CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #10: ADT SPARSE POLYNOMIALS

DUE DATE WITH BRIGHTSPACE: NOVEMBER 30 AT 11:50 PM

THE LEARNING OUTCOMES

- to implement the Abstract Data Type to manipulate a sparse polynomial
- to implement a C++ class that uses linked lists with the value field as a pointer to a struct rather than just to an int
- to practice deep copying and deallocating dynamic memory (that goes on the heap)
- to program without memory leaks and dangling pointers
- to use a merge type algorithm for data that is sorted

TO IMPLEMENT

We will represent a sparse *polynomial with integer coefficients* using the following C++ structures.

```
struct Term {
    int coeff;
    int degree;
};


struct Node {
    Term *term;
    Node *link;
};
```

Use a linked list to represent the ***non-zero*** terms of a polynomial (this data structure is appropriate for sparse polynomials).

For example, given the polynomial

$$\mathbf{p = 2 - 3x^3 + 4x^5}$$

we will represent this sparse polynomial as a linked list of Terms of type Node, ordered by increasing degree. That is, the terms are

2*x^0 -3*x^3 4*x^5

The *degree* of the polynomial p in x is the exponent of the x in the term with the highest exponent. For the example above, the degree of *p* is 5.

We could build a linked list of terms for the polynomial p using the following C++ code.

```
Term *t1,*t2,*t3;
t1 = new Term;   // allocate memory for the term
t1->coeff = 2;
t1->degree = 0;
t2 = new Term{-3, 3};
t3 = new Term{4, 5};

Node * p, *q, *r;
p = new Node;   // allocate memory for the node
p->term = t1;
q = new Node;
q->term = t2;
r = new Node;
r->term = t3;
p->link = q;
q->link = r;
r->link = nullptr;
```

But instead we will overload the >> operator to read in the polynomial. First he number of terms is given and then pairs of coefficient and degree corresponding to the terms (given in any order).

Provided for you is the file poly.h with the functionality required for a class Sparse including the copy constructor, overloaded assignment operator and destructor

- construct the zero polynomial
- construct a polynomial given the passed coefficient and degree of the one term
- read a polynomial (some of the code is provided)
- determine if the polynomial is the zero polynomial
- output the polynomial (just about all the code has been provided)
- add two polynomials
- subtract two polynomials
- multiply a polynomial by a scalar constant
- get the coefficient of a term given the degree
- get the degree of the polynomial
  the zero polynomial is a special case and has a degree of -1

To input the polynomial 2 x^0 - 3 x^1 + 4 x^3 we suggest you input first the number of terms, then the coefficients and degrees like this (ignoring new lines, you do not need to enter in separate lines)

```
3
2  0
-3  1
4  3
```

The output would look like this `2 - 3*x^1 + 4*x^3`

To add two polynomials, since the terms are ordered in ascending degree, you need to add using a **merge**. And similarly for subtraction (which you can do using addition and multiplication by -1 so you do not need to re-write merge again).

The operator + and – have been declared as friend operator functions and multiplication is a member operator function. In all cases (and in derivative), a new Sparse object is returned that is independent of the arguments and calling objects.

Think about your representation of the zero polynomial. I suggest that you use the nullptr for the zero polynomial, although you don't have to.

Delete any terms which have zero coefficients after adding. We only store the non-zero terms in the polynomial. Think of multiplication by 0.

Do not just "patch" your output function to avoid printing terms with a zero coefficient. Do not store terms with a zero coefficient.
The zero polynomial is a special case because it is the coefficient zero..
Return memory that is not used anymore to the freestore (to the heap).

RUNNING TIME COMPLEXITY

For your testing, we have provided a program called test_poly and a file cmds.txt has been provided.
Compile and link with

```
make test
```

See the attached user manual.
And a unittest file has been started. It is not comprehensive. Compile and link with

```
make
```

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE

- poly.h **add as much code as you want in the private part of the class**
- poly.cpp **add your code and document it**
- myOutput.txt **YOUR results of cmds.txt**
- unittest_poly.cpp **add tests as a BONUS and indicate this in the README.txt**
- README.txt **telling us if you did the BONUSes differentiation and more tests**

Files to include in the zip file (that you will not have modified)
doctest.h, cmds.txt, Makefile, test_poly.cpp, poly_utils.h, poly_utils.cpp.
Do not include any executables or object files.

BONUS:

Implement differentiation