Materia: _____ Fecha: __/__/__

Tema: _____

UNIDAD III TDA Polinomio

2.1.1 Descripción del TDA Polinomio.

El prefijo "poli" significa muchos. Polinomio entonces significa muchos términos. Las expresiones Polinomiales que contienen uno, 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 5 términos en adelante.

Polinomio es la suma finita de expresiones Ax^m (si es de una variable) o de la forma $Ax^m y^n$ (si es de 2 variables).

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_2 x^2 + a_1 x + a_0 :$$

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 variables

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 presenta todos los exponentes desde 0 hasta el mayor incluso

Polinomios idénticos: Son 2 polinomios del mismo grado, con las mismas variables, coeficientes iguales (Términos iguales).

Polinomio opuesto: Son aquellos que tienen los mismos 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.

En primer lugar, debemos identificar el conjunto de operaciones que definiremos. Para ello, deberemos considerar como se va a 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 grado que corresponde al polinomio almacenado.

Materia: _____ Fecha: __/__/__

Tema: _____

- un método para sumar dos polinomios
- un método para restar dos polinomios
- etc.

2.1.2 Especificación del TDA Lista.

Partiendo de lo establecido en la unidad 1. Especificación informal tenemos lo sgte:

Elementos que conforman la estructura

TDA Polinomio (VALORES son coeficientes y grados del Tipo entero, OPeraciones crea, escCero, grado, poner_termino, coeficiente, sumar, multiplicar, restar, numero_terminos, Exponente)

OPERACIONES

Crea (P: Polinomio)
Utilidad: Sirve para inicializar el polinomio
Entrada: Polinomio P
Salida: Ninguna
Precondición: Ninguna
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 ← para que Ya te haga esto primero cumpli este requisito
Poscondición: Ninguna

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

Coeficiente (P: Polinomio, Exp: Entero) devuelve (coficiente del termino)
Utilidad: Devuelve el coficiente que corresponde al termino con exponente Exp.
Entrada: polinomio P
Salida: Numero
Precondición: Polinomio no Cero
Poscondición: Ninguna

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

tilibra

Fecha: __/__/__

Materia:

Tema:

Poner_termino (P: polinomio; coef, exp: entero)
Utilidad: Modifica polinomio P asignando 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 del termino con grado exp.

Numero_terminos (P: Polinomio) devuelve (Nro. Terminos)
Utilidad: Determinar el numero de terminos que tiene el polinomio
Entrada: Polinomio P
Salida: Numero
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: Numero
Precondición: Que el nroter exista
Poscondición: Ninguna

Sumar (P1, P2: Polinomio; ES P: Polinomio)
Utilidad: Realiza la suma P1 + P2 y la almacena en el polinomio P.
Entrada: Polinomio P1, P2
Salida: Ninguno
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: Ninguno
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

Materia: Fecha: __/__/__

Tema:

2.1.3 Aplicaciones 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.

Ej 1: Implementar un algoritmo que Busque un elemento en la Lista L y retorne la Dirección donde se encuentra, caso contrario NULO

Ej 1': Implementar un algoritmo que dado un Polinomio P asigne su derivada a P1

```
Derivada (Polinomio P, E S Polinomio P1)
  Inicio
    Para cada i=1 hasta P.numero_terminos ()
      inicio
        exp = P.exponente (i)
        co = P.coeficiente ( ex)
        P1. poner_termino ( co*ex, ex-1)
      fin
  fin
```

$P(x) = 5x^3 + 3x^1 + 2x^0$

Derivada

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

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

```
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.
```

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

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

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

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

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

https://www.youtube.com/watch?v=d7r90m4kcuM.

2.1.4 Implementación de la clase Polinomio

2.1.4.1 Implementación con Listas

Para la implementación del TDA Polinomio se utilizara una lista POL que contendrá los coeficientes y exponentes de cada termino, se hace notar que la cantidad de términos estará dada por la longitud de la lista.

Materia:

Tema:

Fecha: __/__/__

Definiendo la Polinomio

Tipo de Datos

Clase polinomio

Atributos

Pol : Lista

Métodos

Privado

Direccion Buscar Exponente (Exp. Entero)

Direccion BuscarTerminoN (i: Entero)

Publico

Crea ()

Booleano EsCero ()

Entero Grado ()

Entero Coeficiente (EXP: Entero)

Asignar Coeficiente (coef, exp: Entero)

Poner_termino (coef, exp: entero)

Entero. numero_terminos()

Entero exponente (nroter : entero)

Sumar (P1, P2: Polinomio)

restar (P1, P2: Polinomio)

multiplicar (P1, P2: Polinomio)

Fin definicion clase.

Direccion Polinomio_Buscar Exponente (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 (Dir Exp = NULO)

Si pol. recupera (dir) = exp entonces

Dir Exp = Dir

Dir = pol. siguiente (pol. siguiente(dir))

Fin mientras

Retornar DirExp

Caso Contrario

// excepcion polinomio no tiene términos

FIN

Direccion BuscarTerminoN (i: Entero)

inicio

Dir = pol. primero

Ut = 0

Si dir <> nulo entonces

dirter = Nulo

mientras (dir <> nulo) y (Dirter = Nulo)

↓

Materia: Fecha: __/__/__

Tema:

```
        nt = nt + 1
        Si nt = i entonces
            DirTer = Dir
            Dir = Pol.Siguiente (Pol.Siguiente (dir))
        Fin mientras
        Retornar DirTer
        Caso Contrario
        // exception polinomio no tiene terminos
Fin.
Polinomio.Crear
inicio
   Pol.crear // llamada al constructor de Pol
Fin
https://youtu.be/XxYKHH2ds28
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 terminos
Fin
https://youtu.be/xUiQHDHKgMA
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 termino
Fin
https://youtu.be/eHYVSMBXOBg
polinomio.asignarcoeficiente (coef, exp: entero)
inicio
   Dir = BuscarExponente (exp)
   Si dir <> nulo entonces
```

Materia:

Tema:

Fecha: __/__/__

```
dirCoef = pol.anterior (dir)
Pol. modifica (dirCoef, Coef)
Si coef = 0 entonces
        Pol. Suprime (dir)
        Pol. Suprime (dirCoef)
    Caso Contrario
    // exception polinomio no tiene ese termino
Fin
booleano  Polinomio_poner_termino (coef, exp: entero)
  Inicio
    Dir Exp = Buscar Exponente (exp)
    Si Dir Exp <> Nulo entonces
        DirCoef = pol. anterior (dirExp)
        Pol. modifica (dirCoef, pol.recupera(Dir coef) + coef)
        Si pol. recupera (dirCoef) = 0 entonces
            Pol. suprime (dirExp)
            pol. suprime (dirCoef)
        Caso contrario
            Si coef <> 0 entonces
                Pol. inserta Ultimo (exp)
                Pol. inserta (pol. fin, coef)

Fin
https // youtu.be / jvwNLZQrFYM
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
    // exception no existe ese número de termino.
Fin
https://youtu.be/K46UDZZA99g
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
```

tilibra

Materia: Fecha: __/__/__

Tema:

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 Pi = 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 //
Fin.

2.1.4.2 Implementación con Vector

Para la Implementación del TDA polinomio se utilizara 2 vectores y un Atributo denominado nt, donde los vectores serán los que contendrán los coeficientes y exponentes de cada termino, se hace notar que nt determinara el número de términos que contiene el polinomio

Definiendo la polinomio
Constante max = 100
Tipo de Datos
Clase Polinomio
Atributos
VC // coef
VE: Arreglo(MAX) // expo
nt: Entero
Metodos
Crea()
IsCero() devuelve (booleano)
Grado() devuelve (Grado del polinomio)
Coeficiente(Exp: Entero) devuelve (Coeficiente de Termino)

Materia:

Tema:

Fecha: __/__/__

```
        AsignarCoeficiente ( coef, exp: Entero)
        Poner-termino (coef, exp: entero)
        Numero-terminos () devuelve (Nro. Terminos)
        exponente (nroter: entero) devuelve ( Grado)
        Sumar ( P1, P2 : Polinomio)
          restar ( P1, P2 : Polinomio)
          multiplicar ( P1, P2 : Polinomio)
        Fin definición clase.
Constructor Polinomio.Crear
  Inicio
     nt=0
  fin
  https:// youtube/m-SU3xrfmxg.
Publico Polinomio. Escero()
   inicio
     retornar ( nt = 0)
  fin
Entero polinomio.grado()
 Inicio
    Si nt >0 entonces
       max = ve [1]
       para cada i:=1 hasta nt
          Si ve[i] > max entonces max=ve[i]
       retornar max
       Caso contrario.
         // error no existe terminos
Fin
https://youtu.be/7jsr66x-hEc
Polinomio.asignarcoeficiente( coef, exp: entero)
 Inicio
    lug = // Existe exponente (exp) en la estructura revisando vector ve
    Si lug <> -1 entonces
       vc [lug] = coef
       Si vc[lug] = 0 entonces
            // desplazar 1 elemento hacia la posición lug
          nt = nt - 1
       Caso contrario
          // exception error no existe termino con ese exp.
Fin
Polinomio.poner-termino (coef, exp: entero)
Inicio
    lug = // existe exponente (exp) en la estructura revisando vector ve
    Si lug <> -1 entonces
          vc [lug] = vc [lug] + coef
          Si vc[lug] = 0 entonces
             // desplazar 1 elemento hacia posición lug
             nt = nt - 1
```

Materia: Fecha: __/__/__

Tema:

```
    caso contrario
    nt = nt + 1
    vc[nt] = coef
    ve[nt] = exp
Fin
https://youtube/KVGCdp49EEco
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 vc[i]
       fin
    //error no existe termino con ese exponente
  Fin
https://youtu.be/KSKSe1m4WAs
Entero polinomio.exponente (nroter: entero) devuelve (Grado)
Inicio
    Retornar ve [nroter]
fin
https://youtu.be/QT1WU154J50
Polinomio.Suma (p1, p2: polinomio)
Inicio
  // poner polinomio en 0
  para cada i=1 hasta p1.numero_terminos
    inicio
       exp = p1.exponente (i)
       co = p1.coeficiente (ex)
       poner_termino (co, ex)
    fin
  para cada i=1 hasta p2.numero_terminos
    Inicio
       exp= p2.exponente (i)
       co= p2.coeficiente (ex)
       poner_terminos (co, ex)
    fin
Fin
Polinomio.resta (p1, p2: polinomio)
Inicio
  // poner polinomio en 0
  para cada i=1 hasta p1.numero_terminos
    inicio
```

Materia: Fecha: __/__/__

Tema:

↓

ex = p1. exponente (i)
co = p1. Coeficiente(ex)
poner_ terminos (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
Fin
Polinomio. multiplicación (p1, p2: polinomio)
Inicio
// Desarrollo del algoritmo //
fin

2.1.4.3 Implementación con Simulación de Memoria (usando la clase CSMemoria)

Esta forma de implementación es netamente académica en virtud a que lo que hace es una mera compensión sobre los punteros, para ello se entiende que se usará como Memoria nuestra clase CSMemoria implementada en la unidad uno.

Definiendo la clase Polinomio

Tipo de dato
Nodo
Coef Entero
Exp Entero
Sig. Puntero a Nodo (valor entero para esta implementación)
fin
Dirección Puntero a Nodo (valor entero para esta implementación)

Clase Polinomio
Atributos
Ptr_Poli: Dirección
Nz : Entero
Metodos
Privado
Dirección BuscarExponente (Exp Entero)
Dirección BuscarTerminoN(i : Entero)
Publico.
Crear ()
EsCero () devuelve (booleano)
Grado() devuelve (Grado de Polinomio)
Coeficiente (Exp: Entero) devuelve (coeficiente de Termino)
AsignarCoeficiente (coef, exp: Entero)
poner_terminos (coef, exp: Entero)
numero_terminos () devuelve (NZ, Terminos)
exponente (nroter: entero) devuelve (Grado)
Sumar (p1, p2 : Polinomio)
restar (p1, p2 : Polinomio)

Materia: Fecha: __/__/__

Tema:

multiplicar (P1, P2: Polinomio)
fin definicion clase
Implementación clase polinomio utilizando Simulador de Memoria (SMemoria.
Direccion Polinomio. BuscarExponente (Exp: Entero)
Inicio
Dir = ptr_Pol
if Dir <> nulo entonces
dir Ex = NULO
Mientras (Dir <> nulo) y (dir Ex = Nulo)
Si M.obtener_Dato (dir, '->exp') = Exp entonces
dir Ex = dir
dir = M.obtener_dato (dir, '->sig')
Fin mientras
Retornar dir Ex
Caso Contrario
// Exception no existe ese termino
Fin.
Direccion Polinomio. BuscarTerminoN (i: Entero)
Inicio
Dir = ptr_pol
if Dir <> nulo entonces
dir Ter = Nulo
nt = 0
Mientras (Dir <> nulo) y (dirTer = Nulo)
Nt = nt + 1
Si nt = i entonces
dir ter = dir
Retornar dir Ter
fin.
Polinomio.Crear()
Inicio nt = 0 ptr poli = nulo
fin
https://y.u-to.be/YVqDRyr9M7Q
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
Max G = M.obtener_dato (dir, '->exp')
Mientras (Dir <> nulo)
Si M.obtener_dato (dir, '->exp') > Max G entonces
Max G = M.obtener_dato (dir, '->exp')
dir = M.obtener_dato (dir, '->sig')
Fin mientras
Retornar Max G
Fin // https://youtu.be/13E6H19h

tilibra

Materia: Fecha: __/__/__

Tema:

```
Entero polinomio.coeficiente ( Exp: Entero) devolue (coeficiente de termino)
Inicio
    Dir = buscar Exponente (exp)
    Si dir <> nulo entonces
                Retornar m.obtener_dato (dir, '->coef')
            Caso Contrario
// Fin // https://youtu.be/cGgHiJK68
Asignar Coeficiente ( coef, exp: Entero)
Inicio
    Dir = buscar Exponente (exp)
    Si dir <> nulo entonces
            m. poner_dato (dir, '->coef', coef)
            Si coef = 0 entonces
                    //eliminar nodo Dir
Fin             Caso Contrario
Polinomio. poner_termino ( coef, exp: Entero)
Inicio  existe = buscar Exponente (exp)
        Si existe = nulo
          entonces
          aux = H.New_Espacio ('coef, exp, Sig')
          Si aux <> nulo entonces
                  m. poner dato (aux, '->coef', coef)
                  m. poner dato (aux, '->exp', exp)
                  Ptr_poli = AUX
          Caso contrario
            Nuevo Coef = m.obtener_dato (existe, '->coef') + coef
            m. poner dato (existe, '->coef', Nuevo Coef)
fin // https://youtu.be/nSKGPUKlm96
entero polinomio.numero_terminos() devuelve (w.q.terminos)
Inicio  retornar nt
fin
entero polinomio.exponente ( nroter: entero) devuelve (dirnodo)
Inicio
    dir = buscartermino N ( nroter)
    dir dir <>nulo entonces
                retornar m. obtener dato (dir, '->exp')
            Caso contrario
              //no existe ese termino
fin
https://youtu.be/adgHiJ@968C
```

Materia: Fecha: / /

Tema:

Polinomio. suma (P1, P2: polinomio)
Inicio
// poner polinomio en 0
Para cada i=1 hasta P1. numero_terminos
Inicio
exp = p1. exponente (i)
Co = P1. coeficiente (ex)
Poner_termino (co, ex)
fin
Para cada i=1 hasta P2. numero_terminos
Inicio
exp = P2. exponente (i)
Co = P2. coeficiente (ex)
Poner_termino (coef, exp)
fin
fin

Polinomio_resta (P1, P2: polinomio)
Inicio // poner polinomio 0
Para cada i=1 hasta P1. numero_terminos
inicio
exp = P1. exponente (i)
Co = P1. coeficiente (ex)
fin Poner_termino (co, ex)
Para cada i=1 hasta P2. numero_terminos
inicio
exp = P2. exponente (i)
Co = P2. coeficiente (ex) * -1
Poner_termino (co, ex)
fin
fin

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

2.4.4 Implementación Clase punteros

En esta fase de implementación planteada lo que se resalta son los cambios que tienen que hacerse al código de la implementación con el simulador de memoria. Considerando el cual se está trabajando con punteros, es decir que se tiene de cabe poso los cambios fundamentales en los algoritmos ya vistos quedando para resolver las definiciones formales en C++

Definición la clase Polinomio
Tipo de Dato
Nodo
Coef. Entero
Exp Entero
Sig. Puntero a Nodo
fin

Materia: Fecha: __/__/__

Tema:

Dirección Puntero a Nodo
Clase Polinomio
Atributos
Ptr_Poli: Dirección
Nt Entero
Metodos
Privado
Dirección BuscarExponente (Exp Entero)
Dirección BuscarTerminoN (i: Entero)
Publico
crea()
EsCero () devuelve (booleano)
Grado () devuelve (Grado del Polinomio)
coeficiente (Exp: Entero) devuelve (coeficiente de Termino)
Asignar coeficiente (coef, exp: entero)
Poner_terminos () devuelve (exp: entero)
numero_terminos () devuelve (Nro. Terminos)
exponente (nroter entero) devuelve (Grado)
Sumar (P1, P2: Polinomio)
restar (P1, P2: Polinomio)
multiplicar (P1, P2: Polinomio)
Fin definición clase

Implementaciones clase Polinomio utilizando Simulador de Memoria (SIMemoria.

Dirección Polinomio.BuscarExponente (Exp Entero)
Inicio
Dir = ptr_poli
if Dir <> nulo entonces
dirEx = Nulo
Mientras (Dir <> nulo) y (dirEx = Nulo)
Si dir → Exp = Exp entonces
dirEx = dir
dir = dir → sig
Retornar dirEx
// exception no existe ese término
Fin

Dirección Polinomio.BuscarTerminoN (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 = dir → sig
Fin mientras
Retornar dirTer
Caso Contrario
// exception no existe terminos
fin

Materia: Fecha: / /

Tema:

Polinomio. Crear ()
Inicio
nt = 0
ptr_poli := nulo
fin

Booleano polinomio. Es Cero () devuelve (booleano)
Inicio
Retornar (nt = 0)
fin

Entero polinomio. Grado () devuelve (Grado del polinomio)
Inicio
Dir = ptr_poli
if Dir <> nulo entonces
Max G = dir → exp
Mientras (Dir <> nulo)
Si dir → Exp > MaxG entonces
Max G = dir → exp
dir = dir → sig
fin mientras
Retornar MaxG
// exception no existe ese termino
Fin

Entero Polinomio. coeficiente (Exp: Entero) devuelve (coeficiente de Termino)
Inicio
Dir = buscar Exponente (exp)
Si dir <> nulo entonces
Retornar dir → coef
Caso contrario
// excepcion no existe ese termino
Fin

Asignar Coeficiente (coef, exp: Entero)
Inicio
Dir = buscar Exponente (exp)
Si dir <> nulo entonces
dir → coef = coef
Si coef = 0 entonces
Caso Contrario //elimina nodo Dir
// no existe ese termino
fin

Polinomio. exponente (nvoter: entero) devuelve (Grado)
Inicio
dir = buscarTermino NC (Nvoter)
Si dir <> nulo entonces
retornar dir → exp
Caso contrario
// no existe ese termino

fin

Materia:

Tema:

Fecha: __/__/__

```
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_poli
            ptr_poli = Aux
            nt = nt + 1
    Que contrario Caso contrario //error espacio memoria
    Nuevo coef = existe→ coef + coef
    existe→ coef = Nuevo coef // 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
        exp = p1. exponente (i)
        co = p1. coeficiente (ex)
fin     Poner_termino (co,ex)
Para cada i=1 hasta P2. numero_terminos
inicio  exp = 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  exp = p2. exponente (i)
        co = p2. coeficiente (ex) * -1
        poner_termino (co,ex)
    fin
fin
Polinomio.numero_terminos() devuelve (Nro. terminos)
Inicio
    retornar nt
fin
Polinomio.multiplicacion (P1, P2 : polinomio)
Inicio
    // Desarrollo del algoritmo //
fin.
```

Materia: Fecha: __/__/__

Tema:

Polis.cpp

```
#pragma hdrstop
#include "Polis.h"
Polis::Polis(){
        mem= new CsMemoria();
        ptr.poli=nulo;
        nt=0;
}
Polis::Polis(CsMemoria*m){
        mem=m;
        ptr.poli=NULO;
        nt=0;
}
int Polis::buscar.exponente(int exp){
        int aux=ptr.poli;
        while(aux!=NULO){
          int aux.exp=mem->obtener.dato(aux,exp);
            if(aux.exp)==exp)
                return aux;
          aux=mem->obtener_dato(aux,sig);
        }return NULO;
}
int Polis::buscar_termino(int i){
        int c=1;
        int aux=ptr.poli;
        while(aux!=NULO){
            if(c==i)
                return aux;
          c++;
        }aux=mem->obtener_dato(aux,sig);
        return NULO;
}
bool Polis::es_cero(){

        return ptr.poli==nulo;

}
int Polis::grado(){
        if(!e.cero()){
            int max=mem->obtener.dato(ptr.poli,exp);
            int aux=ptr_poli;
            while(aux!=NULO){
            int aux_exp=mem->obtener_dato(aux,exp);
                if(aux_exp->max)
                    max=aux_exp;
                aux=mem->obtener_dato(aux,sig);
            }
            return max;
        }
}
```

Materia: Fecha: __/__/__

Tema:

```
int PoliS :: coeficiente (int exp){
    int dir = busar_exponente (exp);
        if (dir != NULO)
            return mem -> obtener_dato (dir, coef);
        else
            cout << " No existe ese termino" << endl;
}
int PoliS :: anterior (int dir) {

    int aux = ptr_nulo;
    int aux = NULO;
    while aux != NULO && aux != dir){
            ant = aux;
            aux = mem -> obtener_dato (aux, sig);
    }
    return ant;
}

void PoliS :: Suprime (int dir) {
    if (dir == ptr_poli) {

        ptr_poli = mem -> obtener_dato (ptr_poli, sig);
    }
    else {

        int aux = ptr_poli;
        int ant = NULO;
            while (aux != NULO && aux != dir) {
                ant = aux;

                aux = mem -> obtener_dato (aux, sig);
            }
            int ante = ant;
            int dir_sig = mem -> obtener_dato (dir, sig);
            mem -> poner_dato (ante, sig, dir_sig);
        }

    mem -> delete_espacio (dir);
}
```

Materia: Fecha: / /

Tema:

```
void Polis::asignar_coeficiente(int coef, int exp){
    int dir = buscar_exponente(exp);
    if (dir != NULO){
        mem->poner_dato(dir, coef, exp);
        if (coef==0){
            suprime(dir);
            nt--;
        }
    }
}

void Polis::poner_termino(int coef, int exp){
    int dir = buscar_exponente(exp);
    if (dir == NULO){
        if (coef != 0){
            int nuevo = mem->new_espacio(datos);
            mem->poner_dato(nuevo, coef, coef);
            mem->poner_dato(nuevo, exp, exp);
            ptr_poli = nuevo;
        }
    }
    else{
        int new_coef = coef + mem->obtener_dato(dir, coef);
        mem->poner_dato(dir, coef, new_exp);
        if (new_coef == 0){
            suprime(dir);
            nt--;
        }
    }
}
int Polis::numero_termino(){
    return nt;
}
int Polis::exponente(int nro_ter){
    int dir = buscar_termino(nro_ter);
    if (dir != NULO)
        return mem->obtener_dato(dir, exp);
    else
        cout << "No existe ese termino" << endl;
}
```

Materia:

Tema:

Fecha: __/__/__

```
String Polis::to_str(){
    string r = "";
    int n = numero_terminos();
    for(int i=1; i<=n; i++){
        int exp = exponente(i);
        int coef = coeficiente(exp);
        if(coef > 0)
            r += "+";
        r += to_string(coef) + "x^" + to_string(exp);
    }
    return r;
}

void Polis::derivar(Poli p, polis z){
    n+1 = 1;
    if(es_cero()){
        cout << "El polinomio es cero: 0";
    } else {
        while(i<=numero_terminos()){
        // int exp1 = 0
            ex = p->exponente(i);
            coef = p->coeficiente(exp);
            z->poner_termino(coef, exp, exp-1);
        }
}

float Polis::evaluar(float x){
    float sum=0
    for(int i=0; i<numero_termino(); i++){
        int exp = exponente(i+1);
        int coef = coeficiente(exp);
        Sum 1 = coef * pow(x, exp);
    }
    return Sum;
}
```

Materia:

Tema:

Fecha: __/__/__

PoliS.h

```
# Ondef  PoliSH
# define PoliSH

# include <iostream>
# include <String>
# include " UMemoria.h"

Using namespace std;

Const String datos = " coef, exp, sig"
Const string coef = " → coef";
Const String exp = " → exp";
Const String sig = " → sig";

Class PoliS {
Private:
      CSMemoria * mem;
       int ptr_poli;
       int nt;

       int buscar_exponente (int, exp);
       int buscar_termino ( int i);
       int anterior (int dir);
       void Supreme (int dir);
public:

       PoliS ();
       PoliS ( CSMemoria * mem);
       bool es_cero ();
       int grado ();
       int coef_en_exp (int exp);
       void asignar_coeficiente (int coef, int exp);
       void poner_termino (int coef, int exp);
       int numero_termino ();
       int exponente (int nro_ter);
       void derivar ( PoliS P1, PoliS P2);
       float evaluar ( float z);

};
# endif
```

Materia:

Tema:

Fecha: __/__/__

Poli V.h

```
#ifndef PoliVH
#define PoliVH

#include <iostream>
#include <string>

using namespace std;
const int MAX=10;

Class Polinomio {
Private:
    int VC[MAX];
    int VE[MAX];
    int nt;
Public:

    Polinomio ();
    bool es_cero();
    int grado ();
    Void asignar(coeficiente (int exp,int coef);
    Void Poner_termino (int coef, int exp);
    int coeficiente (int exp);
    int exponente (int nro);
    int numero_termino ();
    Void Suma (Polinomio P1);
    Void resta (Poli P1, Poli P2);
    Void multiplicar (Poli P1, Poli P2);
    Float evaluar (float x);
    Void mostrar (float x);

    Void mostrar ();
    Void destruir ();
};
#endif
```

Materia: Fecha: __/__/__

Tema:

PoliV.cpp

```
#include "PoliV.h"

Polinomio::Polinomio(){
    nt = 0;
}

void eliminar_posicion(int *v, int k, int nt){
    for(int i=k; i<nt-1; i++){
        v[i] = v[i+1];
    }
}

int existe_exponente(int Exp, int ve[MAX], int nt){
    for(int i=0; i<nt; i++){
        if(ve[i]==Exp
            return i;
    }
    return -1;
}

bool polinomio::es_cero(){
    return nt==0;
}

void Polinomio::asignarCoeficiente(int exp, int coef){
    int lug = existe_exponente(exp, ve, nt);
    if(lug!=-1){
        vc[lug] = coef;
        if(ve[lug]==0)
        {
            eliminar_posicion(vc, lug, nt);
            eliminar_posicion(ve, lug, nt);
            nt--;
        }
    }else
    cout<<"Polinomio no tiene ese termino en ese EXP";
}
void Polinomio::poner_termino(int coef, int exp){
    int lug = existe_exponente(exp, ve, nt);
    if(lug!=-1){
        vc[lug] = vc[lug] + coef;
        if(vc[lug]==0){
            eliminar_posicion(vc, lug, nt);
            eliminar_posicion(ve, lug, nt);
            nt--;
        }
}
```

Materia: Fecha: __/__/__

Tema:

```
    else {
        nt++;
        vc[nt-1] = coef;
        ve[nt-1] = Exp;
    }
}
int Polinomio :: coeficiente (int exp){
    if (exp >= 0 && exp <= grado ()){
        for (int i=0; i<nt; i++){
            if (ve[i] == exp)
                return vc[i];
        }
        cout << "Error no existe termino con ese exponente\n";
}
int Polinomio :: exponente (int nro){
    if (nro < nt)
        return ve[nro];
    else
        cout << "Error fuera de Rango\n";
}
int Polinomio :: grado (){
    if (nt > 0){
        int max = ve[0];
        for (int i=0; i<nt; i++){
            if (ve[i] > max)
                max = ve[i];
        }
        return max;
    }
    else
        cout << "no existen termino s\n";
}
int Polinomio :: numero_terminos (){
        return nt;
}
void Polinomio :: Suma (Polinomio otro)
{   for (int i=0; i<otro.numero_terminos(); i++){
        int Expl = otro.exponente(i);
        int coefl = otro.coeficiente(Expl);
        Poner_termino (coefl, Expl);
    }
}
```

Materia: Fecha: __/__/__

Tema:

```
void Polinomio::resta (Polinomio otra) {
for (int i=0; i< otro.numero_terminos(); i++) {
        int exp2 = otro.exponente(i);
        int coef2 = -otro.coeficiente(exp2);
        Poner_termino(coef2, exp2);
    }
}

Void Polinomio::multiplicar (Polinomio otra)
for (int i=0; i < otro.numero_terminos(); i++)
{
    for (int j=0; j<nt; j++)
    {
        int exp1 = this→ exponente(i);
        int coef1 = this→ coeficiente(exp1);
        int Exp2 = otro.exponente(j);
        int coef2 = otro.coeficiente(Exp2);
        int coef3 = coef1 * coef2;
        int Exp3 = exp1 + Exp2;
        Poner_termino(coef3, Exp3);
    }
}

Void Polinomio::Mostrar() {
String ret = "";
for (int i=0; i<nt; i++) {
    int exp = this→ exponente(i);
    int coef = this→ coeficiente(exp);
    String signo;
    if (coef >0)
        signo = "+";
    else {
        signo = "-";
        coef = coef * -1;
    }
    ret = ret + signo + to_string(coef) + "x^" + to_string(exp);
}
cout << ret << endl;
}
```

Fecha: __/__/__

Materia:

Tema:

```
float Polinomio :: evaluar (float x){
    float elev = 1;
        float res = 0;
        int i = 1;
        while (i <= numero_terminos()){
            if (ve[i] == 0){
                elev = 1;
            }else{
                elev = 1;
                for (int K = 1; K <= ve[i]; K++){
                    elev = elev * x;
                }
            }
            res = res + (elev * vc[i]);
        i++;
        }
    return res;
}

void Polinomio :: Derivada ()
{   int i = 1;
    if (Escero()){
        cout << "El polinomio es cero :0";
    }else{
        while (i <= numero_terminos()){
            vc[i] = vc[i] * ve[i];
            ve[i] = ve[i] - 1;
            if (vc[i] == 0){
                for (int K = i; K < numero_terminos(); K++){
                    vc[K] = vc[K+1];
                    ve[K] = vc[n+1];
                    n = n - 1;
                }
            }else{
                i++;
            }
        }
    }
}
```

Materia: Fecha: __/__/__

Tema:

Polip.cpp

```
Polip :: Polip (){
    ptr = poli = NULL;
    nt = 0;
}
Nodo * Polip :: buscar_exponente (int exp){
    NodoPo * aux = ptr_poli;
    while (aux != NULL){
        if (aux -> exp == exp)
            return aux;
    }
    return NULL;
}

NodoPo * polip :: buscar_termino_n (int i){
    int c = 1;
    Nodo Po * aux = ptr_poli;
    while (aux){
        if (c == i)
            return aux;
        c++;
        aux = aux -> sig;
    }
    return NULL;
}

bool Polip :: grado (){
    if (!escero ()){
        int max = ptr_poli -> exp;
        NodoPo * aux = ptr_poli;
        while (aux){
            if (aux -> exp > max)
                max = aux -> exp;
            aux = aux -> sig;
        }
        return max;
    }
}
```

Materia:

Tema:

Fecha: __/__/__

```
int PoliP :: coeficiente (int exp){
    NodoPo * dir = buscar_exponente (exp);
    if (dir)
        return dir -> coef;
    else
        cout << "No existe ese termino" << endl;
}
NodoPo* PoliP :: anterior (NodoPo* dir){
    NodoPo* aux = ptr_poli;
    NodoPo* ant = NULL;
    while (aux && aux != dir){
        ant = aux;
        aux = aux -> sig;
    }
    return ant;
}
void PoliP :: Suprime (NodoPo* dir){
    if (dir == ptr_pol){
        ptr_poli = ptr_poli -> sig;
    } else {
        NodoPo* ante = ant
        ante -> sig = dir -> sig;
    }
    delete(dir);
}
void PoliP :: asignar_coeficiente (int coef, int exp){
    NodoPo* dir = poner_exponente (exp);
    if (dir){
        dir -> coef = coef
        if (coef == 0){
            Suprime (dir);
            nt--;
        }
    }
}
void PoliP :: poner_termino (int coef, int exp){
    NodoPo* dir = buscar_exponente (exp);
    if (!dir){
        if (coef != 0){
            NodoPo* nuevo = new NodoPo();
            nuevo -> coef = coef;
            nuevo -> exp = exp;
            nuevo -> sig = ptr_pol;
            ptr_poli = nuevo;
            nt++;
```

Materia:

Tema:

Fecha: __/__/__

```
    }
  } else {
  int new_coef = coef + dir -> exp;
      dir -> coef = new_coef;
      if (new_coef == 0){
          Suprime (dir);
          n--;
      }
    }
}
int Poli P :: numero_terminos() {
    return n;
}
int Poli P :: exponente (int nro_ter) {
    Nodo P* dir = buscar_termino.n(nro_ter);
    if (dir)
        return dir -> exp;
    else
        cout << " No existe ese termino" << endl;
}
String Poli P :: to_Str() {
    String r = "";
    int n = numero_terminos();
    for (int i = 1 ; i <= n ; i++) {
        int Coef = Coeficiente (exp);
        int Exp = exponente (i);
        if ( coef > 0)
            // r = r;
            r = r + "+";
            r += to_string (Coef) + "x^" + to_string (exp);
        }
        return r;
}
float Poli P :: evaluar ( float v) {
    float Sum = 0;
    for (int i=0; i < numero_terminos (); i++) {
        int exp = exponente ( i+1)
        int coef = Coficiente ( exp)
    // P -> poner_termino ( coef * exp, exp - 1);
        Sum += coef * pow (x, exp)
    }
    return sum;
}
```

Materia:

Tema:

Fecha: __/__/__

```
void PoliP::derivada (PoliP * p){
    for (int i=0; i < numero_termino (); i++){
        int exp = exponente (i+1);
        int coef = coeficiente (exp);
        p -> poner_termino (coef * exp, exp-1);
    }
}
PoliP * PoliP::suma (PoliP * a, PoliP * b){
    PoliP * p = new PoliP();
    for (int i=0; i < a -> numero_terminos (); i++){
        int exp = a -> exponente (i+1);
        int coef = a -> coeficiente (exp);
        p -> poner_termino (coef, exp);
    }
    for (int i=0; i < b -> numero_terminos (); i++){
        int exp = b -> exponente (i+1);
        int coef = b -> coeficiente (exp);
        b -> poner_termino (p, exp);
    }
    return p;
}
PoliP * PoliP::resta (PoliP * a, PoliP * b){
    PoliP * p = new PoliP();
    for (int i=0; i < a -> numero_terminos (); i++){
        int exp = a -> exponente (i+1);
        int coef = a -> coeficiente (exp);
        p -> poner_termino (coef, exp);
    }
    for (int i=0; i < b -> numero_terminos (); i++){
        int exp = b -> exponente (i+1);
        int coef = b -> coeficiente (exp);
        p -> poner_termino (coef, exp);
    }
    return p;
}
PoliP * PoliP::multiplicar (PoliP * a, PoliP * b){
    PoliP * p = new PoliP();
    for (int i=1; p <= b -> numero_terminos (); p++){
        int expB = b -> exponente (i)
        int coefB = b -> coeficiente (expB);
        for (int i=1; i <= a -> numero_terminos (); i++){
            int expA = a -> exponente (i);
            int coefA = a -> coeficiente (expA);
            int new_coef = coefA + coefB
        }
        return p;
}
```

tilibra

Materia:
Tema:
Fecha: __/__

PoliP.h

```
#ifndef PoliPH
#define PoliPH

#include <iostream>
#include <string>

using namespace std;
struct NodoPo {
        int coef;
        int exp;
        NodoPo* sig;
};
class PoliP {
 private:
      NodoPo* ptr_poli;
       int nt;

       NodoPo* buscar_exponente(int exp);
       NodoPo* buscar_termino (int i);
       NodoPo* anterior (NodoPo* dir);
       void supprimere (NodoPo dir);
 public:
      PoliP ();
      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);
        float evaluar (float x);
        void derivada (PoliP* p);
        PoliP* suma (PoliP*a, PoliP*b);
        PoliP* resta (PoliP*a, PoliP*b);
       PoliP* multiplicar (PoliP*a, PoliP*b);
       string to_str();

};
#endif
```

Materia:

Tema:

Fecha: __/__/__

PoliL.h

```
#ifndef PoliLH
#define PoliLH

#include <iostream>
#include <string>
#include "ListaL.h"

using namespace std;
class PoliL {
  private:
     Lista P *Pol;

     Nodo L * buscar_exponente (int exp);
     Nodo L * buscar_termino (int i);
  Public:
     PoliL();
     bool es_cero();
     int grado();
     void asignar_coeficiente (int coef, int exp);
     int coeficiente (int exp);
     void poner_termino (int coef, int exp);
     int numero_terminos();
     int exponente (int nroter);
     string to_str();

};
#endif
```

Materia: Fecha: / /

Tema:

PoliL.cpp

```
#pragma hdrstop
#include "PoliL.h"

PoliL::PoliL(){
    Pol = new ListaP();
}
NodoL* PoliL::buscar_exponente(int exp){
    NodoL* aux = pol->primero();
    while (aux != NULL){
        NodoL* aux = pol->primero();
        while (aux) = NULL
        if (pol->recupera(sig) == exp)
            return sig;
            aux = pol->siguiente(pol->siguiente(aux));
        }
    return NULL;
}
NodoL* PoliL::buscar_termino(int i){
    int c=1;
    NodoL* aux = pol->primero();
    while (aux != NULL){
        if (c==i)
            return aux;
        aux = pol->siguiente(pol->siguiente(aux));
        c++;
    }
    return NULO;
}
bool PoliL::es_cero(){
    return pol->vacia();
}
int PoliL::grado(){
    if (!es_cero()){
        NodoL* aux = pol->siguiente(pol->primero());
        int max = pol->recupera(aux);
        while (aux != NULL){
            if (pol->recupera(aux) > max)
                max = pol->recupera(aux);
            else
                aux = pol->siguiente(pol->siguiente(aux));
        }
        return max;
    } else
        cout << "Lista vacia\n";
}
```

tilibra

Materia:

Tema:

Fecha: __/__/__

```
int PoliL :: Coeficiente (int exp) {
        if (!es_cero) {
            NodoL * dir_exp = buscar_exponente (exp));
                if (dir_exp != NULL) {
                        NodoL * dir_coef = pol -> anterior (dir_exp);
                }
                else
                    cout << "No existe ese exponente \n";
        }
}

void public asignar coeficiente (int coef, int exp) {
        if (!es_cero ()) {
        // NodoL * dir_exp = buscar
            NodoL * dir_coef = buscar_exponente (exp));
            if (dir != NULL) {
                    NodoL * dir_coef = pol -> anterior (dir_exp);
                    pol -> predima (dir_coef, coef);
                    if (coef == 0) {
                            pol -> Suprime (dir_exp);
                            pol -> Suprime (dir_coef);
                    }
            }
        else
            cout << "no existe ese termino \n";
        }
}
void PoliL :: poner_termino (int coef, int exp) {
        NodoL * dir_exp = buscar_exponente (exp);
        if (dir_exp != NULL) {
            NodoL * dir_coef = pol -> anterior (dir_exp);
            int new_coef = pol -> recupera o dir_coef) + coef;
            pol -> modifica (dir_coef, new_coef);
            if (new_coef == 0) {
                    pol -> suprime (dir_exp);
                    pol -> Suprime (dir_coef);
            }
        } else {
            if (coef != 0) {
                    pol -> inserta_último (coef);
                    pol -> inserta_último (exp);
            }
        }
}
```

tilibra

Materia: Fecha: __/__/__

Tema:

```
int PoliL::numero_terminos(){
    return pol->longitud();
}

int PoliL::exponente(int nro_ter){
    if(!es_cero()){
        Nodo L* dir_coef = buscar_termino(nro_ter);
        if(dir_coef != NULL){
            Nodo L* dir_exp = pol->siguiente(dir_coef);
            return pol->recupera(dir_exp);
        }
        else
            cout<<"No existe termino\n";
    }
}

String PoliL::to_str(){
    String r = "";
    int n = numero_termino();
    for(int i=1; i<n; i++){
        int exp = exponente(i);
        int coef = coeficiente(exp);
        if(coef > 0)
            r += "+";
        r += to_string(coef) + "x^" + to_string(exp);
    }
    return r;
}
```

tilibra