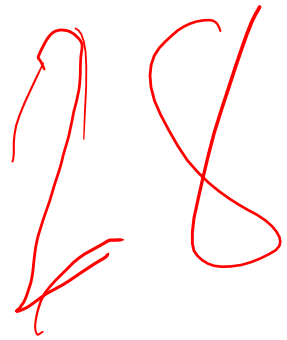
# Let’s solve it



(Developer)

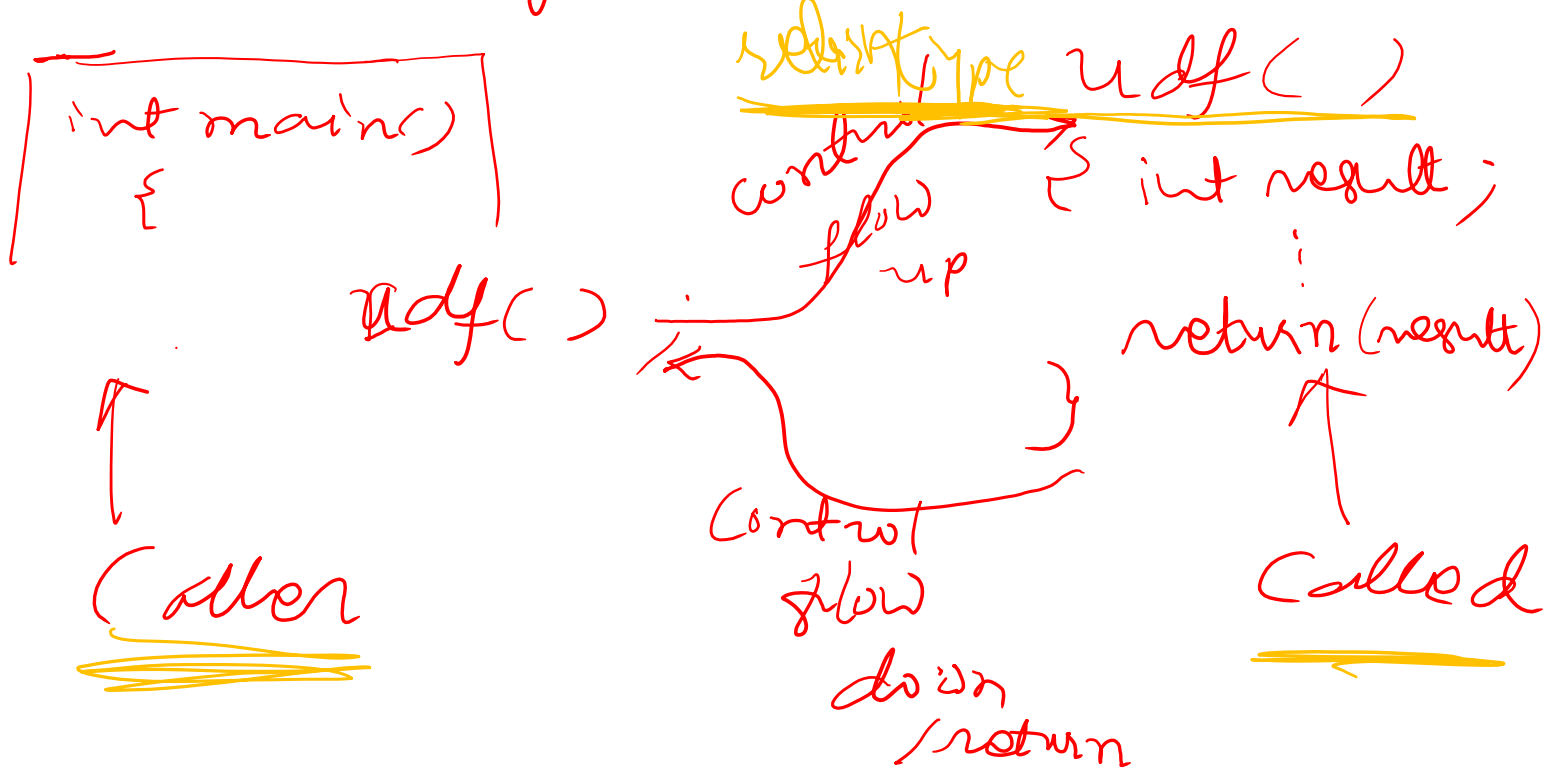
# User Defined Functions

Categories/types of functions based on presence/absence of elements of functions.

return type function_name ( formal arguments )

1. No return type and no formal arguments
2. Return type but no formal arguments
3. No return type but having formal arguments
4. Return type and having formal arguments

What is return type?




This result i.e. int result; is returned from called UDF to caller like main using 'return' keyword.

Two things of importance:

- Execution control flow and
- Data flow — from caller to called
  — from called to caller mostly result

mostly it input to UDF or dummy space location

# Limitation of 'return' keyword in C.

It can theoretically return only single variable (entity).

i.e `return(x);`
or
`return x;`

It can NOT do
`return (x, y, z);`
X

---

Practically we may combine multiple things into one using 'struct' and return that. We will learn later.

Type1 No return type and no arguments

Mostly this type of function doesn't do any data movement either ways.

caller —data→ called

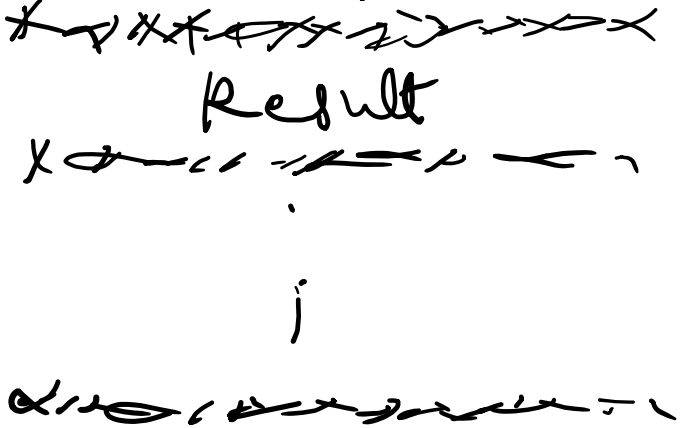
NO

called —data→ caller

i.e.

```
void displaystarline(void)
{
    for(int i=0; i<=80; i++)
        printf("*");
}
```

i.e.
For any marksheet or report



Type 2 Return type but no arguments

Random number generator

From C library

man 3 rand ↵ Linux command

int rand(void); // void indicates 'none'

Also see.

---

Later we will learn concept of local static variable ) Storage class

But imagine a function which

gives you next sequence number

automatically once called first time and onwards as a program level.

```
int getNSequence(void)
{
    static int s=0;
    s+=1;
    return(s);
}
```

```
int main()
{
    printf("%d\n",
        getsNSeq());
    :
    :
    printf("%d\n",
        getsNSeq());
}
```

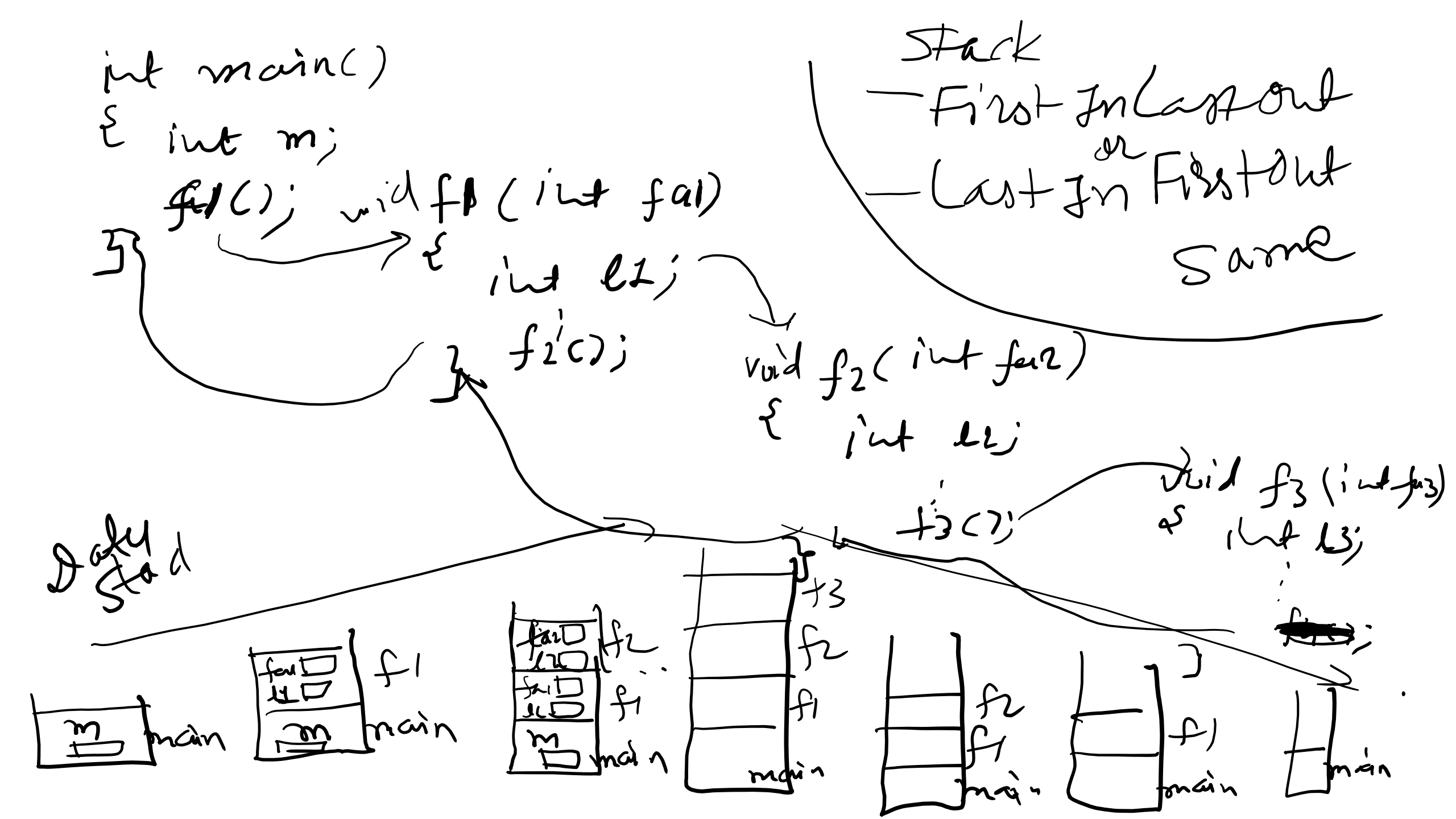
output

1
2
3

Type #3 & Type #4

Where formal arguments are involved,
We need to learn and understand
data stack grow and shrink with
activation frame records upon function call (grow)
and return (shrink).

Note that function formal arguments and
any variables declared within function
(also called local or automatic variables)
gets memory fresh every time when the
function is called on top of stack and
destroys when finishes.


int main()
{ int m;
f1(); void f1(int fa1)
{ int l1;
f2();
}
void f2(int fa2)
{ int l2;
f3();
void f3(int fa3)
{ int l3;
Stack
— First In Last Out
or
— Last In First Out
Same
Daily Stand
m
main
fa1
l1
f1
fa2
l2
f2
f3
f2
f1
main

Please, note that function call and return flow of execution also internally manages location to jump and location to resume upon return with property of stack only.

Type 3 No return type but having formal arguments

i.e. display factorial of a given number by caller.

```
void displayFactorial(int);
int main()
{
    int n=5;
    displayFactorial(n);
};
void displayFactorial(int a)
{
    int result=1, i;
    for(i=1; i<=a; i++)
    {
        result = result * i;
    }
    printf("%d", result);
    return(result);
}
```

Type #4

Return type and having formal arguments too

```
int findFactorial(int n);

int main()
{
    int n=5;
    int result;
    result=findFactorial(n);
    printf("%d\n",result);
}
int findFactorial(int a)
{
    int result=1;
    for(i=1;i<=a;i++)
    {
    result=result*i;
    }
    return(result);
}
```