Industrial Data
ISSN: 1560-9146
iifi@unmsm.edu.pe
Universidad Nacional Mayor de San Marcos
Perú

Ruiz Lizama, Edgar
Un tipo abstracto de datos polinomio en C++
Industrial Data, vol. 7, núm. 2, julio-diciembre, 2004, pp. 46-51
Universidad Nacional Mayor de San Marcos
Lima, Perú

Revista de la Facultad de Ingeniería Industrial
Vol. (7) 2: pp. 46-51 (2004) UNMSM
ISSN: 1560-9146 (impreso) / ISSN: 1810-9993 (electrónico)

UN TIPO ABSTRACTO DE DATOS POLINOMIO EN C++

Recepción: Noviembre de 2004 / Aceptación: Diciembre 2004

(1) Edgar Ruiz Lizama

RESUMEN

El artículo presenta un programa en C++ que define una clase de nombre polinomio para implementar un tipo abstracto de datos (TAD), polinomio mediante una clase que realiza aritmética de polinomios con operaciones como suma, producto de una constante por un polinomio, producto de monomio por polinomio y la derivada de un polinomio. Se presenta el código del programa y se discuten sus detalles, bondades y limitaciones. La implementación se realiza en el compilador Dev C++ 4.1; un compilador GNU con licencia GPL.

Palabras Claves: Implementación de una clase polinomio. Derivada de un polinomio. Constante por polinomio.

A POLYNOMIAL IN C++ DATA ABSTRACT TYPE ABSTRACT

This article presents a program in C++ which defines a class of polynomial name in order to implement a data abstract type (TAD), a polynomial through a class that performs polynomials arithmetic with operations such as addition, the product of a constant by a polynomial, the product of a monomial by a polynomial and a polynomial derived. The program code is presented, and its details, features and limitations are discussed. Implementation is done in the Dev C++ 4.1 compilator; a GNU compilator with GPL licence.

Key words: Implementation of a polynomial class. Polynomial derived. Constant.

(1) Ingeniero Industrial. Profesor del Departamento de Ingeniería de Sistemas e Informática, UNMSM.
E-mail: eruizl@unmsm.edu.pe

INTRODUCCIÓN

Los polinomios y sus aplicaciones son muy importantes en la ciencia e ingeniería. Los métodos numéricos requieren que los algoritmos proporcionen variedad de esquemas aplicados al modelo matemático del problema. Existen productos de software como Matemática® y Matlab® que permiten trabajar con polinomios, pero no poseen la variedad y flexibilidad que el investigador pueda requerir para alguna aplicación particular. Surge entonces la inquietud ¿En un lenguaje como C++, se puede implementar la aritmética de polinomios? La respuesta es planteada en este artículo.

FUNDAMENTOS MATEMÁTICOS

Expresión Algebraica

Una expresión algebraica es escogida de un cuerpo numérico, que generalmente, es el de los números reales o el de los números complejos.

Polinomio

Un polinomio es una expresión algebraica formada por la suma de un número finito de términos, cada uno de los cuales es el producto de un conjunto finito de números que son los coeficientes seguido de la(s) variable(s) y un exponente. Un término puede incluir también un signo negativo correspondiente al número -1 contenido en este producto. Usualmente el exponente es un entero positivo, aunque puede ser un entero negativo o incluso una fracción.

Ejemplos de polinomios:

a. Un polinomio: $3x^4 + 1.5x^3 + 2x^2 - 5x + 1.7$

b. Un monomio: $2.5x^3$

IMPLEMENTACIÓN DEL TAD POLINOMIO

Un polinomio como tal, no se encuentra definido en un lenguaje de computación, por ello se requiere crear un TAD mediante una clase a la cual se denomina polinomio.

La implementación del TAD polinomio se basa en el concepto de lista enlazada. Una lista enlazada; es una secuencia finita de elementos donde cada elemento o nodo, conceptualmente está compuesto por dos campos: el campo dato y el campo puntero o enlace al siguiente elemento.

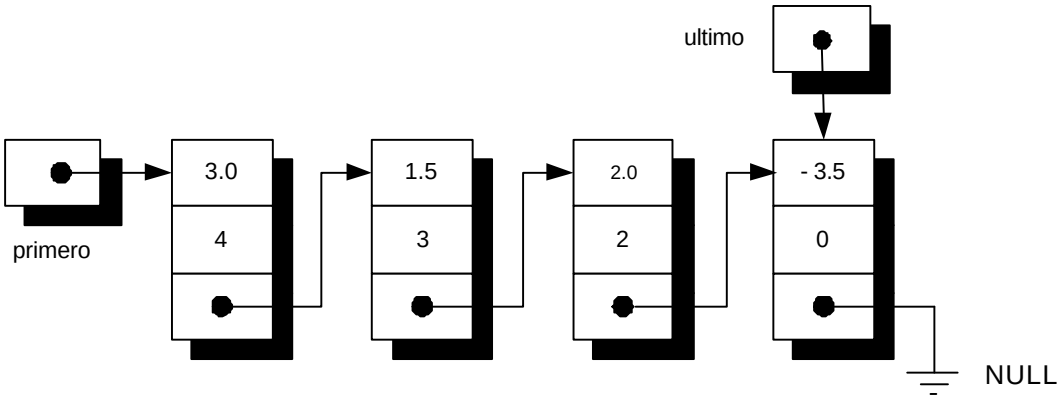

Figura 1. Polinomio representado como una lista enlazada

Para acceder a la información en una lista, es conveniente establecer un puntero al primer nodo. C++ permite la implementación de estructuras dinámicas de datos mediante la asignación dinámica de memoria, esto significa que los nodos se van creando y/o destruyendo conforme se requieren.

Un polinomio puede representarse, mediante una lista enlazada, donde cada nodo representa un término y está compuesto por tres campos: el primero es el coeficiente, el segundo el exponente y el tercero un puntero al siguiente nodo. Adicionalmente, para gestionar la lista se requiere un puntero al primer nodo y otro puntero al último nodo (veáse la Figura 1).

Cada nodo o término es concebido como una estructura (struct en C++), de allí que la implementación de la clase TAD polinomio tendrá como dato un nodo definido como un tipo POLY y permitirá gestionar objetos polinomio. Las declaraciones de la estructura y de la clase se presentan en la Figura 2.

Además de datos miembro; toda clase requiere de funciones miembro o métodos que se encargan de realizar el trabajo del TAD que se implementa. Las funciones miembro *creaNodoPoly(co, ex)* y *creaPoly()*; permiten crear un nodo y un polinomio respectivamente. El listado de ambas funciones se muestra en la Figura 3.

```
struct nodoPoly{ // un nodo es un termino del polinomio
   float coef; // coeficiente
   int exp; // exponente
  struct nodoPoly *sig; // es recursiva pues se referencia asi misma
};
typedef struct nodoPoly *POLY;

class polinomio{
public:
    void creaPoly(); // crear un polinomio
   void creaNodoPoly(float, int); // crea un termino del polinomio
  polinomio const_mult_poly(float); // constante por polinomio
   void printPoly(); // imprimir terminos del polinomio
private:
    POLY primero; //puntero al primer termino del polinomio
   POLY ultimo;  //puntero al ultimo termino del polinomio
};
```

Figura 2. Declaraciones de un nodo y de la clase polinomio

```
//definicion de funciones miembro
void polinomio :: creaNodoPoly(float co, int ex)
{
    POLY t; //nuevo termino;
   t = new nodoPoly; // asignacion dinamica
    //almacenar datos
    t->coef = co;
     t->exp = ex;
     t->sig = NULL;
     if (ultimo == NULL)
        primero = t;
    else
       ultimo->sig = t;
   ultimo = t;
}
//crear un polinomio
void polinomio :: creaPoly()
{
    float co;
    int ex, n;

     ultimo = NULL;
   cout<<"Ingrese numero de terminos del polinomio: ";
    cin>>n;
    for (int i = 0; i<n; i++)
   {   cout<<"\nIngrese coeficiente: ";  cin>>co;
       cout<<"Ingrese exponente: ";    cin>>ex;
      // crear nuevo nodo del polinomio
        creaNodoPoly(co, ex);
    }
}
```

Figura 3. Listado de las funciones miembro creaPoly() y creaNodoPoly(co, ex)

```
void polinomio :: printPoly()
{
    POLY actual;
   actual = primero;
   cout<<"\nP(x) = ";
    while( actual != NULL)
   {   if (actual -> exp == 0) // ultimo termino
          cout<<actual->coef;
      else  // otro termino
         cout<<actual->coef<<"X^"<<actual->exp<<" + ";
       actual = actual->sig;
    }
   cout<<endl;
}
```

Figura 4. Listado de las funciones miembro printPoly()

```
polinomio polinomio :: const_mult_poly( float con )
{
   float c;
    POLY p;
   p = primero;
    ultimo = NULL;
    while (p != NULL)
   {    c = con * p->coef;
        if (c != 0)
           creaNodoPoly(c, p->exp);
       p = p->sig;
   }
  return *this;
}
```

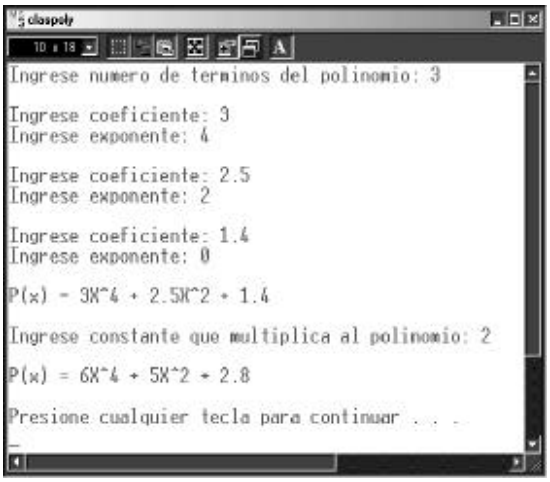

Figura 5. Operación «multiplicar una constante por un polinomio»: Código y ejecución

```
polinomio polinomio :: sumaPoly( polinomio a,
polinomio b )
{
   float c;
   POLY p, q;
   p = a.primero;
   q = b.primero;
   ultimo = NULL;
    while (p != NULL && q != NULL)
   {  if (p->exp == q->exp)
     {  c = p->coef + q->coef;
        if (c != 0)
           creaNodoPoly(c,p->exp);
       p = p->sig;
       q = q->sig;
     }else
        if (p->exp > q->exp)
        {  creaNodoPoly(p->coef, p->exp);
          p = p->sig;
        }
        else {
           creaNodoPoly(q->coef, q->exp);
          q = q->sig;
        }
   }
    while( p != NULL)
   {    creaNodoPoly(p->coef, p->exp);
       p = p->sig;
   }
    while( q != NULL)
   {    creaNodoPoly(q->coef, q->exp);
       q = q->sig;
   }
  return *this;
}
```

Figura 6. Operación «sumar dos polinomios»: Código y ejecución

```
polinomio polinomio :: monomioXpolinomio(float co,
int ex, polinomio a)
{
   float c;
   int d;
    POLY p;

     p = a.primero;
    ultimo = NULL;
    while (p != NULL)
   {   c = co * p->coef;
       d = ex + p->exp;
       if (c != 0)
           creaNodoPoly(c,d);
      p = p->sig;
    }
  return *this;
}
```

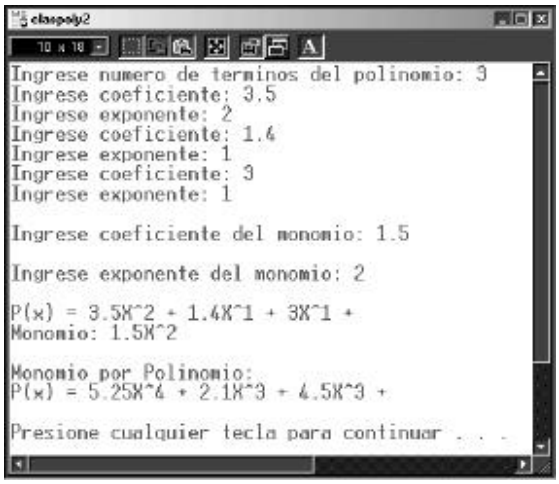

Figura 7. Operación «multiplicar un monomio por un polinomio»: Código y ejecución

El código de la función miembro *printPoly( )* que permite imprimir o mostrar un polinomio se muestra en la Figura 4.

Operación Constante por Polinomio

La operación de multiplicar una constante por un polinomio consiste en multiplicar la constante por cada coeficiente de los términos que componen el polinomio.

La función miembro *const_mult_poly* devuelve un objeto del tipo polinomio que es el objeto resultante de la operación. Observe que se utiliza el puntero **this* para hacer referencia al objeto resultante. El código que implementa esta operación se presenta en la Figura 5, así como su ejecución.

Operación Suma de Polinomios

Para sumar dos polinomios se debe tener en cuenta los exponentes de los términos, si estos son iguales entonces se suman los coeficientes, siendo el exponente el mismo. Si los exponentes son diferentes se crea un nuevo término o nodo.

Es importante destacar que el resultado es otro término por lo que el código debe invocar la creación de un nuevo nodo para almacenar el resultado nodo a nodo.

La Figura 6, presenta el código y la ejecución del programa que suma dos polinomios. Al igual que con la función que multiplica constante por polinomio el objeto resultante es devuelto mediante el puntero **this*.

```
polinomio polinomio :: derivaPolinomio(polinomio a)
{
   float c;
   int d;
    POLY p;

    p = a.primero;
    ultimo = NULL;
    while (p != NULL)
   {   c = p->coef * p->exp;
       d = p->exp - 1;
       if (c != 0)
           creaNodoPoly(c,d);
      p = p->sig;
    }
  return *this;
}
```

Figura 8. Operación «derivar un polinomio»: Código y ejecución

Operación Multiplicar un Monomio por un Polinomio

Para multiplicar un monomio por un polinomio, dado que un monomio consta de un término; el coeficiente y el exponente se almacenan en variables simples (c y d en el código de la Figura 7), las que son usadas para multiplicar los términos del polinomio.

Asimismo, la Figura 7 muestra la ejecución del programa que multiplica un monomio por un polinomio.

Operación Derivada de un Polinomio

Para derivar un polinomio se toma cada término multiplicando el coeficiente por su exponente y luego disminuyendo el exponente en un grado. El término resultante se almacena en un nuevo nodo por lo que se llama a la función *creaNodoPoly(c,d)*.

En la Figura 8, se presenta el código y la ejecución del programa para derivar un polinomio.

CONCLUSIONES

El artículo muestra como es posible la creación de tipos abstractos de datos para propósitos particulares, en este caso implementar un polinomio y algunas de sus operaciones. Esto es posible debido a que C++ es un lenguaje multiparadigma y dentro de la programación orientada a objetos pueden implementarse nuevos tipos abstractos de datos mediante clases.

Se cree conveniente modificar el programa para hacerlo genérico; es decir que trabaje con otros tipos de datos y no sólo con coeficientes reales ni coeficientes enteros. Para lograr este se recomienda la utilización de patrones o plantillas que son una utilidad importante que permite el lenguaje C++.

Finalmente, puede utilizarse el concepto de sobrecarga de operadores para adicionar un nuevo comportamiento a los operadores aritméticos primitivos de C++, tal que permitan trabajar directamente con polinomios.

BIBLIOGRAFÍA

1. Gamma, Erich; Helm, Richard & Others. (1999). *Introduction to Oriented Design in C++.* 1ra. Ed. Addison Wesley Publishing Company, Inc. USA.

2. Ruiz L., E. y Hinojosa L., H.(2003). *Implementación de un Tipo Abstracto de Datos para Gestionar Conjuntos Usando el Lenguaje de Programación C++.* Revista Industrial Data, Vol. (6)2: pp. 56-62.

3. Ruiz L., E. (2004). *Un Programa en C++ que Implementa Grupos Abelianos*. Revista Industrial Data Vol. (7)1: pp. 55-60.

4. Stroustrup, Bjarne. (2002). *El Lenguaje de Programación C++.* Edición especial. Adisson Wesley - Pearson Educación S.A. España.

5. Volodymyr M. (2004). *Typelist & C++ Poynomial Meta-Arithmetic*. En C/C++ Users Journal, August, USA.