Tema: N°
Fecha: / /

Luis Jhon Pancca Choque

TEMA III

1. DESCRIPCIÓN DEL TDA POLINOMIO

El prefijo "Poli" significa muchos, Polinomio significa entonces muchos términos, Las expresiones polinomiales que contiene una, dos, tres o cuatro términos se conocen respectivamente como monomio, binomio, trinomio y cuatrinomio. Cabe aclarar que no se les dan nombres especiales a los polinomios de cinco términos en adelante

Polinomio es la suma finita de expresiones Ax^n (si es una variable) o de la forma $Ax^m y^n$ (si es de dos variables)

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_2 x^2 + a_1 x + \dots$$

Donde:

a es una constante,
m y n son los exponentes enteros no negativos
x, y son las variables

Notación: Usualmente se usa la siguiente notación:

P(x) : Polinomio de una variable,
P(x,y) : Polinomio de dos variable

Polinomios especiales

Polinomio ordenado: Respecto a una variable, Es aquel polinomio donde los exponentes de dicha variable están ordenados de menor a mayor o viceversa

Polinomio Completo: Respecto a una variable, es aquel donde dicha variable representa todos los exponentes desde el cero hasta el mayor

Polinomios Idénticos: Son dos polinomio del mismo grado, con la misma variable, coeficientes iguales (Términos iguales)

Polinomio Opuesto: Son aquellos que tienen los mismo términos del mismo grado con signos contrarios, es decir que la suma de dos polinomios opuestos es igual al polinomio nulo

En esta unidad vamos a construir el TDA Polinomio, comenzando por la identificación de sus operaciones y realizando tanto la especificación de su funcionamiento como la implementación

N°: Tema:
Fecha: / /

Luchiano Péncora Choque

En primer lugar debemos identificar el conjunto de operaciones que definiremos para ello, deberemos considerar como será usar el nuevo tipo. En nuestro caso, podemos pensar, por ejemplo, en:

- Un método para asignar un valor a una variable del nuevo tipo. La forma más simple de hacerlo es construir una función que asigne un valor al coeficiente de un determinado monomio. La asignación de un polinomio completo se puede realizar con llamadas sucesivas a esa función.
- Un método para obtener el valor real que corresponde a un determinado monomio
- Un método para obtener el valor que corresponde a un determinado monomio almacenado
- Un método para sumar dos polinomios
- Un método para restar dos polinomios
- etc.

3.1.2 Especificación del TDA Lista

Partiendo de lo establecido en la Unidad 1 Especificación Informal tenemos los siguientes: Elementos que conforman la estructura

TDA Polinomio (VALORES son coeficientes y grados del tipo Entero, OPERACIONES Crea, es_cero, grado, poner_termino, coeficiente, sumar, multiplicar, restar, numero_terminos, exponentes)

OPERACIONES

crea (P: Polinomio)
Utilidad: Sirve para inicializar el polinomio
Entrada: Polinomio P
Salida: Ninguno
Poscondición: Polinomio inicializado sin términos

Es_Cero (P: Polinomio) devuelve (booleano)
Utilidad: Devuelve verdadero si P es un polinomio sin términos
Entrada: Polinomio P
Salida: Booleano
Precondición: Ninguna
Poscondición: Ninguna

Tema: _____ N°: _____

Fecha: ____/____/____

Luis Mixto Poncede Chagua

Grado (P: Polinomio) devuelve (Grado del Polinomio)
Utilidad: Devuelve valor que indica grado polinomio
Entrada: Polinomio P
Salida: Número
Precondición: El polinomio es no cero
Poscondición: Ninguna

coeficiente (P: Polinomio, Exp: Entero) devuelve (coeficiente de Termino)
Utilidad: Devuelve el coeficiente que corresponde al termino
Entrada: Polinomio P
Salida: Número
Precondición: Polinomio no Cero
Poscondición: Ninguna

AsignarCoeficiente (P: Polinomio, exp: Entero)
Utilidad: modifica el valor del coeficiente del término que tiene el grado exp
Entrada: Polinomio P
Salida: Ninguna
Precondición: Polinomio no Cero
Poscondición: Polinomio no modificado en el valor coeficiente de termino con grado exp.

Poner_termino (P: Polinomio; coef, exp: entero)
Utilidad: Modifica Polinomio P asignado el termino con coeficiente (coef) y exponente (exp)
Entrada: Polinomio P
Salida: Ninguna
precondición: Coeficiente y grado de tipo entero.
Poscondición: Polinomio modificado en el valor del coeficiente de termino con grado exp.

numero_terminos (P: Polinomio) devuelve (Num. Terminos)
Utilidad: Determinar el número de terminos que tiene el polinomio
Entrada: Polinomio P
Salida: Número
Precondición: Ninguna
Poscondición: Ninguna

Exponente (P: Polinomio; nroter: entero) devuelve (Grado)
Utilidad: Retorna el grado del termino ubicado en el lugar (nroter)
Entrada: Polinomio P
Salida: Número
Precondición: Que el nroter exista
Poscondición: Ninguna

Nº _____ Tema: _____

Fecha: ___/___/___

Evaluación Parcial Clase

Sumar (P1, P2 : Polinomio ; ES P: Polinomio)
Utilidad: Realiza suma P1+P2 y almacena en el polinomio P
Entrada: Polinomio P1, P2
Salida: Ninguna
Precondición: Ninguna
Poscondición: Polinomio P tiene la suma de P1 y P2

restar (P1, P2 : Polinomio ; ES P: Polinomio)
Utilidad: Realiza la resta P1-P2 y la almacena en el polinomio P
Entrada: Polinomio P1, P2
Salida: Ninguna
Precondición: Ninguna
Poscondición: Polinomio P tiene la Resta de P1 y P2

Multiplicar (P1, P2: Polinomio; ES P: Polinomio)
Utilidad: Realiza la multiplicación P1*P2 y la almacena en el polinomio P
Entrada: Polinomio P1, P2
Salida: Ninguna
Precondición: Ninguna
Poscondición: Polinomio P tiene la multiplicación de P1 y P2

1.13 Aplicación con Polinomio

En esta sección se puede apreciar que ya estamos en condiciones para plantear algoritmos usando el TDA Polinomio, abstrayéndonos de la forma como este implementado

EJE1 Implementar un algoritmo que busque un elemento en la Lista L y retorne la Dirección donde se encuentra caso contrario Nulo

EJE1 Implementar un algoritmo que busque un elemento Polinomio P asigne su derivada a P1

Derivada (Polinomio P, ES Polinomio P1)
Inicio
  Para cada i=1 hasta P.numero_terminos()
  Inicio
    ex = P.exponente(i)
    co = P.coeficiente(ex)
    P1.poner_terminos(co*ex, ex-1)
  Fin
Fin

$P(x)=5x^3+3x^2+2x^0$

Derivada

$P1(x)=15x^2+3x^0$

Tema: Lucháro Pericena Choque

N°

Fecha: / /

Ej 2. Dado un polinomio P elabore un algoritmo que muestre la integral de dicho polinomio

$\int(15x^2+3x^0)$

$\int 15x^2 + \int 3x^0$

$15\int x^2 + 3\int x^0$

$\frac{15x^3}{3} + \frac{(3x)}{1} + C$

$(5x^3) + (3x^1) + C$

```
Mostrar_Integral (Polinomio P)
  Inicio
    Para cada i = 1 hasta P.numero_terminos()
    Inicio
      ex = P.exponente(i)
      co = P.coeficiente(ex)
    mostrar("(", co, "x^", ex+1, ")/", (ex+1), "+")
    Fin
    mostrar "C"
  Fin
```

2.1 Implementación de clase polinomio

Para la implementación con Listas del TDA polinomio se utiliza un lista POL que contendrá los coeficientes y exponentes de cada termino, se hace notar que la cantidad de terminos estara dada por la longitud de la lista

Declarando la Polinomio

Tipos de datos

Clase polinomio

Atributos

Pol: Lista

Metodos

//(Pol: Lista)

Privado

Direccion BuscarExponente (Exp: Entero)

Direccion BuscarTermino (i: Entero)

Publico

Crea()

Booleano EsCero()

Entero Grado Grado()

Entero coeficiente (Exp: Entero)

Asignar Coeficiente (coef, exp: Entero)

Poner_Termino (coef, exp: Entero)

Sumar (P1, P2, Polinomio)

restar (P1, P2: Polinomio)

multiplicar (P1, P2: Polinomio)

Fin declaración de clases

Fecha: ___/___/___

Luis Alan Pérez Chávez

Dirección Polinomio, BuscarExponente(Exp: Entero)
// pol < coef, exp, coef, exp, coef, exp ... >

inicio
    Dir = Pol.siguiente(Pol.primero)
    Si dir <> nulo entonces
        dirExp = Nulo
        Mientras (dir <> nulo) y (DirExp = Nulo)
        si pol.recupera(dir) = exp entonces
            DirExp = Dir

        Dir = Pol.Siguiente (Pol.siguiente (dir))
    fin mientras
        retornar DirExp
    Caso contrario
    // excepción polinomio no tiene términos
fin

Dirección BuscarTermino N (I: Entero)
Inicio
    Dir = Pol.primero
    Nt = 0
    Si dir <> nulo entonces
        dirTer = Nulo
        Mientras (dir <> nulo) y (DirTer = Nulo)
        Nt = nt + 1
        Si nt = i entonces
        DirTer = Dir
    Dir = Pol.Siguiente (pol.siguiente(dir))
    Fin mientras
        Retornar DirTer
        Caso Contrario
        // Excepción polinomio no tiene términos
    fin

Polinomio Crear
inicio
    pol.crear // llamar al constructor de Pol
fin

Tema: _____ Nº _____

Fecha: ___/___/___

Lucrecia Pérez Chaya

```
Polinomio.EsCero ()
  Inicio
    retornar (Pol.longitud = 0)
  Fin

entero polinomio.grado()
 inicio
    Dir = Pol. Siguiente (Pol. primero)
    Si dir <> nulo entonces
        MaxG = Pol. recupera (dir)
        Mientras dir <> nulo
          Si Pol. recupera (dir) > maxG entonces
           MaxG = Pol. recupera (dir)
           Dir = Pol. siguiente (Pol. siguiente (dir))
    Fin mientras
        Retornar maxG
    Caso contrario
        // exception polinomio no tiene términos
    Fin

Entero polinomio. coeficiente (exp : entero)
 Inicio
    Dir = BuscarExponente (exp)
    Si dir <> nulo entonces
                Retornar Pol recupera (Pol. Anterior (dir))
        Caso contrario
        // exception polinomio no tiene ese término
    Fin

Polinomio. asignarCoeficiente (coef, exp : entero)
Inicio
    Dir = Buscar Exponente (exp)
    Si dir <> nulo entonces
        dir Coef = Pol. anterior (dir)
        Pol. modifica (dir Coef, coef)
        Si Coef = 0 entonces
                Pol. suprime (dir)
                Pol. Suprime (dir Coef)
        Caso contrario
                // Exception polinomio no tiene ese término
 Fin
```

N° ______ Tema: ______

Fecha: ___ / ___ / ___

Luis Lino Pencama Choque

```
booleano polinomio.poner_termino (coef, exp : entero)
inicio
    DirExp = BuscarExponente (exp)
    si DirExp <> Nulo entonces
        DirCoef = Pol.anterior (dirExp)
        Pol.modifica(dirCoef, Pol.recupera (DirCoef) + coef)
        si pol.recupera (dirCoef) = 0 entonces
            pol.suprime (dirExp)
            pol.suprime (dirCoef)
    Caso contrario
        si coef <> 0 entonces
            pol.insertUltimo (exp)
            pol.inserta (pol.fin, coef)
    fin

Entero polinomio.numero_terminos() devuelve (Nro Terminos)
inicio
    Retornar pol.longitud div 2
fin

Entero polinomio.exponente (nroter : entero) devuelve (grado)
inicio
    Dir = BuscarTerminoN(nroter)
    si dir <> Nulo entonces
        Retornar pol.recupera (siguiente (dir))
    Caso contrario
        // excepcion no existe ese numero de termino
fin

Publico polinomio.suma (p1, p2 : Polinomio)
Inicio
    // poner polinomio en 0
  Para cada i = 1 hasta P1.numero_terminos
    inicio
        ex = P1.exponente (i)
        co = P1.coeficiente (ex)
        poner_termino (co, ex)
    fin
Para cada i = 1 hasta P2.numero_terminos
    inicio
        ex = P2.exponente (i)
        co = P2.coeficiente (ex)
        poner_termino (co, ex)  fin
fin
```

Tema: Nº:
Fecha: / /

Luchita Peñaloza Choque

Polinomio.resta (P1, P2 : Polinomio)
Inicio
// poner polinomio en 0
para cada i = 1 hasta P1.numero_terminos
inicio
ex = P1.exponente(i)
co = P1.coeficiente(ex)
poner_termino(co, ex)
fin

Para cada i = 1 hasta P2.numero_terminos
inicio
ex = P2.exponente(i)
co = P2.coeficiente(ex) * -1
poner_termino(co, ex)
fin
Fin

Polinomio.multiplicación (P1, P2 : polinomio)
Inicio
// Desarrolle el algoritmo
fin

2.-

Para la implementación del TDA polinomio se utilizara dos vectores y un atributo denominado nt, donde los vectores serán los que contendran los coeficientes y exponente de cada termino, se hace notar que nt determinara el número de terminos que contiene el polinomio

Definido la Polinomio
constante max = 100
Tipos de datos
Clase Polinomio
Atributos
VC // Coeficiente
VE : Arreglo (MAX) // exponente
nt : Entero
Metodos
crea()
Es Cero () devuelve (booleano)
Grado () devuelve (Grado del Polinomio
Coeficiente (exp : Entero) devuelve (coeficiente de termino)
Asignar Coeficiente (coef, exp : Entero)

WINNER

N° _____ Tema: _____

Fecha: ___/___/___

Continuación Parcial Opaque

Poner_termino (coef, exp: entero)
numero_Terminos () devuelve (Nro. Terminos)
existente (nro: entero) devuelve (Nro Terminos)
Sumar (P1, P2: Polinomio)
restar (P1, P2: Polinomio)
multiplicar (P1, P2: Polinomio)
Fin definición clase

Constructor Polinomio_Crear
inicio
nt = 0
fin

Public Polinomio_Es Cero ()
inicio
retornar (nt > 0
fin

Entero Polinomio_grado()
inicio
si nt > 0 entonces
max = vc[1]
para cada i = 1 hasta nt
si vc[i] > max entonces max = vc[i]
retornar max;
caso contrario
// error no existe termino
fin

Polinomio_asignarcoeficiente (coef, exi entero
inicio
lug = // Existe exponente (exp) en la estructura recibida
vector vc
si lug <> -1 entonces
vc[lug] = coef
si vc[lug] = 0 entonces
// desplazar 1 elemento hacia la posición lug
nt = nt - 1
caso contrario
// excepción error no existe termino con ese exp
fin

Tema:

N°

Fecha: / /

Lunes 20 Primera Chaque

Polinomio poner_termino (coef, exp : entero)
Inicio
  lug = // Existe exponente (exp) en la estructura recorriendo vector ve
  Si lug <> -1 entonces
    ve[lug] = ve[lug] + coef
    Si ve[lug] = 0 entonces
      // desplazar 1 elemento hacia la posicion lug
      nt = nt - 1
  Caso contrario
    nt = nt + 1
    ve[nt] = coef
    ve[nt] = exp
Fin

Entero polinomio_numero_terminos() devuelve (Nro. Terminos)
Inicio
  Retornar nt
Fin

Entero polinomio_coeficiente (exp : entero)
Inicio
  Si exp >= 0 y exp <= grado() entonces
    para cada i = 1 hasta nt
    Inicio
      Si ve[i] = exp entonces retornar ve[i]
    Fin
  // error no existe terminos con ese exponente
Fin

Entero Polinomio_exponente (nrotor : entero) devuelve (grado)
Inicio
  Retornar ve[nrotor]
Fin

Polinomio_suma (P1, P2 : Polinomio)
Inicio
  // poner polinomio en 0
  Para cada i = 1 hasta P1.numero_terminos
  Inicio
    ex = P1.exponente(i)
    co = P1.coeficiente(ex)
    Poner_termino (co, ex)
Fin

WINNER

Fecha: / / Tema:

Evaluación Parcial Chaque

```
Para cada i = 1 hasta P2.numero_terminos
inicio
    ex = P2.exponente(i)
    co = P2.coeficiente(ex)
    poner_termino(co, ex)
    fin
fin

Polinomio resta(P1, P2 : Polinomio)
Inicio
    // poner polinomio en 0
    Para cada i = 1 hasta P1.numero_terminos
    inicio
        ex = P1.exponente(i)
        co = P1.coeficiente(ex)
        poner_termino(co, ex
    fin
    Para cada i = 1 hasta P2.numero_terminos
    inicio
        ex = P2.exponente(i)
        co = P2.coeficiente(ex)*-1
        poner_termino(co, ex)
    fin
fin

Polinomio multiplicacion(P1, P2: Polinomio)
  Inicio
      // desarrolle el algoritmo //
```

2.1.43

Esta forma de implementación es netamente académica en virtud a lo que busca es una mejor comprensión sobre las pertenencias, para ello se entiende que se usara como memoria nuestra clase CS memoria implementada en la unidad uno

Definiendo la clase polinomio

```
Tipo de Datos
        Nodo
            Coef Entero
            Exp Entero
            Sig Puntero a Nodo (valor interno para
                                esta implementación)
            fin
```

Luishiño Pericena Chope

Direccioń puntero a Nodo (valor entero para esta implementacioń)

```
Clase Polinomio
    Atributos
        Ptr_Pol Direccioń
          Nt    Entero
    Metodos
        Privado
            Direccioń BuscarExponente (Exp Entero)
            Direccioń TerminoN( i : Entero)
        Publico
        crea()
        Es Cero() devuelve (booleano)
        Grado () devuelve (grado del polinomio)
        Coeficiente (Exp: Entero) entero devuelve (coeficiente de termino)
        Asignar Coeficiente ( coef, exp : Entero)
        Poner_Termino (coef, exp : Entero)
        numero_Terminos () devuelve (nro. Terminos)
        Exponente (nroto : entero) devuelve (grado)
        Sumar ( P1, P2 : Polinomio)
        restar (P1, P2 : Polinomio)
        Multiplicar (P1, P2: Polinomio)
    Fin definicioń clase
```

Implementacioń clase polinomio utilizando Simulador de Memoria CS memoria

```
Direccioń polinomio.BuscarExponente ( Exp Entero)
  Inicio
      Dir = Ptr_Pol
      If Dir <> nulo entonces
          dirEx = Nulo
        Mientras (Dir <> nulo) y (dirEx = Nulo)
        Si M.Obtener_Dato (dir, '->exp') = exp entonces
            dirEx = dir
      Dir = M.Obtener_dato(dir, '->sig')
    Fin mientras
          retornar dirEx
      Caso contrario
          // exception no existe ese termino
    Fin
```

N° Tema:
Fecha: / /

Resolución Parcial chequeo

Dirección polinomio. BuscarTermino N (I: Entero)
Inicio
  Dir = Ptr_Poli
  if Dir <> nulo entonces
      dirTer = Nulo
      Nt = 0
      Mientras (Dir <> nulo) y (dirTer = Nulo)
  Nt = nt + 1
  Si nt = I entonces
      dirTer = dir
  dir = M. obtener_dato(dir, '→ sig')
Fin mientras
Retornar dirTer
Caso contrario
  // excepción no existe termino
Fin

Polinomio_crear()
Inicio
  nt = 0
  ptrpoli = nulo
Fin

Booleano polinomio. EsCero() devuelve (booleano)
Inicio
  retornar (nt = 0)
Fin

Entero polinomio_Grado() devuelve (Grado del polinomio)
Inicio
  Dir = Ptr_Poli
  if Dir <> Nulo entonces
      MaxG = M. obtener_dato (dir, '→ exp')
      Mientras (Dir <> nulo)
          Si M. obtener_Dato(dir, '→ exp') > MaxG entonces
      MaxG > M. obtener_Dato (dir, '→ exp')
      dir = M. Obtener_dato (dir, '→ sig')
  Fin mientras
  Retornar MaxG
  Caso Contrario
      // excepción no existe ese termino
Fin

Tema:

N°

Fecha: / /

Gustavo Perdomo Chaque

Entero Polinomio.coeficiente (exp: Entero) devuelve (coeficiente de Termino)

inicio

Dir = buscar Exponente (exp)

Si dir <> nula entonces

Retornar m.obtener_dato(dir).coef

caso contrario

// exception no existe ese termino

Fin

Polinomio.suma (P1, P2: Polinomio)

inicio

// poner Polinomio en cero

para cada i=1 hasta P2.numero_terminos

inicio

exp = P2.exponente(i)

co = P1.coeficiente(ex)

Poner_termino (co, ex)

Fin

Para cada i = 1 hasta P2.numero_terminos

inicio

ex = p2.exponente(i)

co = P2.exponente(ex)

Poner_termino (co, ex)

Fin

Fin

Polinomio Resta (P1, P2: Polinomio)

inicio

// poner polinomio en cero

para cada i=1 hasta P1.numero_Terminos

inicio

ex = P1.exponente(i)

co = P1.coeficiente(ex)

Poner_termino (co, ex)

Fin

Para cada i = 1 hasta P2.numero_terminos

inicio

ex = P2.exponente(i)

co = P2.coeficiente(ex) * -1

Poner_Terminos (co, ex)

Fin

WINNER

N°: Tema: Lunes 10 Parcial Chaque

Fecha: / /

Polinomio Multiplicación (P1, P2: Polinomio)
Inicio
// Desarrolle el algoritmo //
Fin

2.1.4.1 Implementación con Punteros

En esta forma de implementación planteada lo que se resalta son los cambios que tienen que hacerse al código de la implementación con el simulador de memoria estudiados que ahora se está trabajando con punteros, es así que se tiene de color rojo los cambios fundamentales en definición de la clase Polinomio

Tipo de dato
Nodo
Coef Entero
Exp Entero
Sig Puntero Nodo
Fin

Dirección de puntero a Nodo

Clase Polinomio
Atributos
Ptr_Poli Dirección
Nt Entero
Metodos
Privado
Dirección BuscarExponente (Exp Entero)
Dirección BuscarTermino (i: Entero)
Publico
Crear()
EsCero() devuelve (booleano)
Grado() devuelve (grado del Polinomio)
Coeficiente (Exp: Entero) devuelve (coeficiente de termino)
AsignarCoeficiente (coef, exp: entero)
Poner_Termino (coef, exp: Entero)
Numero_Terminos (coef, exp: entero)
numero_terminos() devuelve (Nro Terminos)
exponente (n_oter: entero) devuelve (Grado)
sumar (P1, P2: Polinomio)
restar (P1, P2: Polinomio)
Multiplicar (P1, P2: Polinomio)
Fin de definición clase

N°
Fecha: / /

Luishiño Pexcara Chaque

Implementación clase polinomio utilizando simulador de memoria
Direcciones polinomio.BuscarExponente(Exp Entero)
Inicio
Dir = Ptr_Pol
if Dir <> nulo entonces
dirEx = Nulo
Mientras (Dir <> Nulo) y (dirEx = Nulo)
si dir -> exp = Exp entonces
dirEx = dir
dir = dir -> sig
fin mientras
retornar dirEx
caso contrario
// Exception no existe ese termino
fin

Direccion Polinomio.BuscarTerminoN(I: Entero)
Inicio
Dir = ptr_Pol
if Dir <> nulo entonces
dirTer = Nulo
Nt = 0
Mientras (Dir <> nulo) y (dirTer = Nulo)
Nt = nt + 1
Si nt = I entonces
dirTer = dir
dir = dir -> sig
fin mientras
Retornar dirTer
caso contrario
exception no existe Termino
fin

Polinomio.Crear()
inicio
nt = 0
ptr -> Pol = nulo
Fin

Booleano polinomio.Escero() devuelve (booleano)
inicio
Retornar (nt = 0)
fin

WINNER

Fecha: / / Tema:

Ejercicio Polinomio Grado

Entero polinomio.Grado() devuelve (Grado del polinomio)
inicio
    Dir = Ptr_Pol;
    if Dir <> nulo entonces
        MaxG = dir → exp
        mientras (Dir <> nulo)
            si dir → exp → MaxG entonces
                MaxG = dir → exp
            dir = dir → sig
        fin mientras
        Retornar MaxG
    Caso contrario
        // excepción no existe ese termino
fin

Entero Polinomio.coeficiente(Exp: Entero) devuelve (Coeficiente de terminos)
inicio
    Dir = buscarExponente(exp)
    Si dir <> nulo entonces
        Retornar dir → coef
    Caso contrario
        // excepción no existe ese termino
fin

Asignar Coeficiente(Coef, exp : Entero)
inicio
    Dir = buscar Exponente(exp)
    Si dir <> nulo entonces
        dir → coef = Coef
    caso contrario
        // no existe ese termino
fin

Polinomio poner_termino (Coef, exp : Entero)
inicio
    existe = buscar Exponente(exp)
    Si existe = nulo
        entonces
    aux = New Nodo
    si aux <> nulo entonces
        aux → coef = coef
        aux → exp = exp
        aux → sig = Ptr_Pol;
    Ptr_Pol = aux
    nt = nt + 1

Tema:

N°

Fecha: / /

Luzhiño Pencen Chique

// error espacio memoria
caso contrario
NuevoCoef = existe → coef + Coef
existe → coef = NuevoCoef
// Eliminar Nodo si nuevo Coef es 0
fin

Polinomio Suma (P1, P2 : Polinomio)
inicio
// poner polinomio en 0
Para cada i = 1 hasta P1.numero_Terminos
inicio
ex = P1.exponente(i)
co = P1.coeficiente(ex)
Poner_Termino (co, ex)
fin
Para cada i = 1 hasta P2.numero_terminos
inicio
ex = P2.exponente(i)
co = P2.coeficiente(ex)
Poner_termino (co, ex)
fin
fin

Para cada i = 1 hasta P2.numero_terminos
inicio
ex = P2.exponente(i)
co = P2.coeficiente(ex)
Poner_termino(co, ex)
fin
fin

Polinomio.resta (P1, P2: Polinomio)
inicio
// poner polinomio en cero
Para cada i = 1 hasta P1.numero_terminos
inicio
ex = P1.exponente(i)
co = P1.coeficiente(ex)
poner_termino(co, ex)
fin
Para cada i = 1 hasta P2.numero_terminos
inicio
ex = P2.exponente(i)
co = P2.coeficiente(ex)*-1
Poner_termino(co, ex)
fin
fin

WINNER

N° _____ Tema: _____

Fecha: ___/___/___

Lushizo Penicone Orozco

Polinomio.numero_terminos() devuelve (nro Terminos)
inicio
    retornar nt
fin

Polinomio.multiplicacion (P1, P2: Polinomio)
inicio
    // Desarrolle el algoritmo //
fin

Implementación Polinomio Lista

.h

```
Using namespace std;
class Poli {
 Private
     Lista P * poli;
     Nodo L* buscar_exponente (int exp);
     Nodo L* buscar_Termino (int i);

Public:
     Poli2();
  bool es = cero();
  int grado();
  int coeficiente (int exp)
  void asignar - coeficiente (int coef, int exp);
  void poner_termino(int coef, int exp);
  int numero_terminos();
  int exponente (int nro_ter);
  string to_str();
 }
# End.f
```

Nombre Patricia Chaque

```
PoliL: PoliL() {
    Pol = new listaP();
}
Nodo * PoliL :: new listaP();
}

Nodo * PoliL :: buscar_exponente (int exp){
    Nodal * aux = pol -> primero();
    while (aux != NULL){
        Nodo2 * sig = pol -> siguiente (aux);
        if (pol -> recupera (sig) == exp)
            return sig;
        aux = Pol -> Siguiente (Pol -> Siguiente (aux));
    }
    return Null;
}
    Nodo2 * PolL :: buscar_termino (int i){
        int c = 1
        Nodo2 * aux = pol -> promedio();
            while (aux) = NULL){
                if (c == i)
                return aux;
            aux = Pol -> Siguiente (Pol -> Siguiente (aux));
        c++;
    }   return Null;
    }

bool PolL :: es_cual(){
    return pol -> vacia();
}

int PolL :: Grado(){
        if (! es.cond){
    Nodo2 * aux = Pol -> Siguiente (Pol -> primero());
    int max = pol -> recupera (aux);
    while (aux != Null){
        if ( pol -> recupera (aux) -> max)
            max = pol -> recupera (aux);
            if (aux == pol -> fin())
                break;
        else
            aux = Pol -> Siguiente (Pol -> Siguiente ((aux));
    }
        return max;
    }
    else   cout << "Lista vacia \n";
}
```

N° _____ Tema: _____

Fecha: ____/____/____

```
int PolL::coeficiente(int exp){
    if(!es_cero()){
Nodo * dir_exp = buscar_exponente(exp);
    if(dir_exp != NULL){
      Nodo * dir_coef = pol->anterior(dir_exp);
      return pol->recuperar(dir_coef)
    }
      else
          cout << "no existe \n";
  }
}

Void PolL::asigna coeficiente(int coef, int exp);
    if(dir_exp != NULL){
Nodo* dir_coef = Pol->anterior(dir_exp)
    Pol->modifica(dir_coef, coef);
      if(coef == 0){
          Pol->suprime(dir_exp);
          Pol->suprime(dir_exp);
  }
  }   else
          cout << "no existe termino \n";
  }
}
```