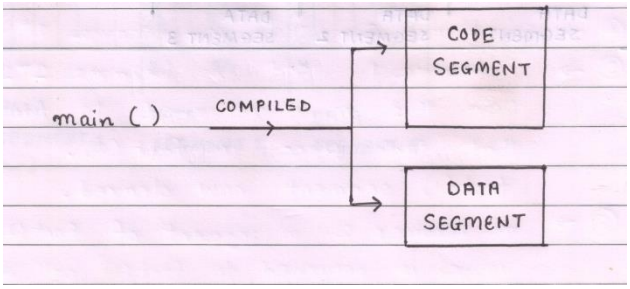
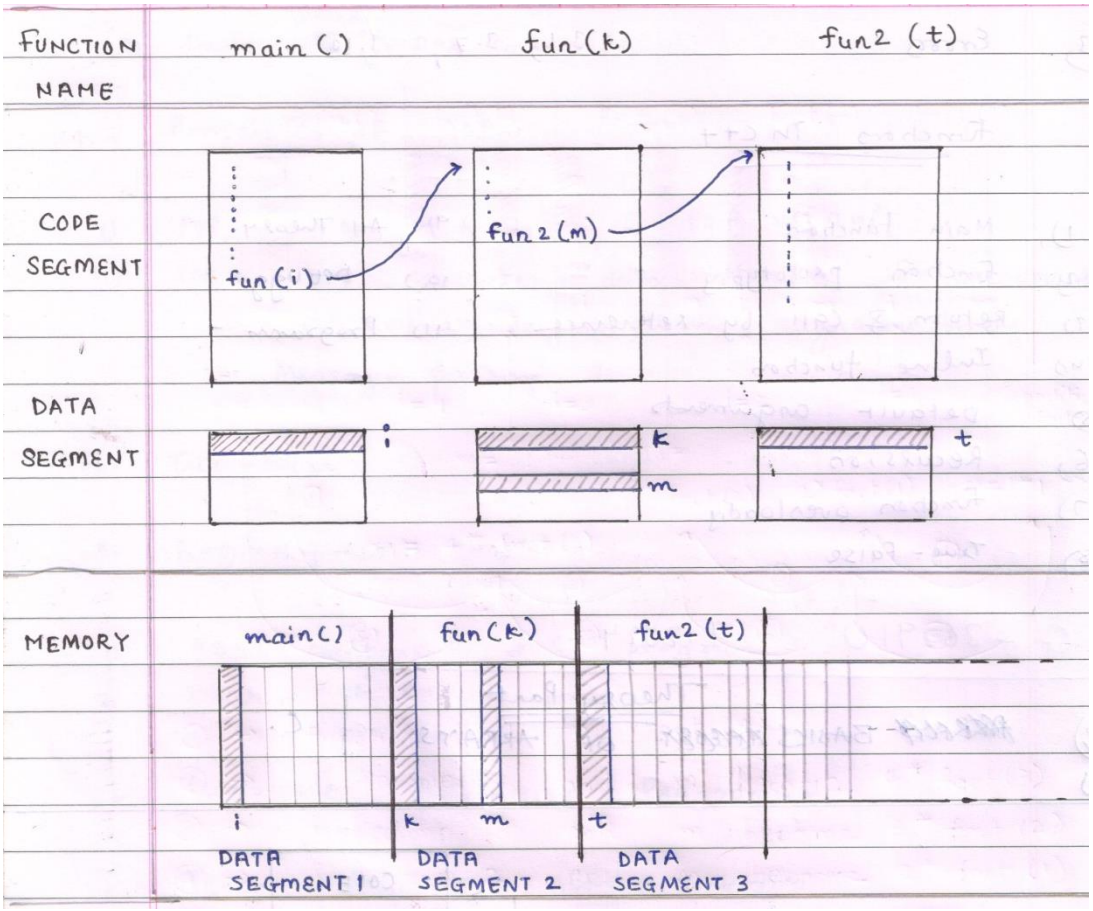
When the function, say main () is compiled, it is compiled in two parts:

a) Code segment
b) Data segment



Consider the following diagram,

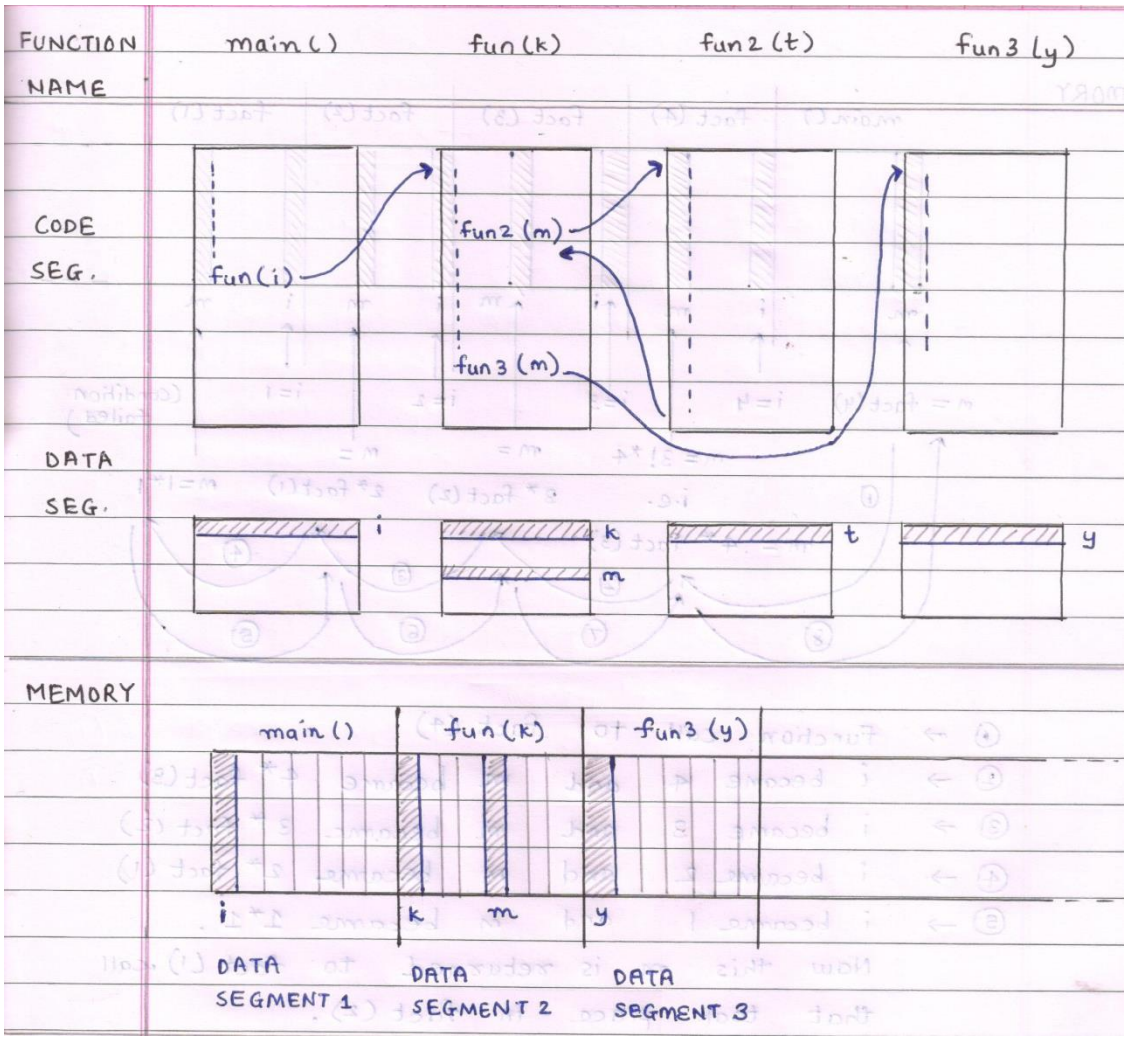


Initially the function main () starts its execution. This creates a code segment and a data segment of main. In memory, the stack of data items declared in main is stored in 'Data Segment 1' say.

At a certain point, fun (i) in main is called. The value of actual parameter 'i' is copied to formal parameter 'k'. For fun () also, its own code segment and data segment is created. In memory, the stack of data items declared in fun is stored in 'Data Segment 2'.

Inside fun (), there is a call to another function fun2 (). The value of 'm' is copied to 't'. Again code segment and data segment of fun2 () is created. In memory, the stack of data items declared in fun2 is stored in 'Data Segment 3'.

Once the fun2 is executed completely, the fun is executed again from the next step where fun2 was called. Once the control is back to fun, 'Data Segment 3' is completely removed from memory. While executing the rest of fun, if there is any further call to other function, say fun3, the data stack of fun3 is created at same place where 'Data Segment 3' was formed earlier.



Thus the data stack memory allocation is dynamic. This allows the reuse of same memory increasing the memory efficiency of the program.

The above theory was necessary to understand, *how does recursive functions execute?*

Consider the following program of finding factorials:

```
#include <stdio.h>
int main()
{       int m;
        m = fact (4);
        printf("Fact(4) = %d\n", m);
        return 0;
}
int fact (int i)
{       int m = 1;
        if ( i > 1)
                m = i*fact (i – 1);
        return m;
}
```

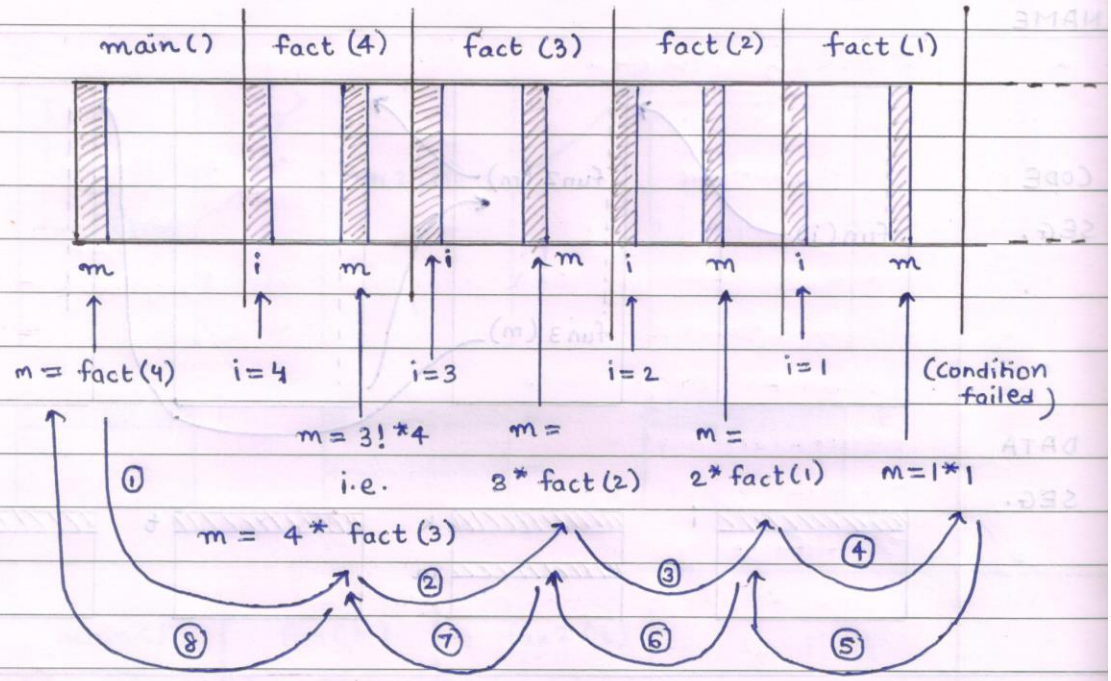
Output: Fact(4) = 24

The explanation of how the function is executed recursively is given on the next page.

Note that the variables 'i' and 'm' are different of different data segments. This means 'm' variable of data segment fact(3) and that of fact(2) have different memory locations.

The proof that the execution takes place in this order is given with the help of the program named, *Understanding factorial steps.*

MEMORY



① → Function call to fact (4)

② → i became 4 and m became 4* fact(3)

③ → i became 3 and m became 3 * fact (2)

④ → i became 2 and m became 2* fact (1)

⑤ → i became 1 and m became 1*1.
Now this m is returned to fact (1) call that took place in fact (2).
fact(1) segment now cleared.

⑥ → m becomes 2 in segment of fact (2). This value is returned to fact(2) call in fact(3).

⑦ → m becomes 6 in segment of fact (3). This value is returned to fact (3) call in fact (4).

⑧ → m becomes 24 in segment of fact (4). This m is returned to main(). m in main becomes 24 which is answer.