Todays Content

1. Recursion Intro
2. Steps to Recursive codes

Use of Recursion:

1. Sorting
2. Trees : BT/BST
3. BackTracking
4. Dynamic Programming
5. 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 exist stack
3. Even if function return type is void, we can write return;

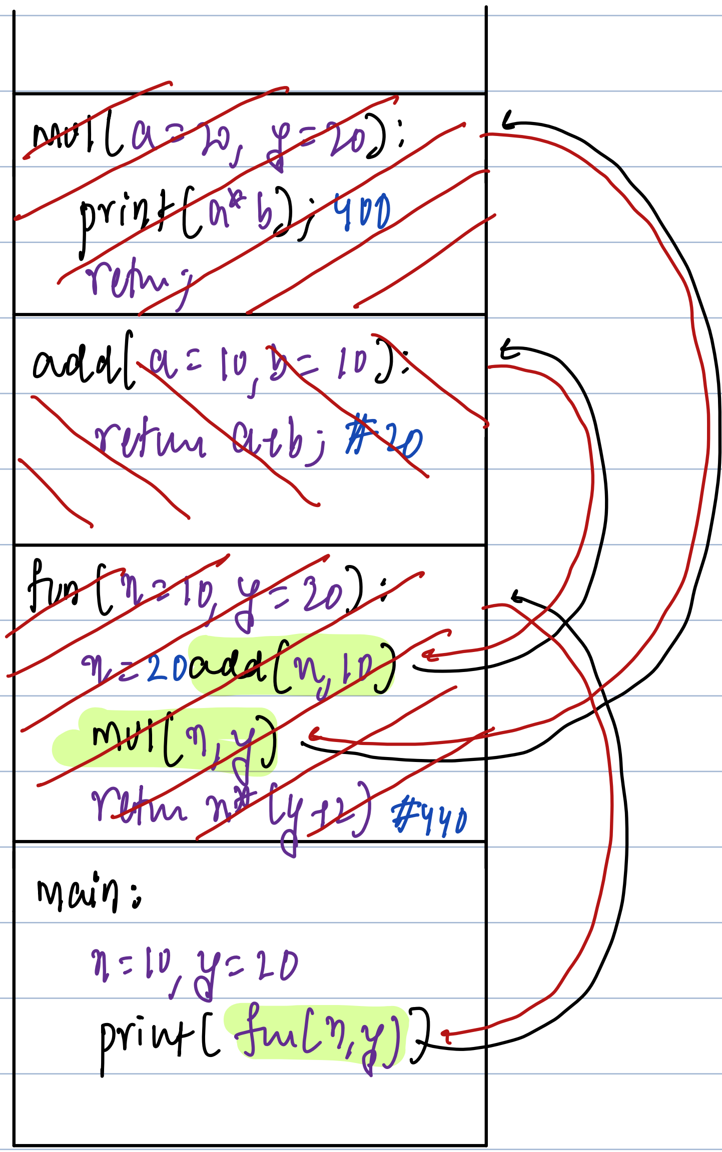
Function Calls

```
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 Problem with Subproblem
smaller Instance of same problem

sum(5) = 1+2+3+4+5
sum(5) = sum(4) + 5 # Sub Problem = sum(4)

sum(N) = 1+2+3+ .. 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 with subproblems. # {Recursive step}
Base Condition: Input for which recursion stops # {Stop}

Ass: Calculate & return sum of 1st N natural numbers

```
int sum(int N){
    if(N==1){ return 1;}
    return sum(N-1) + N
}
```
1+2+3.. 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:

If base condition is not there, code will never stop : TLE Time Limit Exceeded

Before we reach TLE, will reach it's memory limit, because each function call stored in stack & we will get MLE. Stack overflow.

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

1+2+3.. N-1 + N

Inf: On online servers   Ignore it

Iteration = 10^8

Space limit = 1MB = 10^6 Bytes

Int = 4B = 10^5 int[] = $4*10^5$

long = 8B = 10^5 [ ] = $8*10^5$

bool = 1B = 10^6 ≈ $1*10^6$

Array size = [10^5 - 10^6]

How many function calls can stack hold at once?

1. Each function call = approx 10B
2. We can have 10^5 function calls at once in stack

fact(5) = 5*4*3*2*1
fact(5) = 5*fact(4)   # Sub Problem = fact(4)

fact(N) = $\underline{1*2*..N-1}$*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 with subproblems. # {Recursive step}
Note: Subproblems will work according to assumption.
Base Condition: Input for which recursion stops # {stop}

0! = 1.

Ass: Given N, Calculate & return N!

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

```
main(){
    int a = fact(3) : 6
}

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

    int fact(N=2){ : b
        if(N==0){return 1}
        return fact(N-1) * N
    }          1         2

        int fact(N=1){ : c
            if(N==0){return 1}
            return fact(N-1) * N
        }          1         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)
print(5)

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

Inc(N) : Inc(N-1)
print(N)

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

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

48 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 to 1 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)
print(5)

DecInc(N) : N N-1 N-2 .. 1 1 .. N-2 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)
}
```