

Práctica de Laboratorio #3 **Programación en C**

Objetivo

Que el estudiante aprenda el procedimiento necesario para desarrollar una aplicación en donde se utilicen funciones en el lenguaje de programación C, haciendo énfasis en el paso de argumentos por referencia, la manipulación de punteros, y el uso de punteros dobles.

Teoría

La programación en C permite un control detallado sobre la memoria, lo cual es esencial para el desarrollo eficiente de sistemas embebidos y otras aplicaciones de bajo nivel. Un aspecto clave de este control es el uso de punteros, que son variables que almacenan direcciones de memoria en lugar de valores de datos. Los punteros se pueden utilizar para pasar argumentos por referencia a funciones, lo que permite que una función modifique directamente los valores de las variables en la función llamante.

Además, los punteros dobles (es decir, punteros que apuntan a otros punteros) son una herramienta poderosa en C. Estos se utilizan comúnmente para manejar estructuras de datos dinámicas, como matrices multidimensionales, listas enlazadas, y otras estructuras complejas. Comprender la manipulación de punteros y punteros dobles es crucial para escribir código eficiente y flexible en C.

Paso de Argumentos por Referencia

Cuando se pasan argumentos a una función en C, estos pueden ser pasados por valor o por referencia. En el caso del paso por valor, se envía una copia del valor al parámetro de la función, por lo que cualquier modificación del parámetro dentro de la función no afecta al valor original. Por otro lado, al pasar un argumento por referencia, se pasa la dirección de la variable en lugar del valor en sí. Esto se logra utilizando punteros. Como resultado, cualquier cambio realizado en el parámetro dentro de la función afecta directamente a la variable original.

Manipulación de Punteros y Punteros Dobles

En C, los punteros son una herramienta clave para trabajar con memoria dinámica y estructuras de datos. Los punteros pueden ser incrementados, decrementados, y desreferenciados para acceder al valor en la ubicación de memoria a la que apuntan. Los punteros dobles, o punteros a punteros, permiten la manipulación de estructuras más complejas, como arrays de punteros o funciones que necesitan modificar un puntero pasado como argumento.



Materiales y Equipo (Proporcionados por el Laboratorio)

- PC con Windows 10 y Keil μ Vision5.

Procedimiento

Siga los siguientes pasos prestando atención a las indicaciones de su instructor. Si tiene alguna duda o no está seguro de cómo proceder, pregunte a su instructor.

1. Ejecute Keil μ Vision5 y cree un proyecto para el microcontrolador genérico ARM Cortex-M4, utilice como nombre de su proyecto "**Lab3_SuNombre_Carnet**" y como ubicación, una carpeta con el mismo nombre.
2. Cree un archivo con el nombre **main.c** y agregue el siguiente código.

```
#include "ARMCM4_FP.h"                // Device header

static const char numbers[] = "36, 60, 48";

int main(void){

    int result = myGCD(numbers);


    while (1);

}
```

3. En esta práctica, el estudiante aprenderá a implementar una función en C que calcule el máximo común divisor (MCD) de un conjunto de números enteros que se pasan como una cadena de caracteres. La función tendrá el siguiente prototipo:

```
int myGCD(const char *set);
```

La función myGCD recibirá un único argumento, que es un puntero a una cadena de caracteres (**const char *set**). Esta cadena contendrá los números enteros separados por comas, por ejemplo: "**36, 60, 48**". La función deberá:

- a) **Parsear** la cadena para extraer los números enteros.
 - b) **Calcular** el MCD de los números extraídos.
 - c) **Retornar** el MCD como un entero.
4. Guarde el archivo y luego compile su proyecto. Si obtuvo errores y/o warnings solúcelos y documéntelos en su reporte.
 5. Finalmente vamos a utilizar el depurador de Keil. Para ello, presione las teclas CTRL+F5 o haga clic en el siguiente botón .

6. Antes de ejecutar la aplicación, agregue la variables **result** al **Watch 1**, esto para verificar el correcto funcionamiento de las subrutinas. Puede utilizar **printf** y la ventana de depuración si así lo desea.
7. Ejecute la aplicación paso a paso para verificar su correcto funcionamiento.
8. Ahora, deberá implementar una función en C que calcule todas las combinaciones posibles de **n** elementos dentro de un conjunto de **k** elementos. Agregue la declaración y definición en el mismo archivo. La función deberá seguir el siguiente prototipo:

```
void myCombinations(int n, const char *set, char *comb);
```

- a) **n**: Especifica el número de elementos que deben incluirse en cada combinación, **n** tiene que ser menor o igual a 3.
 - b) **set**: Es una cadena de caracteres que describe un conjunto de elementos en el siguiente formato: "A = {rojo, verde, azul}". La cardinalidad del conjunto debe de ser menor o igual a 5.
 - c) **combinations**: Es un puntero a una cadena de caracteres donde se almacenarán las combinaciones generadas. Las combinaciones deben seguir el formato: "{rojo, verde}, {rojo, azul}, ...".
9. Modifica el **main()** para poder probar su correcto funcionamiento, puede apoyarse utilizando la función **printf** si así lo desea.
 10. Guarde el archivo y luego compile su proyecto. Si obtuvo errores y/o warnings solúcelos y documéntelos en su reporte.
 11. Ejecute la aplicación paso a paso para verificar su correcto funcionamiento.
 12. Si ha llegado hasta acá, notifíquelo a su auxiliar quien le hara varias preguntas sobre el desarrollo de la práctica, si ha respondido todas las preguntas. Pídale a un compañero que le de una palmada en la espalda.