Let’s solve it

Recursion

A function calling itself.

Iterative

$5! \Rightarrow 5 \times 4 \times 3 \times 2 \times 1$

```
result=2;
for (i=3; i<=n; i++)
{
    result = result * i;
}
```

Recursive

$n! \Rightarrow n * (n-1)!$

$5 \times 4!$

$4! \Rightarrow 4 \times 3!$

$3! \Rightarrow 3 \times 2!$

$2! \Rightarrow 2 \times 1!$

$1! \Rightarrow 1$

Terminating condition.

without terminating condition
it will be infinite loop.

$\sum_{i=1}^{n} i \Rightarrow n + \sum_{i=1}^{n-1} i$

$\Leftarrow 1 \Rightarrow 1$ //

?

GNU $\Rightarrow$ GNU is Not Unix

? ~~iterating~~

w/o ~~termination~~ condition — X

Iterative way

formal arg

```
int iterativeFact(int n)
{
    int result = 1;
    for(i=1; i<=n; i++)
    {
        result = result * i;
    }
    return(result);
}
```

actual arg

```
int main()
{
    int r;
    r = iterFact(5);
    printf("%d", r);
}
// printf("%d", iterFact(5));
```

What is the advantage of recursive solution, whenever possible?

It is programmer's friendly.

printing only n^{th} term of fibonacci

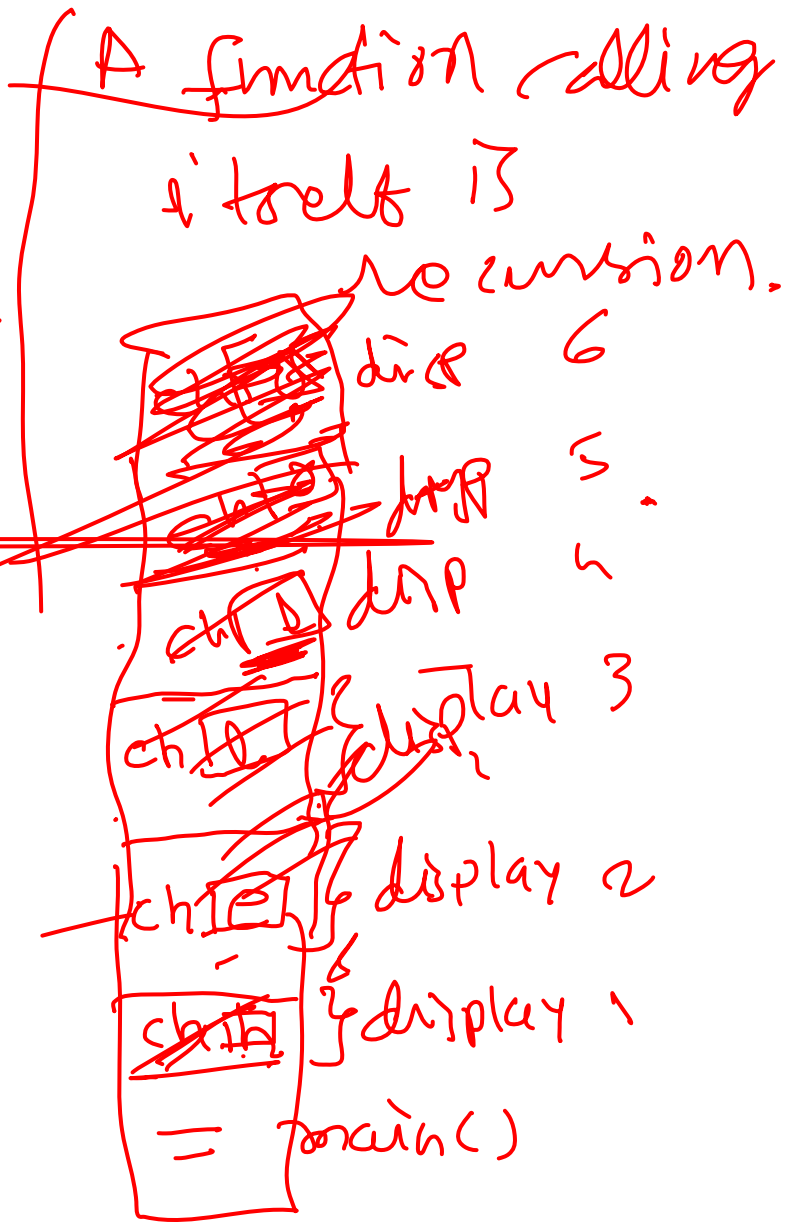
fib(n) ⇒ fib(n-1) + fib(n-2) || p.s. fn and back ?

fib(1st) ⇒ _

fib(2nd) ⇒ _

```
display()
{
  char ch;
  ch=getchar();
  if(ch==EOF)
    return;
  else
    display();
  putchar(ch);
}
```

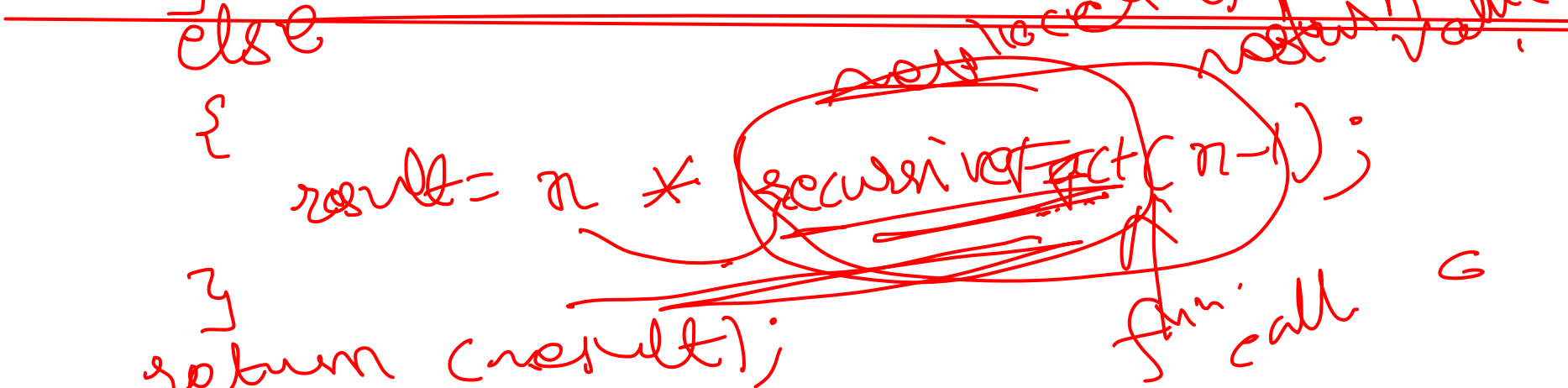
A function calling itself is recursion.

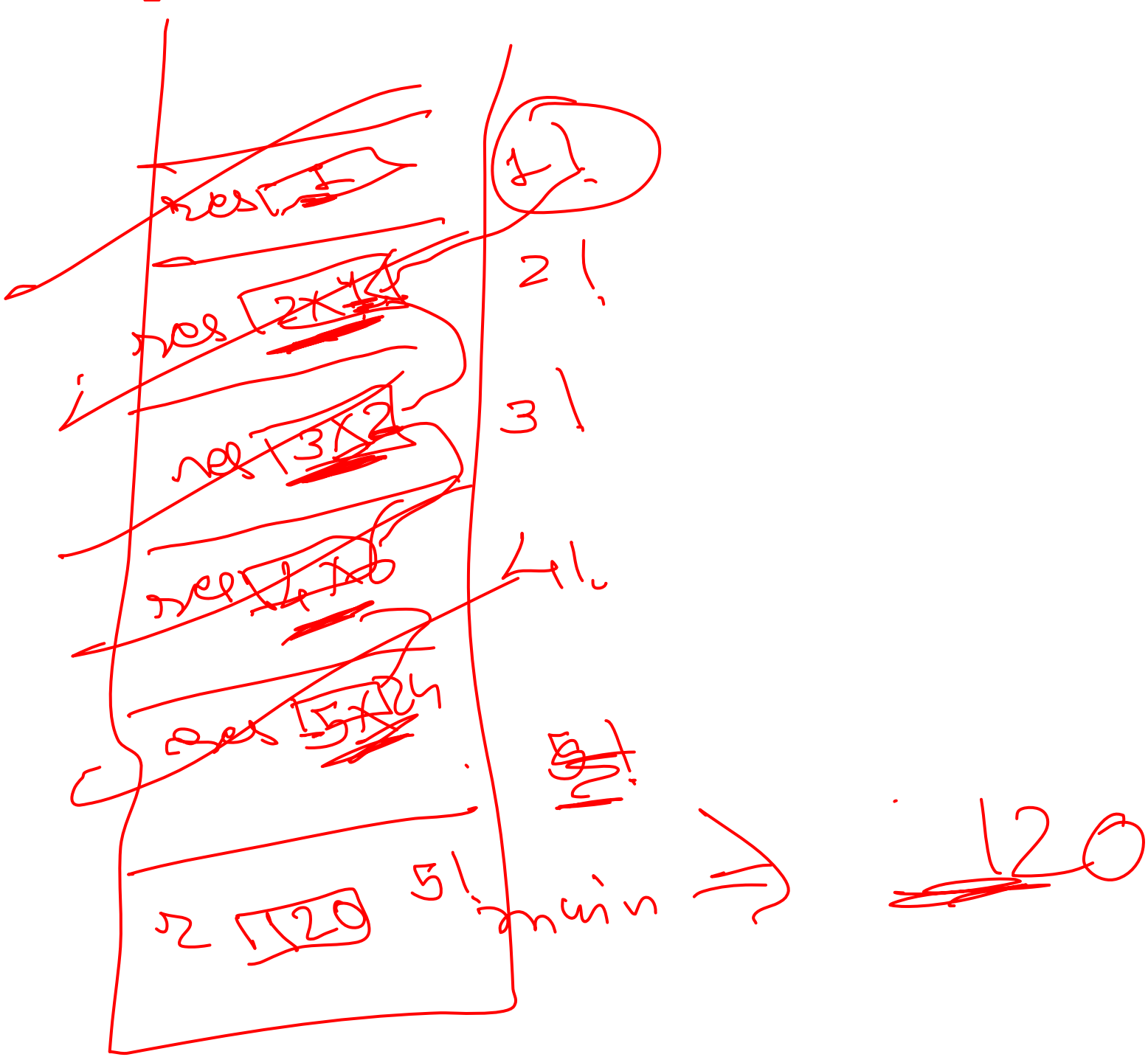


hello

output: olleh

```
int recursiveFact(int n)
{
    int result;
    // terminating condition
    if (n == 1)
    {
        result = 1; // return(1);
    }
    else
    {
        result = n * recursiveFact(n-1);
    }
    return (result);
}
```




res = 1
1!
res = 2 * 1
2!
i
res = 3 * 2
3!
res = 4 * 6
4!
res = 5 * 24
5!
res = 120
5! main => 120