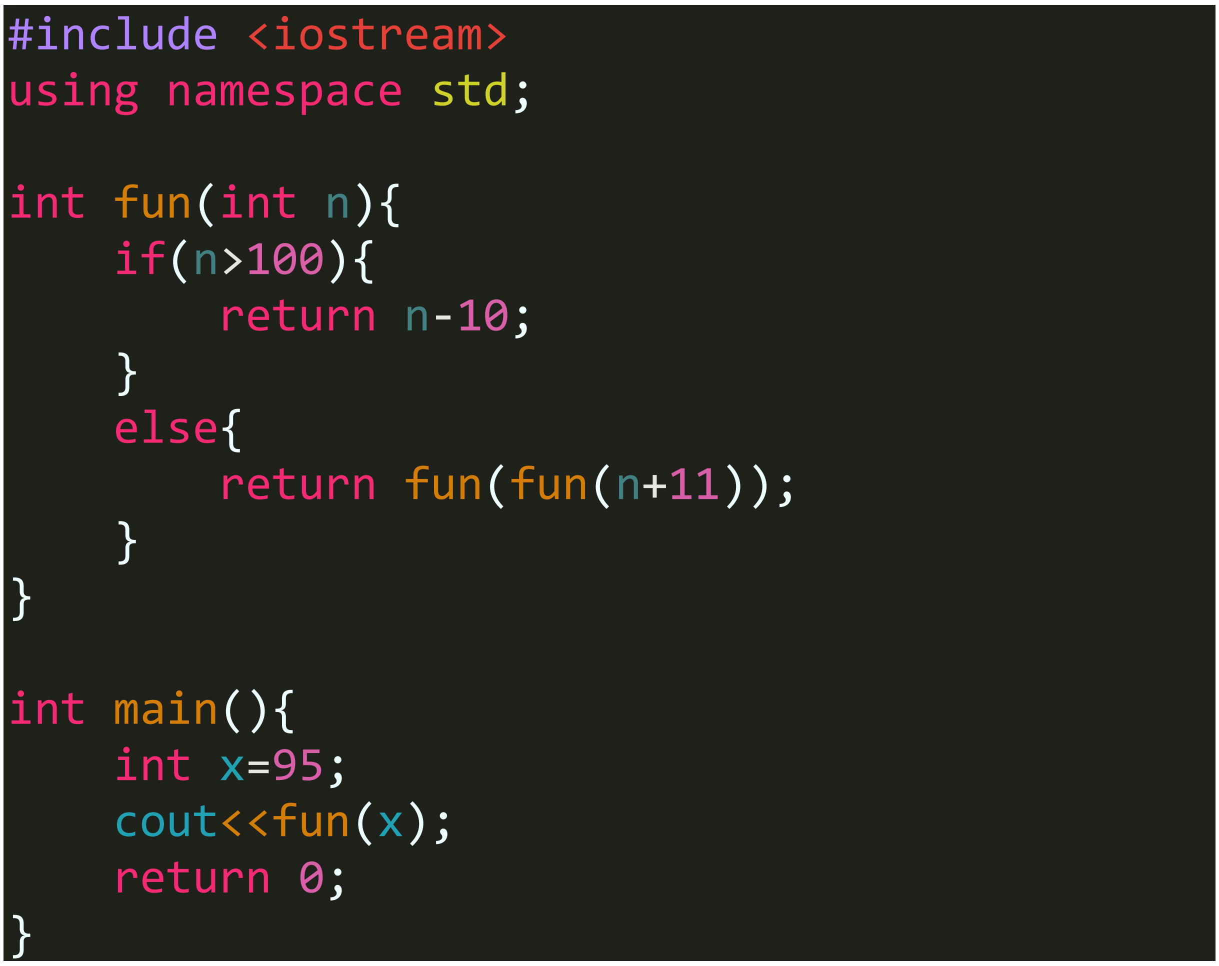
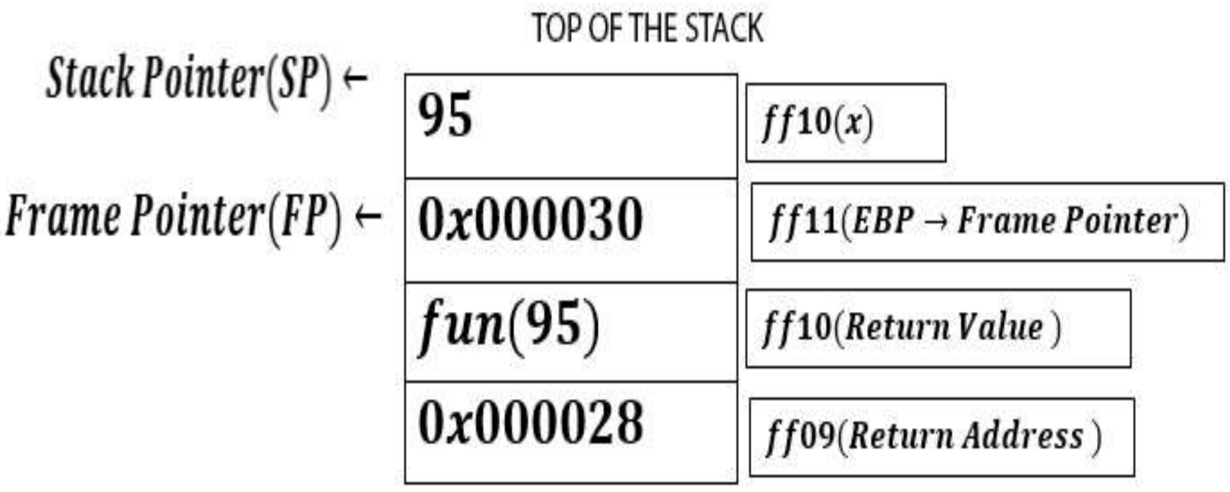
Nested Recursion



```cpp
#include <iostream>
using namespace std;

int fun(int n){
    if(n>100){
        return n-10;
    }
    else{
        return fun(fun(n+11));
    }
}

int main(){
    int x=95;
    cout<<fun(x);
    return 0;
}
```

$x = 95$, *Hence main function' stack frame 's function call is* : $fun(95)$.

Main Function's Stack Frame



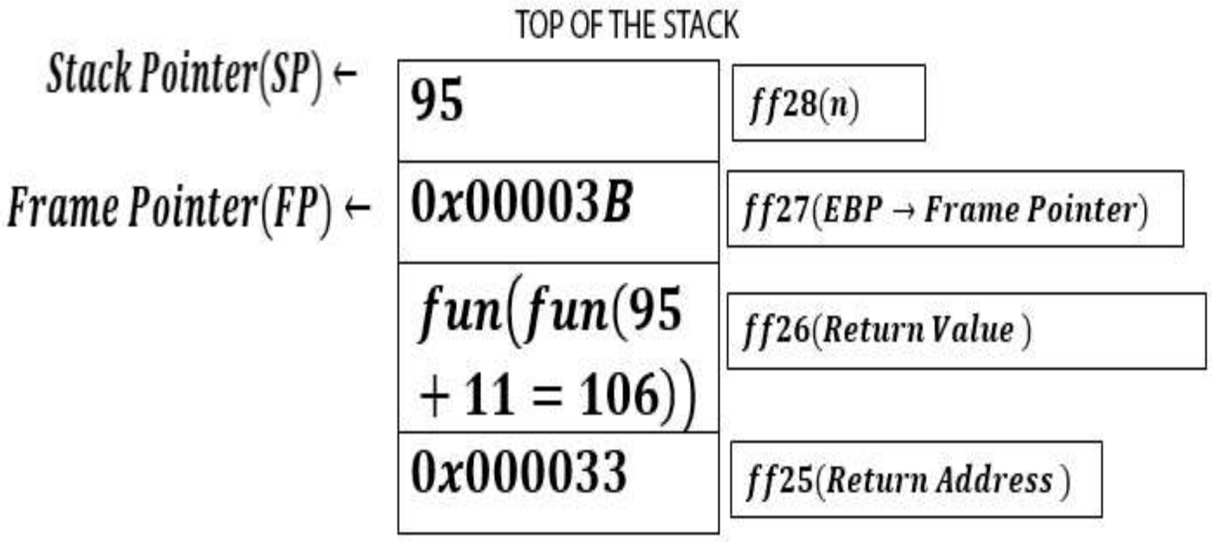
Now fun(95) will create a Stack Frame.

fun(95)'s Stack Frame

First it enters to if statement
and checks $\rightarrow$ $\mathbf{95 > 100}$.

$\mathbf{95 > 100}$ ***is false hence now it calls :***

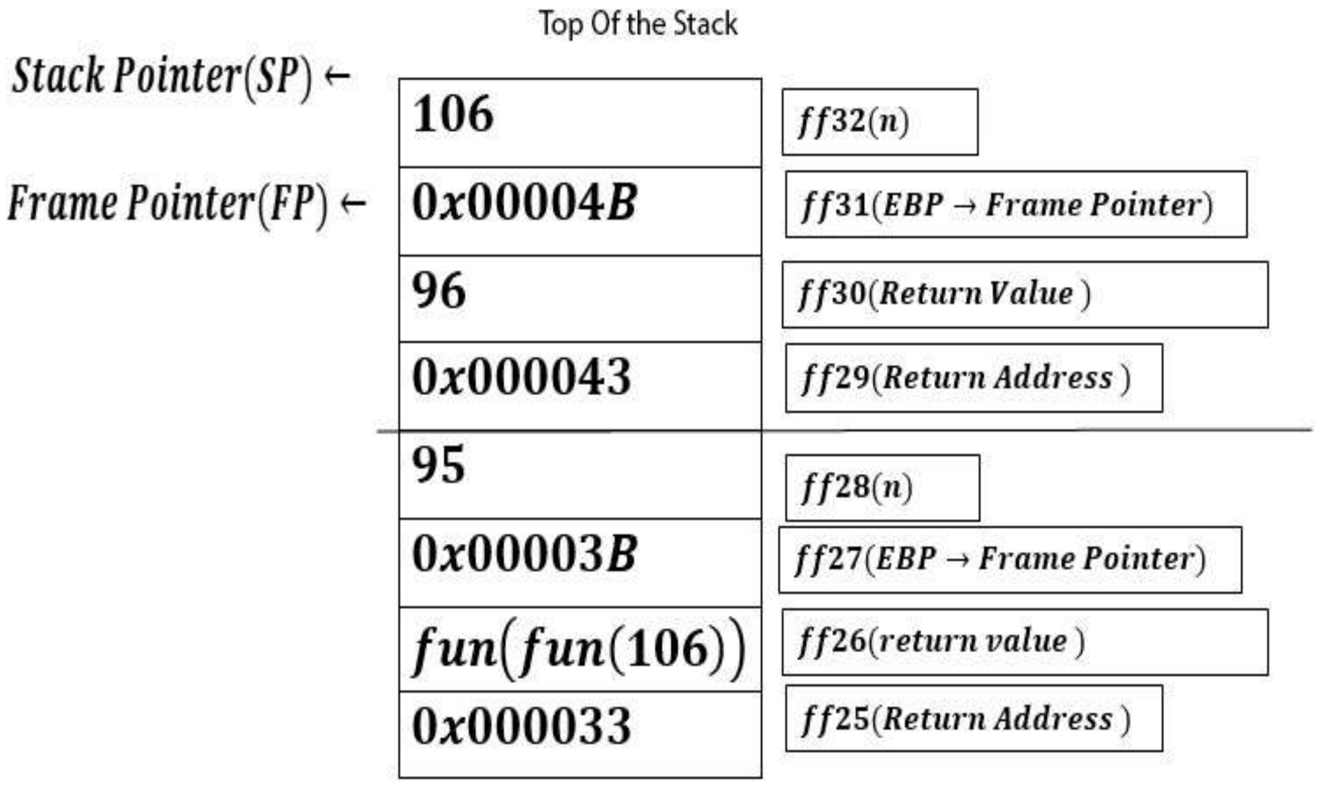
$fun(fun(95 + 11))$



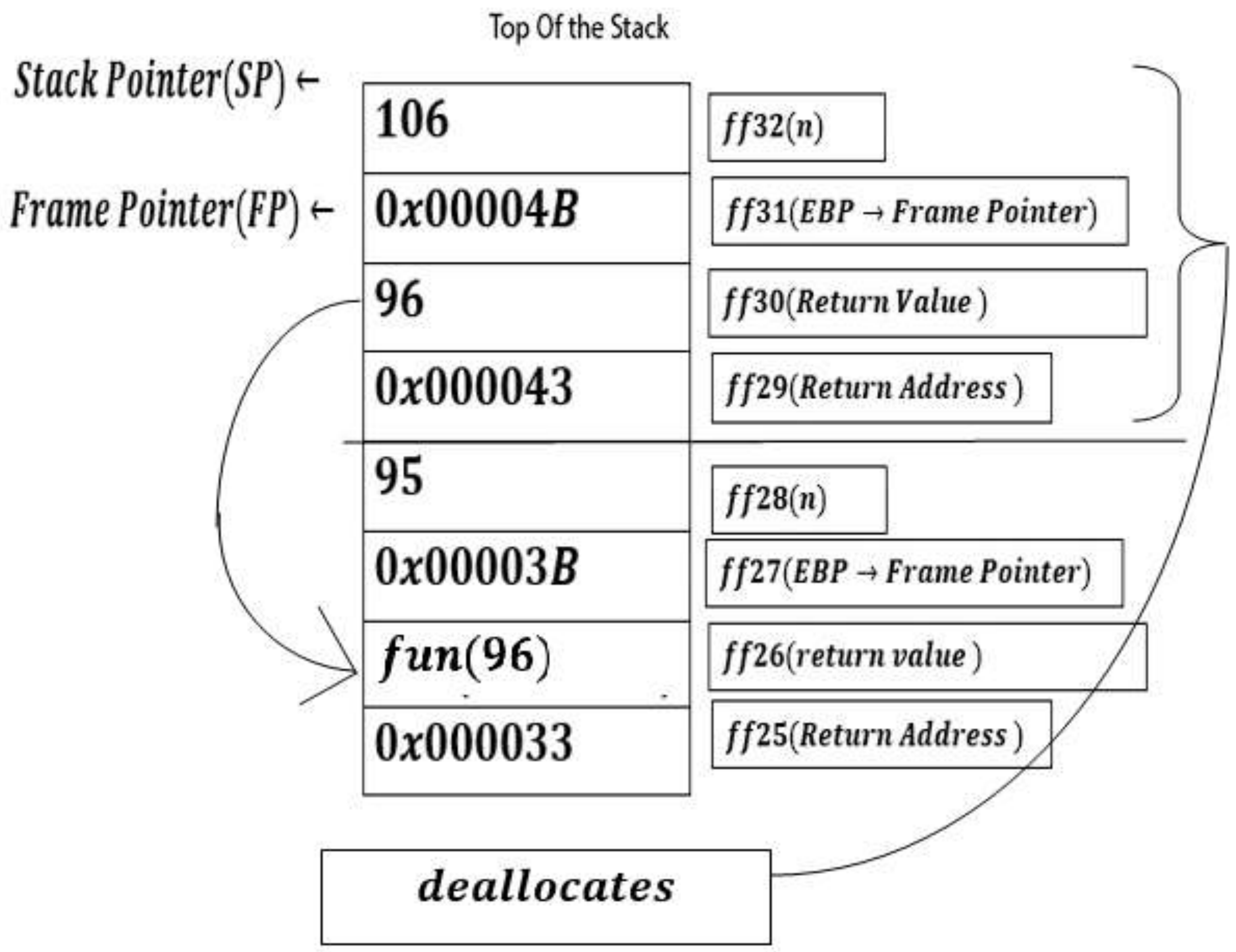
$\boldsymbol{There\ will\ be\ a\ temporary\ stack\ frame\ created, fun(106).}$

$\boldsymbol{if(106 > 100) is\ true\ , hence\ it\ will\ return\ 106 - 10 = 96.}$

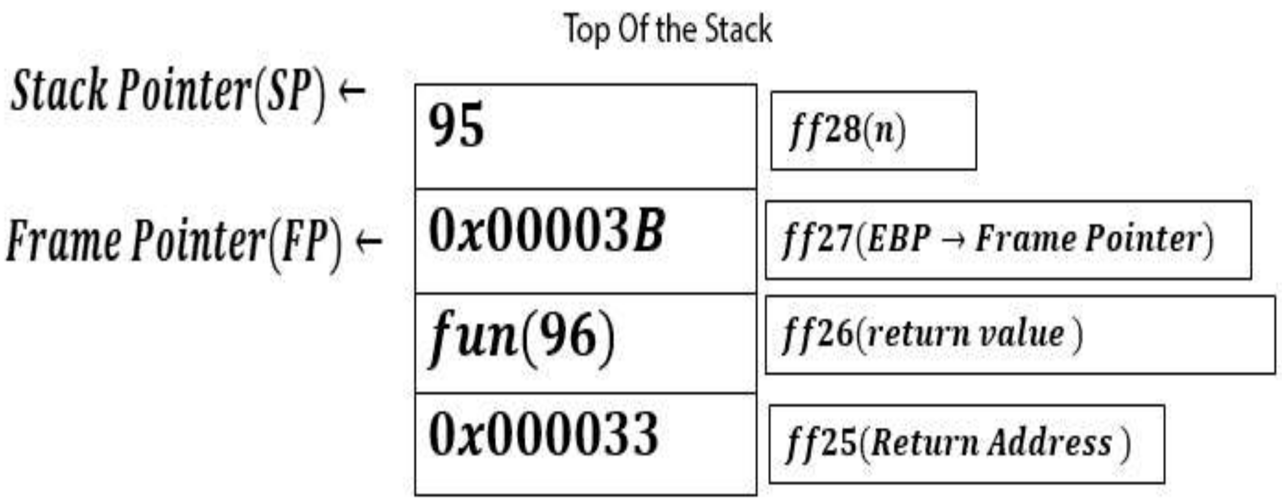
$\underline{fun(106)'s\ Stack\ Frame}$



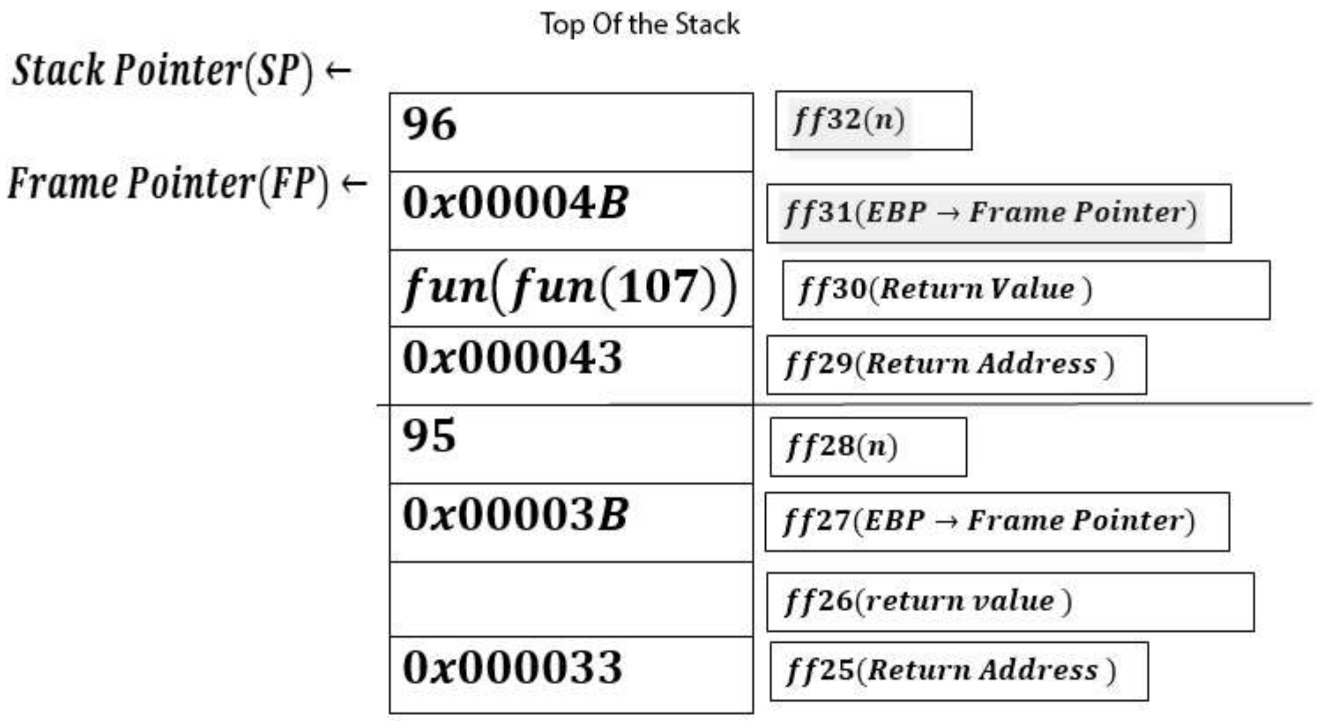
$Now\ PC(Program\ Counter) will\ recieve\ the$
$address:\ 0x000037, the\ stack\ frame$
$of\ fun(106)\ will\ get\ popped\ out\ and$
$return\ value\ 96\ will\ be\ pushed\ to$
$fun(95)'s\ stack\ frame.$


Top Of the Stack
Stack Pointer(SP) ←
Frame Pointer(FP) ←
106
0x00004B
96
0x000043
95
0x00003B
fun(96)
0x000033
ff32(n)
ff31(EBP → Frame Pointer)
ff30(Return Value)
ff29(Return Address)
ff28(n)
ff27(EBP → Frame Pointer)
ff26(return value)
ff25(Return Address)
deallocates

That is:

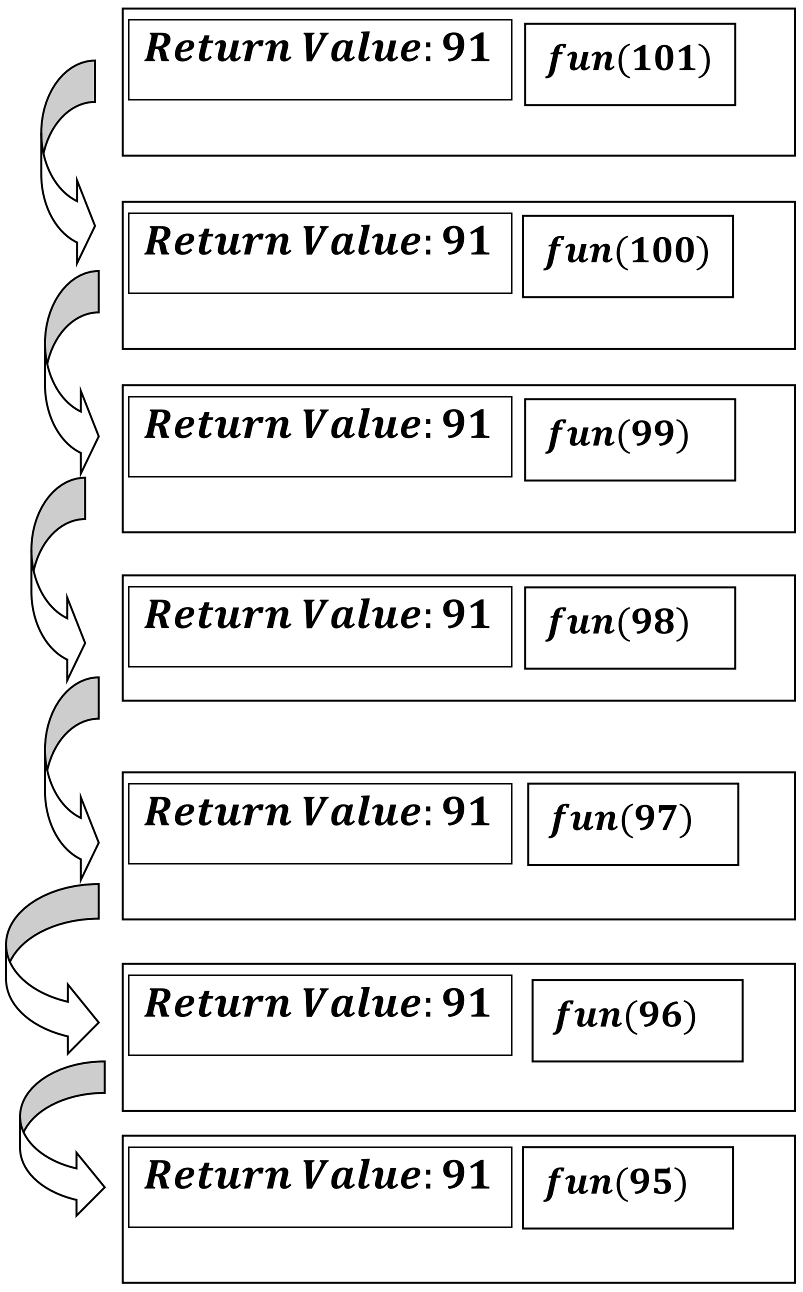


$fun(96)'s\ Stack\ Frame$



Hence it goes like: –

$$\begin{tikzcd}
\boxed{fun(95)} \arrow[d] \\
\boxed{fun(fun(95+11)) \rightarrow fun(95+11=106)\, returns\ 96} \arrow[d] \\
\boxed{fun(96)} \arrow[d] \\
\boxed{fun(fun(96+11)) \rightarrow fun(96+11=107)\ returns\ 97} \arrow[d] \\
\boxed{fun(97)} \arrow[d] \\
\boxed{fun(fun(97+11)) \rightarrow fun(97+11=108)\ returns\ 98} \arrow[d] \\
\boxed{fun(98)} \arrow[d] \\
\boxed{fun(fun(98+11)) \rightarrow fun(98+11=109)\ returns\ 99} \arrow[d] \\
\boxed{fun(99)} \arrow[d] \\
\boxed{fun(fun(99+11)) \rightarrow fun(99+11=110)\ returns\ 100} \arrow[d] \\
\boxed{fun(100)} \arrow[d] \\
\boxed{fun(fun(100+11)) \rightarrow fun(100+11=111)\ returns\ 101} \arrow[d] \\
\boxed{fun(101) \rightarrow returns\ 91}
\end{tikzcd}$$



Pop Operation

Hence all these functions stack frames will get deallocated and return value 91 will get pushed each time in stack frame till it reach fun(95) and then fun(95)also gets popped out (deallocated)and we get the return value 91.

Implicit Recursion

A specific sort of recursion called implicit recursion occurs when a function calls itself without making an explicit recursive call.

This can occur when a function calls another function, which then calls the original code once again and starts a recursive execution of the original function.

***As example below**:*

<u>Product of all elements in a vector using recursion</u>

```cpp
#include <iostream>
#include <vector>

using namespace std;

int calculateProduct(vector<int> &numbers)
{
    if (numbers.empty())
    {
        return 1;
    }
    else
    {

        int firstElement = numbers[0];
        vector<int> remainingNumbers(numbers.begin() +
1, numbers.end());
        int productOfRemaining =
calculateProduct(remainingNumbers);

        return firstElement * productOfRemaining;
    }
}

int main()
{
    vector<int> numbers = {1, 2, 3, 4, 5};

    int product = calculateProduct(numbers);
     cout << "The product of all elements is: " <<
product << endl;

    return 0;
}
```

i.e. at first it takes out the first digit of the vector out ,which we observe ,

$$\boldsymbol{int\ firstelement = numbers[0].}$$

Next we create a vector of remaining numbers where, it will contain from the second element till last element.

$\boldsymbol{vector < int > remainingNumbers(numbers.begin() + 1, numbers.end());}$

Hence as it will call recursively, remaining numbers will have:

***Remaining Numbers*: 2, 3, 4, 5**

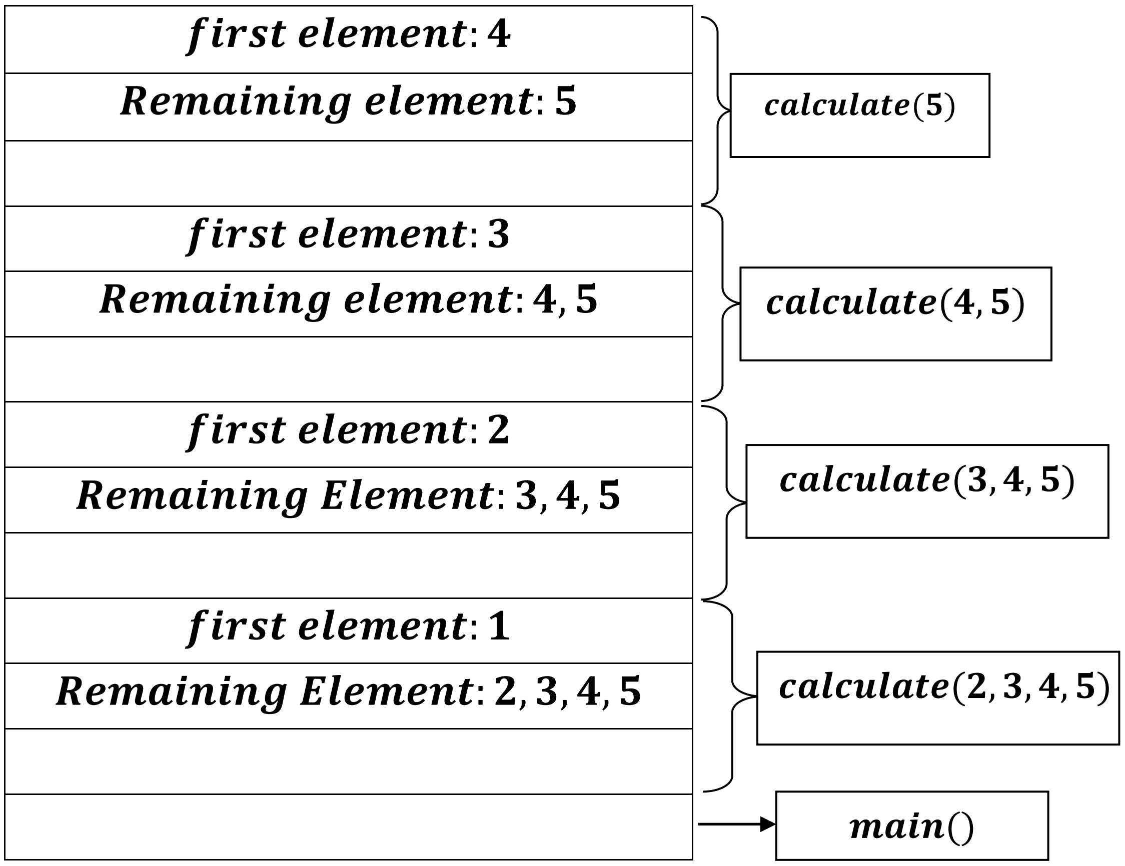
***Remaining Numbers*: 3, 4, 5**

***Remaining Numbers*: 4, 5**

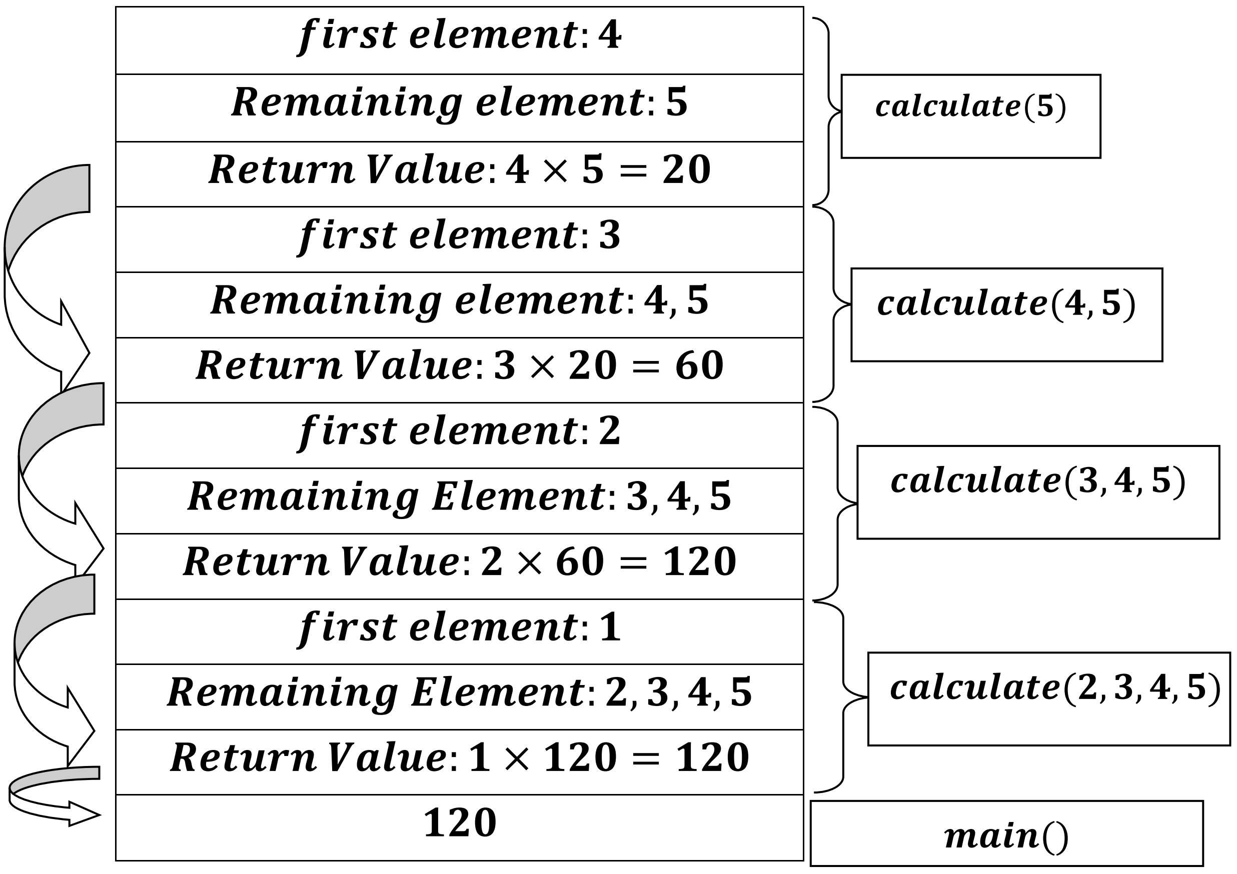
***Remaining Numbers*: 5**

Lets make the stack simple and view whats actually is happenning?

Push Operation



<u>Pop Operation</u>



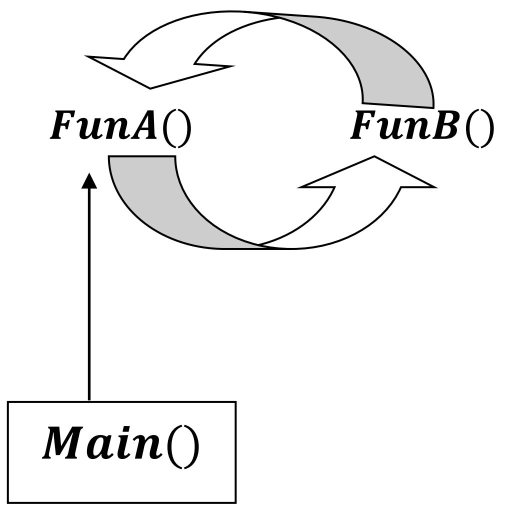
***Note**: In the above example remainingNumbers is called earlier then original recursive function i.e. calculateProduct(remainingNumbers) is called. Hence it is called implicit function. One of the implicit function type is <u>Indirect function.</u>*

Indirect Recursion

```cpp
#include<iostream>
using namespace std;

//Function Declaration
void funB(int ) ;
void funA(int );

 void funA(int n)
{
    if(n>0)
    {
        cout<<n<<" ";
        funB(n-1);
    }
}
void funB(int n)
{
    if(n>1)
    {
        cout<<n<<" ";
        funA(n/2);
    }
}

int main()
{
    funA(20);
    return 0;
}
```

Indirect recursion refers to a situation where a group of functions call each other in a circular manner, creating a cycle of function call.

In the above FunA() calls FunB() then FunB() calls FunA() i.e., FunA() calls FunB() then original FunA() function is called.



That is two function is mutually dependent also known as <u>mutual recursion.</u>

****