C Programming Lecture 13

Sunday, 23 June 2024 6:54 PM

Static Functions

```c
static void func() {
    printf("Hello World!");
}
```

file1.c

```c
int x(){
}
y();
```

file2.c

```c
#include "file1.c"
```

x() ✓
y() ✓
z() X

Non-static functions are global by default means that the function can be accessed outside the file also, but if we declare the function as static, then it limits the function scope. The static function can be accessed within a file only.

```c
static int z(){
}
```

Example (GATE CS 2023)

The integer value printed by the ANSI-C program given below is _____

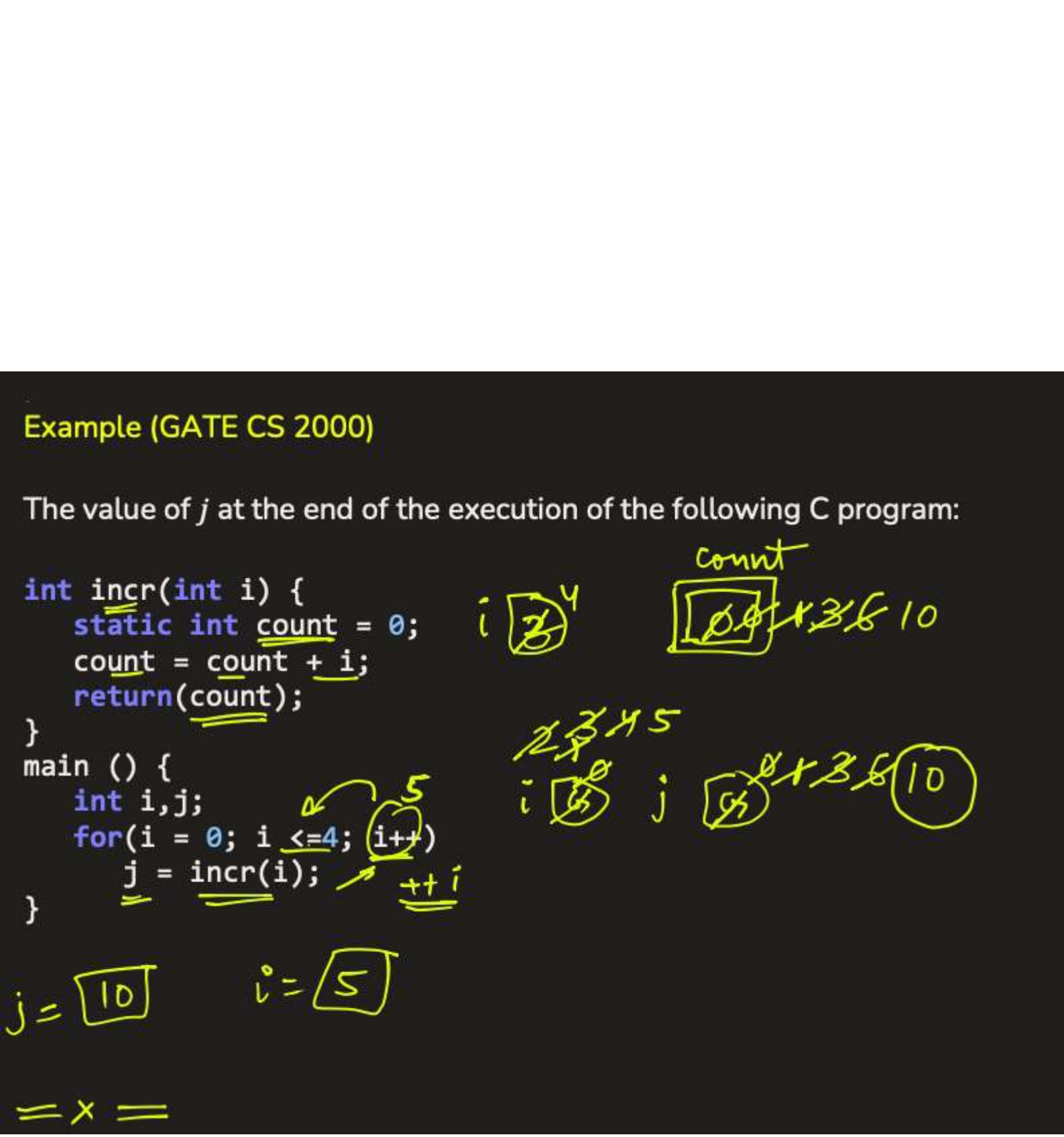
```c
#include<stdio.h>

int funcp(){
    static int x = 1;
    x++;
    return x;
}

int main(){
    int x,y;
    x = funcp();
    y = funcp()+x;
    printf("%d\n", (x+y));
    return 0;
}
```

$y = 3 + x^2;$

Example (GATE CS 2000)

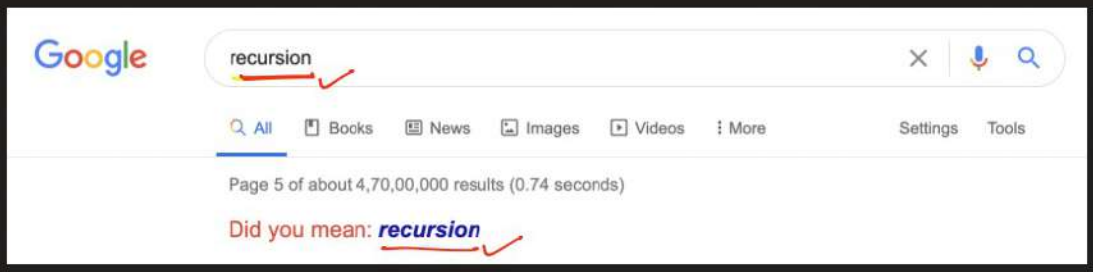
The value of *j* at the end of the execution of the following C program:

```
int incr(int i) {
    static int count = 0;
    count = count + i;
    return(count);
}
main () {
    int i,j;
    for(i = 0; i <=4; i++)
        j = incr(i);
}
```



Recursion in C → Function calling itself

- A problem can be split into several problems of same kind, but simpler ones.
- Allows breaking of complex tasks into simpler problems.
- For eg, suppose you have a box full of 5 rupee coins and you have to count how much money you have. To save some time, you can ask a friend to help and split the task. When you both finish counting, you can aggregate the results.



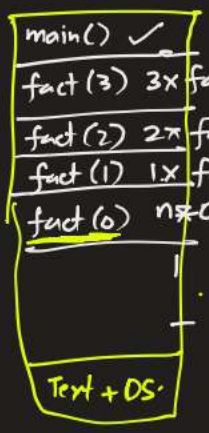
- Recursive functions are the functions that call themselves. They have two main parts:
  - **General (Recursive) case**: Here the problem space is made smaller and smaller
  - **Base case**: The case for which the solution can be stated directly.

```
fact(n){          Base case
    if(n==0) return 1;
    return n*fact(n-1);
                  Recursive case
}
main(){
→fact(3); (6)
}
```

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

$$f(n) = \begin{cases} n \times f(n-1), & \text{if } n > 0 \rightarrow \text{Recursive case} \\ 1, & \text{if } n = 0 \rightarrow \text{Base case} \end{cases}$$

Recurrence relation



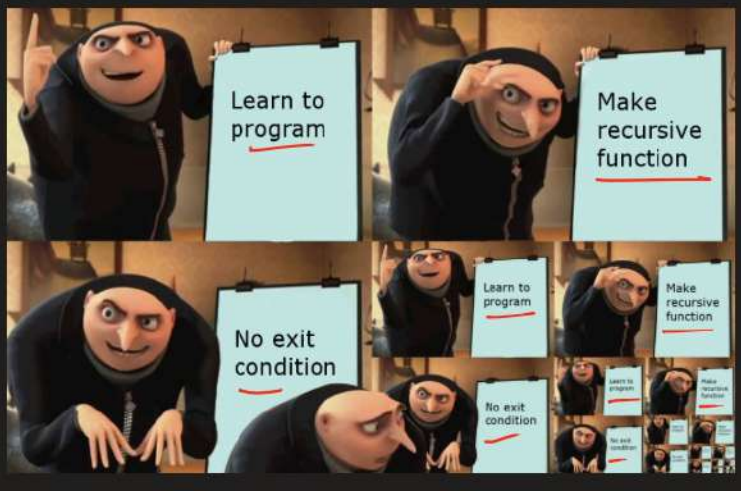
Segmentation fault

Mem. exceeded

Let's take an example of calculating factorial using recursion,
n! = (n) * (n-1) * (n-2) * (n-3) .... 2 * 1
We can generalize this expression,
n! = (n) * (n-1)!

$$n! = \begin{cases} 1 & \text{if } n = 0 \text{ (base case)} \\ n * (n-1)! & \text{if } n > 0 \text{ (recursive case)} \end{cases}$$

- Key differences b/w Recursion & Iteration:
  - **Termination**: In case of recursion, the termination is decided by the base case whereas in iteration, termination depends upon the value of control variable.
  - **Memory**: Recursion takes more memory as for every function call it occupies a memory block in the stack, whereas iteration doesn't requires any extra storage.
  - **Execution Time**: Iteration executes faster than recursion as there is no function calls and return statements.
  - **Code Complexity**: Recursive code is smaller and simples as compared to the iterative counterpart.



Example

- Factorial using recursion

```c
#include<stdio.h>
unsigned long long fact(int n){
    if(n==0)    return 1; // Base Case
    return n*fact(n-1); // Recursive Case
}
int main(){
    int n = 20;
    printf("%llu", fact(n));
    return 0;
}
```

- Factorial using iteration

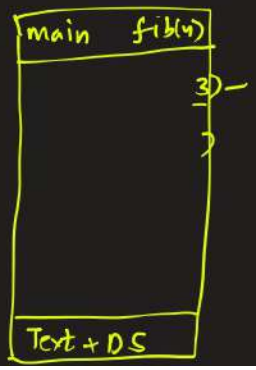
```c
#include<stdio.h>
int main(){
    int n = 20;
    unsigned long longans = 1;
    for(int i=1; i<=n; i++)
        ans = ans*i;

    printf("%llu", ans);
    return 0;
}
```

}

Example

- nth Fibonacci number using recursion

```
#include<stdio.h>
unsigned long long fib(int n){
    if(n==1)    return 0; // Base Case
    if(n==2)    return 1; // Base case
    return fib(n-1) + fib(n-2); // Recursive Case
}
int main(){
    int n = 50;
    printf("%llu", fib(n));
    return 0;
}
```



fib(4)

How many activation records for fib() get inserted in the stack?

Recursion Tree (5)

fib(4)

1st	2nd	3rd	4th	5th	6th
0	1	1	2	3	5

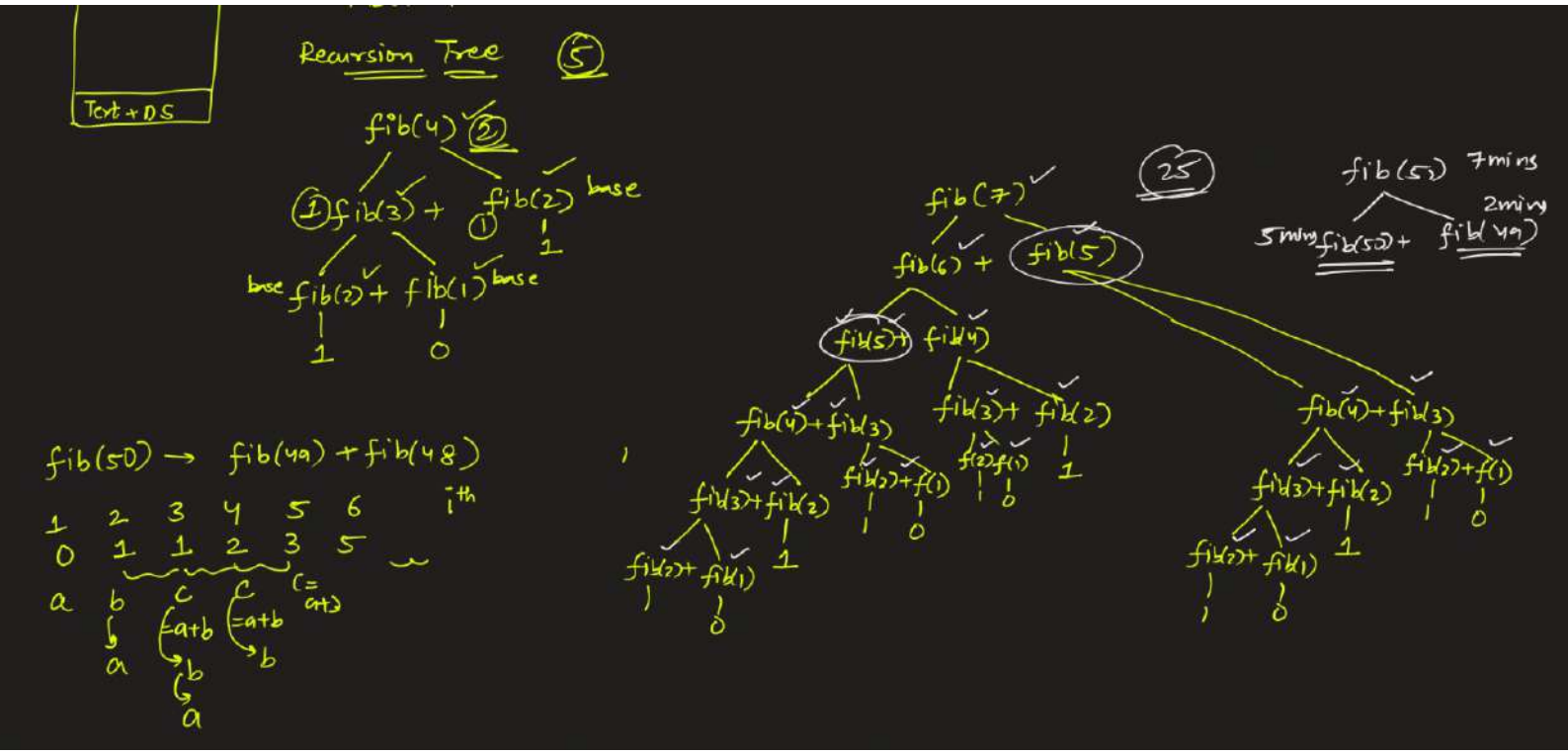
recursive case

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2), & n > 2 \\ 1, & n = 2 \\ 0, & n = 1 \end{cases}$$ base case

20th fibonacci number ?

30th fibonacci number ?

40th


Text + DS
Recursion Tree (5)
fib(4) (2)
(1) fib(3) + (1) fib(2) base
1
base fib(2) + fib(1) base
1 0
fib(50) → fib(49) + fib(48)
i th
1 2 3 4 5 6
1 0 1 1 2 3 5
a b c c (= a+b)
a
a+b
b
a+b
b
a
fib(7)
25
fib(6) + fib(5)
fib(5) + fib(4)
fib(4) + fib(3)
fib(3) + fib(2)
fib(2) + fib(1)
1
0
fib(3) + fib(2)
1
fib(2) + fib(1)
1
0
fib(3) + fib(2)
1
fib(2) + fib(1)
1
0
fib(4) + fib(3)
fib(3) + fib(2)
1
fib(2) + fib(1)
1
0
fib(5) 7 mins
2 mins
5 mins fib(50) + fib(49)

- nth Fibonacci number using iteration

```c
#include<stdio.h>
int main(){
    int n = 50;
    unsigned long long a = 0, b = 1, ans;
    if(n==1)     ans = a;
    if(n==2)     ans = b;
    for(int i=3; i<=n; i++) {
        ans = a+b;
        a = b;
        b = ans;
    }

    printf("%llu", ans);
    return 0;
}
```

}

=x=

Example

```
int f(int x, int y) {
    if (x == 0)
        return y;
    else
        return f(x - 1, x + y);
}
```

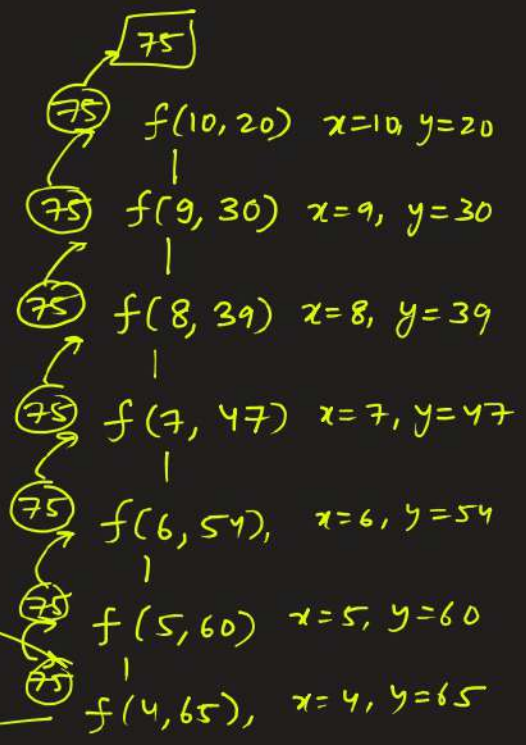
What will the above function return for f(10, 20) ?

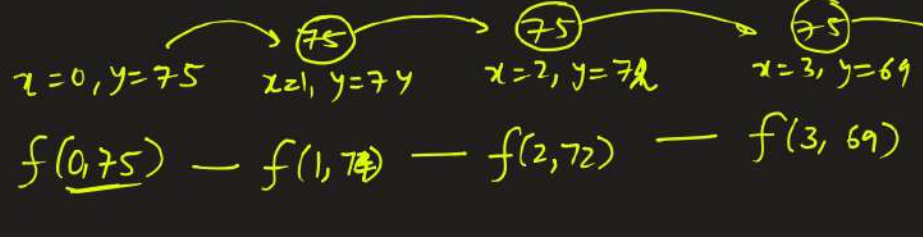
a) 45
b) 55
c) 65
d) 75

$(1+2+3+\dots+10)+20$ ✓

$\frac{10(11)}{2} + 20$ ✓

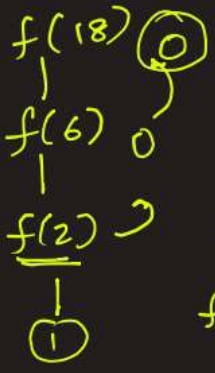
$= 75$ ✓

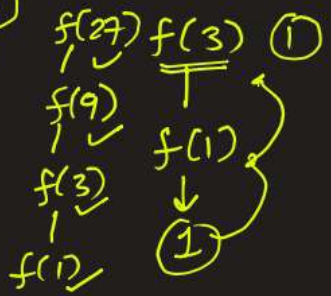


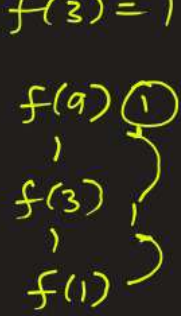


Example

```
int f(int x) {
    if (x == 0 || x == 1)
        return x;
    if (x%3 != 0)
        return 0;
    else
        return f(x/3);
}
```

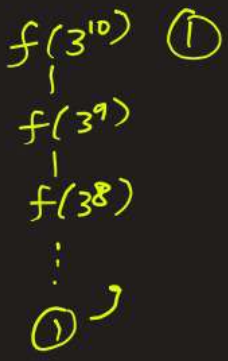


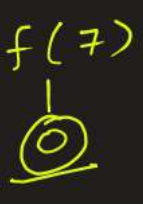


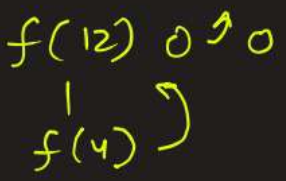


Choose the correct option for the above function?

a) Returns 1 when n is a power of 3, otherwise 0
b) Returns 0 when n is a power of 3, otherwise 1
c) Returns 0 when n is a power of 3, otherwise 0
d) Returns 1 when n is a power of 3, otherwise 1







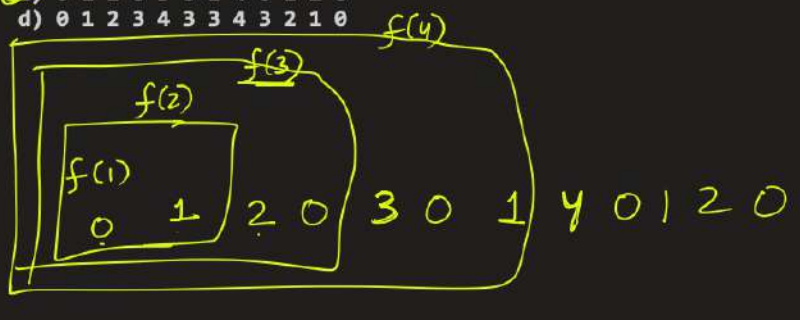
Example

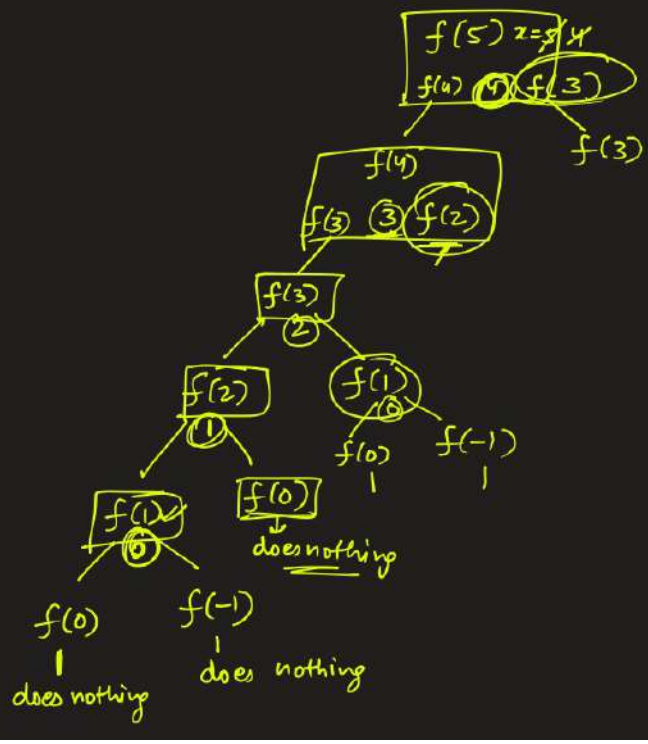
```
void f(int x) {
   if(x > 0)
   {
      f(--x);
      printf("%d", x);
      f(--x);
   }
}
```

f(1) x=1
f(0)
print(0)
f(-1)

What will be the output of the above function call for f(5) ?

a) 0 1 2 3 4 3 2 1 0 1 2 3
b) 0 1 2 3 0 1 2 3 0 1 2 3
c) 0 1 2 0 3 0 1 4 0 1 2 0
d) 0 1 2 3 4 3 3 4 3 2 1 0





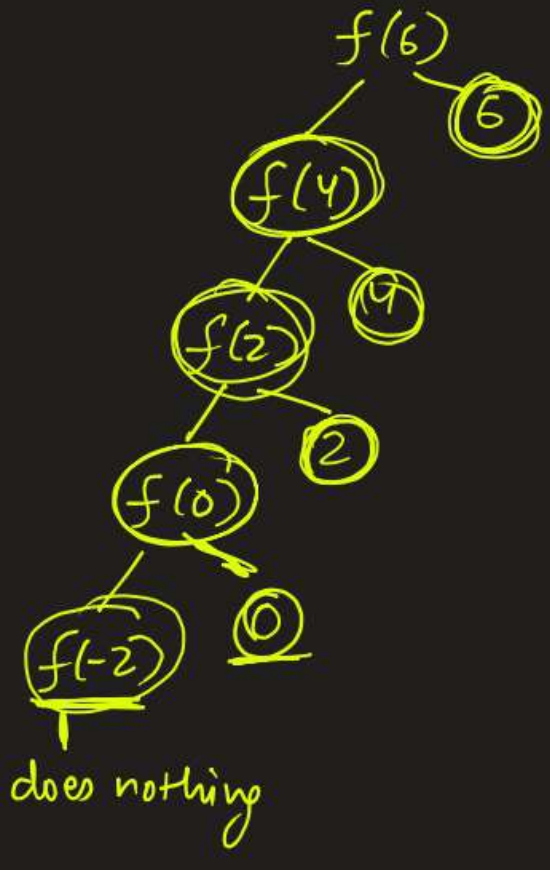
Example

```
int f(int x) {
    if (x < 0)
        return;
    f(x-2);
    printf("%d", x);
}
```

What will the above function return for f(6) ?

a) 2 4 6
b) 6 4 2
c) 6 6 6
d) None of the above

0 2 4 6



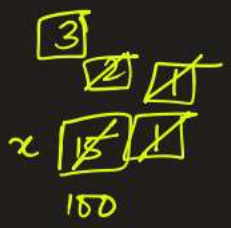
Example

```c
#include<stdio.h>

int fun(int n, int* fp)
{
    int t, f;

    if(n <= 2) {
        *fp = 1;
        return 1;
    }
    t = fun(n - 1, fp);
    f = t + *fp;
    *fp = t;
    return f;
}

int main()
{
    int x = 15;
    printf("%d\n", fun(5, &x));

    return 0;
}
```

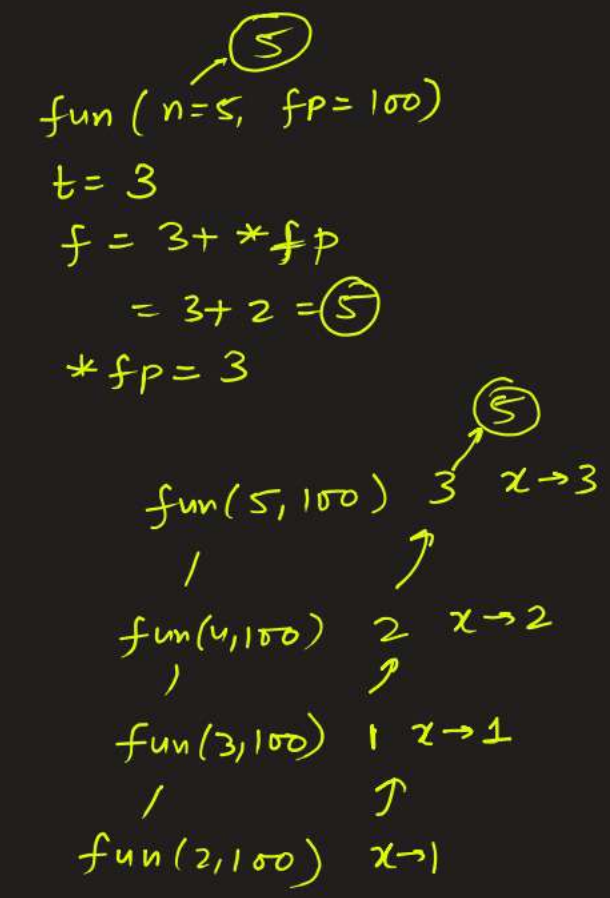
base

t (4) f (7)

(5)

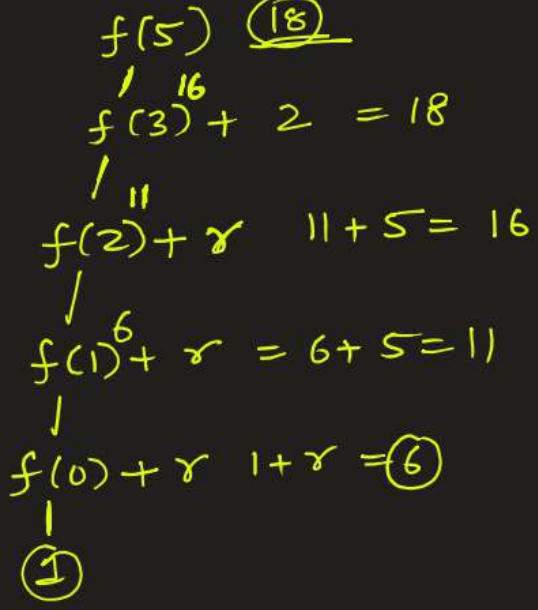


5 ✓



GATE CS 2008

```
int f(int n)
{
   static int r = 0;
   if(n <= 0) return 1;
   if(n>3)
   {
      r = n;
      return f(n-2)+2;
   }
   return f(n-1)+r;
}
```



What is the value of f(5)?

(A) 5

(B) 7

(C) 9

(D) 18