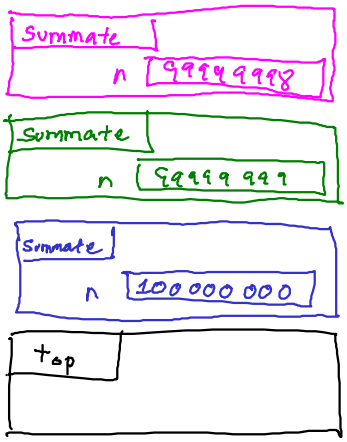
```
let rec summate (n:int) : int =
    if n=0 then 0 else n + summate (n-1)
;;
summate 100000000
```

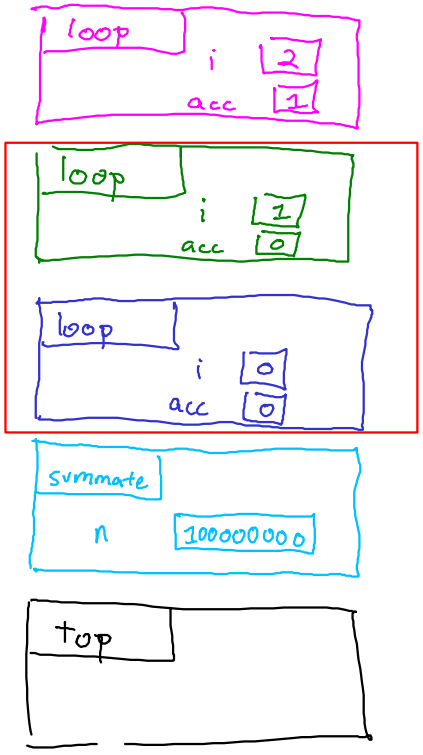


summate 100000000

100000000 + summate 99999999

100000000 + 99999999 + if 99999998=0 then 0 else 99999998+ summate (99999998 -1)

```
let summate (n:int) : int =
    let rec loop (acc:int) (i:int) : int =
        if i=n+1 then acc else
            loop (acc+i) (i+1)
    in
    loop 0 0
;;
summate 100000000
```



allocated memory which will never be used again

```
let rec loop (acc:int) (i:int) : int =
    if i=100000000+1 then acc else
        loop (acc+i) (i+1)
in
if 2=100000000+1 then 1 else
    loop (1+2) (2+1)
```

Normal call

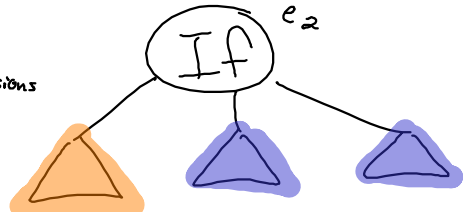
1. Allocate stack memory
2. Do work
3. Recurse
4. Do work
5. Deallocate stack memory

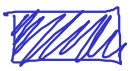
"Tail call" optimization

1. Allocate stack memory
2. Do work
3. Deallocate stack memory
4. Recurse

An expression e_1 is a tail of another expression e_2 iff e_1 is a descendant of e_2, e_1 is the last step of e_2, and the result of e_1 is the result of e_2.

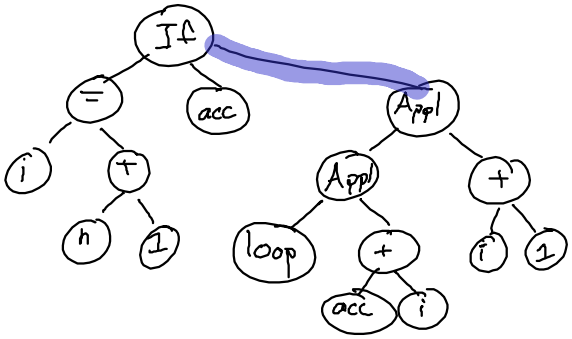
Which subexpressions of Gull expressions are tail expressions of their parent?



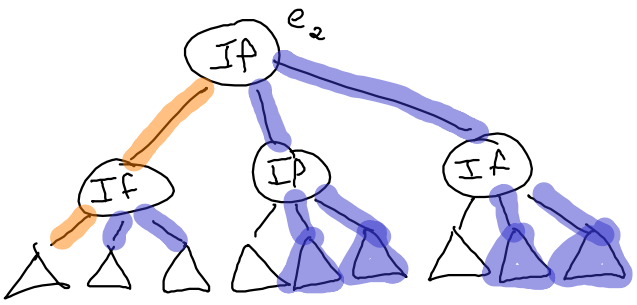

tail expr


not-tail expr

```
type expr =
  | EInt of int
  | EBool of bool
  | EUnaryOp of unary_operator * expr
  | EBinaryOp of binary_operator * expr * expr
  | ETuple of expr list
  | ELet of string * expr * expr
  | EVar of string
  | EIf of expr * expr * expr
  | EAppl of expr * expr
  | ESet of expr * expr * expr
```




```
let rec loop (acc : int) (i : int) : int =
  if i = n+1 then acc else
    loop (acc + i) (i+1)
in
```



A tail call is an application expression which is a tail expression of a function body.