return BSTSearch(current->right, value);
        }
    }
}
```

Get all Combinations of Elements from Arrays in Javascript

```
function getCombinations(params, combinationsResults){
    if(params.length == 0) return combinationsResults;
    var head = params[0];
    var tail = params.slice(1);
```

```
    var combinationsResultsCurrent = [];
    if(Array.isArray(head)){
        _.uniq(head).forEach(function(item){
            if(combinationsResults.length == 0){
                combinationsResultsCurrent.push(item);
            } else {
                combinationsResults.forEach(function(previousResultItem){
                    combinationsResultsCurrent.push(previousResultItem.concat([item]));
                });
            }
        });
    } else {
        if(combinationsResults.length == 0){
            combinationsResultsCurrent.push(head);
        } else {
            combinationsResults.forEach(function(previousResultItem){
                combinationsResultsCurrent.push([previousResultItem].concat([head]));
            });
        }
    }
    return getCombinations(tail, combinationsResultsCurrent);
```

example:

Suppose we have a query with IN caluses:

```
SELECT * FROM custom_table WHERE user_id = 'user1' AND location IN ('home', 'work', AND date
IN ('2017-01-10', '2017-01-11'))
```

AND would like to get all combinations of parameters to generate queries without IN conditions:

```
SELECT * FROM custom_table WHERE user_id = [value for user_id] AND location = [value for
possible location] AND date = [value for possible date]
```

In order to get all possible combinations of parameters for equivalent queries, we can run the function above:

```
var params = ['user1', ['home', 'work'], ['2017-01-10', '2017-01-11']];
```

Read Getting started with Recursion online: https://riptutorial.com/recursion/topic/2682/getting-started-with-recursion

Credits

S. No	Chapters	Contributors
1	Getting started with Recursion	Ayan, Community, distalx, Filip Allberg, Honza Brabec, Niko Gamulin, Rainbacon