Recursion

- So far, we've talked about looping as the mechanism to repeatedly execute a block of code.
- We've also talked about functions and the caller/callee relationship
- So what if we have a function call itself?
  - I.e. the caller is the same function as the callee
- This is known as *recursion* and it one of the most powerful ways to control a program.

Base Cases

- If a function is going to call itself, how will the function eventually stop calling itself?
- If the function doesn't have a way to stop calling itself, the function will call itself for forever (essentially an infinite loop) until your computer runs out of resources.
- We fix this problem by creating a **base case** that doesn't call the function again
- Example: A factorial function
  - Definition of a factorial:
  - $n! = \prod_{k=1}^{n} k$
  - How can we write this in a recursive manner?
  - How can we write the factorial of n as a function of the factorial of $n-1$?
  - $n! = n * (n-1)!$
  - Ok, so we can write the factorial of n as a function of the factorial of $n-1$. But what should the base case be?
    - When $n = 1$, we stop
  - In C, this code is incredibly simple to write:

```
int factorial(int n){
  // base case
  if(n == 1)
  {
    return 1;
  }
  // otherwise, recurs into factorial(n * 1) (this is called the
     recursive case)
  else
  {
    return n * factorial(n-1);
  }
}
```

- How would you write this function using a for loop?

```c
int factorial_loop(int n){
  int fac = 1;
  for (int i = 1; i <=n; i++){
    fac *= i;
  }
  return fac;
}
```

- The parallel to the **base case** is the **recursive case** where the function calls itself.
- The recursive case should make some progress towards the base case, otherwise the program may never terminate

The Fibonacci Sequence

- Fibonacci (introduced the idea in 1202) wondered a simple question has an interesting mathematical formulation: how many rabbits could be born in a year?
- He assumed the following conditions:
  - Begin with one male rabbit and female rabbit that have just been born.
  - Rabbits reach sexual maturity after one month.
  - The gestation period of a rabbit is one month. (How long it takes to give birth - for humans it's 9 months typically)
  - After reaching sexual maturity, female rabbits give birth every month.
  - A female rabbit gives birth to one male rabbit and one female rabbit.
  - Rabbits do not die.
- This is best shown with this diagram:

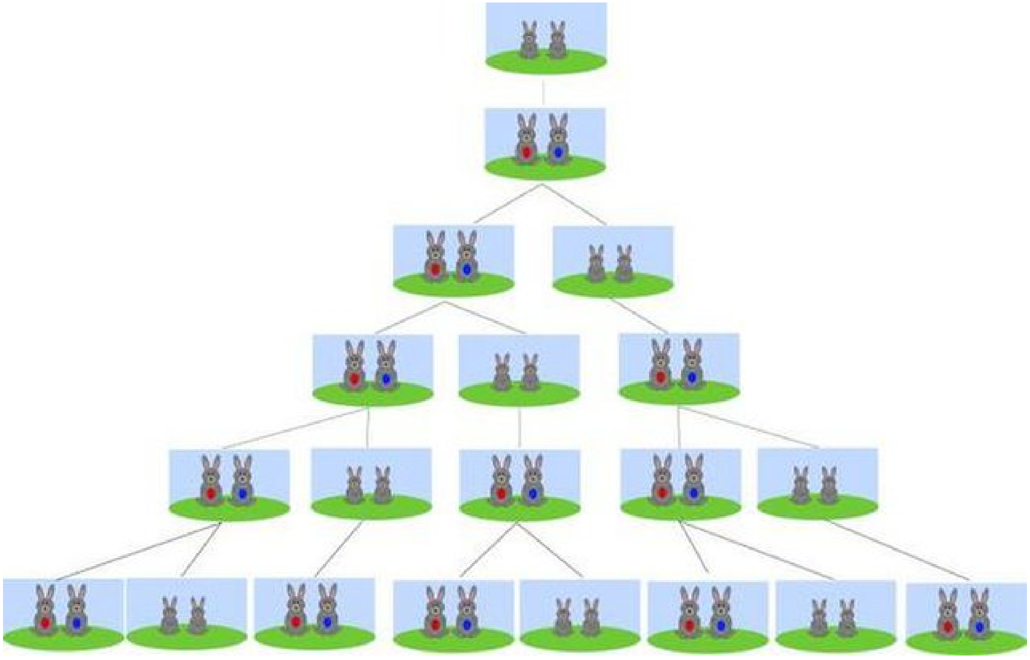

Figure 1: fibonacci_rabbits.jpg

- After one month, the first pair is not yet at sexual maturity and can't mate.
- At two months, the rabbits have mated but not yet given birth, resulting in only one pair of rabbits.
- After three months, the first pair will give birth to another pair, resulting in two pairs.
- At the fourth month mark, the original pair gives birth again, and the second pair mates but does not yet give birth, leaving the total at three pair.
- This continues until a year has passed, in which there will be 233 pairs of rabbits.
- Why Care?
  - Fibonacci's observation extends far beyond breeding rabbits. This pattern shows up in nature everywhere - growth pattern of sunflower seeds, hurricanes, galaxies. Tons of spirals in nature follow this pattern

Formal definition

- $f_n = f_{n-1} + f_{n-2}$
- Initial values at 1 and 2 for f_{n-1} and f_{n-2}, respectively (this is a hint for our base case!
  - $f(0) = 0$
  - $f(1) = 1$
  - $f(n) = f(n-1) + f(n-2)$
- How would you write the recursive version to output

```c
int fib(int n){
  if (n == 0)
  {
    return 0;
  }
  else if (n == 1)
  {
    return 1;
  }
  else
  {
    return fib(n-1) + fib(n-2);
  }
}
```

- Think about how this executes in terms of caller/callee
  - The recursive call chases down a ‘rabbit hole’ to get to the base cases, and then starts to return values up to the initial caller, where n is the initial input.
- How would you write this function using a for loop?

```c
int fib_loop(int n){
  int first = 0, second = 1, next;
  for (int i = 0 ; i <= n ; i++ )
  {
    if ( i <= 1 )
    {
      next = i;
    }
    else
    {
      next = first + second;
      first = second;
      second = next;
    }
  }
  return next;
}
```

- Possible to write using a loop, but less clear, and farther away from the underlying math.

Exercises

1. Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as parameter.

```c
#include<stdio.h>

int sum_of_range(int);

int main()
{
  int n;
  int sum;

  printf("Input the last number of the range starting from 1: ");
  scanf("%d", &n);

  sum = sum_of_range(n);
  printf("The sum of numbers from 1 to %d : %d\n\n", n, sum);

  return 0;
}

int sum_of_range(int n)
{
  if (n == 1)
  {
    return 1;
  }
  else
  {
    return n + sum_of_range(n - 1);
  }
}
```

2. Write a program in C to count the digits of a given number using recursion

```c
#include<stdio.h>

int num_digits(int n, int count);

int main()
{
```

```c
  int n, count = 0;
  printf("Input  a number: ");
  scanf("%d", &n);

  count = num_digits(n, count);

  printf("The number of digits in the number is :  %d \n\n", count);
  return 0;
}

int num_digits(int n){
  if (n < 10)
  {
    return 1;
  }
  else
  {
    return 1 + num_digits(n/10);
  }
}
```

3. Write a program in C to convert a decimal number to a binary number using recursion.

```c
#include <stdio.h>

long convert_to_binary(int decimal, long binary, long factor);

int main()
{
  long binary = 0;
  int decimal;

  printf("Input any decimal number: ");
  scanf("%d", &decimal);

  // seed a binary value of 0 and a factor of 1
  binary = convert_to_binary(decimal, 0, 1);
  printf("The Binary value of decimal number %d is: %ld\n\n", decimal,
      binary);
  return 0;
}
long convert_to_binary(int decimal, long binary, long factor)
{
```

```
  long binary_digit;

  if (decimal == 0)
  {
    return binary;
  }
  else
  {
    binary_digit = decimal % 2;
    binary = binary + binary_digit * factor;
    factor = factor * 10;
    return convert_to_binary(decimal / 2, binary, factor);
  }
}
```