Todays Content

1. Recursion Intro
2. Steps to Recursive codes

Use of Recursion:

Sorting

Trees : BT & BST

BackTracking

Dynamic Programming

Graphs

Rules for Function:

1. Every time a function call is made, it is stored on top of stack
2. When function returns or it's complete execution it will exit stack
3. Even if function return type is void, we can write return;

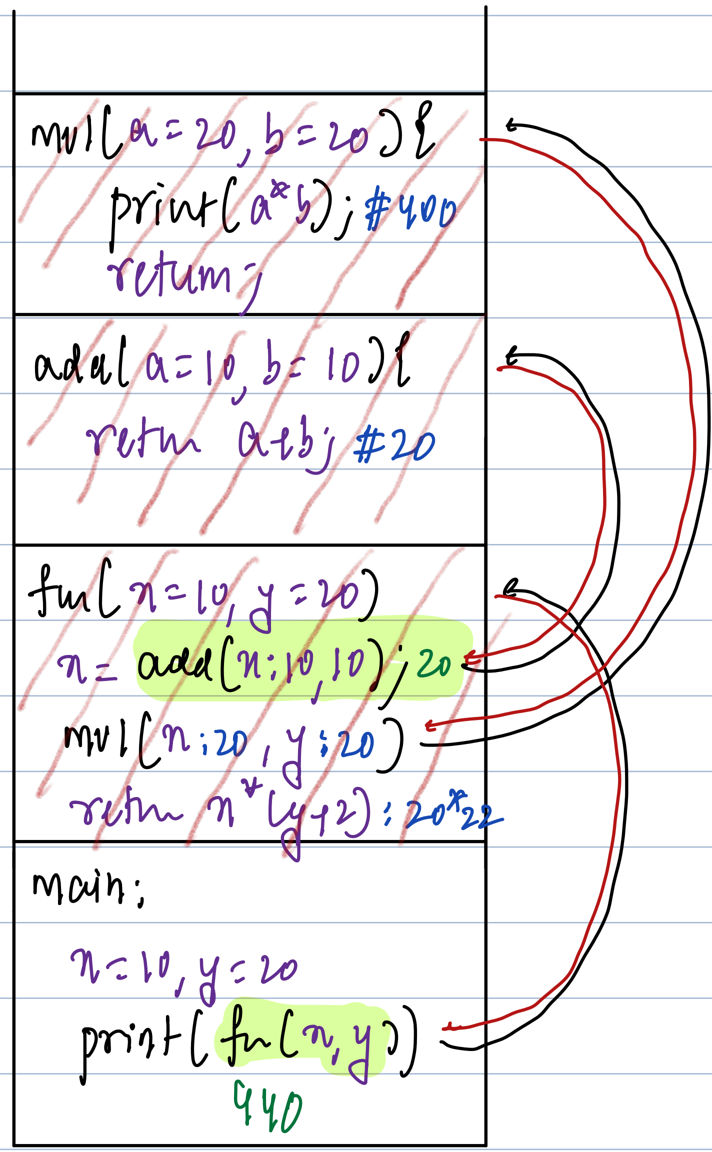
Function Calls

```
void mul(int a, int b){
    print(a*b);
    return;
}

int add(int a, int b){
    return a+b;
}

int fun(int x, int y){
    ✓ x = add(x, 10);
    ✓ mul(x,y);
    return x*(y+2)
}

main(){
    int x=10, y=20; ;
    print(fun(x,y))
}
```



Recursion: Function calling itself

Solving a Problem using SubProblem

Same Problem smaller Input

sum(5) = 1+2+3+4+5

sum(5) = sum(4)+5   # Sub Problem = sum(4)

sum(N) = 1+2+.. N-1 + N

sum(N) = sum(N-1) + N   # Sub Problem = sum(N-1)

Steps to write Recursive Code:

Assumption: Decide what your function does # {input, does, return}

Main logic: Solve problem using Subproblems # Recursive step

Base Condition: Input for which recursion needs to stop

Ass: Given N, calculate & return sum of first N natural numbers

```
int sum(int N){
    if(N==1){ return 1;}
    return sum(N-1)+N
}
```

```
main( ){
  print(sum(4));
}
            10

int sum(N=4){ : a
  if(N==1){ return 1;}
  return sum(N-1) + N
}
          6    + 4

int sum(N=3){ : b
  if(N==1){ return 1;}
  return sum(N-1) + N
}
          3         + 3

int sum(N=2){ : c
  if(N==1){ return 1;}
  return sum(N-1) + N
}
          1         + 2

int sum(N=1){ : d
  if(N==1){ return 1;}
  return sum(N-1) + N
}
```

```
sum(N=1):
  if N==1: return
  return sum(N-1) + N

sum(N=2):
* if N==1: return
  return sum(N-1):1 + N

sum(N=3):
* if N==1: return
  return sum(N-1):3 + N

sum(N=4):
* if N==1: return
  return sum(N-1):6 + N

main()
  print(sum(4)):10
```

Importance of Base Conditions:

```
int sum(int N){
    return sum(N-1) + N;
}
```

If no base conditions codes goto ∞ loop = TLE : Time Limit Exceeded.

Inf: In recursion, function calls are stored on top of stack, before we goto ∞ loop, stack will reach it's limit: stack overflow

MLE: Memory Limit Exceeded

Online Servers:

Time limit = 1sec = 10^8 iterations

Space limit : TODO

Stack limit :

Heap limit :

Array size limit in global vs local :

Issue in below code: #Never reach base condition, always keep it at start

```
int sum(int N){
    return sum(N-1) + N;
    if(N==1){ return 1;}
}
```

fact(5) = 1*2*3*4*5

fact(5) = fact(4)*5   # Sub Problem = fact(4)

fact(N) = 1*2*..N-1*N

fact(N) = fact(N-1)*N   # Sub Problem = fact(N-1)

Assumption: Decide what your function does   # {input, does, return}

Main logic: Solve problem using subproblems # Recursive step

Have a believe that subproblem will work as per assumption.

Base Condition: Input for which recursion needs to stop

0! = 1.

Ass: Given N, calculate & return N!

```
int fact(int N){
    if(N==0){ return 1;}
    return fact(N-1) * N
}
```

$\underbrace{fact(N-1)}_{(N-1)!} * N$

```
main(){
    int a = fact(3)
}
                6
int fact(N = 3){
  * if(N==0) {return 1}
    return fact(N-1) * N
}
                2
    int fact(N = 2){
      * if(N==0) {return 1}
        return fact(N-1) * N
    }
                1
        int fact(N = 1){
          * if(N==0) {return 1}
            return fact(N-1) * N
        }
                1
            int fact(N = 0){
                if(N==0) {return 1}
                return fact(N-1) * N
            }
```

38 Given N, print all numbers from 1 to N in inc Order

Inc(5) : 1 2 3 4 5

Inc(5) : Inc(4) # 1 2 3 4 5

print(5)

Inc(N) : 1 2 3 4 .. N-1 N

Inc(N) : Inc(N-1)

print(N)

Ass: Given N, print numbers from 1 .. N & return nothing

```
void Inc(int N){
    if(N==0){return;}
    Inc(N-1)
    print(N)
}
```

1 2 3 4 .. N-1 N

```
main(){                Output: 1 2 3
    Inc(3)
}

void Inc( N=3 ){
  * if( N==0) { return; }
  ✓ Inc(N-1)
  ✓ print(N)
}

void Inc( N=2 ){
  * if( N==0) { return; }
  ✓ Inc(N-1)
  ✓ print(N)
}

void Inc( N=1 ){
  * if( N==0) { return; }
  ✓ Inc(N-1)
  ✓ print(N)
}

void Inc( N=0 ){
    if( N==0) { return; }
    Inc(N-1)
    print(N)
}
```

Q8 Given N, print all numbers from N to 1 in dec Order

Dec(5) : 5 4 3 2 1

Dec(5) : print(5)
Dec(4)

Dec(N) : N N-1 N-2 .. 1

Dec(N) : print(N)
Dec(N-1)

Ass: Given N print N..1 in desc order & return nothing

```
void Dec(int N){
    if( N==0) {return;}
    print(N)
    Dec(N-1);
}
```

5Q Given N, print all numbers from N N-1 ... 1 1 2 .. N Dec Inc Order

DecInc(5) : 5 4 3 2 1 1 2 3 4 5

DecInc(5) : print(5)

DecInc(4) : 4 3 2 1 1 2 3 4

print(5)

DecInc(N) : N N-1 .. 1 1 .. N-1 N

DecInc(N) : Print(N)

DecInc(N-1)

Print(N)

Ass: Given N : Print N N-1.. 1 1..N-1 N & return nothing

```
void DecInc(int N){
    if(N==0){return;}
    Print(N);
    DecInc(N-1);
    Print(N);
}
```