Recursion

Written by Rachel J. Morris, last updated 2016-04-10

Terminal

File Edit View Search Terminal He

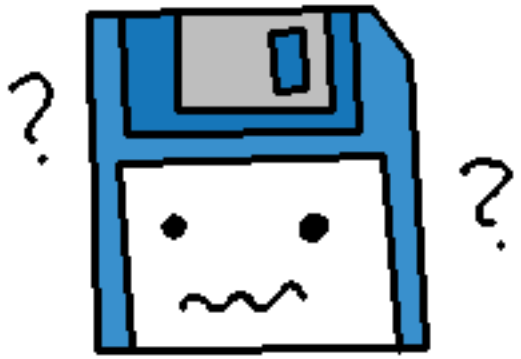
```
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
Stop hitting yourself
```

hit.cpp - /home/rejcx/PROJECTS/Teaching/CPP-C

File Edit Search View Document Project Build Tools Help

hit.cpp ×

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Person
6  {
7      public:
8      Person( const string& name )
9      {
10         m_name = name;
11     }
12
13     void Hit( const Person& p )
14     {
15         cout << "Stop hitting " << p.m_name << endl;
16         Hit( p );
17     }
18
19     private:
20     string m_name;
21 };
22
23 int main()
24 {
25     Person yourself( "yourself" );
26     yourself.Hit( yourself );
27
28     return 0;
29 }
30
```



Learning to think recursively

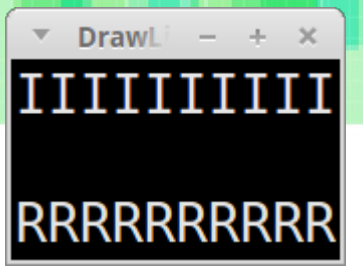
Recursion

Recursion, generally, is when a function calls itself.

The main idea behind this is that a problem can be solved with the same logic, and by splitting the problem into smaller bits.

Recursion is used a lot with searching and sorting algorithms.

Recursion



Iterative

```
void DrawLine_Iterative( int amount, char symbol )
{
    for ( int i = 0; i < amount; i++ )
    {
        cout << symbol;
    }
}
```

Sometimes, we can implement something as a simple loop...

Recursive

```
void DrawLine_Recursive( int amount, char symbol )
{
    if ( amount == 0 )  return;

    cout << symbol;
    DrawLine_Recursive( amount-1, symbol );
}
```

...or as a recursive function, with relative ease.

However, sometimes a recursive function makes for the simplest implementation

Recursion

Recursive

```
void DrawLine_Recursive( int amount, char symbol )
{
    if ( amount == 0 )  return;

    cout << symbol;
    DrawLine_Recursive( amount-1, symbol );
}
```

When creating a function that will call itself, we need to make sure to include a

Base case (aka stopping case)

so the function does not loop forever.

But if that base case is not hit, we will do the process over again, and call the function with new parameters.

Recursion

Iterative

```
int Factorial_Iterative( int number )
{
    int value = number;
    for ( int i = number-1; i > 0; i-- )
    {
        value *= i;
    }
    return value;
}
```

A recursive function can be void, or it can return a value.

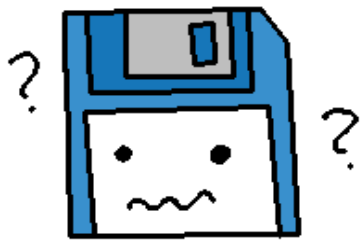
Recursive

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

It can take some work to think in terms of “recursion” instead of “iteration”.

Build your function a step at a time, and test after each addition.



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1

1

Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

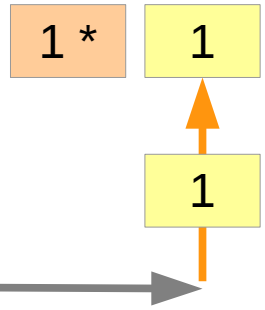
Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

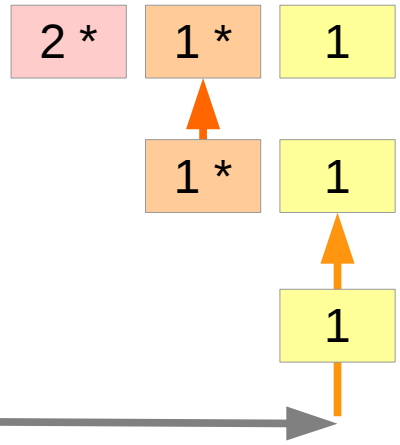
Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

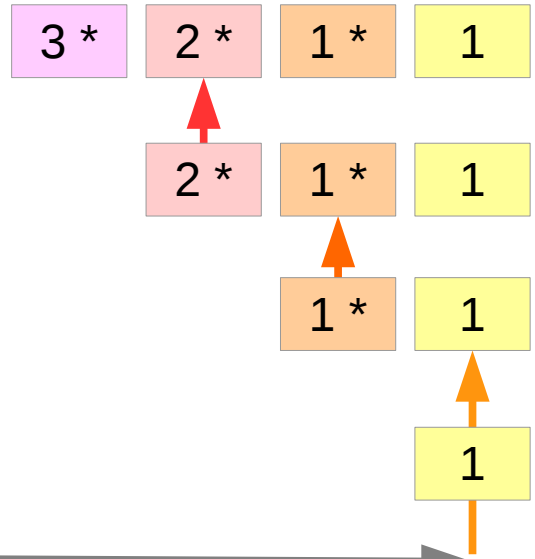
Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of
`Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of
`Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of
`Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

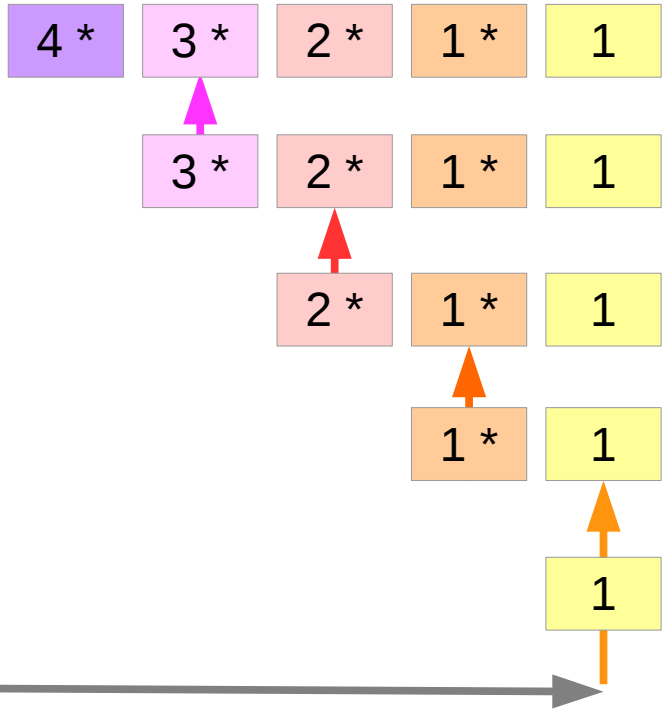
Number isn't 0, so multiply 2 by the result of
`Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of
`Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

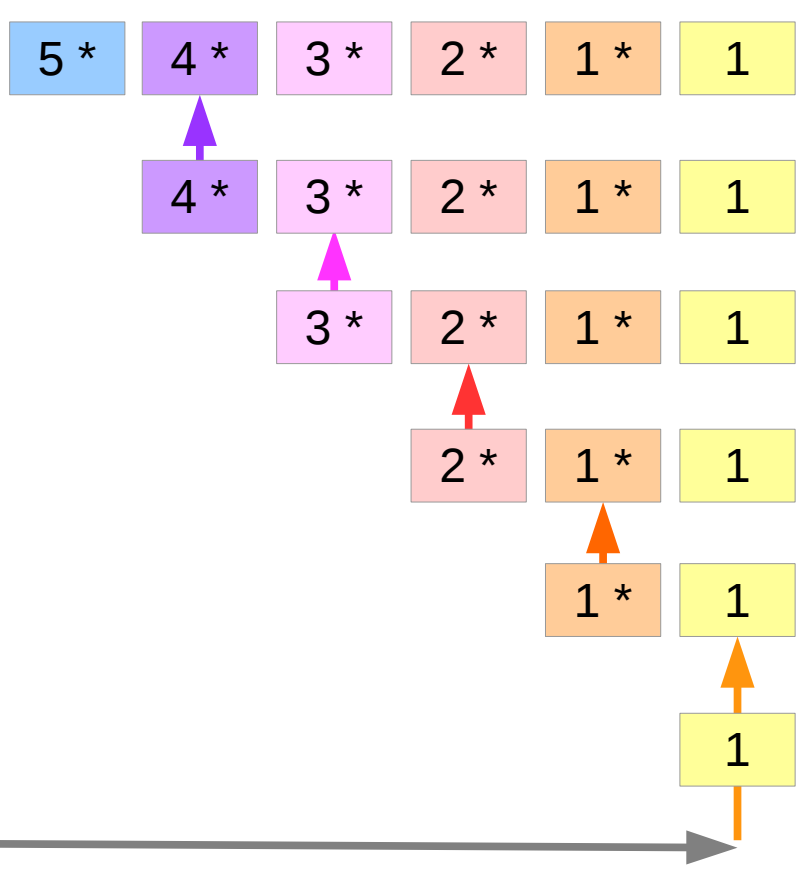
Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

Number is 0, so return 1



Recursion

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

120

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`

Call #4:
`Factorial_Recursive( 2 );`

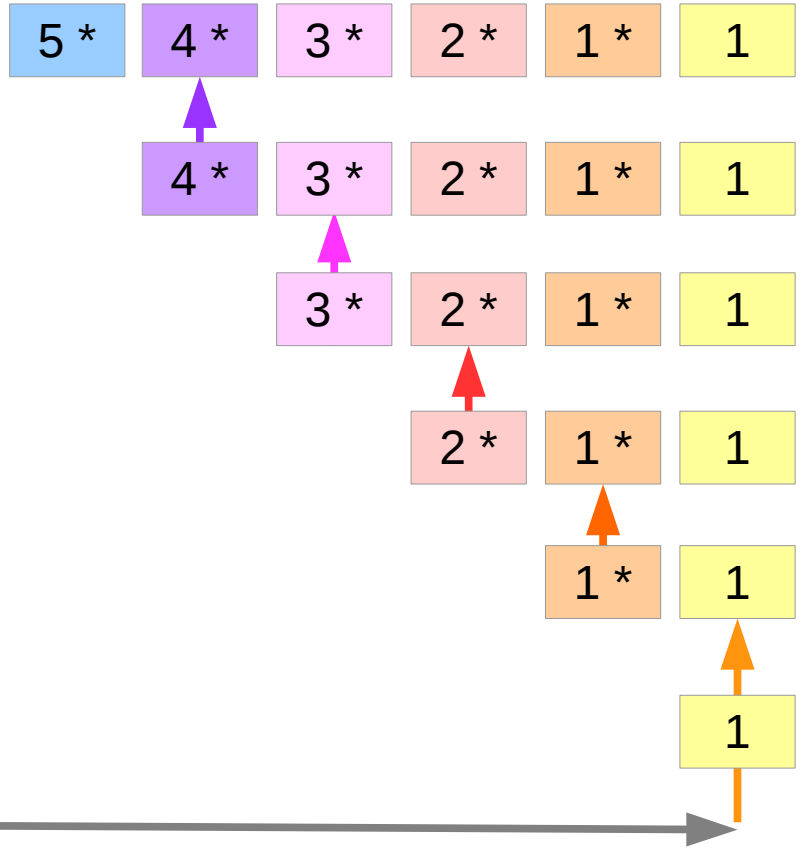
Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`

Call #5:
`Factorial_Recursive( 1 );`

Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`

Call #6:
`Factorial_Recursive( 0 );`

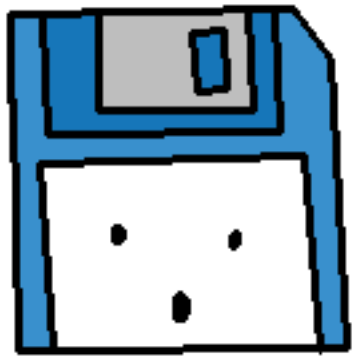
Number is 0, so return 1



Recursion

That was a simple example, but as functions that use recursion get more complex, it might still help to step through and write down variable values at each step.

Where does recursion really come in handy?



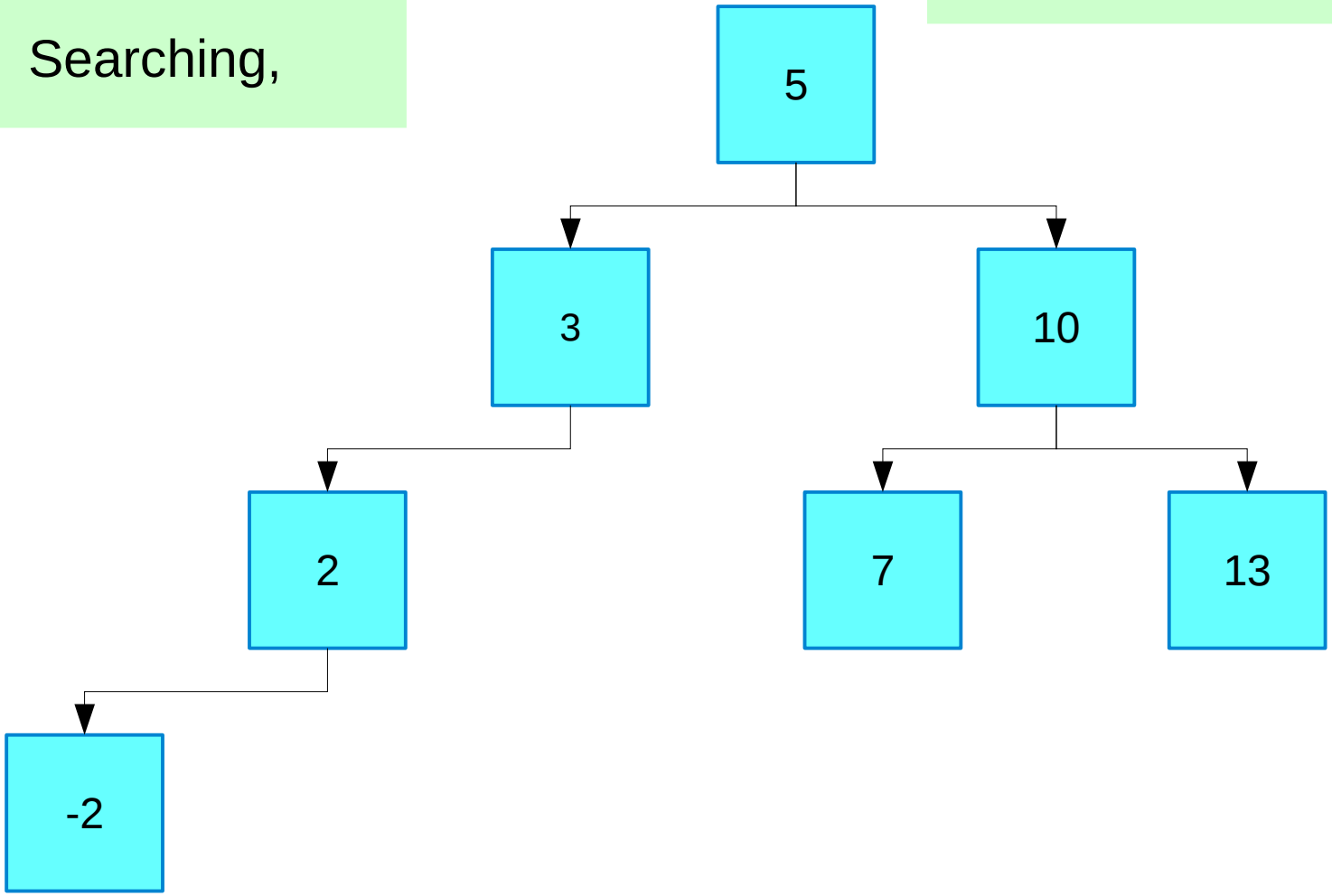
Recursion

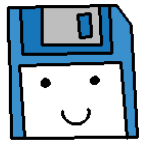
Data.

Sorting,

Building data structures

Searching,





Recursion

Here are some practice problems to work on...

Practice

1. Write a recursive function that can be used to generate a Fibonacci sequence, up to some length specified by the user.

2. Write a recursive function that can be used to compute the value of X^n , by multiplying *X* by itself *n* amount of times.

3. Write a recursive function that will reverse a string,
so the first letter is last and vice versa, through the entire string.