



## Enunciado de la Práctica

### OBJETIVO:

Implementar en C++ un nuevo tipo de datos que modele polinomios de cualquier grado.

### PROCEDIMIENTO:

#### Clase Monomio

Diseñar una clase Monomio que modele monomios por medio de su coeficiente y grado. Implementar los siguientes métodos:

- Método constructor sin argumentos que establece los valores coeficiente y grado a cero.
- Método constructor que establece los valores coeficiente y grado a los valores pasados por parámetro. Un monomio de coeficiente cero y grado  $g$ , es un monomio de coeficiente cero y grado cero.
- Método destructor.
- Métodos mutadores y accedentes (a los atributos).
- Leer (): leer por pantalla un monomio.
- Visualizar () : permite visualizar un monomio por pantalla.
- Métodos operadores que permitan realizar las siguientes operaciones:

```
Monomio m1, m2;  
float x;  
cout << "Introducir los datos del monomio 1" << endl;  
m1.leer();  
cout << "Introducir un número" << endl;  
cin >> x;  
m2 = m1 * x;  
m2 = x * m1;  
m1 *= x;
```

## Clase Polinomio

Haciendo uso del tipo de la clase anterior, diseñar la clase Polinomio. Los objetos de ésta clase disponen de los atributos: Grado del polinomio y array de monomios (memoria dinámica).

Los métodos a tener en cuenta son:

- Método constructor por defecto que permite representar el polinomio vacío.
- Método destructor.
- Métodos operadores necesarios para poder ejecutar las siguientes instrucciones:

```
Polinomio p1, p2, p3, p4;
Monomio m1;
float x;
cout << "Introducir los datos del polinomio 1" << endl;
cin >> p1;
p4 = p1;           // operación de asignación
cout << "Introducir los datos del polinomio 2" << endl;
cin >> p2;
p3 = p1 - p2;     //resta de polinomios
p3 = p1 + p2;     //suma de polinomios
cout << "La suma de los polinomios es: " ;
cout << p3;
p1 += p2;
p1 += x;         // suma de un polinomio y un real
cin >> x;
cout << "El valor del polinomio en el punto" << x << "es: ";
cout << p3(x);
p3 = p1 * x;     //producto de un polinomio por un real
p3 = x * p1;     //producto de un polinomio por un real
```

## Notas:

- El constructor sin argumentos debe inicializar con el valor NULL el puntero al vector de monomios. Con ellos se representa el polinomio vacío. Todos los métodos han de tener en cuenta la posibilidad de que alguno de los argumentos sea el polinomio vacío.
- Se ha de implementar el operador de asignación. Ojo, la copia de objetos de la clase Polinomio no se hace Bit a Bit, ya que uno de los atributos es un puntero.
- Se ha de implementar el destructor. Ojo, uno de los atributos es un puntero. ¿Qué tiene que hacer el destructor?
- Al hacer  $p1 = p2$  para asignar  $p2$  a  $p1$ , debe liberarse la memoria ocupada por los monomios del polinomio que pusiese contener  $p1$  antes de la asignación (excepto en el caso de que  $p1$  y  $p2$  sean del mismo grado: en ese caso puede reutilizarse).
- Al sumar dos polinomios del mismo grado, el resultado puede ser de grado menor. Incluir en las pruebas casos en los que el resultado de la suma sea del mismo grado, casos en los que sea de grado cero y casos en los que resulte el polinomio nulo

- $P(x) = 0$ . Realizar pruebas en los que algunos de los argumentos de la suma, o ambos, sean el polinomio vacío.
- El operador `()` se utiliza para evaluar un polinomio en un punto (que se pasará como argumento). Si es el polinomio vacío, devuelve 0 como resultado. (Incluye ese caso en las pruebas).
  - El operador `<<` se utiliza para obtener en un flujo de salida una representación del polinomio, formada por su grado y sus coeficientes entre llaves y separados por comas. El grado se encuentra en la primera posición separados de los coeficientes por punto y coma:
 
$$\{\text{grado\_n}; \text{coef\_0}, \text{coef\_1}, \dots, \text{coef\_n}\}$$
 Por ejemplo, el polinomio  $P(x) = 4x^2 + 3x - 6$  se representará en el flujo de salida de la siguiente forma: `{2; -6, 3, 4 }`.  
 Si es un polinomio vacío, se representará `{0;}`.
  - El operador `>>` se utiliza para leer de un flujo de entrada un polinomio dado en la representación anterior. Al igual que el operador de asignación, debes tener en cuenta que al hacer `cin>>p1` podría ser necesario liberar memoria. No consideres posibles errores de entrada, puedes suponer que los datos estarán en el flujo de entrada en el formato correcto.
  - Los flujos no tienen por qué ser `cin` y `cout`. Haz las pruebas necesarias para que los operadores `>>` y `<<` funcionen con cualquier flujo de entrada y salida.

## CONDICIONES DE ENTREGA

- 1) La práctica podrá ser realizada en grupos de **dos** alumnos.
- 2) La fecha límite de entrega: 9 de Mayo de 2004 a la hora de clase.
- 3) La práctica se debe codificar en Borland C++ X.
- 4) Se entregará un disco de 3<sub>1/2</sub> en cuya etiqueta debe aparecer el nombre de cada uno de los miembros que realizan la práctica. El contenido es el siguiente:
  - El **proyecto** que engloba todos los módulos del programa (archivos `.h` de cabecera y `.cpp` de implementación y archivo `.cpp` con el programa principal).
  - Un BREVE documento `.doc` o memoria de la práctica con los siguientes apartados (como mínimo):
    - Índice de contenidos.
    - Autores.
    - Breve descripción de las clases y los métodos implementados. Test de pruebas y resultados significativos (es decir con los casos más complicados y difíciles incluidos).
- 5) Una vez entregada la práctica, los alumnos deberán defenderla cuando se les convoque.

## **CRITERIOS DE EVALUACIÓN**

- 1) La práctica puntúa el 20% de la nota final de la asignatura.
- 2) Para poder evaluar la práctica, deberá compilar correctamente.
- 3) La memoria es tan importante como el programa, si no se alcanza un mínimo admisible, la práctica estará suspensa.
- 4) La nota final de la práctica se divide en 2 partes: buen funcionamiento del programa y la defensa de la práctica.
- 5) La defensa de la práctica se hará en parejas. Dicha defensa consiste en una serie de preguntas, modificaciones al programa etc. El objetivo es comprobar que los dos miembros del equipo han trabajado por igual.

## **RECOMENDACIONES**

Desarrollar el programa por etapas, e ir añadiendo operadores de forma progresiva y comprobando que cada operador funciona correctamente antes de pasar al siguiente. Compilar y probar pequeñas funcionalidades para corregir los errores, no esperar al final.