1 Problem — Pointers

Given the following correct program in C,

1. give the correct type definitions for pointer variables \( \text{ra}, \text{rb}, \text{rc}, \text{raa}, \text{rbb}, \) and \( \text{rcc} \).

2. draw pictures that show all of the variables and their contents similar to the pictures as shown in lecture 11, pages 11 and 12. Your pictures should show the variables and their values just before the statements marked *1* and *2*, respectively.

3. show the output from this program.

4. write a statement involving a pointer expression using the variables in this program which is ILLEGAL given your declared types.

```c
main() {int a, b, c;
    ??? ra; ??? rb; ??? rc; ??? raa; ??? rbb; ??? rcc;
    a = 5; b = 4; c = 2;
    ra = &a;
    rb = &b;
    rc = &c;
    ra = rb;
    raa = &rb;
    rc = *raa;
*1*    rcc = raa;
    rc = &a;
    rbb = &rc;
    rb = &c;
    *ra = 4;
    *rb = *ra + 7;
*2*    printf("%d %d %d\n",a,b,c);
    printf("%d %d\n",*ra,*rb);
    printf("%d %d\n",**raa,**rbb,**rcc);
}
```
2 Problem — Freeing Memory

Here is a code fragment from our singly-linked list example from class.

```c
/* DEALLOCATE LIST */
for (current_cell = head;
    current_cell != NULL;
    current_cell = current_cell->next)
    free(current_cell);
```

1. Is there a safety issue with this code? Explain.

2. How can you rewrite this code to make it safe? You may introduce new variables, if needed.

3 Compiler Optimization and Aliasing

Assume the following program fragment without any control flow branches (straight line code). Your job is it to implement a compiler optimization called “constant folding” for straight line code. This optimization identifies program variables with values that are known at compile time. Expressions that consist of only such variables can be evaluated at compile time.

```c
begin
    int a, b, c;
    ... /* some other declarations */
    a = 5;
    b = 7;
    ... /* no statements that mention ‘‘a’’ or ‘‘b’’ */
    c = a + b; /* c == 12 ? */
    print c;
end.
```

Would it always be safe for the compiler optimization of constant folding to replace the assignment “c = a + b” by “c = 12”? Note that there are no assignments to variables a or b between “b = 7” and “c = a + b”. The control flow is linear, so there are no branches. If you believe that constant propagation may not be safe, give an example where constant propagation would be indeed unsafe (incorrect), without violating any of the above assumptions about the code fragment. Note: You can add declarations of other variables and other statements that do not mention a or b.
4 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.

program main()
{ int A, b;
  procedure f()
  { int c;
    procedure g()
    { int c;
      c = 33;
      ... = ...b...  //<<<-------- (*A*)
      print A,b,c; //<<<-------- (*1*)
    end g;
    print A,b; //<<<------------- (*2*)
    A = 1; b = 2; c = 3;
    call g();
    print c; //<<<------------- (*3*)
    end f;
  } procedure g()
  { int A,b;
    A = 4; b = 9;
    call f();
    print A,b; //<<<-------------(*4*)
  end g;
  A = 5;  b = 3;
  print A,b; //<<----------------- (*5*)
  call g();
  print A,b; //<<-----------------(*6*)
  end main;
}

1. Show the output of the entire program execution. Label the output with the location of the print statement (e.g.: (*2*): ...)).

2. Show the stack-frames on the runtime stack when the execution reaches statement (*A*). Use the conventions used in class where the runtime stack grows downwards, i.e., the top stack frame corresponds to the main procedure. Within each stack frame, only show the contents of the local variables.