Recursion

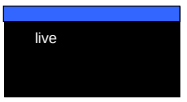
What Does this Program Do?

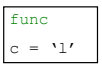
```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:

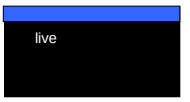


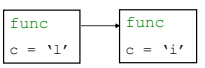


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:

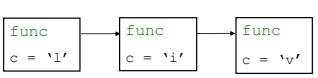




What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

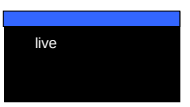
Input:

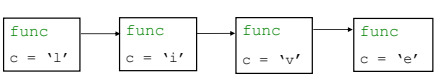


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:



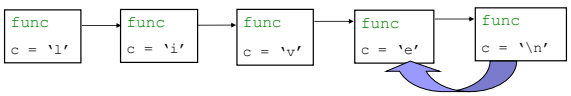


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:

```
live
e
```

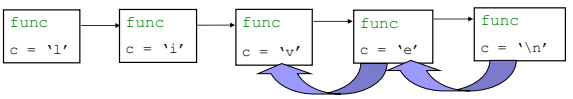


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:

```
live
ev
```

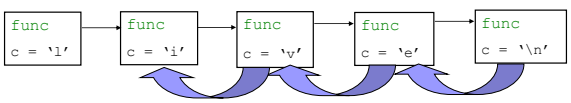


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

Input:

```
live
evi
```

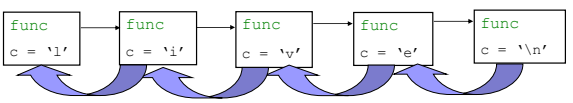


What Does this Program Do?

```
void func(){
        char c;
        c = getchar();
        if (c == '\n')
                return;
        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```

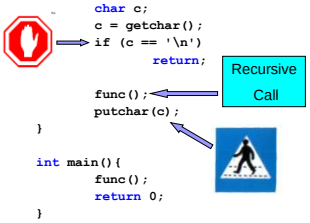
Input:

```
live
evil
```



Reverse Print

```
void reverse_print(){
        char c;
        c = getchar();
        if (c == '\n')
                return;

        func();
        putchar(c);
}

int main(){
        func();
        return 0;
}
```



Exercise

- Write a program that receives an integer and returns its sum of digits.
- We do not know the number of digits in advance.
- Example: the sum of digits of 369 is 18.
- Before we begin, let us recall the recursion “rules”.

The Three Rules of Recursion

1. Base (termination) condition
2. Decomposition to smaller instances
3. Use solutions to smaller instances to solve the original problem

Solution

`sum_digits.c`

GCD – Greatest Common Divisor

- Definition:

For two or more non-zero integers,
The GCD is the largest positive integer that divides both numbers without a remainder.

Examples:
GCD(12, 8) = 4
GCD(12, 4) = 4
GCD(12, 5) = 1

GCD – Euclid's Algorithm

GCD(616, 143) ?
616 % 143 = 44 ➔
GCD(143, 44)
143 % 44 = 11 ➔
GCD(44, 11)
44 % 11 = 0
➔ 11

GCD - Code

```
#include <stdio.h>

int gcd(int m, int n)
{
    if (m == 0)
        return n;
    if (n == 0)
        return m;

    return gcd(n, m % n);
}
```

```
int main()
{
    int num1, num2, div;

    printf("Enter two positive numbers: ");
    scanf("%d %d", &num1, &num2);

    div = gcd(num1, num2);

    printf("the GCD is %d\n", div);

    return 0;
}
```

Fast Power Calculation

- $x^y = \underbrace{x*x*\ldots*x}_{y \text{ times}}$
- Recursive definitions (assume non-negative y):

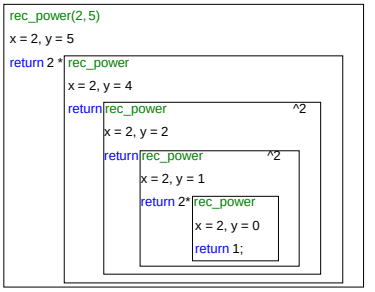
$$x^y = \begin{cases} 1, & y = 0 \\ x*x^{y-1}, & y \text{ odd} \\ (x^{y/2})^2, & y \text{ even} \end{cases}$$

Fast Power - Code

```
int rec_power(int x, int y)
{
    int tmp;

    if (y == 0)
        return 1;

    if (y % 2 != 0)
        return x * rec_power(x, y - 1);
    else
    {
        tmp = rec_power(x, y / 2);
        return tmp*tmp;
    }
}
```

Fast Power - Run

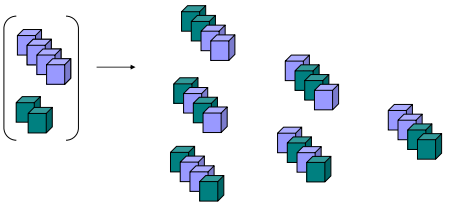
rec_power(2, 5)



Binomial Coefficient

$\binom{n}{k}$ =

- Definition 1: The coefficient of x^k in the expansion of the binomial power $(1 + x)^n$.
- Definition 2: The number of ways that *k* elements can be "chosen" from the set {1, 2, 3, …, *n*}.



Binomial Coefficient

- Think recursion:
- If the nth element is selected – there are k-1 elements to choose from {1, …, n-1}.
- If the nth element is not selected – there are k elements to choose from {1, …, n-1}.

Binomial Coefficient

Recursive formula:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Termination Criteria:

k > n → 0

k = 0 → 1

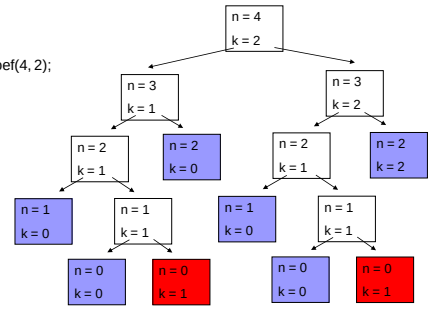
k == n → 1

Binomial Coefficient – from Formula to Code

```
int binomial_coef(int n, int k)
{
    // termination criteria
    if ((k == 0) || (n == k))
        return 1;
    if (n < k)
        return 0;

    return binomial_coef(n-1, k-1) + binomial_coef(n-1, k);
}
```

Binomial Coefficient – Recursion Tree

```
int main()
{
    binomial_coef(4, 2);
    return 0;
}
```



Reverse digits

- Receive an Integer number, reverse the order of its digits
- We already wrote the iterative version.

How can we solve it recursively?

Reverse Digits

- Example: reverse_digits(573824)

- Wishful Thinking: If I only knew reverse_digits(73824)…
- Problem – for a number with d digits, the last digit should be multiplied by 10^d.

  But how can I know d in advance?
- Solution – Call a helper function that diffuses the temporal result throughout the recursion.

- What is the termination criterion?

Reverse Digits - Code

```
int helper(int n, int res)
{
    if (n == 0)
        return res;

    return helper(n/10, 10*res + n%10);
}

int reverse_digits(int num)
{
    return helper(num, 0);
}
```

```
int main()
{
    int num;

    printf("Insert an integer\n");
    scanf("%d", &num);

    printf("%d\n", reverse_digits(num));

    return 0;
}
```