ECS 140A Homework 6 – Problem 4

1 Prolog

Step 3: Working Code

```
my_reverse(X, Y) :- my_reverse_helper(X, Y, []).
my_reverse_helper([X | T], Y, B) :- my_reverse_helper(T, Y, [B | X]).
my_reverse_helper([], Y, Y).
```

Step 4: Debug Process

Bug 1

```
?- my_reverse([1, 2, 3], [3, 2, 1]).
false.
```

This test case resulted in false because I appended items from `X` to the buffer `B` in order as I saw them. Instead, it should have been prepended to the items in `B`.

Fixed code:

```
my_reverse(X, Y) :- my_reverse_helper(X, Y, []).
my_reverse_helper([X | T], Y, B) :- my_reverse_helper(T, Y, [X | B]).
my_reverse_helper([], Y, Y).
```

Bug 2

```
?- my_reverse(X, [1, 2, 3]).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 0.7Gb, global: 0.2Gb, trail: 34.1Mb
ERROR:   Stack depth: 4,472,949, last-call: 0%, Choice points: 4,472,941
ERROR:   Possible non-terminating recursion:
ERROR:     [4,472,949] user:my_reverse_helper(_53680972, [length:3], [length:4,472,938])
ERROR:     [4,472,948] user:my_reverse_helper([length:1|_53681016], [length:3], [length:4,472,937])
   Exception: (4,472,948) my_reverse_helper(_53680888, [1, 2, 3], [_53680886, _53680874, _53680862, _53680850, _53680838, _53680826, _5
3680814, _53680802|...]) ? abort
% Execution Aborted
?- my_reverse([1, 2, 3], X).
X = [3, 2, 1].
```

This code is able to verify if the lists are the reverse of each other. Interestingly, I am able to get the correct output when I query with a variable as the second argument, but not as the first as seen above. It looks like there is infinite recursion occurring. To fix this, I put the base case like fact before the helper rule since Prolog tries to find solutions individually and in order.

Fixed code:

```
my_reverse([H | T], Y) :- my_reverse_helper(T, Y, [H]).
my_reverse_helper([], Y, Y).
my_reverse_helper([H | T], Y, B) :- my_reverse_helper(T, Y, [H | B]).
```

Bug 3

```
?- my_reverse([], []).
false.
```

This output false because I forgot to account for when the first list is empty.

Fixed code:

```
my_reverse([], []).
my_reverse([H | T], Y) :- my_reverse_helper(T, Y, [H]).
my_reverse_helper([], Y, Y).
my_reverse_helper([H | T], Y, B) :- my_reverse_helper(T, Y, [H | B]).
```

Results of given test cases:

```
?- my_reverse([1, 2, 3], [3, 2, 1]).
true.

?- my_reverse(X, [3, 2, 1]).
X = [1, 2, 3] .

?- my_reverse([3, 2, 1], X).
X = [1, 2, 3].
```

Results of additional test cases:

```
?- my_reverse([], []).
true.

?- my_reverse([], [1]).
false.

?- my_reverse([1], []).
false.

?- my_reverse([2, 4, 6], [1, 3]).
false.

?- my_reverse([1, 3], [2, 4, 6]).
false.

?- my_reverse([5, 10, 15, 20, 25, 30], [30, 25, 20, 15, 10, 5]).
true.

?- my_reverse([5, 10, 15, 20, 25, 30], [30, 25, 25, 15, 10, 5]).
false.

?- my_reverse([5, 10, 15, 25, 25, 30], [30, 25, 20, 15, 10, 5]).
false.

?- my_reverse(X, []).
X = [] .

?- my_reverse([], X).
X = [].

?- my_reverse(X, [1]).
X = [1] .

?- my_reverse([1], X).
X = [1].
```

Step 5: Add Documentation

```
my_reverse([], []). % base case for empty list
my_reverse([H | T], Y) :- my_reverse_helper(T, Y, [H]).
my_reverse_helper([], Y, Y). % base case when run out of chars to add to buffer
my_reverse_helper([H | T], Y, B) :- my_reverse_helper(T, Y, [H | B]). % keep adding first
    char to buffer and repeat on tail
```

Step 6: Extra Test Cases Used

- `my_reverse([], [])`: true
- `my_reverse([], [1])`: false
- `my_reverse([1], [])`: false
- `my_reverse([2, 4, 6], [1, 3])`: false
- `my_reverse([1, 3], [2, 4, 6])`: false
- `my_reverse([5, 10, 15, 20, 25, 30], [30, 25, 20, 15, 10, 5])`: true
- `my_reverse([5, 10, 15, 20, 25, 30], [30, 25, 25, 15, 10, 5])`: false
- `my_reverse([5, 10, 15, 25, 25, 30], [30, 25, 20, 15, 10, 5])`: false
- `my_reverse(X, [])`: X = []
- `my_reverse([], X)`: X = []
- `my_reverse(X, [1])`: X = [1]
- `my_reverse([1], X)`: X = [1]