Creating a Data Type in C

Polynomial functions are one of the most basic and useful elements of mathematics. Recall that a polynomial of degree $n$ is a function of the form:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

The coefficients $a_i$ are constants and the coefficient $a_n$ cannot be zero. Very few programming languages have library support for polynomials.

For this assignment, you will use the `struct` mechanism in C to implement a data type that represents polynomial functions. In order to avoid issues of numerical precision, we will restrict our attention to the case that the coefficients and $x$ are integer values. So, a polynomial might be modeled using the C `struct`:

```c
struct Poly {
    uint8_t Degree;    // degree between 0 and 255 inclusive
    int64_t *Coeff;    // array of coefficients (allocated dynamically)
};

typedef struct Poly Polynomial;
```

So, a `Polynomial` object $P$ that represents $p(x) = 17 + 9x + 23x^2 + 44x^3$ would be initialized like this:

```
P Degree: 3
Coeff:   17 9 23 44
```

Every properly initialized `Polynomial` object $P$ is required to satisfy the following polynomial contract:

1. $P.Coeff$ points to an array of dimension $P.Degree + 1$.
2. $P.Coeff[k]$ is the coefficient of $x^k$, for $k = 0$ to $P.Degree$.

A data type consists of a collection of values and a set of operations that may be performed on those values. For a polynomial type, it would make sense to provide some mathematical operations, such as:

- computing a scalar multiple of a polynomial
- computing the sum and difference of two polynomials
- computing the derivative of a polynomial
- evaluating a polynomial at a given value of $x$

For example:

```c
/**
 * Initializes *Scaled to represent K times *Source
 * Pre:  Scaled != NULL,
 *       Source != NULL,
 *       Source != Scaled
 * Post: Source->Degree == Scaled->Degree
 *       Source->Coeff != Scaled->Coeff
 *       Scaled->Coeff[i] == K * Source->Coeff[i] for i = 0:Scaled->Degree
 * Returns: false if *Scaled cannot be properly initialized, true otherwise
 */
bool Polynomial_Scale(Polynomial* const Scaled, const Polynomial* const Source, const int64_t K);
```
A few points should be made. The polynomial type raises a deep-copy issue, since the coefficient array is allocated dynamically. Since C does not provide automatic support for making a deep copy of structured variables, the functions we will implement are designed to pass pointers to polynomial objects.

This provides an excuse to make good use of the const qualifier, applied to the pointer and/or its target, as appropriate. In the case of the function above, there is no logical reason the function should modify the polynomial object *Source, or the pointer to it which is passed into the function, so both are qualified as const, and while the function does need to modify the polynomial object *Target, there is no reason for it to modify the pointer to it.

Here is the rationale for each of the post-conditions for Polynomial_Scale():

- Scaled points to a Polynomial object
- Source points to a properly-initialized Polynomial object

  The function must be given access to two Polynomial objects. The target of Source must be a properly initialized Polynomial object. If not, the function cannot be expected to initialize the target of Scaled in a way that makes sense.

- Source !≠ Scaled

  The function must be given access to two different Polynomial objects. It is likely that your implementation of this would deal correctly with the case that Scaled and Source point to the same object, but that is not a requirement for the assignment. In a library version of this data type, it would actually be useful to allow Scaled and Source to refer to the same object, since that would give the user an easy way to modify an existing Polynomial, rather than being required to create a new one.

Here is the rationale for each of the post-conditions for Polynomial_Scale():

- Scaled->Degree == Source->Degree
- Scaled->Coeff != Source->Coeff

  The scaled polynomial obviously has the same degree as the original; on the other hand, the scaled polynomial is entirely distinct from the original, so they should certainly not share a coefficient array.

- Scaled->Coeff[i] == K * Source->Coeff[i] for i = 0:Scaled->Degree

  Each coefficient of the scaled polynomial must be the appropriate multiple of the corresponding coefficient in the original polynomial.

There are also some hidden issues with designing this function. The preconditions guarantee that *Scaled exists, but say nothing about the content of the target. If *Scaled already has a pointer to an array, the function must be careful to not create a memory leak. On the other hand, we would not want to perform an unnecessary memory allocation.

There is also the question of whether stated preconditions should be checked within the function. The need for efficiency would argue against; after all, the preconditions have been stated, so it's the caller's fault if they are not satisfied, and checking them would require extra steps at runtime. And, some preconditions are essentially impossible to check. The need for robustness would argue in favor of checking (checkable) preconditions, if violations of them could result in serious runtime errors.

You should consider these points carefully when designing your solution to this assignment.

See the posted C header file for a full list of the required functions and their interfaces.

There are a number of other useful mathematical operations, such as multiplication of two polynomials, which we will not support, in order to keep this assignment reasonably small. A practical implementation would have many more features.
Besides purely mathematical operations, in order to support client code, we will also support the following operations:

- a "constructor" to initialize a new polynomial object
- an function to make a deep copy of a polynomial object
- a function to set a polynomial object to represent the zero polynomial

**Some C Notes**

We will compile your code with gcc version 4.4 or 4.5. I am not aware of any differences between those versions that would cause any differences in results. We will not test your code with any other version, or with any other compiler.

We will compile your code with the following switches: `-std=c99 -Wall`. Expect problems if you use additional switches, or if you do not use both of those.

We do not require your code to compile without warnings, although that is certainly desirable. You are strongly encouraged to be sure that you understand any warning messages you see when compiling your code, and that you are sure that any warning messages you do not eliminate are benign.

**What to Submit**

You will submit a single `.c` file, containing nothing but the implementation of the specified C functions. Be sure to conform to the specified function interfaces.

Your submission will be compiled with a test driver and scored according to how many cases your solution handles correctly. Your submission that achieves the best score will also be evaluated by a TA, who will assess the quality of your design and implementation. Particular attention will be paid to computational efficiency, and how you handle memory management issues.

You will be allowed up to ten submissions for this assignment. Test your function thoroughly before submitting it. Make sure that your function produces correct results for every test case you can think of.

The *Student Guide* and other pertinent information, such as the link to the proper submit page, can be found at:

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you must include the following pledge statement in the submitted file:

```c
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

**Failure to include this pledge in a submission is a violation of the Honor Code.**