ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

REALIZACIÓN DE CONCEPTOS ABSTRACTOS DE LA TEORÍA DE GRUPOS POR MEDIO DE UN MODELO COMPUTACIONAL

IMPLEMENTACIÓN DE UN MODELO COMPUTACIONAL EN EL LENGUAJE C++ A LOS GRUPOS DE KLEIN, D_4 Y U_8.

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE MATEMÁTICO

CARLOS DAVID GONZÁLEZ LÓPEZ

carlos.gonzalez04@epn.edu.ec

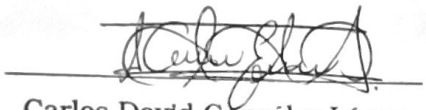
DIRECTOR: EURO DE JESUS LUCENA DELGADO

euro.lucena@epn.edu.ec

DMQ, SEPTIEMBRE 2022

CERTIFICACIONES

Yo, CARLOS DAVID GONZÁLEZ LÓPEZ, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.



Carlos David González López

Certifico que el presente trabajo de integración curricular fue desarrollado por Carlos David González López, bajo mi supervisión.

Euro de Jesus Lucena Delgado

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Carlos David González López

Euro de Jesus Lucena Delgado

AGRADECIMIENTOS

En primer lugar, agradezco a mis padres y a mis hermanos por todo el cariño y el apoyo brindado durante todos estos años. Agradezco a mis amigos con los que camine durante toda la carrera. Agradezco a todos mis profesores, en especial al Doc. Francisco Lara, Doc. Nerio Borges y al Doc. Euro de Jesus Lucena quienes me guiaron en la realización de este trabajo. Gracias a todos.

DEDICATORIA

A mis padres,

la razón de mi ser.

RESUMEN

Este trabajo trata sobre la implementación de conceptos abstractos de la teoría de grupos por medio de un modelo computacional. Para la realización del trabajo se ha recopilado información de la teoía de grupos y de la programación orientada a objetos en el lenguaje C++, siendo las principales fuentes de consulta la versión en español del Álgebra moderna de Israel Nathan Herstein y la cuarta edición del libro The C++ Programming Language de Bjarne Stroustrup.

El enfoque del trabajo es cualitativo, basado en la descripción de elementos abstractos de forma deductiva, yendo de la exposición de los conceptos generales al estudio de los más específicos, concretamente en tres casos de grupos algebraicos.

El objetivo principal de este trabajo es implementar tres grupos algebraicos mediante un modelo computacional en el lenguaje C++ con el fin de visualizar los elementos abstractos que conforman dichos grupos. El resultado obtenido ha sido la implementación de los grupos de Klein, D_4 y U_8, que permite observar de mejor manera los conceptos de grupo, grupo abeliano, subgrupo y subgrupo normal.

Palabras clave: Grupos en el lenguaje C++, Grupo de Klein, Grupo D_4, Grupo U_8.

ABSTRACT

This paper is about the implementation of the groups theory abstract concepts through a computational model. Information on Groups theory and Object-oriented programming in C++ language were collected for the realization of this paper. The principal bibliography are Modern Algebra spanish's version of Israel Nathan Herstein and the fourth edition of The C++ Programming Language of Bjarne Stroustrup.

The paper´s focus is qualitative, describing the abstract elements in a deductive way, since the general concepts exposition until the more specifics study.

The principal goal is to implement three algebraic groups through a computational model in C++ language, to visualize the abstract elements of that ones. The result obtained was the Klein´s, D_4 and U_8 groups implementation, which allows show better the group, abelian group, subgroup and normal subgroup concepts.

Keywords: Groups in C++ language, Klein´s group, Group D_4, Group D_4.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

El álgebra es una de las áreas más importantes de las matemáticas. Su importancia radica en el estudio de conceptos abstractos entre los cuales se destancan las estructuras algebraicas como los grupos, anillos, campos y espacios vectoriales. Una estructura algebraica es un conjunto de objetos que se relacionan por medio de una operación. Dichas estructuras son fundamentales en diversas áreas de las matemáticas como la geometría, teoría de números, análisis, topología y matemática aplicada. Es por ello que este componente es considerado como un hilo unificador que entrelaza casi todas las matemáticas y su estudio es de fundamental importancia para el desarrollo de las mismas.

1.1. Objetivo general

Implementar tres grupos algebraicos mediante un modelo computacional en el lenguaje C++ con el fin de visualizar los elementos abstractos que conforman dichos grupos.

1.2. Objetivos específicos

1. Implementar los elementos y operaciones de los grupos de Klein, D_4 y U_8 mediante las clases en el lenguaje C++ para la construcción

dichos grupos.

2. Testear las implementaciones de los grupos de Klein, D_4 y U_8 mediante sus propiedades para confirmar su concordancia con las definiciones abstractas.

3. Determinar si un subconjunto es subgrupo o subgrupo normal de los tres grupos mediante la implemetación en C++, para poder estudiar de mejor manera estos objetos.

1.3. Alcance

Empleando la programación orientada a objetos se necesita, entender e implementar la definición de grupo en el sentido del lenguaje C++. Luego, se procede a realizar implementaciones de ciertos grupos y de conceptos matemáticos de la teoría de grupos como la noción de pertenencia, subgrupo y subgrupo normal. Finalmente, se procede a la realización de distintas pruebas de los grupos implementados como la visualización del funcionamiento de su operación mediante una tabla.

1.4. Marco teórico

En esta sección se exponen las definiciones y resultados de la Teoría de Grupos necesarios para comprender de mejor manera el trabajo realizado. Dicha información toma como referencia el libro [1].

1.4.1. Teoría de Grupos

Definición 1.1. Un conjunto no vacío de elementos G se dice que forma un grupo si en G está definida una operación binaria, llamada producto y denotada por $(\cdot)$ tal que:

- $a, b \in G$ implica que $a \cdot b \in G$.
- $a, b, c \in G$ implica que $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- Existe un elemento $e \in G$ tal que $a \cdot e = e \cdot a = a$, para todo $a \in G$.

- Para todo $a \in G$ existe un elemento $a^{-1} \in G$ tal que $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Para tener una mejor apreciación de este concepto matemático, a continuación se expone como ejemplo el grupo D_4.

Grupo D_4

El grupo diedral D_n es el grupo de simetrías de los polígonos regulares de n lados. En el caso particular $n = 4$, el grupo D_4 es el grupo de las simetrías de un cuadrado y consta de 8 elementos.

Elementos del grupo D_4

Para poder visualizar de mejor manera los elementos del grupo D_4 como las simetrías de un cuadrado vamos a numerar sus vértices del 1 al 4.

- El primer elemento, es el elemento **neutro**, se lo nota como "id" y es el cuadrado sin grirarlo ni rotarlo en ninguna direción. Como se observa en la figura 1.1a.
- El segundo elemento, se lo nota como "g_1" y es la rotación de 90° del cuadrado en sentido antihorario alrededor de su centro. Como se observa en la figura 1.1b.

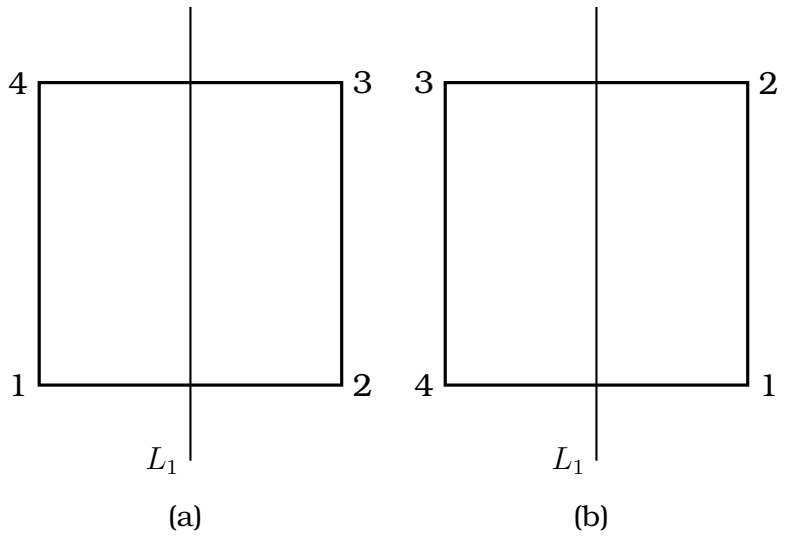


Figura 1.1: Elemento neutro y g_1 del grupo D_4

- El tercer elemento, se lo nota como "g_2" y es la rotación de 180° del cuadrado en sentido antihorario alrededor de su centro. Como se observa en la figura 1.2a.
- El cuarto elemento, se lo nota como "g_3" y es la rotación de 270° del cuadrado en sentido antihorario alrededor de su centro. Como se observa en la figura 1.2b.

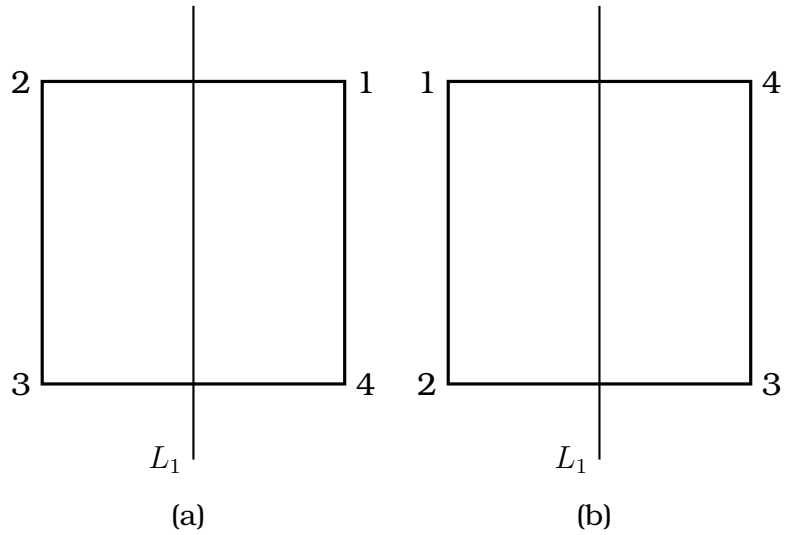


Figura 1.2: Elementos g_2 y g_3 del grupo D_4

- El quinto elemento, se lo nota como "f_1" y es la reflexión del cuadrado respecto a L_1. Como se observa en la figura 1.3a.
- El sexto elemento, se lo nota como "f_2" y es la reflexión de g_1 respecto a L_1. Como se observa en la figura 1.3b.
- El séptimo elemento, se lo nota como "f_3" y es la reflexión de g_2 respecto a L_1. Como se observa en la figura 1.4a.
- El octavo elemento, se lo nota como "f_4" y es la reflexión de g_3 respecto a L_1. Como se observa en la figura 1.4b.

Estos elementos se relacionan mediante su operación que viene dado por la tabla 1.1. En la cual se puede constatar que se cumplen las condiciones de grupo descritas en la definición 1.1.

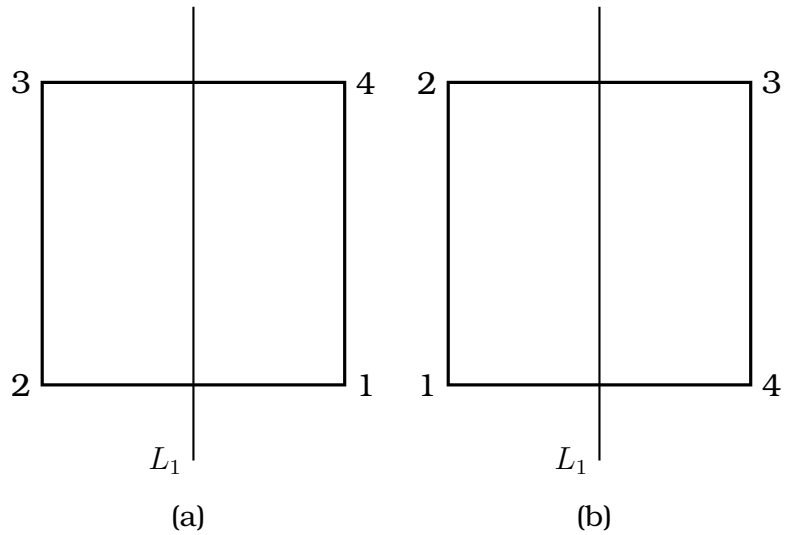


Figura 1.3: Elementos f_1 y f_2 del grupo D_4

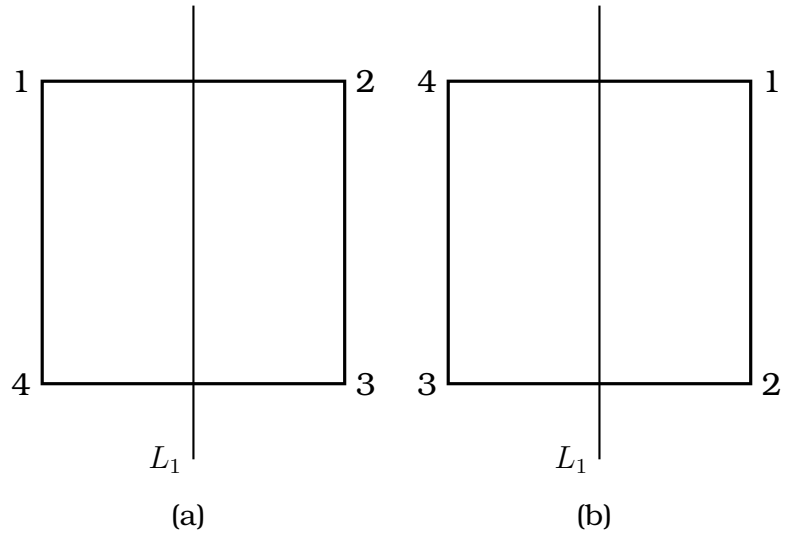


Figura 1.4: Elementos f_3 y f_4 del grupo D_4.

	id	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{f}_1$	$\mathbf{f}_2$	$\mathbf{f}_3$	$\mathbf{f}_4$
id	id	g_1	g_2	g_3	f_1	f_2	f_3	f_4
$\mathbf{g}_1$	g_1	g_2	g_3	id	f_2	f_3	f_4	f_1
$\mathbf{g}_2$	g_2	g_3	id	g_1	f_3	f_4	f_1	f_2
$\mathbf{g}_3$	g_3	id	g_1	g_2	f_4	f_1	f_2	f_3
$\mathbf{f}_1$	f_1	f_4	f_3	f_2	id	g_3	g_2	g_1
$\mathbf{f}_2$	f_2	f_1	f_4	f_3	g_1	id	g_3	g_2
$\mathbf{f}_3$	f_3	f_2	f_1	f_4	g_2	g_1	id	g_3
$\mathbf{f}_4$	f_4	f_3	f_2	f_1	g_3	g_2	g_1	id

Cuadro 1.1: Tabla de la operación del grupo de D_4.

Definición 1.2. Un grupo G se dice que es **abeliano o conmutativo** si para cualesquier $a, b \in G$ se tiene que: $a \cdot b = b \cdot a$.

Grupo de Klein

El grupo de Klein es un ejemplo de grupo algebraico abeliano. Este grupo puede ser visto como las simetrías de un rectángulo, consta de 4 elementos y cada elemento es su propio inverso.

Elementos del grupo de Klein

Al igual que antes, para visualizar de mejor manera los elementos del grupo de Klein como las simetrías de un rectángulo vamos a numerar los vértices de dicho rectángulo del 1 al 4.

- El primer elemento, es el elemento **neutro**, se lo nota como "id" y es el rectángulo sin grirarlo ni rotarlo en ninguna direción. Como se observa en la figura 1.5.

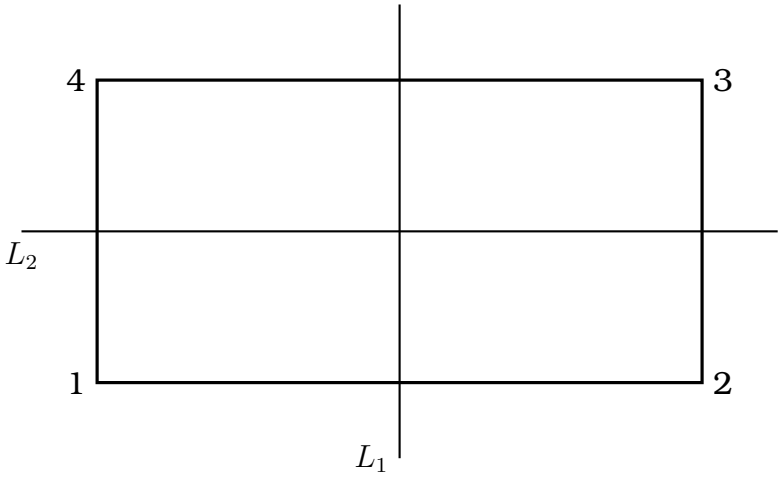


Figura 1.5: Elemento neutro del grupo de Klein.

- El segundo elemento, se lo nota como "r" y es la rotación de 180° del rectángulo en cualquier dirección alrededor de su centro. Como se observa en la figura 1.6.
- El tercer elemento, se lo nota como "v" y es la rotación de 180° del rectángulo en cualquier dirección alrededor del eje vertical L_1 . Como se observa en la figura 1.7.

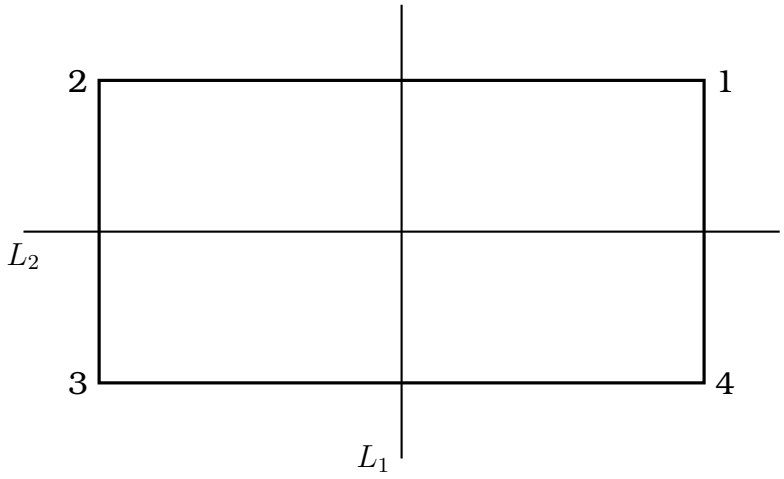


Figura 1.6: Elemento r del grupo de Klein.

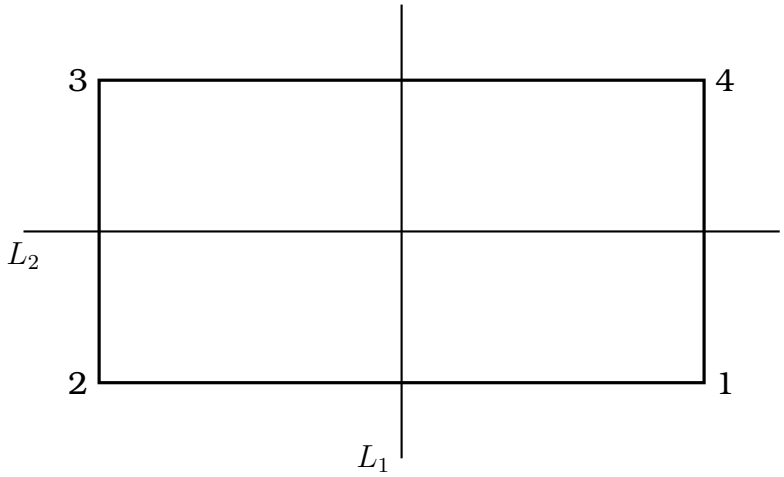


Figura 1.7: Elemento v del grupo de Klein.

- El cuarto elemento, se lo nota como "h" y es la rotación de 180° del rectángulo en cualquier dirección alrededor del eje horizontal L_2 . Como se observa en la figura 1.8.

Estos elementos se relacionan mediante su operación que viene dada por la tabla 1.2. La cual muestra que se cumplen las definiciones 1.1 y 1.2. Lo cual indica que en efecto el grupo de Klein es un grupo abeliano.

	id	**r**	**v**	**h**
id	id	r	v	h
r	r	id	h	v
v	v	h	id	r
h	h	v	r	id

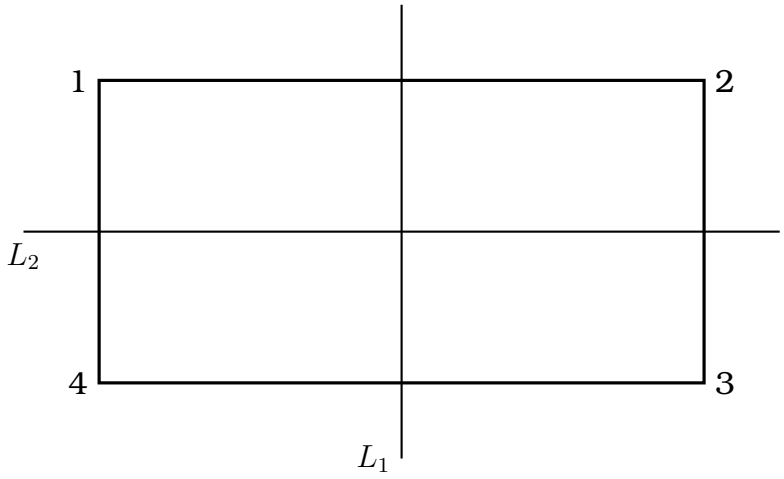


Figura 1.8: Elemento h del grupo de Klein.

Cuadro 1.2: Tabla de la operación del grupo de Klein.

Grupo U_8

Otro ejemplo de grupo abeliano es el grupo U_8. Para definir este grupo, es necesario abordar las siguientes definiciones y resultados.

Proposición 1.1. (*Algoritmo de Euclides*). Sean a, b números enteros, tal que $b \neq 0$. Entonces existen m, r números enteros únicos tales que:

$$a = mb + r \quad \text{donde } 0 \leq r < |b|$$

A r se lo denomina el residuo de la división de a y b.

Definición 1.3. Sean a, b números enteros, tal que $b \neq 0$. Decimos que b es *divisor de a* si existe un número entero m tal que $a = mb$.

Definición 1.4. Sean a, b números enteros, el entero positivo c se dice que es el *máximo común divisor* de a y b si:

- c es un divisor de a y de b;
- Cualquier divisor de a y de b es divisor de c.

Se nota por (a, b) al máximo común divisor de a y b.

Definición 1.5. Los números enteros a y b se dicen *primos relativos* si $(a, b) = 1$.

El grupo U_8 es el grupo de los números primos relativos a 8 que son menores a 8. Es decir es el conjunto $\{1,3,5,7\}$ con su operación definida de la siguiente forma.

$$\begin{aligned} * : \{1,3,5,7\} x \{1,3,5,7\} &\longrightarrow \{1,3,5,7\} \\ (i,j) &\rightarrow \quad i * j = k \end{aligned} \tag{1.1}$$

Donde k es el residuo de la división de $i \cdot j$ y 8. Por ejemplo,

$$3 * 7 = 5$$

pues, $3 \cdot 7 = 21 = 2 \cdot 8 + 5$.

Notar que gracias a la proposición 1.1, $*$ está bien definida. Además, gracias a la tabla 1.1 se tiene que $(\{1,3,5,7\}, *)$ cumple con las definiciones 1.1 y 1.2, lo que hace a U_8 un grupo abeliano.

	1	**3**	**5**	**7**
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

Cuadro 1.3: Tabla de la operación del grupo U_8.

Definición 1.6. Sean n un número natural y G un grupo con n elementos, n se dice que es el *orden de G* y se nota por $o(G)$.

Definición 1.7. Un subconjunto H de un grupo G se dice que es *subgrupo de G*, si H es un grupo respecto al producto de G. Notar que G es subgrupo de sí mismo.

A continuación se muestran algunos subgrupos de los grupos antes expuestos. Para ello, primero es necesario la siguiente proposición.

Proposición 1.2. Si H es un conjunto finito no vacío de un grupo G, y H es cerrado respecto al producto de G, entonces H es subgrupo de G.

Los siguientes conjuntos son todos subconjuntos propios del grupo D_4 que cumplen con la proposición 1.2. Es decir, son todos los subgrupos de D_4.

$$H_1 = \{id, g_2\} \qquad H_5 = \{id, f_4\}$$
$$H_2 = \{id, f_1\} \qquad H_6 = \{id, g_1, g_2, g_3\}$$
$$H_3 = \{id, f_2\} \qquad H_7 = \{id, g_2, f_1, f_3\}$$
$$H_4 = \{id, f_3\} \qquad H_8 = \{id, g_2, f_2, f_4\}$$

Figura 1.9: Subgrupos del grupo D_4.

De igual forma, todos los subgrupos propios del grupo de Klein son:

$$K_1 = \{id, r\}$$
$$K_2 = \{id, v\}$$
$$K_3 = \{id, h\}$$

Figura 1.10: Subgrupos del grupo de Klein.

Y por último, todos los subgrupos propios del grupo U_8 son:

$$J_1 = \{1, 3\}$$
$$J_2 = \{1, 5\}$$
$$J_3 = \{1, 7\}$$

Figura 1.11: Subgrupos del grupo U_8.

Definición 1.8. Un subgrupo N de G se dice que es un *subgrupo normal de G* si para todo $g \in G$ y todo $n \in N$, se tiene que gng^{-1}.

Los subgrupos de D_4 que cumplen con la definición 1.8, es decir, que son normales, son:

$$H_1 = \{id, g_2\}$$
$$H_6 = \{id, g_1, g_2, g_3\}$$
$$H_7 = \{id, g_2, f_1, f_3\}$$
$$H_8 = \{id, g_2, f_2, f_4\}$$

Figura 1.12: Subgrupos normales del grupo D_4.

En el caso del grupo de Klein y de U8 todos sus subgrupos cumplen con la definición 1.8, es decir, todos sus subgrupos son normales. Esto es debido a que estos grupos son abelianos.

Capítulo 2

Metodología

Para el desarrollo del componente en este trabajo se hace uso del método deductivo, empezando por los conceptos de estutructuras algebraicas, como lo es la definición de grupo, hasta exponer grupos específicos que ayudan a visualizar dichas definiciones abtractas.

Este trabajo tiene un enfoque cualitativo, basado en la descripción de elementos abstractos mediante la recopilación de información, la cual se obtuvo tras la realización de un análisis documental. Esta descripción de conceptos abstractos se representa a través de un estudio de tres casos usando la implementación orientada a objetos en el lenguaje C++.

2.1. Implementación

Para exponer la implementación de los grupos es necesario abordar las siguientes definiciones acerca de la programación orientada objetos en lenguaje C++. Para ello, nos basaremos en el libro [3].

Definición 2.1. Una *declaración* es un enunciado que introduce un nombre en el programa. Esta especifica un tipo para la entidad nombrada:

- Un *tipo de dato* define un conjunto de posibles valores y un conjunto de operaciones para un objeto.
- Un *objeto* es cierto espacio de memoria que tiene el valor de algún tipo de dato.
- Un *valor* es un conjunto de bits interpretados según un tipo de dato.
- Una *variable* es un objeto con nombre.

Por ejemplo, en C++ el tipo de dato *int* con sus operaciones +,-,*,/ es lo que más aproximado a los números enteros.

Definición 2.2. Las *clases* en C++ se las define como una herramienta para crear nuevos tipos de datos.

Las clases en C++ es la herramienta que se utilizó para implementar los grupos expuestos en la capítulo anterior. Las implementaciones del grupo de Klein y D_4 son muy similares, solo cambiando en la definición de los elementos que conforman dichos grupos. Es por esto que este capítulo se divide en dos partes, la primera que muestra como se implementaron los grupos de Klein y D_4, y la segunda que muestra como se implementó el grupo U_8.

2.1.1. Grupos de Klein y D_4

Para exponer como se implementaron los grupos de Klein y D_4, se tomará la implementación del grupo D_4 para exhibirla, pues como se mencionó anteriormente dichas implementaciones son las mismas en estructura.

Definición de los elementos del grupo

En los grupos de Klein y D_4 sus elementos se puede considerar como funciones que van del conjunto $\{1,2,3,4\}$ en sí mismo. Por ejemplo, en el grupo de D_4, los elementos id y g_2 son las siguientes funciónes.

$$\begin{aligned} id : \{1,2,3,4\} &\longrightarrow \{1,2,3,4\} \\ x &\rightarrow \quad id(x) = x \end{aligned} \tag{2.1}$$

$$\begin{aligned} g_2 : \{1,2,3,4\} &\longrightarrow \{1,2,3,4\} \\ x &\rightarrow \quad g_2(x) \end{aligned} \tag{2.2}$$

tal que

$$g_2(x) = \begin{cases} x+2 \quad si \quad x = 1,2 \\ \\ x-2 \quad si \quad x = 3,4 \end{cases}$$

las cuales coinciden con lo expuesto en en las gráficas 1.1a y 1.2a.

Debido a esto, en la implementación se declaran cada una de estas funciones haciendo uso de la definición de un tipo de dato. Por ejemplo, en el grupo D_4 se lo hizo como se muestra en la figura 2.1.

```
typedef int (*funcion_de_D4) (int);  /***Defino mi tipo de dato en este
caso, funcion, que es el tipo de dato que es elemento del grupo***/
int id(int);
int g1(int);
int g2(int);
int g3(int);
int f1(int);
int f2(int);
int f3(int);
int f4(int);
```

Figura 2.1: Definición de las funciones que son elementos del grupo D_4

En este caso se llama al tipo de dato como "$funcion\_de\_D4$" el cual evalúa y devuelve un tipo de dato int, es decir, evalúa y retorna un entero. En particular, los valores del conjunto $\{1,2,3,4\}$. Cada una de estas funciones se implementan siguiendo lo visto en (2.1) y (2.2). Por ejemplo, $g1$ y $f4$ se implementaron como en la figura 2.2. Lo cual coincide con las definiciones dadas en el capítulo 1 y que están representadas en las figuras 1.1b y 1.4b, respectivamente.

```
int g1 (int a)
{
    if (a==1)
        return 4;
    else if(a==4)
        return 3;
    else if(a==2 || a==3)
        return a-1;
}

int f4 (int a)
{
    if (a==2 ||a==4)
        return a;
    else if(a==1)
        return 3;
    else
        return 1;
}
```

Figura 2.2: Implementación de las funciones g1 y f4

Definición del grupo D_4

Una vez creado este nuevo tipo de dato y definidos todos los elementos del grupo, se procede a crear el grupo, usando una clase. El grupo D_4 como clase fue implementado como se muestra en la figura 2.3.

La herramienta clase se compone de dos partes que facilitan la construcción de ideas abstractas. Estas dos secciones son la privada y la pública.

Según [3] la parte privada es el conjunto de funciones y tipos de datos que solo los que crearon la clase pueden acceder y son herramientas para construir la misma. Por ejemplo, en la clase D4 se tiene la única variable privada "$funcion$", la cual es del tipo "$funcion\_de\_D4$" y será la que defina el elemento del grupo D_4 dentro de la clase D4.

Por otro lado, según [3] la parte pública es el conjunto de funciones y tipo de datos que todos pueden usar. Por ejemplo, en la clase D4 se implementó la función pública "$imprimir()$" que muestra en pantalla elemento de D_4, y que cualquier persona que use la clase D4 puede acceder.

En la figura 2.3 se muestra las funciones y tipos de dato que se usaron para construir la clase D4. A continución se detalla cada una de ellas.

```
class D4
{
 public:
    //Constructores
     D4(funcion_de_D4);// Constructor con un argumento
     D4(funcion_de_D4,funcion_de_D4, funcion_de_D4, funcion_de_D4,
     funcion_de_D4,funcion_de_D4, funcion_de_D4, funcion_de_D4,
     funcion_de_D4); //Constructor con mas argumentos
     //Operadores
     bool operator==(const D4&) const;
     D4 operator*(const D4&) const;
     friend ostream& operator<<(ostream& , const D4& );
     //Otras funciones miembro
     void imprimir();
     D4 inverso() const;
     bool esta_en(vector<D4> );
     //Arreglo auxiliar
     static funcion_de_D4*giro;
 private:
    funcion_de_D4 funcion;
};
```

Figura 2.3: Implementación de la clase D4

Arreglo auxiliar

A lo largo de la implementación de la clase, se necesita tener a disposición los elementos del grupo, ya sea en eñ producto de dos elementos o para encontrar el inverso de alguno. Para ello, se incluye un arreglo de tipos de datos $funcion\_de\_D4$, al cual se lo denomina "$giro$" y almacena todos los elementos del grupo antes definidos. Su tamaño es igual al orden del grupo, es decir, en el caso de D_4 su tamaño es 8.

Constructores

De acuerdo con [3] un constructor es la forma de especificar como se debe inicializar un objeto de la clase. En el caso de la clase D4 se definieron los siguientes dos constructores, para definir un objeto de la clase D4 que representa un elemento del grupo D_4.

- El primer constructor para inicializar un objeto de la clase D4 necesita de un argumento que es un tipo de dato "$funcion\_de\_D4$". Este constructor se programó como se muestra en la figura 2.4

```
    D4::D4(funcion_de_D4 fun)
{
    funcion = fun;
    giro[0] = id;
    giro[1] = g1;
    giro[2] = g2;
    giro[3] = g3;
    giro[4] = f1;
    giro[5] = f2;
    giro[6] = f3;
    giro[7] = f4;
}
```

Figura 2.4: Implementación del primer constructor de la clase D4

Este constructor especifica el objeto de la clase asignando el argumento "fun" a la variable privada "$funcion$", y colocando cada función elemento del grupo en el arreglo "$giro$" en el orden presentado en la tabla 1.1.

- Y el segundo constructor para inicializar un objeto de la clase D4 necesita de nueve argumentos que son datos de tipo "$funcion\_de\_D4$". Este constructor se programó como se expone en la figura 2.5.

```
  D4::D4(funcion_de_D4 fun,funcion_de_D4 fun1, funcion_de_D4 fun2,
        funcion_de_D4 fun3,funcion_de_D4 fun4, funcion_de_D4 fun5,
        funcion_de_D4 fun6,funcion_de_D4 fun7,funcion_de_D4 fun8)
{
    funcion = fun;
    giro[0] = fun1;
    giro[1] = fun2;
    giro[2] = fun3;
    giro[3] = fun4;
    giro[4] = fun5;
    giro[5] = fun6;
    giro[6] = fun7;
    giro[7] = fun8;
}
```

Figura 2.5: Implementación del segundo constructor de la clase D4

Este constructor especifica el objeto de la clase asignando el argumento "fun" a la variable privada "$funcion$", y poniendo el resto de argumentos en el arreglo "$giro$" controlando de esa forma el orden en que apareceran los elementos en la tabla de multiplicación del grupo.

Operadores

Un operador acorde a [3] es una forma en la que los programadores proporcionan una notación más conveniente y convencional para manipular objetos de la clase. Por ejemplo, uno de los operadores que se sobrecargaron en la clase D4 es $*$ el cual es la notación que le damos al producto del grupo D_4 y está definido por la tabla 1.1.

- El primer operador sobrecargado es *, el cual como se mencionó antes representa el producto del grupo D_4 y se implementó como en muestra en la figura 2.6

```
D4 D4::operator*(const D4 & k) const
{
    if(funcion(k.funcion(1))==1)
    {
        if(funcion(k.funcion(2))==2)
            return id;
        else
            return f2;
    }
    else if(funcion(k.funcion(1))==2)
    {
        if(funcion(k.funcion(2))==3)
            return g3;
        else
            return f1;
    }
    else if(funcion(k.funcion(1))==3)
    {
        if(funcion(k.funcion(2))==4)
            return g2;
        else
            return f4;
    }
    else if(funcion(k.funcion(1))==4)
    {
        if(funcion(k.funcion(2))==3)
            return f3;
        else
            return g1;
    }
}
```

Figura 2.6: Implementación del operador * de la clase D4

Debido a que los elementos del grupo fueron implementados como funciones, su producto se lo implementó como la composición de funciones. Así, para la implementación de este operador se evaluó

la composición de las dos funciones en 1 y 2, se y retornó la función cuyo valor en 1 y 2 coinciden con los anteriores.

Notar que solo se necesitó la evaluación en 1 y 2, pues los valores de las funciones al evaluarlas solo coinciden con otra en un valor de su dominio. Por ejemplo, las funciones g_3 y f_1 tienen el mismo valor al evaluarlas en 1, sin embargo, sus valores en el resto del dominio son distintos, esta es la razón por la que solo se necesitó de la comparación en otro valor adicional al 1, en este caso se escogió el 2. Además, tener en cuenta que se pudo escoger cualquier par de valores del dominio, se escogió 1 y 2 por conveniencia.

- El segundo operador sobrecargado es $==$, el cual es booleano y nos indica si dos elementos del grupo son iguales o no. Este operador se implementó como en la figura 2.7. Para esta implementación se usó la definición de igualdad de funciones. Es decir, comprobar la igualdad de la funcion evaluada punto por punto en todos los elementos de su dominio, en este caso y al igual que antes fue suficiente comprobar dicha igualdad en 1 y 2.

```
bool D4::operator==(const D4 & k) const
{
    if(funcion(1)==(k.funcion)(1) && funcion(2)==(k.funcion)(2))
            return true;
    else
            return false;
}
```

Figura 2.7: Implementación del operador == de la clase D4

- El tercer operador sobrecargado es $<<$, el cual nos muestra en pantalla la tabla de l producto del grupo D_4, la cual tendrá el orden que se le haya asignado al declarar el objeto de la clase D4. Para la sobrecarga de este operador se utilizó una función auxiliar denominada "$decode$".

  Esta función recibe como argumentos dos tipos de datos int a y b, y retorna el nombre del elemento del grupo D_4 el cual evaluado en 1 y 2 coincide con a y b respectivamente. La implementación de esta

función es similar a la sobrecarga del operador * y se muestra en la figura 2.8.

```
string decode(int a, int b)
{
    if(a==1)
    {
        if(b==2)
            return "id";
        else
            return "f2";
    }
    else if(a==2)
    {
        if(b==3)
            return "g3";
        else
            return "f1";
    }
    else if(a==3)
    {
        if(b==4)
            return "g2";
        else
            return "f4";
    }
    else if(a==4)
    {
        if(b==1)
            return "g1";
        else
            return "f3";

    }
}
```

Figura 2.8: Implementación de la función "*decode*" de la clase D4

La implementación del operador $<<$ se muestra en la figura 2.9 y consiste en crear dos variables a y b de tipo *int*. Luego, en un loop a estas dos variables se les asignan los valores de cada función en el arreglo "*giro*" evaluadas en 1 y 2, para después gracias a la función "*decode*" imprimir ordenadamente el nombre de cada elemento del arreglo "*giro*", obteniendo así el encabezado de la tabla. De forma similar se imprime la primera columna. Finalmente, usando un doble loop, a las variables a y b se les asignan los valores de las composiciones de las funciones en el arreglo "*giro*" evaluadas en 1 y 2 para posteriormente imprimir el nombre de cada una de estas

composiciones gracias a la función "*decode*".

```
ostream& operator<<(ostream& os, const D4& k)
{
     int a,b;
     os<<"␣␣␣"<<'|';
     //Se imprime la primera linea
     for(int j=0;j<8;j++)
    {
        a=k.giro[j](1);
        b=k.giro[j](2);
        os<<decode(a,b)<<"␣␣";
    }
    os<<"\n";
    os<<"-----------------------------------";
    os<<"\n";
    for(int j=0;j<8;j++)
    {
        // Se imprime la primera columna
        a=k.giro[j](1);
        b=k.giro[j](2);
        os<<decode(a,b)<<"␣"<<'|';
        for(int i=0;i<8;i++)
        {
            //Se imprime la multiplicacion de las funciones
            a=k.giro[j](k.giro[i](1));
            b=k.giro[j](k.giro[i](2));
            os<<decode(a,b)<<"␣␣";
        }
        os<<"\n";
    }
    return os;
}
```

Figura 2.9: Implementación del operador $<<$ de la clase D4

Otras funciones miembro

Para que la clase D4 esté acorde a la definición de grupo es necesario implementar funciones miembro adicionales, como el inverso de un objeto "x" de la clase D4, que representa el inverso del elemento del mismo nombre del grupo D_4.

- La primera función miembro adicional es la denominada "*inverso*", la cual nos retorna un objeto de la clase D4 que representa el inverso de un determinado elemento del grupo D_4. Su implementación se muestra en la figura 2.10.

```
D4 D4::inverso() const
{
for(int i=0;i<=7;i++)
    if(funcion(giro[i](1))==1 && funcion(giro[i](2))==2)
        return giro[i];
}
```

Figura 2.10: Implementación de la función "*inverso*" de la clase D4

Esta implementación consiste en evaluar la composición de la función de la cual queremos su inverso y cada función en el arreglo "*giro*" en 1 y 2. Luego, se elige la función en el arreglo con la cual la composición evaluada en 1 y 2, sea igual a 1 y 2. Es decir, se elige la función en el arreglo con la cual la composición es igual a la "*id*". Notar que al igual que antes, solo es suficiente evaluar las funciones en 1 y 2 para establecer la igualdad entre funciones.

- La segunda función miembro adicional es la denominada "*imprimir*", la cual muestra en pantalla el nombre del objeto de la clase D4 a la que se le aplica esta función. Su implementación se muestra en la figura 2.11.

```
void D4::imprimir()
{
    int a,b;
    a=funcion(1);
    b=funcion(2);
    cout<<decode(a,b)<<"\n";
}
```

Figura 2.11: Implementación de la función "*imprimir*" de la clase D4

Esta implementación consiste en crear dos variables de tipo *int* a y b, a las cuales se les asignan los valores de la función del objeto de la clase D4 evaluada en 1 y 2, respectivamente. Luego se hace uso de la función "*decode*" para mostrar el nombre de dicho objeto.

- La tercera y última función miembro adicional es la denominada "*esta_en*", la cual indica si un objeto de la clase D4 está en un vector de objetos de D4. Esta función es la forma de comprobar si un elemento del grupo D_4 pertenece o no a un subconjunto de dicho grupo. Gracias a esto posteriormente podremos establecer si un subconjunto de D_4 es un subgrupo.

```
bool   D4::esta_en(vector<D4> v)
{
    for(int i=0;i<v.size();i++)
        if (funcion == (v[i]).funcion)
            return true;
}
```

Figura 2.12: Implementación de la función "*esta_en*" de la clase D4

La implementación de esta función se puede visualizar en la figura 2.12. La cual consiste en recibir como argumento un vector v de elementos de D4, para luego gracias al operador " == " se compara la función del objeto de D4 el cual se quiere comprobar si está en v, con cada una de las funciones en dicho vector, retornando verdadero si coincide con alguna función de v.

Funciones externas a la clase D4

Ya se mostraron todos los elementos y funciones que compone la clase D4 para que cumpla con la definición de grupo 1.1, ahora se mostrarán dos funciones que no pertenecen a la clase D4 pero que ayudan a definiciones que tienen relación con la teoría de grupos como las de subgrupo y subgrupo normal.

- La primera de estas funciones es la denominada "*es_subgrupo*", la cual indica si un vector de objetos de la clase D4 es un subgrupo de D_4.

```
bool es_subgrupo(vector<D4> w)
{
     int p=0;
       for(int i=0;i<w.size();i++)
           for(int j=0;j<w.size();j++)
                 if ((w[i]*w[j]).esta_en(w))
                     p++;
    if(p==w.size()*w.size())
        return true;
}
```

Figura 2.13: Implementación de la función es_subgrupo

La implementación de esta función se muestra en la figura 2.13. La cual recibe como argumento un vector w de elementos de D4, luego usando el resultado 1.2 se establece que w es subgrupo si este es cerrado respecto al operador *. Para ello, se inicializa una variable p de tipo int y gracias a un doble loop y a la función "$esta\_en$" se va constatanto si el producto entre los elementos de w pertenecen a w, si cada par de elementos cumple con esto a p se le suma 1. Al finalizar todas estas comprobaciones, si p es igual al cuadrado del tamaño de w, la función retorna verdadero, caso contrario retorna falso.

- La segunda de estas funciones es la denominada "$es\_subgrupo\_normal$", la cual indica si un vector de objetos de la clase D4 es un subgrupo normal de D_4.

  La implementación de esta función se muestra en la figura 2.14. La cual recibe como argumento un vector w de elementos de D4, luego usando la definición 1.8 se establece si w es subgrupo normal. Para ello, se crea todos los elementos del grupo D_4 y un vector K donde se ponen todos estos elementos. Despúes usando la función anterior primero se verifica si w es subgrupo, pues de no serlo, inmediatamente retorna falso.

  Por el contrario, si w es subgrupo se inicializa una variable p de tipo int y gracias a un doble loop y a las funciones "$inverso$" y "$esta\_en$" se va constatanto si el producto entre los elementos de K con los elementos de w y el inverso de los elementos de K pertenecen a w, si cada tripleta de factores cumple con esto, a p se le suma 1.

  Al finalizar todas estas comprobaciones, si p es igual a la multiplicación del tamaño de w con el tamaño de K, la función retorna verdadero, caso contrario retorna falso.

```
bool es_subgrupo_normal(vector<D4> w)
{
    D4 i(id);
    D4 g1g1(g1);
    D4 gg2(g2);
    D4 gg3(g3);
    D4 ff1(f1);
    D4 ff2(f2);
    D4 ff3(f3);
    D4 ff4(f4);

    vector<D4> K;
    K.push_back(g1g1);
    K.push_back(gg2);
    K.push_back(i);
    K.push_back(gg3);
    K.push_back(ff1);
    K.push_back(ff2);
    K.push_back(ff3);
    K.push_back(ff4);

    if(es_subgrupo(w))
    int p=0;
    {
    for (int j=0; j<K.size(); j++)
        for(int i=0;i<w.size();i++)
          if ((K[j]*w[i]*(K[j]).inverso()).esta_en(w))
                p++;
    if(p==w.size()*K.size())
        return true;
    else
        return false;
    }
    else
        return false;
}
```

Figura 2.14: Implementación de la función es_subgrupo_normal

2.1.2. Grupo U_8

Como se estudió en el capítulo anterior para definir el grupo U_8 es necesario el máximo común divisor, el cúal se lo implementó siguiendo el *Algoritmo de Euclides* que se encuentra en el teorema 4.1 del libro [2] y se muestra en la figura 2.15.

```
int mcd(int a,int b)
{
if(b==0)
        return a;
else
{
   int r_1=a % b;

   if(r_1==0)
     return b;
   else
   {
    int q=b/r_1;
    int r_2=b % r_1;
    int r=r_2 % r_1;
    while (r!=0)
        {
       q=r_1/r_2;
        r=r_1 % r_2;
        r_1=r_2;
        r_2=r;
        }
return abs(q*r_2+r_1);
   }
}
}
```

Figura 2.15: Implementación del máximo común divisor

Definición del grupo U_8

Al igual que el grupo D_4 creamos el grupo U_8 usando una clase como se puede observar en la figura 2.16.

La parte privada de la clase U8 está conformada por dos variables de tipo int, la primera denominada "j" la cual representa el número que es elemento del grupo U_8. Y la segunda es la llamada "n" la cual es igual a 8 ya que alrededor de 8 se construye el grupo, cabe resaltar que podremos crear cualquier otro grupo U_k solo tomando $n = k$.

La parte privada de la clase U8 está conformada por un constructor, dos operadores y otras cinco funciones miembro.

```
class U8
{
public:
//Constructor
    U8(int );
//Operadores
    U8 & operator=(const U8&) ;
    U8 operator*(const U8&) const;
// Otras funciones miembro
    U8 inverso() const;
    void imprimir() const;
    static void lista_elementos();
    static void tabla();
    bool esta_en(vector<U8> );
private:
    int unsigned j;
    static const unsigned int n=8;
};
```

Figura 2.16: Implementación de la clase U8

Constructor

En este caso solo se tiene un constructor y fue implementado como se muestra en la figura 2.17. Este constructor para inicializar un elemento de la clase U8 recibe un argumento de tipo int denominado "arg_j".

```
U8::U8(int arg_j)
{
if( mcd(arg_j,n)!=1)
    j=1;
else
    j=arg_j;
}
```

Figura 2.17: Implementación del constructor de la clase U8

Recordemos que el grupo U_8 es el conjunto de los números primos relativos a 8 y que son menores a 8. Así, este constructor especifica el objeto de la clase U_8 asignando el argumento "arg_j a la variable privada j si este es primo relativo a 8, caso contrario a j se le asigna 1 que es el elemento neutro del grupo U_8.

Operadores

- El primer operador sobrecargado es *, el cual representa el producto del grupo U_8 y se que implementó como en muestra en la figura 2.18.

```
U8 U8::operator*(const U8& k)const
{
U8 result(1);
result.j=(j*k.j)%n;
return result;
}
```

Figura 2.18: Implementación del operador * de la clase U8

Este operador recibe y retorna un tipo de dato U8. Para ello, primero se crea un objeto de U8 denominado "$result$" el cual es construido con el elemento neutro 1 como argumento. Luego coincidiendo con la definición del producto del grupo U_8 dada en 1.1, se asigna a la variable privada j de $result$ el residuo de la división entre el producto de las variables privadas j de los objetos que se están multiplicando con la variable privada n. Finalmente, el operador devuelve el objeto de U8 $result$.

- El segundo operador sobrecargado es $==$, el cual es booleano y nos indica si dos elementos del grupo son iguales o no. Este operador se implementó como en la figura 2.19.

```
bool U8::operator==(const U8 & k) const
{
    if(j== k.j)
        return true;
    else
        return false;
}
```

Figura 2.19: Implementación del operador == de la clase U8

Este operador retorna verdadero si las variables privadas j de los objetos comparados son iguales. Caso contrario retorna falso.

Otras funciones miembro

- La primera función miembro adicional es la denominada "$inverso$", la cual nos retorna un objeto de la clase U8 que representa el inverso de un determinado elemento del grupo U_8. Su implementación se muestra en la figura 2.20.

```
U8 U8 :: inverso () const
{
    for(int i=1; i<=n;i++)
        if((i*j) % n==1)
            return U8(i);
}
```

Figura 2.20: Implementación de la función "$inverso$" de la clase U8

Esta implementación consiste en buscar el número i entre 1 y 8 tal que el residuo de la división entre la multiplicación de i por j y n sea igual a 1. Una vez encontrado este número la función retorna un objeto de la clase de U8 creado con dicho número como su argumento.

- La segunda función miembro adicional es la denominada "$imprimir$", la cual muestra en pantalla el elemento de la clase U8 al que se le aplica esta función. Su implementación se muestra en la figura 2.21.

```
void U8::imprimir() const
{
cout << j;
}
```

Figura 2.21: Implementación de la función "$imprimir$" de la clase U8

Esta implementación consiste en mostrar en pantalla la variable privada j que almacena el elemento del grupo U_8.

- La tercera función miembro adicional es la denominada "$esta\_en$", la cual indica si un objeto de la clase U8 está en un vector de objetos de U8. Esta función ayuda a comprobar si un elemento del grupo U_8 pertenece o no a un subconjunto de dicho grupo. Gracias a esto posteriormente podremos establecer si un subconjunto de U_8 es un subgrupo.

```
bool   U8::esta_en(vector<U8> v)
{
    for(int i=0;i<v.size();i++)
        if (j== (v[i]).j)
            return true;
}
```

Figura 2.22: Implementación de la función "$esta\_en$" de la clase U8

La implementación de esta función se puede visualizar en la figura 2.22. De forma similar que en la clase D4, esta función consiste en recibir como argumento un vector v de elementos de U8 , luego gracias al operador " == " se compara el objeto de U8 el cual se quiere comprobar si está en v, con cada una de los elementos en dicho vector, retornando verdadero si coincide con alguno de estos.

Funciones externas a la clase U_8

Ya se mostraron todos los elementos y funciones que compone la clase U8 para que cumpla con la definición de grupo 1.1, ahora se mostrarán dos funciones que no pertenecen a la clase U8 pero que ayudan a definiciones que tienen relación con la teoría de grupos como las de subgrupo y subgrupo normal.

- La primera de estas funciones es la denominada "$es\_subgrupo$", la cual indica si un vector de objetos de la clase U8 es un subgrupo del grupo U_8.

  La implementación de esta función se muestra en la figura 2.23. La cual recibe como argumento un vector w de elementos de U8, luego usando el resultado 1.2 se establece que w es subgrupo si este es cerrado respecto al operador *. Para ello, se inicializa una variable p de tipo int y gracias a un doble loop y a la función "$esta\_en$" se va constatanto si el producto entre los elementos de w pertenecen a w, si cada par de elementos cumple con esto, a p se le suma 1. Al finalizar todas estas comprobaciones, si p es igual al cuadrado del tamaño de w, la función retorna verdadero caso contrario retorna falso.

```
bool es_subgrupo(vector<U8> w)
{
    int p=0;
   if((w[0].inverso()*w[0]).esta_en(w)==1)
   {
       for(int i=0;i<w.size();i++)
           for(int j=0;j<w.size();j++)
               if ((w[i]*w[j]).esta_en(w))
                   p++;
    if(p==w.size()*w.size())
        return true;
   }
    else
        return false;
}
```

Figura 2.23: Implementación de la función "*es_subgrupo*" de la clase U8

- La segunda de estas funciones es la denominada "*es_subgrupo_normal*", la cual indica si un vector de objetos de la clase U8 es un subgrupo normal del grupo U_8.

  La implementación de esta función se muestra en la figura 2.24. La cual recibe como argumento un vector w de elementos de U8, luego usando la definición 1.8 se establece si w es subgrupo normal. Para ello, se crean todos los elementos del grupo U_8 y un vector U donde se ponen todos estos elementos haciendo uso de la función $push\_back$ de la clase vector. Despúes usando la función anterior primero se verifica si w es subgrupo, pues si no lo es inmediatamente retorna falso.

  Por el contrario, si w es subgrupo se inicializa una variable p de tipo int y gracias a un doble loop y a las funciones "*inverso*" y "*esta_en*" se va constatanto si el producto entre los elementos de U con los elementos de w y el inverso de los elementos de U pertenecen a w, si cada tripleta de factores cumple con esto a p se le suma 1.

  Al finalizar todas estas comprobaciones, si p es igual a la multiplicación del tamaño de w con el tamaño de U, la función retorna verdadero caso contrario retorna falso.

```
bool es_subgrupo_normal(vector<U8> w)
{
    U8 uno(1);
    U8 tres(3);
    U8 cinco(5);
    U8 siete(7);

    vector<U8> U;
    U.push_back(uno);
    U.push_back(tres);
    U.push_back(cinco);
    U.push_back(siete);

 int p=0;
    if(es_subgrupo(w))
    {
    for (int j=0; j<U.size(); j++)
        for(int i=0;i<w.size();i++)
          if ((U[j]*w[i]*(U[j]).inverso()).esta_en(w))
                p++;
    if(p==w.size()*U.size())
        return true;
    else
        return false;
    }
    else
        return false;
}
```

Figura 2.24: Implementación de la función "*es_subgrupo_normal*" de la clase U8

Capítulo 3

Resultados, conclusiones y recomendaciones

3.1. Resultados

En este capítulo se expondrán los resultados obtenidos del trabajo realizado en los capítulos anteriores. Para ello, se realizan varías pruebas que comprueben que las implementaciones hechas cumplan con las definiciones de cada uno de los grupos mostrados en el capítulo 1, además de determinar si un conjunto es subgrupo o subgrupo normal.

3.1.1. Grupo D_4

En esta parte se hacen las comprobaciones necesarias para que la clase D4 cumpla con la definición del grupo D_4. Todas las funciones y operadores implementados en el capítulo 2 nos serán de ayuda para dichas comprobaciones. En la figura 3.1 se detallan las comprobaciones implementadas.

Para estas comprobaciones primero se crean los ocho elementos del grupo D_4.

```
    int main()
{
    D4 i(id);
    D4 gg1(g1);
    D4 gg2(g2);
    D4 gg3(g3);
    D4 ff3(f3);
    D4 ff(f1);
    D4 ff2(f2);
    D4 ff4(f4);

    cout<<"Resultados \n";

    cout<<"Comprobacion del operador *\n";
    cout<<"g1 * f1 = ";
    (gg1*ff).imprimir();

    cout<<"Comprobacion del operador ==\n";
    cout<<"1.g2*id es igual a g2? "<< ((gg2 * i) == gg2)<<"\n";
    cout<<"2.f2*f1 es igual a f4? "<< ((ff2 * ff) == ff4)<<"\n";

    cout<<"Comprobacion del inverso\n";
    cout<<"1.El inverso de g3 es: ";
    gg3.inverso().imprimir();
    cout<<"2.El inverso de f3 es: ";
    ff3.inverso().imprimir();

    cout<<"Comprobacion de la Clausura de D4 y del operador<<";
    cout<<"1.La tabla en cierto orden\n";
    cout<<i;
    cout<<"2.La tabla en otro orden\n";
    D4 dd(f1,g1,id,g2,g3,f3,f4,f2,f1);
    cout<<dd;

    vector<D4> H5;
    H5.push_back(i); H5.push_back(ff4);

    vector<D4> H8;
    H8.push_back(i);
    H8.push_back(ff2);
    H8.push_back(gg2);
    H8.push_back(ff4);

    vector<D4> H;
    H.push_back(i); H.push_back(gg1); H.push_back(ff);

    cout<<"Comprobacion de la nocion de pertenencia\n";
    cout<<"1.f1*f4*f1 esta en H5? "<<(ff*ff4*ff).esta_en(H5)<<"\n";
    cout<<"2.f2 esta en H8? "<<ff2.esta_en(H8)<<"\n";
    cout<<"3.g1*f1 esta en H? "<<(gg1*ff).esta_en(H)<<"\n";

    cout<<"Comprobacion de la nocion de subgrupo \n";
    cout<<"1.H5 es un subgrupo de D4? "<<es_subgrupo(H5)<<"\n";
    cout<<"2.H8 es un subgrupo de D4? "<<es_subgrupo(H8)<<"\n";
    cout<<"3.H es un subgrupo de D4? "<<es_subgrupo(H)<<"\n";

    cout<<"Comprobacion de la nocion de subgrupo normal\n";
    cout<<"1.H5 es un subgrupo normal de D4? "<<es_subgrupo_normal(H5)<<"\n";
    cout<<"2.H8 es un subgrupo normal de D4? "<<es_subgrupo_normal(H8)<<"\n";
}
```

Figura 3.1: Implementación de las comprobaciones de D_4

- La primera comprobación es la de la operación *, en ella se toma al azar dos elementos y se los multiplica. En este caso, se tomaron los elementos g_1 y f_1 cuya multiplicación es igual a f_2, lo cual se puede comprobar en la tabla 1.1 y lo cual se cumple como se muestra en la figura 3.2.

- La segunda comprobación es la del operador ==, en ella se preguntan si $g_2 * id$ es igual a g_2 y si $f_2 * f_1$ es igual a f_4. Lo cual es verdadero y falso, respectivamente, esto se puede comprobar en la tabla 1.1 y en efecto se cumple como se muestra en la figura 3.2.

- La tercera comprobación es la de la existencia del inverso, en ella se pide el inverso de dos elementos g_3 y f_3. Los cuales son g_1 y f_3, respectivamente, lo cual se puede comprobar en la tabla 1.1 y en efecto se cumple como se muestra en la figura 3.2.

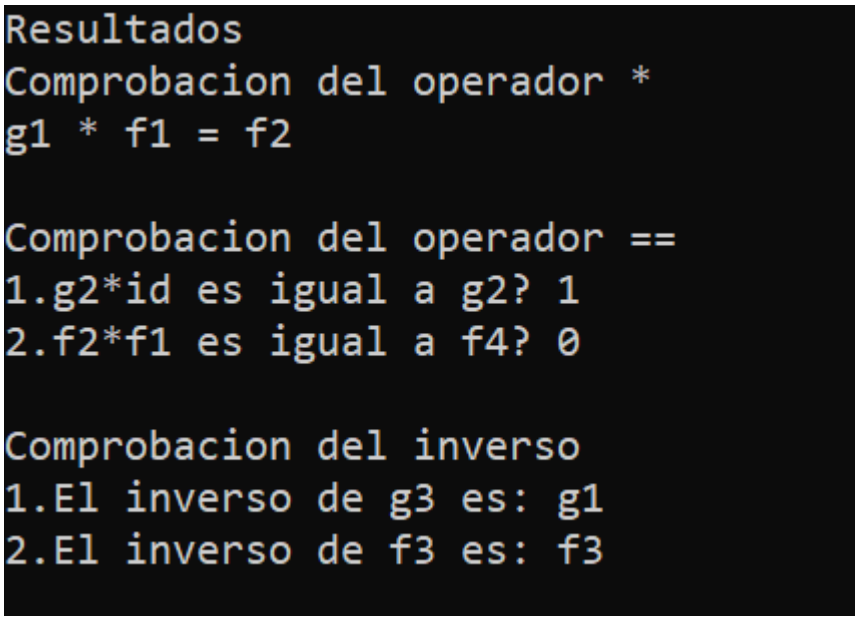


Figura 3.2: Comprobaciones de los operadores *, == y el inverso del grupo D_4

- La cuarta comprobación se muestra en la figura 3.3 y es la de la Clausura de D_4 y el operador $<<$. En ella, gracias al operador $<<$ se muestra en pantalla la tabla del producto del grupo D_4 como en la tabla 1.1, y en otro orden haciendo uso del segundo constructor de la clase D4.

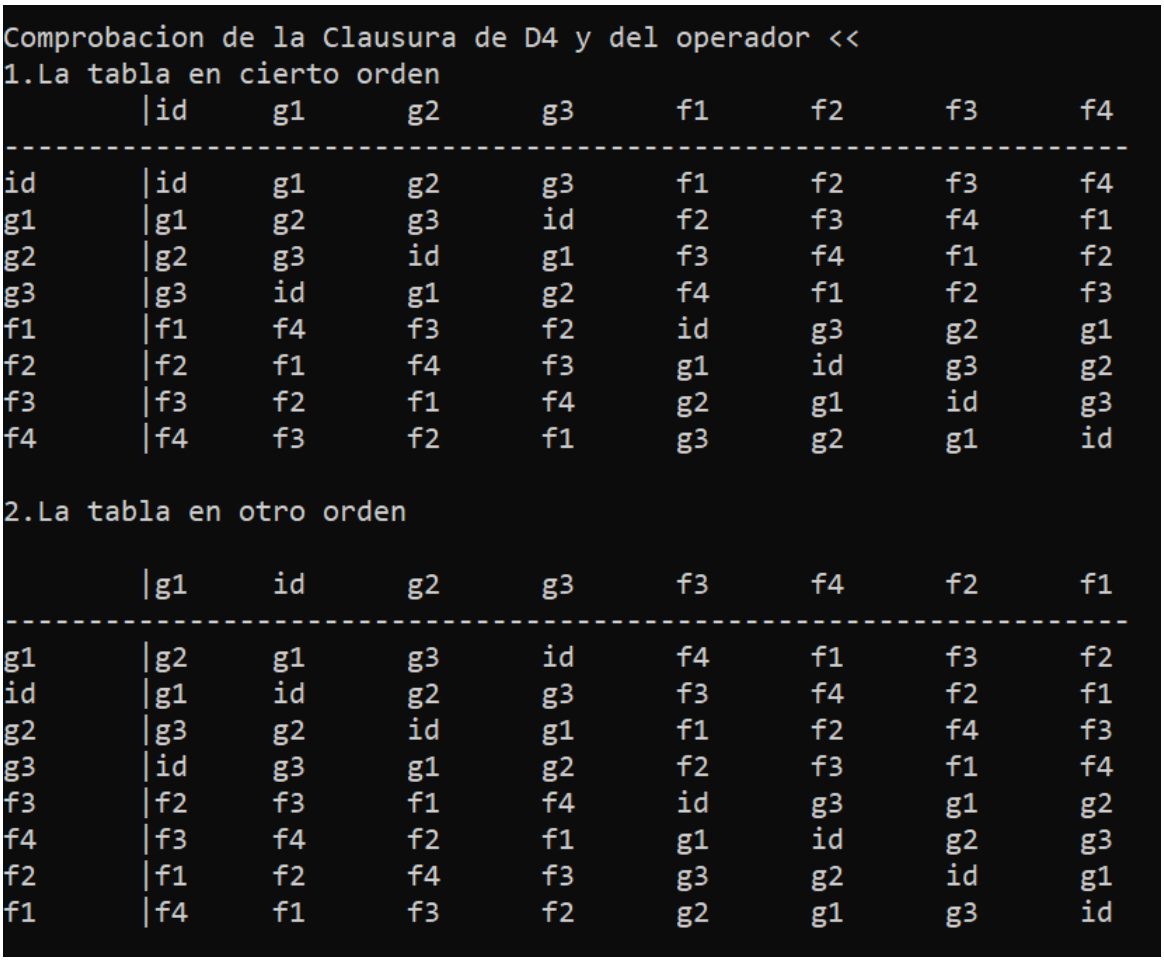


Figura 3.3: Comprobación de la Clausura de D_4 y el operador $<<$

Para las próximas comprobaciones se crean tres vectores de elementos de D4, los cuales representan subconjuntos del grupo D_4. Estos subconjuntos son los siguientes.

$$H_5 = \{id, f_4\}, \quad H_8 = \{id, f_2, g_2, f_4\}, \quad H = \{id, g_1, f_1\}$$

- La quinta comprobación se muestra en la figura 3.4 y es la de la noción de pertenencia. En esta, se pregunta si $f_1 * f_4 * f_1$ está en H_5, si f_2 está en H_8 y si $g_1 * f_1$ está en H, lo cual es falso, verdadero y falso, respectivamente. Pues $f_1 * f_4 * f_1 = g_1 * f_1 = f_2$ y coincide con lo que se muestra en la tabla 1.1.
- La sexta comprobación se muestra en la figura 3.4 y es la de la noción de subgrupo. En esta, se pregunta si H_5, H_8 y H son subgrupos, lo cual es verdadero en los dos primeros casos y falso en el tercero.

Los dos primeros casos se corroboran en el capítulo 1 en la figura 1.9. Y el tercero es falso pues como se expuso antes, $g_1 * f_1$ no está en H por lo tanto H no es cerrado respecto al producto de D_4 y de acuerdo a las definiciones 1.7 y 1.1 H no es subgrupo de D_4.

- La última comprobación se muestra en la figura 3.4 y es la de la noción de subgrupo normal. En esta, se pregunta si los subconjuntos que antes se comprobaron que eran subgrupos, son subgrupos normales. Es decir, si H_5 y H_8 son subgrupos normales. Lo cual es falso y verdadero, respectivamente. El segundo caso coincide con lo que se enunció en el capítulo 1 en la figura 1.12. Por otro lado, el primero se sigue debido a que f_1 es su propio inverso y como se mostró con anterioridad $f_1 * f_4 * f_1$ no está en H_5, por lo tanto este subgrupo no cumple con la definición 1.8, la de subgrupo normal .

```
Comprobacion de la nocion de pertenencia
1.f1*f4*f1 esta en H5? 0
2.f2 esta en H8? 1
3.g1*f1 esta en H? 0

Comprobacion de la nocion de subgrupo
1.H5 es un subgrupo de D4? 1
2.H8 es un subgrupo de D4? 1
3.H es un subgrupo de D4? 0

Comprobacion de la nocion de subgrupo normal
1.H5 es un subgrupo normal de D4? 0
2.H8 es un subgrupo normal de D4? 1
```

Figura 3.4: Comprobaciones de las nociones de pertenencia, subgrupo y subgrupo normal de D_4

3.1.2. Grupo de Klein

En esta parte se hacen las comprobaciones necesarias para que la clase Klein cumpla con la definición del grupo de Klein. Las funciones implementadas en el capítulo 2 serán de ayuda para dichas comprobaciones. En la figura 3.5 se detallan la implementación de dichas comprobaciones.

```
    int main()
{
    Klein i(id);
    Klein rr(r);
    Klein vv(v);
    Klein hh(h);

    cout<<"Resultados␣\n";

    cout<<"Comprobacion␣del␣operador␣*\n";
    cout<<"r␣*␣h␣=␣";
    (rr*hh).imprimir();

    cout<<"Comprobacion␣del␣operador␣==␣\n";
    cout<<"1.h*id␣es␣igual␣a␣h?␣"<< ((hh*i) == hh)<<"\n";
    cout<<"2.h*r␣es␣igual␣a␣r?␣"<< ((hh*rr) == rr)<<"\n";
    cout<<"3.v*r␣es␣igual␣a␣r*v?␣"<< ((vv*rr) == (rr*vv))<<"\n";

    cout<<"Comprobacion␣del␣inverso\n";
    cout<<"1.El␣inverso␣de␣v␣es:␣";
    vv.inverso().imprimir();
    cout<<"2.El␣inverso␣de␣r␣es:␣";
    rr.inverso().imprimir();

    cout<<"Comprobacion␣de␣la␣Clausura␣y␣del␣operador␣<<\n";
    cout<<"1.La␣tabla␣en␣cierto␣orden\n";
    cout<<i;
    cout<<"2.La␣tabla␣en␣otro␣orden\n";
    Klein dd(r,h,id,v,r);
    cout<<dd;

    vector<Klein> Kl;
    Kl.push_back(i);
    Kl.push_back(hh);

    vector<Klein> K;
    K.push_back(i);
    K.push_back(rr);
    K.push_back(vv);

    cout<<"Comprobacion␣de␣la␣nocion␣de␣pertenencia\n";
    cout<<"1.id␣esta␣en␣Kl?␣"<<i.esta_en(Kl)<<"\n";
    cout<<"2.r␣*␣v␣esta␣en␣K?␣"<<(rr*vv).esta_en(K)<<"\n";

    cout<<"Comprobacion␣de␣la␣nocion␣de␣subgrupo␣\n";
    cout<<"1.Kl␣es␣un␣subgrupo␣del␣grupo␣de␣Klein?␣"<<es_subgrupo(Kl)<<"\n";
    cout<<"2.K␣es␣un␣subgrupo␣del␣grupo␣de␣Klein?␣"<<es_subgrupo(K)<<"\n";

    cout<<"Comprobacion␣de␣la␣nocion␣de␣subgrupo␣normal\n";
    cout<<"1.Kl␣es␣un␣subgrupo␣normal␣del␣grupo␣de␣Klein?
␣␣␣␣"<<es_subgrupo_normal(Kl)<<"\n";
    cout<<"2.K␣es␣un␣subgrupo␣normal␣del␣grupo␣de␣Klein?
␣␣␣␣"<<es_subgrupo_normal(K)<<"\n";
}
```

Figura 3.5: Implementación de las comprobaciones del grupo de Klein

De forma similar al grupo D_4, primero se crean los cuatro elementos del grupo de Klein.

- La primera comprobación es la de la operación *, en esta realiza el producto entre los elementos r y h el que es igual a v y se puede comprobar en la tabla 1.2 y lo cual en efecto se cumple como se muestra en la figura 3.6.

- La segunda comprobación es la del operador ==, en ella se preguntan si $h*id$ es igual a h, si $h*r$ es igual a r y si $v*r$ es igual a $r*v$. Lo cual es verdadero, falso y verdadero, respectivamente, lo que en efecto se cumple como se muestra en la figura 3.6 y se puede comprobar en la tabla 1.2. Además, notar que el tercer caso es un ejemplo de que el grupo de Klein es abeliano.

- La tercera comprobación es la de la existencia del inverso, en esta se pide el inverso de dos elementos v y r. Estos son v y r, respectivamente, lo cual se cumple como se muestra en la figura 3.6 y se puede comprobar en la tabla 1.2.

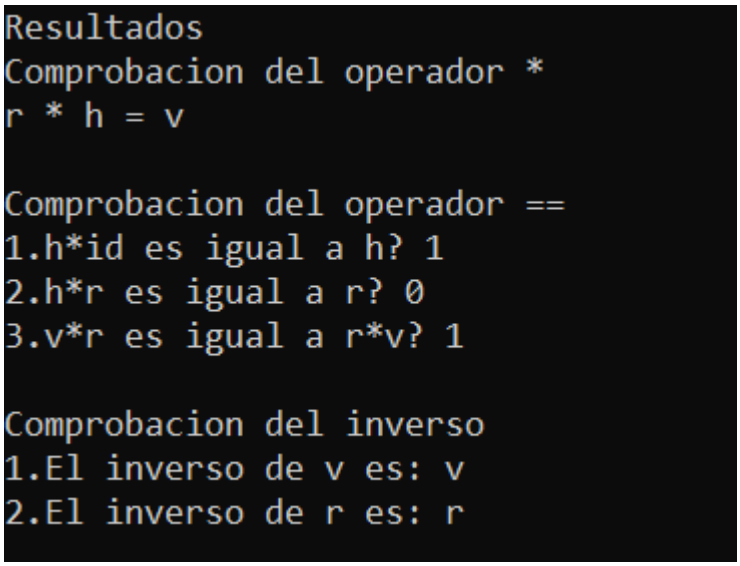


Figura 3.6: Comprobaciones de los operadores *, == y el inverso del grupo de Klein

- La cuarta comprobación se muestra en la figura 3.7 y es la de la Clausura del grupo de Klein y el operador $<<$. En en la cual, gracias al operador $<<$ se muestra en pantalla la tabla del producto del

grupo de Klein como en la tabla 1.2, y en otro orden haciendo uso del segundo constructor de la clase Klein.

```
Comprobacion de la Clausura y del operador <<
1.La tabla en cierto orden
        |id      r       v       h
--------------------------------
id      |id      r       v       h
r       |r       id      h       v
v       |v       h       id      r
h       |h       v       r       id

2.La tabla en otro orden

        |h       id      v       r
--------------------------------
h       |id      h       r       v
id      |h       id      v       r
v       |r       v       id      h
r       |v       r       h       id
```

Figura 3.7: Comprobación de la Clausura del grupo de Klein y el operador $<<$

Para las próximas comprobaciones se crean dos vectores de elementos del grupo de Klein, que representan subconjuntos de dicho grupo. Estos subconjuntos son los siguientes.

$$K_1 = \{id, h\}, \quad K = \{id, r, v\}$$

- La quinta comprobación se muestra en la figura 3.8 y es la de la noción de pertenencia. En esta, se pregunta si id está en K_1 y si $r * v$ está en K lo cual es verdadero y falso, respectivamente. Pues $r * v$ es igual a h y no pertenece a K.
- La sexta comprobación se muestra en la figura 3.8 y es la de la noción de subgrupo. En esta, se pregunta si K_1 y K son subgrupos, lo cual es verdadero en el primer caso y falso en el segundo. La veracidad del primer caso se evidencia en el capítulo 1 en la figura

1.10. Y la falsedad del segundo se debe a que $r * v$ no está en K como se expuso antes. Por tanto K no es cerrado respecto al producto del grupo de Klein y de acuerdo a las definiciones 1.7 y 1.1 K no es subgrupo del grupo de Klein.

- La última comprobación se muestra en la figura 3.8 y es la de la noción de subgrupo normal. En esta, se pregunta si los subconjuntos K_1 y K son subgrupos normales. Lo cual es verdadero y falso, respectivamente. El primer caso es debido a que todos los subgrupos del grupo de Klein son subgrupos normales, como se mencionó en el capítulo 1. Por otro lado, el segundo se sigue debido a que K no es subgrupo, por lo tanto, tampoco es subgrupo normal.

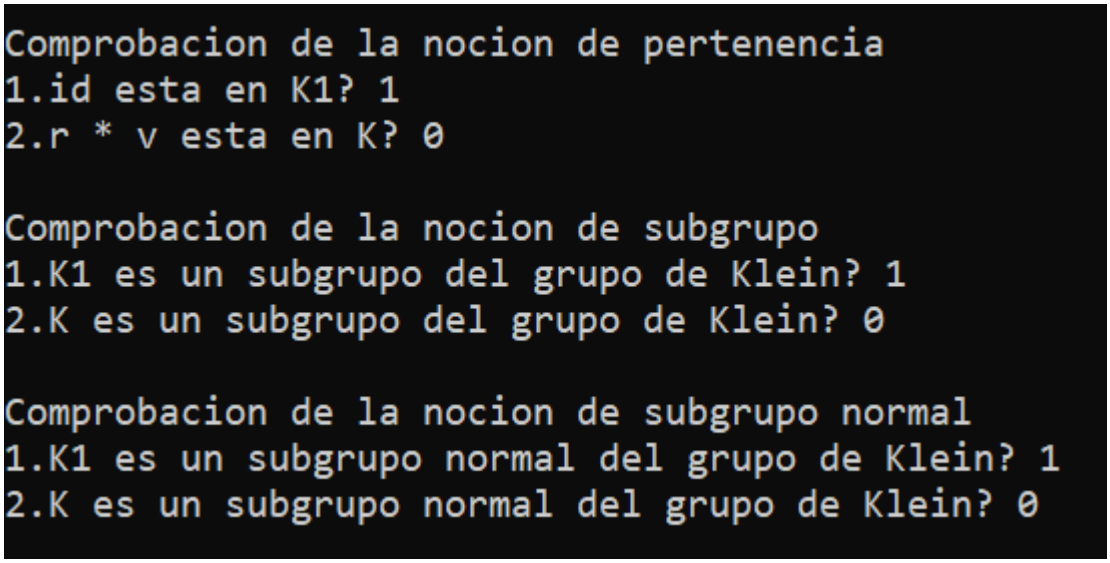


Figura 3.8: Comprobaciones de las nociones de pertenencia, subgrupo y subgrupo normal del grupo de Klein.

3.1.3. Grupo U_8

En esta parte se hacen las comprobaciones necesarias para que la clase U8 cumpla con la definición del grupo U_8. Las funciones implementadas en el capítulo 2 serán de ayuda para dichas comprobaciones. En la figura 3.9 se detallan la implementación de dichas comprobaciones.

De forma similar a los grupos anteriores, primero se crean los cuatro elementos del grupo U_8.

```
    int main()
{
    U8 uno(1);
    U8 tres(3);
    U8 cinco(5);
    U8 siete(7);

    cout<<"Resultados \n";

    cout<<"Comprobacion del operador *\n";
    cout<<"3 * 7 = ";
    (tres*siete).imprimir();

    cout<<"Comprobacion del operador == \n";
    cout<<"1.5*1 es igual a 5? "<< ((cinco*uno) == cinco)<<"\n";
    cout<<"2.7*5 es igual a 7? "<< ((siete*cinco) == siete)<<"\n";
    cout<<"3.3*5 es igual a 5*3? "<< ((tres*cinco) == (cinco*tres))<<"\n";

    cout<<"Comprobacion del inverso\n";
    cout<<"1.El inverso de 3 es: ";
    tres.inverso().imprimir();

    cout<<"2.El inverso de 7 es: ";
    siete.inverso().imprimir();

    cout<<"Comprobacion de la Clausura\n";
    uno.tabla();

    vector<U8> J1;
    J1.push_back(uno);
    J1.push_back(cinco);

    vector<U8> J;
    J.push_back(uno);
    J.push_back(tres);
    J.push_back(cinco);

    cout<<"Comprobacion de la nocion de pertenencia\n";
    cout<<"1.5 esta en J1? "<<cinco.esta_en(J1)<<"\n";
    cout<<"2.3*5 esta en J? "<<(tres*cinco).esta_en(J)<<"\n";

    cout<<"Comprobacion de la nocion de subgrupo \n";
    cout<<"1.J1 es un subgrupo de U8? "<<es_subgrupo(J1)<<"\n";
    cout<<"2.J es un subgrupo del U8? "<<es_subgrupo(J)<<"\n";

    cout<<"Comprobacion de la nocion de subgrupo normal\n";
    cout<<"1.J1 es un subgrupo normal de U8? "<<es_subgrupo_normal(J1)<<"\n";
    cout<<"2.J es un subgrupo normal de U8? "<<es_subgrupo_normal(J)<<"\n";
}
```

Figura 3.9: Implementación de las comprobaciones del grupo U_8

- La primera comprobación es la de la operación *, en esta realiza el producto entre los elementos 3 y 7 que es igual a 5 y se puede evidenciar en la tabla 1.3 y lo cual en efecto se cumple como se muestra en la figura 3.10.

- La segunda comprobación es la del operador ==, en esta se pregunta si $5 * 1$ es igual a 5, si $7 * 5$ es igual a 7 y si $3 * 5$ es igual a $5 * 3$. Esto es verdadero, falso y verdadero, respectivamente, lo que en efecto se cumple como se muestra en la figura 3.10 y se puede comprobar en la tabla 1.3. Además, notar que al igual que en el grupo de Klein, el tercer caso es un ejemplo de que el grupo de U_8 es abeliano.

- La tercera comprobación es la de la existencia del inverso, en esta se pide el inverso de dos elementos, 3 y 7. Estos son 3 y 7, respectivamente, lo cual se cumple como se muestra en la figura 3.10 y se puede corroborar en la tabla 1.3.

```
Resultados
Comprobacion del operador *
3 * 7 = 5

Comprobacion del operador ==
1.5*1 es igual a 5? 1
2.7*5 es igual a 7? 0
3.3*5 es igual a 5*3? 1

Comprobacion del inverso
1.El inverso de 3 es: 3
2.El inverso de 7 es: 7
```

Figura 3.10: Comprobaciones de los operadores *, == y el inverso del grupo U_8

- La cuarta comprobación se muestra en la figura 3.11 y es la de la Clausura del grupo U_8. En en la cual, gracias a la función miembro "$tabla()$" se muestra en pantalla la tabla del producto del grupo U_8 acorde a lo presentado en la tabla 1.3.

  De forma similar a los grupos anteriores, para las próximas comprobaciones se crean dos vectores de elementos del grupo U_8, que

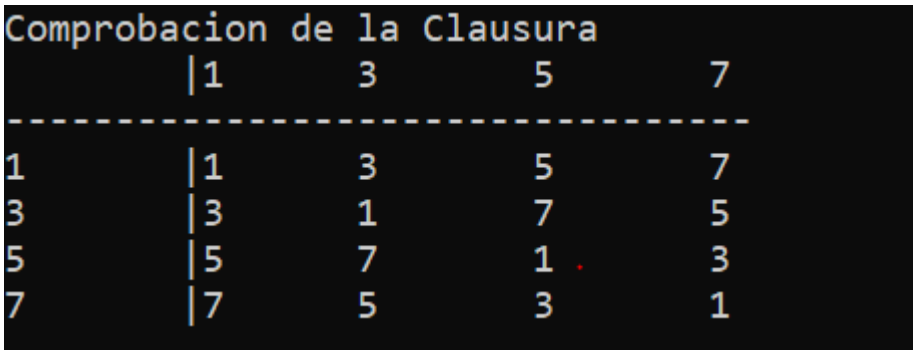


Figura 3.11: Comprobación de la Clausura del grupo U_8

representan subconjuntos del mismo. Estos subconjuntos son los siguientes.

$$J_1 = \{1, 5\}, \quad J = \{1, 3, 5\}$$

- La quinta comprobación se muestra en la figura 3.12 y es la de la noción de pertenencia. En esta, se pregunta si 5 está en J_1 y si $3 * 5$ está en J lo cual es verdadero y falso, respectivamente. Esto debido a que $3 * 5$ es igual a 7 que no pertenece a J.

- La sexta comprobación se muestra en la figura 3.12 y es la de la noción de subgrupo. En esta, se pregunta si J_1 y J son subgrupos, lo cual es verdadero en el primer caso y falso en el segundo. La veracidad del primer caso se evidencia en el capítulo 1 en la figura 1.11. Y la falsedad del segundo se debe a que $3 * 5$ no está en J como se expuso antes. Por tanto J no es cerrado respecto al producto del grupo U_8 y de acuerdo a las definiciones 1.7 y 1.1 J no es subgrupo de U_8.

- La última comprobación se muestra en la figura 3.12 y es la de la noción de subgrupo normal. En esta, se pregunta si los subconjuntos J_1 y J son subgrupos normales. Lo cual es verdadero y falso, respectivamente. Lo primero es debido a que todos los subgrupos del grupo U_8 son subgrupos normales, como se mencionó en el capítulo 1. Por otro lado, lo segundo se sigue porque J no es subgrupo y por lo tanto, tampoco es subgrupo normal.

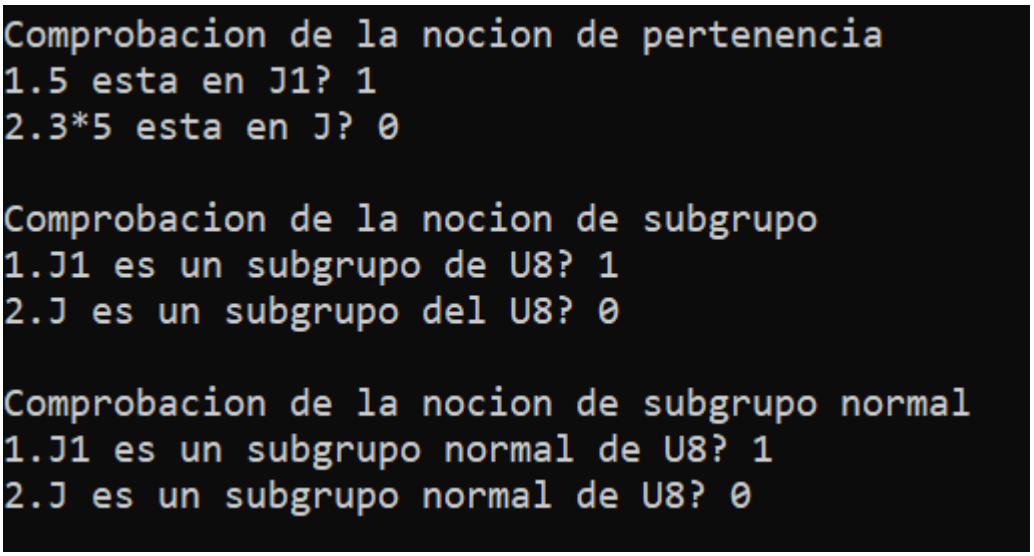


Figura 3.12: Comprobaciones de las nociones de pertenencia, subgrupo y subgrupo normal del grupo U_8.

3.2. Conclusiones y recomendaciones

3.2.1. Conclusiones

- Como se expuso anteriormente gracias a la programación orientada a objetos en el lenguaje C++ se implementaron con éxito los grupos de Klein, D_4 y U_8, cumpliendo así con el primer objetivo específico del trabajo realizado.
- La comprobaciones antes mostradas aseguran que las implementaciones realizadas de los respectivos grupos, en efecto, están en concordancia con sus definiciones en la teoría de grupos, de esta forma satisfaciendo el segundo objetivo específico planteado.
- Como resultado de todo lo relizado con anterioridad, desde la implementación de los grupos en cuestión hasta sus respectivas comprobaciones, se pudo determinar si un subconjunto es subgrupo o subgrupo normal de los grupos. Lo cual acata lo propuesto en el tercer objetivo específico.
- Finalmente, en este trabajo se materializó la definición abstracta de grupo algebraico, por medio de tres ejemplos. Lo interesante y lo nuevo de este trabajo es mostrar como funcionan estos conceptos

abstractos a través de objetos que se pueden manipular y que ayudan a una mejor comprensión de los mismos. Así, cumpliendo con conformidad el objetivo principal de este trabajo.

3.2.2. Recomendaciones

- A los interesados a ampliar lo hecho en este trabajo, se recomienda usar la herramienta de subclase en el lenguaje C++ para la implementación de grupos cuya definición es similar. Por ejemplo, los grupos de Klein y D_4, cuyos elementos son vistos como funciones, pueden ser implementados como subclases de una clase de grupos cuyos elementos sean funciones. De esta forma, optimizando las implementaciones y teniendo la posibilidad de agregar a esta clase más grupos similares como el grupo S_3.

Capítulo A

Título anexo

A continuación se anexa el código completo de la implementación.

```
#include <iostream>
#include<vector>
#include<cmath>
#include<stdio.h>
#include<string>

using namespace std;

typedef int (*funcion_de_Klein) (int);
int i(int);
int r(int);
int v(int);
int h(int);

typedef int (*funcion_de_D4) (int);
int id(int);
int g1(int);
int g2(int);
int g3(int);
int f(int);
int f2(int);
int f3(int);
int f4(int);
```

```
int mcd(int a,int b)
{
if(b==0)
    {
        return a;
    }
else
{
   int r_1=a%b;
   if(r_1==0)
    return b;
   else
    int q=b/r_1;
    int r_2=b%r_1;
    int r=r_2%r_1;
    while (r!=0)
        {
       q=r_1/r_2;
        r=r_1%r_2;
        r_1=r_2;
        r_2=r;
        }
return abs(q*r_2+r_1);
}
}
/** CLASE U8 **/
 class U8
 {
 public:
    U8(int );
    bool operator==(const U8 & k) const;
    U8 operator*(const U8&) const;
    U8 inverso() const;
    void imprimir() const;
    bool esta_en(vector<U8> );
    static void lista_elementos();
    static void tabla();

 private:
    int unsigned j;
    static const unsigned int n=8;
 };
```

```
/// Constructor con un argumento
U8::U8(int arg_j)
{
if( mcd(arg_j,n)!=1)
    j=1;
else
    j=arg_j;
}
///Funciones miembro
U8 U8 :: inverso () const
{
    for(int i=1; i<=n;i++)
        if((i*j)%n==1)
            return U8(i);
}
void U8::imprimir() const
{
cout << j;
}
void U8::tabla()
{
      cout<<"\t"<<'|';
      for(int j=1;j<=n;j++)
    {
        if(mcd(j,n) == 1)
         cout<<j%n<<"\t";
    }
    cout<<"\n";
    cout<<"--------------------------------";
    cout<<"\n";
      for (int i = 1; i <= n; i++)
        if(mcd(i,n) == 1)
        {
            cout<<i%n<<"\t"<<'|';
           for(int r = 1;r <= n;r++)
            {
            if ( mcd(r,n)==1)
                cout<< (i*r)%n<<"\t";
            }
            cout<<endl;
        }
}
```

```
/// Sobrecarga de Operadores

bool U8::operator==(const U8 & k) const
{
    if(j== k.j)
        return true;
    else
        return false;
}

U8 U8::operator*(const U8& k)const
{
U8 result(1);
result.j=(j*k.j)%n;
return result;
}

/// Funciones Externas

bool   U8::esta_en(vector<U8> v)
{
    for(int i=0;i<v.size();i++)
        if (j== (v[i]).j)
            return true;
}

bool es_subgrupo(vector<U8> w)
{
    int p=0;
   if((w[0].inverso()*w[0]).esta_en(w)==1)
   {
       for(int i=0;i<w.size();i++)
           for(int j=0;j<w.size();j++)
                if ((w[i]*w[j]).esta_en(w))
                    p++;

    if(p==w.size()*w.size())
        return true;
   }

    else
        return false;
}
```

```
bool es_subgrupo_normal(vector<U8> w)
{
    U8 uno(1);
    U8 tres(3);
    U8 cinco(5);
    U8 siete(7);

    vector<U8> U;
    U.push_back(uno);
    U.push_back(tres);
    U.push_back(cinco);
    U.push_back(siete);

 int p=0;
    if(es_subgrupo(w))
    {
    for (int j=0; j<U.size(); j++)
    {
        for(int i=0;i<w.size();i++)
        {
          if ((U[j]*w[i]*(U[j]).inverso()).esta_en(w))
                p++;
        }
    }
    if(p==w.size()*U.size())
        return true;
    else
        return false;
    }
    else
        return false;
}
```

```
/** CLASE D4 **/
   class D4
{
 public:
     D4(funcion_de_D4);
     D4(funcion_de_D4,funcion_de_D4, funcion_de_D4, funcion_de_D4,
     funcion_de_D4,funcion_de_D4, funcion_de_D4, funcion_de_D4,
     funcion_de_D4);
     bool operator==(const D4&) const;
     D4 operator*(const D4&) const;
     void imprimir();
     D4 inverso() const;
     bool esta_en(std::vector<D4> );
     static funcion_de_D4*giro;
     friend ostream& operator<<(ostream& , const D4& );
 private:
    funcion_de_D4 funcion;
};
funcion_de_D4* D4::giro = new funcion_de_D4[8];

///**Defino las funciones de D4**///
int id (int a)
{
    return a;
}

int g1 (int a)
{
    if (a==1)
        return 4;
    else if(a==4)
        return 3;
    else if(a==2 || a==3)
        return a-1;
};

int g2 (int a)
{
    if (a<=2)
        return a+2;
    else
        return a-2;
};
```

```
int g3 (int a)
{
    if (a==4)
        return 1;
    else
        return a+1;
};


int f1 (int a)
{
    if (a==1 || a==3)
        return a+1;
    else
        return a-1;
};


int f2 (int a)
{
    if (a==1 ||a==3)
        return a;
    else if(a==2)
        return 4;
    else
        return 2;
};


int f3 (int a)
{
    if (a==1)
        return 4;
    else if(a==2)
        return 3;
    else if(a==3)
        return 2;
    else
        return 1;
};
```

```
int f4 (int a)
{
    if (a==2 ||a==4)
        return a;
    else if(a==1)
        return 3;
    else
        return 1;
};

/// Constructores

D4::D4(funcion_de_D4 fun)
{
    funcion = fun;
    giro[0] = id;
    giro[1] = g1;
    giro[2] = g2;
    giro[3] = g3;
    giro[4] = f;
    giro[5] = f2;
    giro[6] = f3;
    giro[7] = f4;
}

D4::D4(funcion_de_D4 fun,funcion_de_D4 fun1,funcion_de_D4
fun2,funcion_de_D4 fun3,funcion_de_D4 fun4, funcion_de_D4
fun5,funcion_de_D4 fun6,funcion_de_D4 fun7,funcion_de_D4 fun8)
{
    funcion = fun;
    giro[0] = fun1;
    giro[1] = fun2;
    giro[2] = fun3;
    giro[3] = fun4;
    giro[4] = fun5;
    giro[5] = fun6;
    giro[6] = fun7;
    giro[7] = fun8;
}
```

```
/// Funcion Auxiliar
string decode(int a, int b)
{
    if(a==1)
    {
        if(b==2)
            return "id";
        else
            return "f2";
    }
    else if(a==2)
    {
        if(b==3)
            return "g3";
        else
            return "f1";
    }
    else if(a==3)
    {
        if(b==4)
            return "g2";
        else
            return "f4";
    }
    else if(a==4)
    {
        if(b==1)
            return "g1";
        else
            return "f3";
    }
}

/// Sobrecarga de Operadores
bool D4::operator==(const D4 & k) const
{
        if(funcion(1)==(k.funcion)(1) && funcion(2)==(k.funcion)(2))
           return true;
        else
            return false;
}
```

```
D4 D4::operator*(const D4 & k) const
{
    if(funcion(k.funcion(1))==1)
    {
        if(funcion(k.funcion(2))==2)
            return id;
        else
            return f2;
    }
    else if(funcion(k.funcion(1))==2)
    {
        if(funcion(k.funcion(2))==3)
            return g3;
        else
            return f1;
    }
    else if(funcion(k.funcion(1))==3)
    {
        if(funcion(k.funcion(2))==4)
            return g2;
        else
            return f4;
    }
    else if(funcion(k.funcion(1))==4)
    {
        if(funcion(k.funcion(2))==3)
            return f3;
        else
            return g1;
    }
}

ostream& operator<<(ostream& os, const D4& k)
{
     int a,b;
     os<<"\t"<<'|';
     for(int j=0;j<8;j++)
    {
        a=k.giro[j](1);
        b=k.giro[j](2);
        os<<decode(a,b)<<"\t";
    }
    os<<"\n";
```

```
    os<<"-----------------------------------------------------------------";
    os<<"\n";
    for(int j=0;j<8;j++)
    {
        a=k.giro[j](1);
        b=k.giro[j](2);
        os<<decode(a,b)<<"\t"<<'|';
        for(int i=0;i<8;i++)
        {
            a=k.giro[j](k.giro[i](1));
            b=k.giro[j](k.giro[i](2));
            os<<decode(a,b)<<"\t";
        }
        os<<"\n";
    }
    return os;
}

/// Funciones miembro

void D4::imprimir()
{
    int a,b;
    a=funcion(1);
    b=funcion(2);
    cout<<decode(a,b)<<"\n";
}

D4 D4::inverso() const
{
for(int i=0;i<=7;i++)
    if(funcion(giro[i](1))==1 && funcion(giro[i](2))==2)
        return giro[i];
}

/// Funciones Externas
bool   D4::esta_en(vector<D4> v)
{
    for(int i=0;i<v.size();i++)
        if (funcion == (v[i]).funcion)
            return true;

}
```

```
bool es_subgrupo(vector<D4> w)
{
     int p=0;
       for(int i=0;i<w.size();i++)
           for(int j=0;j<w.size();j++)
                 if ((w[i]*w[j]).esta_en(w))
                     p++;
    if(p==w.size()*w.size())
        return true;
}
bool es_subgrupo_normal(vector<D4> w)
{
    D4 i(id);
    D4 gg1(g1);
    D4 gg2(g2);
    D4 gg3(g3);
    D4 ff(f1);
    D4 ff2(f2);
    D4 ff3(f3);
    D4 ff4(f4);
    vector<D4> K;
    K.push_back(gg1);
    K.push_back(gg2);
    K.push_back(i);
    K.push_back(gg3);
    K.push_back(ff);
    K.push_back(ff2);
    K.push_back(ff3);
    K.push_back(ff4);
    if(es_subgrupo(w))
    {
    int p=0;
    for (int j=0; j<K.size(); j++)
        for(int i=0;i<w.size();i++)
          if ((K[j]*w[i]*(K[j]).inverso()).esta_en(w))
                p++;
    if(p==w.size()*K.size())
        return true;
    else
        return false;
    else
        return false;
}
```

```
/** CLASE KLEIN **/
    class Klein
{
 public:
     Klein(funcion_de_Klein);
     Klein(funcion_de_Klein,funcion_de_Klein, funcion_de_Klein,
     funcion_de_Klein, funcion_de_Klein);
     bool operator==(const Klein&) const;
     Klein operator*(const Klein&) const;
     void imprimir();
     Klein inverso() const;
     bool esta_en(std::vector<Klein> );
     static funcion_de_Klein*giro;
     friend ostream& operator<<(ostream& , const Klein& );
 private:
    funcion_de_Klein funcion;
};
funcion_de_Klein* Klein::giro = new funcion_de_Klein[4];

///**Defino las funciones de Klein**///

int i (int a)
{
    return a;
}

int r (int a)
{
    if (a==1 || a==2)
       return a+2;
    else
        return a-2;
};

int v (int a)
{
    if (a==1 || a==3)
        return a+1;
    else
        return a-1;
};
```

```
int h (int a)
{
    if (a==1)
        return 4;
    else if(a==2)
        return 3;
    else if(a==3)
        return 2;
    else if(a==4)
        return 1;
};

/// Constructores
Klein::Klein(funcion_de_Klein fun)
{
    funcion = fun;
    giro[0] = i;
    giro[1] = r;
    giro[2] = v;
    giro[3] = h;
}
Klein::Klein(funcion_de_Klein fun,funcion_de_Klein fun1,
funcion_de_Klein fun2,funcion_de_Klein fun3,funcion_de_Klein fun4)
{
    funcion = fun;
    giro[0] = fun1;
    giro[1] = fun2;
    giro[2] = fun3;
    giro[3] = fun4;
}

/// Funcion Auxiliar
string decode(int a)
{
    if(a==1)
        return "id";
    else if(a==2)
        return "v";
    else if(a==3)
        return "r";
    else if(a==4)
        return "h";
}
```

```
/// Sobrecarga de Operadores
bool Klein::operator==(const Klein & k) const
{
        if(funcion(1)==(k.funcion)(1))
            return true;
        else
            return false;
}

Klein Klein::operator*(const Klein & k) const
{
    if(k.funcion(funcion(1))==1)
        return i;
    else if(k.funcion(funcion(1))==2)
        return v;
    else if(k.funcion(funcion(1))==3)
        return r;
    else if(k.funcion(funcion(1))==4)
        return h;
}
ostream& operator<<(ostream& os, const Klein& k)
{
     int a;
     os<<"\t"<<'|';
     for(int j=0;j<4;j++)
    {
        a=k.giro[j](1);
        os<<decode(a)<<"\t";
    }
    os<<"--------------------------------";
    for(int j=0;j<4;j++)
    {
        a=k.giro[j](1);
        os<<decode(a)<<"\t"<<'|';
        for(int i=0;i<4;i++)
        {
            a=k.giro[j](k.giro[i](1));
            os<<decode(a)<<"\t";
        }
        os<<"\n";
    }
    return os;
}
```

```
/// Funciones miembro

void Klein::imprimir()
{
    int a;
    a=funcion(1);
    cout<<decode(a)<<"\n";
}

Klein Klein::inverso() const
{
for(int i=0;i<=3;i++)
    if(funcion(giro[i](1))==1)
        return giro[i];
}

/// Funciones Externas
bool   Klein::esta_en(vector<Klein> v)
{
    for(int i=0;i<v.size();i++)
        if (funcion == (v[i]).funcion)
            return true;
}

bool es_subgrupo(vector<Klein> w)
{
    int p=0;
   if((w[0].inverso()*w[0]).esta_en(w)==1)
   {
       for(int i=0;i<w.size();i++)
           for(int j=0;j<w.size();j++)
                if ((w[i]*w[j]).esta_en(w))
                    p++;

    if(p==w.size()*w.size())
        return true;
   }

    else
        return false;
}
```

```
bool es_subgrupo_normal(vector<Klein> w)
{
    Klein ii(i);
    Klein rr(r);
    Klein vv(v);
    Klein hh(h);

    vector<Klein> K;
    K.push_back(rr);
    K.push_back(vv);
    K.push_back(ii);
    K.push_back(hh);

 int p=0;
    if(es_subgrupo(w))
    {
    for (int j=0; j<K.size(); j++)
    {
        for(int i=0;i<w.size();i++)
        {
          if ((K[j]*w[i]*(K[j]).inverso()).esta_en(w))
                p++;
        }
    }
    if(p==w.size()*K.size())
        return true;
    else
        return false;
    }
    else
        return false;
}
```

Referencias Bibliográficas

[1] Israel Nathan Herstein. *Álgebra moderna*. Editorial Trillas, México D.F., 1980.

[2] Victor Shoup. *A Computational Introduction To Number Theory And Algebra*. Cambridge University Press, New York, 2005.

[3] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2013.