# Test Plan - Polynomials

James Viner

## Public Functions

`poly_create_term()`

Specific tests for this function were not written given that the code was provided by the spec and the success of all other tests is predicated on its ability to function as intended.

`poly_destroy()`

Specific tests for this function were not written given that the code was provided by the spec and the success of all other tests is predicated on its ability to function as intended.

`poly_poly_print()`

Specific tests for this function were not written given that the code was provided by the spec.

`poly_to_string()`

Testing for this function involves ensuring the correct string conversion of a given polynomial in the case of both positive, negative, and terms whose coefficient becomes zero before conversion.

`poly_add()`

Testing for this function involves ensuring a given polynomial successfully combining both given polynomials and returning the resultant chain in order.

`poly_sub()`

Testing for this function involves ensuring the given polynomial in the second argument has its terms successfully modified to have coefficients with inverse signs as well as combining both polynomials and returning the resultant chain in order.

`poly_equal()`

Testing for this function involves comparing a set of identical polynomials as well as several sets of polynomials that differ in some way, such as signage, number of terms, or coefficient/exponent values.

`poly_eval()`

Testing for this function involves comparing the resultant double to the known-correct solution to ensure validity.

`poly_iterate()`

Testing for this function involves ensuring the user's arbitrary transform function is called recursively on each node in the chain and that the resultant terms are sorted and with unique exponents.

## Private Functions

`sort_poly()`

Testing for this function involves performing a transformation that normally would leave terms out-of-order without explicit sorting and validating the result. This function may also be combined with a simplification function later, and if that becomes the case, testing will also need to ensure resultant polynomials are in their most simplified form.

`swap_poly()`

Specific tests for the function were not written given that the code is a small helper function for sort_poly and the success of that function is predicated on this function's ability to perform as intended.

`sizeof_poly()`

Specific tests for the function were not written given that the code is a small helper function for poly_string and the success of that function is predicated on this function's ability to perform as intended.

## Non function-specific Testing

Testing in this section will be combined into testing from the previous sections, although some edge-cases may get their own specific tests as they arise during development.

### `Zero as Coefficient`

All functions will need to be able to avoid printing or performing operations on all functions with zero as a coefficient.

### `NULL pointers`

All functions will need to be able to avoid performing operations on NULL pointers and return an appropriate value when they do so.

### `NULL polynomial`

All functions will need to be able to avoid performing operations on NULL pointers and return an appropriate value when they do so.

### `Correct Signage`

All functions will need to display the correct signage symbols for any particular output.