**Assignment 3 CE4703: Technical Report**

**Group: 04**

Mike Vriesema 17212359

Luke O'Sullivan Griffin 17184614

Ryan O'Connor 17209382

Conall McAteer 18173586

Gearoid Kirwan 17213266

***Task: A text document containing an explanation of your designed data type, your specification and your pseudo-code for all operations.***

**Explanation of data type:**

Our data type is called Polynomial.

* Polynomial is a struct that is then typedef to polynomial. It contains a polyNode pointer called head (start of polynomial) and a polyNode pointer called current that keeps track of current node.
* PolyNode is a struct n typedef to polyNode that contains a term called d and a struct n pointer called successor which points to the next node in the sequence.
* Term is a struct that contains a double called coefficient and an int called exponent.

**Your specification:**

The polynomials must be stored in a struct.

The structs will be stored in a dynamic data storage type eg VLA,Linked List etc.

Functions must be created to:

* Create a polynomial,
* Delete a polynomial,
* Add 2 polynomials,
* Subtract 2 polynomials,
* Multiply a polynomial by a double,
* Divide a polynomial by a double,
* Normalize a polynomial,
* Order a polynomial,
* Display a polynomial in standard output.
* Add and subtract polynomial must take in pointers to two polynomials and return a pointer to a polynomial where the new polynomial is the result of adding or subtracting the two polynomials respectively.
* Multiply and divide polynomial should take in a pointer to a polynomial and a double value entered by the user and return a pointer to the same polynomial where the new polynomial contains the result of the original polynomial multiplied and divided by the given double value respectively.
* Normalize poly function will take in a pointer to a polynomial, normalize it by adjusting the coefficient of the highest exponent to 1 and return a pointer to that same polynomial.
* Order polynomial must take in a pointer to a polynomial and print the order of the polynomial by getting the highest exponent.
* Display polynomial will take in a pointer to a polynomial and print it to the terminal in the standard form.
* These functions must be declared in a header file and the header file included in the C file.
* The main file will include a driver menu where these functions will be showcased using hardcoded values.

Assignment of roles:

Luke: Divide and Multiply

Gearoid: Display and Order

Conall: Normalize and Delete

Ryan: Add and Subtract

Mike: Header File,Create, Insert After, GoToNext and GoToHead

**Pseudo-code for all operations:**

**Pseudo-code for polynomial.h:**

Create include guards to avoid the problem of double inclusion

Create an enum called polyError

contains error codes for the polynomial, It contains ok, illegalNode, and noMemory

Create a struct called term

contains a double and an int called coefficient and exponent

Create a struct for a node called polyNode

contains a term and a struct with pointer to n called d and successor node

Create a struct called polynomial

contains a polyNode pointer called head and a polyNode pointer called current

Declare all functions

Close the include guard

**Pseudo-code for the polynomial.c:**

Open function called createPoly that takes in an exponent and returns a polynomial pointer

declare int called exponent

print asking for degree of polynomial

scan for the int input

declare polynomial pointer called list

list equal allocate memory to list for size of polynomial

if list not null

head of list equal to allocated memory slot at polyNode pointer for polyNode

if the head of list is not null

point the head to the tail and point the current to the head

for int I equal exponent I less than equal to zero decrement i

call insertAfter passing in list and i

else

call free function and pass in list

list equal null

return list

open deletePoly function passing in polynomial pointer called poly

declare polynode pointer called next

if poly is not null

current pointer equal to head

while successor of head pointer is not null

next equal successor of head

head successor equal head successor successor

call free function passing in next

call free function passing in head pointer

call free function passing in poly

print that polynomial is deleted

else

print that polynomial must be created first

open insertAfter function passing in polynomial pointer poly and int exponent and double coefficient

declare a polyerror called returnvalue and set it to ok

declare a polyNode pointer called newnode

declare a term called newdata

set the newdata exponent equal to the exponent

allocate memory for new node

if newnode is null

returvalue equal noMemory

else

newnode pointer d equals newdata

newnode successor equals null

if poly head pointer equals null

poly head pointer equals newnode

poly current pointer equals newnode

else

polyNode pointer temp equals poly head

while temp successor not null

temp equals temp successor

temp successor equals newnode

return returnvalue

open function gotoHead passing in a polynomial pointer called poly

set current to head

open function gotoNextNode passing in polynomial pointer poly

polyError result equal to ok

if current successor is not null

current equal to successor

else

result equals illegalNode

return polyError result

open function accessdata passing in polynomial pointer called poly

if current not equal to the head and the current is not null

return address of current d pointer

else

return null

open function addPoly and pass in two polynomial pointers called p1 and p2

if p1 and p2 are not equal to null

call function gotoHead for p1

call function gotoHead for p2

print that they are being added

while gotoNextNode function call for p1 equals ok and same for p2

term pointer a equals result of function accessData passing in p1

term pointer b equals result of function accessData passing in p2

declare double aCoeff and equal it to coefficient from a

declare double bCoeff and equal it to coefficient from b

declare double answer equal it to the addition of aCoeff and bCoeff

equal coefficient in a equal to answer

call function displayPoly and pass in p1

else

print that no polynomials were found

return polynomial pointer p1

open function subPoly and pass in two polynomial pointers called p1 and p2

if p1 and p2 are not equal to null

call function gotoHead for p1

call function gotoHead for p2

print that they are being subtracted

while gotoNextNode function call for p1 equals ok and same for p2

term pointer a equals result of function accessData passing in p1

term pointer b equals result of function accessData passing in p2

declare double aCoeff and equal it to coefficient from a

declare double bCoeff and equal it to coefficient from b

declare double answer equal it to the subtraction of aCoeff and bCoeff

equal coefficient in a equal to answer

call function displayPoly and pass in p1

else

print that no polynomials were found

return polynomial pointer p1

open function called multiplyPoly and pass in a polynomial pointer p and a double called value

call function gotoHead and ass in p

if p is not null

print that its being multiplied by variable value

while gotoNextNode function passing in p is equal to ok

declare a term pointer called polyTerm and set it to result of accessData passing in p

declare double called multiplyResult equal it to the coefficient from polyTerm

equal multiplyResult to multiplyResult times value

let coefficient from polyterm equal multiplyResult

print that its displaying new polynomial

call function displayPoly and pass in p

else

print create a polynomial

return polynomial pointer p

open function dividePoly and pass in polynomial pointer p and double value

if p is not null

print its dividing by the value

while gotoNextNode returns ok when p is passed in

declare a term pointer called polyTerm and equal it to the result of function accessData when p passed in

declare double divideResult and equal it to coefficient from polyTerm

divideResult equal divideResult divided by value

coefficient from polyterm equal to divideResult

print New polynomial

call function displayPoly passing in p

else

print create a polynomial

return the polynomial pointer p

open function normalizePoly passing in polynomial pointer p

call function gotoHead passing in p

if p is not null

if gotoNextNode passing in p is equal to ok

term pointer polyTerm equal accessData function result passing in p

declare double coeff equal it to coefficient in polyTerm

call function dividePoly and pass in p and coeff

else

print create a polynomial

return polynomial pointer p

open function called orderPoly and pass in polynomial pointer p

if p is not null

call function gotoHead and pass in p

if gotoHead for p returns ok

term pointer d equal to result of accessData function passing in p

print order of polynomial and then the exponent in d

else

print that you must create a polynomial first

open displayPoly function passing in polynomial pointer myPoly

if myPoly is not null

call function gotoHead and pass in myPoly

if mylist head success is equal to null

print that no polynomial found

else

do

if current in myPoly equal head in myPoly

print polynomial equals

else

term pointer d equal result of accessData when myPoly is passed in

if coefficient in d is greater than zero

if exponent in d is equal to one

print the coefficient of d

else if exponent in d is zero

print the coefficient of d

else

print coefficient and exponent of d

else

if exponent in d equal 1

print the coefficient of d

else if exponent of d equals zero

print the coefficient of d

else

print the coefficient and exponent of d

while gotoNextNode is equal ok when myPoly is passed in

print two new lines

else

print that you must create a polynomial first

**Pseudo-code for mainProgram.c:**

Open function main

Create two polynomial pointers named “poly1” and “poly2” and set both values to null.

Print to screen that the program is creating new polynomials and to input the values for the first and second polynomial.

Send the hardcoded exponent/order values into createPoly.

Set “poly1” and “poly2” equal to the return of the method createPoly after each time it’s called.

Display “poly1” and “poly2” that were just created on the screen using the displayPoly function and passing in the “poly1” and “poly2” variable names that was just created.

Print to the screen that the program is Adding the polynomials.

Call the addPoly function and pass in both “poly2” and “poly1”

Print to the screen that the program is Subtracting the polynomials.

Call the subPoly function and pass in both “poly1” and “poly2”

Print to the screen that the program is Multiplying the polynomial.

Call the multiplyPoly function and pass in “poly1” and a hardcoded value.

Print to the screen that the program is Dividing the polynomial.

Call the dividePoly function and pass in “poly2” and a hardcoded value.

Print to the screen that the program is normalizing the polynomial.

Print to the screen that the program is returning the order of polynomial

Call the orderPoly function and pass in the “poly1” variable.

Print to the screen that the program is deleting polynomial.

Call the deletePoly function and pass in the “poly1” variable.

Print to screen that the program is displaying the polynomial

Call the displayPoly function and pass in the “poly1” variable.

Print to the screen that we are exiting the program

Return exit success to indicate successful program execution status.

**Pseudo-code for the makefile:**

List all the targets that need to be executed in the makefile inside the exec

Set variables

one called poly used for polynomial

one called main used for mainProgram

Set the variable name all to the elements of the exec

Compile the polynomial.c and polynomial.h files into polynomial.o

Compile the mainProgram.c file into mainProgram.o

Link the components

Run the application

Clean

Remove files ending in .o and the project executable