Assignment 4: Memory Management in SaC

Task 1: Reference counting in SaC

Consider the following SaC code:

```
#define BILLION 1000000000

int foo (int[n] a)
{
    b = modarray( a, [0], a[0] + 1);
    c = modarray( b, [1], a[1] + 42);
    res = c[1] + c[2];
    return res;
}

int bar (int[n] a)
{
    if( foo (a) == 44) {
        res = a[22];
    } else {
        res = a[2] + a[14];
    }
    return res;
}

int main() {
    a = genarray ( [BILLION], 0);
    x = bar (a);
    return x;
}
```

Sketch how the above code would be compiled into pseudo C code using the following pseudo instructions for dealing with memory and reference counting:

`ALLOC_INT (<size>, <id>)` for allocating `<size>` many integer elements and initialising them with `0`s and allocating a reference counter with an initial value of `1`.

`COPY_DATA (<size>, <from-id>, <to-id>)` for copying `<size>` many integer elements from the memory pointed to by `<from-id>` to the memory pointed to by `<to-id>`.

`INC_RC (<id>, <num>)` for incrementing the reference counter of the array `<id>` by the value `<num>`.

`DEC_RC (<id>, <num>)` for decrementing the reference counter of the array `<id>` by the value `<num>` and potentially freeing the memory in case the reference counter reaches `0`.

`RC (<id>)` for reading the reference counter of the array `<id>`

Hand-out: 24/02/2025 Hand-in: 6/03/2025

`SIZE (<id>)` for obtaining the size of the array `<id>` (in elements)

You can use `<id>[<num>]` for denoting data selections in C and `<id>[<num>] = <expr>;` for denoting destructive updates in C. You can omit all variable declarations.

Apply the **naive compilation scheme** outlined in the lecture which translates each application of a function and each application of a built-in operation in the same way, irrespective of the context it occurs in.

Task 2

Annotate your pseudo C code with C-style comments that denote the reference counter values for all vectors during runtime. Use this to validate that all memory that will be allocated at runtime also is being freed! You can assume that the function `genarray` allocates the array `a`, populates it with `0`'s, and returns it with `RC (a) == 1`.

Task 3

Think about possible ways to optimise reference counting. How can you optimise the function `foo`? How can you optimise the function `bar`? Think about ways that are concerned with reference counting operations only **and** think about ways that rely on possible semantics-preserving code-reorderings that can be done before injecting reference counting. Repeat Task 2 for the optimised code.

Task 4

Would it be possible to get rid of all increment operations on reference counts in the given example? If so, what consequences would this have in general?