CIS*3190 Software for Legacy Systems

Assignment 2 - Ada Programming (Polynomial Arithmetic)

Maneesh Wijewardhana (1125828)

- NOTE: Throughout the menu and nested user options, the program accounts for bad user input
- Main Flow
  - ◇ Menu is printed which consists of the following operations:
    - ∗ Input a Polynomial
      1. readPOLY() procedure is called with two null head nodes
      2. If there already exists a polynomial, the program will ask the user to overwrite one by selecting the index (1..2)
      3. A series of Put() and Get() calls occur to get user input and the combinations of them get added as a polyNodePtr (poIr, coeff, next)
      4. The program will make sure negative exponents are not added and if the same exponent is added twice, they will simply be added together to avoid any confusion
      5. 0 coefficients are simply not added to the linkedlist of polyNodePtrs based on the assignment description
      6. This loops until the user enters 999 in the exponent option screen after which the polynomial is outputted using writePOLY()
      7. Before writing to the screen, sortPOLY() is called which uses a modified bubble sort algorithm to sort the polynomials in decreasing order
    - ∗ Output a Polynomial
      1. This option calls a modified writePOLY() procedure called writePOLYmain() where the only difference is an index is specified to print a polynomial (At any other point where a polynomial is being printed, writePOLY() is being used)
      2. I then separate each node in the polynomial by a "+" if our count is not 0 and the coefficient is more than 0
      3. Then I do a series of if statements to make sure 1 and -1 coefficients print as just "x" and "-x" respectively and ensure a 0 exponent just results in the coefficient being printed
    - ∗ Add Polynomials
      1. I loop through both polynomials and use my addPolyNode() procedure to add up coefficients that have the same power, Otherwise I input a new node with the same value
      2. After this loop, I loop once again separately to make sure I add any remaining terms from both of the linked list into our new one
      3. sortPOLY() followed by writePOLY() is called on the newly created linkedlist
    - ∗ Subtract Polynomials
      1. I make sure that if the two polynomials are the same, subtracting will just print out "0"
      2. Otherwise, I ask the user the order of subtraction they want (a-b or b-a) after which, I convert the all the nodes in the second polynomial to negative
      3. Then I call addpoly() to add the two polynomials and make sure to revert the negative polynomial back to its original
      4. sortPOLY() followed by writePOLY() is called on the newly created linkedlist
    - ∗ Multiply Polynomials
      1. I loop through both lists and multiply one node by all the other nodes in the other list
      2. The true flag in addPolyNode() is used to indicate that we want to make sure like terms are added up afterwards
      3. sortPOLY() followed by writePOLY() is called on the newly created linkedlist
    - ∗ Evaluate Polynomial
      1. I prompt the user to choose the polynomial they would like to solve and the "x" value to solve for
      2. I loop through the linked list and keep track of a result variable which I keep adding onto after calculating the new values
- Ada seemed mediocre at best to solve this problem due to the verbose nature of user defined data types such as linked lists and arrays. This resemblance comes naturally from the Java programming language where certain structures have a difficult time accessing one another if not defined correctly. Overall though, after jumping through the hurdle of defining my own linkedlist structure, the logic to solve the problem was trivial. I was able to use knowledge learned in languages such as C to write the functions properly.
- The ability to define a header file (.ads) for each of the .adb files made Ada a good choice for this problem. Having a lot of functions that do different things and take in different parameters makes it messy when inputting into one single file. However, being able to recall back to the C language and have a header file with structure definitions, function signatures, etc, made the code way cleaner and easier to understand.