# Seminario de lenguajes
## Opción C

### Práctica 3 - 2022

1. **Repaso práctica anterior:** Implementar de forma sencilla el programa `cat(1)` sin argumentos leyendo de la entrada estándar. Ver como se podría usar para copiar archivos. Comparar con el comando de *en GNU/Linux*.

2. **Repaso práctica anterior:** Implementar de forma sencilla el programa `head(1)` sin argumentos leyendo de la entrada estándar. Ver como se podría incorporar que la cantidad de líneas a leer sea "parametrizada". Implementar una versión del programa que funcione contando bytes, como `head -c`. Comparar con el comando de *en GNU/Linux*.

3. Escriba un programa que implemente la función:

```c
void convertir(int, char[ ] s, int b);
```

   (a) La función debe convertir el entero **i**, en el string **s** en su representación en base **b**.

   (b) La base debe ser mayor que 1 y menor que 37 dado que iría de base-2 hasta base-36, que usaría los dígitos: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ

   (c) Implemente la función considerando que **i** puede ser negativo, y se empleará el mismo símbolo (**-**) para su representación en base-**b**.

4. ¿Qué imprime el siguiente fragmento de código?

```c
char *arr1 = "Hola mundo";
char arr2[20];

printf("%d ", sizeof(arr1));
printf("%d ", sizeof(arr2));
```

5. Describa las diferencias esenciales entre un arreglo multidimensional y un arreglo de punteros. Por ejemplo:

```c
char arr1[ ][15] = {"uno", "dos", "tres"};
char arr2[5][15];
char *arr3[ ] = {"uno", "dos", "tres"}
char *arr4[4];
```

   (a) Para cada caso mencionado muestre cómo se representarían físicamente en memoria. Puede emplear un gráfico de la representación.

   (b) Indique el tamaño de las variables **arr1**, **arr2**, **arr3**, **arr4**.

6. Realizar un programa que dado un arreglo de enteros lo muestre ordenado de menor a mayor. *No es necesario leer el arreglo desde la entrada estándar, use arreglos de prueba.*

7. Realizar una función que reciba dos arreglos de enteros y almacene en un tercer arreglo la suma de sus elementos.

8. Realizar una función que dado un string, lo envíe a la salida estándar en orden inverso. Haga dos implementaciones de la función: iterativa y recursiva. Ver como cambiarla para que en lugar de enviarlo a la salida lo devuelva como retorno o argumento de salida.

9. Realizar una función que dado un string, devuelva la cantidad de palabras que contiene.

10. Analice lo que hacen las funciones de la librería *string.h*:

```
char *strcat(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *dest, const char *src);
size_t strlen(const char *s);
char *strstr(const char *haystack, const char *needle);
```

11. Implemente las funciones del punto anterior. En cada caso puede usar otra de las funciones a implementar. Los prototipos deben ser iguales, pero los nombres de las funciones a implementar deber comenzar con **my_**, por ejemplo **my_strlen**.

12. Implemente las siguientes funciones:

```
/*
 * Retorna si la palabra recibida es capicua.
 */
int es_palindromo(const char *word)

/*
 * Encripta el string src en la variable dest y, retorna el mismo
 * valor que dest. Para encriptar debe aplicar el complemento a 1 de
 * cada char.
 */
char* encripto(char *dest, const char *src)
```

13. Escriba un programa que cuente la cantidad de palabras capicúa. Debe emplear la función **es_palindromo** del ejercicio anterior. *Para probar el ejercicio, inicialice una variable con una cadena de caracteres que contenga palabras.*

14. Escriba un programa que analice un texto indicando el número de aquellas líneas que tienen palabras capicúas e indique la cantidad de palabras capicúas en la línea. Debe emplear la función **es_palindromo** del ejercicio 10. *Para probar el ejercicio, inicialice una variable con una cadena de caracteres que contenga palabras. El fin de línea es \n.*

15. Escriba un programa que reciba parámetros desde su ejecución e imprima todos los parámetros recibidos en pantalla.

16. Depure el siguiente código para descubrir por qué falla:

```
#include <stdio.h>
#include <ctype.h>

void capitalizar(char *cadena){
      printf("Forma original: '%s'\n", cadena);
      // Pasamos el primer caracter a mayúsculas
      cadena[0] = toupper(cadena[0]);
      printf("Capitalizado: '%s'\n", cadena);
      puts("--");
```

```
}

int main(int argc, char **argv){
      char cadena1[] = "foo";
      char *cadena2 = "bar";

      capitalizar(cadena1);
      capitalizar(cadena2);

      return 0;
}
```

17. Escriba un programa que reciba una clave de acceso a través de argumentos al main. Si la palabra clave es correcta el programa deberá imprimir un texto secreto, si la palabra clave no es correcta deberá imprimir la cantidad de parámetros recibidos.

    Por ejemplo:

```
$ ./programa abracadabra
C is quirky, flawed, and an enormous success. - Dennis Ritchie

$ ./programa algo
1 argumento/s

$ ./programa otra cosa
2 argumento/s
```

    (a) ¿Qué sucede sí la clave de acceso tiene espacios?

    (b) ¿Qué hay en el argumento argv[0]?

18. Implemente un programa que reciba como argumentos un número decimal y una base numérica e imprima en pantalla el número convertido a la base pedida. Utilice la función definida en la pregunta 3.

    Por ejemplo:

```
$ ./convertir 20050 2
100111001010010

$ ./convertir 20050 16
4e52
```

19. Realice un programa que reciba un argumento y luego, leyendo desde la entrada estándar determine si el argumento como string aparece en la lectura. Comparar con `grep -qF` (ver opción de `-m1`) en *GNU/Linux*. Modificar el programa para que determine la cantidad de ocurrencias del string. Comparar con `grep -cF` en *GNU/Linux*.

20. Escribir un programa que funcione de forma similar a `seq(1)` en *GNU/Linux* con la siguiente interfaz: `seq [INCREMENT] FIRST LAST`.

21. Re-escribir la implementación de `grep(1)` para que funcione como `grep -nF PATTERN` y, además, si se invoca con `-h|-help` muestre el uso, o, si faltase algún argumento lo indique en el error estándar.

22. Escribir un programa que funcione como `tr(1)` en *GNU/Linux* sin opciones/flags. Los argumentos, `SET1` y `SET2` deben ser tratados como strings de chars tradicionales sin secuencias "especiales".

23. Incorporar al programa `tr` el argumento/flag `-d|-delete` y, además, si se invoca con `-h|-help` muestre como el uso, o, si faltase algún argumento lo indique en el error estándar.

24. Escribir un programa que reciba como argumentos:

    - **-s:** suma los siguientes 2 parámetros.
    - **-r:** resta los siguientes 2 parámetros.
    - **-d:** divide los siguientes 2 parámetros.
    - **-m:** multiplica los siguientes 2 parámetros.
    - **-i:** imprime todos los parámetros recibidos.
    - **-h:** imprime un texto de ayuda.

    El programa realiza una operación entre los parámetros ingresados, por ejemplo, si se ingresa `-r 20 5`, el programa realizará la resta entre ambos parámetros, y lo informará de la siguiente manera:

    ```
    El resultado de la resta es 15.
    ```

    Si se ingresa `-r 20 5 -i`, el programa imprimira los parámetros y el resultado de la siguiente manera:

    ```
    Los parámetros ingresados son "-r" "20" y "5"
    El resultado de la resta es 15.
    ```

    Si se ingresa el parámetro `-h`, SOLO se imprimirá la siguiente ayuda:

    ```
    El programa imprime el resultado de enviar una operación y sus
    correspondientes parámetros. El resultado será impreso en
    pantalla.

    Argumentos:
            -s: suma los siguientes 2 parámetros.
            -r: resta los siguientes 2 parámetros.
            -d: divide los siguientes 2 parámetros.
            -m: multiplica los siguientes 2 parámetros.
            -i: imprime todos los parámetros recibidos.
            -h: imprime un texto de ayuda.
    ```

    Ejemplos de uso:

    ```
    $ ./main -r 20 5 -i
    Los parámetros ingresados son "-r" "20" y "5"
    El resultado de la resta es 15.
    ```

25. Cree una función con la siguiente firma/prototipo `int *crear_vector_int(int n)` que aloque de forma dinámica un arreglo de tamaño `n` y lo retorne.

    Cree además una función que permita liberar la memoria reservada.

26. Cree una función que permita aumentar o reducir el espacio reservado por la función `crear_vector_int()`.

27. Cree una función con la siguiente firma/prototipo
`int replace(const char *l, const char *e1, const char *e2, char **r)`
La cual reemplaza las ocurrencias de `e1` por `e2` en `l` y el resultado se devuelve en `r`. Ver de hacer una implementación que solo reemplaza la primera ocurrencia y otra que reemplaza todas las ocurrencias. Luego hacer un prorama que la utilce. Comparar con el programa `sed(1)` en *GNU/Linux* con la opción `s/exp/replacement/` o agregando `g` al final, `s/exp/replacement/g`. Considerar `e1` y `e2` como strings de chars tradicionles sin uso de expresiones regulares.

28. Escriba un programa que implemente el tipo de datos pila de chars utilizando memoria dinámica. Deberá implementar las funciones:

    - `crear_pila()`: Inicializa la pila.
    - `destruir_pila()`: Libera la memoria utilizada por la pila.
    - `apilar() /* o push */`: Apila un elemento reservando memoria de ser necesario.
    - `desapilar() /* o pop */`: Desapila un elemento (no libera memoria).

29. Utilice las funciones y variables definidas en el ejercicio anterior para escribir un programa que lea un texto de la entrada estándar, de a un char, hasta EOF y lo imprima invertido (i.e. la última letra se imprimirá primero, etc...).

## Ejercicio adicional

Investigue usando GDB el motivo por el cuál el siguiente programa deja acceder a cualquier usuario que ponga un password largo (probar con 7 o 13 chars) pero funciona correctamente para passwords más cortos:

```c
#include <stdio.h>
#include <string.h>

int login(char *user, char *passwd){
   int entrar = 0;
   char p[6];
   char u[6];
   printf("Usuario: "); gets(u);
   printf("Password: "); gets(p);
   if (strcmp(user, u) == 0 && strcmp(passwd, p) == 0){
      entrar = 1;
   }
   return entrar;
}

int main(){
   char user[] = "admin";
   char passwd[] = "12345";
   if (login(user, passwd)){
      puts("Pudo ingresar con el usuario admin");
   }
   else{
      puts("Nombre de usuario o password incorrecto");
   }
   return 0;
```

```
}
```

Ejemplo de uso:

```
$ ./login
Usuario: a
Password: aaaaaaaaaaaa
Nombre de usuario o password incorrecto
$ ./login
Usuario: a
Password: aaaaaaaaaaaaa
Pudo ingresar con el usuario admin
```

¿Cómo solucionaría este bug? ¿Por qué no falla con 6 o 12 chars? ¿Por qué el programa lanza "Violación de segmento" para passwords demasiado largos?

Si está compilando en un sistema de 64 bits también pruebe compilar con la opción -m32 y observe cuantos chars son necesarios en el password para acceder.

Ayudas, usando GDB:

- Establecer un breakpoint en la línea 4 (`break 4`)
- Ejecutar el programa (`run`)
- Ver las direcciones de memoria de las variables:

```
printf "%p\n", &entrar
printf "%p\n", p
```