Computer Programming & Problem Solving
CS100

Mrs Sanga G. Chaki
Department of Computer Science and Engineering
National Institute of Technology, Goa
May, 2023

Recursion

Recursion

1. Recursion means "defining a problem in terms of itself"
2. A process by which a function calls itself repeatedly.
3. This technique provides a way to break complicated problems down into simple problems which are easier to solve.
4. It is an example of divide and conquer algorithms

- **Used for repetitive computations in which each action is stated in terms of a previous result.**

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

Recursion

1. Adding two numbers together is easy to do.
2. But adding a range of numbers is more complicated.
3. In the following example, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers:

sum(5) = 5 + sum(4)

= 5 + (4+sum(3))

= 5 + (4+(3+sum(2)))

= 5 + (4+(3+(2+sum(1))))

Recursion – Example Code

```c
#include <stdio.h>
int sum_n(int n){ //recursive function
    if(n==1){
        return 1;
    }
    else{
        return (n+sum_n(n-1)); //recursive function call
    }
}
int main() {
    int x = 10;
    int r = sum_n(x);
    printf("Result = %d",r);
    return 0;
}
```

Recursion

- **For a problem to be written in recursive form, two conditions are to be satisfied:**
  - **It should be possible to express the problem in recursive form.**
  - **The problem statement must include a stopping condition**

```
fact(n)  =  1,                  if  n = 0
         =  n * fact(n-1),      if  n > 0
```

Recursion - Recap

Some example problems which can be solved using recursion:

1. Sum of n natural numbers

2. Factorial

3. Fibonacci Series (0, 1, 1, 2, 3, 5, 8, 13, ….)

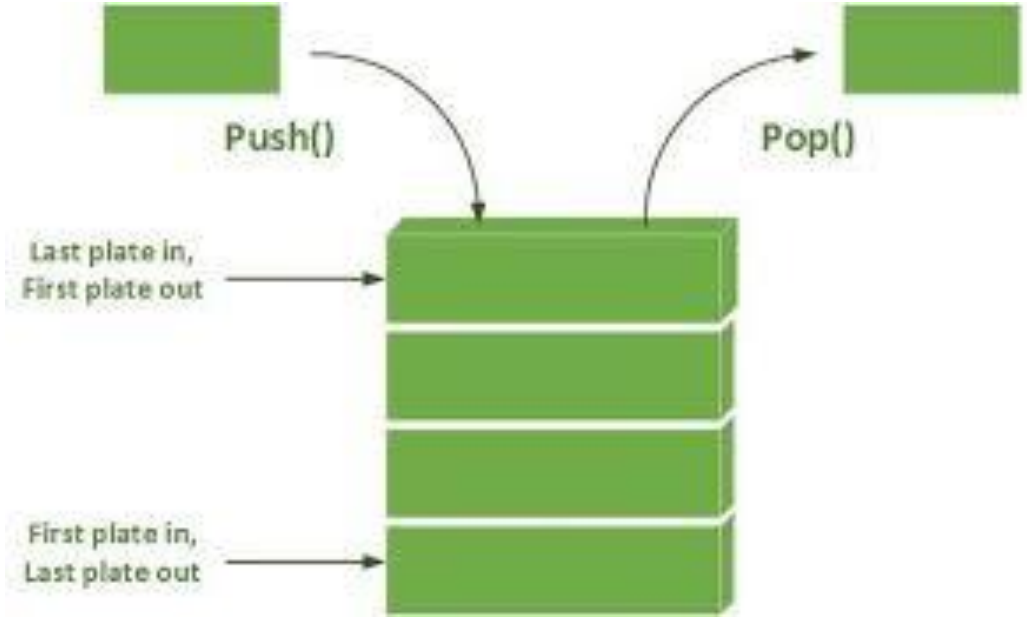
Recursion – Mechanism of Execution


```
long  int  fact (n)
int  n;
{
    if    (n == 0)
        return (1);
    else
        return  (n * fact(n-1));
}
```

- **Mechanism of execution**
  - When a recursive program is executed, the recursive function calls are not executed immediately.
    - They are kept aside (on a stack) until the stopping condition is encountered.
    - The function calls are then executed in reverse order.

Recursion – What is a stack?

1. **A stack is a Last In First Out (LIFO) data structure.**
2. **The last item to get stored on the stack (Push operation) is the first one to get out of it (Pop operation).**



Recursion - Mechanism of Execution

Example :: Calculating `fact(4)`

- First, the function calls will be processed:

```
fact(4) = 4 * fact(3)
fact(3) = 3 * fact(2)
fact(2) = 2 * fact(1)
fact(1) = 1 * fact(0)
```

- The actual values return in the reverse order:

```
fact(0) = 1
fact(1) = 1 * 1 = 1
fact(2) = 2 * 1 = 2
fact(3) = 3 * 2 = 6
fact(4) = 4 * 6 = 24
```

Recursion – Example Code – The recursive function

```c
#include <stdio.h>
int rec_fact(int z){
    printf("\n-----Function rec_fact is called-----");
    if(z==0){
        printf("\nz = %d is equal to 0",z);
        return 1;
    }
    else{
        printf("\nz = %d is not equal to 0, so doing recursive
            call",z);
        return (z*rec_fact(z-1));
    }
}
```

Recursion – Example Code – The main function

```
15  int main() {
16      int n, r;
17      printf("Enter number: ");
18      scanf("%d",&n);
19      r = rec_fact(n);
20      printf("\nFactorial of %d = %d ",n,r);
21      return 0;
22  }
```

Recursion – Example Code – The results

Output

```
/tmp/odYb4zWRS2.o
Enter number: 3
-----Function rec_fact is called-----
z = 3 is not equal to 0, so doing recursive call
-----Function rec_fact is called-----
z = 2 is not equal to 0, so doing recursive call
-----Function rec_fact is called-----
z = 1 is not equal to 0, so doing recursive call
-----Function rec_fact is called-----
z = 0 is equal to 0
Factorial of 3 = 6
```

Recursion – Fibonacci Numbers

- **Fibonacci number f(n) can be defined as:**

```
f(0)  =  0
f(1)  =  1
f(n)  =  f(n-1) + f(n-2),   if  n > 1
```

  - The successive Fibonacci numbers are:

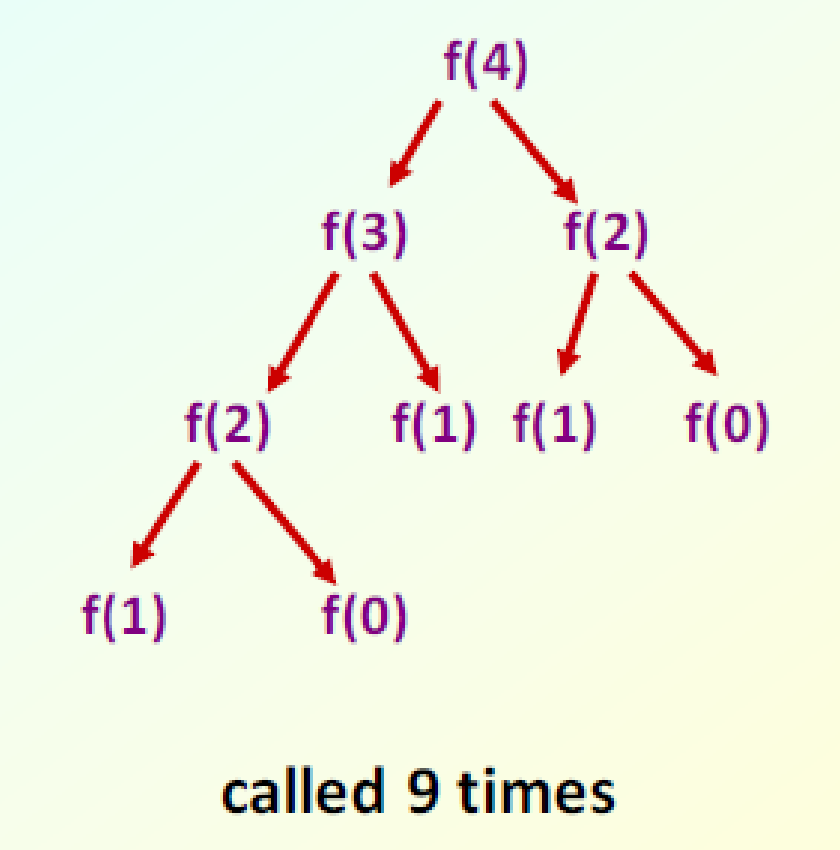
    0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- **Function definition:**

```
int  f (int n)
{
     if  (n  < 2)    return (n);
     else  return (f(n-1) + f(n-2));
}
```

Recursion – Fibonacci Numbers

How many times the function is called when evaluating f(4) ?



Same thing is

calculate multiple

times – inefficient.

Recursion – What to avoid

1. **Avoid Fibonacci-style recursive programs which result in an exponential “explosion” of calls.**
2. **Avoid using recursion in performance situations.**
3. **Recursive calls take time and consume additional memory.**

Recursion vs Iteration

Recursion vs. Iteration

- **Repetition**
  - Iteration: explicit loop
  - Recursion: repeated function calls
- **Termination**
  - Iteration: loop condition fails
  - Recursion: base case recognized
- **Both can have infinite loops**
- **Balance**
  - Choice between performance (iteration) and good software engineering (recursion).