ASSIGNMENT 2

- ACTIVATION RECORDS

CASE – 1) RECURSIVE FUNCTIONS

Let’s take for instance a function that calculates the factorial of a number.

CODE :

```
def fact(n):
    if(n > 0) :
        return (n * fact (n -1))
    else
        return 1
fact(3)
```

The procedure will be as follows:

fact(3)

calls returns

fact(2)

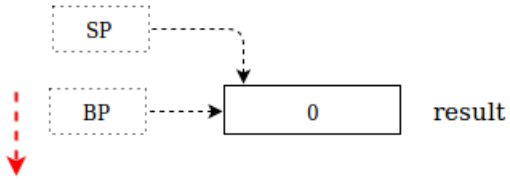
calls returns

fact(1)

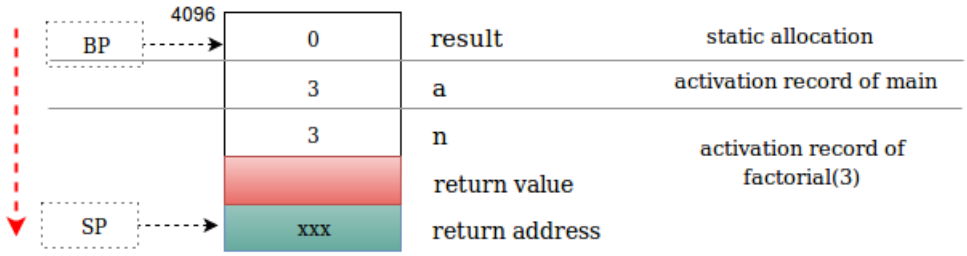
calls returns

fact(0)

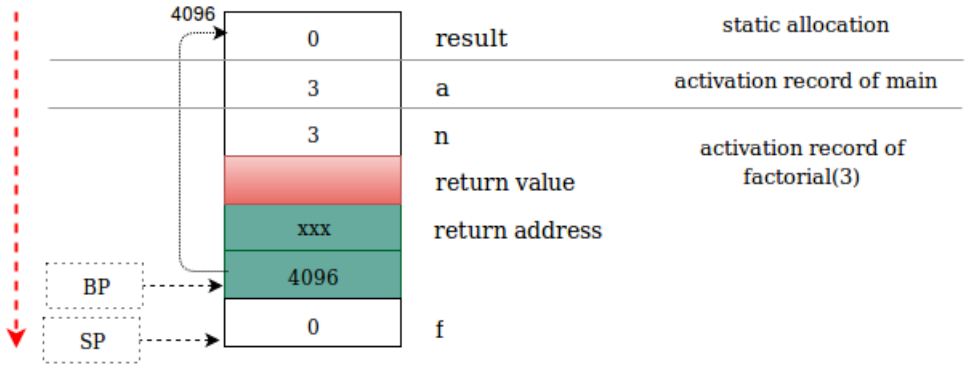
1. The global variables are allocated statically in the initial portion of the stack. Assume that stack begins at 4096



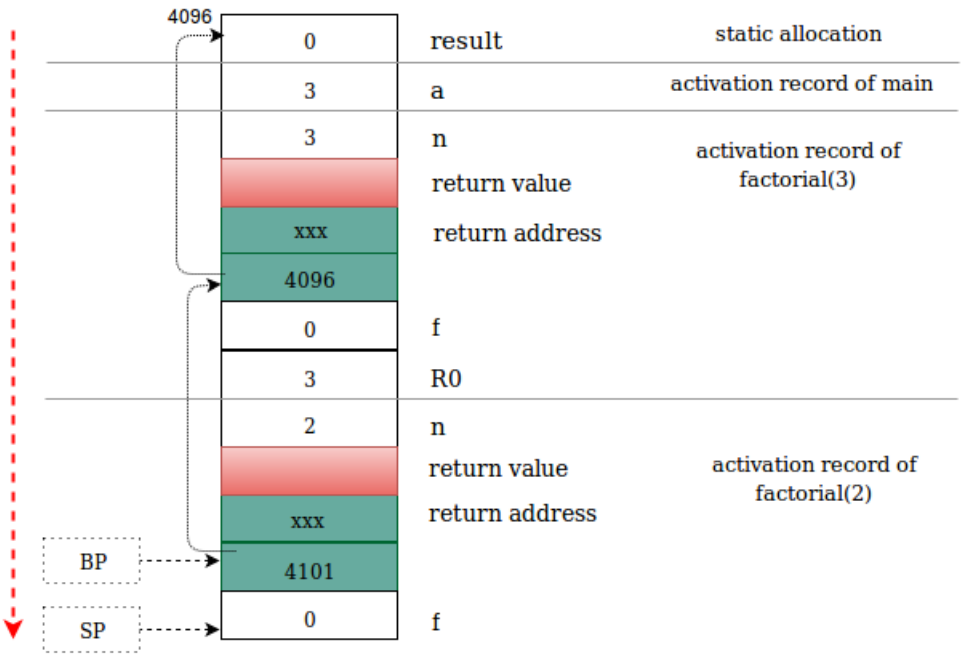
2. Assuming that user inputs 3 resulting in a=3, the main functions sets up stack locations for its local variables and calls the function factorial(3) after setting up a part of the callee's activation record.



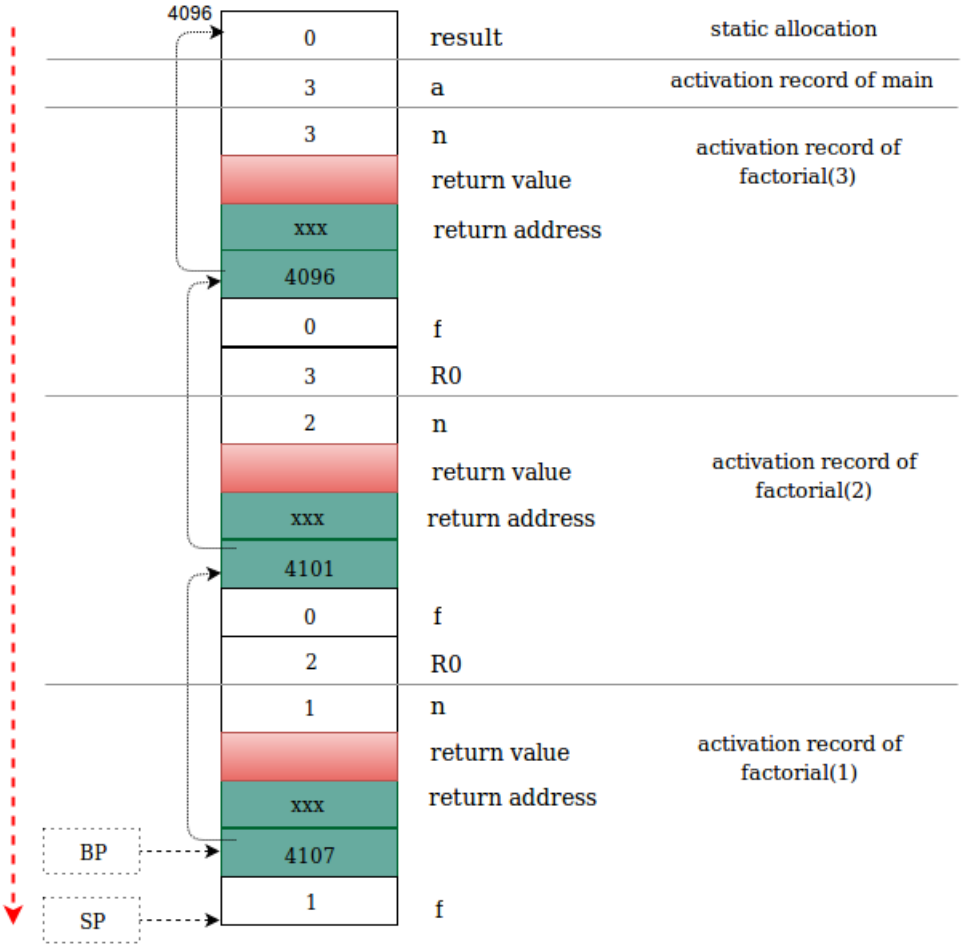
3. Factorial(3) saves the old Base pointer and sets up locations for its local variables.



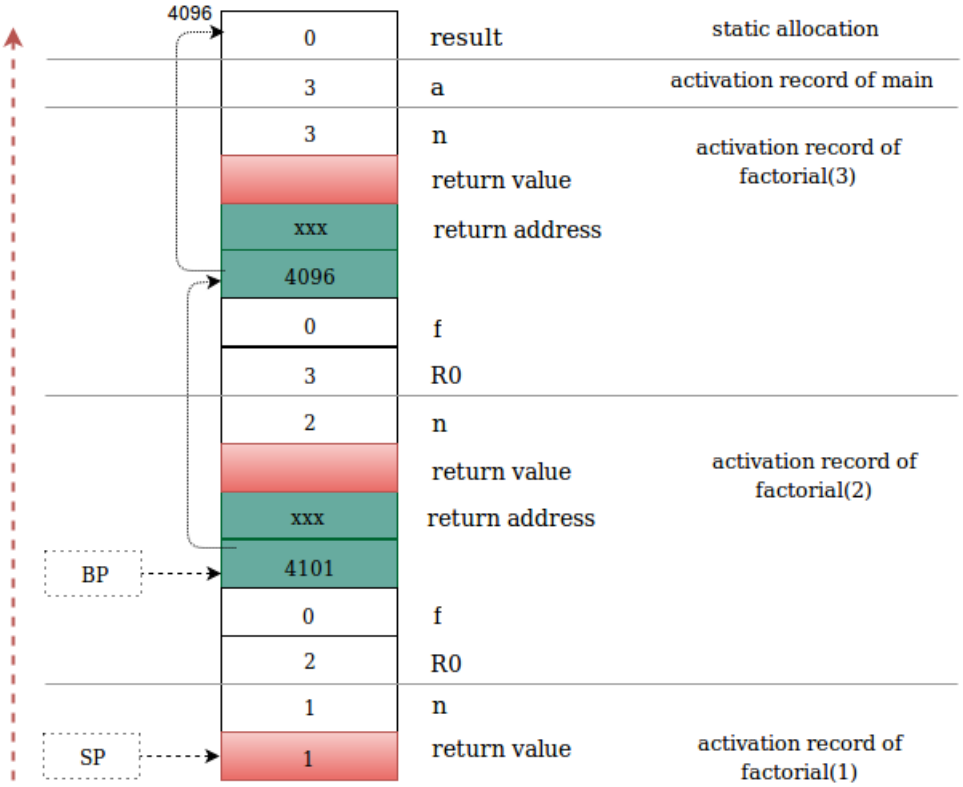
4. Factorial(3) calls factorial(2) and the activation record of factorial(2) is setup similar to the above steps. The register R0 is assumed to be used for temporary storage of the value of n in the expression *n * factorial(n-1)* i.e, 3.



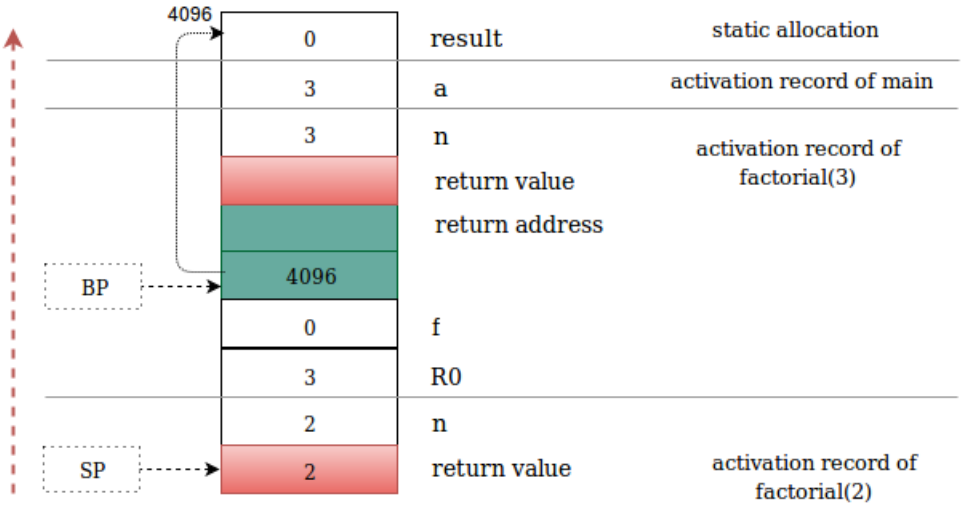
5. Activation record for factorial(1) (called by factorial(2)) is seup similarly.



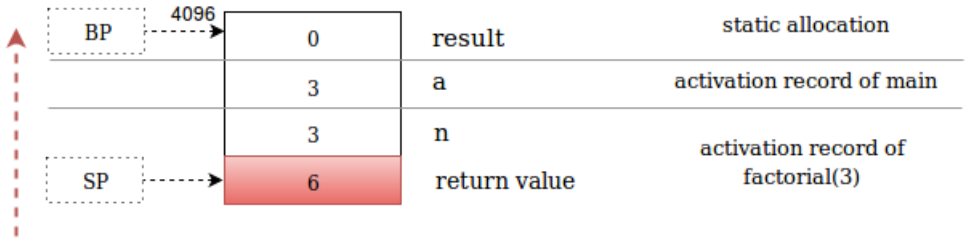
6. factorial(1) calculates the result and returns it by setting the value at return value location and pops off it local variables and sets back the base pointer.



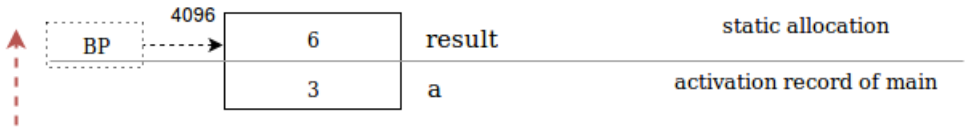
7. Similarly, factorial(2) calculates the steps and pops off its activation record till the result value after setting back the old base pointer.



8. Similarly, factorial(3) also calculates the result and returns it to the main function.



9. Main function calculates and sets the 'result' variable.



CASE – 2) STACK SMASHING

Let's take an example where the code tries to access forbidden regions of computer memory.

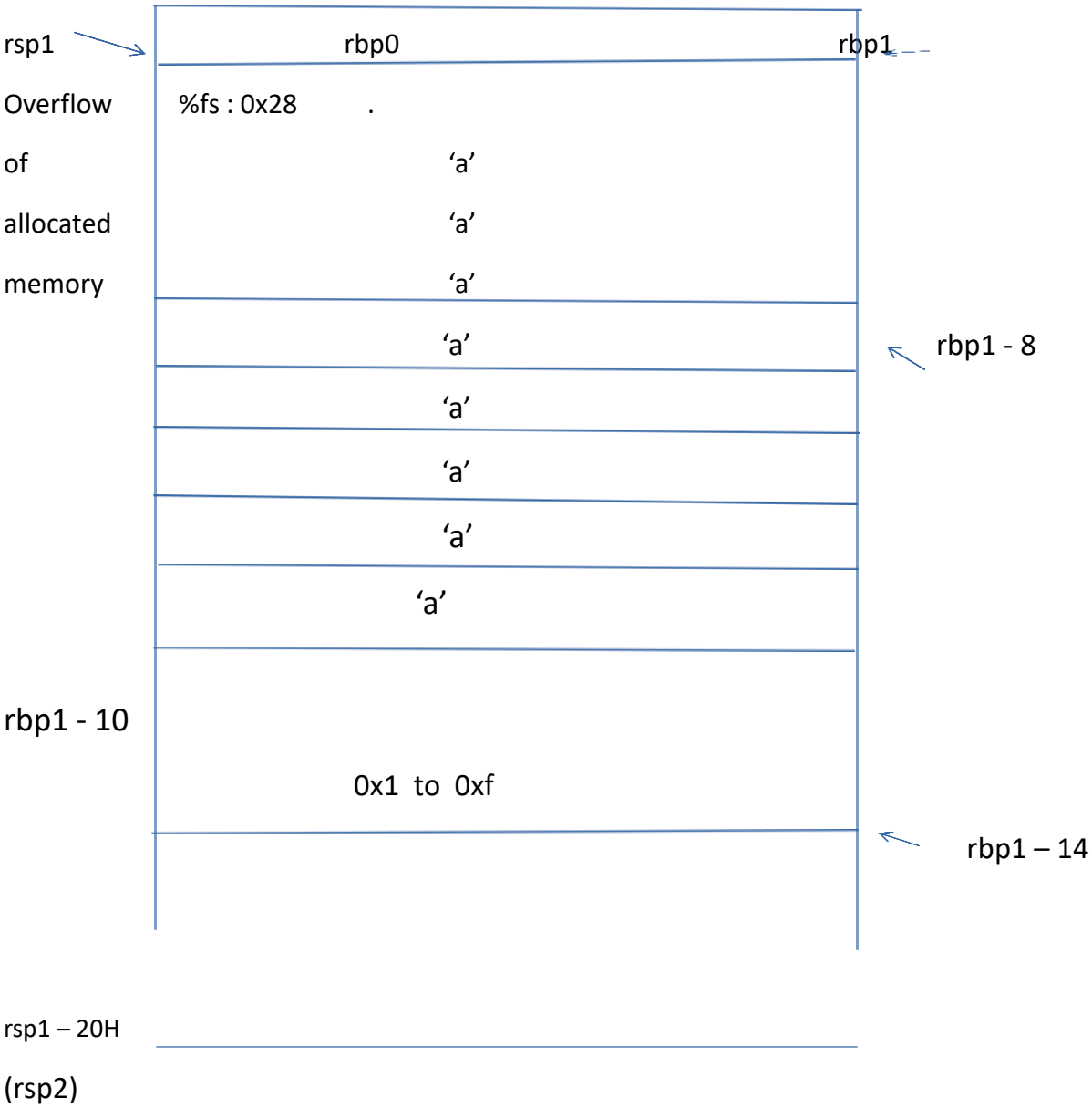
CODE :

```
#include<stdio.h>

int main(){
        char str[10];
        int i;

        for(i = 0; i < 20; i++)
                str[i] = 'a';

        return 0;
}
```



rax --> %fs = 0x28 eax = 0

rdx content is changed due to overwriting of 'a'

So rdx content is no more %fs : 0x28

Therefore , Stack_check_fail is called

And Stack Smashing takes place.