1 Problem 1 — Lexical/Dynamic Scoping

Assume variable names written as capital letters use dynamic scoping and variable names written as lower case letters use static (lexical) scoping. Assume that procedures return when execution reaches their last statement. Assume that all procedure names are resolved using static (lexical) scoping. Show the output of the entire program execution. Label the output with the location of the print statement (e.g.: (*2*): ...).

program main()
{ int A, b;
  procedure f()
  { int c;
    procedure g()
    { int c;
      c = 35;
      ... = ...b... //<<<---------- (*A*)
      print A,b,c; //<<<---------- (*1*)
      end g;
    }
    print A,b; //<<<---------- (*2*)
    A = 0; b = 0; c = 0;
    call g();
    print c; //<<<---------- (*3*)
    end f;
  }
  procedure g()
  { int A,b;
    A = 3; b = 8;
    call f();
    print A,b; //<<<----------(*4*)
}
end g;
}
A = 1;  b = 2;
print A,b;  //<<<---------- (*5*)
call g();
print A,b;  //<<<----------(*6*)
end main;
}

2  Problem 2 — Lexical/Dynamic Scoping

This problem is an extension of the previous problem.

1. Show the program with all lexically scoped variable names (lower case) replaced by their (level,offset) representation.

2. Show the runtime stack when execution reaches the point marked (*1*), (*3*), (*4*), and (*6*) in the code. Assume that program “main” has its own frame on the stack. Make sure you label all the stack frames with the corresponding program/procedure names and include the allocated local variables (and their particular values) within the frame. Include all control links and access links between the activation records (stack frames), and the value of the frame pointer FP by drawing an arrow to the corresponding location within the stack. Use the frame layout in the figure below.
3. Give the RISC machine code for the non-local access to variable b at program point (*A*). The access will need to load the content of variable b into a register. Use the RISC machine instructions LOAD, STORE, LOADI, ADD, SUB as used in the non-local data access example as discussed in the lecture.
Problem 3 – Parameter Passing

```
program foo()
{
    a, b integer;
    procedure bar(integer x, integer y)
    {
        z: integer;
        
        z = 4;                  /* 1 */
        x = x - y + z;          /* 2 */
        y = 3;                  /* 3 */
    }
    // statement body of foo
    a = 0;
    b = 1;
    call bar(a, b);
    print a, b; }
```

Use the RISC machine instructions LOAD, STORE, LOADI, ADD, SUB as used in the non-local data access example as discussed in the lecture to show the code that needs to be generated for the body of procedure bar (statements /*1*/ through /*3*/). Assume that

1. Register r0 contains the frame pointer (fp) value.

2. Formal parameter x is call-by-reference, and formal parameter y is call-by-value. Assume that bar’s parameters x and y have been correctly initialized as part of the procedure call of bar.
3. Use the stack frame layout as shown above. The figure shows the run-
time stack when the program execution reaches program point /*0*/
in procedure bar.

What values for a and b does the program print?

Problem 4 – Parameter Passing

Assume that you don’t know what particular parameter passing style a pro-
gramming language is using. In order to find out, you are asked to write a
short test program that will print a different output depending on whether a
call-by-value, call-by-reference, or call-by-value-result parameter passing style
is used. Your test program must have the following form:

    program main()
    {   a integer;
        procedure foo(integer x)
        {
            // statement body of foo
        }

        // statement body of main
        a = 1;
        call foo(a);
        print a;
    }

The body of procedure foo must only contain assignment statements. For
instance, you are not allowed to add any new variable declarations.

1. Write the body of procedure foo such that print a in the main program
will print different values for the different parameter passing styles.

2. Give the output of your test program and explain why your solution
works.