COMSM1201 : Data Structures & Algorithms

Dr. Neill Campbell
Neill.Campbell@bristol.ac.uk

University of Bristol

October 20, 2021

Table of Contents

Simple Recursion

- When a function calls itself, this is known as recursion.

Simple Recursion

- When a function calls itself, this is known as recursion.
- This is an important theme in Computer Science that crops up time & time again.

Simple Recursion

- When a function calls itself, this is known as recursion.
- This is an important theme in Computer Science that crops up time & time again.
- Can sometimes lead to very simple and elegant programs.

Simple Recursion

- When a function calls itself, this is known as recursion.
- This is an important theme in Computer Science that crops up time & time again.
- Can sometimes lead to very simple and elegant programs.
- Let's look at some toy examples to begin with.

Simple Recursion

- When a function calls itself, this is known as recursion.
- This is an important theme in Computer Science that crops up time & time again.
- Can sometimes lead to very simple and elegant programs.
- Let's look at some toy examples to begin with.

Simple Recursion

- When a function calls itself, this is known as recursion.
- This is an important theme in Computer Science that crops up time & time again.
- Can sometimes lead to very simple and elegant programs.
- Let's look at some toy examples to begin with.

```c
#include <stdio.h>
#include <string.h>

#define SWAP(A,B) {char temp; temp=A;A=B;B=temp;}

void strrev(char *s, int n);

int main(void)
{
    char str[] = "Hello World!";
    strrev(str, strlen(str));
    printf("%s\n", str);
    return 0;
}

/* Iterative Inplace String Reverse */
void strrev(char *s, int n)
{
    for(int i=0, j=n-1; i<j; i++, j--){
        SWAP(s[i], s[j]);
    }
}
```

Execution :

```
!dlroW olleH
```

Recursion for *strrev()*

```c
#include <stdio.h>
#include <string.h>

#define SWAP(A,B) {char temp; temp=A;A=B;B=temp;}

void strrev(char *s, int start, int end);

int main(void)
{
    char str[] = "Hello World!";
    strrev(str, 0, strlen(str)-1);
    printf("%s\n", str);
    return 0;
}

/* Recursive : Inplace String Reverse */
void strrev(char *s, int start, int end)
{
    if(start >= end){
        return;
    }
    SWAP(s[start], s[end]);
    strrev(s, start+1, end-1);
}
```

Execution :

```
!dlroW olleH
```

Recursion for *strrev()*

```c
#include <stdio.h>
#include <string.h>

#define SWAP(A,B) {char temp; temp=A;A=B;B=temp;}

void strrev(char *s, int start, int end);

int main(void)
{
    char str[] = "Hello World!";
    strrev(str, 0, strlen(str)-1);
    printf("%s\n", str);
    return 0;
}

/* Recursive : Inplace String Reverse */
void strrev(char *s, int start, int end)
{
    if(start >= end){
        return;
    }
    SWAP(s[start], s[end]);
    strrev(s, start+1, end-1);
}
```

Execution :

```
!dlroW olleH
```

- We need to change the function prototype.

Recursion for *strrev()*

```c
#include <stdio.h>
#include <string.h>

#define SWAP(A,B) {char temp; temp=A;A=B;B=temp;}

void strrev(char *s, int start, int end);

int main(void)
{
    char str[] = "Hello World!";
    strrev(str, 0, strlen(str)-1);
    printf("%s\n", str);
    return 0;
}

/* Recursive : Inplace String Reverse */
void strrev(char *s, int start, int end)
{
    if(start >= end){
        return;
    }
    SWAP(s[start], s[end]);
    strrev(s, start+1, end-1);
}
```

Execution :

```
!dlroW olleH
```

- We need to change the function prototype.
- This allows us to track both the start and the end of the string.

The Fibonacci Sequence

A well known example of a recursive function is the Fibonacci sequence. The first term is 1, the second term is 1 and each successive term is defined to be the sum of the two previous terms, i.e. :

fib(1) is 1
fib(2) is 1
fib(n) is fib(n-1)+fib(n-2)

1,1,2,3,5,8,13,21, ...

Iterative & Recursive Fibonacci

```c
#include <stdio.h>

#define MAXFIB 25

int fibonacci(int n);

int main(void)
{

    for(int i=1; i<MAXFIB; i++){
        printf("%d = %d\n", i, fibonacci(i));
    }

    return 0;

}

int fibonacci(int n)
{
    if(n <= 2){
        return 1;
    }
    int a = 1;
    int b = 1;
    int next;
    for(int i=3; i<=n; i++){
        next = a + b;
        a = b;
        b = next;
    }
    return b;
}
```

Iterative & Recursive Fibonacci

```c
#include <stdio.h>

#define MAXFIB 25

int fibonacci(int n);

int main(void)
{

    for(int i=1; i<MAXFIB; i++){
        printf("%d = %d\n", i, fibonacci(i));
    }

    return 0;

}

int fibonacci(int n)
{
    if(n <= 2){
        return 1;
    }
    int a = 1;
    int b = 1;
    int next;
    for(int i=3; i<=n; i++){
        next = a + b;
        a = b;
        b = next;
    }
    return b;
}
```

Execution :

```
1 = 1
2 = 1
3 = 2
4 = 3
5 = 5
6 = 8
7 = 13
8 = 21
9 = 34
10 = 55
11 = 89
12 = 144
13 = 233
14 = 377
15 = 610
16 = 987
17 = 1597
18 = 2584
19 = 4181
20 = 6765
21 = 10946
22 = 17711
23 = 28657
24 = 46368
```

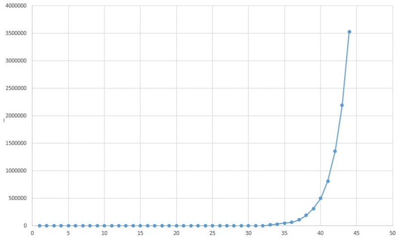
Iterative & Recursive Fibonacci

```c
#include <stdio.h>

#define MAXFIB 25

int fibonacci(int n);

int main(void)
{

    for(int i=1; i<MAXFIB; i++){
        printf("%d = %d\n", i, fibonacci(i));
    }

    return 0;

}

int fibonacci(int n)
{
    if(n == 1) return 1;
    if(n == 2) return 1;
    return( fibonacci(n-1)+fibonacci(n-2));
}
```

Iterative & Recursive Fibonacci

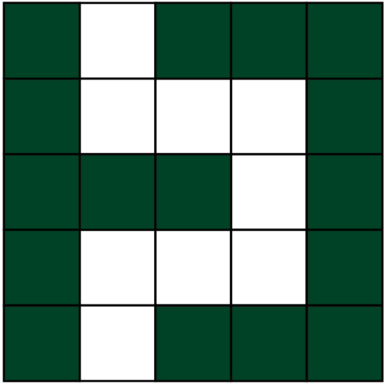
```c
#include <stdio.h>

#define MAXFIB 25

int fibonacci(int n);

int main(void)
{

    for(int i=1; i<MAXFIB; i++){
        printf("%d = %d\n", i, fibonacci(i));
    }

    return 0;

}

int fibonacci(int n)
{
    if(n == 1) return 1;
    if(n == 2) return 1;
    return( fibonacci(n-1)+fibonacci(n-2));
}
```

It's interesting to see how run-time increases as the length of the sequence is raised.



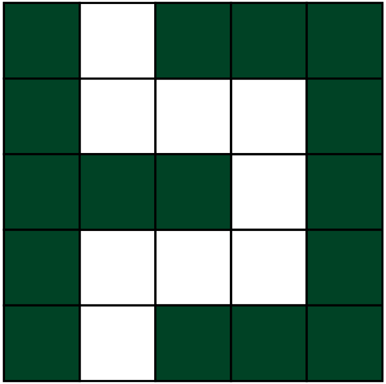
Maze Escape

The correct route through a maze can be obtained via recursive, rather than iterative, methods.



Maze Escape

The correct route through a maze can be obtained via recursive, rather than iterative, methods.



```
bool explore(int x, int y, char mz[YS][XS])
{
  if  mz[y][x] is exit return true;

  Mark mz[y][x] so we don't return here

  if we can go up :
    if(explore(x, y+1, mz)) return true

  if we can go right :
    if(explore(x+1, y, mz)) return true

  Do left & down in a similar manner

  return false; // Failed to find route
}
```

Permuting

- Here we consider the ways to permute a string (or more generally an array)

Permuting

- Here we consider the ways to permute a string (or more generally an array)
- Permutations are all possible ways of rearranging the positions of the characters.

Execution :

```
ABC
ACB
BAC
BCA
CBA
CAB
```

Permuting

- Here we consider the ways to permute a string (or more generally an array)
- Permutations are all possible ways of rearranging the positions of the characters. Execution :

```
ABC
ACB
BAC
BCA
CBA
CAB
```

Permuting

- Here we consider the ways to permute a string (or more generally an array)
- Permutations are all possible ways of rearranging the positions of the characters.

Execution :

ABC
ACB
BAC
BCA
CBA
CAB

```c
// From e.g.  http://www.geeksforgeeks.org
#include <stdio.h>
#include <string.h>

#define SWAP(A,B) {char temp = *A; *A = *B; *B = temp;}

void permute(char *a, int s, int e);

int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}

void permute(char *a, int s, int e)
{
   if (s == e){
     printf("%s\n", a);
     return;
   }
   for (int i = s; i <= e; i++){
      SWAP((a+s), (a+i)); // Bring one char to the front
      permute(a, s+1, e);
      SWAP((a+s), (a+i)); // Backtrack
   }
}
```

Self-test : Power

- Raising a number to a power `n = 2^5` is the same as multiple multiplications `n = 2*2*2*2*2`.

Self-test : Power

- Raising a number to a power `n = 2^5` is the same as multiple multiplications `n = 2*2*2*2*2`.
- Or, thinking recursively, `n = 2 * (2^4)`.

Self-test : Power

- Raising a number to a power `n = 2^5` is the same as multiple multiplications `n = 2*2*2*2*2`.
- Or, thinking recursively, `n = 2 * (2^4)`.

Self-test : Power

- Raising a number to a power $\mathtt{n} = 2\hat{\ }5$ is the same as multiple multiplications $\mathtt{n} = 2*2*2*2*2$.
- Or, thinking recursively, $\mathtt{n} = 2 * (2\hat{\ }4)$.

```c
/* Try to write power(a,b) to computer a^b
   without using any maths functions other than
   multiplication :
   Try (1) iterative then (2) recursive
   (3) Trick that for n%2==0, x^n = x^(n/2)*x^(n/2)

*/

#include <stdio.h>

int power(unsigned int a, unsigned int b);

int main(void)
{

   int x = 2;
   int y = 16;

   printf("%d^%d = %d\n", x, y, power(x,y));

}

int power(unsigned int a, unsigned int b)
{
}
```