Name:	Harshit Jain
User ID:	hmj5262

Problem 1

1. 35 26 25
   5 10 15
2. 35 26 15
   35 26 15
3. 35 26 25
   5 10 15
4. 55 26 25
   5 10 15

Problem 2

1. 10 39 25
   10 15
2. 44 44 25
   44 15
3. 10 39 25
   39 15
4. 49 49 64
   49 15

Problem 3

1. False. Maintenance of the subroutine call stack is the responsibility of the callee. Explanation: The callee is responsible for saving and restoring the caller's context, including the return address and any callee-saved registers, on the stack before executing the subroutine.

2. True. Modern machines typically pass subroutine parameters in stacks. Explanation: In most modern machines, subroutine parameters are passed in a stack-based calling convention.

3. False. Subroutine calling convention gives the callee the responsibility to save all the registers that it modifies. Explanation: The callee is responsible for saving and restoring any registers that it modifies during the subroutine call, but the caller is responsible for saving and restoring any registers that it needs to preserve across the call.

4. True. Parameter-passing in C is call-by-value as well as call-by-reference (when using pointers). Explanation: In C, function parameters are passed by value by default, but it is possible to pass a pointer to a variable and modify the value indirectly, effectively passing it by reference.

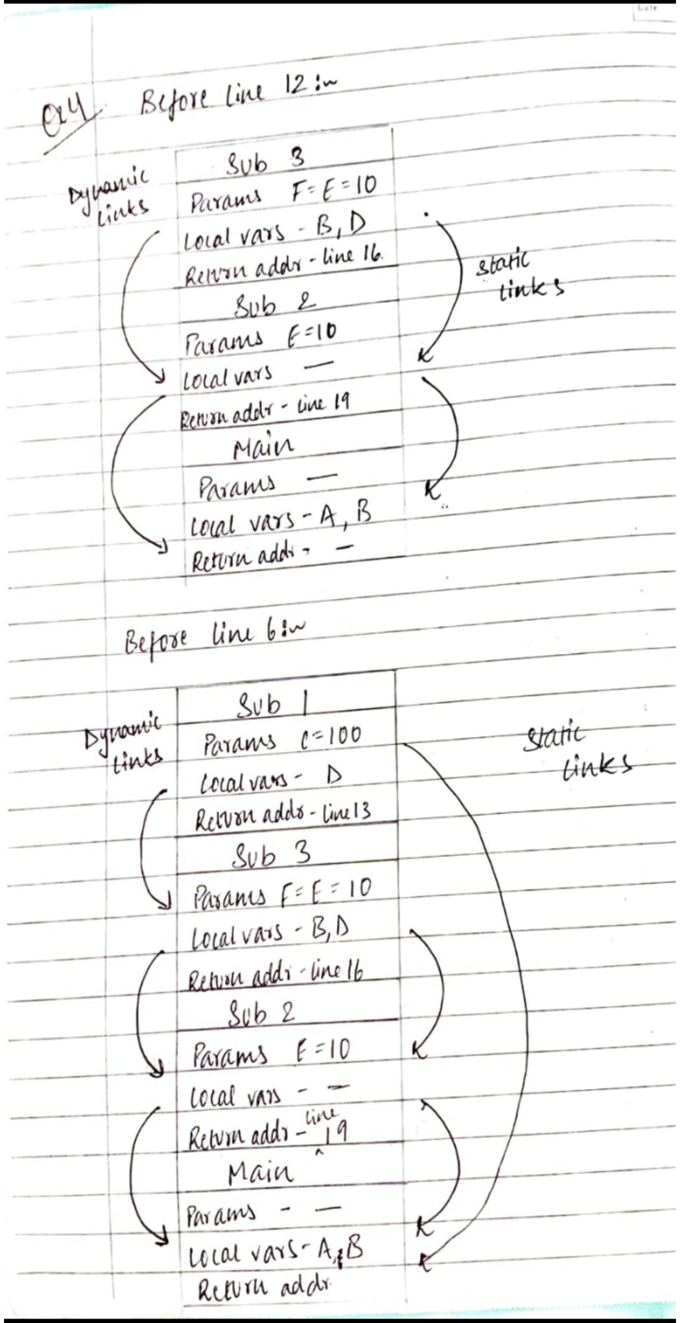
5. False. The calling sequence convention used by a compiler is not specified by the language and can vary depending on the platform and the optimization level. Explanation: The calling sequence convention used by a compiler is not part of the language specification and can vary depending on the platform and the optimization level.

6. True. It is the caller's responsibility to maintain the static chain during a subroutine call. Explanation: The static chain is used to access variables in higher-level scopes, and it is the caller's responsibility to set up the static chain correctly before calling a subroutine.

7. True. Pascal is call by value but can do call by reference with the help of the keyword 'var'. Explanation: In Pascal, function parameters are passed by value by default, but the 'var' keyword can be used to pass a parameter by reference.

8. False. For C++ function header "void f1(int p1, float& p2);", $p1$ is passed by value and $p2$ is passed by reference. Explanation: In the given function header, $p1$ is passed by value, and $p2$ is passed by reference.

Problem 4



Q4. Before line 12:-

Dynamic links

Sub 3
Params F=E=10
Local vars - B, D
Return addr - line 16.
Sub 2
Params E=10
Local vars —
Return addr - line 19
Main
Params —
Local vars - A, B
Return addr - —

static links

Before line 6:-

Dynamic links

Sub 1
Params C=100
Local vars - D
Return addr - line13
Sub 3
Params F=E=10
Local vars - B, D
Return addr - line 16
Sub 2
Params E=10
Local vars - —
Return addr - line 19
Main
Params - —
Local vars - A, B
Return addr

Static links

Problem 5

In call-by-sharing, also known as call-by-object-sharing, the function receives a reference to the object as a parameter, but the function does not receive the object itself. The object is still stored in the caller's memory, and any changes made to the object within the function are reflected in the caller's memory.

Call-by-sharing is similar to call-by-reference, but there is a subtle difference. In call-by-reference, the function receives a reference to the variable itself and any changes made to the variable within the function are reflected in the caller's memory. On the other hand, in call-by-sharing, the function receives a reference to the object, but not to the variable itself. Any changes made to the object within the function are reflected in the caller's memory, but if the function reassigns the reference to a new object, this change is not reflected in the caller's memory.

Call-by-value, on the other hand, passes a copy of the value of the variable to the function. Any changes made to the variable within the function are not reflected in the caller's memory.

Problem 6

1. We should use Dynamic scoping as we want to swap the values in the caller. We would want the arguments to be evaluated in caller's environment.

2. Yes, there will be problems while calling *exchange*(s,t) as we are using pass-by-name. Line 5 will become $t = t$ creating a confusion between a local variable and existing variable. To fix it, we should name the local variable.

3. Yes, there will be a problem here. Since, parameter $'a'$ is being changed first, value of i will change first. This will lead to some different index in array or index-out-of-bounds error. This leads to wrong implementation of function. To fix this problem, we can implement it in a following way:

```
function exchange(a,b)
    var t = b
    start
        b = a
    a = t
    end
```