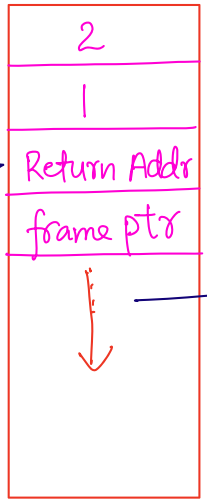
Stack during function call

```
int main() {
    foo(1, 2);
    // do something else
}
```



function arguments

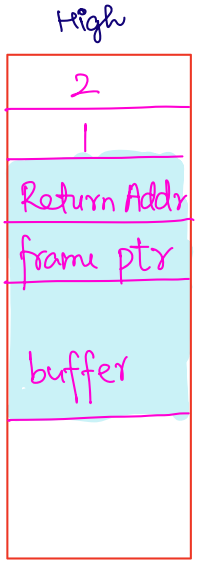
next instruction

→ local variables in this function scope.

Suppose inside foo, we declare a character array of size (x) but memcpy into it, something of size > x.

memcpy(destination, source, num_bytes)

↳ it doesn't implement boundcheck!



What will attacker overflow buffer with?

x
x
x

→ populate this region with 'x'

→ here he will put some malicious code.

↳ let this address be (x)

→ mem copied into buffer, but it has overflowed

↳ return address is modified!!

local char array inside function

⇒ After overflow, the return address will point to (x) and IP will jump to location (x) on function return & start executing bad code.

Note that in this attack, the bad code is injected into stack and IP is fetching instructions from stack and executing them.

↓

We can mark the pages of stack as non-executable to prevent this type of attack.

The attacker instead of polluting the return address with some address on stack (non executable), can replace it to some other instruction address which is executable.

↓

To prevent this, we use ASLR, where we randomise the base addresses of stack, heap, libraries everytime the program is ran.

↓

Attacker would not be able to predict at which location is his desired code present. It would change in every run.