Computer Science 2

Spring 2018 – Dr. Gurka Project #7: Polynomial

February 19 / 20 Specifications

**Description**

Create a polynomial class with linked lists and implement some basic functions.

# Specifications

1. Polynomials are stored in a linked list of Term objects (two classes: Polynomial and Term). Term objects have two data fields: coefficient and exponent, both positive integers. Polynomial will have some methods that create linked list functionality, but not all list operations are needed; include only those used by this program. Do not use a separate linked list class.
2. The list will be non-circular. Make a decision on whether to use a doubly-linked list and/or a tail reference. Explain your decisions in the cover letter; why are your choices good for the polynomial project?
3. Basic polynomial functions: read a polynomial from the user, read a polynomial from a file, add two polynomials (output to screen and file), evaluate a polynomial (output to screen and file). Evaluate takes a value for x (the variable), then calculates the polynomial’s value.
4. Input / output. The program will work interactively (good for testing during development), and with files. All input (user and file) is guaranteed to be correct.
5. Input files. There will be two final input files, with the same format. The first is test data from you, and will be four add operations and four evaluate operations, in any order. This file is created from a test plan with eight cases. For each case, first, describe what you are testing (examples: adding with missing terms, adding with no common terms, evaluating with zero); second, give the specific data being tested; third, give the expected result. A test plan template will be posted on Moodle. The second file will be from me (posted on Moodle), with an unknown number of add and evaluate operations. I’ll do additional testing on submitted projects.
6. Each operation takes three lines in the file:
   1. Add has the word “add” (line #1), then the two polynomials in normal form (one per line).
   2. Evaluate has the word “evaluate” (line #1), followed by the polynomial (in normal form) to evaluate (line #2), followed by the x value (line #3).
7. Output log file. This file will record the operations you used for final testing (taken from the input file) and the results. It starts with a heading, which includes your name. Then, for each operation, it gives the operation (add or evaluate), the polynomial(s) used, the x value (for evaluate), and the result (new polynomial for the add operation, final value for evaluate). Put a blank line between each operation’s data. Be sure to check this file to validate your answers.
8. Methods in the polynomial class.
   1. add two polynomials: some terms may be missing, polynomials may be different lengths
   2. read polynomial from file: prompt user for filename; assumptions: the file exists and is readable, the polynomials are correctly formatted
   3. read polynomial from user, assumption: perfect user
   4. print polynomial – coefficients of 1 do not print, terms with a zero coefficient do not print
   5. evaluate polynomial – x values will be integers
9. Polynomial syntax, normal form.
   1. polynomials are ordered from high-order term to low-order term
   2. coefficients of one are not displayed

x^2 not 1x^2

* 1. terms with zero coefficients are not stored or displayed

0x^3

* 1. superscripts – use carets (^)
  2. white space – exactly one space between terms and plus signs (input and output), no spaces in terms
     1. example: 25x^3 + 6x + 2

1. Further specifications.
   1. main is in a separate driver class
   2. x will be the only variable
   3. no multiple terms of the same order (e.g., two x^2 terms)
   4. no subtraction in the polynomial
2. Development. Use incremental development: first create a design for program structure, then create a stubbed form of the program. Continue by creating and testing one operation at a time. Comment out the stubs once the methods are implemented.
3. Collaboration. Since we have already gone over parts of this in class, do not do any additional joint work with classmates, and do not solicit solutions from anyone. Do not take code from online sources. Turning in work that is not your own will result in a grade of zero for the project.
4. Plan to allow time for getting my help as needed.
5. User testing extra credit. Have a classmate run your program and comment on the user interface, especially usability: was it easy to understand the input required? were messages clear? was the output clearly labeled? etc. Allow yourself time after the testing to improve your program. Discuss the testing session in your cover letter. This discussion should be detailed, not just a couple sentences saying it went fine and he or she liked it.
6. Swapped test plans extra credit. Exchange test plans with a classmate to further test your program. Hand in their test file and your output file.
7. Other extra credit. If you finish early and your program is well tested, you can see me to choose (more) extra credit.
8. Reminder: no extra credit on late assignments; don't start extra credit until the project is finished and tested.
9. Reminder: to be counted on time, projects must posted on Moodle before class. No drafts accepted.
10. Post on Moodle: source code (.java file(s)), your input data file, output from your input file, output from my input file, cover letter with additional discussions, any additional extra credit. Interactive work was for development, so no output for that is handed in. Name all files clearly. As usual, do not hand in any incomplete or incorrect programs (on the on-time date), and check the specifications carefully.

**Due Dates**

* test plan and test file (see #5): Wed., Feb. 28 / Thurs., March 1, 2:00
* final project: Mon., March 5 / Tues., March 6, 2:00