Exercise 1: Optimization pitfalls in C 10 Points

For the following examples, say if the optimization is correct or not. State your assumptions if you think it is correct or give a reason why it could be incorrect. A few words is enough!

Assumptions:

- C11 language standard (in case you want to be thorough...)
- no multi-threading
- the rest of the program could be anything, but contains no errors or undefined behavior
- all variables are read at the end of the program
- reading uninitialized variables does not result in undefined behavior (no trap representations, not in register)
- programs containing undefined behavior can be replaced by anything

1.

```c
int a;
a = f();
a = 0;
```

2.

```c
int a;
a = 0;
if(a){
    f();
}else{
    g();
}
```

3.
int a;
do {
    a = b;
} while(a);

do {
    a = b;
} while(a);

c = a+10 <= b+10;
c = a <= b;
c = a+10 <= b+10;
c = a <= b;

c = a+10 <= b+10;
c = a <= b;

int a,c;
float b;
a = c; b = c;
c = a != b;
c = 0;

float a,b,c;
c = (a-b)+b;
c = a;

int a,b;
b = a/a;
b = 1;

int a[5];
struct { int b; } b;
int c;
c = a != &a[0] ||
    &b.b != (int*)&b;

void *a,*b;
a = malloc(1000);
a = 0;
b = malloc(1000);
b = malloc(1000);

Solution:

1. Not correct if \( f() \) has side effects.

2. Correct since the first branch will always be dead.
3. Not correct for volatile int b;

4. Correct since signed integer overflow is undefined behavior.

5. Not correct if b + 10 overflows but a + 10 does not (unsigned integer overflow is defined (wraps around)).

6. Not correct since not all int values can be represented by float (gaps between integers once the significants bits are exhausted), e.g., 16777217. Replacing a != (int)b by 0 would therefore be wrong. But also the case above can be wrong: a is implicitly cast to float (comparisions need the same type, see casting rules), so one would assume that the loss of precision appears on both sides, but this need not be the case since the results of floating expressions may be represented in greater precision and range than that required by the type, e.g., when values are left on the FPU-stack (e.g. b is saved in a 64 bit register, a will be implicitly cast to float (64 bit) but the type of the expression (float) a is allowed to be more precise and therefore allowed to stay in a FPU-register which is 80 bit wide (to reduce rounding errors)).

7. Not correct if it comes to catastrophic cancellation (see loss of significance). Other example: b ∈ {∞, −∞, NaN}.

8. Correct since division by zero is undefined. Most implementations abort with a floating point exception.

9. Correct since the first item of an array or struct is guaranteed to be at offset zero. Apart from that there are no guarantees about the memory layout! The compiler might add padding between fields for better alignment. Also, comparison of pointers to different variables is undefined behavior.

10. Debatable: Not correct if the first malloc makes the second (always) run out of space and you assume that optimizations are not allowed to change memory consumption. We want to preserve the behavior, but we usually also want to allow programs that are faster and/or need less memory.

Exercise 2: Simple imperative language, CFG, state 10 Points

Consider the following program code.

```c
int r = 0;
for (int i=0; i<n; i++)
    r += a[i] * a[i];
```

a) Translate the program to our simplified imperative language (don’t compute the index in an extra register for loads).

b) Draw the corresponding Control Flow Graph.

c) Write down the path (π = …) that represents the computation of the program with n = 1.
d) Give the states for the first 4 target nodes and the last node of the path with the starting state $s_0 = (\rho, \mu) = (\{A_0 \rightarrow 0, n \rightarrow 3\}, \{0 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 4\})$.

Solution:

a) $r = 0$;
   $i = 0$;
   L:
   if $(i < n)$ {
   $R_1 = M[A_0 + 1 \cdot i]$;
   $R_2 = M[A_0 + 1 \cdot i]$;
   $R_3 = R_1 \cdot R_2$;
   $r = r + R_3$;
   $i = i + 1$;
   goto L;
   }

b)
c) 
\[ \pi = (0, r = 0, 1)(1, i = 0, 2)(2, \text{Pos}(i < n), 3)(3, R_1 = M[A_0 + 1 \cdot i], 4) \]
\[ (4, R_2 = M[A_0 + 1 \cdot i], 5)(5, R_3 = R_1 \cdot R_2], 6) \]
\[ (6, r = r + R_3, 7)(7, i = i + 1, 2)(2, \text{Neg}(i < n), 8) \]

d) 
\[ s_1 = \llbracket r = 0 \rrbracket s_0 = (\rho_0(r \mapsto [0] \rho_0), \mu_0) = (\{A_0 \mapsto 0, n \mapsto 3, r \mapsto 0\}, \mu_0) \]
\[ s_2 = (\rho_1(i \mapsto 0), \mu_0) \]
\[ s_3 = s_2 \]
\[ s_4 = (\rho_3(R_1 \mapsto \mu_0([A_0 + 1 \cdot i] \rho_3)), \mu_0) = (\rho_3(R_1 \mapsto 2), \mu_0) \]
\[ s_8 = (\{A_0 \mapsto 0, n \mapsto 3, r \mapsto 29, i \mapsto 3, R_1 \mapsto 4, R_2 \mapsto 4, R_3 \mapsto 16\}, \mu_0) \]