

# ● UN PROGRAMA EN C++ QUE IMPLEMENTA GRUPOS ABELIANOS

(1) Edgar Ruiz L

## RESUMEN

El artículo presenta un programa en C++ que implementa el concepto algebraico de grupo conmutativo o abeliano. Se muestra todo el código del programa que gestiona objetos de la clase abeliano como un nuevo tipo abstracto de dato, TAD utilizando conceptos como la sobrecarga de operadores. La implementación se ha realizado en el compilador Dev C++ 4.1; un compilador GNU con licencia GPL.

**Palabras Claves:** Grupo conmutativo o abeliano. Sobrecarga de operadores. Tipo abstracto de datos.

## C++ PROGRAM IMPLEMENTING THE ABELIAN GROUP ABSTRACT

This article presents a C++ Program implementing the Commutative or Abelian Group Algebraic Concept. The whole Program code that manages abelian class objects as a new data abstract type, TAD, using concepts such as operators overcharge is shown. Implementation has been carried out in a Dev C++ 4.1 compiler, a GNU compiler with GPL licence.

**Key Words:** Commutative or abelian group. Operators overcharge. Data abstract type.

(1) Docente de la Facultad de Ingeniería Industrial, UNISM.  
Email: eruizl@unism.edu.pe

## INTRODUCCIÓN

### Definición de Grupo

Sean un conjunto no vacío  $G$  y una función  $*$ . El par  $(G, *)$  es un grupo si y solo si  $*$  es una ley interna en  $G$ , asociativa, con neutro, y tal que todo elemento de  $G$  admite inverso respecto de  $*$ .

En forma simbólica se tiene:

**Definición:**  $(G, *)$  es un grupo si y solo si se verifican los axiomas

[ ]

[ ]

[ ]

[ ]

Si además se verifica:

[ ]

Entonces, el grupo se llama conmutativo o abeliano.

### Ejemplos de grupos

El conjunto de los números enteros, con la operación suma, es decir,  $(\mathbb{Z}, +)$ , tiene estructura de grupo abeliano. Igualmente los conjuntos  $\mathbb{Q}$  (números racionales),  $\mathbb{R}$  (números reales) y  $\mathbb{C}$  (números complejos).

El conjunto  $\mathbb{Q}^* = \mathbb{Q} \setminus \{0\}$  de los números racionales no nulos, forma con la operación producto un grupo abeliano. Igualmente ocurre con los conjuntos  $\mathbb{R}^*$  y  $\mathbb{C}^*$ . Obsérvese que, para formar grupos multiplicativos, siempre tenemos que excluir el elemento nulo, que no tiene inverso (pero el producto de números no nulos es no nulo).

El conjunto

[ ]  
(conjunto de polinomios con coeficientes reales), con la operación suma de polinomios, forma un grupo abeliano. No puede formar grupo con el producto, puesto que no todos los polinomios tienen inverso.

>>> Un Programa de C++ que Implementa Grupos Abelianos

Dados dos enteros positivos  $m$  y  $n$ , si se considera:

(conjunto de matrices  $m \times n$  con elementos reales), este conjunto forma, con la operación suma de matrices, un grupo abeliano.

Dado un entero positivo  $n$ , consideramos el conjunto

(matrices cuadradas de orden  $n$  e invertibles). Es fácil comprobar que este conjunto, dotado de la operación producto de matrices, constituye un grupo que no es abeliano.

Dado un conjunto cualquiera  $E$ , se define:

(conjunto de las biyecciones de  $E$ ); no es difícil comprobar que, si " $\circ$ " denota la operación composición de aplicaciones,  $(B(E), \circ)$  constituye un grupo que, en general, no sería abeliano.

**Ejemplo de aplicación**

En el conjunto  $Z$  de los enteros se define  $*$  mediante:

(1)

Puede afirmarse que el par  $(Z, *)$  es un grupo abeliano. En efecto, se verifican:

De (2) y (3) resulta que:

Si  $e$  es neutro, entonces  $\square$   
 Por (1):  $\square$  y resulta:  $\square$

Análogamente se prueba que  $-3$  es neutro a izquierda.

$G_4$ . Todo elemento de  $G$  es inversible respecto de  $*$

Si  $\square$  es inverso de  $\square$  entonces debe verificarse:

Teniendo en cuenta (1) y que  $\square$

Luego:  $\square$

De modo análogo se prueba el inverso a izquierda.

De acuerdo con (1) y con la conmutatividad de la suma ordinaria en  $Z$ .

**Aplicaciones de la teoría de grupos**

La teoría de grupos y en particular los grupos abelianos tiene aplicaciones en campos como la criptografía o encriptación de datos, seguridad informática, espacios topológicos, la topología de redes de comunicación y de datos, gráficas de superficies y teoría de grafos.

**IMPLEMENTACIÓN DEL TAD ABELIANO**

El objetivo del presente trabajo es implementar o definir un tipo abstracto de datos (TAD) que permita gestionar objetos que correspondan a la teoría de grupos; en este caso los grupos conmutativos o abelianos. Para implementar un tipo abstracto de datos en una computadora deben tenerse en cuenta dos cosas:

1. El conjunto de datos que tendrá el TAD y
2. Las operaciones definidas para dichos datos.

Para el TAD abeliano se define los enteros (aunque bien pueden ser los reales o los racionales u otro tipo de dato primitivo). Adicionalmente se definirán las siguientes operaciones que forman grupos abelianos.

**1. Leyes internas**

- $*$  tal que  $a * b = a + b + 3$
- $+$  tal que  $a + b = a + b + 1$
- $-$  tal que  $a - b = a - b - 3$
- $/$  tal que  $a / b = a / b + 2$

**2. Operadores de asignación**

- $=$  asignación
- $*=$  asignación con  $*$
- $+=$  asignación con  $+$
- $-=$  asignación con  $-$
- $/=$  asignación con  $/$

```

#include <iostream.h>
#include <stdlib.h>
//Autor: Edgar Ruiz Lizama
//Definicion de grupo abeliano
class abeliano{
private :
    int dato;
public :
    abeliano(); // constructor vacio
    abeliano(int); // constructor alternativo
    abeliano(const abeliano&); // constructor de copia
    //Producto
    friend abeliano operator*(const abeliano&, const
abeliano&);
    //Verifica propiedad asociativa
    friend bool pAsociativa(const abeliano&, const
abeliano&, const abeliano&);
    //Verifica propiedad conmutativa
    friend bool pConmutativa(const abeliano&, const
abeliano&);
    //Verifica existencia del elemento neutro o identidad
    friend abeliano operator-(const abeliano& );
    //Verifica existencia de inversos
    friend abeliano operator~(const abeliano& );
    //Suma
    friend abeliano operator+(const abeliano&, const
abeliano&);
    //Resta
    friend abeliano operator-(const abeliano&, const
abeliano&);
    //Division
    friend abeliano operator/(const abeliano&, const
abeliano&);
    //operadores de asignacion
    abeliano& operator=(const abeliano&);
    abeliano& operator*=(const abeliano&);
    abeliano& operator+=(const abeliano&);
    abeliano& operator-=(const abeliano&);
    abeliano& operator/=(const abeliano&);
    //entrada salida
    void print();
    friend ostream& operator<<(ostream&, const abeliano&);
    friend istream& operator>>(istream&, abeliano&);
};
// constructores
abeliano :: abeliano()
{
    dato = 0;
}
abeliano :: abeliano(int num)
{
    dato = num;
}
abeliano :: abeliano(const abeliano& a)
{
    dato = a.dato;
}
//Definicion de ley interna *
abeliano operator*(const abeliano& a, const abeliano& b)
{
    abeliano dd = a.dato + b.dato + 3;
    return dd;
}
//propiedades
bool pAsociativa(const abeliano& a, const abeliano& b,
const abeliano& c)
{
    if (((a*b)*c).dato == (a*(b*c)).dato)
        return true;
    else
        return false;
}
bool pConmutativa(const abeliano& a, const abeliano& b)
{
    if ((a*b).dato == (b*a).dato)
        return true;
    else
        return false;
}
//Existencia del neutro o identidad: lo devuelve
abeliano operator-(const abeliano& a)
{
    abeliano e;
    e.dato = -a.dato;
    if ((a*e).dato == a.dato && a.dato == (a*e).dato)
        return e;
}
//Existencia del inverso
abeliano operator~(const abeliano& a)
{
    abeliano e = -a;
    abeliano ainv = -6 - a;
    if ((a*ainv).dato == e.dato)
        return ainv;
}
// otros grupos abelianos
//Definicion de ley interna +
abeliano operator+(const abeliano& a, const abeliano& b)
{
    abeliano dd = a.dato + b.dato + 1;
    return dd;
}
//Definicion de ley interna -
abeliano operator-(const abeliano& a, const abeliano& b)
{
    abeliano dd = a.dato - b.dato - 3;
    return dd;
}
//Definicion de ley interna /
abeliano operator/(const abeliano& a, const abeliano& b)
{
    if (b.dato != 0)

```

Figura 1a. Listado del programa *classabel2.cpp*

>>> Un Programa de C++ que Implementa Grupos Abelianos

```

{
    abeliano dd = a.dato / b.dato + 2;
    return dd;
}
else
    exit(1); //error
}
//Sobrecarga para la asignacion de abelianos
abeliano& abeliano :: operator=(const abeliano& b)
{
    dato = b.dato;
    return *this;
}
abeliano& abeliano :: operator*=(const abeliano& b)
{
    *this = *this * b;
    return *this;
}
abeliano& abeliano :: operator+=(const abeliano& b)
{
    *this = *this + b;
    return *this;
}
abeliano& abeliano :: operator-=(const abeliano& b)
{
    *this = *this - b;
    return *this;
}
abeliano& abeliano :: operator/=(const abeliano& b)
{
    *this = *this / b;
    return *this;
}
void abeliano :: print()
{
    cout<<dato<<endl;
}
//sobrecarga para el operador <<
ostream& operator<<(ostream& os, const abeliano& r)
{
    os<<r.dato;
    return os;
}
//sobrecarga para >>
istream& operator>>(istream& is, abeliano& r)
{
    is>>r.dato;
    return is;
}
int main() //classabel.cpp
{
    abeliano a(3), b(2), c, d(1);
    abeliano inv, neutro;
    cout<<"Abelianos"<<endl;
    cout<<"a = "; a.print();
    cout<<"b = "; b.print();
    c = a * b;
    cout<<"c = a * b = "; c.print();
    cout<<"d = "; d.print();

    if (pAsociativa(a,b,c))
        cout<<"\nSe cumple la Asociatividad en * con a, b, c
!..."<<endl;
    else
        cout<<"\nNo se cumple la Asociatividad"<<endl;
    if (pConmutativa(a,b))
        cout<<"\nSe cumple la Conmutatividad en * con a , b
!..."<<endl;
    else
        cout<<"\nNo se cumple la Conmutatividad"<<endl;

    cout<<"Existencia del neutro de a = "; neutro = -a;
    neutro.print();
    cout<<"Existencia del inverso de a = "; inv = ~a;
    inv.print();

    cout<<"\nOtras grupos abelianos"<<endl;
    c = a + b;
    cout<<"c = a + b = "; c.print();
    c = a - b;
    cout<<"c = a - b = "; c.print();
    c = a / b;
    cout<<"c = a / b = "; c.print();

    // Asignacion
    d = c;
    cout<<"\nAsignacion: d = c => "; d.print();
    cout<<" a = "; a.print();
    cout<<" b = "; b.print();
    a *= b;
    cout<<"\nAsignacion: a *= b=> "; a.print();
    a += b;
    cout<<"\nAsignacion: a += b => "; a.print();
    a -= b;
    cout<<"\nAsignacion: a -= b => "; a.print();
    a /= b;
    cout<<"\nAsignacion: a /= b => "; a.print();
    cout<<endl;

    //probando la sobrecarga para << y >>
    abeliano abel(5);
    cout<<"Probando operadores de entrada/salida"<<endl;
    cout<<abel<<endl;
    cin>>abel;
    cout<<abel;
    cout<<endl;
    system("PAUSE");
    return 0;
}

```

Figura 1b. Listado del programa *classabel2.cpp*

```

E:\Dev-C++\Bin\lcpp\classabel2.exe
Abelianos
a = 3
b = 2
c = a * b = 8
d = 1

Se cumple la Asociatividad en * con a, b, c ?...
Se cumple la Conmutatividad en * con a , b ?...
Existencia del neutro de a = -3
Existencia del inverso de a = -3

Otros grupos abelianos
c = a + b = 6
c = a - b = -2
c = a / b = 3

Asignacion: d = c => 3
a = 3
b = 2

Asignacion: a *= b=> 8
Asignacion: a += b => 11
Asignacion: a -= b => 6
Asignacion: a /= b => 5

Probando operadores de entrada/salida
5
?
?
Presione una tecla para continuar . . . -

```

Figura 2. Salida del programa *classabel2.cpp*

### 3. Operadores de entrada / salida

- << operador de extracción de flujo
- >> operador de inserción de flujo

Para la implementación de estas leyes internas de grupos abelianos se emplea la sobrecarga de operadores, concepto que en programación orientada a objetos, consiste en añadir comportamiento adicional a los operadores definidos en el lenguaje a fin de adecuarlos a los nuevos tipos abstractos de datos definidos por los programadores.

A modo de ejemplo se menciona que el operador + esta definido para sumar enteros, y flotantes pero no para sumar dos objetos del tipo abeliano; de allí la necesidad de sobrecargar estos operadores para que respondan a nuevos requerimientos.

Se incluyen funciones como: print para imprimir un objeto abeliano, pAsociativa y pConmutativa que de-

vuelven valores de verdad acerca de si los abelianos cumplen o no la propiedad correspondiente. En las Figuras 1a y 1b se presenta la implementación de la clase abeliano.

En la Figura 2, se presenta la ejecución del programa que implementa la clase abeliano.

### CONCLUSIONES

El artículo muestra la posibilidad de crear tipos abstractos de datos abstractos para propósitos particulares.

En este artículo para implementar el concepto algebraico de los grupos conmutativos o abelianos. Esto es posible debido a que C++ es un lenguaje multiparadigma y dentro de la programación orientada a objetos pueden implementarse nuevos tipos abs-

>>> *Un Programa de C++ que Implementa Grupos Abelianos*

tractos de datos mediante clases y sobrecarga de operadores.

Un grupo abeliano se define mediante una ley interna que define una operación particular. Observe que para el programa presentado se ha definido más de una ley, solo con la finalidad de mostrar el funcionamiento de objetos abelianos. En implementaciones particulares es deseable rescribir el programa incluyendo solo la ley interna con la que se desea trabajar.

El programa se ha escrito en lenguaje C++ pero también puede escribirse en Java, C#, u otro que soporte la programación orientada a objeto.

**BIBLIOGRAFÍA**

1. Gamma, Erich; Helm, Richard & Others. (1999). *Introduction to Oriented Design in C++*. 1ra. Ed. Addison Wesley Publishing Company, Inc. U.S.A.
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. Stroustrup, Bjarne. (2002). *El Lenguaje de Programación C++, Edición especial*. Addison Wesley - Pearson Educación S.A. España.