Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería

Generador de Contraseñas

Equipo #8
Integrantes:
Delgado Cambray Adolfo
Gómez Monroy Jesús Eduardo
López Camal Diego Andrey
Urquiza Villa Marco Antonio

Domingo 26 de mayo de 2024

***Abstract*—Este proyecto presenta un programa en C y ASM, con el fin de generar y evaluar contraseñas, para ello se hace uso de funciones en lenguaje ensamblador para verificaciones especificas de caracteres. El menú del programa permite generar contraseñas aleatorias, evaluar la seguridad de una contraseña dada y salir del programa. La integración de C y ensamblador optimiza el rendimiento para tareas críticas, mientras que el Makefile tiene la finalidad de automatizar la compilación. En resumen, el programa ofrece una solución eficiente para mejorar la seguridad de las contraseñas.**

I. Introducción

El objetivo de este proyecto es desarrollar un programa en lenguaje C el cual permite generar contraseñas aleatorias y evaluar la seguridad de contraseñas ingresadas por el usuario, en función de criterios predefinidos. Además, el programa proporciona un menú interactivo que permite al usuario elegir entre generara una nueva contraseña o evaluar la seguridad de una contraseña existente.

El proceso para generar contraseñas implica la selección aleatoria de caracteres ASCII dentro de un rango especifico para garantizar la variedad y complejidad de la contraseña generada. Por otro lado, la evaluación de la seguridad de la contraseña implica verificar la presencia de caracteres mayúsculas, minúsculas, dígitos y caracteres especiales, así como la longitud de la contraseña. La función encargada de evaluar usa estos criterios para clasificar la seguridad de la contraseña en baja, media o alta.

En este proyecto se hace uso tanto de lenguaje C y lenguaje ensamblador (ASM). El código en C proporciona la estructura general del programa, incluyendo el menú interactivo, las funciones para generar y evaluar contraseñas, así como la integración de las funciones en ensamblador. Por otro lado, el código en ensamblador implementa funciones específicas de bajo nivel para realizar verificaciones detalladas sobre contraseñas, como la búsqueda de caracteres específicos y la determinación de la longitud de la cadena.

Además, este proyecto hace uso de un Makefile para automatizar el proceso de compilación del programa, lo que facilita la gestión del código fuente y la generación de ejecutables tanto en modo debug como en modo reléase. Esto proporción flexibilidad y eficiencia en el desarrollo y la distribución del software. En resumen, el proyecto combina el poder del lenguaje C y del lenguaje ensamblador para crear un programa robusto y eficiente que cumple con los requisitos de generación y evaluación de contraseñas de manera segura y confiable.

II. Descripción del Funcionamiento

A. Generador de Contraseñas

La función `generador_contrasena` se encarga de generar una contraseña aleatoria y mostrarla al usuario. Aquí está su funcionamiento detallado:

1) **Inicialización del Generador de Números Aleatorios**
   - Utiliza la función `srand(time(NULL))` para inicializar el generador de números aleatorios con una semilla basada en el tiempo actual. Esto garantiza que cada ejecución del programa produzca una contraseña única.
2) **Generación de Longitud Aleatoria**
   - Elige una longitud aleatoria para la contraseña en el rango de 15 a 23 caracteres.
3) **Generación de Caracteres Aleatorios**
   - Utiliza un bucle para generar caracteres aleatorios y almacenarlos en un buffer.
   - Para cada posición en el buffer, se genera un valor aleatorio entre 32 y 126, que corresponde al rango de caracteres imprimibles en la tabla ASCII.
   - Estos valores se convierten en caracteres ASCII y se añaden al buffer.
4) **Finalización del Buffer**
   - Se agrega un carácter nulo al final del buffer para marcar el final de la cadena.
5) **Impresión de la Contraseña Generada**
   - La contraseña generada se imprime en la pantalla para que el usuario la vea y la utilice.

1) Funciones Utilizadas en la Generación de Contraseñas: Durante la generación de contraseñas, la función `generador_contrasena` hace uso de las siguientes funciones:

- `srand` y `rand`
  - `srand` se utiliza para inicializar el generador de números aleatorios con una semilla basada en el tiempo.
  - `rand` se utiliza para generar valores aleatorios que determinan la longitud y los caracteres de la contraseña.
- `printf`
  - Utilizada para imprimir la contraseña generada en la pantalla.
- `time`
  - Utilizada para obtener el tiempo actual como semilla para `srand`, asegurando así que las contraseñas generadas sean diferentes en cada ejecución.
- **Operaciones Matemáticas**
  - Se realizan operaciones matemáticas para generar la longitud y los valores aleatorios que representan los caracteres de la contraseña.

B. Verificador de Contraseñas

La función `evaluacion_contrasena` se encarga de evaluar la seguridad de una contraseña ingresada por el usuario. Aquí está su funcionamiento detallado:

1) **Solicitud de Contraseña al Usuario**
   - La función solicita al usuario que ingrese una contraseña para evaluar su seguridad.
2) **Evaluación de Seguridad**
   - Utiliza varias funciones para verificar diferentes aspectos de la contraseña ingresada, como la presencia de mayúsculas, minúsculas, dígitos y caracteres especiales, así como la longitud de la contraseña.

- Basándose en los resultados de estas verificaciones, la función determina el nivel de seguridad de la contraseña.

3) **Clasificación del Nivel de Seguridad**
   - Clasifica el nivel de seguridad de la contraseña en tres categorías: baja, media o alta, según los criterios definidos.
   - Si la contraseña cumple con ciertos criterios (como tener al menos una mayúscula, una minúscula, un dígito, un carácter especial y una longitud mínima), se considera de alta seguridad.
   - Si no cumple todos los criterios, pero cumple algunos, se considera de seguridad media.
   - Si no cumple con ninguno o solo unos pocos criterios, se considera de baja seguridad.

4) **Impresión del Nivel de Seguridad**
   - Finalmente, la función imprime en pantalla el nivel de seguridad determinado para la contraseña ingresada.

1) Funciones Utilizadas en la Verificación de Contraseñas: Durante la verificación de contraseñas, la función `evaluacion_contrasena` hace uso de las siguientes funciones:

- **Funciones Externas**
  - La función hace uso de varias funciones externas definidas en archivos ASM para verificar aspectos específicos de la contraseña, como la presencia de mayúsculas, minúsculas, dígitos y caracteres especiales, así como la longitud de la contraseña.
- `printf` y `fgets`
  - Utilizada para imprimir mensajes en la pantalla y para obtener la contraseña ingresada por el usuario desde la entrada estándar.
- **Operaciones de Comparación**
  - Se realizan operaciones de comparación para evaluar si la contraseña cumple con ciertos criterios de seguridad.

C. Requisitos Previos

ARCHIVO: main.c

- `main.c`
  - Este archivo contiene la función principal `main`, que actúa como punto de entrada del programa.
  - Implementa un menú interactivo que permite al usuario elegir entre generar una contraseña aleatoria o verificar la seguridad de una contraseña ingresada.
  - Llama a las funciones correspondientes dependiendo de la opción seleccionada por el usuario.

Funciones en C

- `main(void)`
  - Función principal que gestiona el flujo del programa.
  - Presenta un menú al usuario.
  - Según la opción seleccionada por el usuario, llama a la función correspondiente para generar una contraseña o verificar la seguridad de una contraseña.
- `generador_contrasena(void)`
  - Genera una contraseña aleatoria y la muestra al usuario.
  - Hace uso de la función `rand()` para generar números aleatorios.
  - Utiliza `printf()` para mostrar la contraseña generada en la pantalla.
- `evaluacion_contrasena()`
  - Evalúa la seguridad de una contraseña ingresada por el usuario.
  - Llama a funciones externas definidas en ASM para verificar diferentes aspectos de la contraseña, como la presencia de mayúsculas, minúsculas, dígitos y caracteres especiales, así como su longitud.
  - Clasifica el nivel de seguridad de la contraseña y lo muestra al usuario.

Funciones en Ensamblador (ASM)

- `buscar_mayuscula()`
  - Verifica si una cadena contiene al menos una letra mayúscula.
  - Utiliza comparaciones con los valores ASCII de las letras mayúsculas.
- `buscar_minuscula()`
  - Verifica si una cadena contiene al menos una letra minúscula.
  - Utiliza comparaciones con los valores ASCII de las letras minúsculas.
- `buscar_digito()`
  - Verifica si una cadena contiene al menos un dígito numérico.
  - Utiliza comparaciones con los valores ASCII de los dígitos numéricos.
- `buscar_caracter()`
  - Verifica si una cadena contiene caracteres especiales.
  - Utiliza comparaciones con los valores ASCII para determinar si un carácter está dentro de un rango específico.
- `longitud()`
  - Calcula la longitud de una cadena.
  - Itera sobre la cadena hasta encontrar el carácter nulo (`\0`) que marca el final de la cadena.

Makefile

- El Makefile define las reglas para compilar el programa.
- Especifica cómo compilar archivos fuente en C y ensamblador, así como cómo enlazarlos para crear el ejecutable final.

Objetivos del Makefile

- `debug`
  - Compila el programa con información de depuración.

  - Genera un ejecutable llamado `programa`.
- `release`
  - Compila el programa sin información de depuración, generando un ejecutable optimizado.
  - Genera un ejecutable llamado `programa`.
- `clean`
  - Elimina los archivos generados durante el proceso de compilación, incluyendo ejecutables y archivos objeto.

D. Compilación y Ejecución

Compilar en Modo Debug o Release

- **Modo Debug**
  - Este modo incluye información de depuración que es útil para el desarrollo y la resolución de problemas.
  - `make debug`
- **Modo Release**
  - Este modo produce un ejecutable optimizado para su uso en producción.
  - `make release`

Limpieza de Archivos Generados

- **Clean**
  - Después de compilar, si deseas limpiar los archivos generados (ejecutables y archivos objeto), puedes usar el siguiente comando:
  - `make clean`

Ejecución del Programa

- **Ejecutar el Modo Debug**
  - `./programa`
- **Ejecutar el Modo Release**
  - `./programa`

III. Organización del Código

A. ARCHIVO: main.c

- **Librerías Utilizadas**
  - Para este archivo C solo se hace uso de las siguientes librerías:
    * `<stdio.h>`
    * `<stdlib.h>`
    * `<time.h>`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

- **Funciones Externas**
  - Se declaran funciones que se encuentran en los archivos .asm. Estas funciones son los parámetros que nos ayudarán con la función `evaluacion_contrasena`.

```c
extern int buscar_mayuscula(char *cadena);
extern int buscar_minuscula(char *cadena);
extern int buscar_digito(char *cadena);
extern int buscar_caracter(char *cadena);
extern int longitud(char *cadena);
void generador_contrasena(void);
void evaluacion_contrasena(void);
```

- **Función main**
  - Esta función principal presenta un menú al usuario para elegir entre generar una contraseña, verificar la seguridad de una contraseña o salir del programa.
  - El usuario ingresa una opción y se ejecuta la función correspondiente según la elección.

```c
int main(void){
    int opcion;

    do{
        printf("****** Menu ******\n");
        printf("[1] Generador de contrasena.\n");
        printf("[2] Verificador de contrasena.\n");
        printf("[3] Salir del programa.\n");
        printf("Ingresa tu eleccion:");
        scanf("%d", &opcion);
        getchar(); // Limpiar el buffer de entrada

        switch(opcion) {
            case 1:
                generador_contrasena();
                break;
            case 2:
                evaluacion_contrasena();
                break;
            case 3:
                printf("Cerrando programa\n");
                break;
            default:
                printf("Valor invalido\n");
        }
    } while(opcion != 3);

    return 0;
}
```

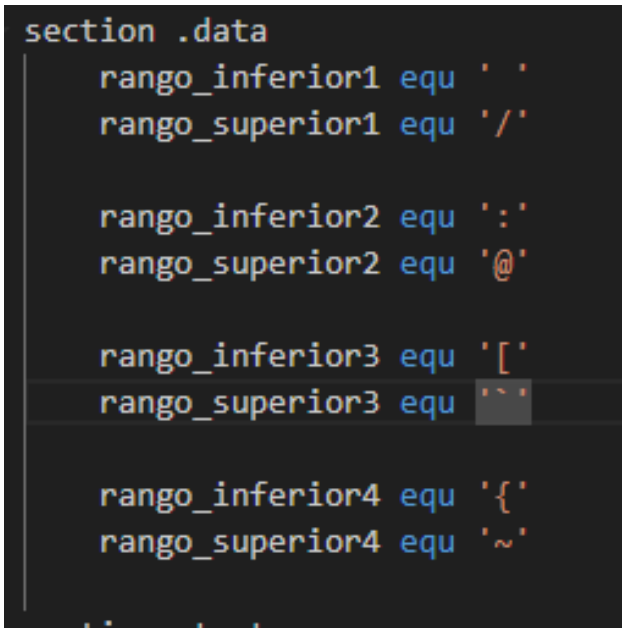
- **Función generar_contrasena**
  - Esta función genera una contraseña aleatoria con una longitud entre 15 y 23 caracteres.
  - Cada carácter es un símbolo ASCII visible.

```c
void generador_contrasena(void){
    srand(time(NULL)); // Inicializa la semilla del generador de números aleatorios
    int numero = rand() % 9; // Genera un número aleatorio entre 0 y 8
    int valor;
    numero += 15; // Ajusta el tamaño de la contraseña a un valor entre 15 y 23
    char buffer[numero];
    for(int i = 0; i < numero; i++){
        valor = 0;
        valor = rand() % 95; // Genera un número aleatorio entre 0 y 94
        buffer[i] = valor + 32; // Convierte el número a un carácter ASCII visible
    }
    buffer[numero - 1] = '\0'; // Termina la cadena con un carácter nulo
    printf("La contrasena generada es: %s \n", buffer);
}
```

- **Función verificar_contrasena**
  - Esta función evalúa la seguridad de una contraseña ingresada por el usuario.
  - Hace uso de funciones externas para verificar la presencia de mayúsculas, minúsculas, dígitos, caracteres especiales y la longitud de la contraseña.
  - Asigna un nivel de seguridad basado en estas verificaciones.

B. ARCHIVO: caracteres.asm

- **Sección .data**
  - En esta sección, se definen los rangos de caracteres especiales. Estos rangos son utilizados para identificar si un carácter de la cadena pertenece a alguno de estos intervalos.



- **Sección .text**
  - En este bloque comienza la sección de código. Se define la función `buscar_caracter`, como global para que pueda ser llamada desde otros archivos. Se inicializa el registro `rcx` a cero, este se usará como contador para recorrer la cadena.

```
section .text
    global buscar_caracter

buscar_caracter:
    xor rcx, rcx      ; Inicializa el contador a cero
```

- **Bucle Principal**
  - Se compara el carácter actual con el valor 10, para verificar el final de la cadena. Si se encuentra con un salto de línea, se salta a la etiqueta `fin`.

```
principal:
    cmp byte [rdi + rcx], 10
    je fin

    mov bl, byte [rdi + rcx]

    cmp bl, rango_inferior1
    jg filtro1
```

- **Filtros, Etiqueta True y Fin**
  - Los siguientes bloques comparan el carácter actual con los rangos especiales definidos. Si el carácter se encuentra dentro de alguno de los rangos, se salta a la etiqueta `true`. Si no, se incrementa el contador `rcx` para pasar al siguiente carácter y se vuelve al inicio del bucle principal.
  - Si se encuentra un carácter especial, se establece el valor de retorno en `rax` a 1 y se retorna de la función.
  - Si se llega al final de la cadena, se establece el valor de retorno en `rax` a 0 y se retorna de la función.

```
filtro1:
    cmp bl, rango_superior1
    jl true
    cmp bl, rango_inferior2
    jge filtro2
    inc rcx
    jmp principal

filtro2:
    cmp bl, rango_superior2
    jle true
    cmp bl, rango_inferior3
    jge filtro3
    inc rcx
    jmp principal

filtro3:
    cmp bl, rango_superior3
    jle true
    cmp bl, rango_inferior4
    jge filtro4
    inc rcx
    jmp principal

filtro4:
    cmp bl, rango_superior4
    jle true
    inc rcx
    jmp principal

true:
    mov rax, 1
    ret

fin:
    xor rax, rax
    ret
```

C. ARCHIVO: longitud.asm

- **Sección .text**
  - En esta sección, se define la función `longitud` como global para que pueda ser llamada desde otros archivos. La función comienza inicializando el registro `rcx` a cero, que se utilizará como contador para recorrer la cadena.
- **Bucle loop**
  - Este es el bucle principal que recorre cada carácter de la cadena. La dirección base de la cadena se pasa en el registro `rdi`.
  - Comparación con salto de línea: Se compara el carácter actual con `0x0A` (salto de línea). Si se encuentra un salto de línea, se salta a la etiqueta `comparar`.
  - Comparación con el carácter nulo: Se compara el carácter actual con `0` (carácter nulo). Si se encuentra el final de la cadena, también se salta a la etiqueta `comparar`.
  - Incremento del contador: Si no se encuentra ni un salto de línea ni el final de la cadena, se incrementa el contador `rcx` para pasar al siguiente carácter.
  - Repetir el bucle: Se vuelve al inicio del bucle para procesar el siguiente carácter.
- **comparar**
  - Una vez que se ha recorrido la cadena o se ha encontrado un salto de línea, se compara el valor del contador `rcx` con 12.
    - Si la longitud es mayor o igual a 12: Si `rcx` es mayor o igual a 12, se salta a la etiqueta `true`.
    - Si la longitud es menor a 12: Si `rcx` es menor a 12, se limpia el registro `rax` (poniéndolo a 0) y se retorna de la función. Esto indica que la cadena no cumple con la longitud mínima requerida.
- **Etiqueta true**
  - Si se ha determinado que la longitud de la cadena es mayor o igual a 12 caracteres, se mueve el valor 1 al registro `rax` y se retorna de la función. Esto indica que la cadena cumple con la longitud mínima requerida.

```
section .text
    global longitud

longitud:
    xor rcx, rcx

loop:
    cmp byte [rdi + rcx], 0x0A
    je comparar
    cmp byte [rdi + rcx], 0
    je comparar
    inc rcx
    jmp loop

comparar:
    cmp rcx,12
    jge true
    xor rax, rax
    ret

true:
    mov rax, 1
```

D. ARCHIVO: mayuscula.asm

- **Sección .data**
  - En esta sección se definen los límites para el rango de caracteres de letras mayúsculas en ASCII. `rango_inferior` está establecido como 'A' y `rango_superior` como 'Z'.
- **Bucle búsqueda**
  - En este bucle, se compara el carácter actual de la cadena con 0, lo que indica el final de la cadena. Si se encuentra el final de la cadena, el programa salta a la etiqueta `fin`. Si no, se mueve el carácter actual a `bl` para la comparación.
- **Verificación de mayúscula**
  - Aquí se compara el carácter actual (`bl`) con los límites definidos para las letras mayúsculas. Si el carácter está fuera de este rango, se salta a la etiqueta `no_mayuscula`.
- **Etiqueta NO Mayúscula y Fin**
  - Si el carácter actual no es una letra mayúscula, se incrementa el contador `rcx` para pasar al siguiente carácter y se vuelve al bucle de búsqueda. Si se llega al final de la cadena sin encontrar ninguna mayúscula, se establece el valor de retorno (`rax`) en 0 y se sale de la función.

```
section .data
    rango_inferior equ 'A'
    rango_superior equ 'Z'

section .text
    global buscar_mayuscula

buscar_mayuscula:
    xor rcx, rcx      ; Inicializa el contador a cero, correcto para usar con el índice

busqueda:
    cmp byte [rdi + rcx], 0
    je fin            ; Salta a fin si el carácter actual es NULL (fin de cadena)

    mov bl, byte [rdi + rcx]  ; Mueve el carácter actual a BL para la comparación

    ; Verifica si el carácter es un dígito
    cmp bl, rango_inferior
    jl no_mayuscula
    cmp bl, rango_superior
    jg no_mayuscula

    ; Si es un dígito, termina la función y devuelve 1
    mov rax, 1
    ret

no_mayuscula:
    inc rcx           ; Avanza al siguiente carácter
    jmp busqueda

fin:
    ; Si no se encontraron dígitos, devuelve 0
    xor rax, rax
    ret
```

E. ARCHIVO: minuscula.asm

- **Sección .data**
  - En esta sección se definen los límites para el rango de caracteres de letras minúsculas en ASCII. `rango_inferior` está establecido como 'a' y `rango_superior` como 'z'.
- **Sección .text**
  - Esta parte inicializa la función `buscar_minuscula` y establece el contador `rcx` a cero, el cual se utilizará como un índice para recorrer la cadena.
- **Bucle búsqueda**
  - En este bucle, se compara el carácter actual de la cadena con 10, que es el valor ASCII para un salto de línea. Si se encuentra un salto de línea, el programa salta a la etiqueta `fin`. Si no, se mueve el carácter actual a `bl` para la comparación.
- **Verificación de minúscula**
  - Aquí se compara el carácter actual (`bl`) con los límites definidos para las letras minúsculas. Si el carácter está fuera de este rango, se salta a la etiqueta `no_minuscula`.
- **Etiqueta NO minúscula y Fin**
  - Si el carácter actual no es una letra minúscula, se incrementa el contador `rcx` para pasar al siguiente carácter y se vuelve al bucle de búsqueda. Si se llega al final de la cadena sin encontrar ninguna minúscula, se establece el valor de retorno (`rax`) en 0 y se sale de la función.

```
section .data
    rango_inferior equ 'a'
    rango_superior equ 'z'

section .text
    global buscar_minuscula

buscar_minuscula:
    xor rcx, rcx      ; Inicializa el contador a cero, correcto para usar con el índice

busqueda:
    cmp byte [rdi + rcx], 10
    je fin            ; Salta a fin si el carácter actual es NULL (fin de cadena)

    mov bl, byte [rdi + rcx]  ; Mueve el carácter actual a BL para la comparación

    ; Verifica si el carácter es un dígito
    cmp bl, rango_inferior
    jl no_minuscula
    cmp bl, rango_superior
    jg no_minuscula

    ; Si es un dígito, termina la función y devuelve 1
    mov rax, 1
    ret

no_minuscula:
    inc rcx           ; Avanza al siguiente carácter
    jmp busqueda

fin:
    ; Si no se encontraron dígitos, devuelve 0
    xor rax, rax
    ret
```

F. ARCHIVO: numero.asm

- **Sección .data**
  - En esta sección se definen los límites para el rango de caracteres de dígitos en ASCII. `rango_inferior` está establecido como '0' y `rango_superior` como '9'.
- **Sección .text**
  - Esta parte inicializa la función `buscar_digito` y establece el contador `rcx` a cero, el cual se utilizará como un índice para recorrer la cadena.
- **Bucle de búsqueda**
  - En este bucle, se compara el carácter actual de la cadena con 0, lo que indica el final de la cadena. Si se encuentra el final de la cadena, el programa salta a la etiqueta `fin`. Si no, se mueve el carácter actual a `bl` para la comparación.
- **Verificación de dígito**
  - Aquí se compara el carácter actual (`bl`) con los límites definidos para los dígitos. Si el carácter está fuera de este rango, se salta a la etiqueta `no_digito`.
- **Etiqueta NO dígito y Fin**
  - Si el carácter actual no es un dígito, se incrementa el contador `rcx` para pasar al siguiente carácter y se vuelve al bucle de búsqueda. Si se llega al final de la cadena sin encontrar ningún dígito, se establece el valor de retorno (`rax`) en 0 y se sale de la función.

```
section .data
    rango_inferior equ '0'
    rango_superior equ '9'

section .text
    global buscar_digito

buscar_digito:
    xor rcx, rcx      ; Inicializa el contador a cero, correcto para usar con el índice

busqueda:
    cmp byte [rdi + rcx], 0
    je fin            ; Salta a fin si el carácter actual es NULL (fin de cadena)

    mov bl, byte [rdi + rcx]  ; Mueve el carácter actual a BL para la comparación

    ; Verifica si el carácter es un dígito
    cmp bl, rango_inferior
    jl no_digito
    cmp bl, rango_superior
    jg no_digito

    ; Si es un digito, termina la función y devuelve 1
    mov rax, 1
    ret

no_digito:
    inc rcx           ; Avanza al siguiente carácter
    jmp busqueda

fin:
    ; Si no se encontraron dígitos, devuelve 0
    xor rax, rax
    ret
```

G. ARCHIVO: MAKEFILE

1) Debug: Este código permite automatizar la compilación del programa en modo de depuración. Se hace uso de `gcc` para compilar los archivos objeto generados por la compilación de los archivos de código fuente y ensamblador. La opción `-g` habilita la generación de información de depuración para el depurador. El principal objetivo de esta regla es generar un ejecutable con información de depuración para facilitar la depuración del código.

2) Release: Este código permite automatizar la compilación del programa en modo de liberación. Se hace uso de `gcc` para compilar los archivos objeto, pero sin la opción `-g`, lo que significa que se genera un ejecutable optimizado sin información de depuración. El principal objetivo de esta regla es generar un ejecutable optimizado y listo para su distribución o uso en producción.

3) `main.o`, `caracter.o`, `longitud.o`, `mayuscula.o`, `minuscula.o`, `numero.o`: Este código permite automatizar la compilación de cada uno de los archivos de código fuente y los submódulos escritos en ensamblador en archivos objetos. Utilizan `gcc` para compilar los archivos `.c` y `nasm` para ensamblar los archivos `.asm`. Las opciones `-c` y `-f elf64` son comunes en ambas, indicando la compilación de un archivo objeto y que el formato de salida es ELF de 64 bits.

4) Clean: Este código permite automatizar la limpieza, que permite borrar los archivos ejecutables y los archivos objeto generados durante la compilación. Utiliza el comando `rm` para eliminar los archivos específicos. Los archivos que se

eliminan son `programa`, `programa_debug` (los cuales son ejecutables) y todos los archivos objetos.

IV. Detalles Técnicos

A. *Interacción entre C y ASM*

El lenguaje C y el lenguaje ensamblador (asm) son lenguajes que, por separado, ofrecen una gran cantidad de herramientas para desarrollar programas incluso más complejos que el presentado aquí. Sin embargo, el uso combinado de ambos en un proyecto ofrece varias ventajas, ya que se obtiene lo mejor de cada uno.

Por un lado, el lenguaje ensamblador nos da control de bajo nivel, un rendimiento máximo y un tamaño de código reducido. Por otro lado, C nos ofrece gran portabilidad, una amplia base de bibliotecas, compatibilidad con otros lenguajes y, lo más importante, una mayor facilidad de uso.

Para este proyecto, el equipo decidió implementar las funciones fundamentales en ensamblador, mientras que la parte de la interfaz interactiva, que sirve como puente entre el usuario y el programa, se realizó en C, al igual que dos funciones más, de las cuales una hace uso de las funciones fundamentales mencionadas.

- **Generación de Contraseñas**
  - La función de generación de contraseñas se implementa en C, aprovechando la facilidad de uso y la capacidad de gestionar operaciones de alto nivel, como la generación de números aleatorios y la manipulación de cadenas.
- **Verificación de Contraseñas (ASM)**
  - Las funciones de verificación de contraseñas (como `buscar_mayuscula`, `buscar_minuscula`, `buscar_digito`, `buscar_caracter` y `longitud`) se implementan en ensamblador. Esto permite una verificación rápida y eficiente de los criterios de seguridad, utilizando comparaciones directas con valores ASCII y manipulación directa de memoria.

V. Ejemplos y Pruebas

A continuación, se muestra un ejemplo sobre el uso de este programa. En este se hace uso del comando `make release` para crear los objetos que serán usados para que el programa se ejecute de forma correcta (figura 1).

```
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$ make release
gcc -c main.c
nasm -f elf64 -o caracter.o caracter.asm
nasm -f elf64 -o longitud.o longitud.asm
nasm -f elf64 -o mayuscula.o mayuscula.asm
nasm -f elf64 -o minuscula.o minuscula.asm
nasm -f elf64 -o numero.o numero.asm
gcc -o programa main.o caracter.o longitud.o mayuscula.o minuscula.o numero.o
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$
```

Se muestra el menú que se creó en el archivo `main.c`, en el cual se puede generar una contraseña, evaluar la contraseña y salir del programa (figura 2).

```
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$ ./programa
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:
```

Después de generar la contraseña, se vuelve al menú automáticamente, permitiendo evaluar la contraseña generada o evaluar alguna ya existente (figura 3).

```
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$ ./programa
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:1
La contrasena generada es: ldvWWa,iP;7I*{j?Q5K4W,
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:
```

Se ingresa una contraseña desde la consola y, al evaluarla, se devuelve un mensaje indicando el nivel de seguridad de la contraseña, en este caso, es baja (figura 4).

```
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$ ./programa
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:1
La contrasena generada es: ldvWWa,iP;7I*{j?Q5K4W,
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:2
Ingresa la contrasena a evaluar:jnsdjhsj76
Nivel de seguridad Baja
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:3
```

A continuación, se ingresa una contraseña con un nivel de seguridad alto, mostrando cómo el programa se está ejecutando de forma exitosa (figura 5).

```
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$ ./programa
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:2
Ingresa la contrasena a evaluar:#4\ujJKJ0-k8()
Nivel de seguridad Alta
****** Menu ******
[1] Generador de contrasena.
[2] Verificador de contrasena.
[3] Salir del programa.
Ingresa tu eleccion:3
Cerrando programa
dark_chuponsito@DarkChuponsitoPC:~/Escritorio/Org_final$
```

VI. Conclusión

El proyecto de generador de contraseñas integra el uso de lenguajes C y ensamblador para crear un programa robusto y eficiente, capaz de generar y evaluar contraseñas de manera segura y confiable.

El generador de contraseñas en C se encarga de inicializar el generador de números aleatorios, seleccionar una longitud aleatoria para la contraseña y generar los caracteres aleatorios correspondientes. Este enfoque garantiza la unicidad y la complejidad de las contraseñas generadas al utilizar una semilla basada en el tiempo actual.

Por otro lado, las funciones de verificación en ensamblador realizan comprobaciones detalladas de seguridad, como la búsqueda de caracteres específicos (mayúsculas, minúsculas, dígitos y caracteres especiales) y la determinación de la longitud de la contraseña. Estas verificaciones permiten clasificar la seguridad de las contraseñas en baja, media o alta, asegurando que las contraseñas cumplan con los criterios de seguridad establecidos.

El uso de un `Makefile` automatiza el proceso de compilación, facilitando la gestión del código fuente y la generación de ejecutables en diferentes modos. Esto aporta flexibilidad y eficiencia en el desarrollo y distribución del software.

Este proyecto demuestra cómo la combinación de C y ensamblador puede aprovechar lo mejor de ambos mundos: la facilidad de gestión y la capacidad de alto nivel del lenguaje C junto con la eficiencia y precisión del lenguaje ensamblador, resultando en una solución eficaz para la generación y evaluación de contraseñas.