1 Problem — Pointers

Given the following correct program in C,

1. give the correct type definitions for pointer variables ra, rb, rc, raa, rbb, and rcc.

2. draw a picture that shows all of the variables and their contents similar to the picture as shown, for example, in lecture 12, page 8. Also indicate whether the object lives on the stack or on the heap. Your picture should show the variables and their values just before the first print statement (*).

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
int main() {int a, b, c;
    ??? ra; ??? rb; ??? rc; ??? raa; ??? rbb; ??? rcc;
    a = 3; b = 2; c = 1;
    ra = &a;
    rb = &b;
    rc = &c;
    ra = rb;
    raa = &rb;
    rc = *raa;
    rcc = raa;
    rc = &a;
    rbb = &rc;
    rb = &c;
    *ra = 4;
    *rb = *ra + 4;
    /* (*) */
    printf("%d %d %d\n",a,b,c);
    printf("%d %d\n",*ra,*rb);
    printf("%d %d %d\n",**raa,**rbb,**rcc);
}
2 Problem — Freeing Memory

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

/* 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 can introduce new variables, if needed.

3 Problem — Compiler Optimization and Aliasing

Assume the following program fragment without any control flow branches (straight line code). Your job is 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. In our project, we do constant folding for ILOC instructions, not on the source code itself.

begin
    int a, b, c;
    ... /* some other declarations */
    a = 4;
    b = 3;
    ... /* no statements that mention 'a' or 'b' */
    c = a - b; /* c == 1 ? */
    print c;
end.

Would it always be safe for the compiler optimization of constant folding to replace the assignment “c = a - b” by “c = 1” ? Note that there are no assignments to variables a or b between “b = 3” and “c = a - b”. The control flow is linear, so there are no branches. Give an example where constant propagation would be not be safe (incorrect) in this situation, 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  Problem — Heap vs. Stack

Briefly address the following questions (2-3 sentences):

1. Why can we not dynamically allocate objects on the stack? That way, we would avoid having to deal with the heap.

2. Your language definition allows the runtime stack to contain at most one stack frame for each procedure at any point in time during program execution. Do you still need a runtime stack?

3. Could we change our runtime environment to allocate all stack frames on the heap? If yes, what are the advantages / disadvantages?