Design Plan - Polynomials

James Viner

Project Summary

The library libpoly is a suite of functions that enable the creation, simplification and basic usage of polynomials where ’x’ is the only allowed variable.

Features Targeted

Automated Testing

Write automated unit tests for all appropriate (I'm interpreting this to mean 'public') functions. Anything the user of the library will be interacted with should be tested for the most common use cases as well as any particular edge-cases that I expect might cause issues (such as a subtraction operation that results in 0 as the answer for the entire polynomial).

Polynomial multiplication

Enable the multiplication of polynomials through use of this function. This will involve walking the linked list, multiplying all existing coefficients from the first argument by the second argument's coefficient, and adding the exponents in the same way. Afterwards, simplification will need to be performed to ensure each node is still in the correct place.

Polynomial simplification

Make the finished polynomial from any given mathematical operation be simplified down as much as possible. This shouldn't be too difficult, just requiring the actual mathematical portion of any math-intensive function to be written properly along with having a separate function that re-orders each node making up the polynomial in descending order by exponent size.

Architecture

Data

Each individual term in a given polynomial will be made up of an integer coefficient, an unsigned integer exponent, a pointer to the previous node, and a pointer to the next node, making the overall structure a doubly-linked list. The reason for going for a doubly-linked list over a normal linked list would be for ease of sorting, allowing an insertion sort algorithm to be implemented directly on the list.

Significant Functions

```
struct term *poly_create_term(int coefficient, unsigned int exp)
```

Creates a new term by some given combination of an exponent and a coefficient.

```
void poly_destroy(polynomial *p)
```

Frees memory used by a polynomial.

```
void poly_print(const polynomial *p)
```

Prints a polynomial to stdout.

```
char *poly_to_string(const polynomial *p)
```

Returns a malloced string for a given polynomial with the parameter as x. This should be doable by walking the linked list and building out the finished string one node at a time.

```
polynomial *poly_add(const polynomial *a, const polynomial *b)
```

Returns a malloced polynomial summing two given polynomials together.

```
polynomial *poly_sub(const polynomial *a, const polynomial *b);
```

Returns a malloced polynomial subtracting the second polynomial from the first. This can be done by calling the poly_add function with the second argument's coefficient multiplied by -1.

```
bool poly_equal(const polynomial *a, const polynomial *b)
```

Returns true if both polynomial arguments have the same terms, otherwise false. This should involve walking both polynomial node chains at the same time and comparing the exponents/coefficients.

```
double poly_eval(const polynomial *p, double x)
```

Evaluates a polynomial by substituting the 'x' for a given value.

```
void poly_iterate(polynomial *p, void (*transform)(struct term *))
```

Calls an arbitrary transform function on each term in a given polynomial.

```
static void simplify_poly(polynomial *p, unsigned int idx)
```

Private function that simplifies a given polynomial node chain and re-orders it in descending order by exponent. Will need to be called at least once in all other functions to ensure that everything is ordered properly and simplified.

User Interface

Users of the library will be able to use all public functions in order to create, transform, and destroy polynomials. Helper functions will not be publicly accessible, but provide additional internal-use utility to the public functions in the library.

General Approach

The objective here will be to start by writing some of the necessary functions for combining individual terms into complex polynomials like poly_add or poly_sub, then moving onto the more complex ones that involve mathematical operations or recursively performing operations on the polynomial node chain. Helper functions will be added as necessary to provide internal functionality that doesn't need to be explicitly shown to the user.