Name: Harshit Jain
User ID: hmj5262

Problem 1

1. Code A and Code B differ in terms of the scope of the variable ”count”.

   In Code A, the variable ”count” is declared as a static variable inside the function ”globalCounter”. This means that the variable ”count” retains its value between function calls, and its scope is limited to the function ”globalCounter”.

   In Code B, the variable ”count” is declared as a local variable inside the function ”globalCounter”, which means that its value is reset to 0 each time the function is called and its scope is limited to the function ”globalCounter”.

2. The output of Code A will be:

   count $= 1$

   count $= 2$

   count $= 3$

   count $= 4$

   count $= 5$

   The output of Code B will be:

   count $= 0$

   count $= 0$

   count $= 0$

Problem 2

1. Yes, C is a statically scoped language. In a statically scoped language, the scope of a variable is determined at compile time and remains fixed throughout the program's execution. In other words, the scope of a variable is the part of the program where it is accessible and can be used.

   In C, the scope of a variable is determined by its location within the source code and the scope rules specified by its declaration.

2. (a) The scope of a static local variable is the block scope. The lifetime of a static local variable is the entire program.

   (b) The scope of a static global variable is the file scope. The lifetime of a static global variable is the entire program.

   (c) The scope of a non-static local variable is limited to the block - block scope. The lifetime of a non-static local variable is limited to the duration of a function call.

   (d) The scope of a non-static global variable is the file scope. The lifetime of a non-static global variable is the entire program.

Problem 3

1. Global Scope:

Name	**Type**
x	int

first():

Name	**Type**
x	int

second():

Name	**Type**
x	int

third():

Name	**Type**
x	int

Code block within third():

Name	**Type**
x	int

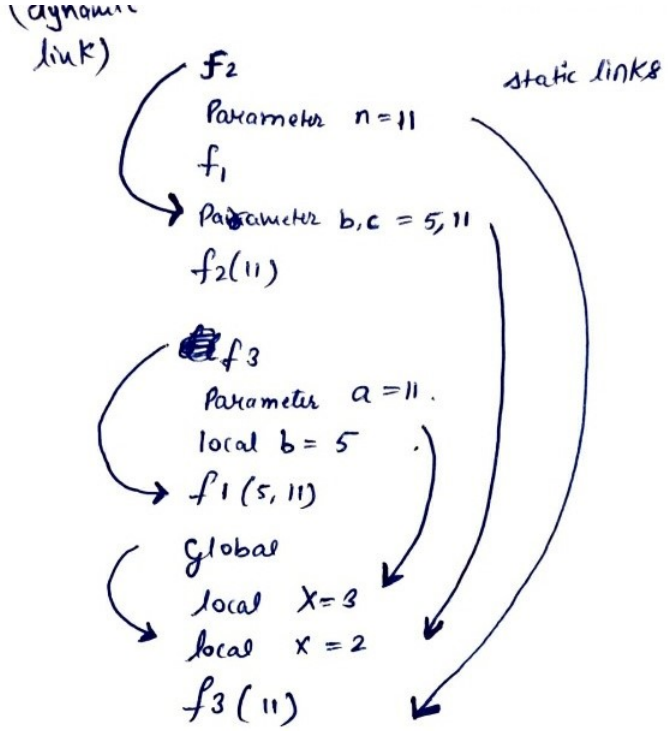
2. If the language uses static scoping rules, the expected output from the print statement would be **25** and **10**.

   When the *third* function is called from the *first* function, the *x* parameter in the *third* function is assigned the value of *x* from the *first* function, which is 25. The *x* in the global scope remains unchanged at 10. When the *second* function is called within the code block in *third*, it will use the value of *x* from the global scope, which is 10.

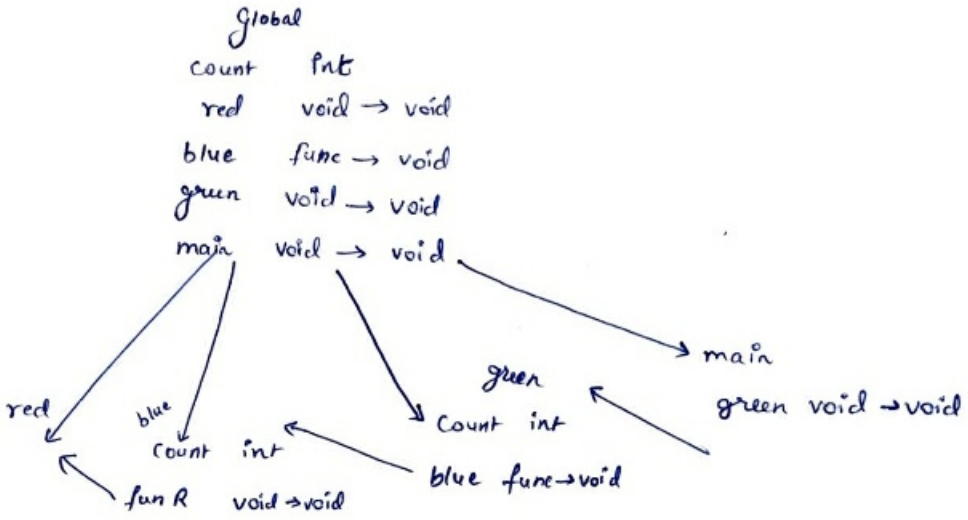
3. If the language uses dynamic scoping rules, the expected output from the print statement would be **30** and **30**.

   Dynamic scoping means that the value of a variable is determined by the most recent call in the call chain. In this case, the *x* in the code block within *third* is assigned the value of 30, which would be used by both *third* and *second*, since *second* is called within *third*.

Problem 4


(dynamic
link)
f_2
static links
Parameter n = 11
f_1
Parameter b, c = 5, 11
$f_2(11)$
f_3
Parameter a = 11.
local b = 5
$f_1(5, 11)$
Global
local X = 3
local x = 2
$f_3(11)$

Problem 5



1. blue 0, green 0, counter value 1
   (a) Global is the top level thus edits global count
2. blue 1, green 0 , counter value 0
   (a) This is because red goes back one later according to the table and thus edits blues local count
3. blue 0, green 1, counter value 0
   (a) This is because red is passed through at green, editing greens local count according to the table (blue accepts red)

Problem 6

1. **Storage Allocation**

   (a) *A* is a static object therefore stored on static

   (b) *C* is assigned to the object created using new therefore stored on heap

   (c) *D* is neither static nor using new therefore stored on stack

   (d) *B* is assigned to the *C* created in foo which is on heap therefore *B* points to heap

   (e) *C* also points to something on heap

2. **Lifetime**

   (a) *A* : Entire program

   (b) *C* : When it gets deleted 13141567

   (c) *D* : It is on stack therefore it's lifetime ends when it goes out of scope 1415