UNIDAD III TDA Polinomio

2.1.1 Descripción del TDA Polinomio.

El prefijo **"poli"** significa muchos. Polinomio significa entonces 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 cinco términos en adelante.

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

$$P(x) = a_nx^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots\ldots\ldots\ldots + a_2x^2 + a_1x + a_0 :$$

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 dos 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.
- 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 siguiente:
Elementos que conforman la estructura

TDA Polinomio (**VALORES son coeficientes y grados del tipo Entero, OPERACIONES crea, escero, 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.

EsCero(P: Polinomio) devuelve (booleano)
Utilidad: Devuelve verdadero si P es un polinomio sin términos
Entrada: Polinomio P
Salida: Booleano
Precondición: Ninguna.
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 (Coeficiente de Termino)
Utilidad: Devuelve el Coeficiente que corresponde al termino con exponente Exp.
Entrada: Polinomio P
Salida: Numero
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 modificado en el valor del coeficiente de termino con grado exp.

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 de termino con grado exp.

numero_terminos(P:Polinomio) devuelve (Nro. Terminos)
Utilidad: Determinar el número de términos 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: 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 multiplicacion p1*p2 y la almacena en el polinomio P.
Entrada: Polinomio P1, p2
Salida: Ninguna
Precondición: Ninguna
Poscondición: Polinomio P tiene la multiplicacion de p1 y p2

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 esta implementado.

Ej1: Implementar un algoritmo que Busque un elemento en la lista L y retorne la Direccion donde se encuentra, caso contrario NULO

Ej1: Implementar un algoritmo que dado un 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_termino( co*ex, ex-1)
        fin
     fin
```

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

Derivada

$P1(x)=\mathbf{15x^2 + 3x^0}$

Ej2: 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
```

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

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

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

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

$\mathbf{(5x^3) + (3x^1) + C}$

https://www.youtube.com/watch?v=d7Y9Om4KCUM

2.1.4 Implementacion 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.

Definiendo la Polinomio

```
Tipo de Datos
        Clase Polinomio
          Atributos
                Pol : Lista
          Metodos
                Privado
                    Direccion BuscarExponente(Exp Entero)
                    Direccion BuscarTerminoN(I : Entero)
               Publico
                crea ()
                Booleano EsCero()
                Entero Grado()
                Entero coeficiente ( Exp : Entero)
                AsignarCoeficiente (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 definición clase
```

Direccion 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
                    // exception polinomio no tiene términos.
```

Fin

Direccion BuscarTerminoN(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
                    // exception polinomio no tiene términos.
```

fin

Polinomio.Crear

```
 inicio
    pol.crear // llamar al constructor de Pol.
 fin
```

https://youtu.be/XxYKHM2ds28

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
```

https://youtu.be/xUjQH0HKgM4

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
```

https://youtu.be/eHyVSMBY0Bg

polinomio.asignarcoeficiente (coef,exp : entero)

```
Inicio
   Dir = BuscarExponenente(exp)
   Si dir<> nulo entonces
                       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 término.
Fin
```

```
booleano polinomio.poner_termino  (coef,exp : entero)
Inicio
  DirExp = BuscarExponente(exp)
  si DirExp<>Nulo entices
                    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.insertaUltimo(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 numero de termino
Fin
```

https://youtu.be/K4GUDZZA99o

```
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
```

```
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//
 in
```

2.1.4.2 Implementación con Vector.

Para la implementación del TDA Polinomio se utilizara dos 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,                          // Coeficientes
                         VE  :    Arreglo(MAX)    // Exponentes
                          nt : Entero
                        Metodos
                    crea ()
                    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() 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://youtu.be/m-SN3xrfmxg

```
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 contraio
           // 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 posicion 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 la posicion lug
                  nt = nt -1
          caso contrario
            nt = nt +1
            vc[nt]=coef
            ve[nt]=exp
Fin
```
https://youtu.be/kVGCdp49EEcç

```
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/KJKJe1m4WAs

```
Entero polinomio.exponente ( nroter: entero) devuelve (Grado)
Inicio
        Retornar ve[nroter]
Fin
```
https://youtu.be/QT1WUlS4J50

```
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

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//
 in
```

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 busca es una mejor comprensión sobre los punteros, para ello se entiende que se usara 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 implementacion)
fin
Direccion  Puntero a Nodo (valor entero para esta implementación)

Clase Polinomio
    Atributos
        Ptr_Poli  Direccion
        Nt        Entero
    Metodos
      Privado
            Direccion BuscarExponente(Exp Entero)
            Direccion BuscarTerminoN(I : Entero)

        publico
    crea ()
    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() devuelve (Nro. Terminos)
    exponente ( nroter: entero) devuelve (Grado)
    sumar(P1, P2: Polinomio)
    restar(P1, P2: Polinomio)
    multiplicar (P1, P2: Polinomio)
fin definición clase
```

Implementación clase polinomio utilizando Simulador de Memoria CSmemoria.

```
Direccion polinomio.BuscarExponente(Exp Entero)
Inicio
  Dir = ptr_Poli
  If Dir <> nulo entices
            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
```

```
Direccion 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=M.obtener_dato(dir,'->sig')
                        Fin mientras
                       Retornar dirTer
                     Caso contrario
                              // exception no existe terminos
fin

Polinomio.Crear()
   inicio
      nt = 0
      ptrpoli=nulo
   fin
```
https://youtu.be/YVqDRyr9M70

```
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
                              // exception no existe ese termino
 Fin
```
https://youtu.be/L_Fr3W2pptI

```
Entero polinomio.coeficiente ( Exp : Entero) devuelve (Coeficiente de Termino)
Inicio
        Dir = buscarExponente(exp)
        Si dir<>nulo entonces
                          Retornar m.obtener_dato(dir,'->coef')
                       Caso contrario
                         // exception no existe ese termino
Fin
```
https://youtu.be/cC6CBvWTzoU

```
AsignarCoeficiente (coef,exp: Entero)
Inicio
    Dir = buscarExponente(exp)
    Si dir <> nulo entonces
                        m.poner_dato(dir,'->coef',coef)
                       si coef=0 entonces
                                  // eliminar nodo Dir

                   Caso contrario
                     // no existe ese termino
Fin


polinomio.poner_termino( coef, exp : entero)
inicio
      existe = buscarExponente(exp)
      si existe = nulo
           entonces
           aux = M.New_Espacio('coef,exp,sig')
           si aux <> nulo entonces
                        m.ponerdato(aux,'->coef',coef)
                        m.ponerdato(aux,'->exp',exp)
                        m.ponerdato(aux,'->sig',prt_poli)
                        Ptr_Poli = Aux
                        nt =nt +1
                   caso contrario
                         //error espacio memoria
         caso contrario
               NuevoCoef = m.obtenerdato(existe,'->coef') + Coef
               m.ponerdato(existe,'->coef', NuevoCoef)
               // Eliminar nodo si NuevoCoef es 0
    fin
https://youtu.be/rFR_w9e9h88


entero polinomio.numero_terminos() devuelve (Nro. Terminos)
inicio
   retornar nt
fin


entero polinomio.exponente ( nroter: entero) devuelve (Grado)
inicio
   dir = buscarterminoN (nroter)
   di dir <> nulo entonces
                        retornar m.obtenerdato(dir,'->exp')
                  caso contrario
                        // no existe ese termino
fin
https://youtu.be/aItWkW9ZXTc
```

```
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

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//
 fin
```

2.1.4.4 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 considerando que ahora se está trabajando con punteros, es así que se tiene de color rojo los cambios fundamentales en los algoritmos ya vistos quedando por resolver las definiciones formales en c++.

Definiendo la clase Polinomio

Tipo de dato

```
        Nodo
              Coef  Entero
              Exp    Entero
              sig     Puntero a Nodo
        fin
```

Dirección Puntero a Nodo

```
Clase Polinomio
        Atributos
            Ptr_Poli  Dirección
             Nt        Entero
         Metodos
           Privado
              Direccion BuscarExponente(Exp Entero)
              Direccion BuscarTerminoN(I : Entero)
           publico
              crea ()
              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() devuelve (Nro. Terminos)
              exponente ( nroter: entero) devuelve (Grado)
              sumar(P1, P2: Polinomio)
              restar(P1, P2: Polinomio)
              multiplicar (P1, P2: Polinomio)
     fin definición clase
```

Implementación clase polinomio utilizando Simulador de Memoria CSmemoria.

```
Direccion polinomio.BuscarExponente(Exp Entero)
Inicio
  Dir = ptr_Poli
  If Dir <> nulo entices
                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_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
```

```
Polinomio.Crear()
   inicio
      nt = 0
      ptr_poli=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 entices
                        MaxG=dir->exp
                        Mientras (Dir<> nulo)
                             Si  dir->exp > MaxG entonces
                                        MaxG= dir->exp
                             dir=dir->sig
                       Fin mientras
                       Retornar MaxG
                    Caso contrario
                           // exception no existe ese termino
 Fin
Entero polinomio.coeficiente ( Exp : Entero) devuelve (Coeficiente de Termino)
Inicio
        Dir = buscarExponente(exp)
        Si dir<>nulo entonces
                          Retornar dir->coef
                        Caso contrario
                          // exception no existe ese termino
Fin

AsignarCoeficiente (coef,exp: Entero)
Inicio
    Dir = buscarExponente(exp)
    Si dir <> nulo entonces
                          dir->coef=coef
                          si coef=0 entonces
                                    // eliminar nodo Dir

                       Caso contrario
                         // no existe ese termino
fin


polinomio.exponente ( nroter: entero) devuelve (Grado)
inicio
   dir = buscarterminoN (nroter)
   di dir <> nulo entonces
                          retornar dir->exp
                    caso contrario
                          // no existe ese termino
fin
```

```
polinomio.poner_termino( coef, exp : entero)
inicio
      existe = buscarExponente(exp)
      si existe = nulo
           entonces
           aux = New Nodo
           si aux <> nulo entonces
                              aux->coef=coef
                              aux->exp=exp
                              aux->sig=prt_poli
                              Ptr_Poli = Aux
                              nt =nt +1
                        caso contrario
                                 //error espacio memoria
          caso contrario
                 NuevoCoef = existe->coef + Coef
                 existe->coef= NuevoCoef
                 // Eliminar nodo si NuevoCoef 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
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.numero_terminos() devuelve (Nro. Terminos)
inicio
retornar nt
fin

polinomio.multiplicacion(p1, p2 : polinomio)
Inicio
//Desarrolle el algoritmo//
in