ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

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

IMPLEMENTACIÓN DE CONCEPTOS ABSTRACTOS RELACIONADOS AL GRUPO S_3 EN EL LENGUAJE DE PROGRAMACIÓN C++

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

HERNÁN PAVEL ESTRELLA GORDILLO

hpeg2009@gmail.com

DIRECTOR: EURO DE JESÚS LUCENA DELGADO

eurold.86@gmail.com

DMQ, ABRIL 2023

CERTIFICACIONES

Yo, HERNÁN PAVEL ESTRELLA GORDILLO, 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.

Firmado electrónicamente por:
HERNAN PAVEL
ESTRELLA GORDILLO

Hernán Pavel Estrella Gordillo

Certifico que el presente trabajo de integración curricular fue desarrollado por Hernán Pavel Estrella Gordillo, bajo mi supervisión.

Firmado electrónicamente por:
EURO DE JESUS
LUCENA DELGADO

Euro de Jesús 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.

Hernán Pavel Estrella Gordillo

Euro de Jesús Lucena Delgado

RESUMEN

La realización del presente trabajo se enfoca en la Teoría de Grupos, la cual es una parte de la matemática que ayuda a modelar fenómenos físicos a través de los grupos simétricos y diédricos. En el documento se presenta una implementación de grupos mediante el uso del lenguaje de programación C++. Previo a la realización de dicho programa se considera una serie de pasos preliminares para entender plenamente las definiciones relacionadas con grupos.

El punto de partida se enfoca en el análisis de la bibliografía presentada, con la cual se conceptualiza los grupos S_3 y D_3, y las definiciones de estructuras computacionales en el lenguaje C++. Luego, se presenta una implementación de cada grupo con sus respectivos operadores y mapas asociados con visualización en pantalla. Además, se muestra las diferentes abstracciones del grupo elegido mediante un menú de opciones y son observadas en pantalla.

Finalmente, gracias a los desarrollos anteriores se prueba la definición de la función de isomorfismo que se ha planteado. Igualmente, se presenta varias representaciones de las imágenes de dicha función con ingreso del usuario.

Palabras clave: Grupos, Programación, C++, Isomorfismos, POO.

ABSTRACT

This paper focuses on Group Theory, a part of mathematics that helps model physical phenomena through symmetric and dihedral groups. The article presents an implementation of groups using the C++ programming language. Before realizing such a program, a series of preliminary steps must be considered to understand the definitions related to groups.

The starting point focuses on the analysis of the presented bibliography, which conceptualizes groups S_3 and D_3, and the definition of computational structures in the C++ language. Then, the implementation of each group is presented with its respective operators and associated maps with on-screen visualization. In addition, different abstractions of the chosen group are shown through a menu of options and observed on the screen.

Finally, through the previously mentioned developments, the proposed definition of the isomorphism function can be proven. Likewise, diverse image representations of said function are presented with user input.

Keywords: Groups, Programming, C++, Isomorphisms, OOP.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

En el presente trabajo se usará la Programación Orientada a Objetos para desarrollar una aplicación que permita calcular las diferentes características que tiene el Grupo $(S_3, \circ)$, donde S_3 es el grupo de funciones generado por las permutaciones de 3 elementos y $\circ$ representa a la composición de funciones. Resulta productivo tener una calculadora de grupos puesto que nos ahorra tiempo al realizar la operación composición.

En primer lugar, se creará el ambiente para implementar las funciones asociadas al grupo S_3, como el lenguaje de C++ tiene la opción de implementar clases, se añadirá estas funciones para realizar las propiedades del grupo. En este tipo de estructura se podrá añadir constructores, operadores y funciones internas para los elementos de S_3.

A continuación, se analizará el grupo $(D_3, \circ)$ para obtener una implementación diferente a nuestro grupo de interés, pero manteniendo las estructuras de programación que se implementó con S_3. Seguidamente, se construirá un mapa entre los dos grupos, se demostrará que esta función en realidad cumple con la definición de isomorfismo.

Finalmente se elaborará un código que enlazará los grupos creados para visualizar el isomorfismo que se creó con anterioridad.

1.1. Objetivo general

Diseñar un modelo computacional del grupo S_3 con sus propiedades mediante implementaciones en C++.

1.2. Objetivos específicos

1. Desarrollar la teoría de grupos mediante conceptos algebraicos y computacionales.
2. Formular la implementación del grupo S_3 y D_3 utilizando la programación orientada a objetos en C++.
3. Ejemplificar conceptos relacionados grupos tales como asociatividad, elemento inverso, igualdad, subgrupos cíclico e isomorfismos acudiendo a lo implementado en C++.

1.3. Alcance

Las aplicaciones matemáticas implementadas mediante computadora han sido de gran utilidad para los seres humanos. Al momento existen escasas implementaciones de Teoría de Grupos en computadoras, especialmente en el lenguaje de programación C++. El objetivo de este trabajo es utilizar los conceptos matemáticos sobre grupos y de programación para realizar una implementación en el lenguaje C++.

El alcance del trabajo se limita al estudio del grupo S_3 y D_3 los cuales se definirá de acuerdo con la bibliografía consultada previamente. Se identificará las propiedades de mayor utilidad, las cuales se pueden implementarán en ambos casos, además se entrelazará los programas a través del concepto matemático de isomorfismo.

1.4. Marco teórico

Este proyecto se enfoca en la implementación del grupo $S3$ con sus propiedades en el lenguaje de programación C++, para ello es necesario tener claros varios conocimientos sobre Estructuras Algebraicas, Programación Estructurada y Programación Orientada a Objetos.

En la presente sección y en base a los libros [2],[3] y [6] se describirán los componentes necesarios sobre a Teoría de Grupos que luego se implementarán en el lenguaje C++.Además en los libros [1], [4],[5] y [7] se obtendrá información sobre la programación estructurada la cual es importante para obtener una mejor calidad y diafanidad en el desarrollo de las instrucciones en C++-.

1.4.1. Grupos

Definición 1.1. *Sea G un conjunto no vacío y la operación binaria $\odot$ llamada producto, decimos que el par $(G, \odot)$ es un* Grupo *si cumple con:*

1. *Clausura: Para todo $a, b \in G$ se tiene que $a \odot b \in G$.*
2. *Asociativa: Para todo $a, b, c \in G$ se tiene que $a \odot (b \odot c) = (a \odot b) \odot c$.*
3. *Elemento neutro: Existe un elemento $e \in G$ tal que para todo $a \in G$ se tiene que $a \odot e = e \odot a = a$.*
4. *Elemento inverso: Para todo $a \in G$, existe un elemento $a^{-1} \in G$ tal que $a \odot a^{-1} = a^{-1} \odot a = e$.*

Ejemplo 1.1. *Vamos a tomar un subconjunto de los números enteros $\mathbb{Z}$, y al cual le dotaremos de una operación producto para que el subconjunto se convierta en grupo.Así, definimos el conjunto $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$, y el siguiente mapa:*

$$+_n \colon \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$$

$$(a, b) \mapsto +_n(a, b) = \begin{cases} a + b & \textbf{si} \quad a + b < n \\ \\ a + b - n & \textbf{si} \quad a + b \geq n \end{cases}$$

Para asociar esta operación a la suma usual, vamos a mejorar su notación. Notemos $+_n(a,b) = a +_n b$*, la cual es una operación y le llamaremos* Suma modulo n*. A partir de esto, verifiquemos la definición de grupo para* $(\mathbb{Z}_n, +_n)$*:*

- *Clausura: Dados* $a, b \in \mathbb{Z}_n$*, la suma modulo* n *de estos pertenece a* $\mathbb{Z}_n$*, es decir,* $a +_n b \in \mathbb{Z}_n$*.Esto se da por la definición propia de la operación*
- *Asociativa: Dados* $a, b, c \in \mathbb{Z}$*, veriquemos* $a +_n (b +_n c) = (a +_n b) +_n c)$ *se cumpla. Por el Anexo A.1, sabemos que la asociación de la suma modulo* n *se escribe:*

$$(a +_n b) +_n c = \begin{cases} a+b+c & \textbf{si} \quad a+b+c < n \\ a+b+c-n & \textbf{si} \quad a+b \geq n \text{ y } a+b+c < 2n \\ a+b+c-2n & \textbf{si} \quad a+b+c \geq 2n \end{cases}$$

Veamos por casos

- *Si* $a+b+c < n$*, tenemos que* $b+c < n$

$$\begin{aligned}(a +_n b) +_n c &= a+b+c \\ &= a + (b +_n c).\end{aligned}$$

Y $a + (b +_n c) < n$*, por tanto*

$$a + (b +_n c) = a +_n (b +_n c).$$

En consiguiente,

$$(a +_n b) +_n c = a +_n (b +_n c). \tag{1.1}$$

- *Si* $a+b \geq n$ *y* $a+b+c < 2n$*.*
  Para este caso se supone que $b+c \geq n$*, dado que en el anterior se tenía el complemento.Así*

$$b+c-n = b +_n c$$

Por tanto,

$$\begin{aligned}(a +_n b) +_n c &= a + b + c - n \\ &= a + (b +_n c)\end{aligned}$$

y $a + (b +_n c) < n$ *de donde*

$$a + (b +_n c) = a +_n (b +_n c)$$

Finalmente

$$(a +_n b) +_n c = a +_n (b +_n c). \tag{1.2}$$

- *Si* $a + b + c \geq 2n$ *lo que implica que* $b + c \geq 2n - a$*. Como* $\mathbb{Z}_n$ *es finito y ordenado por consiguiente para cada* $x \in \mathbb{Z}_n$ *se cumple que* $x \leq n - 1$*. En particular lo anterior se cumple para* $a \in \mathbb{Z}_n$ *y así*

$$\begin{aligned}b + c &\geq 2n - a \\ &\geq 2n - (n - 1) \\ &\geq n.\end{aligned}$$

Por consiguiente, $b + c \geq n$ *luego*

$$\begin{aligned}(a +_n b) +_n c &= a + b + c - 2n \\ &= a + (b + c - n) - n \\ &= a + (b +_n c) - n\end{aligned}$$

Revisando la hipótesis, podemos deducir que $a + (b +_n c) \geq n$ *con lo cual*

$$a + (b +_n c) - n = a_n + (b +_n c).$$

En consecuencia,

$$(a_n + b) +_n c = a_n + (b +_n c). \tag{1.3}$$

De (1.1), (1.2) y (1.3) se tiene que

$$(a_n + b) +_n c = a_n + (b +_n c)$$

donde

$$a +_n (b +_n c) = \begin{cases} a+b+c & \textbf{si} \quad a+b+c < n \\ a+b+c-n & \textbf{si} \quad b+c \geq n \textbf{ y } a+b+c < 2n \\ a+b+c-2n & \textbf{si} \quad a+b+c \geq 2n \end{cases}$$

- *Elemento neutro: Tomemos al elemento neutro como el* $0 \in \mathbb{Z}_n$*, tal que para cada* $a \in \mathbb{Z}_n$*, se verifica que* $a +_n 0 = 0 +_n a = a$*. En efecto, para cada* $a \in \mathbb{Z}_n$ *se cumple que* $a < n$ *y por tanto>*

$$a +_n 0 = a + 0 = 0 = 0 + a = 0 +_n = a.$$

- *Elemento inverso: Para cada elemento* $a \in \mathbb{Z}_n$*, existe un elemento inverso que definimos como* $-a = n - a \in \mathbb{Z}$*. Notemos que* $a + (n - a) = n$*, por tanto*

$$a +_n (-a) = a + (n - a) - n = 0.$$

Notación

Para las siguientes secciones vamos a introducir la siguiente notación:

Sean T, U funciones, tales que $T \circ U$ existe. Ahora, vamos a notar la composición de funciones como:

$$T \circ U := TU$$

En el caso de que la composición $T \circ T$ exista, se escribirá:

$$T \circ T := T^2$$

Esta notación nos ayudará para realizar la composición de funciones en los grupos que definiremos en las siguientes líneas.

Vamos a definir el grupo de permutaciones utilizando como base lo expuesto en el libro [2], donde con antelación se definen los siguientes conceptos

Definición 1.2. *Una* Permutación de un conjunto no vacío A *es una función* $\phi : A \to A$ *que es biyectiva.*

Ejemplo 1.2. *Definamos como el conjunto* $A = \{1, 2, 3\}$*, y la función* f *tal que:*

$$\begin{aligned} f\colon \{1,2,3\} &\to \{1,2,3\} \\ 1 &\mapsto 1 \\ 2 &\mapsto 3 \\ 3 &\mapsto 2 \end{aligned}$$

De la propia definición de f*, podemos ver que la función es biyectiva.Por tanto,* f *es una* Permutación.

Ejemplo 1.3. *Definamos como el conjunto* $A = \{1, 2, 3\}$*, y la función* r *tal que:*

$$\begin{aligned} r\colon \{1,2,3\} &\to \{1,2,3\} \\ 1 &\mapsto 3 \\ 2 &\mapsto 1 \\ 3 &\mapsto 2 \end{aligned}$$

De la propia definición de r*, podemos ver que la función es biyectiva.Por tanto,* r *es una* Permutación.

Definición 1.3. *El* Producto de Permutaciones *es la composición de funciones.*

Ejemplo 1.4. *De los ejemplos anteriores sabemos que* f*,*r *tienen el mismo conjunto* $A = \{1, 2, 3\}$ *como dominio y recorrido.*

Tenemos la siguiente composición:

$$\begin{aligned} f \circ r\colon A &\to A \to A \\ 1 &\mapsto 3 \mapsto 2 \\ 2 &\mapsto 1 \mapsto 1 \\ 3 &\mapsto 2 \mapsto 3 \end{aligned}$$

Es decir:

$$\begin{aligned} fr\colon A &\to A \\ 1 &\mapsto 2 \\ 2 &\mapsto 1 \\ 3 &\mapsto 3 \end{aligned}$$

De la definición de composición de r con f, podemos ver que la función es biyectiva.Por tanto, fr es una Permutación.

Notemos que el Producto de Permutaciones *preserva la* Permutación.

Teorema 1.1. *Si A es un conjunto no vacío, definimos S_A como el conjunto de todas las permutaciones de A. Entonces S_A es un grupo bajo el producto definido anteriormente.*

La demostración del teorema se encuentra en ([2],p. 43).

Definición 1.4. *Sea $A = \{1, 2, \ldots, n\}$, el grupo de todas las permutaciones de A es el* Grupo Simétrico de n elementos a permutar, *y se denota por S_n. Además, el grupo tiene $n!$ funciones.*

Ejemplo 1.5. *En este ejemplo vamos a conocer al grupo S_3, sabemos primero que debemos permutar 3 elementos y que el grupo tiene $3! = 6$ funciones.En el siguiente gráfico podemos ver la definición de las funciones:*

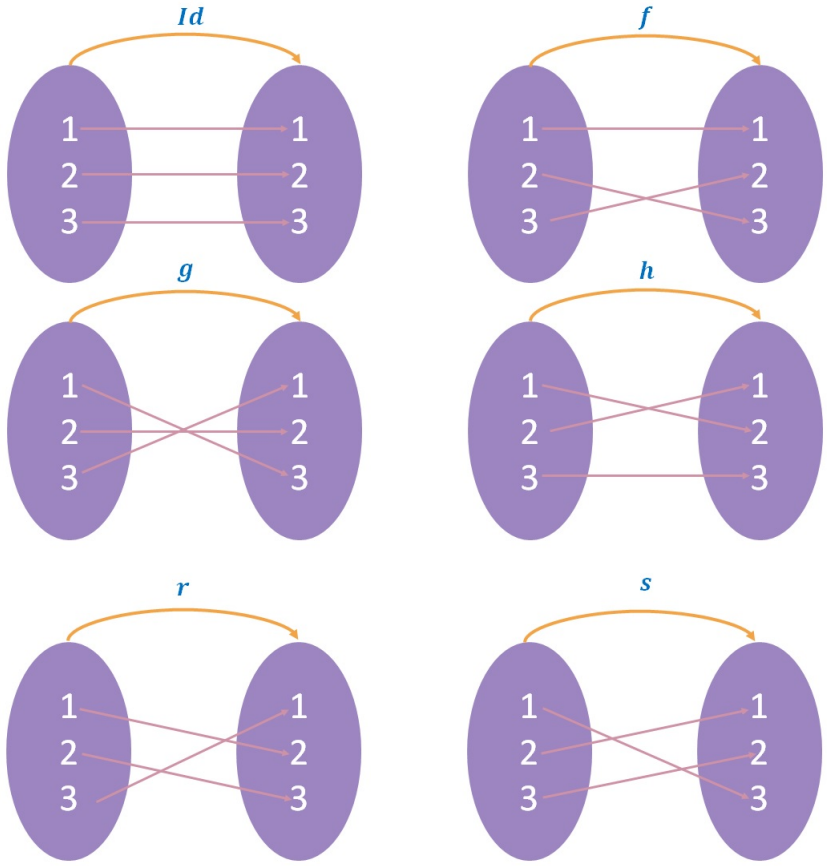


Figura 1.1: Funciones de S_3

También podemos observar con la notación usual de permutaciones:

$$id = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \quad f = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \quad g = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 2 \end{pmatrix}$$

$$h = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \quad r = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad s = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

Figura 1.2: Funciones de S_3 como permutaciones

Para definir el grupo cuyos elementos son rotaciones y reflexiones de un polígono regular utilizaremos como base libro [2] y [6], donde se presentan las siguientes definiciones previas:

Definición 1.5. *Definimos un polígono regular como la figura geométrica compuesta por una secuencia de segmentos de rectas iguales que encierran un plano. A los segmentos de rectas les llamamos lados y a los puntos donde se intersecan les llamamos vértices. Para nuestro objetivo notaremos al polígono de n lados con $\mathbb{P}_n$ y cada vértice será nombra con un número entero no negativo desde 0 hasta $n-1$, en sentido de antihorario.*

Ejemplo 1.6. *El presente ejemplo muestra el polígono regular de 3 lados, es decir el triangular equilatero:*

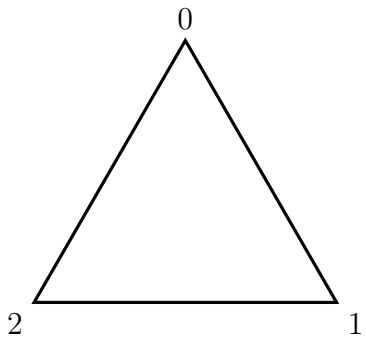


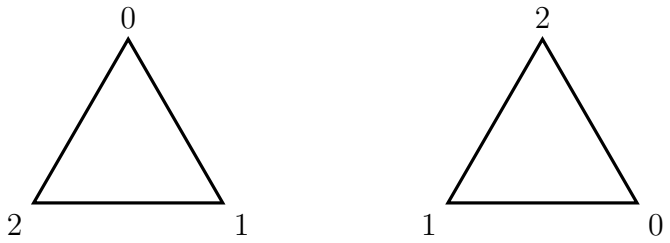
Figura 1.3: Polígono regular de 3 lados(Triángulo)

Definición 1.6. *Dado $n \geq 3$, definimos D_n como el conjunto de las funciones sobreyectivas $\phi\colon \mathbb{Z}_n \to \mathbb{Z}_n$ y que cumple que dado el segmento de recta entre los vértices i y j es un lado de $\mathbb{P}_n$ si y solo si el segmento de recta entre los vértices $\phi(i)$ y $\phi(j)$ es un lado de $\mathbb{P}_n$.*

Ejemplo 1.7. *En el presente ejemplo, vamos a asociar la definición del conjunto $\mathbb{Z}_3$ con el conjunto D_3, mediante una función sobreyectiva. Hay que recordar que D_3 también está asociado al Triangulo de la Figura 1.3. Así definimos:*

$$\begin{aligned} \bar{\phi}\colon \mathbb{Z}_3 &\to \mathbb{Z}_3 \\ 0 &\mapsto 2 \\ 1 &\mapsto 0 \\ 2 &\mapsto 1 \end{aligned}$$

Con esta función aplicada a nuestra Figura 1.3, se convierte en:



(a) Triángulo Original (b) Triángulo aplicado $\bar{\phi}$

Figura 1.4: Aplicación de la función $\bar{\phi}$

Definición 1.7. *Definimos producto de los elementos D_n como composición de funciones del propio conjunto.*

Teorema 1.2. *Para $n \geq 3$, el $(D_n, \circ)$ es grupo, el cual se denomina* Grupo Diedral *de orden n.*

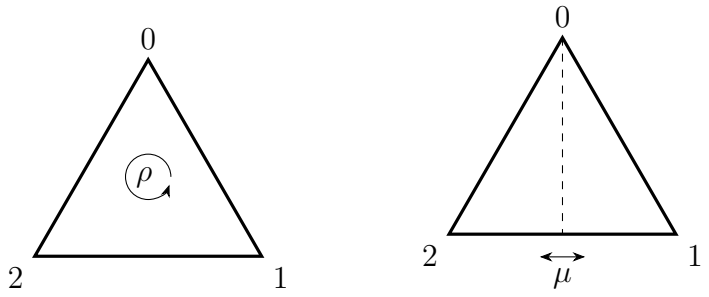
La demostración del teorema se encuentra en ([2],p. 47).

Ejemplo 1.8. *Definamos la función ρ que representará la rotación y μ que represará la reflexión de Figura 1.3, respectivamente.*

$$\begin{aligned} \rho\colon \mathbb{Z}_3 &\to \mathbb{Z}_3 \\ k &\mapsto \rho(k) = k +_3 1 \end{aligned}$$

$$\begin{aligned} \mu\colon \mathbb{Z}_3 &\to \mathbb{Z}_3 \\ k &\mapsto \mu(k) = \begin{cases} 3-k & \text{si} \quad k \neq 0 \\ 0 & \text{si} \quad k = 0 \end{cases} \end{aligned}$$

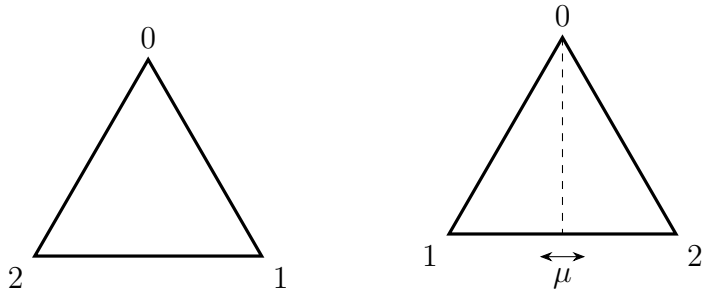
En la Figura 1.5, podemos ver las direcciones de rotación y reflexión propuestas por estas funciones.



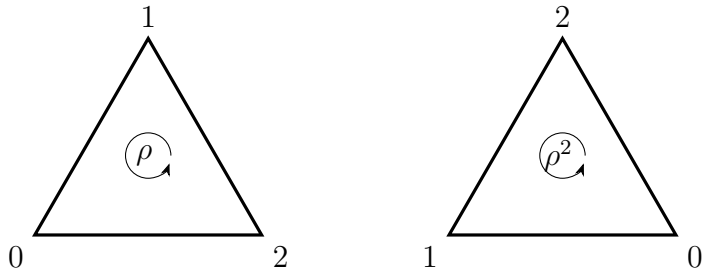
(a) Función de Rotación (b) Función de Reflexión

Figura 1.5: Funciones para D_3

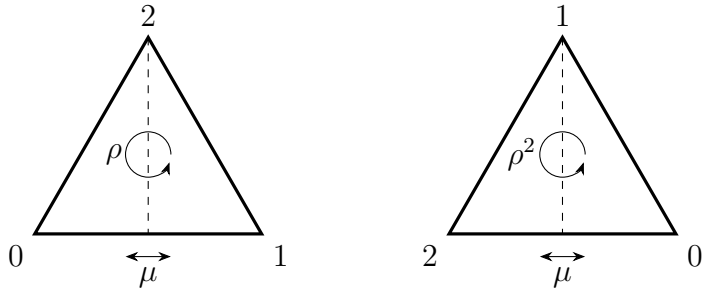
Las siguientes figuras muestran la composición de las funciones ρ y μ:



(a) Función id (b) Función de μ



(c) Función ρ (d) Función de ρ^2



(e) Función $\mu\rho$ (f) Función de $\mu\rho^2$

Figura 1.6: Rotaciones y Reflexiones D_3

Por tanto $D_3 = \{id, \rho, \rho^2, \mu, \mu\rho, \mu\rho^2\}$*, en efecto la función* ρ *cubre todas las rotaciones y* μ *es una reflexión que con la composición con* ρ *completa las otras reflexiones del triángulo equilátero(Figura 1.3).*

Las siguientes definiciones y teoremas son extraídas del libro [3], en el cual se encuentran las demostraciones correspondientes.

Definición 1.8. *Sea* $(G, \odot)$ *un grupo decimos que es* Abeliano *si para todo* $a, b \in G$ *se cumple que* $a \odot b = b \odot a$*.*

Definición 1.9. *Sea* H *un subconjunto no vacío de* G*, se dice que es un* Subgrupo *de* G *si respecto a la operación producto de* G*, el par* $(H, \odot)$ *es un grupo.*

Teorema 1.3. *Sea* H *un subconjunto no vacío del grupo* G*. Decimos que el par* $(H, \odot)$ *es un subgrupo del grupo* $(G, \odot)$ *si y solo si*

1. *Para todo* $a, b \in H$ *se tiene que* $a \odot b \in H$*.*
2. *Para todo* $a \in H$ *se tiene que* $a^{-1} \in H$*.*

Ejemplo 1.9. *Sea* R *subconjunto de* S_3*, tal que*

$$R = \{id, r, s\}$$

con la operación composición de funciones es subgrupo, en efecto:

$id \circ id = id^2 = id$	$r \circ id = r(id) = r$	$s \circ id = s(id) = s$
$id \circ r = (id)r = r$	$r \circ r = r^2 = s$	$s \circ r = r^2 = id$
$id \circ s(id)s = s$	$r \circ s = rs = id$	$s \circ s = rs = r$

En lo anterior se puede comprobar que cumple con las condiciones para ser subgrupo según el Teorema 1.3.

Definición 1.10. *Si* G *es un grupo finito, definimos al* Orden del Grupo *como el número de elementos de este grupo, y lo denotamos como* $o(G)$*.*

Definición 1.11. *Si* G *es un grupo y* $a \in G$*, definimos al* Orden del Elemento a *es el entero positivo* n *más pequeño tal que* $a^n = e$*. Si no existe el* n*, decimos que a tiene* a *es de orden infinito.*

Teorema 1.4. *Si G es un grupo finito y H es un subgrupo de G entonces $o(H)$ es un divisor de $o(G)$.*

Teorema 1.5. *Sea G un grupo, para cada $g \in G$ el conjunto*

$$\langle g \rangle = \{g^k : k \in \mathbb{Z}\}$$

es un subgrupo de G. Se le denomina el Subgrupo Cíclico generado por g

Definición 1.12. *La función ϕ de $(G, \odot)$ en $(\bar{G}, \circledast)$ se dice* Homomorfismo *si para cada $a, b \in G$, se tiene que*

$$\phi(a \odot b) = \phi(a) \circledast \phi(b)$$

Definición 1.13. *Sea la función $\phi : (G, \odot) \to (\bar{G}, \circledast)$ un homomorfismo, decimos que ϕ es un* Isomorfismo *si ϕ es biyectiva.*

Definición 1.14. *Sean $(G, \odot)$ y $(\bar{G}, \circledast)$ grupos se dicen que son* Isomorfos *si existe un isomorfismo $\phi : (G, \odot) \to (\bar{G}, \circledast)$.En este caso denotamos como $G \approx \bar{G}$.*

1.4.2. Programación Orientada a Objetos

Al iniciar la programación en un lenguaje específico debemos comprender los diferentes términos a utilizar en la implementación. En particular, existen elementos propios del lenguaje C++ que comprenden desde la declaración de números enteros hasta estructuras más avanzadas y conformadas por varias estructuras adicionales. También hay que destacar que se programará en un código que es conocido por gran parte de las personas dedicadas al desarrollo de aplicaciones.

Conforme a la complejidad de resolver un problema planteado con implementación en el lenguaje de programación C++, se necesita estructuras de mayor abstracción. De donde nace la idea de utilizar una programación que consiste en dividir el problema en diferentes secciones o subprogramas que contribuyan para la creación de la solución definitiva, a esto lo llamamos *Programación Modular*. Esta forma de programación fue un tema fundamental en la creación del lenguaje C++, puesto que con ello lleva a la manipulación de estos subprogramas.

A continuación, tomando como referentes los libros [1], [4], [5], y [7] desarrollaremos las herramientas que vamos a utilizar en la creación de nuestro programa. No se incluirá definiciones que no contribuyan al programa o métodos intrínsecos que no necesitan mayor comprensión para la utilización

1.4.3. Tipos de Datos

Para cada entidad en el lenguaje C++, hay que asignar un tipo de dato que ayuda a diferenciar las operaciones predefinadas.Por ejemplo, si declaramos dos variables enteras y procedemos a dividir el programa nos retorne un error pues existen divisiones que resultan en un número que no es entero.

Definición 1.15. *Se define* Tipo de Dato *como el identificador de cada entidad en C++ tal que se pueda realizar operaciones y manejarlos. Estos identificadores además de dar propiedades para manejarlos, también da limitaciones en el uso de las funciones internas de C++.*

Definición 1.16. *Definimos como* Datos Fundamentales *a aquellos que vienen predefinidos por el lenguaje de programación. En nuestro caso vamos a utilizar:*

- bool: *Como lo indica su nombre es un dato lógico, es decir tiene dos valores el de verdadero y el de falso. Nos ayuda con la expresión de resultados lógicos como ver si dos elementos son iguales.*
- int: *Este dato identifica a los valores enteros y sus operaciones. Tiene variantes de acuerdo al uso de memoria y se utiliza en diferentes sistemas de creación numérica.*
- void: *Este dato se utiliza en la creación de funciones que se ejecutan pero no devuelva ningún valor.*
- string: *Este dato se utiliza en la creación de una cadena de texto, es decir se le puede asignar una palabra pero además se puede acceder a cada letra de esta.*

En varios casos necesitamos asignar un nombre al dato, esto suele ocurrir cuando en el dato se tiene varias asignaciones diferentes. Esto se realiza con el comando **typedef** *seguido de una declaración.*

Si tenemos un dato A y asignamos su tipo escribimos **Tipo_Dato** A. *Además, junto al tipo de datos podemos declarar* **const** *que ayuda a declarar constantes de una manera más adecuada.*

Definición 1.17. *Junto a la definición de datos, podemos crear una secuencia contigua de algún dato, a esto le llamamos* Arreglo *o* Array. *Para asignar el tamaño del arreglo se lo realiza entre corchetes de la siguiente manera* **Tipo_Dato** A[**int** tamaño].

Además, mediante el operador **new** *se asigna un arreglo de forma dinámica, reservando memoria específica.*

Ejemplo 1.10. *En este ejemplo vamos a observar como se asigna tipo a los datos, la creación de un arreglo y la asignación de un nuevo nombre.*

```
1  int main()
2  {
3      //Declaracion de a,b,c,d como enteros
4      int a,b,c,d;
5      //Asignacion el valor de 6 a d
6      d=6;
7      //Creacion de un arreglo de tamano d
8      int A[d]; //1
9      //Asignacion de valores al arreglo A
10     A[d]={a,b,c,d,a,d};
11     //Asignacion de un nuevo nombre de dato
12     typedef int Arreglo[d];
13     //Con lo anterior lo siguiente es lo mismo que en 1
14     Arreglo A;
15 };
```

Figura 1.7: Asignación de Datos C++

1.4.4. Funciones y Clases

Definición 1.18. *A la colección de líneas de código que generan un proceso y que devuelve un valor se lo define como* Función.*Por lo general, para la inicialización se ingresa parámetros con datos. Se usan para descomponer el problema en tareas simples, es un primer acercamiento a la programación en módulos. La utilización de estas reduce el tamaño del código. Se declara como* **Tipo_Dato** Nombre_funcion (Parámetros)*. Es conveniente declarar al inicio del código.*

Ejemplo 1.11. *En el siguiente ejemplo se define la función suma de enteros:*

```
1 int Suma(int a,int b)
2 {
3     return a+b;
4 }
```

Figura 1.8: Declaración de función

Definición 1.19. *Los* Tipos Abstractos de Datos(TAD) *se define como el manejo de la abstracción de datos, en los cuales se conceptualiza el tipo de dato y el significado de las operaciones que se les otorga.*

Como veremos a continuación la implementación interna de los *TAD* no se visualiza y no se tiene acceso, impidiendo la manipulación directa de estos.

Definición 1.20. *Una* clase *es una implementación del usuario donde se crea una nueva definición de dato en el lenguaje de programación. Se inicializa con:*

```
1 class NombredeClase{...};
```

Figura 1.9: Declaración de Clase

Entre las llaves se ingresaran las definiciones y los métodos que se pueden aplicar a los objetos de la clase.

Definición 1.21. *Al elemento declarado con el tipo de clase se lo denomina como Un* Objeto.

Definición 1.22. *A partir de la definición de una clase, podemos distinguir dos entornos:*

1. Público, *sus miembros son accesibles desde el ámbito interno y externo de la clase.*
2. Privado, *sus miembros son accesibles desde únicamente del ámbito interno de la clase.*

Definición 1.23. *Definimos* Subclase *como la clase que se deriva de otra. Por ejemplo, tenemos como clase padre la* **Class** Figura *y como subclases(clases hijas) al cuadrado* **Class** Cuadrado :: public Figura.

En la declaración se observa el término public, esto hace referencia a que la subclase solo tiene acceso a los datos públicos de clase padre, podemos crear métodos con los mismos nombres en las subclases y el compilador los entenderá que cada método corresponde a clases diferentes.

Definición 1.24. *Las* Variables Miembros *son la representación interna de la clase se ubican en el entorno privado, para proteger de modificaciones. Son independientes de otras implementaciones de la clase.*

Definición 1.25. *Las* Funciones Miembros *son las funciones con o sin parámetros que se declaran en la clase, estas son propias de la clase y realizan los procedimientos para el funcionamiento de los objetos de la clase, como describimos anteriormente al tener una clase padre podemos definir con el mismo nombre a las funciones que pertenecen a diferentes clases que se derivan de una clase padre.*

Definición 1.26. *El* constructor *de una clase nos permite declarar una función propia que construye objetos del tipo de la clase y que con ellos se los puede manipular. Tienen las siguientes características:*

- *Tienen el mismo nombre que la clase.*
- *Se definen dentro o fuera de la clase.*
- *No retornan valores.*

Además, se dividen en dos tipos:

- Constructor por defecto, *no tiene ningún argumento, a pesar de no tener un parámetro también inicializa la creación de objetos.*
- Constructor con argumentos, *tiene argumentos de diferentes tipos, con estos procede a inicializar los objetos y también realiza una evaluación en el ingreso para distinguir cada objeto.*

Los constructores garantizan un espacio asignado a los objetos en nuestra memoria y podemos invocarlos en cualquier momento.Implementación:

```
1  //Subclases
2  class Padre{//...//};
3  class Hija:: public Padre{
4      public://Entorno Publico
5      NombredeClase();//Constructor por defecto
6      NombredeClase1(tipodedato1 arg1, tipodedato1 arg2,...);
           //Constructor con argumentos
7      tipoddedato1 funcionmiembro();//Funcionsin argumentos
8      tipoddedato1 funcionmiembro(Parametros);//Funcion con
           argumentos
9      //...
10     private: //Entorno Privado
11     tipodedato1 variable_miembro1;
12     //...} ;
```

Figura 1.10: Impementación Clase

Definición 1.27. *La* Sobrecarga de Operadores *es un instrumento en nuestro lenguaje de programación que nos ayuda a tener una operación con el mismo nombre pero para diferentes datos. Un ejemplo natural es la suma de enteros y la suma de fracciones a los dos les llamamos sumas a pesar que tenga procedimientos diferentes. En C++, ocurre lo mismo podemos asignar un operador para distintos tipos de datos pero con el mismo nombre. Se define como una función, es decir:* **Tipo_Dato operator +** (Parámetros).

Definición 1.28. *Las variables miembros pueden ser* Variables Estáticas, *estas son variables son comunes en todas las instancias de la clase y accesible en todas las funciones miembro.*

```
1 class NombredeClase {
2     public:
3     NombredeClase operator* (const variable&)const; //
          Sobrecarga de operador
4     static tipodedato1 variable; //Variable Estatica//...
5     private://...
6 };
7 tipodedato1 NombredeClase::variable = 5; //Inicializacion
```

Figura 1.11: Declaración de Sobrecarga y Variable Estatica

El **const** aparece dos veces en la sobrecarga, en la primera ocurrencia indica que la variable no puede ser modificada y además con & evita que se realice copias innecesarias. En la segunda aparición hace que la función no pueda modificar a los objetos.

Definición 1.29. *Llamaremos* Función Amiga *a la que tiene el derecho de acceder a los miembros públicos y privados de la clase. Para la inicialización se debe escribir* **friend class** ClaseFriend *y se dice que A es amiga de B,los miembros de A se acceden desde B.Esta definición no es transitiva si una clase A es amiga de B, no se puede decir que B es amiga de A.*

```
1 class Clase A{
2     friend class B //Amiga de B, a la clase A se puede
          acceder desde B//...
3 };
4 class B{
5     //...
6 } ;
```

Figura 1.12: Clase Amiga

Capítulo 2

Metodología

En el presente capítulo analizaremos la teoría de grupo y las estructuras del lenguaje de programación para luego describir la construcción de los grupos S_3 y D_3 con sus respectivas propiedades. Finalmente, diseñaremos el isomorfismo entre los grupos anteriores e implementaremos en C++ con el fin de percibir que tienen la misma estructura.

2.1. Diseño e Implementación del Grupo S_3

De acuerdo con nuestro Capítulo 1, vamos a utilizar la estructura de clase. Pero antes del uso de este elemento vamos a diseñar las funciones que representa a las permutaciones de nuestro grupo $S3$.

Nos centraremos en la creación de funciones en C++ con cada elemento de $S3$. Como hemos visto en el Ejemplo 1.5 cada miembro de $S3$ es una función de $\{1, 2, 3\}$ en $\{1, 2, 3\}$.

Construyamos una función genérica en C++, que reciba un dato entero y nos devuelva un entero modificado de acuerdo con una condición para cada función en $S3$. La modificación se realizará con el condicional If-Else.

Funciones

Notemos los siguientes parámetros para la construcción de funciones:

- Para id, nos retornaran los mismos ingresos.
- Para f, si se ingresa 1 retorna el mismo, caso contrario se condiciona para permutar los otros números.
- Para g, si se ingresa 2 retorna el mismo, caso contrario se condiciona para permutar los otros números.
- Para h, si se ingresa 3 retorna el mismo, caso contrario se condiciona para permutar los otros números.
- Para r, si se ingresa un número n tal que $n < 3$, nos devuelve $n + 1$. Caso contrario nos devuelve $n - 2$.
- Para s, si se ingresa un número n tal que $n - 1 < 1$, nos devuelve $n + 2$. Caso contrario nos devuelve $n - 1$.

Los siguientes diagramas de flujos nos ejemplifican los procesos, particularmente para f y r:

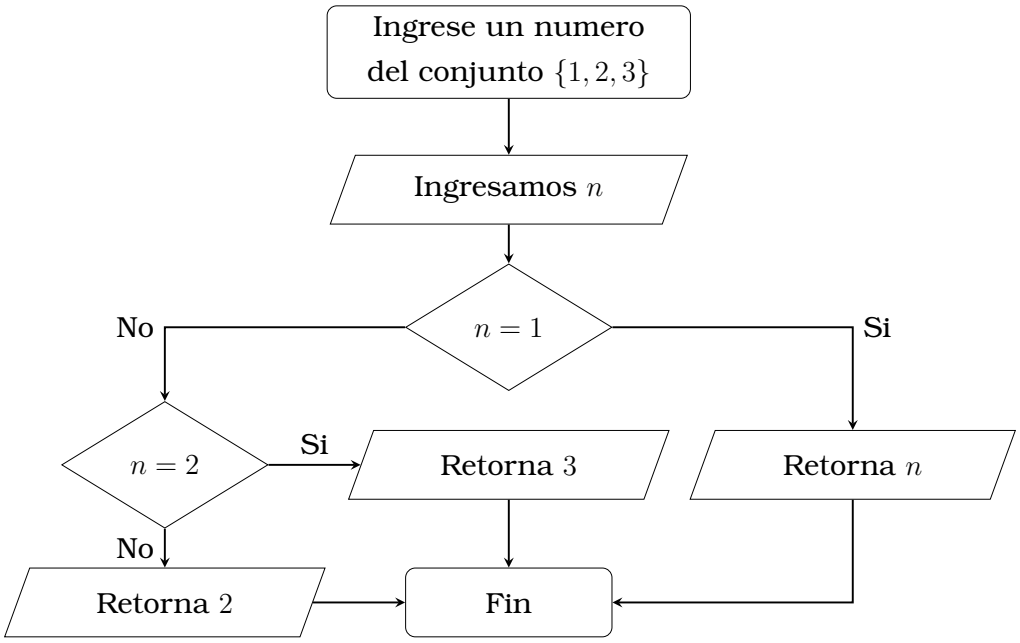


Figura 2.1: Diagrama de Flujo para f

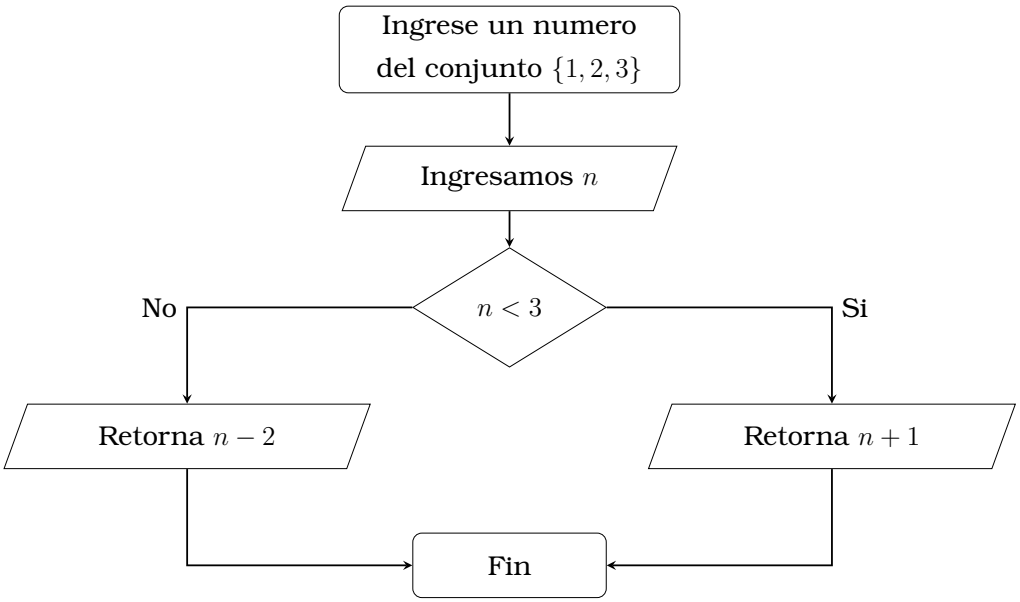


Figura 2.2: Diagrama de Flujo para r

A partir de la construcción de las funciones en el lenguaje C++, vamos a usar un especificador para tener nombres más entendibles. La definición de la distinción será como un sinónimo de todas los elementos de S_3 mediante **typedef**.Finalmente construimos la clase S_3 para precisar métodos y miembros específicos.

Construcción de Clase

Entorno Privado

En este entorno ingresamos nuestro especificador llamado *funcion_S3* y el proceso *nombre* quien nos identificará la función y nos devolverá la nomenclatura de esta. Los datos en el entorno quedan protegidos de manipulación directa del usuario externo.

Entorno Público

Por otro lado, en este entorno configuramos diferentes funciones que dependan de nuestro nuevo dato S_3. Así, tenemos:

- Constructores que manipularan el nuevo dato creado y la creación de la tabla de verdad.

- Operadores donde se definen los cálculos que se realizan en el grupo, tal como producto, inverso e igualdad.
- Funciones miembros que se encargarán de exponer la nomenclatura de cada elemento de S_3, orden de elemento y creación de grupos cíclico .
- Variable estática que asigna funciones de S_3 como apuntadores en un arreglo.

Constructores

Vamos a precisar los constructores que iniciará los objetos de nuestra clase. Tenemos varios constructores para este proceso, los detallaremos a continuación:

Constructor S3(funcion_S3)

Este constructor nos ayudará para acceder a las funciones de S_3 ya implementadas. Además, mediante la asignación dinámica de arreglos con apuntadores se construirá una ordenación de los elmentos de S_3 con la cual generamos la tabla de verdad.

Constructor S3(funcion_S3,...,funcion_S3)

Este constructor asignará determinadas funciones a posiciones del arreglo dinámico de punteros, con lo cual generamos una tabla de verdad que puede mover sus posiciones.

Constructor S3(int,int)

En este constructor se realizarán comparaciones mediante el ingreso de dos números del conjunto de llegada de las funciones en S_3 e identificará a la función buscada.Nos ayudará con la operación producto, que se implementará en la sección de operadores.

Identificación

Como se observa en la definición de las funciones de S_3 son representaciones de permutaciones. Por ejemplo, la función f es la permutación:

$$f = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$$

Figura 2.3: Permutación f

Las otras permutaciones podemos ver en la Figura 1.2.

Como vemos cambian de posición los elementos, con lo cual toma 2 elementos que permutan y luego compara sus posiciones. Así, fijamos dos valores enteros a, b, donde representan las primeras posiciones de los elementos a permutar:

- Si $a = 1$ y $b = 2$, se nos asigna la función id.
- Si $a = 1$ y $b \neq 2$, se nos asigna la función f.
- Si $a = 2$ y $b = 1$, se nos asigna la función h.
- Si $a = 2$ y $b \neq 2$, se nos asigna la función r.
- Si $a \neq 1$, $a \neq 2$ y $b = 2$, se nos asigna la función g.
- Si $a \neq 1$, $a \neq 2$ y $b \neq 2$, se nos asigna la función s.

Operadores

Para la construcción de operadores utilizaremos la siguiente simbología:

- $*$: Producto de Permutaciones.
- $==$: Igualdad de Permutaciones.
- $!=$: Permutación Diferente.
- $-$: Elemento Inverso.

Producto de Permutaciones

En este caso, vamos a construir un operador que al ingresar dos funciones nos retorne la composición de funciones. Por tanto, la definimos con un valor de salida S3, y los datos a ingresar son dos valores S3. Para ocupar menos memoria, un valor de entrada se ingresará como variable constante.

También tendremos en cuenta las evaluaciones de dos elementos en las funciones ingresadas mediante nuestro nuevo dato y su implementación privada. Así tendremos de retorno dos enteros que mediante el constructor S3(int,int) nos identificará la función.

Igualdad de Permutaciones

Construimos un operador que al ingresar dos funciones nos retorne un elemento booleano. Por tanto, la definimos con un valor de bool, y los datos a ingresar son dos valores S3. Se trabajará de izquierda a derecha, pero como estamos comparando no conlleva a confusiones.

Internamente el operador recibe dos enteros los cuales se evalúan en las funciones de S_3 que se van a comparar. A continuación si ambos resultados de lo anterior son iguales se retorna el valor verdadero, caso contrario se retorna falso.

Permutación Diferente

Su diseño es igual al anterior, solo cambia en las comparaciones. Es decir, si comparamos las dos evaluaciones, en el caso que son sean diferentes se retorna verdadero, caso contrario se retorna falso.

Inverso

El operador inverso se puede construir gracias al arreglo dinámico de funciones de S_3 implementado en el constructor. La idea para la construcción de este operador es la definición de grupo en el literal 4. Realizamos las evaluaciones de $\{1, 2\}$ en la función proporcionada por el usuario y los enteros proporcionados por este mapa los aplicamos en cada elemento

del arreglo- El resultado comparamos con 1 y 2, respectivamente, si son iguales identificamos el elemento inverso de la función.

Funciones Miembro

Mostrar

La función mostrar nos permite acceder a los datos privados de tal manera que con esta obtengamos los nombres de cada función y se pueda mostrar en consola. Se basa en ingresar dos enteros los cuales procederán a ingresar en el método privado para la identificación de la función.

Tabla

El procedimiento realiza evaluaciones mediante dos iteraciones con el arreglo dinámico, en cada una de las iteraciones procede a calcular dos valores en las funciones de S3. Estos valores proceden a invocar el método privado nombre para que nos brinde la identificación de la función. Si se ingresa una permutación diferente a la original la función tabla nos retornará la tabla en el nuevo orden.

Cíclico

El proceso cíclico aprovecha las habilidades del operador Do-While para evaluar la función ingresada las veces que sean necesarias para convertirse en la función identidad. En el mismo proceso también nos retorna cada función en la que se convierte.

Orden

El mismo proceso que en Cíclico, pero en vez de retornar funciones realiza el subproceso de contar las iteraciones, este valor es el orden del elemento de S3.

2.2. Diseño e Implementación del Grupo D_3

Seguido del diseño anterior, continuamos con la implementación del grupo D_3. Para comenzar diseñemos las funciones en $\mathbb{Z}_n$ que representarán el movimiento de los segmentos de recta dentro del polígono $\mathbb{P}_3$.

Como vimos en el ejemplo 1.8 debemos implementar una función que sea la suma en $\mathbb{Z}_3$ y la función inverso de $\mathbb{Z}_3$ en el contexto de C++.A continuación de esto implementaremos las rotaciones y reflexiones.

Funciones asociadas a $\mathbb{Z}_3$

Suma en $\mathbb{Z}_3$

En la implementación de la suma en Z_3 se basa en dos premisas. La primera es que solo admite ingreso de enteros en Z_3, es decir, si el elemento de entrada no está en el conjunto la implementación no procede.Se inicializa con el ingreso es de dos elementos.

La segunda premisa es el uso del condicional If-Else para determinar que la suma no supere 3, en efecto se conoce la condición de la definición de suma en Z_n. En el código se denominará como **sumd3**. El siguiente diagrama nos ayudará a visualizar el procedimiento.

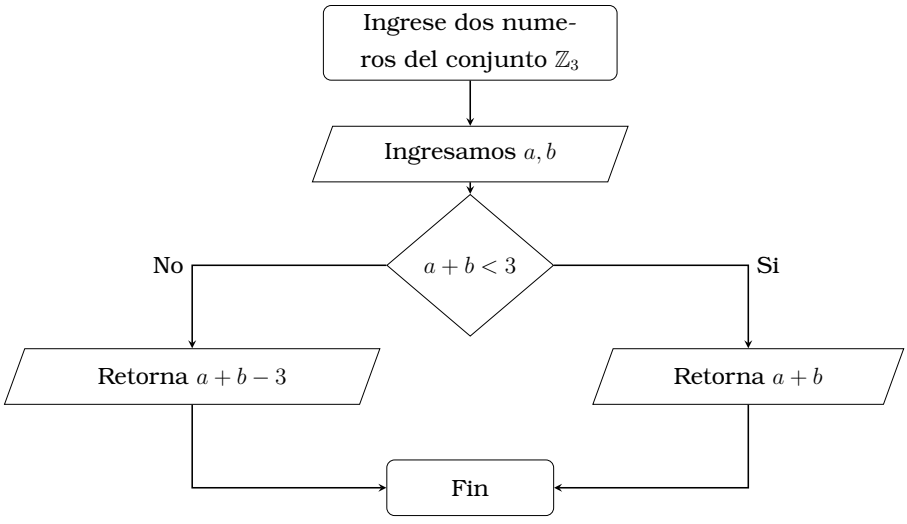


Figura 2.4: Diagrama de Flujo Suma en $\mathbb{Z}_3$

Inverso en $\mathbb{Z}_3$

Como en la implementación anterior se basa las mismas premisas anteriores. Pero utilizaremos If-Else para determinar que el valor ingresado sea diferente de 0 y se inicializa con un elemento.En el código se denominará como **ind3**.Visualicemos el proceso en el siguiente diagrama:

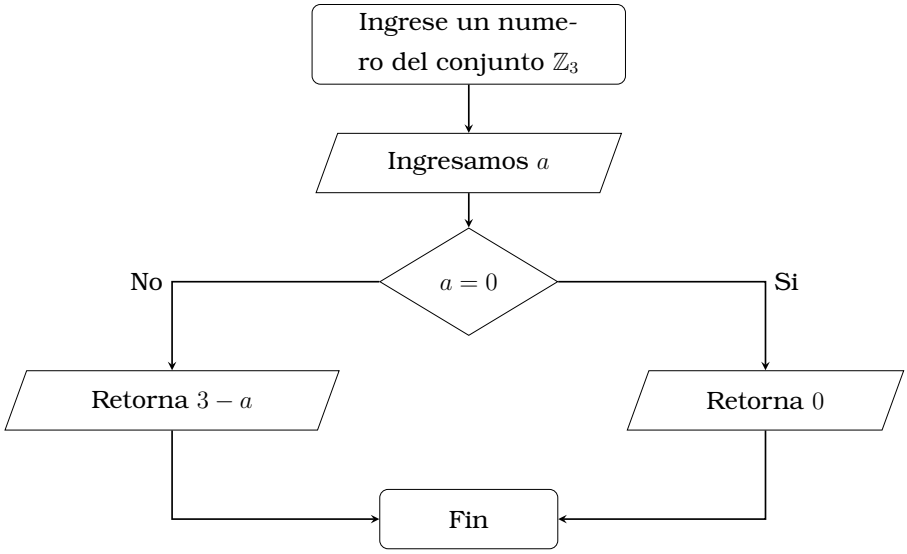


Figura 2.5: Diagrama de Flujo Inverso de $\mathbb{Z}_3$

Notación en C++

Vamos a utilizar la siguiente notación en el código de C++:

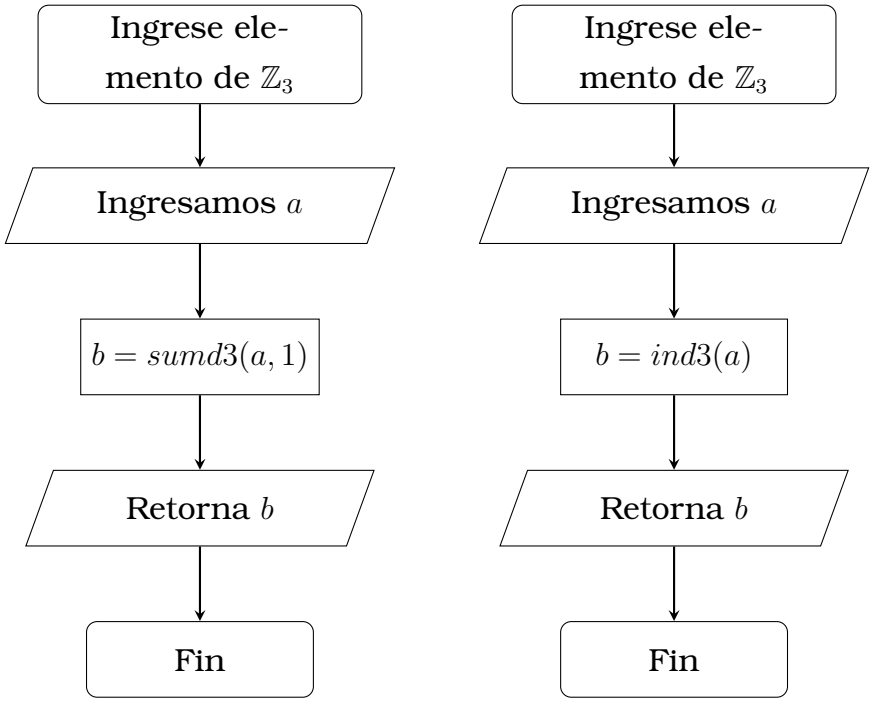
- Para la id utilizaremos e.
- Para la ρ utilizaremos $r90$.
- Para la ρ^2 utilizaremos $r180$.
- Para la μ utilizaremos u.
- Para la $\mu\rho$ utilizaremos $ur90$.
- Para la $\mu\rho^2$ utilizaremos $ur180$.

Funciones

Notemos las siguientes funciones:

- Para e, nos retorna las mismas entradas.
- Para $r90$, es una rotación de $90°$ en sentido antihorario de nuestro triangulo y definida como ρ. La función nos retorna de la suma en $\mathbb{Z}_3$ entre un valor ingresado externamente y de 1.
- Para $r180$, como vimos en el capítulo anterior es la composición de $r90$ con sigo misma. Por tanto, nos retorna valores de la composición.
- Para u, es una reflexión en la altura del triangulo que va desde el vértice 0. En la definición de funciones asociadas a D_3 tenemos μ que realiza este proceso.La función nos retorna el inverso de $\mathbb{Z}_3$.
- Para $ur90$, nos devuelve la composición entre u y $r90$.
- Para $ur180$, nos devuelve la composición entre u y $r180$.

Los siguientes diagramas de flujos nos ejemplifican los procesos, particularmente para $r180$ y u:



(a) Diagrama de Flujo $r180$

(b) Diagrama de Flujo u

Figura 2.6: Diagrama de Flujo de elementos de D_3

A partir de la construcción de las funciones en el lenguaje C++, vamos a definir un nuevo tipo de dato. Este dato va ser definido como las funciones que pertenecen al grupo D_3. La definición se da mediante el uso de la estructura Clase.

Construcción de Clase

Entorno Privado

En este entorno ingresamos nuestro nuevo dato llamado *funcion_D3* y el proceso *nombre* quien nos identificará la función y nos devolverá la nomenclatura de esta. Los datos en el entorno quedan protegidos de manipulación directa del usuario externo.

Entorno Público

Por otro lado, en este entorno configuramos diferentes funciones que dependan de nuestro nuevo dato D_3. Así, tenemos:

- Constructores que manipularan el nuevo dato creado y la creación de la tabla de verdad.
- Operadores donde se definen los cálculos que se realizan en el grupo, tal como producto, inverso e igualdad.
- Funciones miembros que se encargarán de exponer la nomenclatura de cada elemento de D_3, orden de elemento y creación de grupos cíclico .
- Variable estática que asigna funciones de D_3 como apuntadores en un arreglo.

Constructores

Vamos a identificar las diferencias entre los constructores de la clase S_3 y D_3. Tenemos varios constructores para este proceso, los detallaremos a continuación:

Constructor D3(funcion_D3)

El constructor es similar al constructor de S_3, pero en este caso se accede las funciones de D_3 y de igual manera se crea un arreglo dinámico para la generación de la tabla de verdad.

Constructor D3(funcion_D3,...,funcion_D3)

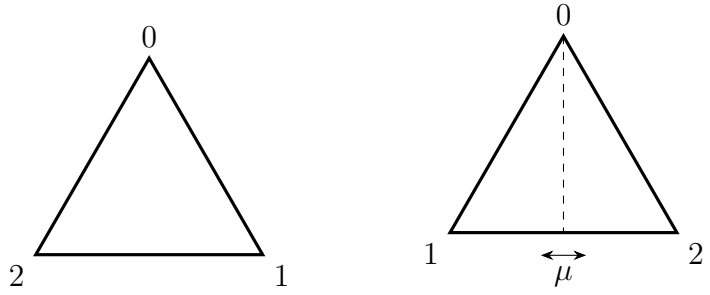
El constructor es idéntico al constructor en S_3, pero con la diferencia de acceder a las funciones de D_3.

Constructor D3(int,int)

Como el implementado para S_3 se realizan comparaciones mediante el ingreso de dos números del conjunto de llegada de las funciones en D_3 e identificará a la función buscada.Nos ayudará con la operación producto, que se implementará en la sección de operadores.

Identificación

En la definición de D_3 vimos que son rotaciones y reflexiones de un triangulo rectángulo, además están asociadas a una función definida en $\mathbb{Z}_3$. Por ejemplo, la reflexión con respecto al vértice 0 corresponde a la función μ del ejemplo , analicemos la siguiente figura:



(a) Triángulo original (b) Aplicación μ

Figura 2.7: Identificación D_3

Las rotaciones y reflexiones son combinaciones de las funciones ρ y μ vistas en el ejemplo

Como vemos cambian de posición los vértices, con lo cual toma 2 vértices que cambian y luego comparo para analizar la función asociada. Así, fijamos dos valores enteros a, b, donde representan las primeros vértices:

- Si $a = 0$ y $b = 1$, se nos asigna la función id.
- Si $a = 0$ y $b \neq 1$, se nos asigna la función μ.
- Si $a = 1$ y $b = 0$, se nos asigna la función $\mu\rho^2$.
- Si $a = 1$ y $b \neq 1$, se nos asigna la función ρ.
- Si $a \neq 0$, $a \neq 1$ y $b = 1$, se nos asigna la función $\mu\rho$.
- Si $a \neq 0$, $a \neq 1$ y $b \neq 1$, se nos asigna la función ρ.

Operadores

Para la construcción de operadores utilizaremos la siguiente simbología:

- $*$: Producto de Permutaciones.
- $==$: Igualdad de Permutaciones.
- $!=$: Permutación Diferente.
- $-$: Elemento Inverso.

Producto de Rotaciones y Reflexiones

En este caso, vamos a construir un operador que al ingresar dos funciones asociadas a las rotaciones y reflexiones nos retorne los valores de su composición. Por tanto, la definimos con un valor de salida D3, y los datos a ingresar son dos valores D3. Para ocupar menos memoria, un valor de entrada se ingresará como variable constante.

También tendremos en cuenta las evaluaciones de dos elementos en las funciones ingresadas mediante nuestro nuevo dato y su implementación privada. Así tendremos de retorno dos enteros que mediante el constructor D3(int,int) nos identificará la función.

Igualdad de Rotaciones y Reflexiones

Construimos un operador que al ingresar dos funciones nos retorne un elemento booleano. Por tanto, la definimos con un valor de bool, y los datos a ingresar son dos valores D3. Se trabajará de izquierda a derecha, pero como estamos comparando no conlleva a confusiones. Opera de forma similar a la Igualdade de Permutaciones.

Rotaciones y Reflexiones Diferente

Su diseño es igual al anterior, solo cambia en las comparaciones. Es decir, si comparamos las dos evaluaciones, en el caso que son sean diferentes se retorna verdadero, caso contrario se retorna falso.

Inverso

El operador inverso se puede construir gracias al arreglo dinámico de funciones de D_3 implementado en el constructor. La idea para la construcción de este operador es la definición de grupo en el literal 4. Realizamos las evaluaciones de $\{0, 1\}$ en la función proporcionada por el usuario y los enteros proporcionados por este mapa los aplicamos en cada elemento del arreglo. El resultado comparamos con 0 y 1, respectivamente, si son iguales identificamos el elemento inverso de la función.

Funciones Miembro

Las funciones miembros son idénticas a las funciones miembros en...., pero haremos una descripción breve con los cambios realizados:

- **Mostrar:** Realiza el mismo procedimiento que la sección 2.1.
- **Tabla:** Construye la tabla de verdad del grupo D_3, de forma similar al visto en la sección 2.1.
- **Cíclico:** Mediante Do-While evaluar la función ingresada las veces que sean necesarias para convertirse en la función identidad.

- **Orden:** El mismo proceso que en Cíclico, pero en vez de retornar funciones realiza el subproceso de contar las iteraciones, este valor es el orden del elemento de D_3.

Las dos clases construidas anteriormente van hacer hijas de la clase *Grupo*, para que se mantenga los nombres de las funciones en todo ámbito y que el compilador distinga al tipo que corresponde.

2.3. Construcción Isomorfismo entre S_3 y D_3

En la siguiente sección vamos a definir un mapa entre los grupos que hemos venido trabajando, a continuación demostraremos que cumple con la definición de Isomorfismo. Como último paso se describirá la implementación que se realizó en C++.

Definición del Isomorfismo

Vamos a definir $\phi : (S_3, \circ) \to (D_3, \circ)$, donde:

$$\begin{aligned} \phi\colon (S_3, \circ) &\to (D_3, \circ) \\ id &\mapsto \phi(id) = id \\ f &\mapsto \phi(f) = \mu \\ g &\mapsto \phi(g) = \mu\rho \\ h &\mapsto \phi(h) = \mu\rho^2 \\ r &\mapsto \phi(r) = \rho \\ s &\mapsto \phi(s) = \rho^2 \end{aligned}$$

Demostración. Por la propia definición de ρ se cumple que la función es biyectiva. Nos falta demostrar que se cumple que para cada $t, u \in S_3$:

$$\phi(tu) = \phi(t)\phi(u) \tag{2.1}$$

Fijemos $\rho \in S_3$, verifiquemos la condición 2.1 :

r	$\phi((r)(\bullet))$	$\phi(r)\phi(\bullet)$
id	$\phi((r)(id)) = \phi(r) = \rho$	$\phi(r)\phi(id) = (\rho)(id) = \rho$
f	$\phi((r)(f)) = \phi(h) = \mu\rho^2$	$\phi(r)\phi(f) = (\rho)(\mu) = \mu\rho^2$
g	$\phi((r)(g)) = \phi(f) = \mu$	$\phi(r)\phi(g) = (\rho)(\mu\rho) = \mu$
h	$\phi((r)(h)) = \phi(g) = \mu\rho$	$\phi(r)\phi(h) = (\rho)(\mu\rho^2) = \mu\rho$
r	$\phi((r)(r)) = \phi(s) = \rho^2$	$\phi(r)\phi(r) = (\rho)(\rho) = \rho^2$
s	$\phi((r)(s)) = \phi(id) = id$	$\phi(r)\phi(s) = (\rho)(\rho^2) = id$

Como vemos se cumple para $r \in S_3$, fijo.

Ahora fijemos $f \in S_3$, verifiquemos la condición 2.1:

f	$\phi((f)(\bullet))$	$\phi(f)\phi(\bullet)$
id	$\phi((f)(id)) = \phi(f) = \mu$	$\phi(f)\phi(id) = (\mu)(id) = \mu$
f	$\phi((f)(f)) = \phi(id) = id$	$\phi(f)\phi(f) = (\mu)(\mu) = id$
g	$\phi((f)(g)) = \phi(r) = \rho$	$\phi(f)\phi(g) = (\mu)(\mu\rho) = \rho$
h	$\phi((f)(h)) = \phi(s) = \rho^2$	$\phi(f)\phi(h) = (\mu)(\mu\rho^2) = \rho^2$
r	$\phi((f)(r)) = \phi(g) = \mu\rho$	$\phi(f)\phi(r) = (\mu)(\rho) = \mu\rho$
s	$\phi((f)(s)) = \phi(h) = \mu\rho^2$	$\phi(f)\phi(s) = (\mu)(\rho^2) = \mu\rho^2$

Para verificar con los otros elementos de S_3 lo realizaremos mediante nuestra implementación y los resultados veremos en el siguiente capítulo. □

Implementación del Isomorfismo

Para realizar la implementación del Isomorfismo en el lenguaje C++, vamos a utilizar la Clase Amiga. La nueva clase podrá obtener las variables y funciones de nuestra implementación interna de las clases S_3 y D_3. A continuación se detallará como realizamos la implementación.

Clase Amiga

Creamos la Clase Isomorfismo, la cual nos ayuda a implementar las funciones que admiten datos de las clases previamente programadas. En

este caso solo vamos a utilizar el entorno público. La declaramos como **Friend Classs** Isomorfismo en la **Clase S3** y **Clase D3**, con esto continuamos escribiendo el código respectivo.

Entorno Público

En este entorno vamos a declarar cuatro funciones que dependa de los objetos de S_3 y D_3, dos de las funciones es la implementación de isomorfismo entre S_3 y D_3, y el isomorfismo inverso. Las otras dos nos ayuda a calcular el resultado de la aplicación de isomorfismo a un producto de funciones.

Isomorfismo

Vamos a programar la función que vimos en la sección (), primero el código recibirá una función de S_3, esta va a realizar la evaluación en 1 y 2, obteniendo sus respectiva imagen. Estos valores pertenecen al conjunto $\{1, 2, 3\}$, a ellos se le restará una unidad para realizar la transformación a elementos que pertenezcan a $\{0, 1, 2\}$, que es la respectiva imagen de D_3. Con los resultados se identificará la función correspondiente a D_3 y obtenemos el resultado del isomorfismo.

Para el inverso de nuestro isomorfismo la implementación se basa en las mismas líneas de código descritas anteriormente, cambiando los datos de S_3 a D_3 y viceversa. Además en lugar de restar realizamos una suma.

Isomorfismo de un producto

Vamos a programar la función que vimos en la sección ().Con la condición que realice un producto internamente. Al comenzar la implementación recibirá dos función de S_3, la primera función realizará una evaluación en 1 y 2, los resultado serán nuevamente evaluados en la segunda función. Con el hallazgo de los valores se identificará la función correspondiente a S_3. Como este procedimiento es igual al implementado con el operador producto, sabemos que tenemos el producto de dos funciones, ahora apliquemos el isomorfismo, para lo cual utilizamos las mismas

instrucciones vistas anteriormente.

Para el isomorfismo inverso de un producto se realiza la misma implementación con las indicaciones de cambiar a los tipos correspondientes.

2.4. Creación de un menú

Del libro [1], tomamos varias ideas para la creación de nuestro menú y tener acceso a todas las opciones implementadas. Para la creación vamos a usar funciones declaradas en el tipo **void**. Mientras para que el menú permanezca estable usamos un ciclo **Do-While** con ellos si queremos terminar el programa digitaremos un carácter asignado y se terminará el programa o en el caso de un sub-menú nos regresará al principal. Finalmente crearemos los casos con la función **switch**, donde nos direccionará a las funciones para observar las operaciones.

Con la creación del menú también debemos crear otras subimplementaciones que serán las encargadas de llamar a nuestras clases previamente implementadas. Además crearemos una función que será de ingreso para nuestras funciones, recibirá datos en cadena y los transformará en datos de los elementos del grupo. Por ejemplo al escribir “id” en la entrada de datos, la implementación de ingreso la transforma en la función $id \in S_3$.

Menú Principal

Este menú nos muestra tres opciones:

1. Grupo S_3
2. Grupo D_3
3. Salir

Al escoger una de las dos primeras opciones nos direcciona a un submenú. En cambio si tomamos la opción salir, entonces se termina el programa.

Menú S_3

En este menú tenemos las siguientes opciones:

1. **Producto:** En esta opción nos pedirá dos elementos de S_3, para luego mostrar en consola los elementos ingresados y el producto de ellos.
2. **Elemento Inverso:** Ingresaremos un elemento de S_3 y nos mostrará su inverso.
3. **Orden del elemento:** Ingresaremos un elemento $a \in S_3$ y nos muestra el número entero positivo n tal que $a^n = id$
4. **Grupo cíclico:**
5. **Asociatividad:** Nos pedirá el ingreso de S_3 elementos para mostrarnos las operaciones relacionadas con la asociatividad y al final comprueba si son iguales.
6. **No Abeliano:** Al ingresar dos elementos de S_3 nos indica que existen elementos no conmutan.
7. **Tabla:** Nos muestra una tabla multiplicativa de todos los elementos de S_3.
8. **Tabla con ingreso de elementos:** Permite el ingreso manual de S_3 y con estos nos muestra una tabla multiplicativa de todos los elementos ingresados.
9. **Arreglo multiplicado con el ingreso:** Ingresaremos un elemento de S_3 y mostrará la multiplicación de este con cada elemento de S_3.
10. **Isomorfismo con ingreso:** Con el ingreso de un elemento de S_3 invoca al isomorfismo y nos entrega el elemento de D_3 correspondiente
11. **Producto Isomorfismo:** Mediante el ingreso de dos elementos de S_3, realiza la comprobación de la condición (2.1) .
12. **Tabla Comparativa Isomorfismo:** Con el ingreso de un elemento de S_3 se creará una pequeña tabla para comprobar la condición (2.1).
13. **Salir:** Sale del menú.

Menú D_3

En este menú tenemos las siguientes opciones:

1. **Producto:** En esta opción nos pedirá dos elementos de D_3, para luego mostrar en consola los elementos ingresados y el producto de ellos.
2. **Elemento Inverso:** Ingresaremos un elemento de S_3 y nos mostrará su inverso.
3. **Orden del elemento:** Ingresaremos un elemento $a \in D_3$ y nos muestra el número entero positivo n tal que $a^n = id$
4. **Grupo cíclico:**
5. **Asociatividad:** Nos pedirá el ingreso de D_3 elementos para mostrarnos las operaciones relacionadas con la asociatividad y al final comprueba si son iguales.
6. **No Abeliano:** Al ingresar dos elementos de S_3 nos indica que existen elementos no conmutan.
7. **Tabla:** Nos muestra una tabla multiplicativa de todos los elementos de D_3.
8. **Tabla con ingreso de elementos:** Permite el ingreso manual de D_3 y con estos nos muestra una tabla multiplicativa de todos los elementos ingresados.
9. **Arreglo multiplicado con el ingreso:** Ingresaremos un elemento de D_3 y mostrará la multiplicación de este con cada elemento de D_3.
10. **Isomorfismo con ingreso:** Con el ingreso de un elemento de D_3 invoca al isomorfismo y nos entrega el elemento de S_3 correspondiente
11. **Producto Isomorfismo:** Mediante el ingreso de dos elementos de D_3, realiza la comprobación de la condición (2.1) .
12. **Tabla Comparativa Isomorfismo:** Con el ingreso de un elemento de D_3 se creará una pequeña tabla para comprobar la propiedad (2.1).
13. **Salir:** Sale del menú.

Capítulo 3

Resultados, conclusiones y recomendaciones

3.1. Resultados

Al iniciar la implementación nos mostrará el siguiente menú, que nos llevará sub-menús dependiendo del grupo donde realicemos las operaciones, visualizamos lo siguiente:

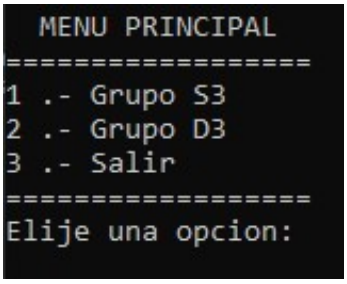
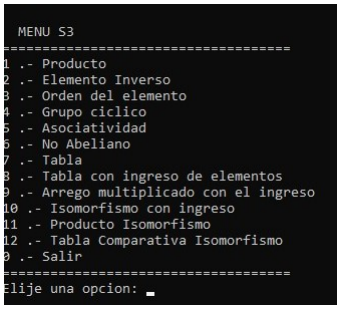
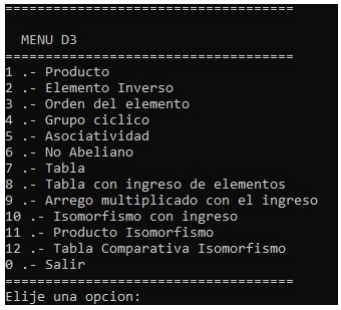


Figura 3.1: Menú Principal



(a) Sub-menú S_3



(b) Sub-menú D_3

Figura 3.2: Sub-Menús

3.1.1. Resultados del Grupo S_3

Las funciones que fueron implementadas inicialmente y representan las permutaciones de tres elementos. Estas funciones son los elementos de S_3, y nos ayudaron a la construcción de una clase en el sentido del lenguaje C++. En esta estructura se implementó diversos constructores, operadores y funciones que nos llevaron a obtener la operación producto de permutaciones y que como principal resultado obtuvimos la siguiente tabla de verdad:

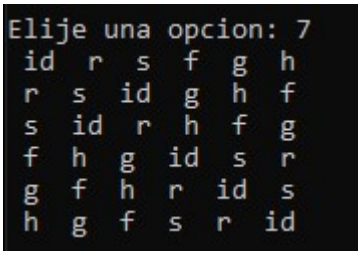


Figura 3.3: Tabla de Verdad de S_3

La tabla nos ayuda a tener fácil acceso a la operación composición sin realizar un proceso manual que resulta mas largo. Siguiendo el mismo procedimiento nos interesó visualizar una tabla que sea diferente en el sentido del orden, y con la idea implementamos un código que nos realice este procedimiento y obtuvimos lo siguiente:

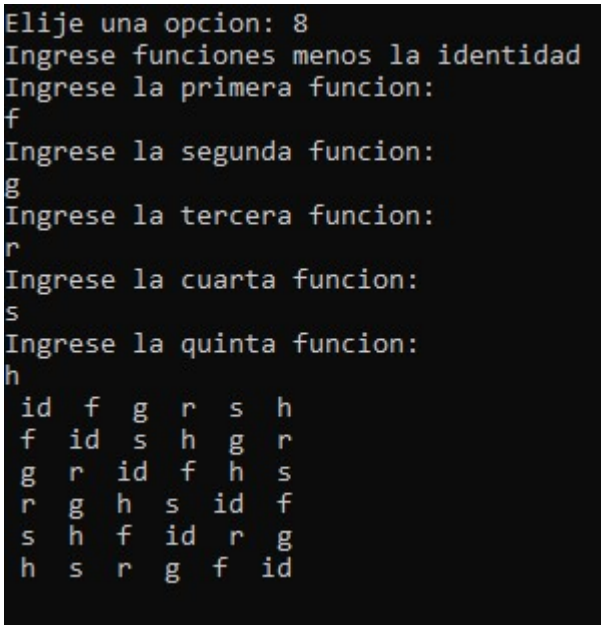


Figura 3.4: Tabla de Verdad de S_3 Modificable

El tratamiento anterior nos ayudó a la implementación que nos permite visualizar en pantalla el orden del elemento y su grupo cíclico asociado:

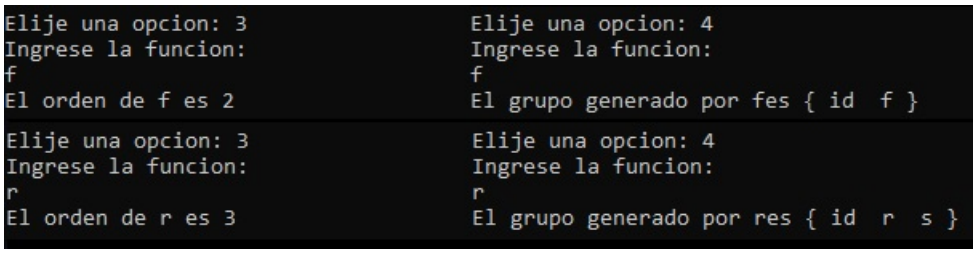


Figura 3.5: Orden de Elemento y Subgrupo Cíclico Asociado

Con el fin de conocer el inverso de cada elemento de S_3 se logró implementar un operador que mediante el ingreso de una función nos retorna su inverso. Presentemos los inversos que se visualiza con el ingreso de cada elemento de S_3:

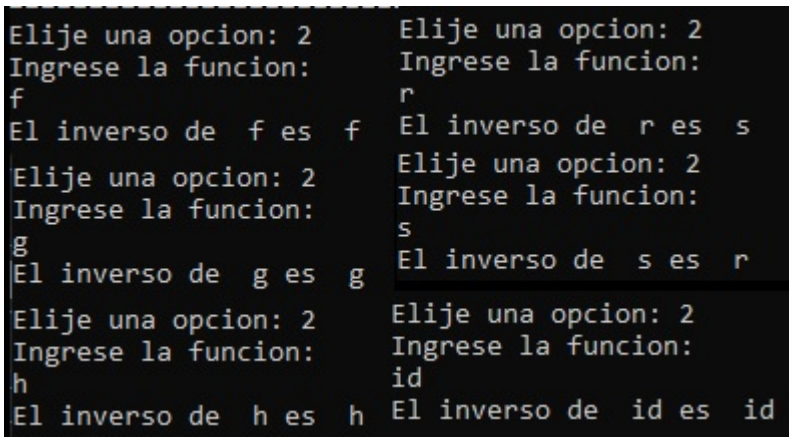


Figura 3.6: Elementos Inversos de S_3

Finalmente se implementó el operador igualdad para comparar que todas las funciones que se ingresaron fueran diferentes. Pero nos llevó a experimentar con dos propiedades que tiene el grupo S_3, la primera es conocer que todo elemento de S_3 tiene la propiedad asociativa, el código nos permite realizar esto con el ingreso de 3 elementos.

Debo acotar que se realiza con ingreso del usuario, puesto que si realizamos la comprobación con todos los elementos no se podría visualizar de forma completa.La otra propiedad es la existencia de algún elemento de S_3 que verifica que el grupo no es abeliano. Así, tenemos los siguientes resultados:

```
Elije una opcion: 5
Ingrese la primera funcion:
f
Ingrese la segunda funcion:
r
Ingrese la tercera funcion:
g
Analicemos:
( f* r)* g= f*( r* g)
     g* g= f* f
       id= id
Son iguales?
SI
```

```
Elije una opcion: 5
Ingrese la primera funcion:
r
Ingrese la segunda funcion:
f
Ingrese la tercera funcion:
g
Analicemos:
( r* f)* g= r*( f* g)
     h* g= r* r
       s= s
Son iguales?
SI
```

Figura 3.7: Propiedad Asociativa S_3

```
Elije una opcion: 6
Comprobacion que no es abeliano
Ingrese la primera funcion:
f
Ingrese la segunda funcion:
g
El producto de  f* g es distinto al producto de  g* f
Puesto que el primer producto es  r y el segundo producto es s
Elije una opcion: 6
Comprobacion que no es abeliano
Ingrese la primera funcion:
h
Ingrese la segunda funcion:
f
El producto de  h* f es distinto al producto de  f* h
Puesto que el primer producto es  r y el segundo producto es s
```

Figura 3.8: Grupo No Abeliano

3.1.2. Resultados del Grupo D_3

Para el grupo D_3, se realizó las mismas implementaciones y por tanto tenemos resultados similares pero se cambian las funciones asociadas a permutaciones con funciones asociadas a rotaciones y reflexiones de un triángulo rectángulo.Tenemos las dos tablas:

```
Elije una opcion: 7
  e   |  r90  | r180  |   u   |  ur90 | ur180 |
 r90  | r180  |   e   |  ur90 | ur180 |   u   |
r180  |   e   |  r90  | ur180 |   u   |  ur90 |
  u   | ur180 |  ur90 |   e   | r180  |  r90  |
 ur90 |   u   | ur180 |  r90  |   e   | r180  |
ur180 |  ur90 |   u   | r180  |  r90  |   e   |
```

Figura 3.9: Tabla de Verdad de D_3

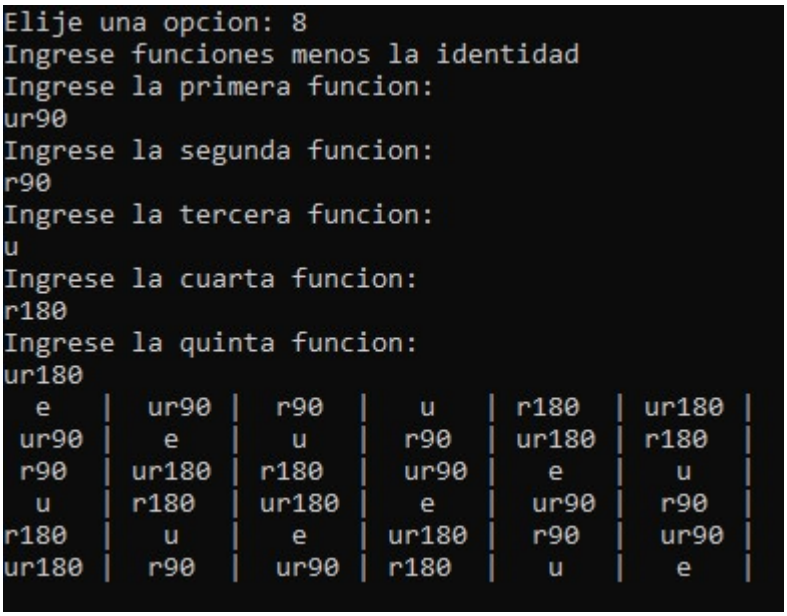


Figura 3.10: Tabla de Verdad de D_3 Modificable

Se tiene también el orden del elemento y su grupo cíclico asociado:

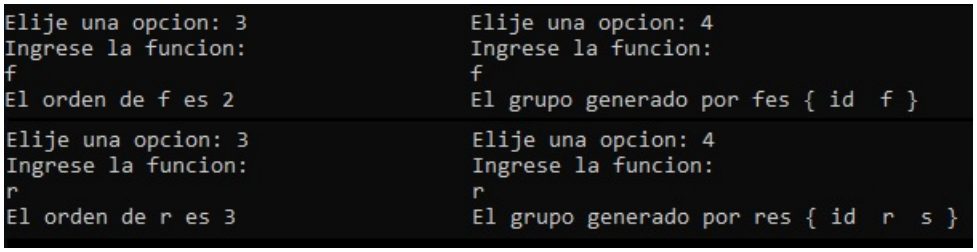


Figura 3.11: Orden de Elemento y Subgrupo Cíclico Asociado

La presentación de los inversos que se visualiza con el ingreso de cada elemento de D_3:

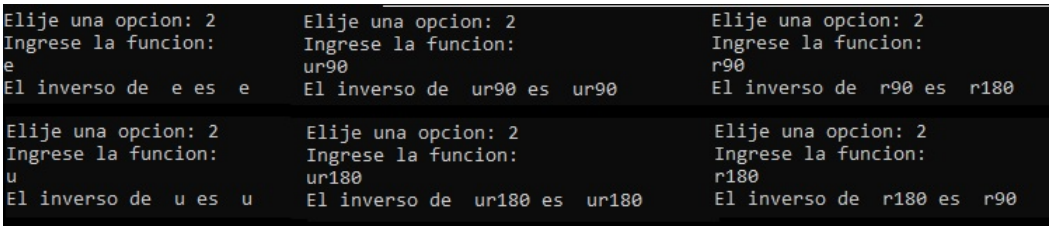


Figura 3.12: Elementos Inversos de D_3

El operador igualdad para comprobar las propiedades de asociatividad y que el grupo no es abeliano:

```
Elije una opcion: 5
Ingrese la primera funcion:
ur90
Ingrese la segunda funcion:
r90
Ingrese la tercera funcion:
r180
Analicemos:
( ur90* r90)* r180= ur90*( r90* r180)
 ur180* r180= ur90* e
 ur90= ur90
Son iguales?
SI
```

```
Elije una opcion: 5
Ingrese la primera funcion:
r90
Ingrese la segunda funcion:
r180
Ingrese la tercera funcion:
u
Analicemos:
( r90* r180)* u= r90*( r180* u)
 e* u= r90* ur90
 u= u
Son iguales?
SI
```

Figura 3.13: Propiedad Asociativa D_3

```
Elije una opcion: 6
Comprobacion que no es abeliano
Ingrese la primera funcion:
r90
Ingrese la segunda funcion:
u
El producto de  r90* u es distinto al producto de  u* r90
Puesto que el primer producto es  ur180 y el segundo producto es  ur90
Elije una opcion: 6
Comprobacion que no es abeliano
Ingrese la primera funcion:
r90
Ingrese la segunda funcion:
ur180
El producto de  r90* ur180 es distinto al producto de  ur180* r90
Puesto que el primer producto es  ur90 y el segundo producto es  u
```

Figura 3.14: Grupo No Abeliano

3.1.3. Resultado Isomorfismo

Los resultados anteriores nos llevaron a construir y visualizar el isomorfismo entre S_3 y D_3, primero mediante el ingreso de cada elemento de S_3 nos retornará la correspondiente función en D_3:

```
Elije una opcion: 10
Ingrese la primera funcion:
id
 e
Elije una opcion: 10
Ingrese la primera funcion:
r
 r90
Elije una opcion: 10
Ingrese la primera funcion:
s
 r180
```

```
Elije una opcion: 10
Ingrese la primera funcion:
f
 u
Elije una opcion: 10
Ingrese la primera funcion:
g
 ur90
Elije una opcion: 10
Ingrese la primera funcion:
h
 ur180
```

Figura 3.15: Isomorfismo entre S_3 y D_3

De forma similar con el ingreso de un elemento S_3 nos retornará la correspondiente función en D_3:

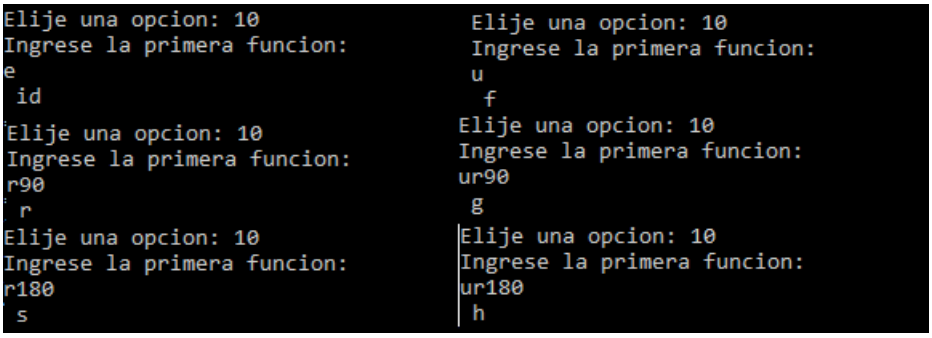


Figura 3.16: Isomorfismo entre D_3 y S_3

En el capítulo anterior estamos comprobando mediante tablas la condición (2.1) que se debe cumplir para que sea un isomorfismo. Así, conseguimos implementar para visualizar las siguientes tablas:

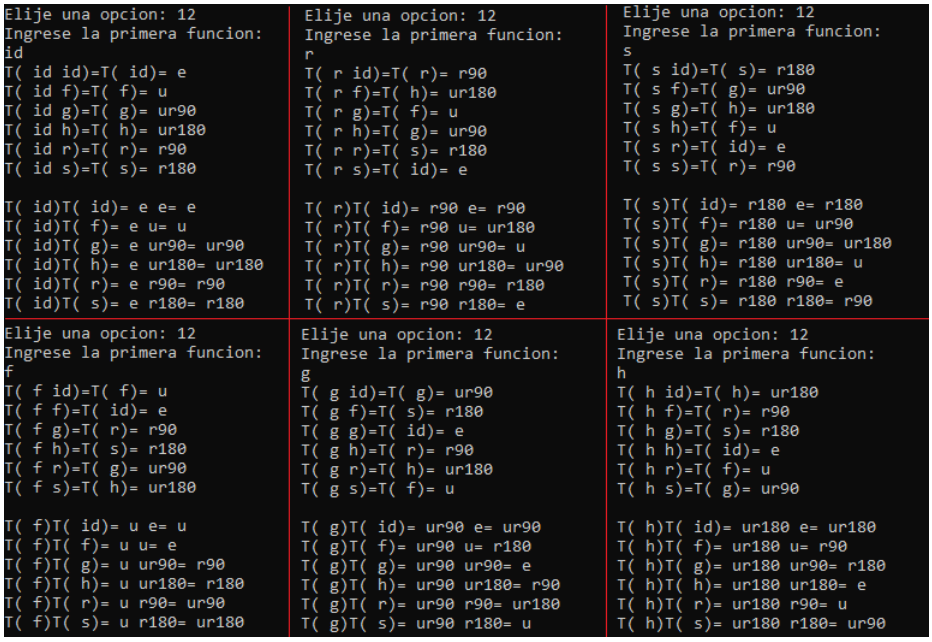


Figura 3.17: Condición de Isomorfismo S_3

Claramente se ve que se cumple dicha condición y por tanto nuestra función construida en el anterior capítulo es un isomorfismo. De igual manera para el isomorfismo inverso, que es la función que va de D_3 a S_3, tenemos:

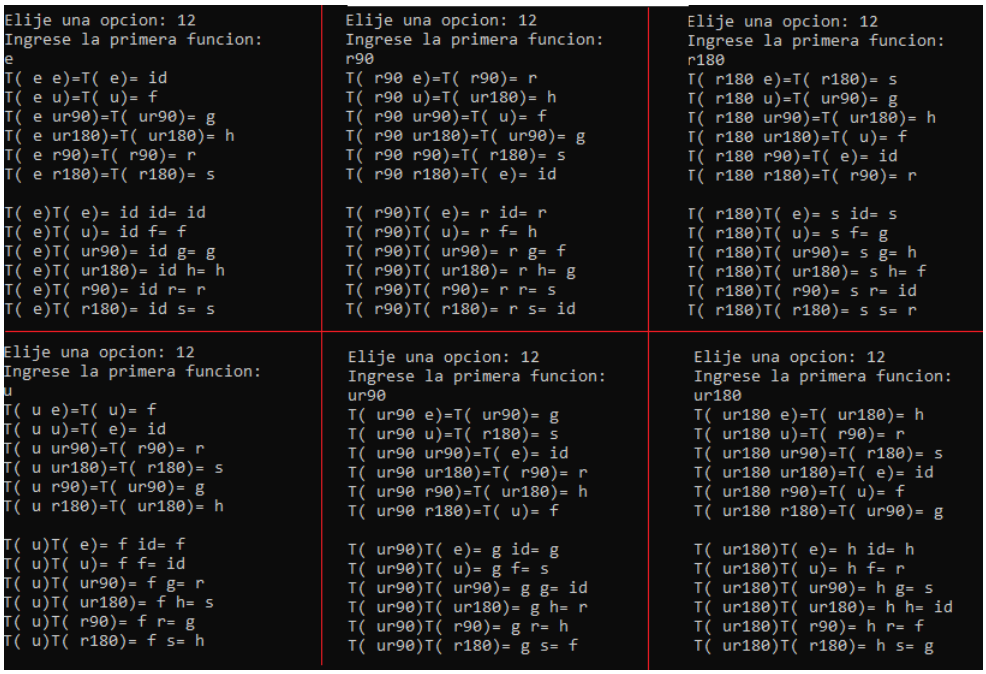


Figura 3.18: Condición de Isomorfismo D_3

Realizamos una última implementación que podemos ingresar dos elementos de S_3 y comprobar la condición (2.1), se tiene:

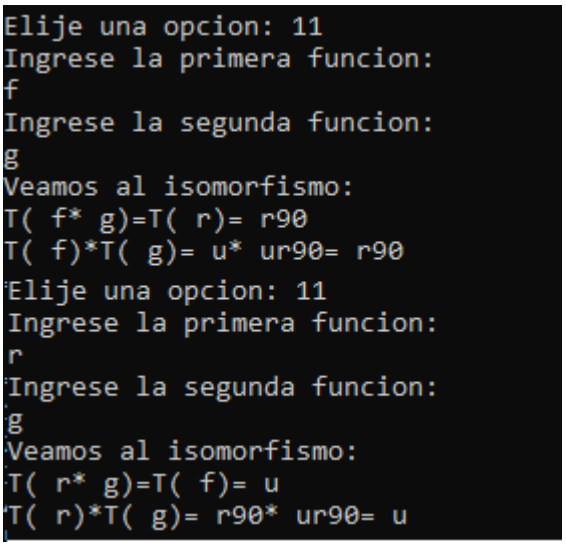


Figura 3.19: Condición Isomorfismo con Ingreso S_3

De igual manera realizamos con el ingreso de dos elementos de S_3 y comprobar la condición (2.1), se tiene:

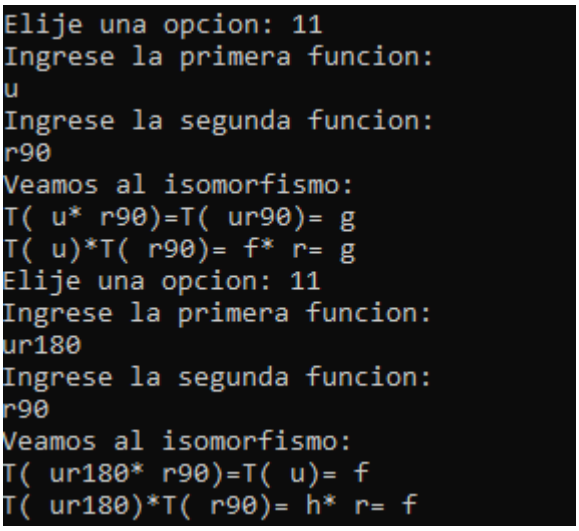


Figura 3.20: Condición Isomorfismo con Ingreso D_3

3.2. Conclusiones y recomendaciones

3.2.1. Conclusiones

Por medio de la bibliografía se logró obtener conceptos de la Teoría de Grupos y de Programación Orientada a Objetos. Por un lado, se introdujo definiciones de grupo, subgrupo, subgrupo cíclico, orden del grupo, orden del elemento, homomorfismo e isomorfismo. Con estos conceptos claros, se realizó la construcción de definiciones del grupo de permutaciones, en particular del grupo S_3 y del grupo Diedral, singularmente del grupo D_3. Por otro lado, se estudió las nociones de programación en C++, entre ellos están las abstracciones de tipos de datos, renombrar un tipo de dato, arreglos, funciones, clases y componentes de clase, junto a estos se colocaron implementaciones para su comprensión. Las concepciones que se realizaron nos ayudaron en la creación del programa donde se visualizaran propiedades de los grupos S_3 y D_3.

A continuación se crearon varias implementaciones relacionadas a las grupos S_3 y D_3, primero la construcción en base a la conceptualización que se realizó. Al mismo tiempo se creo las clases de S_3 y D_3, en donde se colocaron los mapas correspondientes a los grupos y con ello se creó varias instancias que realizaron las operaciones principales, como es el producto, elemento inverso e igualdad. Junto a ello, para la visualización

en pantalla se crearon funciones miembros que permitieron representar las funciones, tablas de verdad, orden de elemento y generación de grupo cíclico. Lo fundamental en lo ejecutado en las implementaciones fue llegar a la construcción de la clase amiga que realizó una interacción entre las clases S_3 y D_3 y se elaboró el isomorfismo entre ellas.

Una vez logrado el procesamiento de las clases construidas se creó menú y sub-menús de acceso a las operaciones con las cuales se realizó ejemplificaciones que presentamos en la sección de resultados. Visualizamos las tablas de verdad, los elementos inversos, el orden de elementos y la generación de subgrupos cíclicos, en estos dos últimos nos comprueban el teorema relacionado a la divisibilidad del orden de grupo con el orden del elemento. Finalmente con la construcción de la relación de las clases S_3 y D_3 se logró comprobar que la definición planteada del mapa entre S_3 y D_3 es un isomorfismo, además de exhibir la condición (2.1) mediante ingreso de funciones. Se destaca que con la construcción del isomorfismo entre S_3 y D_3 también le acompaña el isomorfismo entre D_3 y S_3. Cabe señalar que esta fue nuestra finalidad principal para la concepción del programa.

En este documento, en el capítulo de anexos se encuentran los códigos que se implementaron mediante el lenguaje de programación C++ a través de la plataforma Code::Blocks.Y también se incluirá en un formato magnético los ficheros utilizados.

3.2.2. Recomendaciones

- Se recomienda el estudio de grupos simétricos y diédricos de orden mayor a 3, al mismo tiempo la implementación en C++.
- Para cualquier desarrollo en C++ se sugiere el uso del software Code::Blocks dado que permite la programación en ficheros, lo cual concibe una programación ordenada y permitiendo el acceso a subcódigos.
- Se podría realizar la programación en lenguajes de alto nivel, donde se puede realizar con procedimientos simplificados y visualizaciones de mayor detalle.

Capítulo A

Anexos

A.1. Asociativad Grupo $\mathbb{Z}_n$

En este anexo se presentará la propiedad asociativa del Grupo $(Z_n, +_n)$, seguida de su demostración:

$$(a +_n b) +_n c = \begin{cases} a+b+c & \text{si} \quad a+b+c < n \\ a+b+c-n & \text{si} \quad a+b \geq n \text{ y } a+b+c < 2n \\ a+b+c-2n & \text{si} \quad a+b+c \geq 2n \end{cases}$$

Demostración. Sean $a, b, c \in \mathbb{Z}$, cualesquiera.

- Si $a+b+c < n$, tenemos que

$$a+b < n \quad \text{y} \tag{A.1a}$$
$$a+b = a +_n b. \tag{A.1b}$$

Además, de la hipótesis y de (A.1b) se sigue que $(a +_n b) + c < n$. Por tanto,

$$(a +_n b) +_n c = (a +_n b) + c$$

De donde y de (A.1b) se obtiene que:

$$(a +_n b) +_n c = (a + b) + c. \tag{A.2}$$

- Si $a+b \geq n$ y $a+b+c < 2n$, seguimos que

$$a +_n b = a + b - n \quad \text{y} \tag{A.3a}$$
$$a + b - n + c < n. \tag{A.3b}$$

En consecuencia, $(a +_n b) + c < n$. Además, junto con (A.3a) tenemos como resultado

$$\begin{aligned}(a +_n b) +_n c &= (a +_n b) + c \\ &= (a + b - n) + c.\end{aligned}$$

En resumen,

$$(a +_n b) +_n c = a + b + c - n. \tag{A.4}$$

- Si $a+b+c \geq 2n$ lo que implica que $a+b \geq 2n-c$. Como $\mathbb{Z}_n$ es finito y ordenado por consiguiente para cada $x \in \mathbb{Z}_n$ se cumple que $x \leq n-1$. En particular lo anterior se cumple para $c \in \mathbb{Z}_n$ y así

$$\begin{aligned}a + b &\geq 2n - c \\ &\geq 2n - (n-1) \\ &\geq n\end{aligned}$$

En definitiva,

$$a + b \geq n \quad \text{y} \tag{A.5a}$$
$$a +_n b = a + b - n. \tag{A.5b}$$

De (A.5b) y de la hipótesis se sigue que $(a +_n b) + c \geq n$ y por consiguiente

$$(a +_n b) +_n c = (a +_n b) + c - n \tag{A.6}$$

A causa de lo anterior, de (A.5a) y de (A.5b) concluimos que

$$(a +_n b) + c - n = (a + b - n) + c - n \tag{A.7}$$

Por (A.6) y (A.7) se finaliza con

$$(a +_n b) +_n c = a + b + c - 2n \tag{A.8}$$

En definitiva de (A.2), (A.4) y (A.8), se deduce que

$$(a +_n b) +_n c = \begin{cases} a+b+c & \textbf{si} \quad a+b+c < n \\ a+b+c-n & \textbf{si} \quad a+b \geq n \ \textbf{y} \ a+b+c < 2n \\ a+b+c-2n & \textbf{si} \quad a+b+c \geq 2n \end{cases}$$

□

A.2. Códigos

Funciones S_3

```
//Implementacion de Funciones de S3
int id(int n){return n;}
int f(int n){
    int m=0;
    if(n==1){return n;}
    else{m=n+1;
        if(m>3){return m-2;}
        else{return m;}}
}
int g(int n){
    int m=0;
    if(n==2){return n;}
    else{m=n+1;
        if(m>3){return m-3;}
        else{return m+1;}}
}
int h(int n){
    int m=0;
    if(n==3){return n;}
    else{m=n+2;
        if(m>3){return m-3;}
        else{return m-1;}}
}
int r(int n){int m=0;
    m=n+1;
    if(n<3){return m;}
    else{return n-2};}
int s(int n){int m=0;
    m=n-1;
    if(m<1){return m+3;}
    else{return m;}}
```

Funciones D_3

```
//Implementacion Suma Modulo 3
int sumd3(int n,int m){
    if(n+m<3)
        return n+m;
    else
        return n+m-3;
}
//Implementacion Inversos de Z3
int ind3(int n){
    if(n==0)
        return n;
    else
        return 3-n;
}
//Implementacion de Funciones de D3
int e(int n){return n;}
int r90(int n){
    return sumd3(n,1);
}
int r180(int n){
    int m=r90(n);
    return r90(m);
}
int u(int n){
    return ind3(n);
}
int ur180(int n){
    int m=u(n);
    return r90(m);
}
int ur90(int n){
    int m=u(n);
    return r180(m);
}
```

Clase Grupo

```
class Grupo{
public:
    virtual void mostrar()=0;
    virtual void tabla()=0;
    virtual void ciclico()=0;
    virtual int orden()=0;
    virtual void filaproducto()=0;
};
```

Clase Grupo S_3

```
1  //Declaracion Clase del Grupo S3
2  typedef int (*funcion_S3) (int);
3
4  class S3 : public Grupo
5  {
6      public:
7          //Constructores
8          S3(funcion_S3);
9          S3(funcion_S3,funcion_S3,funcion_S3,funcion_S3,
               funcion_S3,funcion_S3,funcion_S3);
10         S3(int,int);
11         //Operadores
12         bool operator== (const S3&) const;
13         bool operator!= (const S3&) const;
14         S3 operator* (const S3&) const;
15         S3 operator- ()const;
16         //Funciones Miembro
17         void mostrar();
18         void tabla();
19         void filaproducto();
20         void ciclico();
21         int orden();
22         static funcion_S3* permutacion;
23     private:
24         funcion_S3 funcion;
25         string nombre(int,int);
26     //Clase Amiga
27     friend class isomorfismo;
28 };
```

Clase Grupo D_3

```
1  //Declaracion Clase del Grupo D3
2  typedef int (*funcion_D3) (int);
3
4  class D3:public Grupo{
5      public:
6          //Constructores
7          D3(funcion_D3);
8          D3(funcion_D3,funcion_D3,funcion_D3,funcion_D3,
               funcion_D3,funcion_D3,funcion_D3);
9          D3(int,int);
10         //Operadores
11         bool operator== (const D3&) const;
12         bool operator!= (const D3&) const;
13         D3 operator* (const D3&) const;
14         D3 operator- ()const;
15         //Funciones Miembro
16         void mostrar();
17         void tabla();
18         void ciclico();
19         int orden();
20         void filaproducto();
21         static funcion_D3* permutacion;
22     private:
23         funcion_D3 funcion;
24         string nombre(int,int);
25         string nombret(int,int);
26     //Clase Amiga
27     friend class isomorfismo;
28 };
```

Implementación Clase Grupo S_3

```
1  //Implementacion Clase Grupo S3
2  //Inicializa arreglo dinamico
3  funcion_S3* S3::permutacion = new funcion_S3[6];
4  //Inicializa el Constructor para elegir funciones forma
       predeterminada
5  S3::S3(funcion_S3 T){
6      funcion = T;
7      permutacion[0] =id;
8      permutacion[1] =r;
9      permutacion[2] =s;
10     permutacion[3] =f;
11     permutacion[4] =g;
12     permutacion[5] =h;
13 }
14 //Inicializa el Constructor para elegir funciones forma
       ingreso de datos
15 S3::S3(funcion_S3 T,funcion_S3 T1,funcion_S3 T2,funcion_S3
       T3,funcion_S3 T4,funcion_S3 T5,funcion_S3 T6){
16     funcion = T;
17     permutacion[0] =T1;
18     permutacion[1] =T2;
19     permutacion[2] =T3;
20     permutacion[3] =T4;
21     permutacion[4] =T5;
22     permutacion[5] =T6;
23
24 }
25 //Constructor que determina la funcion asociada
26 S3::S3(int a,int b){
27     if(a==1){
28         if(b==2){funcion=id;}
29         else{funcion=f;}
30     }
31
```

```
32      else if(a==2){
33          if(b==1){funcion=h;}
34          else{funcion=r;}
35      }
36      else{
37          if(b==2){funcion=g;}
38          else{funcion=s;}
39      }
40  }
41  //Operadores
42  //Realiza el producto de permutaciones
43  S3 S3::operator* (const S3& t) const{
44      S3 s(funcion(t.funcion(1)),funcion(t.funcion(2)));
45      return s;
46  }
47  //Comprueba la igualdad de permutaciones
48  bool S3 :: operator== (const S3 & t) const{
49      if(funcion(1)==t.funcion(1) && funcion(2)==t.funcion(2)
           ){return true;}
50      else{return false;}
51  }
52   //Comprueba la diferencia de permutaciones
53  bool S3 :: operator!= (const S3 & t) const{
54      if(funcion(1)!=t.funcion(1) && funcion(2)!=t.funcion(2)
           ){return true;}
55      else{return false;}
56  }
57  //Extrae el inverso de una permutacion
58  S3 S3::operator- ()const{
59      for(int i=0;i<6;i++){
60          if(permutacion[i](funcion(1))==1 && permutacion[i](
               funcion(2))==2){return permutacion[i];}
61      }
62  }
```

```
//Funciones Miembro
//Nos muestra la funcion
void S3::mostrar(){
    int a,b;
    a=funcion(1);
    b=funcion(2);
    cout<<nombre(a,b);
}
//Devuelve el nombre de la funcion
string S3::nombre(int a, int b){
    if(a==1){
        if(b==2){return " id";}
        else{return " f";}
    }
    else if(a==2){
        if(b==1)}{return " h";}
        else{return " r";}
    }
    else{
        if(b==2){return " g";}
        else{return " s";}
    }
}
//Crea la tabla de verdad
void S3::tabla(){
    int a,b;
    for(int j=0;j<6;j++){
        for(int i=0;i<6;i++){
            a=permutacion[i](permutacion[j](1));
            b=permutacion[i](permutacion[j](2));
            cout<<nombre(a,b)<<" ";
        }
        cout<<endl;
    }
}
```

```
36 //Crea un vector fila de producto de permutaciones
37 void S3::filaproducto(){
38     int a,b;
39     cout<<"(";
40     for(int i=0;i<6;i++){
41         a=permutacion[i](funcion(1));
42         b=permutacion[i](funcion(2));
43         cout<<nombre(a,b)<<"␣";
44     }
45     cout<<")";
46 }
47 //Nos retorna el orden del elemento de una permutacion
48 int S3::orden(){
49     int a=1,b=2,contador=1;
50     if(funcion(a)==1 && funcion(b)==2){return contador;}
51     else{
52         do{
53             a=funcion(a);
54             b=funcion(b);
55             contador=contador+1;
56         }
57         while(funcion(a)!=1 && funcion(b)!=2);}
58     return contador;}
59 // Nos describe el grupo ciclico de una funcion ingresada
60 void S3::ciclico(){
61     int a=1,b=2;
62     cout<<"{"<<nombre(a,b)<<"␣";
63     do{
64         a=funcion(a);
65         b=funcion(b);
66         cout<<nombre(a,b)<<"␣";
67     }
68     while(funcion(a)!=1 && funcion(b)!=2);
69     cout<<"}";
70 }
```

Implementación Clase Grupo D_3

```
//Implementacion Clase Grupo D3
//Inicializa arreglo dinamico
funcion_D3* D3::permutacion = new funcion_D3[6];
//Inicializa el Constructor para elegir funciones forma
   predeterminada
D3::D3(funcion_D3 T){
    funcion = T;
    permutacion[0] =e;
    permutacion[1] =r90;
    permutacion[2] =r180;
    permutacion[3] =u;
    permutacion[4] =ur90;
    permutacion[5] =ur180;
}
//Inicializa el Constructor para elegir funciones forma
   ingreso de dato
D3::D3(funcion_D3 T,funcion_D3 T1,funcion_D3 T2,funcion_D3
   T3,funcion_D3 T4,funcion_D3 T5,funcion_D3 T6){
    funcion = T;
    permutacion[0] =T1;
    permutacion[1] =T2;
    permutacion[2] =T3;
    permutacion[3] =T4;
    permutacion[4] =T5;
    permutacion[5] =T6;

}
//Constructor que determina la funcion asociada
D3::D3(int a,int b){
    if(a==0){
        if(b==1){ funcion=e;}
        else{funcion=u;}
    }
    else if(a==1){
```

```
32          if(b==0){funcion=ur180;}
33          else{funcion=r90;}
34      }
35      else{
36          if(b==1){funcion=ur90;}
37          else{funcion=r180;}
38      }
39  }
40  //Operadores
41  //Realiza el producto de rotaciones y reflexiones
42  D3 D3::operator* (const D3 & t) const{
43      D3 s(funcion(t.funcion(0)),funcion(t.funcion(1)));
44      return s;
45  }
46  //Comprueba la igualdad de rotaciones y reflexiones
47  bool D3 :: operator== (const D3 & t) const{
48      if(funcion(0)==t.funcion(0) && funcion(1)==t.funcion(1)
           ){return true;}
49      else{return false;}
50  }
51  //Comprueba la diferencia de rotaciones y reflexiones
52  bool D3 :: operator!= (const D3 & t) const{
53      if(funcion(0)!=t.funcion(0) && funcion(1)!=t.funcion(1)
           ){ return true;}
54      else{return false;}
55  }
56  //Extrae el inverso de una rotacion o reflexion
57  D3 D3::operator- ()const{
58      for(int i=0;i<6;i++){
59          if(permutacion[i](funcion(0))==0 && permutacion[i](
               funcion(1))==1){ return permutacion[i];}
60      }
61  }
```

```
//Funciones Miembro
//Nos muestra la funcion
void D3::mostrar(){
    int a,b;
    a=funcion(0);
    b=funcion(1);
    cout<<nombre(a,b);
}
//Devuelve el nombre de la funcion
string D3::nombre(int a, int b){
    if(a==0){
        if(b==1){return " e";}
        else{return " u";}
    }
    else if(a==1){
        if(b==0){return " ur180";}
        else{return " r90";}
    }
    else{
        if(b==1){return " ur90";}
        else{return " r180";}
    }
}
//Crea la tabla de verdad
void D3::tabla(){
    int a,b;
    for(int j=0;j<6;j++){
        for(int i=0;i<6;i++){
            a=permutacion[i](permutacion[j](0));
            b=permutacion[i](permutacion[j](1));
            cout<<nombret(a,b)<<" | ";
        }
        cout<<endl;
    }
}
```

```
36 //Crea un vector fila de producto de rotaciones y
      permutaciones
37 void D3::filaproducto(){
38     int a,b;
39     cout<<"(";
40     for(int i=0;i<6;i++)
41     {
42         a=permutacion[i](funcion(0));
43         b=permutacion[i](funcion(1));
44         cout<<nombre(a,b)<<"␣";
45     }
46     cout<<")";
47 }
48 //Devuelve el nombre de la funcion para tabla de verdad
49 string D3::nombret(int a, int b){
50     if(a==0){
51         if(b==1){return "␣␣e␣␣";}
52         else{return "␣␣u␣␣";}
53     }
54     else if(a==1){
55         if(b==0){return "ur180";}
56         else{return "␣r90␣";}
57     }
58     else{
59         if(b==1){return "␣ur90";}
60         else{return "r180␣";}
61     }
62 }
63 //Crea un vector fila de producto de rotaciones y
      reflexiones
64 int D3::orden(){
65         int a=0,b=1,contador=1;
66         if(funcion(a)==0 && funcion(b)==1){return contador
              ;}
67         else{
68             do{
```

```
69                  a=funcion(a);
70                  b=funcion(b);
71                  contador=contador+1;
72              }
73              while(funcion(a)!=0 && funcion(b)!=1);}
74          return contador;
75      }
76      // Nos describe el grupo ciclico de una funcion
           ingresada
77      void D3::ciclico(){
78          int a=0,b=1;
79          cout<<"{"<<nombre(a,b)<<"␣";
80          do
81          {
82              a=funcion(a);
83              b=funcion(b);
84              cout<<nombre(a,b)<<"␣";
85          }
86          while(funcion(a)!=0 && funcion(b)!=1);
87          cout<<"}";
88      }
```

Clase Isomorfismo

```
1  //Definicion de Clase
2  class isomorfismo{
3  public:
4              D3 isom(S3 &t);
5              D3 isomd(S3 &t,S3 &v);
6              S3 isomm(D3 &t);
7              S3 isommd(D3 &t,D3 &v);
8      };
9      //Implementacion de Clase
10     D3 isomorfismo::isom(S3 &t){
11     D3 a(t.funcion(1)-1,t.funcion(2)-1);
12     return a;
13 }
14 S3 isomorfismo::isomm(D3 &t){
15     S3 a(t.funcion(0)+1,t.funcion(1)+1);
16     return a;
17 }
18 D3 isomorfismo::isomd(S3 &t,S3 &v){
19     S3 w(t.funcion(v.funcion(1)),t.funcion(v.funcion(2)));
20     D3 a(w.funcion(1)-1,w.funcion(2)-1);
21     return a;
22 }
23 S3 isomorfismo::isommd(D3 &t,D3 &v){
24     D3 w(t.funcion(v.funcion(0)),t.funcion(v.funcion(1)));
25     S3 a(w.funcion(0)+1,w.funcion(1)+1);
26     return a;
27 }
```

Ingreso de datos

```
funcion_S3 ingreso_s(string datos){
    if(datos=="id"){ return id;}
    else if(datos=="f"){ return f;}
    else if(datos=="g"){ return g}
    else if(datos=="h"){ return h;}
    else if(datos=="r"){ return r;}
    else if(datos=="s"){ return s;}
    else { cout<<"Error";menuprincipal();}
}
funcion_D3 ingreso_d (string datos){
    if(datos=="e"){ return e; }
    else if(datos=="r90"){ return r90; }
    else if(datos=="r180"){ return r180;}
    else if(datos=="u"){ return u;}
    else if(datos=="ur90"){ return ur90; }
    else if(datos=="ur180"){ return ur180; }
    else { cout<<"Error"; menuprincipal(); }
}
```

Menú

```
//Ingreso Menu Principal
int main(){
    menuprincipal();
    return 0;
}

//Menu Principal Visualizacion
void menuprincipal(){
    char opcion;
    do {
        cout << "\n  MENU PRINCIPAL" << endl;
        cout << "==================" << endl;
        cout << "1 .- Grupo S3" << endl;
        cout << "2 .- Grupo D3" << endl;
        cout << "3 .- Salir" << endl;
        cout << "==================" << endl;
        cout << "Elije una opcion: ";
        cin >> opcion;
        switch(opcion){
            case '1':
                grupoS3();
                break;
            case '2':
                grupoD3();
                break;
            case '3':
                cout << "\n\nFIN DEL PROGRAMA" << endl;
                break;
            default:
                cout << "\n\nOPCION NO VALIDA" << endl;
                break;}
    }while (opcion != '3');
}

```

```
35 //Menu Grupo S3
36 void grupoS3(){
37 int opcion;
38 do {
39     cout << endl;
40     cout << "====================================" << endl;
41     cout << "\n  MENU S3" << endl;
42     cout << "====================================" << endl;
43     cout << "1 . - Producto" << endl;
44     cout << "2 . - Elemento Inverso" << endl;
45     cout << "3 . - Orden del elemento" << endl;
46     cout << "4 . - Grupo ciclico" << endl;
47     cout << "5 . - Asociatividad" << endl;
48     cout << "6 . - No Abeliano" <<endl;
49     cout << "7 . - Tabla " << endl;
50     cout << "8 . - Tabla con ingreso de elementos" << endl;
51     cout << "9 . - Arrego multiplicado con el ingreso" <<
          endl;
52     cout << "10 . - Isomorfismo con ingreso" << endl;
53     cout << "11 . - Producto Isomorfismo" << endl;
54     cout << "12 . - Tabla Comparativa Isomorfismo" << endl;
55     cout << "0 . - Salir" << endl;
56     cout << "====================================" << endl;
57     cout << "Elije una opcion: ";
58     cin >> opcion;
59     switch(opcion){
60         case 1:
61             producto(1);
62             break;
63         case 2:
64             inverso(1);
65             break;
66         case 3:
67             orden(1);
68             break;
69         case 4:
```

```
70              ciclico(1);
71              break;
72          case 5:
73              asociativa(1);
74              break;
75          case 6:
76              abeliano(1);
77              break;
78          case 7:{
79              S3 i(id);
80              i.tabla();
81              break;}
82          case 8:
83              tabla2(1);
84              break;
85          case 9:
86              fila(1);
87              break;
88          case 10:
89              funcionIso(1);
90              break;
91          case 11:
92              ProductoIsom(1);
93              break;
94          case 12:
95              TablaIso1(1);
96              break;
97          case 0:
98              cout << "\n\nFIN DEL PROGRAMA" << endl;
99              break;
100         default:
101             cout << "\n\nOPCION NO VALIDA" << endl;
102             break;}
103     }while (opcion != 0);
104 }
105
```

```
106 //Menu Grupo D3
107 void grupoD3(){
108 int opcion;
109 do {
110     cout << endl;
111     cout << "====================================" << endl;
112     cout << "\n  MENU D3" << endl;
113     cout << "====================================" << endl;
114     cout << "1 .- Producto" << endl;
115     cout << "2 .- Elemento Inverso" << endl;
116     cout << "3 .- Orden del elemento" << endl;
117     cout << "4 .- Grupo ciclico" << endl;
118     cout << "5 .- Asociatividad" << endl;
119     cout << "6 .- No Abeliano" <<endl;
120     cout << "7 .- Tabla " << endl;
121     cout << "8 .- Tabla con ingreso de elementos" << endl;
122     cout << "9 .- Arrego multiplicado con el ingreso" <<
        endl;
123     cout << "10 .- Isomorfismo con ingreso" << endl;
124     cout << "11 .- Producto Isomorfismo" << endl;
125     cout << "12 .- Tabla Comparativa Isomorfismo" << endl;
126     cout << "0 .- Salir" << endl;
127     cout << "====================================" << endl;
128     cout << "Elije una opcion: ";
129     cin >> opcion;
130     switch(opcion){
131         case 1:
132             producto(2);
133             break;
134         case 2:
135             inverso(2);
136             break;
137         case 3:
138             orden(2);
139             break;
140         case 4:
```

```
141             ciclico(2);
142             break;
143         case 5:
144             asociativa(2);
145             break;
146         case 6:
147             abeliano(2);
148             break;
149         case 7:{
150             D3 i(id);
151             i.tabla();
152             break;}
153         case 8:
154             tabla2(2);
155             break;
156         case 9:
157             fila(2);
158             break;
159          case 10:
160             funcionIso(2);
161             break;
162         case 11:
163             ProductoIsom(2);
164             break;
165         case 12:
166             TablaIso1(2);
167             break;
168         case 0:
169             cout << "\n\nFIN␣DEL␣PROGRAMA" << endl;
170             break;
171         default:
172             cout << "\n\nOPCION␣NO␣VALIDA" << endl;
173             break;}
174     while (opcion != 0);
175 }
```

Referencias bibliográficas

[1] Javier Ceballos Sierra. *C/C++ Curso de Programación*. RA-MA Editorial, 2007.

[2] John Fraleigh and Neal Brand. *A First Course in Abstract Algebra*. Pearson Education, 2021.

[3] Israel Nathan Herstein. *Topics in Algebra*. John Wiley & Sons, 1975.

[4] Derek F. Holt, Bettina Eick, and Eamon A. O'Brien. *Discrete Mathematics and its Applications*. Chapman and Hall/CRC, 2005.

[5] Brian Kernighan and Dennisl Ritchie. *El Lenguaje de Programación C*. Pearson Educación, 1991.

[6] Jonathan D. H. Smith. *Introduction to Abstract Algebra*. Chapman & Hall/CRC, 2009.

[7] Bjarne Stroustrup. *The C++ Programming Language*. AT & T, 2000.