1. (1 point) [Invariant checking] Show how loop invariants can be checked, where the loop is given in the form of a do-while template and when it is given in the form of a for template.

2. (1 point) [SSA] Show the SSA form corresponding to the unfolding of the following program. (Unfold for-loop 3 times and inline function call.) Add assertion after the loop that $x < y$ and construct the formula representing this program.

```c
int main(int x, int y)
{
    int result;
    if(x < y)
        x = x + y;
    for (int i = 0; i < 3; ++i)
    {
        y = x + Next(y);
    }
    result = x + y;
    return result;
}

int Next(int x){
    return x + 1;
}
```

3. (1 point) [Invariants] Consider the piece of code. Use the over-approximation technique to check its safety. Find and use invariant to refine the abstraction if necessary.

```c
state_of_lock = unlocked;
do {
    assert(state_of_lock == unlocked);
    state_of_lock = locked;
    old_count = count;
    request = GetNextRequest();
    if (request != NULL) {
        ReleaseRequest(request);
        assert(state_of_lock == locked);
        state_of_lock = unlocked;
        ProcessRequest(request);
        count = count + 1;
    }
} while (old_count != count);
assert(state_of_lock == locked);
state_of_lock = unlocked;
```

4. (1 point) [SSA with pointers] Assume that the program only contains variables of type int and int*, and that dereferenced pointers are only read. Explain how to build SSA with this restriction. Apply your method to the program below.

```c
void my_function(int *p) {
    int j = 0, *q = &j;
    *p = *p + *q;
}
```