

**Instituto Politecnico Nacional**  
**Escuela Superior de Computo**

**Materia: Teoría de la Computación**  
**Grupo: 5BM1**

**Profesor: Genaro Juarez Martínez**  
**Periodo: 2024/02**

**Practica 01:**

**Generador de potencias de un alfabeto binario.**

**Realizado por:**  
**Carrillo Barreiro José Emiliano**

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

**Escuela Superior de Computo**  
**Fecha: 15 de marzo de 2024**

# Índice

<b>1</b>	<b>Resumen.</b>	<b>3</b>
<b>2</b>	<b>Introducción.</b>	<b>3</b>
<b>3</b>	<b>Marco Teorico.</b>	<b>3</b>
3.1	Teoría de Automáatas. . . . .	3
3.2	Alfabetos. . . . .	4
3.3	Cadena de caracteres. . . . .	4
3.4	Cadena vacío. . . . .	4
3.5	Longitud de cadenas. . . . .	4
3.6	Potencias de un alfabeto. . . . .	4
3.7	Concatenacion de cadenas. . . . .	4
3.8	Lenguajes. . . . .	4
<b>4</b>	<b>Desarrollo</b>	<b>5</b>
4.1	Especificaciones . . . . .	5
4.2	Lenguajes . . . . .	5
4.2.1	C++ . . . . .	6
4.2.2	Matlab . . . . .	6
4.3	Librerias . . . . .	7
4.4	Implementación . . . . .	8
4.4.1	Libreria . . . . .	8
4.4.2	Main . . . . .	9
4.4.3	Menu . . . . .	9
4.4.4	Solicitudes . . . . .	13
4.4.5	Logica del problema . . . . .	14
4.4.6	Llamada a graficar . . . . .	15
4.4.7	Graficar . . . . .	16
<b>5</b>	<b>Resultados.</b>	<b>18</b>
<b>6</b>	<b>Conclusiones</b>	<b>19</b>

# Índice de Listados de C++

1	PRACTICA01.h . . . . .	8
2	mainP01.cpp — main () . . . . .	9
3	menuP01.cpp — menu () . . . . .	10
4	menuP01.cpp — ingresarOpcion () . . . . .	11
5	menuP01.cpp — ejecucion () . . . . .	11
6	menuP01.cpp — medirTiempo () . . . . .	12
7	solicitudesP01.cpp — <i>ingresarPotencia</i> () . . . . .	13
8	solicitudesP01.cpp — randomPotencia () . . . . .	13
9	solicitudesP01.cpp — ingresarCaracteres () . . . . .	14
10	solicitudesP01.cpp — opcionGraficar () . . . . .	14
11	P01.h — generarCombinaciones () . . . . .	14
12	P01.h — graficar () . . . . .	15
13	graficoMatlab.m — Script . . . . .	16

# Índice de Listados de MATLAB

1	Código en MATLAB . . . . .	16
---	----------------------------	----

## Índice de figuras

1	Ejecucion de la Practica 01 en terminal. . . . .	18
2	Grafico resultante con potencia = 28 . . . . .	19

# 1 Resumen.

La Práctica 01 combina la generación de combinaciones en C++ con la visualización de datos en MATLAB. En C++, se desarrolla un programa que permite al usuario ingresar la potencia deseada, los caracteres del alfabeto y opcionalmente generar una potencia aleatoria, generando luego todas las combinaciones correspondientes y guardándolas en archivos de texto. Posteriormente, en MATLAB, se lee y traza estos datos, mostrando la cantidad de puntos por cadena y el logaritmo de la cantidad de puntos por cadena en dos subgráficos separados.

## 2 Introducción.

Este reporte presenta un análisis detallado sobre la implementación de generadores de potencias en C++ y MATLAB, centrándose en los conceptos fundamentales de la teoría de autómatas. Exploraremos cómo estos lenguajes de programación pueden utilizarse para diseñar y construir generadores de potencias eficientes, aprovechando conceptos como los alfabetos, las cadenas de caracteres, la longitud de una cadena, potencias de un alfabeto y lenguajes.

La comprensión de estos conceptos es crucial para el diseño y la implementación efectiva de generadores de potencias, ya que proporcionan un marco teórico sólido para abordar problemas relacionados con la generación de secuencias de caracteres en sistemas computacionales. A través de la combinación de la teoría de autómatas con las capacidades de programación de C++ y MATLAB, buscamos no solo desarrollar generadores de potencias eficientes, sino también profundizar en la comprensión de los fundamentos teóricos que subyacen a su funcionamiento.

## 3 Marco Teorico.

### 3.1. Teoría de Automátas.

La teoría de autómatas es un campo fundamental en la ciencia de la computación que se ocupa del estudio de dispositivos abstractos de cálculo. Estos dispositivos, conocidos como *máquinas*, son modelos matemáticos que nos permiten entender y analizar el comportamiento de sistemas computacionales. Desde sus primeros pasos en la década de 1930 con los trabajos de Alan Turing, la teoría de autómatas ha evolucionado y se ha ramificado en diversos aspectos que abarcan desde autómatas finitos simples hasta conceptos más complejos como gramáticas formales y problemas computacionales.

El punto de partida de la teoría de autómatas se encuentra en los estudios de Turing sobre las *máquinas de Turing*, dispositivos abstractos capaces de realizar cualquier cálculo computacional. Turing propuso estas máquinas como un modelo universal de computación, estableciendo así los fundamentos teóricos de lo que hoy entendemos como computación. A partir de este trabajo pionero, otros investigadores comenzaron a explorar variantes más simples de las máquinas de Turing, dando lugar a los autómatas finitos.

Los autómatas finitos son modelos computacionales simples que representan sistemas con un número limitado de estados y una capacidad limitada de procesamiento. Estos dispositivos se utilizan para modelar sistemas de control, reconocer patrones en cadenas de símbolos y resolver problemas de decisión. Además, los autómatas finitos están estrechamente relacionados con las gramáticas formales, ya que ambos se utilizan para describir y generar lenguajes formales.

La teoría de autómatas también abarca el estudio de problemas computacionales y la clasificación de su complejidad. Investigadores como Stephen Cook han contribuido significativamente al campo al desarrollar técnicas para clasificar los problemas en función de su dificultad computacional. Esta clasificación ha llevado a la identificación de problemas que pueden resolverse eficientemente y problemas que son inherentemente difíciles de resolver.[1]

### 3.2. Alfabetos.

Un alfabeto, representado convencionalmente por el símbolo  $\Sigma$ , es un conjunto finito y no vacío de símbolos. Estos símbolos pueden ser números, letras, caracteres especiales, o cualquier otro tipo de elemento que se utilice para formar cadenas de caracteres. Algunos ejemplos comunes de alfabetos incluyen:

1.  $\Sigma = \{0, 1\}$ : el alfabeto binario, utilizado en sistemas informáticos para representar datos de manera binaria.
2.  $\Sigma = \{a, b, \dots, z\}$ : el conjunto de todas las letras minúsculas del alfabeto latino.
3. El conjunto de todos los caracteres ASCII o el conjunto de todos los caracteres ASCII imprimibles, utilizado en programación y comunicaciones para representar texto y caracteres especiales.[1]

### 3.3. Cadena de caracteres.

Una cadena de caracteres, también conocida como palabra en algunos contextos, es una secuencia finita de símbolos seleccionados de un alfabeto específico. Por ejemplo, la cadena «01101» es una cadena del alfabeto binario  $\Sigma = \{0, 1\}$ , mientras que «hola» es una cadena del alfabeto de letras minúsculas del alfabeto latino. Incluso la cadena vacía, representada por  $\varepsilon$ , es una cadena que puede construirse en cualquier alfabeto.[1]

### 3.4. Cadena vacío.

La cadena vacía es aquella que no contiene ningún símbolo y se denota por  $\varepsilon$ . Aunque no contiene símbolos, sigue siendo una cadena válida y puede ser considerada como el elemento neutro de la concatenación de cadenas.[1]

### 3.5. Longitud de cadenas.

La longitud de una cadena se refiere al número de símbolos que contiene. Por ejemplo, la cadena «01101» tiene una longitud de 5, mientras que la cadena vacía tiene una longitud de 0. La notación estándar para indicar la longitud de una cadena  $w$  es  $|w|$ . Por lo tanto,  $|01101| = 5$  y  $|\varepsilon| = 0$ . [1]

### 3.6. Potencias de un alfabeto.

Si  $\Sigma$  es un alfabeto, podemos expresar el conjunto de todas las cadenas de una determinada longitud de dicho alfabeto utilizando una notación exponencial. Definimos  $\Sigma^k$  como el conjunto de todas las cadenas de longitud  $k$ , donde cada símbolo de la cadena pertenece a  $\Sigma$ . [1]

### 3.7. Concatenación de cadenas.

La concatenación de dos cadenas  $x$  e  $y$ , denotada como  $xy$ , consiste en unir la cadena  $x$  seguida de la cadena  $y$ . Por ejemplo, si  $x = "abc"$  y  $y = "def"$ , entonces  $xy = "abcdef"$ . Esta operación es fundamental en la manipulación de cadenas y es comúnmente utilizada en la construcción de algoritmos y programas.[1]

### 3.8. Lenguajes.

Un conjunto de cadenas, todas ellas seleccionadas de un  $\Sigma^*$ , donde  $\Sigma$  es un determinado alfabeto, se denomina lenguaje. Si  $\Sigma$  es un alfabeto y  $L \subseteq \Sigma^*$ , entonces  $L$  es un lenguaje de  $\Sigma$ . Los lenguajes pueden interpretarse como conjuntos de cadenas válidas según ciertos criterios. Por ejemplo, en el lenguaje inglés, el conjunto de palabras válidas representa un lenguaje. Del mismo modo, en programación, el conjunto de programas válidos en un lenguaje de programación determinado constituye un lenguaje. Los lenguajes pueden ser finitos o infinitos, dependiendo de la cantidad de cadenas que los componen.[1]

## 4 Desarrollo de la Practica.

### 4.1. información del sistema.

La siguiente información fue extraida gracias al comando systeminfo en cmd:

```
1 Nombre de host: CARBAJE
2 Nombre del sistema operativo: Microsoft Windows 11 Home
3 Version del sistema operativo: 10.0.22631 N/D Compilacion 22631
4 Fabricante del sistema operativo: Microsoft Corporation
5 Configuracion del sistema operativo: Estacion de trabajo independiente
6 Tipo de compilacion del sistema operativo: Multiprocessor Free
7 Propiedad de: emi.cruzazul@hotmail.com
8 Organizacion registrada:
9 Fecha de instalacion original: 04/03/2024, 8:09:03
10 Tiempo de arranque del sistema: 12/03/2024, 9:41:56
11 Fabricante del sistema: GIGABYTE
12 Modelo el sistema: G5 KF5
13 Tipo de sistema: x64-based PC
14 Procesador(es): 1 Procesadores instalados.
15 [01]: Intel64 Family 6 Model 186
Stepping 2 GenuineIntel ~2400 Mhz
16 Version del BIOS: INSYDE Corp. FD06, 03/11/2023
17 Directorio de Windows: C:\Windows
18 Directorio de sistema: C:\Windows\system32
19 Dispositivo de arranque: \Device\HarddiskVolume1
20 Configuracion regional del sistema: es;Espanol (internacional)
21 Idioma de entrada: en-us;Ingles (Estados Unidos)
22 Zona horaria: (UTC-06:00) Guadalajara, Ciudad de
Mexico, Monterrey
23 Cantidad total de memoria fisica: 16.088 MB
24 Memoria fisica disponible: 7.991 MB
25 Memoria virtual: tamano maximo: 65.240 MB
26 Memoria virtual: disponible: 53.369 MB
27 Memoria virtual: en uso: 11.871 MB
28 Ubicacion(es) de archivo de paginacion: C:\pagefile.sys
29 Dominio: WORKGROUP
30 Servidor de inicio de sesion: \\CARBAJE
31 Revision(es): 5 revision(es) instaladas.
32 [01]: KB5034467
33 [02]: KB5027397
34 [03]: KB5036212
35 [04]: KB5034848
36 [05]: KB5035226
37 Tarjeta(s) de red: 3 Tarjetas de interfaz de red
instaladas.
38 [01]: Realtek PCIe GbE Family
Controller
Nombre de conexion: Ethernet
Estado: Medios
desconectados
39 [02]: Bluetooth Device (Personal Area
Network)
Nombre de conexion: Conexion
de red Bluetooth
Estado: Medios
desconectados
40 [03]: Intel(R) Wi-Fi 6E AX211 160MHz
Nombre de conexion: Wi-Fi
DHCP habilitado: Si
```

### 4.2. Lenguajes usados.

la elección de C++ se basa en su eficiencia y control sobre los recursos, ideal para la implementación eficiente del generador de potencias. MATLAB se selecciona por su facilidad de prototipado y potentes herramientas de cálculo numérico y visualización, facilitando el análisis y la comprensión de los resultados.

Juntos, estos lenguajes ofrecen una solución integral para diseñar, implementar y analizar el generador de potencias de manera eficaz y precisa.

#### 4.2.1. C++

La elección de C++ para la elaboración de la lógica del programa se basa en su eficiencia, control sobre los recursos, flexibilidad y compatibilidad. Estas características hacen que sea una opción sólida y adecuada para implementar un generador de potencias de manera óptima y eficiente. A continuación se enlista a detenimiento las ventajas mencionadas:

1. **Eficiencia y rendimiento:** C++ es conocido por ser un lenguaje de programación de alto rendimiento. Esto significa que los programas escritos en C++ tienden a ejecutarse más rápido y a consumir menos recursos que aquellos escritos en lenguajes de más alto nivel, como MATLAB. Dado que estamos trabajando en la implementación de un generador de potencias, donde la eficiencia es crucial, el uso de C++ puede garantizar un rendimiento óptimo.
2. **Control sobre los recursos:** C++ proporciona un alto grado de control sobre la gestión de memoria y otros recursos del sistema. Esto es especialmente importante en aplicaciones donde se manejan grandes volúmenes de datos o se realizan operaciones intensivas en términos de recursos. En el contexto de la generación de potencias, donde podríamos estar trabajando con grandes conjuntos de datos, este control adicional puede ser beneficioso.
3. **Flexibilidad y versatilidad:** C++ es un lenguaje multiparadigma que permite programar en diferentes estilos, como programación orientada a objetos, programación genérica y programación procedural. Esta versatilidad ofrece la posibilidad de diseñar y estructurar el código de manera óptima según las necesidades específicas del problema. En el caso de la implementación de un generador de potencias, esta flexibilidad puede ser útil para organizar y modularizar el código de manera eficiente.
4. **Compatibilidad y portabilidad:** C++ es un lenguaje ampliamente utilizado y está disponible en una amplia variedad de plataformas y sistemas operativos. Esto garantiza que el código desarrollado en C++ pueda ejecutarse en diferentes entornos sin mayores modificaciones, lo que aumenta la portabilidad y la interoperabilidad de la solución.

#### 4.2.2. Matlab

La elección de MATLAB para la elaboración de la graficación del programa se basa en su facilidad de prototipado, sus capacidades avanzadas de cálculo numérico y matemático, sus herramientas de visualización y su integración con otras herramientas. Estas características hacen que sea una opción sólida y eficaz para la representación visual y el análisis de los resultados del generador de potencias. A continuación se enlista a detenimiento las ventajas mencionadas:

1. **Facilidad de prototipado y desarrollo rápido:** MATLAB es conocido por su capacidad para el prototipado rápido y el desarrollo eficiente de algoritmos. Proporciona una amplia gama de funciones y herramientas integradas que facilitan la implementación de algoritmos complejos con un código más compacto y legible. Esto es especialmente útil en el contexto de la práctica, donde la experimentación y la iteración rápida son fundamentales para el diseño y la optimización del generador de potencias.
2. **Amplio conjunto de herramientas para cálculos numéricos y matemáticos:** MATLAB ofrece una amplia gama de funciones y herramientas especializadas para realizar cálculos numéricos y matemáticos avanzados. Esto incluye funciones para operaciones con matrices, álgebra lineal, transformadas, optimización y simulación, entre otros. Estas herramientas son fundamentales para la manipulación y el procesamiento de datos en el contexto de la generación de potencias.
3. **Gráficos y visualización avanzados:** MATLAB cuenta con potentes capacidades de gráficos y visualización que facilitan la representación visual de datos y resultados. Esto es especialmente importante en el contexto de la práctica, donde es probable que se desee visualizar y analizar los resultados del generador de potencias, como patrones de secuencias generadas o análisis de

rendimiento. Las capacidades de graficación de MATLAB permiten crear gráficos personalizados y visualizaciones interactivas para explorar y comprender mejor los datos generados por el programa.

4. **Integración con herramientas adicionales:** MATLAB se integra bien con otras herramientas y entornos de desarrollo, lo que facilita la incorporación de funcionalidades adicionales o la conexión con sistemas externos si es necesario. Esto puede ser útil para ampliar la funcionalidad del generador de potencias o integrarlo en un flujo de trabajo más amplio.

### 4.3. Librerías usadas

Las siguientes bibliotecas proporcionan las herramientas necesarias para implementar eficientemente un generador de potencias en C++, abordando aspectos clave como la manipulación de cadenas, la lectura y escritura de archivos, el rendimiento y la optimización del tiempo de ejecución, la manipulación de datos en estructuras dinámicas y el control del formato de salida.

#### 1. **iostream:**

- Esencial para interactuar con el usuario a través de la entrada y salida estándar, lo que facilita la comunicación con el programa.

#### 2. **string:**

- Permite la manipulación eficiente de cadenas de caracteres, lo que es fundamental para procesar y generar secuencias de texto en el generador de potencias.

#### 3. **fstream:**

- Necesaria para leer y escribir archivos, lo que facilita la entrada y salida de datos y la posibilidad de almacenar resultados en archivos para su posterior análisis.

#### 4. **cstdlib:**

- Proporciona funciones de utilidad para realizar operaciones comunes de bajo nivel, como conversiones de tipos y gestión de memoria, que pueden ser útiles en la implementación del generador de potencias.

#### 5. **ctime:**

- Permite el manejo de operaciones relacionadas con el tiempo, como la obtención de la hora actual del sistema, lo que puede ser útil para realizar mediciones de tiempo y optimizar el rendimiento del generador de potencias.

#### 6. **vector:**

- Proporciona una estructura de datos dinámica que puede cambiar de tamaño automáticamente, lo que es útil para almacenar y manipular conjuntos de datos variables en la implementación del generador de potencias.

#### 7. **chrono:**

- Permite medir el tiempo de ejecución de partes específicas del programa, lo que es útil para evaluar el rendimiento del generador de potencias y optimizar su eficiencia.

#### 8. **windows.h:**

- Proporciona acceso a funciones del sistema operativo Windows y manipulación de recursos del sistema, lo que puede ser útil si se desarrolla el generador de potencias específicamente para la plataforma Windows.

#### 9. **iomanip:**

- Permite controlar el formato de salida al imprimir datos en la consola, lo que es útil para presentar los resultados del generador de potencias de manera legible y estructurada.

#### 10. **math.h:**



- Facilita el acceso a funciones matemáticas comunes, como operaciones trigonométricas y exponenciales, que pueden ser necesarias en el cálculo de potencias y otras operaciones matemáticas en el generador de potencias.

#### 11. algorithm:

- Ofrece una variedad de algoritmos para operar en contenedores de datos, como buscar, ordenar y manipular elementos, lo que facilita la implementación de operaciones complejas en el generador de potencias.

## 4.4. Implementación

### 4.4.1. PRACTICA01.h

Este archivo, también llamado biblioteca a partir de ahora, tiene solamente una bolque de código donde fragmento de código establece las bases para el programa, incluyendo las bibliotecas necesarias y la declaración de las funciones que se utilizarán. La implementación específica de cada función y la lógica del programa se encuentra en otros lugares del código. A continuación se deja el bloque de código de la biblioteca:

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cstdlib>
5  #include <ctime>
6  #include <vector>
7  #include <chrono>
8  #include <windows.h>
9  #include <iomanip>
10 #include <math.h>
11 #include <algorithm>
12 // #include "matplotlibcpp.h"
13
14 //funciones
15 void graficar(int&);
16 void opcionGraficar(int&);
17 void medirTiempo(std::chrono::high_resolution_clock::time_point);
18 void mostrarBarraProgreso(int&, int&);
19 void generarCombinaciones(std::ofstream&, std::ofstream&, int, int&, int&,
20 char, char, std::string, int);
21 void ejecucion(int&, int, char&, char&);
22 void ingresarPotencia(int&);
23 void randomPotencia(int&);
24 void ingresarCaracteres(char&, char&);
25 int ingresarOpcion();
26 void menu(int&);
27 int main();
```

Listing 1: PRACTICA01.h

A grandes rasgos. Esta biblioteca realiza lo siguiente:

#### 1. Inclusión de bibliotecas:

- Se incluyen diversas bibliotecas estándar de C++ que proporcionan funcionalidades específicas para el programa. Estas bibliotecas incluyen `iostream`, `string`, `fstream`, `cstdlib`, `ctime`, `vector`, `chrono`, `windows.h`, `iomanip`, `math.h` y `algorithm`. Cada una de estas bibliotecas proporciona conjuntos de funciones y herramientas para diferentes propósitos, como entrada/-salida, manipulación de cadenas, manipulación de archivos, generación de números aleatorios, cálculos matemáticos, manejo de tiempo, etc.

#### 2. Declaraciones de funciones:

- Se declaran varias funciones que se utilizarán en el programa. Estas funciones incluyen `graficar()`, `opcionGraficar()`, `medirTiempo()`, `mostrarBarraProgreso()`, `generarCombinaciones()`, `ejecucion()`, `ingresarPotencia()`, `randomPotencia()`, `ingresarCaracteres()`,

`ingresarOpcion ()` y `menu ()`. Cada una de estas funciones lleva un parámetro que indica el estado de ejecución del programa.

### 3. Función `main ()`:

- Se define la función principal `main ()`, que sirve como punto de entrada del programa.
- No se proporciona la implementación de la función `main ()` en este fragmento de código. Se define en otro lugar del archivo o en archivos adicionales del proyecto.

#### 4.4.2. *mainP01.cpp*

En este archivo existe únicamente una función, el siguiente fragmento de código inicia la ejecución del programa, llama a una función `menu ()` para presentar al usuario un menú de opciones, y devuelve un valor de error al sistema operativo al finalizar la ejecución. La lógica específica del programa, incluyendo la implementación de la función `menu ()`, se encuentra en otros archivos que están incluidos a través del archivo de encabezado «`PRACTICA01.h`». A continuación se muestra la función `main ()` en cuestión:

```
1 //MAIN
2 #include "PRACTICA01.h"
3
4 int main(){
5     int error = 0;
6     menu(error);
7
8     return error;
9 }
```

Listing 2: *mainP01.cpp* — `main()`

A grandes rasgos, este fragmento de código realiza lo siguiente:

#### 1. Inclusión de archivos de encabezado:

- Se incluye el archivo de encabezado «`PRACTICA01.h`», que contiene las declaraciones de funciones y/o clases utilizadas en el programa.

#### 2. Definición de la función principal `main ()`:

- Se define la función `main ()` que servirá como el punto de entrada del programa.
- Dentro de la función `main ()`, se declara una variable `error` de tipo entero e inicializada en 0. Esta variable se utiliza para manejar errores o indicar el estado de finalización del programa.

#### 3. Llamada a la función `menu ()`:

- Se llama a la función `menu (error)` pasando la variable `error` como argumento.
- La llamada a la función `menu ()` presenta al usuario un menú de opciones y realice llamadas a otras funciones dada la selección del usuario.
- La variable `error` se pasa como argumento a `menu ()` para que pueda ser actualizada dentro de esa función, para el manejo de errores e indicar el estado de finalización.

#### 4. Retorno de un valor:

- La función `main ()` retorna el valor de la variable `error`, lo que puede ser útil para informar al sistema operativo si el programa se ejecutó correctamente (0) o si ocurrió algún error (código de error distinto de 0).

#### 4.4.3. *menuP01.cpp*

Este código es un programa que presenta un menú con varias opciones para el usuario. La función `menu ()` muestra el menú y solicita al usuario que elija una opción. Dependiendo de la opción seleccionada, el programa realiza diferentes acciones:

1. Si el usuario elige la opción 0, el programa se detiene y muestra un mensaje de despedida.
2. Si elige la opción 1, le permite ingresar una potencia.
3. Si elige la opción 2, genera una potencia aleatoria.
4. Si elige la opción 3, le permite cambiar los caracteres utilizados en el programa.
5. Si elige la opción 4, ejecuta una serie de acciones, incluida la generación de combinaciones y la medición del tiempo transcurrido.

A continuación se ve de manera separada cada función que compone al archivo *menuP01.cpp*:

Este fragmento de código define la función `menu()`, presenta un menú de opciones al usuario y ejecuta las acciones correspondientes según la opción seleccionada. A continuación el bloque de código de la función en cuestión:

```
1  void menu(int& error){
2      int flag = -1;
3      int potencia = 0;
4      int iteraciones = 1;
5      char simbolo0 = '#';
6      char simbolo1 = '.';
7
8      while(flag != 0){
9          std::cout<<"BIENVENIDO A LA PRACTICA 01."<<std::endl;
10         std::cout<<"OPCIONES A REALIZAR"<<std::endl;
11         std::cout<<"1- Ingresar potencia."<<std::endl;
12         std::cout<<"2- Dar potencia random."<<std::endl;
13         std::cout<<"3- Cambiar caracteres."<<std::endl;
14         std::cout<<"4- Ejecutar Programa."<<std::endl;
15         std::cout<<"0- Salir del Programa."<<std::endl;
16
17         flag = ingresarOpcion();
18
19         switch(flag){
20             case 0:
21                 std::cout<<"GRACIAS POR USAR EL PROGRAMA."<<std::endl;
22                 std::cout<<"ADIOS."<<std::endl;
23                 return;
24                 break;
25             case 1:
26                 ingresarPotencia(potencia);
27                 break;
28             case 2:
29                 randomPotencia(potencia);
30                 break;
31             case 3:
32                 ingresarCaracteres(simbolo0, simbolo1);
33                 break;
34             case 4:
35                 ejecucion(potencia, iteraciones, simbolo0, simbolo1);
36                 opcionGraficar(error);
37                 break;
38         }
39     }
40 }
41 }
```

Listing 3: Archivo: *menuP01.cpp* — Funcion: `menu()`

Aquí está, a grandes rasgos, lo que hace esta función:

#### 1. Inicialización de variables:

- Se inicializan varias variables locales, incluyendo `flag`, `potencia`, `iteraciones`, `simbolo0` y `simbolo1`, que se utilizan para controlar el flujo del programa y almacenar datos ingresados por el usuario.

#### 2. Bucle de menú:

- Se inicia un bucle `while` que se ejecutará hasta que `flag` sea igual a 0. Esto permite que el usuario seleccione varias opciones del menú en una sola sesión.

### 3. Menú de opciones:

- Se muestra un mensaje de bienvenida y se presentan las opciones disponibles al usuario. Las opciones incluyen:
  - Ingresar potencia.
  - Dar potencia aleatoria.
  - Cambiar caracteres.
  - Ejecutar programa.
  - Salir del programa.
- Se utiliza la función `ingresarOpcion ()` para obtener la opción seleccionada por el usuario.

### 4. Selección de acción:

- Se utiliza una estructura `switch` para ejecutar la acción correspondiente a la opción seleccionada por el usuario.
- Para cada caso, se llama a una función específica para realizar la acción correspondiente. Por ejemplo, si el usuario selecciona la opción 1, se llama a la función `ingresarPotencia ()` para permitirle al usuario ingresar una potencia.

### 5. Salida del programa:

- Si el usuario selecciona la opción 0, se muestra un mensaje de despedida y se sale del bucle y de la función.

Esta función `ingresarOpcion ()` proporciona una forma de interactuar con el usuario, permitiéndole seleccionar una opción deseada del menú mediante la entrada de un número entero en la consola. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación se anexa el bloque de código que alude a la función mencionada:

```
1  int ingresarOpcion(){
2      int opcion= 5;
3      std::cout << "Ingrese la opcion deseada: ";
4      std::cin >> opcion;
5
6      return opcion;
7  }
```

Listing 4: Archivo: *menuP01.cpp* — Funcion: `ingresarOpcion ()`

A continuación se desglosa, a groso modo, el funcionamiento de la función `ingresarOpcion ()`:

#### 1. Inicialización de la variable `opcion`:

- Se inicializa la variable `opcion` con un valor predeterminado de 5. Este valor predeterminado es útil para manejar situaciones en las que la entrada del usuario no sea válida o no se realice correctamente.

#### 2. Solicitud de entrada al usuario:

- Se muestra un mensaje en la consola solicitando al usuario que ingrese la opción deseada.

#### 3. Lectura de entrada:

- Se utiliza `std::cin` para leer el valor ingresado por el usuario desde la consola y almacenarlo en la variable `opcion`.

#### 4. Retorno del valor ingresado:

- Se devuelve el valor ingresado por el usuario como un entero utilizando la instrucción `return`.

Esta función `ejecucion ()` realiza el trabajo principal del programa, que incluye la generación de combinaciones y la medición del tiempo de ejecución. Además, gestiona la creación y escritura de archivos para almacenar los resultados de las combinaciones generadas. A continuación se anexa el bloque de código que alude a la función mencionada:

```
1 void ejecucion(int& potencia, int iteraciones, char& simbolo0, char& simbolo1){
2     int total = 0;
3     for (int i = 0; i <= potencia; i++)
4         total += pow(2,i);
5
6     //Crear y abrir un archivo
7     std::ofstream archivo("UniversoP01.txt");
8     std::ofstream archivo2("salidaPractica1.csv");
9
10    // Iniciar el cronometro
11    auto start = std::chrono::high_resolution_clock::now();
12
13    // Generar las combinaciones de longitud creciente del alfabeto
14    archivo << "\u03B5" << ",";
15    archivo2<< "0,0" << std::endl;
16    generarCombinaciones(archivo, archivo2, potencia, iteraciones, total,
17                          simbolo0, simbolo1, "", 0);
18
19    // Detener el cronometro y mostrar el tiempo transcurrido
20    medirTiempo(start);
21
22    //Mostrar iteraciones y tiempo cronometro
23    std::cout<<"Numero de iteraciones realizadas: "<<iteraciones<<std::endl;
24
25    // Cerrar el archivo
26    archivo.close();
27 }
```

Listing 5: Archivo: *menuP01.cpp* — Funcion: *ejecucion()*

Se indaga mas en el funcionamiento de la funcion en el siguiente enlistado:

**1. Cálculo del total de combinaciones:**

- Se calcula el total de combinaciones posibles utilizando la fórmula de la suma de potencias de 2, desde 0 hasta la potencia ingresada por el usuario.

**2. Creación y apertura de archivos:**

- Se crean y abren dos archivos de salida: *UniversoP01.txt* y *salidaPractica1.csv* utilizando la clase `std::ofstream`.

**3. Inicio del cronómetro:**

- Se utiliza la biblioteca `std::chrono` para iniciar el cronómetro y medir el tiempo de ejecución de ciertas partes del programa.

**4. Generación de combinaciones:**

- Se generan las combinaciones de longitud creciente del alfabeto, comenzando con la cadena vacía y aumentando gradualmente la longitud.

**5. Detención del cronómetro y medición del tiempo transcurrido:**

- Se detiene el cronómetro y se calcula el tiempo transcurrido desde el inicio de la ejecución hasta este punto.

**6. Impresión de información adicional:**

- Se imprime el número total de iteraciones realizadas durante la ejecución del programa.

**7. Cierre de archivos:**

- Se cierran los archivos de salida una vez que se han completado todas las operaciones de escritura.

Esta función `medirTiempo ()` calcula el tiempo transcurrido entre un punto de inicio y un punto de finalización utilizando la librería `<chrono>` de C++. Aquí está el bloque de código de la función:

```
1 void medirTiempo(std::chrono::high_resolution_clock::time_point start) {  
2     auto stop = std::chrono::high_resolution_clock::now();  
3     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop  
4         - start);  
5     auto milliseconds = duration.count() % 1000;  
6     auto seconds = (duration.count() / 1000) % 60;  
7     auto minutes = (duration.count() / (1000 * 60)) % 60;  
8     std::cout << "Tiempo transcurrido: " << minutes << " minutos, " << seconds  
9         << " segundos, " << milliseconds << " milisegundos" << std::endl;  
10 }
```

Listing 6: Archivo: *menuP01.cpp* — Función *medirTiempo()*

El proceso que sigue la función se describe de la siguiente manera:

1. Calcula la diferencia entre el tiempo de inicio (*start*) y el tiempo de finalización actual.
2. Convierte esa diferencia en milisegundos utilizando *duration\_cast*.
3. Calcula el número de minutos, segundos y milisegundos en función de la duración total.
4. Imprime el tiempo transcurrido en formato *minutos*, *segundos*, *milisegundos*.

#### 4.4.4. solicitudesP01.cpp

El archivo *solicitudesP01.cpp* contiene implementaciones de funciones relacionadas con la manipulación de datos o la interacción con el usuario en el contexto específico de la Práctica 01. Estas funciones incluyen la generación de combinaciones, la manipulación de archivos y la realización de cálculos específicos para resolver el problema abordado en la presente práctica. En esta parte del reporte se explorarán las funciones que la componen.

Esta función *ingresarPotencia ()* proporciona una forma de interactuar con el usuario, permitiéndole ingresar la potencia deseada para realizar ciertas operaciones en el programa. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación el bloque de código de la función en cuestión:

```
1 //Solicitudes  
2 #include "PRACTICA01.h"  
3  
4 void ingresarPotencia(int& potencia){  
5     std::cout << "Ingrese la potencia deseada: ";  
6     std::cin >> potencia;  
7 }
```

Listing 7: Archivo: *solicitudesP01.cpp* — Función *ingresarPotencia()*

A continuación se deja un enlistado de los puntos a destacar de la función:

1. **Solicitud de entrada al usuario:** Se muestra un mensaje en la consola solicitando al usuario que ingrese la potencia deseada.
2. **Lectura de entrada:** Se utiliza *std::cin* para leer el valor ingresado por el usuario desde la consola y almacenarlo en la variable *potencia*.
3. **Almacenamiento del valor ingresado:** El valor ingresado por el usuario se almacena en la variable *potencia*, que se pasa por referencia a la función para que pueda ser modificado dentro de la misma.

Esta función *randomPotencia ()* proporciona una forma de generar de manera aleatoria un valor de potencia y asignarlo a una variable, lo que puede ser útil en situaciones donde no se requiere una potencia específica y se desea una elección aleatoria.

A continuación se deja el bloque de código en cuestión:

```
1 void randomPotencia(int& potencia){  
2     srand(time(0));  
3     potencia = rand() % (28 - 0 + 1) + 0;  
4     std::cout<< "la potencia escogida de manera aleatoria fue:  
5         "<<potencia<<". "<<std::endl;
```

Listing 8: Archivo: *solicitudesP01.cpp* — Funcion `randomPotencia()`

Se detalla a mayor profundidad los puntos importantes de la funcion en el siguiente listado:

1. **Inicialización de la semilla del generador de números aleatorios:** Se utiliza la función `srand ()` junto con `time (0)` para inicializar la semilla del generador de números aleatorios. Esto garantiza que la secuencia de números aleatorios generada sea diferente cada vez que se ejecute el programa.
2. **Generación de la potencia aleatoria:** Se utiliza la función `rand ()` para generar un número aleatorio entre 0 y 28, inclusive. Se aplica el módulo `%` para limitar el rango de valores a un número máximo de 28 y un mínimo de 0. Se suma 0 al resultado final para asegurar que el valor mínimo sea 0.
3. **Asignación del valor de potencia:** El valor generado aleatoriamente se asigna a la variable `potencia`, que se pasa por referencia a la función para que pueda ser modificado dentro de la misma.
4. **Impresión del valor de potencia generado:** Se imprime en la consola el valor de potencia generado de manera aleatoria para informar al usuario sobre la potencia seleccionada.

La función `ingresarCaracteres ()` solicita al usuario que ingrese dos caracteres que representarán los símbolos cero y uno del alfabeto deseado. Luego, almacena estos caracteres en las variables `simbolo0` y `simbolo1`, que se pasan por referencia a la función. A continuación se deja el código:

```
1 void ingresarCaracteres(char& simbolo0, char& simbolo1){  
2     std::cout << "Ingrese el simbolo cero para el alfabeto deseado: ";  
3     std::cin >> simbolo0;  
4     std::cout << "Ingrese el simbolo uno para el alfabeto deseado: ";  
5     std::cin >> simbolo1;  
6     //Crear Archivo para detectar el 0 y uno del universo  
7     std::ofstream archivo("Universo.txt");  
8     archivo<<simbolo0<<','<<simbolo1;  
9     archivo.close();  
10 }
```

Listing 9: Archivo: *solicitudesP01.cpp* — Funcion `ingresarCaracteres()`

La funcion `opcion graficar ()` proporciona una manera de interactuar con el usuario, permitiéndole decidir si desea o no graficar el resultado del programa. La función `graficar ()` se llama si el usuario elige sí, de lo contrario, no se realiza ninguna acción adicional. Se deja el bloque de código que engloba a la función en cuestión:

```
1 void opcionGraficar(int& error){  
2     char opcion = 'Y';  
3     std::cout<<"Desea Graficar el resultado(Y[y]/N[n]): ";  
4     std::cin>>opcion;  
5     if (opcion=='Y' || opcion=='y')  
6         graficar(error);  
7 }
```

Listing 10: Archivo: *solicitudesP01.cpp* — Funcion `opcionGraficar()`

#### 4.4.5. *P01.cpp*

El archivo *P01.cpp* incluye únicamente a la función `generarCombinaciones ()` función encargada de la generación de combinaciones y el cálculo de los unos en cada combinación. Utiliza recursión para explorar todas las posibles combinaciones de longitud `len` y registra los datos en archivos de salida.

A continuación se deja el bloque de código de la función

```
1 #include "PRACTICA01.h"
2
3 // Funcion para generar combinaciones y calcular unos
4 void generarCombinaciones(std::ofstream& archivo1, std::ofstream& archivo2, int
   len, int& i, char simbolo0, char simbolo1, std::string prefix = "", int unos =
   0) {
5     if (len == 0) {
6         return;
7     }
8     for (char c : {simbolo0, simbolo1}) {
9         std::string newPrefix = prefix + c;
10        archivo1 << newPrefix << ", ";
11        archivo2 << i << ', ' << unos + (c == simbolo1) << std::endl;
12        i++;
13        generarCombinaciones(archivo1, archivo2, len - 1, i, simbolo0, simbolo1,
           newPrefix, unos + (c == simbolo1));
14    }
15 }
```

Listing 11: Archivo: *P01.cpp* — Funcion `generarCombinaciones()`

Se detalla a lujo de detalle, esta funcion, la funcion principal para la realizacion de esta practica:

#### 1. Parámetros de entrada:

- `std::ofstream& archivo1`: Referencia a un archivo de salida donde se escribirán las combinaciones generadas.
- `std::ofstream& archivo2`: Referencia a otro archivo de salida donde se escribirán los datos relacionados con las combinaciones.
- `int len`: Longitud de las combinaciones a generar.
- `int& i`: Variable que lleva la cuenta del número de combinaciones generadas.
- `char simbolo0`: Símbolo cero del alfabeto.
- `char simbolo1`: Símbolo uno del alfabeto.
- `std::string prefix = '' ''`: Prefijo de la combinación actual (se inicializa como una cadena vacía).
- `int unos = 0`: Número de unos en la combinación actual (se inicializa como cero).

#### 2. Generación de combinaciones:

- La función utiliza recursión para generar todas las combinaciones posibles de longitud `len`.
- En cada iteración, se agrega uno de los dos símbolos (`simbolo0` o `simbolo1`) al prefijo de la combinación actual.
- Se escriben las combinaciones y los datos relacionados en los archivos de salida proporcionados.

#### 3. Conteo de unos:

- Se lleva un conteo de los unos en la combinación actual para escribir los datos correspondientes en el segundo archivo de salida.

#### 4.4.6. graficarP01.cpp

El archivo *graficarP01.cpp* incluye unicamente a la función `graficar ()`, la cual ejecuta un script de MATLAB para generar un gráfico. Primero, define la ruta del script y luego la ejecuta utilizando la función `system ()`. Se verifica si la ejecución fue exitosa y se imprime un mensaje correspondiente. Después, se espera 15 segundos antes de iniciar la graficación en MATLAB utilizando la función `Sleep ()` de la biblioteca `windows.h`. Esta función facilita la integración de gráficos generados en MATLAB en el flujo de trabajo del programa de C++. A continuacion el bloque de codigo de la funcion en cuestión:



```
1 #include "PRACTICA01.h"
2 #include <windows.h> // Incluir la biblioteca windows.h
3
4 void graficar(int& error){
5     std::string ruta = "MATLAB -r run('graficoMatlab.m')";
6
7     error = system(ruta.c_str());
8
9     // Verificar si la ejecucion fue exitosa
10    if (error == 0) {
11        std::cout << "La ruta se ejecuto correctamente." << std::endl;
12    } else {
13        std::cerr << "Error al ejecutar la ruta." << std::endl;
14    }
15
16    // Esperar un tiempo adicional
17    std::cout << "Esperando 10 segundos antes de cerrar MATLAB..." <<
        std::endl;
18    Sleep(10000); // Espera 10000 milisegundos (equivalente a 10 segundos)
19 }
```

Listing 12: Archivo: *P01.cpp* — Funcion *graficar()*

#### 4.4.7. *graficoMatlab.m*

Este script de MATLAB configura y traza dos subgráficos: uno que muestra la cantidad de puntos por cadena y otro que muestra el logaritmo de la cantidad de puntos por cadena. Se deja el código del script en cuestión:

```
1 % Configurar el trazado
2 scatter_options = '.*';
3
4 % Inicializar variables para el trazado
5 figure;
6
7 % Nombre del archivo y longitud del bloque
8 nombre_archivo = 'salidaPractica1.csv';
9 block_size = 1e6; % Por ejemplo, lee el archivo en bloques de 1 millon de
    tuplas
10
11 % Abrir el archivo para lectura
12 fileID = fopen(nombre_archivo, 'r');
13
14 if fileID == -1
15     error('No se pudo abrir el archivo.');
```

```
16 end
17
18 % Crear subplots
19 subplot(2, 1, 1);
20 hold on;
21
22 % Leer y graficar los datos en bloques
23 while ~feof(fileID)
24     % Leer un bloque de datos del archivo
25     data = textscan(fileID, '%f%f', block_size, 'Delimiter', ',',
        'HeaderLines', 1);
26     if isempty(data{1})
27         break; % No hay mas datos para leer, salir del bucle
28     end
29
30     % Trazar los datos del bloque actual
```

```
31 scatter(data{1}, data{2}, scatter_options);
32 end
33
34 % Configurar etiquetas y titulo
35 xlabel('ID de cadena');
36 ylabel('Cantidad de puntos en la cadena');
37 title('Grafico de la cantidad de puntos por cadena');
38
39 % Crear subplots para el logaritmo
40 subplot(2, 1, 2);
41 hold on;
42
43 % Reiniciar la lectura del archivo
44 frewind(fileID);
45
46 % Leer y graficar los datos en bloques con logaritmo
47 while ~feof(fileID)
48     % Leer un bloque de datos del archivo
49     data = textscan(fileID, '%f%f', block_size, 'Delimiter', ',',
50                     'HeaderLines', 1);
51     if isempty(data{1})
52         break; % No hay mas datos para leer, salir del bucle
53     end
54
55     % Aplicar el logaritmo a la cantidad de puntos
56     data_log = log(data{2});
57
58     % Trazar los datos del bloque actual con logaritmo
59     scatter(data{1}, data_log, scatter_options);
60 end
61
62 % Configurar etiquetas y titulo para el subplot de logaritmo
63 xlabel('ID de cadena');
64 ylabel('Log(Cantidad de puntos en la cadena)');
65 title('Grafico del logaritmo de la cantidad de puntos por cadena');
66
67 % Cerrar el archivo
68 fclose(fileID);
69
70 % Mostrar el grafico
71 hold off;
72
73 pause(15);
74 close all;
75
76 pause(5);
77 exit();
```

Listing 13: Archivo: *graficoMatlab* — Script

Aquí está el resumen de lo que hace:

1. Configuración del trazado:

- Se establece el estilo de los puntos en el gráfico con la variable `scatter_options`.

2. Inicialización de variables y creación de la figura:

- Se inicializan las variables `nombre_archivo` y `block_size`.
- Se abre el archivo especificado en modo de lectura ('r').

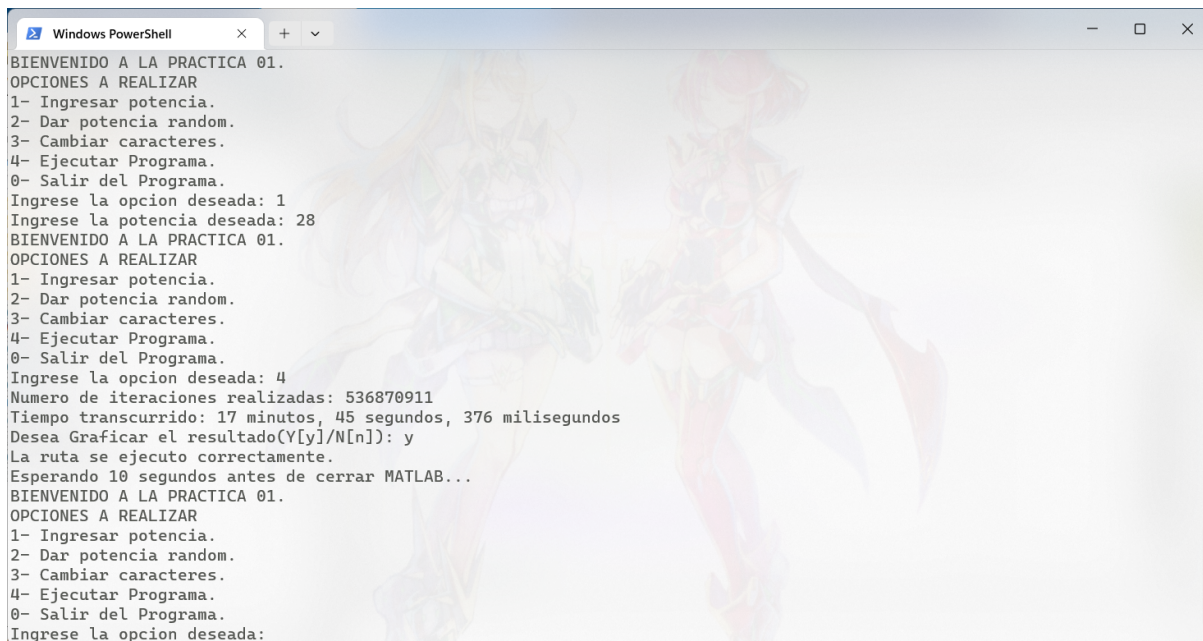
3. Creación de los subgráficos:

- Se crea un subplot con dos filas y una columna.

- En el primer subplot, se trazan los datos directamente desde el archivo.
  - En el segundo subplot, se trazan los datos con el logaritmo de la cantidad de puntos.
4. **Lectura y trazado de datos en bloques:**
- Se lee un bloque de datos del archivo y se trazan los puntos en el gráfico.
  - Se aplica el logaritmo a la cantidad de puntos en el segundo subplot.
5. **Configuración de etiquetas y títulos:**
- Se establecen etiquetas y títulos para ambos subgráficos.
6. **Cierre del archivo y visualización del gráfico:**
- Se cierra el archivo después de leer todos los datos.
  - Se muestra el gráfico.
7. **Pausa antes de cerrar y salir del script:**
- Se realiza una pausa de 15 segundos antes de cerrar el gráfico y salir del script.
  - Se cierra el gráfico y se finaliza el script después de una pausa adicional de 5 segundos.

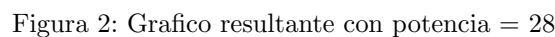
## 5 Resultados.

A continuación se dejan los resultados dados por el programa con una potencia igual a veinte y ocho:



```
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada: 1
Ingrese la potencia deseada: 28
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada: 4
Numero de iteraciones realizadas: 536870911
Tiempo transcurrido: 17 minutos, 45 segundos, 376 milisegundos
Desea Graficar el resultado(Y[y]/N[n]): y
La ruta se ejecuto correctamente.
Esperando 10 segundos antes de cerrar MATLAB...
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada:
```

Figura 1: Ejecucion de la Practica 01 en terminal.

[illegible]

Además, al explorar la visualización de datos en MATLAB, descubrí cómo transformar números y conceptos abstractos en imágenes claras y significativas. La representación gráfica de los datos generados

me proporcionó una nueva perspectiva sobre el comportamiento de los sistemas computacionales, permitiéndome identificar patrones, tendencias y anomalías que de otro modo podrían haber pasado desapercibidos. Esta experiencia me capacitó para comunicar de manera efectiva los resultados de mis análisis y tomar decisiones informadas basadas en la evidencia visual.

Más allá de las habilidades técnicas adquiridas, esta práctica me proporcionó una comprensión más profunda de los conceptos fundamentales que subyacen en el corazón de la computación. Desarrollé una apreciación más amplia de la importancia de la teoría de autómatas en la construcción y el análisis de sistemas computacionales, así como una comprensión más aguda de cómo la visualización de datos puede amplificar nuestra capacidad para comprender y tomar decisiones en entornos complejos.

Como conclusión final, puedo decir, esta práctica me equipó con las habilidades y el conocimiento necesarios para abordar desafíos más complejos en mi viaje en el mundo de la informática y la ciencia de datos. Al integrar teoría y práctica en un viaje de descubrimiento computacional, avancé hacia un mayor dominio de los principios fundamentales y las herramientas necesarias para enfrentar los desafíos del futuro con confianza y perspicacia.

# Referencias

- [1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introducción a la teoría de autómatas, lenguajes y computación*. Madrid: PEARSON EDUCACIÓN S.A., 2007. Formato: 195 x 250 mm. Páginas: 452.