ALX LESSON

0x08 C - Recursion

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

C
Programming
Topics

What is recursion

How to implement recursion

In what situations you should implement recursion

In what situations you should not implement recursion

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

@ALX_2023

02

Learning Objectives

What is recursion

What is recursion

Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem.

What is recursion

A recursive function is a function that calls itself.

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

What is the factorial of 4 ?

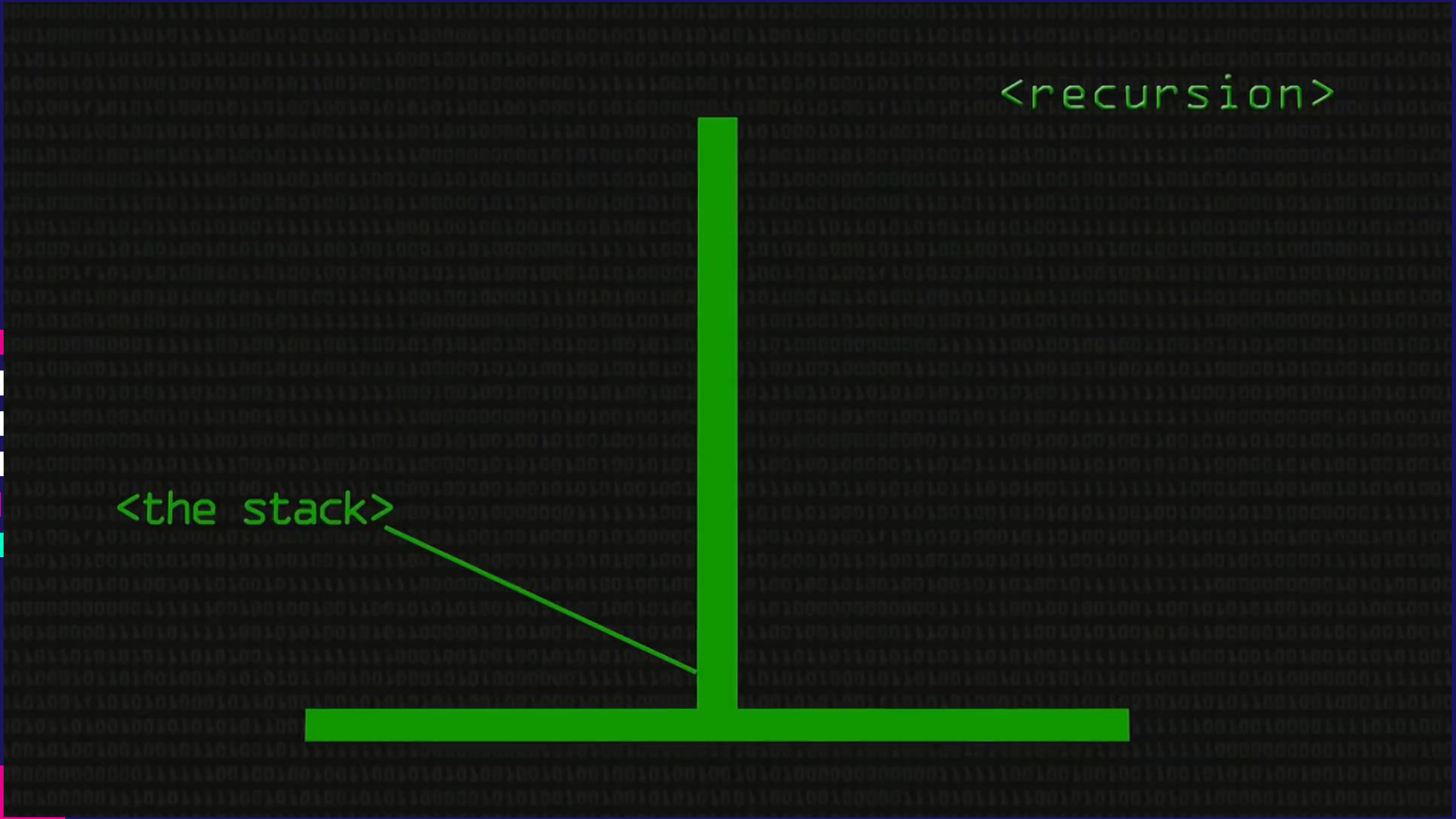
Iterative with factorial

```c
#include <stdio.h>

int main() {
    int num = 4;
    int fact = 1;

    for(int i = 1; i <= num; i++) {
        fact *= i;
    }

    printf("Factorial of %d is %d\n", num, fact);
    return 0;
}
```

Recursion with factorial

```c
#include <stdio.h>

int factorial(int n) {
    if (n < 0) {
        return -1;
    } else if (n == 0) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}

int main() {
    int n = 4;
    int result = factorial(n);

    if (result == -1) {
        printf("Error: input must be non-negative\n");
    } else {
        printf("%d! = %d\n", n, result);
    }

    return 0;
}
```

How to implement recursion



Iterative with alphabet

```c
#include <stdio.h>

int main() {
    char c;

    printf("Alphabet: ");
    for (c = 'a'; c <= 'z'; c++) {
        printf("%c ", c);
    }

    return 0;
}
```

Recursion with alphabet

```c
#include <stdio.h>

void print_alphabet(char letter)
{
    if(letter > 'z') // Base case: stop recursion when letter exceeds
'z'
        return;

    printf("%c ", letter); // Print the current letter
    print_alphabet(letter+1); // Recursive call with the next letter
}

int main()
{
    char start_letter = 'a';
    print_alphabet(start_letter);
    return 0;
}
```

In what situations you should implement recursion

Recursion is commonly used in the following situations:

1. When you want to solve a problem that can be divided into smaller sub-problems of the same kind.

2. When you want to explore all possible paths in a tree-like structure, such as a search or traversal of a data structure.

3. When you want to solve a problem where the solution depends on the solutions of one or more smaller instances of the same problem.

4. When you want to implement backtracking algorithms, which explore all possible paths in a search space to find a solution.

In what situations you should implement recursion

Some implementation examples where recursion can be useful:

- Calculating factorial of a number
- Finding the nth Fibonacci number
- Binary search in a sorted array
- Traversing a binary tree
- Finding the greatest common divisor of two numbers
- Reversing a linked list
- Depth-first search of a graph
- Tower of Hanoi problem

In what situations you should not implement recursion

Here are some situations where recursion may not be the ideal solution:

Limited stack space: Recursion requires a lot of memory allocation on the stack, which can lead to stack overflow if not managed properly. In certain environments, such as embedded systems with limited stack space, recursion may not be a feasible option.

Recursion may not be a good fit for problems that involve a large number of recursive calls or deep recursion. This can lead to stack overflow errors or excessive memory usage.

For example, problems that involve iterating over large datasets or complex algorithms may not be a good fit for recursion.

Additionally, problems that require tail recursion optimization or do not have a clear base case may also not be well-suited for recursion. However, recursion can be a good fit for problems that involve smaller datasets, tree or graph traversal, and certain mathematical or combinatorial problems.

What is the ASCII character set

```
cook@pop-os:~$ ascii -d
    0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

Hexadecimal Numbering System

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

04

Hands on lab Practice

Have a Question
Leave a Comment!

Subscribe

To stay updated with latest videos

Share

To let the others know more

Thanks