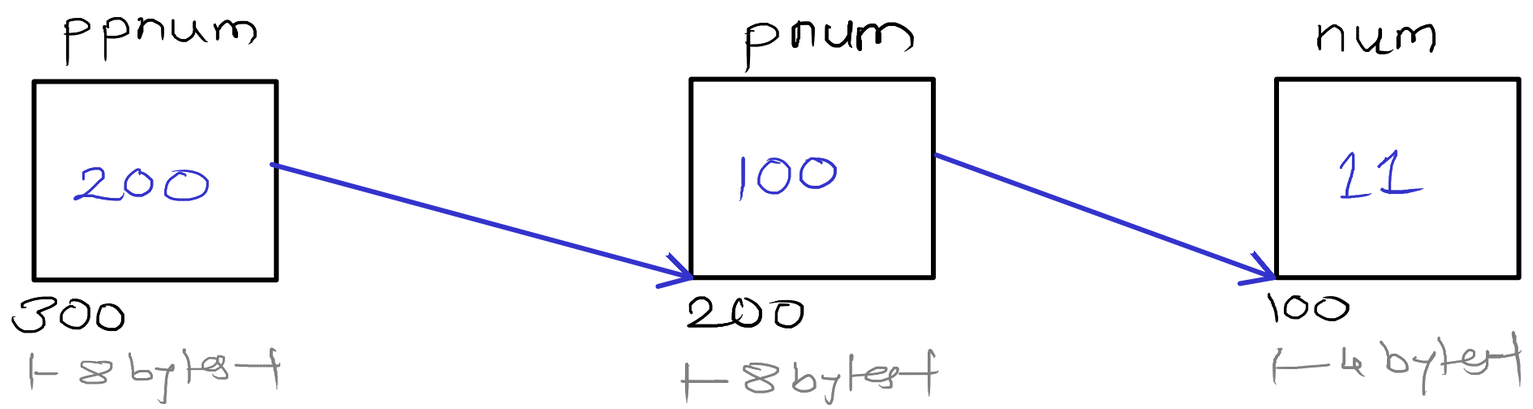
Pointer to Pointer

int **ppnum = &pnum; **int *pnum = #** **int num = 11;**



Pointer to integer pointer **Pointer to integer** **integer variable**

num → 11

ppnum = 200

*ppnum = *200

= 100

**ppnum = **200

= *100

= 11

pnum = 100

*pnum = *100

= 11

SF(pnum) = sizeof(int)

SF(ppnum) = sizeof(int*)

Pointer Arithmetic

- to perform arithmetic, we need two operands (op1 and op2)
- there are two ways to perform arithmetic

1. op1 - pointer
   op2 - integer value
   - only + and - is allowed
   - pointer is incremented or decremented by its scale factor
   - ptr + n = ptr + n * SF(ptr)
   - ptr - n = ptr - n * SF(ptr)

2. op1 - pointer
   op2 - pointer
   - only - is allowed
   - result of subtraction is divided by scale factor of op1
   - op1 - op2 = (op1 - op2) / SF(op1)

- *, / or % is not allowed (no meaningful result will generated)

Recursion

- calling function within same function
- recursion is used when
    - we can define process/formula in terms of itself
    - we should have some terminating condition

eg - Factorial -> 5! = 5 * 4!
= 5 * 4 * 3!
= 5 * 4 * 3 * 2!
= 5 * 4 * 3 * 2 * 1!
= 5 * 4 * 3 * 2 * 1 (terminating condition - 1! = 1)

```
int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```

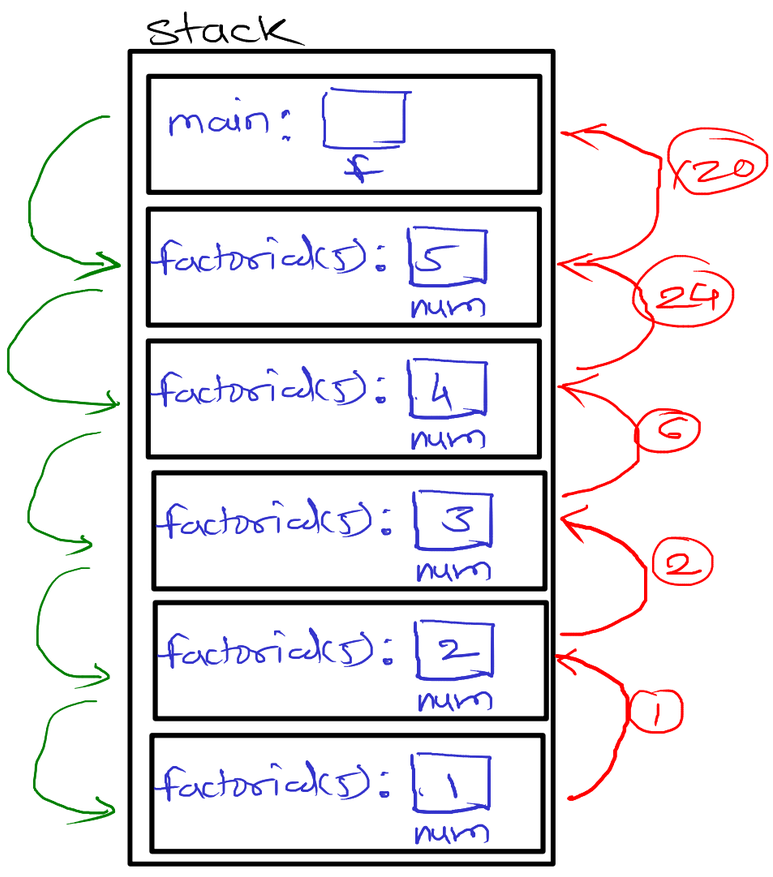
```
int main(void)
{
        int f = factorial(5);
        return 0;
}

int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```

```
int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```

```
int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```

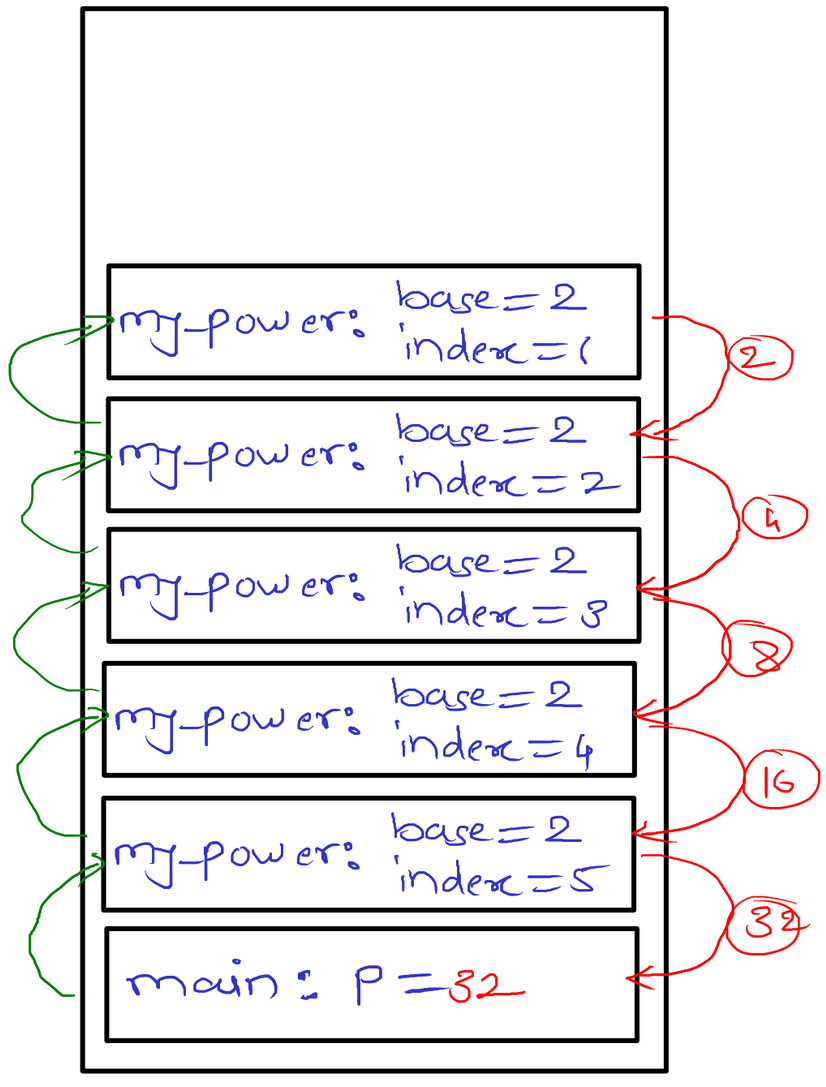
```
int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```

```
int factorial(int num)
{
        if(num == 1)
                return 1;
        return num * factorial(num - 1);
}
```



2^3 = 2 * 2^2
= 2 * 2 * 2^1
= 2 * 2 * 2 (terminating condition = ?^1 = ?)

```
int main(void)
{
        int p = my_power(2, 5);
        return 0;
}

int my_power(int base, int index)
{
        if(index == 1)
                return base;
        return base * my_power(base, index - 1);
}
```



num = 1234

rem = num % 10 --> 4
num /= 10 --> 123

rem = num % 10 --> 3
num /= 10 --> 12

rem = num % 10 --> 2
num /= 10 --> 1

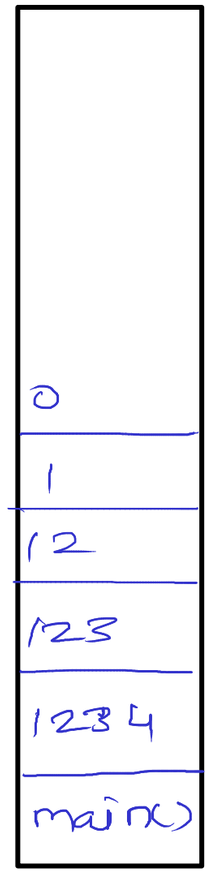
rem = num % 10 --> 1
num /= 10 --> 0

Terminating condition -> num == 0

Tail recursion

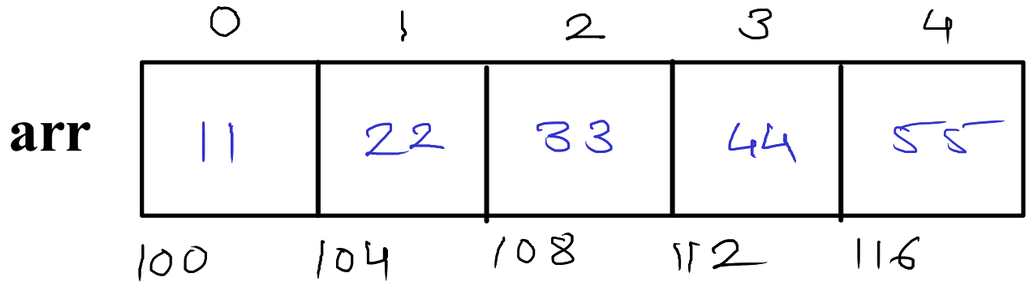
```
void reverse_digits(int num)
{
        if(num == 0)
                return;
        int rem = num % 10;
        printf("%d", rem);
        reverse_digits(num / 10);
}

void forward_digits(int num)
{
        if(num == 0)
                return;
        int rem = num % 10;
        forward_digits(num / 10);
        printf("%d", rem);
}
```

Non-tail recursion



Array

int arr[5] = { 11, 22, 33, 44, 55 };



arr[0] = 11

sizeof(arr[0]) = 4

sizeof(arr) = 20

↑

sum of sizes of all elements

↑

Total size = n * sizeof(arr[0])

Total size = 5 * 4 = 20