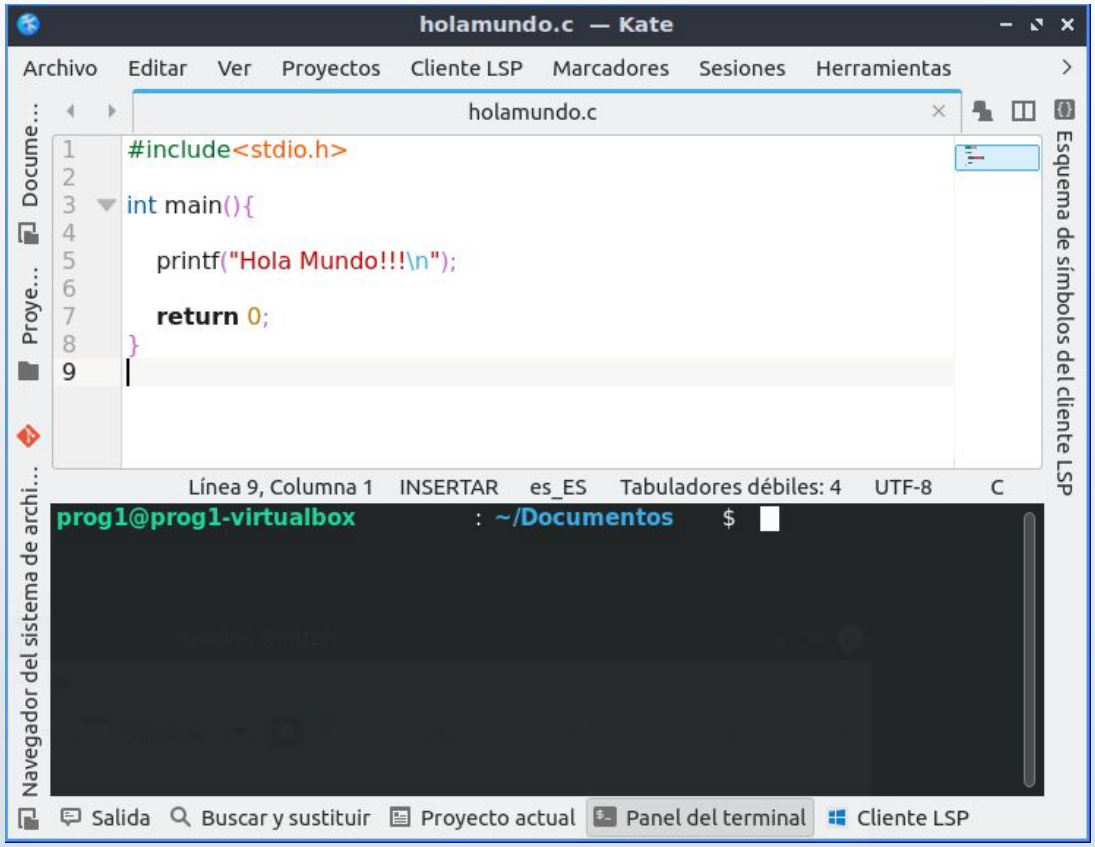



# Programming 1

## Lesson 5. Recursion

**Degree in Computer Engineering**

# Syllabus



1. Concept of recursion
2. Basic outline of a recursive module
3. Examples of recursion
4. Coding recursion in C
5. Characteristics of recursion
6. Tracing a recursive module
7. Exercises

# 1. Concept of recursion



- A module is recursive when among the list of instructions that form it, there is **a call to itself**, directly or indirectly.
- There are many mathematical functions that are naturally defined recursively. For example:
  - Factorial of a number *n*: The factorial of a number *n* is the number *n* multiplied by the factorial of *n-1*.

```
factorial(n) = n * factorial(n-1)
```

  - Power of two numbers: $x^n = x * x^{n-1}$

# 2. Basic outline of a recursive module



- One or more **base cases**
  - There are no recursive calls in them. They specify the "termination condition" or "stop condition" of the recursion.
- One or more **general or recursive cases**
  - It includes one or more calls to the module itself. These recursive calls must solve "simpler" versions of the problem to be solved by the module. In other words, it is a process in which each recursive call to the module itself receives a simpler version of the problem, until the base case is reached.

# 3. Examples of recursion



## Calculation of the factorial of a number

```
factorial(n) = n * factorial(n-1)

factorial(3) = 3 * factorial(2)
                 = 2 * factorial(1)
                       = 1 * factorial(0)
                             = 0 * …
```

When does it stop?

# 3. Examples of recursion



## Calculation of the factorial of a number

The **base case** must be added to stop recursion.

```
If n is equal to 0 Then
   factorial = 1
Else
   factorial = n * factorial of (n-1)
```

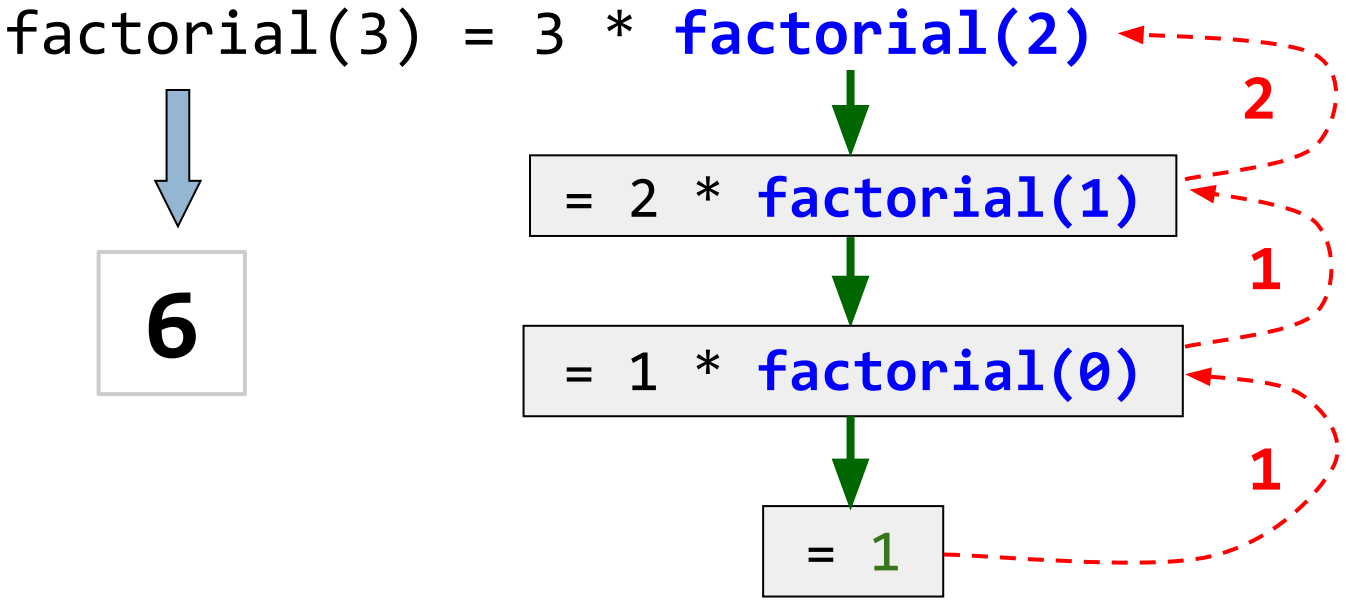
Recursive call to own module (**recursive case**)

# 3. Examples of recursion



## Calculation of the factorial of a number

```
If (n is 0) Then
    factorial = 1
Else
    factorial = n * factorial(n-1)
```



# 4. Coding recursion in C



```c
#include<stdio.h>

// Declaration of modules
int factorial( int n );

int main() {
  int num;

  printf("Enter a number: ");
  scanf("%d", &num);
  printf("The factorial of %d is: %d\n", num, factorial( num ) );

  return 0;
}
```

```c
// Definition of modules
int factorial( int n ) {
  int res;

  if ( n == 0 ) // base case
    res = 1;
  else // recursive case
    res = n * factorial( n - 1 );

  return res;
}
```

# 5. Characteristics of recursion



- Suitable for **solving problems** that can be **defined naturally** in recursive terms.
- **Has its iterative equivalent**.
- **They need more memory** for their execution.
- **They are slower** to execute.

---

**Important**:
Avoid creating infinite recursion, which would cause the program not to stop.

```
void write(int n){
   write(n / 10);
   printf("%d\n", n % 10 );
}
```

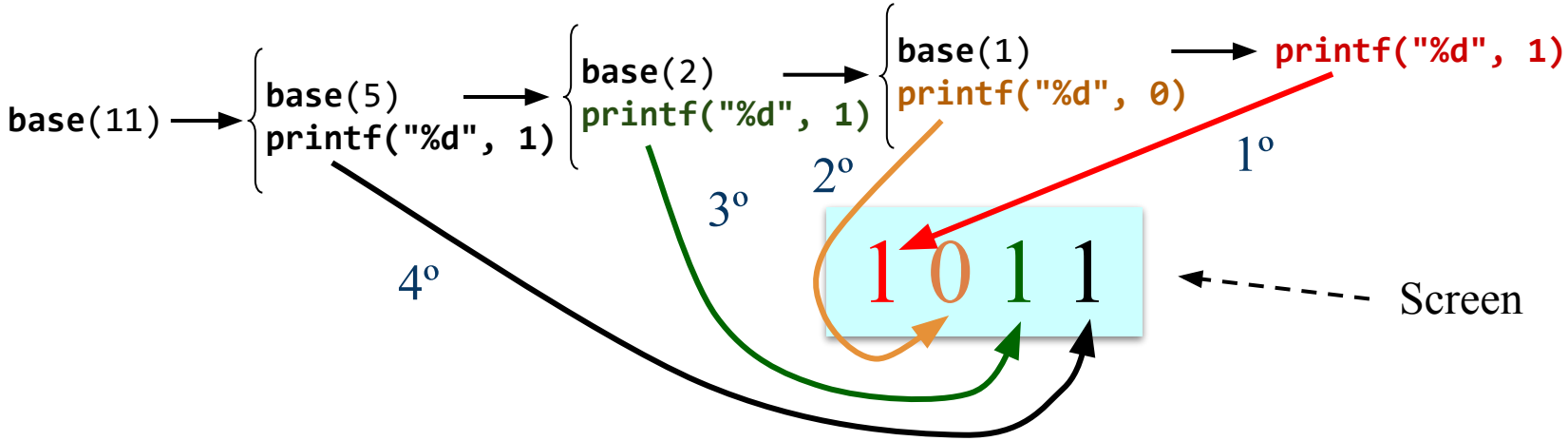
# 6. Tracing a recursive module



```
void base(int n){
     if (n < 2) // base case
        printf("%d", n);
     else { // recursive case
         base(n / 2);
         printf("%d", n % 2);
     }
}
```

```
int main() {
   ...
   base(11);
   ...

   return 0;
}
```



# 6. Tracing a recursive module



## Example:

Given the following module:

```
void recursive (int num){
     if (num != 0){ // recursive case
          recursive(num / 2);
          printf( num % 2 );
     }
}
```

1. What is the output if we make a call to the module as follows: `recursive(16)`?

   A) 00001 B) 11111

   C) 10000 D) 00100

   E) none of the above

2. What is the base case?

# 6. Tracing a recursive module



## What does this code do?

```
int main(){
  char letter;

  printf("Enter a sentence ending in a full stop: ");
  scanf("%c", &letter);
  module(letter);
  printf("%c\n", letter);
  return 0;
}
```

```
void module(char l){
    if (l == '.')  // base case
        printf("\n");
    else {  // recursive case
       scanf("%c", &l);
       module(l);
       printf("%c", l);
    }
 }
```

# 7. Exercises



1. Design a recursive module that for a natural number *n* displays the increasing series of natural numbers from 1 to *n*, i.e. 1 2 3... n.
2. Design a recursive module that for a natural number *n* returns the sum of the squares of the numbers from 1 to *n*. For example, for n=4, the module must return 30 since $1^2+ 2^2 + 3^2 + 4^2 = 30$.
3. Design a module that, given a natural number, displays on the screen the number formed by the same digits in the opposite direction. For example: for the number 2089 it must show 9802.
4. Design a module that receives a number in decimal system and displays its equivalent in binary. For example, for the number 12, it should display 1100.
5. Implement a recursive function that returns the number of odd digits of a number. Example: rec(321)=2, rec(28)=0.