

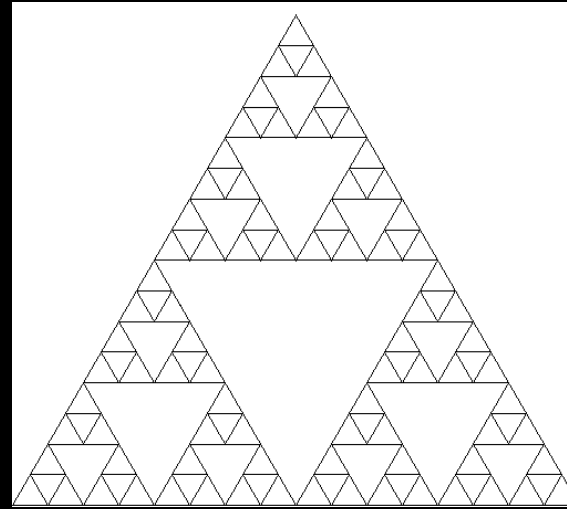
Intro a Algoritmia

Recursión

¿Qué es recursión?



Un proceso es
recursivo si se
puede definir en
funcion de sí
mismo



El clásico

Factorial

$$n ! = n \times (n-1) !$$

$$0! = 1$$



Código


```
1  
2  
3 int factorial(int n){  
4     if(n == 0)  
5         return 1;  
6  
7     int f = n * factorial(n-1);  
8     return f;  
9 }
```

Calcular el factorial de $n = 4$

```
int factorial(int n){  
    if(n == 0)  
        return 1;  
  
    int f = n * factorial(n-1);  
    return f;  
}
```

Calcular el factorial de $n = 4$

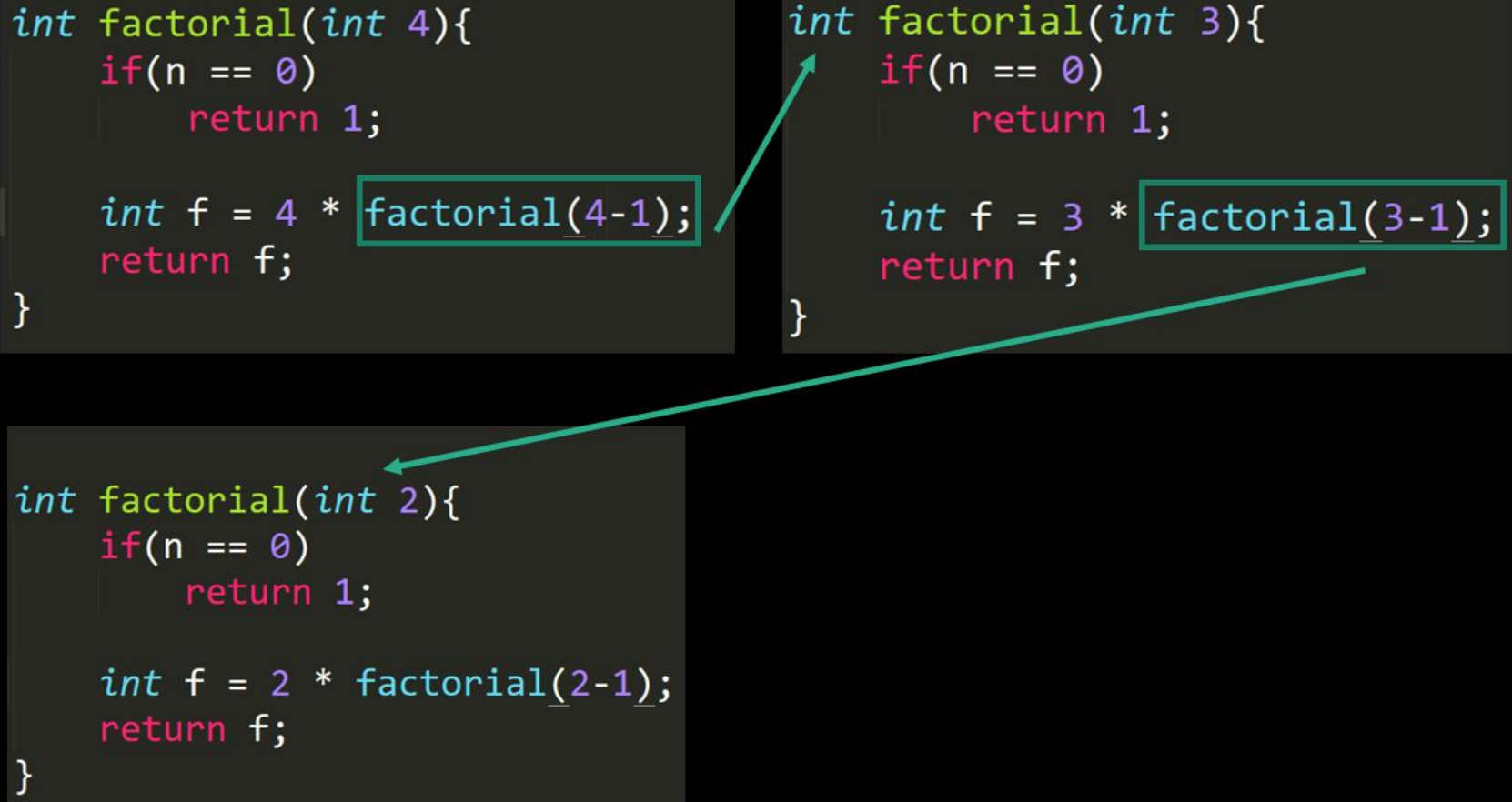
```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```



```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```


Calcular el factorial de $n = 4$

```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```



```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```

```
int factorial(int 2){  
    if(n == 0)  
        return 1;  
  
    int f = 2 * factorial(2-1);  
    return f;  
}
```

Calcular el factorial de $n = 4$

```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```


```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```

```
int factorial(int 2){  
    if(n == 0)  
        return 1;  
  
    int f = 2 * factorial(2-1);  
    return f;  
}
```

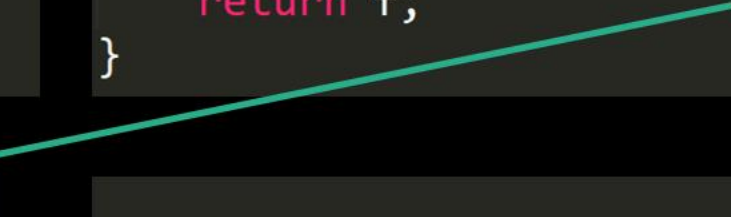
```
int factorial(int 1){  
    if(n == 0)  
        return 1;  
  
    int f = 1 * factorial(1-1);  
    return f;  
}
```

Calcular el factorial de $n = 4$


```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```




```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```



```
int factorial(int 2){  
    if(n == 0)  
        return 1;  
  
    int f = 2 * factorial(2-1);  
    return f;  
}
```



```
int factorial(int 1){  
    if(n == 0)  
        return 1;  
  
    int f = 1 * factorial(1-1);  
    return f;  
}
```



```
int factorial(int 0){  
    if(n == 0)  
        return 1;  
  
    int f = 0 * factorial(0-1);  
    return f;  
}
```

Calcular el factorial de $n = 4$

```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```

```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```

```
int factorial(int 2){  
    if(n == 0)  
        return 1;  
  
    int f = 2 * factorial(2-1);  
    return f;  
}
```

```
int factorial(int 1){  
    if(n == 0)  
        return 1;  
  
    int f = 1 * 1;  
    return f;  
}
```

Calcular el factorial de $n = 4$


```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```

```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * factorial(3-1);  
    return f;  
}
```

```
int factorial(int 2){  
    if(n == 0)  
        return 1;  
  
    int f = 2 * 1;  
    return f;  
}
```

Calcular el factorial de $n = 4$

```
int factorial(int 4){  
    if(n == 0)  
        return 1;  
  
    int f = 4 * factorial(4-1);  
    return f;  
}
```



```
int factorial(int 3){  
    if(n == 0)  
        return 1;  
  
    int f = 3 * 2;  
    return f;  
}
```

Calcular el factorial de $n = 4$

```
int factorial(int n){  
    if(n == 0)  
        return 1;  
  
    int f = 1;  
    for(int i = 1; i <= n; i++)  
        f = f * i;  
    return f;  
}
```

Calcular el factorial de $n = 4$

```
int factorial(int n){  
    if(n == 0)  
        return 1;  
  
    int f = 1;  
    for(int i = 1; i <= n; i++)  
        f = f * i;  
    return f;  
}
```

$\text{factorial}(4) = 24$

Sucesión de Fibonacci

Problema : Hallar el término n
de la sucesión de Fibonacci

La definiremos como:

$$F(1) = 1$$

$$F(2) = 1$$

$$F(n) = F(n-1) + F(n-2)$$



Código

```
3
4 int fibonacci(int n){
5     if( (n == 1) || (n == 2))
6         return 1;
7     return fibonacci(n-1) + fibonacci(n-2);
8 }
```

Ejercicio 1

Pregunta 1 (8 puntos)

Implementar una función recursiva cuya cabecera sea la siguiente: `triangulo(int a, int b)`. Esta función deberá generar un patrón como el siguiente:

```
* * *  
* * * *  
* * * * *  
* * * * *  
* * * *  
* * *
```

El patrón descrito anteriormente es generado por la llamada a `triangulo(3,5)`.

Nota: `a` siempre debe ser menor o igual que `b`

Ejercicio 2

Implementar una función recursiva que permita obtener el mayor número de un arreglo de N números enteros.

Leer el numero N, la cantidad de elementos del arreglo.

Leer los N elementos y guardalos en el arreglo.

Implementar la funcion recursiva.

Ejemplo:

Entrada:

5

10 34 20 16 5

Salida:

34

Al invocar a la funcion
obtenerMayor(arreglo, 0, n)
La funcion debe devolver el mayor numero.

Inicio del codigo:

```
int arreglo[10], n, i;  
scanf("%d",&n);  
for(i = 0; i < n; i++){  
    scanf("%d",&arreglo[i]);  
}  
/* resto del codigo */
```

Ejercicio 3

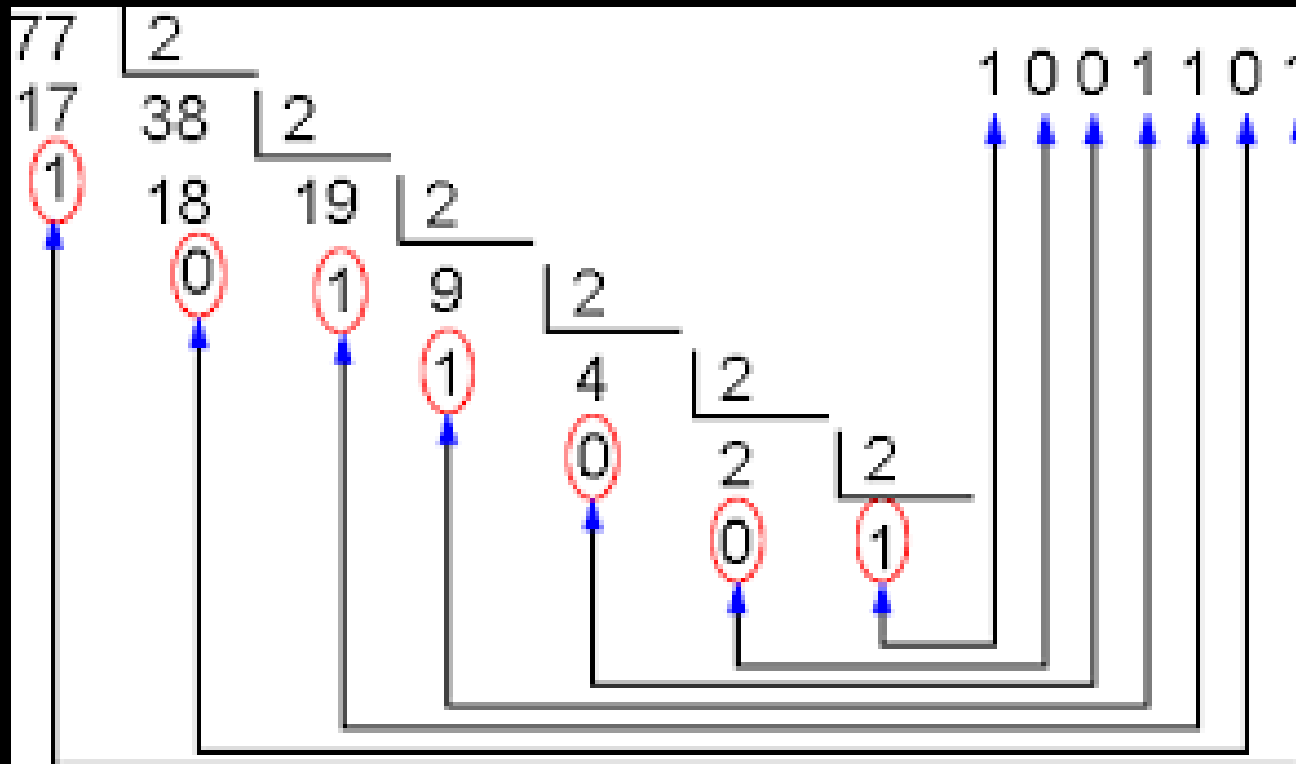
Implementar una función recursiva que reciba como único parámetro u número en base 10 y lo imprima en base 2.

Ejemplo

Entrada: 13

Salida: 1101

Metodo para convertir a binario



Nota que el primer residuo es el ULTIMO numero en imprimirse, mientras que el ultimo residuo es el primero en imprimirse.

Ejercicio 4

Pregunta 2 (12 puntos)

Analizar el siguiente patrón de asteriscos y espacios en blanco, e implementar una función recursiva que pueda generar la siguiente figura:

```
*
* *
  *
* * * *
    *
  * *
    *
* * * * * * * *
      *
    * *
      *
    * * * *
      *
        * *
          *
```

Nota:

- La cabecera de la función debe ser `patron(int n, int i)`.
- `n` siempre es una potencia de 2 mayor que 0.

La figura resulta de
Invocar a la funcion `patron(8,0)`.

Algoritmos recursivos

- Debe contener siempre algún caso base. Esto garantiza que el programa no corra infinitamente.
- La mayoría de algoritmos iterativos pueden implementarse de forma recursiva, lo cual no significa que la forma recursiva sea más eficiente.
- Implementar una solución recursiva suele ser más sencillo cuando la definición del problema es recursiva (factorial, Fibonacci, etc).
- Una función es recursiva si se invoca a sí misma.
- Definición de caso recursivo es generalmente la definición del problema en sí.

Fin de la semana 2