Hack `postgres` Source Code: Vol I

Chapter 1 : C & Rust

1.29 C - Static Scoping

Hmm... C follows `static scoping` , doesn't it?

Yeah. It is designed to obey static scoping rules.

So, how does this static scoping work behind the scenes?

Umm. It is not that simple and bear with me for some time on telling you this.

Ok..

So, think of memory as a box with some compartments. Each compartment in memory is used for some work while a program is executed.

(Writing on white board)..

```
//main-line-0
int _global = 0;

//main-line-2
int main() {

    // main-line-3
    static int _static_local;

    int _local = 1;

    return 0;
}
```

`main-line-0` : Let's assume that there is a compartment named `Initialized Data Segment/ Data Segment` in the memory box. This is used to store any `global` , `static` or `extern` variable's initilaized data. So, when C sees `_global` it just stores value `0` in that compartment.

`main-line-1` : `Stack` is another one which can also be called as `Call Stack` to store function calls in `First In Last Out(FILO)/ Last In First Out(LIFO)` storing order. When C come across any function invocation, it stores that function invocation in this area in the form of `activation record` . `activation record` is something that consist of data about function invocation such as

(1) function input parameters (2) function local variables (3) address of calling function. Back to our `main()`, C stores (1) No input variales (2) `_local` value (We will exclusion of `_static_local` later) (3) No address (I guess).

`main-line-3` : Like `Initialized Data Segment`, there is `Uninitialized Data Segment`. People also call it `bss` that stands for `block started by symbol` (don't know why they just don't stick with name of `Uninitialized Data Segment`). This one is used to store uninitialized `global`, `static` and `extern` variable values. Because of this, the variable `_static_local` doesn't get to stored in stack.

Unlike these compartments, there are two more. One is called `Heap` which is used for dynamic memory allocation. Another one is - `Text Segment` that is used to stored machine code of our program. This compartment generally lies between `Stack` and `Heap` to prevent data overflow from one to another.

Oh.. That's a lot of stuff to grasp. But, where is this `static scopint` ...

Once you know these things, it would be easier to understand that `static scoping`.

Ok.

When we have multiple function calls/multiple blocks, the `static scoping` can be observed explicitly. (writing on white board)

```
// main-line-0
int add(int x, int y);

// main-line-1
int x = 1;

// main-line-2
int main() {

    int x;
    int y = 2;

    // main-line-3
    int z = add(x,y);

}

// add-line-4
int add(int x, int y) {

    // add-line-5
    return (x+y);

}
```

`main-line-0` : function prototype to let the compiler know that there is existence of the function `add` .

`main-line-1` : gloabl variable `x` with `1` stored in `Initialized Data Segment` .

`main-line-2` : function call `main` is stored with an `activation record` consist of (1) No input values (2) Local variable values (3) No calling function address.

`main-line-3` : C invokes function `add` with values in `x` and `y` . C can able figure out the value in `y` in the `main` function scope but it couldn't do the same for `x` being in `main` function scope. So, it has to go outerscope to fetch value for `x` from any global variable if exists. In our case, when it checks in outer scope of `main` , it can see the global variable `x` with value `1` . So, it simply uses this value for `x` and sends those two values to `add` . Here, C just uses an approach of searching the variable in inner scope first and outer scope later. Following this approach by C is what make the people to say that the the C is follower of "static scoping".

`add-line-4` : `add` call is stored with its activation records on top of `main` activation record in stack.

`add-line-5` : As soon as C come across `return` , it just return value to calling function by destroying `add` 's activation record in stack.

Got it..