10/ /2023

Recursion

When a function calls itself directly or undirectly is known as recursion. This is the

bookish language.

function calling itself

void solve() {

Solve ()

When solution of bigger problem is depending on smaller & same type problem, then here recursion will be applied. This is the way by Which we can tell whether recursion will applied or not.

Ext Solve 25 by recursion

Bigger problem

f(n) = 2n

f(n-1) = 2n-1

 $\frac{f(v)}{f(v)} = 3 \times f(v-1)$

function calling itself

| $-\epsilon_{\infty}$ | Solve 51 by recursion. | | | |
|----------------------|---|--|--|--|
| | 5] = 5 X 4] | | | |
| | 4) Bigger 4 Smaller | | | |
| 5 4 | problem problem | | | |
| | Hence recursion can be applied | | | |
| | | | | |
| _ E2C → | Reverse counting n=5 => 5 4 3 2 1 | | | |
| | n=5 => 5 4 3 2 1 | | | |
| | A (= 2) | | | |
| | f(5) = print from 5 to 1 | | | |
| Tall! | 1 (5) Print from 4 to 1 | | | |
| | f(5) = print(5) + f(4) | | | |
| - hy | → Bigger / L→ Smaller | | | |
| 9:0 | f(5) = print (5) + f(4) 4 Bigger Smaller problem ignore problem (just for understanding) | | | |
| | (just for understanding) | | | |
| | Whenever we will be given a problem Statement 4 if we are able to find a | | | |
| | Statement 4 15 we are able to find a | | | |
| | Lormwal or relation in which live | | | |
| | problem then 100 | | | |
| C~ \ | care apply recursion. | | | |
| $\in x \rightarrow$ | Example to understand recursion | | | |
| | Source H 10 steps | | | |
| | 1step Destination | | | |
| | Hence we can divide losteps into one one | | | |
| | steps each & hence recursion can be | | | |
| | applied here. | | | |
| | | | | |
| (x → | Fibonacci Series | | | |
| | | | | |
| | 0 1 2 3 5 8 13 21 | | | |
| | 8 13 21 | | | |

Scarineu with Am

| = | |
|----|--|
| | Here nth term = (n-1)th term + (n-2)th |
| | team |
| 4 | |
| _ | $(n-1)$ th term $\rightarrow f(n-1)$ |
| | $(n-2)^{th}$ term $\Rightarrow f(n-2)$ |
| _ | $f(n) = f(n-1) + f(n-2) \int Relation$ |
| -1 | 1 |
| | function calling itself |
| | Hence recursion is applied for fibonacci series |
| | also. |
| | in transaction to the second of the second o |
| | Understanding recursion by code |
| _ | |
| _ | int factorial (int in)? // Base case |
| _ | // Duse case |
| | // Recursive relation |
| _ | 3 |
| _ | Our recursive code will have 3 components |
| _ | namely base condition, recursive relation 4 |
| _ | processing. |
| 1 | Base case will tell the recursion to stop. |
| 1 | AL: monthspent |
| 2) | It is a mandatory component Recursive relation is also mandatory component. Processing is like sometimes we have to print the values but it is not a mandatory component. |
| 32 | Processing is like sometimes we have to print |
| _ | the values but it is not a mandatory component |
| _ | |
| \ | Code of factorial using recursion int factorial (int n) { |
| \ | int factorial (int n) 2 |
| \ | |

| | Page | | | |
|----------------|--|--|--|--|
| | //Base case | | | |
| Jt', | if(n = = 0 n = = 1) | | | |
| All J. | return 1 | | | |
| | int ans = n * factorial (n-1) // | | | |
| | return ans : Recursive relation | | | |
| | 3 | | | |
| | | | | |
| Note: | e > 9 n most of cases, return keyword is | | | |
| | Used in the base case. | | | |
| | | | | |
| - Aug 1 | Recursive call Stack | | | |
| | - Carsive Car Scacio | | | |
| | void brint Counting (int n) { | | | |
| ~ | Void print Counting (int n) { // Base Case | | | |
| × | if(n = = 0) | | | |
| | retwin | | | |
| | 1/ Processing | | | |
| | cout << n << " "); | | | |
| | // Recursive Relation | | | |
| | print Counting (n-1)i | | | |
| 2.1. | THE SHELL RESIDENCE OF THE PARTY OF THE PART | | | |
| 4 41 | 31 | | | |
| | Stack is basically like plates in the marriage Stack follows LIFO i e last in first out. | | | |
| | Stack follows LIFO i e last in fixet out. | | | |
| d | LE TILLET IN LANGE TO THE STITE OUT | | | |
| | 0/b - 5 4 3 2 1 | | | |
| e (8, 1, 1, 1) | Condition PC(5) | | | |
| ¥ . | satisfied PC(T)= P((4) | | | |
| 3 2. 1. | P(3) | | | |
| | P((3))√ P((2) ← | | | |
| | PC(1) | | | |
| | PC(5)2 | | | |
| | : main()? | | | |
| | | | | |

Scarineu with dam

| | A 11 1 | | | | |
|--------|---|--|--|--|--|
| | In the function call Stack, 1st entry will be | | | | |
| | In the function call stack, 1st entry will be of main function always. As the base case matches the entry of function in stack is removed & control will go to the previously | | | | |
| | matches the entry of function in stack is | | | | |
| | removed & control will go to the breviously | | | | |
| | called function. | | | | |
| | | | | | |
| Note - | When the base case is matched & as the | | | | |
| | return keyword is encountered, the control | | | | |
| | will go to the next line of the call of | | | | |
| | previously called function. | | | | |
| | | | | | |
| | Recursion by Tree | | | | |
| | main() | | | | |
| | | | | | |
| | P((5) 5 | | | | |
| | | | | | |
| | P((4)) 4 | | | | |
| _ | · J | | | | |
| _ | PC(3) 3 | | | | |
| _ | 2000 2 | | | | |
| _ | P((2)) | | | | |
| | $P((1) \rightleftharpoons 1$ | | | | |
| | PC(1) (1) | | | | |
| | P((0) | | | | |
| | | | | | |
| | Recursive call Stack of factorial function | | | | |
| | (activitied) 1 returned | | | | |
| | Jactoriax | | | | |
| | factorial (3) = 2 retwined factorial (3) = 6 retwined | | | | |
| | , and the said | | | | |
| | factorial (4) = 14x6 = 24 returned to | | | | |
| | main () n x factorial (4) = 5 x 24 | | | | |
| | n x fadorio. Ch- sx- | | | | |

| 31 | Function will be popped out once it is | | | | |
|---------------------------------------|---|------------|--|--|--|
| | finished (returned the value). | | | | |
| | a santa a martinal transfer oil a manual | | | | |
| Note- | Whenever a function is called, its | | | | |
| | entry is bushed into the call stack so | | | | |
| | as to track it. | | | | |
| | and was borded and of machine | | | | |
| , , , , , , , , , , , , , , , , , , , | Tree inti donotnima stubronima in | | | | |
| | fact (5) ← 120 | | | | |
| | Land to the state of the state | | | | |
| | 5 × fact(4) € | 24 | | | |
| _ | 4 × fact (3) < 6 | | | | |
| | J_{μ} | 43) | | | |
| | 3 × f | act(2) 2 | | | |
| | | 1 | | | |
| | 2 PC . 4.7 | 2× fact(1) | | | |
| | Head and Tail recursion | | | | |
| | | | | | |
| | | | | | |
| | void solve () { | | | | |
| | //Base case Tail | | | | |
| | 1/ Processing Rewision | | | | |
| | 1/Recursive Relation | | | | |
| | 3 | | | | |
| | Void solve () { | THE MILES | | | |
| | //Base case | 11 | | | |
| | 1/2 Recursive Rela | Head | | | |
| | 11 Processing | | | | |
| | 3 | | | | |
| - | | | | | |
| | | | | | |

Scarifica with cam

| | In tail recursion, processing is done before calling of function whereas in head recursion processing is done after the recursive relation. |
|----|---|
| | calling of function whereas in hard- |
| | brocessing is done after the mead recursion |
| | of the recursive relation. |
| 2× | > Write code for brinting and in |
| | → Write code for printing counting from I to n. |
| | void count (int n) { |
| | //Base Case |
| | if(n=0) |
| | returni |
| | // Recursive Relation |
| | count (n-1); |
| | // Processing |
| | cout << r ; |
| | 3 |
| | The above code is of the head recursion. This |
| | will print counting from I to n as whenever |
| | we encounter return in base case, we move to |
| | the next line of the calling of function. |
| _ | (E) dit |
| | Understanding recursion of fibonacci series |
| _ | We have already derived the relation i.e. |
| _ | We have already derived the relation i.e |
| \ | But the question comes to our mind that |
| \ | What will be the base case. |
| | pand term |
| | 0 1 2 3 5 8 |
| | Ist term |
| 1 | Ist term |
| 1 | (= - 1 -) SW(1) = - |
| 1 | Base case. |
| | We have selected this as the base case as |



| 57 | behind them there are no 2 numbers. Hence | | | |
|--------|---|--|--|--|
| 0.1 | we handled this case explicitly. | | | |
| · mai | Commence with the commence of | | | |
| | Code | | | |
| n ed | I were pairtabox paddind and Babas alakis | | | |
| | int fib (int n) { | | | |
| | //Base case | | | |
| | if $(n = = 1)$ // 1st term | | | |
| | return Oi | | | |
| | if (n = = 2) //2nd term | | | |
| | retwin 1; | | | |
| | //Recursive Relation | | | |
| | int ans = $fib(n-1) + fib(n-2)$; | | | |
| | return ans | | | |
| | 3. | | | |
| | Tree mixer bear and be signed | | | |
| · 16th | 2+1=3 | | | |
| 210 | fib(5) | | | |
| | 1+1=2 | | | |
| | fib(4) < fib(3) | | | |
| 1_0,1 | 1 | | | |
| | $\begin{array}{cccccccccccccccccccccccccccccccccccc$ | | | |
| | fib(3) 0 fib(2) fib(2) fib(1) | | | |
| (| C'I (I) | | | |
| | fib(2) fib(1) | | | |
| | | | | |
| | This is how own algorithm works. First | | | |
| | This is how own algorithm works. First n-1 call will go & then n-2 call will | | | |
| | go. | | | |
| | | | | |
| | fib(2) = 1 2 Base (ase fib(1) = 0 | | | |
| - | FIB(I) = O J | | | |
| Sale | 5% - 38x 1 - 11 &2 3x71 25 3 1 1 1 1 1 1 1 1 | | | |

Scarnieu with dam

Vote > In our entire coding journey, we don't have to see whether head or tail recursion will be applied. It will just be dependent on the problem statement. Also note that the call which is encountered first gets executed. 3+2 = 5 Ans n = 6fib (6) fib(4) 5 7 fib(5) fib(3) ~ fib(3) a fib (4) fib(2) 1+0=1 f16(1) fib(2) fib() fib(3)f b(2) fib(1) The above is the tree for n=6 & we can say that 6th term is 5. Also note that the nodes which are at the end are the calls in which base case condition is Satisfied. Recursion call stack and tree are just for the understanding burpose.

Classmate

Date
Page

| \rightarrow | Very | important | line |
|---------------|------|-----------|------|
|---------------|------|-----------|------|

MAGICAL LINE

Just solve I case and rest recursion

will handle.

Suppose that we need to find 5! & lets assume that 4! is known as recursion has given the answer. So simply multiply 5 & 4! and hence we get the answer.

5 x (4 !

We don't have to go in depth of the magical