1. **Static Scoping:**

In static scoping, to get the value of a non-local variable, a function/a nameless block always refers to the top-level scope, or the chain of top-level environments it can access. In our programming language, since there are no nested definitions, the only top-level environment of each function is the global scope. So, static scoping can be demonstrated by having functions refer to the global scope to get non-local variables rather than looking at the value of the variable -if it exists- at the stage of the calling function.

Consider for example a simple code that declares and initializes two variables at the global scope and defines and initializes other variables of the same name at the scope of the main function that calls upon an add and a subtract function that perform the addition and subtraction of the two variables. Note that these variables are not taken as parameters, otherwise they will be considered local variables to the called functions.

The AL code (in our languages) for this simple code would look like the following:

**DEC VAR1 0004** //Declare variable VAR1

**+0 0 0000 0010** //Initialize VAR1 to 10

**DEC VAR2 0004** //Declare variable VAR1

**+0 0 0000 0005** //Initialize VAR2 to 5

**DEC res 0004** //Declare variable res

**+0 0 0000 0000** //Initialize res to 0

**SPR** //Separate data initialization from code memory

**SCP SUM** //Define the scope SUM (i.e. function SUM)

**ADD 9 VAR1 VAR2** //Add VAR1 to VAR2 and store the result in the **acc**

**HLT** //Halt the function SUM

**SCP DIF** //Define the scope DIF

**SUB 9 VAR1 VAR2** //Subtract VAR1 from VAR2 and store the result in the **acc  
HLT** //Halt the function DIF

**SCP MAIN** //Define the scope MAIN

**DEC VAR1 0004** //Declare variable VAR1

**+0 0 0000 0002** //Initialize VAR1 to 2

**DEC VAR2 0004** //Declare variable VAR1

**+0 0 0000 0001** //Initialize VAR2 to 1

**CLL 0 SUM 0000** //Call the function SUM

**MOV 0 0000 res** //Move what’s in the accumulator to variable res

**OUT 9 res 0000** //Print the variable res

**CLL 0 DIF 0000** //Call the function DIF

**MOV 0 0000 res** //Move what’s in the accumulator to variable res

**OUT 9 res 0000** //Print the variable res

**HLT** //Halt the main program

**SPR** //Separate Code Memory part

Output: 15 5

Since the language is supposed to be statically scoped, the functions SUM and DIF, take the values of variables VAR1 and VAR2 from the global scope and not from the caller (MAIN). Therefore, the outputs are: 15 5.

1. **Dynamic Scoping:**

In dynamic scoping non-local references are retrieved from the caller rather than the top-level environment.

If our language was dynamically scoped, then the output to the same AL program above would be 3 and 1 instead of 15 and 5. Because the functions SUM and DIF, would refer to the values of VAR1 and VAR2 in the caller which is function MAIN.

1. **Handling of Jumps in Static and Dynamic Scoping for nested nameless blocks:**

The following code is written in our PL with some constructs that we did not introduce in the HLPL, but they serve the purpose of discussing dynamic scoping. These constructs are namely the labeling **“L:”** and the **goto** statement. Both of these were used in our AL as labels and unconditional jumps.

Let’s consider the following code written in our language:

entier a : 5, b : 5, num : 2.

demarre

debut

entier C.

si(a = b)

debut

L:

si(num = [a \* 3 + 1])

debut

affiche(“num = 16”).

a : 7.

fin

sinon

debut

affiche(“num = 2”).

fin

fin

si(a = b)

debut

entier num = 16.

goto L.

fin

fin

Here are **the important** steps of execution of this code in both cases of scoping:

1. Variables a, b, and num are declared globally and are initialized to 5, 5, 2 respectively.
2. The first if (si) statement (highlighted in blue) is satisfied since a and b are equal.
3. In the inner if (si) statement (highlighted in yellow), since num is not equal to [a\*3+1], then we execute the else (sinon) statement which displays “num = 2”
4. When we reach the if statement highlighted in pink, and which is satisfied, we declare a local variable num (local to the block of the conditional statement), and we initialize it to 16.
5. Then we jump to the label L, which starts with the conditional statement that checks whether num is equal to 16. Here happens the change between static scoping and dynamic scoping.
6. **Static Scoping:**
7. The nameless conditional block would refer to the top-level environment to retrieve the value of num, which will still be 2, and thus it will print “num = 2”.
8. **Dynamic Scoping:**
9. The nameless conditional block would refer to the environment from which the block was called (i.e., the environment from where we jumped) to get the value of num which is 16, and this it would print “num = 16”.