COMP 1631 Assignment 5 ("Polynomial Playtime")

Given: Tuesday, November 17, 2020
Date Due: Monday, November 30, 2020 before 11:59 pm sharp

Objectives

- to practice using 1-D arrays
- to gain experience using basic array operations
- to practice using repetition in conjunction with arrays
- to practice designing and writing a variety of functions, including several which receive both value and reference arguments
- to learn how to write a menu-driven program
- to continue to develop problem solving skills and, in particular, to become more proficient at breaking a large problem down into smaller sub-problems
- to practice incremental program design and testing

The Problem

Many mathematical functions are expressed as polynomials in one unknown. Solutions for a variety of everyday scientific problems frequently require the manipulation of such polynomials. For example, velocity and acceleration problems from Physics, questions involving pressure, friction, temperature, etc. from Aeronautics and problems involving structural strength from Engineering.

Your job is to write a menu–driven program that will allow the user to perform simple operations on polynomials in one unknown.

A menu-driven program is one in which the user is presented with a list of options from which to choose. For example:

```
====================================

Would you like to

d) delete all of your files
s) stop the CPU
a) initiate auto-destruct
x) exit
Enter choice ==>
```

In this assignment, you will write a menu-driven program which will perform any of the following operations, according to the user's choice:

```
e – evaluate a polynomial
d – differentiate a polynomial
a – add two polynomials
m – multiply two polynomials
r – find the root(s) of a quadratic polynomial
q - quit
```

Once the results of the operation have been displayed, the menu should be redisplayed and the user prompted for another choice. This process will continue until the user chooses to exit (which must, of course, be an option on the menu).

Each menu option should call a function – called a command handler - to handle the processing of that operation. The command handler should interact with the user to obtain all input and then call another function that actually performs the desired operation on the specified polynomials and then output the computed answer.

This assignment will require the design and coding of a number of functions whose specs are detailed below. Use a divide-and-conquer approach – don't try to work on all components of the program at once! Design and code the program one function at a time, testing that your functions are working as you go.

Polynomial Structure

A polynomial is the sum of terms of the form $a\mathbf{x}^n$ where *a* is called the coefficient and *n* is called the exponent. There can be at most one term with a particular exponent.

e.g. $3\mathbf{x}^5 + 2$ is a legal polynomial
$3\mathbf{x}^2 + \mathbf{x}^2$ is not a legal polynomial (has 2 terms with the same exponent x^2)
$\mathbf{x} + 3 + 1$ is not a legal polynomial (has 2 terms with the same exponent x^0)

While the maximum number of terms in a polynomial is logically unbounded, for this assignment, a polynomial will be stored in an array and it can be assumed that the maximum exponent in a polynomial will be 20. Terms in a polynomial have to be maintained in order based on exponent. This is easily accomplished by storing the coefficient of a term in the array element whose index corresponds to the exponent for the term.

e.g. Suppose P was an array holding coefficients for the polynomial $3\mathbf{x}^5 + 2$
Then P[0] would be 2 since $2 = 2\mathbf{x}^0$
and P[5] would be 3
All other entries in P would be 0

You may assume integer coefficients and positive integer exponents. Notice that neither the exponent nor the symbol for the unknown (in this case, **x**) have to be stored – both are implicit in the representation.

In summary then, a ***polynomial*** is to be represented as an ***array of integers*** in which the coefficients of terms are the array contents and the exponents are the array indices. Since **the largest exponent which can be accommodated will have a value of 20, the arrays used in the program must have 21 elements.**

This is to be implemented using the following typedef:

```
typedef int Poly[MAX_SIZE];
```

where MAX_SIZE is a constant with value 21

Supporting Functions

In order to implement functions for each of the menu operations, a number of supporting functions will be needed:

```
void  readPoly ( Poly p, bool& success )
```

This function loads a polynomial by reading data from the user. The input will consist of an unknown number of lines, each of the form

coefficient exponent

where both items are integers and valid exponents are in the range `0` to `20` inclusive. If an out−of−bounds exponent is encountered, the boolean success flag should be set to `false` and reading should be terminated. The end of input, and therefore the end of the polynomial, will be detected when the user types CTRL−d. This will result in the end of file being set for cin.
This function should not do any printing.

For example, the polynomial $4\mathbf{x}^5 + 2\mathbf{x}^4 + 100\mathbf{x}^2 + 14$ could be read in as

```
14 0
100 2
2 4
4 5
[CTRL−d]
```

It would also be possible for a user to enter this polynomial as

```
2 4
100 2
4 5
14 0
[CTRL−d]
```

After entering CTRL-d the input stream is in a fail state and all subsequent reads will fail. How to return to a good state will be discussed in COMP 1633 – for this assignment you are provided a `reset()` function. This function is in a library called `reset.o` with a corresponding `reset.h` header file. If you have forgotten how to use a user-defined library review the assignment 3 writeup. The `reset` function should be called once only after you know the user entered CTRL-d.

```
void printPoly ( const Poly p )
```

This function receives an initialized polynomial and prints it in standard form. The preceding polynomial would be printed as

```
4x^5 + 2x^4 + 100x^2 + 14
```

```
void clearPoly ( Poly p )
```

This function receives a polynomial and sets every coefficient to 0 (in essences, returning an empty polynomial).

Polynomial Operations

These functions do the specified polynomial operation – **they DO NOT interact with the user at all, i.e. they do not perform input nor do they perform output**. There are many polynomial operations but five fairly standard operations have been selected for implementation:

```
void evaluate (const Poly p, int x, int& pValue )
```

This function receives an initialized polynomial and a substitution value for the unknown. It should calculate and return the integer value of the polynomial when x is set to the substitution value.

For example, with a substitution value of 2 for x, the preceding polynomial would evaluate to

```
4(2)^5 + 2(2)^4 + 100(2)^2 + 14 = 574
```

```
void sumPolys (const Poly p1, const Poly p2, Poly sum)
```

This function receives two initialized polynomials as input and constructs a third polynomial representing the sum. The sum of two polynomials is determined by adding coefficients for terms with equal exponents.

For example: if P1 is $3\mathbf{x}^3 + \mathbf{x}^2 + 2$
and P2 is $4\mathbf{x}^2 + 8\mathbf{x} + 7$

the sum is $3\mathbf{x}^3 + 5\mathbf{x}^2 + 8\mathbf{x} + 9$

```
void multiplyPolys (const Poly p1, const Poly p2, Poly product,
                    bool& productFound)
```

This function receives two initialized polynomials as input and constructs a third polynomial representing the product. The product of two polynomials is determined by multiplying all possible pairs of terms.

For example: if P1 is $3\mathbf{x}^3 + \mathbf{x}^2 + 2$
and P2 is $4\mathbf{x}^2 + 8\mathbf{x} + 7$

the product is $12\mathbf{x}^5 + 28\mathbf{x}^4 + 29\mathbf{x}^3 + 15\mathbf{x}^2 + 16\mathbf{x} + 14$

If the product polynomial will not fit in a standard polynomial array (i.e. the maximum exponent would be greater than 20), then no multiplication should be performed and the boolean success flag should be set to false

```
void differentiate (const Poly p, Poly d )
```

This function receives an initialized polynomial and ***constructs*** a second polynomial by mathematically differentiating the input polynomial.

For example, if the input polynomial is $4x^5 + 2x^4 + 100x^2 + 14$

the resulting polynomial is $20x^4 + 8x^3 + 200x$

In general, when a term with coefficient *a* and exponent *n* is differentiated, the new term has coefficient *(a* n)* and exponent *n-1*.

```
void rootsPoly (const Poly p, int& numberOfRealRoots, double& root1,
                double& root2, bool& rootsFound)
```

This function receives an initialized polynomial as input. If the polynomial is not linear or quadratic, no roots can be computed (the Boolean success flag should be set to false). Otherwise, the polynomial may have real roots determined as follows:

A linear polynomial (in the variable x) has the form $ax + b$ where a and b are integers with $a \neq 0$.

To find the roots, we must solve the equation formed by setting the polynomial to zero.

i.e. $ax + b = 0$

Such an equation has only one solution given by $x = \frac{-b}{a}$

e.g. $2x - 3 = 0$ is linear with $a = 2$ and $b = -3$ The solution is $x = \frac{3}{2}$

In the event of one root (as we have here), this root (`1.5`) is stored in `root1` and `root2` is undefined.

A quadratic polynomial (in ***x***) has the form $ax^2 + bx + c$

where a, b, and c are the given coefficients, with $a \neq 0$. To find the roots, we set the polynomial equal to 0.
i.e. $ax2 + bx + c = 0$

An equation of this type has either zero, one, or two (real) solutions, depending on the value of its discriminant $d = b^2 - 4ac$

- if $d > 0$, the equation has 2 distinct solutions, given by

$$x1 = \frac{-b + \sqrt{d}}{2a} \quad \text{and} \quad x2 = \frac{-b - \sqrt{d}}{2a}$$

- if d = 0, the equation has one solution given by $x = \frac{-b}{2a}$
- if d < 0, the equation has no real number solutions

e.g. $x^2 - 3x + 2 = 0$ is quadratic, with a = 1, b = −3, and c = 2

Then $d = (-3)^2 - 4(1)(2) = 9 - 8 = 1 > 0$ so there are 2 roots and the solution is

x1 = 2 x2 = 1

Error Checking

Your program must detect and handle the following errors:

1) The user enters an illegal menu choice — print a message, re–display the menu and get another choice.

2) A polynomial being read in is illegal (an exponent is out of bounds or is duplicated) − report exponent error, terminate the operation, re–display the menu and get another choice.

3) A polynomial being read in is empty (no terms are entered) − report error, re–display the menu and get another choice.

4) The product of multiplication has an exponent out of range – the **`multiplyPolys`** function will set the "product not found flag", but the calling function will output an error message and the program will allow the user to perform the next operation.

5) The polynomial whose roots have been requested is neither linear nor quadratic – the `findRoots` function will set the "roots found flag" to false, but the calling function will output an error message and the program will allow the user to perform the next operation.

Detailed Specifications

Design and implement a modular program which solves the problem as follows:

1) Design and code **the three supporting functions**, starting with the `printPoly` function. Since it takes an array of integers – of any size – you can test it on a smaller sized array that is declared and initialized by the calling function. Then implement the `clearPoly` function as it is simple and straightforward.

   Finally implement the `readPoly` function to read a polynomial from the keyboard. This function should call the `clearPoly` function to clear the array into which the polynomial will be read. Then the polynomial should be read in, term by term, and the coefficient stored in the array according to the exponent. The function should return the polynomial array and a boolean flag indicating whether or not a valid "*read*" was performed. Re-read the previous description of this function so you

completely understand the termination of input and resetting of the input stream. Since the printPoly function exists it can be used to verify that your input routine is working.

2) The main program should be a simple controlling module which displays the menu, gets the user choice and calls the appropriate command handler function to process the request. Here is a high-level algorithm:

```
print welcome message
do
      show menu
      get choice
      call appropriate function to process choice

while choice NEQ exit
```

3) For each of the menu choices, design a command handler function to control the processing for that operation. Initially all of them should be stubs. Then select one and implement it. For example, the **process_addition** command handler function could handle one addition operation in the following way:

```
prompt for a polynomial
call readPoly to read P1
if read was invalid
      print error message
else
      prompt for another polynomial
      call readPoly to read P2
      if read was invalid
            print error message
      else
            call sumPolys to perform the addition
            call printPoly to print P1
            print operator
            call printPoly to print P2
            print line
            call printPoly to print result
return to main
```

Notice that error messages are output by the command handler functions. As well, other than outputting a message there is no attempt to deal with any error. In the event of an error the operation is terminated and processing moves onto the next operation.

4) Now design and implement the corresponding polynomial operation function. In this case `sumPoly`, which takes two input polynomials and returns a third polynomial which is the sum of the two inputs.

5) Repeat steps 3) and 4) for each pair of command handler and polynomial function.

Debugging Hints

If your algorithm is designed properly this shouldn't be necessary. However, if you do have problems with the logic, first trace through your design with the data for which incorrect results are being produced. If you cannot locate the error in this way, inserting temporary **cout** statements (i.e. trace statements) to write out intermediate results in the calculation may help.

Errors in logic can be of two types:

- A major design error or omission. In this case you should go back to your pseudocode and re-design. Fiddling with the program is no good, because even if you get it running right, you probably won't understand its logical structure. This understanding has to come at the pseudocode level, which is not cluttered up with program level details. Remember that this takes practice.

- A fairly minor error. In this case the solution is to make a minor change, not a major one which may just introduce more errors.

Getting Help

You may have difficulty in determining which type of error you have. This takes experience. You should make a good attempt at solving the problem, but don't spend too much time on it. Instead, ask an IA or instructor and we'll point you in the right direction.

Depending on the type of problem, you may be asked to produce pseudocode, and a trace of the algorithm before help is given. This is to ensure that you are in the habit of properly designing your program. Chances are that if you follow these guidelines, you won't need much help anyway!

Restrictions

You may not use any constructs that we have not yet studied in class.

Hand In

Official submission instructions will be supplied a few days before the assignment is due. You will have to design and test your program well before that day so you need to think about data values which will give your program a proper workout and design several sets of input yourself.