

Functions

If we want to do some specific **task** again and again, we write it in a function, and then we use that function for the better programming approach.

```
returntype function_name(parameters) {  
    code
```

```
}
```

Function involves

- ② → Function Definition
- ① → Function Declaration (done before the Function Definition)
- ③ → Function Call.

Parameters (type1, type2, type3)

- Function saves us from the redundancy
- Improves the readability.

In Function call, we pass the arguments which was previously parameters in the function declaration.

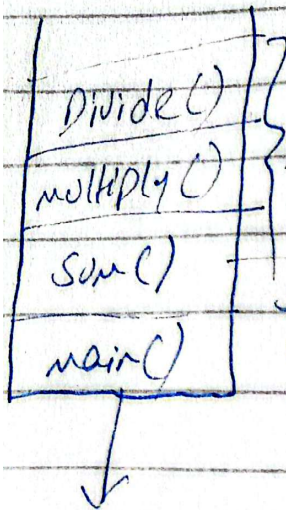
Example)

Factorial of Numbers.

Function memory

- ① Stack memory (static)
- ② Heap (Dynamic)

Functions are in the stack memory.



All will vanish after completing their functionalities.

→ when the return statement comes the control will go to the main() again and

sum() will be removed from the stack

when all the executions are completed in the

main() or return 0

then the main() will

be removed from the stack.

```
int sum(int a, int b)
```

```
sum = a + b;
```

```
return sum;
```

```
int main()
```

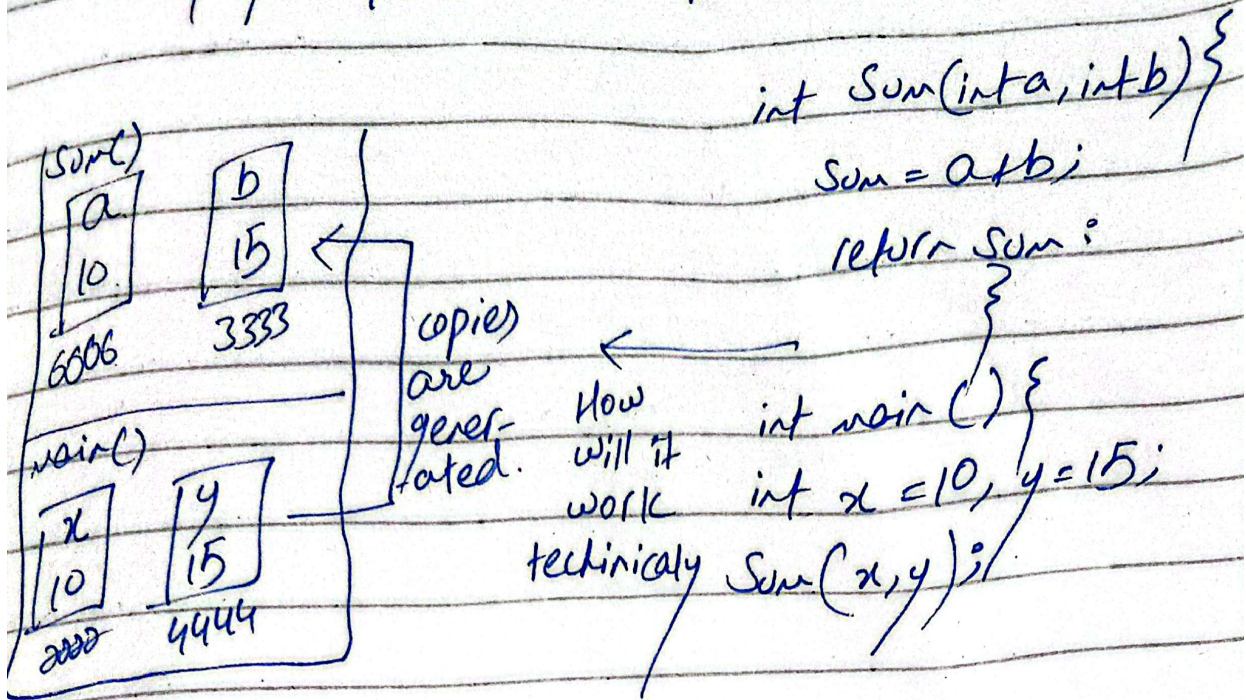
```
sum(14, 15);
```

break point of the function.

→ No statements will be executed after the return statement.

Pass By Value

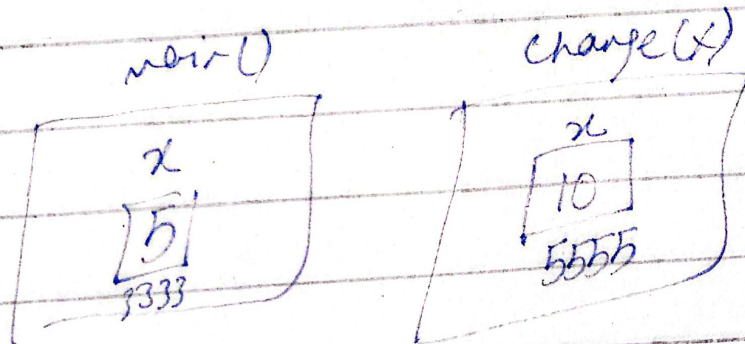
Copy of arguments are passed to function



copies are generated.

although if we keep the `main()` values as `a` and `b` as well it will not affect each other because the memory addresses are different. names are just apparent.

For example



Sum of Digits of Number?

while (num > 0) {

lastdigit = num % 10

num = num / 10

digitSum += lastdigit

}

Binomial coefficient

$${}^nC_r = \frac{n!}{r!(n-r)!}$$

$${}^8C_2 = \frac{8!}{2!(8-2)!} = 28$$

- ① → Prime Numbers
- ② → Prime Numbers series
- ③ → Fibonacci series