

PRÁCTICA 6: Operaciones con polinomios.

OBJETIVOS: Repaso de listas dinámicas. Objetos como datos miembro de otros objetos (introducción a la herencia). Objetos con partes dinámicas. Sobrecarga de operadores.

TEMPORIZACIÓN:

Publicación del enunciado: Semana del 8 de noviembre.

Entrega: Semana del 29 de noviembre.

Límite de entrega (con penalización): Semana del 20 de diciembre.

BIBLIOGRAFÍA

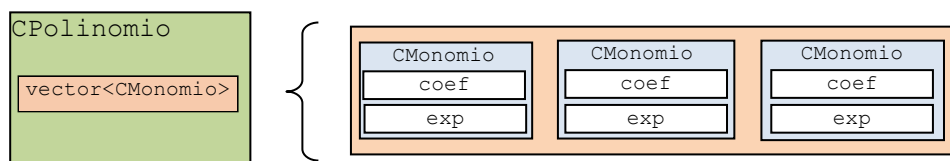
Programación orientada a objetos con C++

C/C++. Curso de programación, 5ª edición, capítulo: Estructuras dinámicas

Autor: Fco. Javier Ceballos

Editorial: RA-MA.

Se trata de escribir un programa que permita realizar operaciones con polinomios, donde un polinomio quedará definido por una lista lineal simplemente enlazada como la mostrada en la siguiente figura:



Se deben completar las clases `CMonomio` y `CPolinomio`, cada una en su(s) propio(s) fichero(s), para realizar un programa que opere con polinomios y ponga a prueba todas las funciones implementadas. Se recomienda realizar la práctica siguiendo los pasos indicados al final del enunciado (NOTAS); destacamos especialmente que se debe implementar cuanto antes el operador de inserción de monomios, `CPolinomio& CPolinomio::operator<<(const CMonomio& mono)`, ya que facilitará enormemente la implementación de otros métodos.

La clase `CMonomio` encapsula un término de un polinomio. Un monomio con coeficiente 3 y exponente 2 equivale a $3x^2$. Todos los métodos de la clase son muy sencillos, así que pueden ser definidos `inline`. Esto quiere decir que el compilador podrá sustituir las llamadas a los métodos por el propio código de éstos.

```
class CMonomio
{
private:
    double m_dCoeficiente;
    int m_nExponente;

public:
    CMonomio(double dCoef=0, int nExp=0)
```

```
        : m_dCoeficiente(dCoef), m_nExponente(nExp) {}  
double GetCoef() const;           // <-- definir aquí  
int GetExp() const;              // <-- definir aquí  
  
void SetCoef(double dCoef);      // <-- definir aquí  
void SetExp(int nExp);           // <-- definir aquí  
  
CMonomio operator-() const  
{ return CMonomio(-m_dCoeficiente, m_nExponente); }  
};  
  
ostream& operator<<(ostream& os, const CMonomio& Mono);
```

El constructor recibe dos parámetros con valores por defecto. Su lista de iniciadores copia los valores recibidos en los datos miembro, o más bien *construye* los datos miembro a partir de los valores recibidos. El cuerpo del constructor no hace nada (llaves vacías).

Se ha sobrecargado el operador *menos unario*, que devuelve un monomio cambiado de signo. Esto permitirá hacer cosas como:

```
CMonomio a(3,2), b;           // a == 3x^2  
b = -a;                       // b == -3x^2
```

No hace falta definir un constructor copia ni sobrecargar el operador de asignación porque los objetos de esta clase no tienen partes dinámicas. La copia que se hará por defecto (miembro a miembro) es perfectamente válida.

Falta definir los métodos `Get...` y `Set...`; Serán métodos `inline`.

Para mostrar un monomio se puede sobrecargar el operador de inserción como se indica a continuación:

```
ostream& operator<<(ostream& os, const CMonomio& mono)  
{  
    os << showpos;                // Poner + para valores positivos  
    if (!mono.GetExp())           // Si la x está elevada a 0,  
        os << mono.GetCoef();    // vale 1 y se muestra sólo el coef.  
    else                          // Si no...  
    {  
        if (mono.GetCoef() != 1) // Mostrar el coeficiente  
            os << mono.GetCoef(); // sólo si es distinto de 1  
        os << noshowpos;         // quitar + para valores positivos  
        if (mono.GetExp() == 1)  // Si el exponente es 1,  
            os << 'x';           // basta "x"  
        else                    // Si no,  
            os << "x^" << mono.GetExp(); // hay que poner "x^..."  
    }  
    os << noshowpos;  
    return os;                  // Devolver ref. al ostream recibido  
}                               // para permitir encadenamiento: cout << m1 << m2...
```

Escribir esta función en el fichero `.cpp` correspondiente.

La clase `CPolinomio` implementa la lista enlazada a partir de un vector de monomios como se indica a continuación:

```
class CPolinomio
{
    private:
        std::vector<CMonomio> m_monomios;
        // Primer término (el de mayor grado)

    public:
        // ...
};

ostream& operator<<(ostream& os, const CPolinomio& Poli);
```

Para esta clase se deberá definir:

- 1) Un constructor sin parámetros que inicie el vector `m_monomios` a cero elementos.
- 2) Un constructor copia que copie un polinomio. Su parámetro debe ser una referencia a un objeto `const`. Implementar el constructor copia de forma que realice la misma iniciación del constructor sin parámetros, seguida de una llamada al operador de asignación indicado en el punto 7.
- 3) Un constructor que reciba un coeficiente y un exponente para construir un polinomio con un sólo monomio. El exponente deberá tener el valor 0 por defecto. Si el coeficiente recibido es 0, la lista deberá dejarse vacía (`m_monomios` con 0 elementos).
- 4) Un constructor que reciba un `vector<CMonomio>` y construya un polinomio formado por los monomios contenidos en dicho vector. Para ello puede utilizar la sobrecarga del operador de inserción especificado un poco más adelante.
- 5) Un constructor que reciba un monomio para construir un polinomio con un sólo término. Si el coeficiente del monomio recibido es 0, la lista deberá dejarse vacía (`m_monomios` con 0 elementos).
- 6) Valore si fuera necesario incluir un destructor.
- 7) Un operador de asignación que copie un polinomio (utilice la sobrecarga del operador de inserción especificado a continuación). Tener en cuenta que el objeto destino puede contener datos anteriores.

El operador de inserción indicado en la página anterior, que sirve para enviar un polinomio a un flujo, llamará a la función `MostrarPoli` siguiente, que, a su vez, llama al operador de inserción de `CMonomio`:

```
void CPolinomio::MostrarPoli(ostream & os) const
{
    int pos = 0;
```

```
if(m_monomios.size())
{
    do
    {
        os << m_monomios[pos++] << ' ';
    }while(pos < m_monomios.size());
}
else
    os << "0 ";
}
```

La introducción de datos en un polinomio se hará mediante una nueva sobrecarga del operador de inserción:

```
CPolinomio& CPolinomio::operator<<(const CMonomio& mono);
```

Al definirlo habrá que tener en cuenta que:

- 1) El monomio recibido puede tener coeficiente 0. En ese caso no habrá que añadirlo al polinomio.
- 2) Si la lista está vacía, bastará añadir un nuevo término.
- 3) Si no existe ningún término con el exponente del nuevo monomio, bastará insertar un nuevo término en la posición adecuada. Habrá que respetar el orden (de mayor a menor según el exponente).
- 4) Si ya existe un término con el exponente del nuevo monomio, habrá que sumar a su coeficiente el coeficiente del nuevo monomio (porque no puede haber exponentes repetidos). Si el resultado es 0 (los coeficientes se anulan), habrá que retirar el término de la lista.

Para poder realizar la inserción del nuevo término habrá que recorrer la lista buscando un término con exponente menor o igual al del monomio que hay que insertar.

```
CPolinomio& CPolinomio::operator<<(const CMonomio& mono)
{
    // si el coeficiente no es cero, se añade al polinomio; si es cero no se añade.
    // si la lista está vacía, nuevo término al principio
    // si la lista no está vacía...
    // buscar un exponente igual o menor que el de "mono"
    // si el exponente es igual que el de "mono"
    // suma los coeficientes
    // si suman cero
    // borrar monomio de la lista
    // devolver el polinomio
    // si el exponente es menor que el de "mono"
    // crear un nuevo monomio
    // e incluirlo correctamente
    // devolver el polinomio
    // si el exponente es mayor que el de "mono"
    // seguir buscando
    // si llegamos al final del vector
```

```
    // crear nuevo monomio e incluirlo al final
    // devolver el polinomio
}
```

Este operador de inserción debe ser puesto a prueba con el siguiente código:

```
CPolinomio P, Q, R, S, T, U, V, W, X, Y, Z, N;
CMonomio m1, m2(2,2), m3(3,3), m4(4,4);

P << m1 << m2 << -m4 << m3;
Q << m4 << m1 << m2 << -m3;
R << m3 << m4 << -m2;
S << m2 << m3 << m4 << -m2;
T << -m2 << -m3 << -m4 << m3;
U << m2 << m3 << m4 << -m4;
V << m2 << m3 << m4 << -m2 << -m3 << -m4;
W << m2 << m3 << -m2 << -m3;
X << m2 << -m2;
Y << m2 << m3 << m4 << m2;
Z << -m2 << -m3 << -m4 << -m3;
N << m2 << m3 << m4 << m4;

cout << "P = " << P << endl
      << "Q = " << Q << endl
      << "R = " << R << endl
      << "S = " << S << endl
      << "T = " << T << endl
      << "U = " << U << endl
      << "V = " << V << endl
      << "W = " << W << endl
      << "X = " << X << endl
      << "Y = " << Y << endl
      << "Z = " << Z << endl
      << "N = " << N << endl;
```

Véase el resultado que se obtiene al final del enunciado.

Esta sobrecarga de << simplificará la implementación de **operator=**, **operator+**, etc.

Se deberá definir una función miembro pública de la clase **CPolinomio** que permita ejecutar una sentencia como la siguiente. Esta función devolverá el mayor exponente de todos los términos del polinomio.

```
int grado = U; // mayor exponente de U
```

Se deberán sobrecargar además los siguientes operadores y otros que puede ver en el resultado mostrado al final del enunciado. La implementación de alguno de estos métodos se podrá realizar en función de otros que ya estén implementados; por ejemplo, el "<" en función de la función miembro del apartado anterior, el "-" binario en función del "+" binario y "-" unario, etc.

```
<, > y ==
- unario
+, -
+==, -==
```

[] y ()

Todos ellos deberán ser definidos como miembros de la clase `CPolinomio`. Los operadores de comparación se limitarán a comparar el grado de los polinomios. Los operadores `+=` y `-=` pueden definirse fácilmente utilizando los demás (`+`, `-`, `y =`).

El operador `[]` recibirá un número de exponente (`int`) y devolverá el coeficiente (`double`) del término que tenga ese exponente. Si ningún término del polinomio tiene ese exponente, devolverá 0.

El operador `()` recibirá un valor de la `x` del polinomio (`double`) y devolverá el valor del polinomio (`double`) para ese valor de `x`. Este operador servirá para convertir a los polinomios en *funciones matemáticas*. Por ejemplo:

```
CPolinomio P;  
P << CMonomio(3,2) << CMonomio(-2,1);    // P == 3x^2 -2x  
double y, x=0.5;  
y = P(x);                                // y = 3*0.5^2 - 2*0.5 = -0.25
```

NOTAS:

Se sugiere el uso de la función `pow`.

Se recomienda realizar la práctica siguiendo los siguientes pasos:

- Implemente los constructores y destructores.
- Escriba una función `main` para probarlos.
- Implemente los operadores `<<`.
- Añada a `main` el código necesario para probarlos.
- Implemente la función `Grado`.
- Añada a `main` el código necesario para probarlos.
- Implemente el resto de operadores.
- Añada a `main` el código necesario para probarlos.

El objetivo de este desarrollo escalonado es compilar y ejecutar en cada paso para descubrir los errores lo antes posible.

Un código como el siguiente:

```
if (x > y)  
    return true;  
else  
    return false;
```

se puede escribir de forma abreviada así:

```
return x > y;
```

RESULTADOS DESPUÉS DE EJECUTAR LA PRÁCTICA:

***** PRACTICA 6 DE PROGRAMACION AVANZADA *****

Construcción de Polinomios

Probando el operador de inserción de monomios (`<<`)
`P = -4x^4 +3x^3 +2x^2`

```
Q = +4x^4 -3x^3 +2x^2
R = +4x^4 +3x^3 -2x^2
S = +4x^4 +3x^3
T = -4x^4 -2x^2
U = +3x^3 +2x^2
V = 0
W = 0
X = 0
Y = +4x^4 +3x^3 +4x^2
Z = -4x^4 -6x^3 -2x^2
N = +8x^4 +3x^3 +2x^2
```

```
CPolinomio A = P (constructor copia)
A = -4x^4 +3x^3 +2x^2
Presione una tecla para continuar . . .
```

```
B = Q (operador de asignación)
B = +4x^4 -3x^3 +2x^2
Presione una tecla para continuar . . .
```

```
P = P (evitar auto-asignación)
P: -4x^4 +3x^3 +2x^2
Presione una tecla para continuar . . .
```

```
Probando el constructor con un coef. y un exp.
C = +2.5
D = +2.5x^5
Presione una tecla para continuar . . .
```

```
Probando el constructor con un monomio
E = +4x^4
Presione una tecla para continuar . . .
```

```
Obteniendo el grado del polinomio.
El grado de U es: 3
Presione una tecla para continuar . . .
```

```
Probando el constructor con vector<CMonomio>
F = +4x^4 +3x^3 +2x^2
Presione una tecla para continuar . . .
```

```
Probando el operador >
P = -4x^4 +3x^3 +2x^2
U = +3x^3 +2x^2
P es de mayor grado que U
Presione una tecla para continuar . . .
```

```
Probando el operador ==
P = -4x^4 +3x^3 +2x^2
Q = +4x^4 -3x^3 +2x^2
P es de igual grado que Q
Presione una tecla para continuar . . .
```

```
Probando el operador - unario
Q = +4x^4 -3x^3 +2x^2
-Q = -4x^4 +3x^3 -2x^2
Presione una tecla para continuar . . .
```

```
Probando el operador +
N = +8x^4 +3x^3 +2x^2
U = +3x^3 +2x^2
N + U: +8x^4 +6x^3 +4x^2
Presione una tecla para continuar . . .
```

```
Probando los operadores - y +=
R = +4x^4 +3x^3 -2x^2
```

```
S = +4x^4 +3x^3
R - S: -2x^2
R += S: +8x^4 +6x^3 -2x^2
Presione una tecla para continuar . . .
```

```
Probando el operador -=
N = +8x^4 +3x^3 +2x^2
Y = +4x^4 +3x^3 +4x^2
N -= Y: +4x^4 -2x^2
Presione una tecla para continuar . . .
```

```
Probando el operador []
Polinomio P: -4x^4 +3x^3 +2x^2
Coeficiente de P[4]: -4
Presione una tecla para continuar . . .
```

```
Probando el operador ()
Polinomio G: +3x^2 -2x , G(0.5) = -0.25
Presione una tecla para continuar . . .
```

```
Probando el operador *
Q = +4x^4 -3x^3 +2x^2
S = +4x^4 +3x^3
Q * S: +16x^8 -1x^6 +6x^5
Presione una tecla para continuar . . .
```

```
Probando el operador *=
P = -4x^4 +3x^3 +2x^2
Q = +4x^4 -3x^3 +2x^2
P *= Q: -16x^8 +24x^7 -9x^6 +4x^4
Presione una tecla para continuar . . .
```

Identificar los métodos que no sean necesarios. Por ejemplo, si en la clase `CPolinomio` no fuera necesario, el constructor copia o cualquier otro método, indíquelo como comentario en la línea anterior a la cabecera de la definición del método, justificando el motivo por el que se ha obtenido esa conclusión.

Operaciones básicas (pos: posición de un elemento en el vector):

Borrar:	<code>m_monomios.erase(m_monomios.begin() + pos);</code>
Insertar:	<code>m_monomios.insert(m_monomios.begin() + pos, mono);</code>
Añadir al final:	<code>m_monomios.push_back(mono);</code>