

Práctica 1:

Introducción al manejo

de C y medidas de

rendimiento



Realizado por:

Daniel García Márquez



1.- Entorno de trabajo

El entorno de trabajo en el cual realizaremos esta práctica contaremos con un equipo con sistema operativo **macOS High Sierra**, con el compilador **GCC** y el depurador **GDB** instalados previamente y **Gedit** como editor de texto.

2.- Introducción a la programación en C

El objetivo de este apartado es familiarizarnos con los entornos de desarrollo en C. Donde aprenderemos a compilar, ejecutar y depurar un programa en lenguaje C. En el guion de la práctica se nos recomienda crear una carpeta en el escritorio llamada "**Practica1**", con el fin de tener organizado el trabajo de esta sesión.

Procederemos a escribir nuestro primer programa en C. Para ello desde nuestro terminal ejecutaremos Gedit con el comando "**sudo gedit**", deberemos escribir nuestra contraseña de usuario. Una vez abierto, escribiremos las siguientes líneas de código:

 A screenshot of the Gedit text editor window. The title bar says "*hola.c (~/Desktop) - gedit". The code in the editor is:


```

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i=0;
6     printf("Hola.\n");
7     printf("El valor de i es = %d \n", i);
8
9 }
```

Figura 1 – Programa en C que devuelve por pantalla una cadena de texto

y guardaremos el archivo como **hola.c** en la carpeta creada previamente. Cerraremos gedit que se sigue ejecutando en la terminal con las teclas **Ctrl+Z** y a través de los comandos **ls** y **cd** llegaremos hasta la dirección de nuestra carpeta donde anteriormente hemos guardado nuestro archivo c.

 A screenshot of a terminal window. The text is:


```

MacBook-Pro-de-Daniel-2:~ Daniel$ cd Desktop
MacBook-Pro-de-Daniel-2:Desktop Daniel$ cd Practica1
MacBook-Pro-de-Daniel-2:Practica1 Daniel$ ls
hola.c
```

Figura 2 – Comandos utilizados para hallar la carpeta Practica1

Una vez colocado en él, lanzaremos el compilador para que realice la compilación de la siguiente manera: "**gcc hola.c**". Este proceso nos creará un fichero binario ejecutable por el sistema denominado "**a.out**" para ejecutar el fichero binario, desde la terminal escribiremos "**./a.out**" y obtendremos una salida por pantalla.



```
[MacBook-Pro-de-Daniel-2:Practical Daniel$ gcc hola.c
[MacBook-Pro-de-Daniel-2:Practical Daniel$ ./a.out
Hola.
El valor de i es = 0
```

Figura 3 – Salida por pantalla del archivo hola.c (**Actividad 1**)

Ejecutando el comando “gcc –help” nos despliega un montón de información acerca de gcc. También podrás encontrar toda esta información más detallada en su página web. Una de las opciones más útiles y que usaremos siempre es la siguiente:

```
-o <file>           Write output to <file>
```

Figura 4 – Opción de gcc que nos permite al compilar guardar el archivo con un nombre determinado “**gcc –o hola hola.c**” (**Actividad 2**)

Vamos a utilizar esta opción para el ejemplo anterior y así poder ver su funcionamiento de un punto de vista más práctico, quedando nuestra carpeta mejor organizada al tener cada fichero binario con su correspondiente nombre sin la extensión .c.

```
[MacBook-Pro-de-Daniel-2:Practical Daniel$ gcc -o hola hola.c
[MacBook-Pro-de-Daniel-2:Practical Daniel$ ./hola
Hola.
El valor de i es = 0
```

Figura 5 – Uso de la opción –o de gcc en un ejemplo práctico.

Actividad 3 - El proceso de generación de un fichero binario ejecutable se compone de dos fases: Una primera de compilación y otra segunda de enlazado (link) de las librerías. ¿Qué comandos tendríamos que ejecutar para realizar esta compilación en dos fases de nuestro programa hola.c? (0.5 p.)

Para ejecutar una compilación en dos fases usaremos los comandos “**gcc –c**”, la opción -c del compilador lo que le dice es que detenga el proceso antes de enlazar, creando los ficheros .o necesarios, y el comando “**gcc –o**” para unir. Por lo que quedaría de la siguiente forma:

```
[MacBook-Pro-de-Daniel-2:Practical Daniel$ gcc -c hola.c
[MacBook-Pro-de-Daniel-2:Practical Daniel$ ls
a.out                  ejemplo2.c          ejercicio6Eficiente.c
ejemplo1               ejercicio6          hola
ejemplo1.c              ejercicio6.c        hola.c
ejemplo2               ejercicio6Eficiente   hola.o
[MacBook-Pro-de-Daniel-2:Practical Daniel$ gcc -o hola2pasos hola.o
[MacBook-Pro-de-Daniel-2:Practical Daniel$ ./hola2pasos
Hola.
El valor de i es = 0
```

Figura 6 – Uso de los comandos –c y –o de gcc para la compilación por partes



Actividad 4 - Acceda a este enlace: <https://www.cs.cmu.edu/~gilpin/tutorial/> Se pide hacer un resumen del documento de no más de media cara de folio A-4. ¿Qué parámetro del compilador necesitamos usar para que funcione nuestro depurador? (1 p.)

En algún punto de la carrera de un programador, necesitará de una herramienta de depuración para adivinar cuál es el problema. Para ello nos muestran la herramienta GDB, que podemos instalar usando el comando “**sudo apt-get install gdb**”. Una vez instalado su uso es muy sencillo, primeramente, tendremos que llamar a `gdb ./hola` si nuestro programa se denomina `hola`. Para proceder a la depuración bastaría con escribir el comando `run` y ahí podríamos ver paso a paso el funcionamiento de dicho programa, y viendo, en qué momento del recorrido es donde falla o salta alguna excepción que nos imposibilite seguir con el correcto funcionamiento del programa.

El parámetro del compilador que necesitamos para que funcione nuestro depurador es el siguiente: **gcc -o hola hola.c -std=c99 -Wall -DTEST --debug**. Para ejecutarlo faltaría hacer `./hola`.

Actividad 5 - ¿Qué es Valgrind? Responda brevemente y céntrese en su aplicación relacionada con la memoria. Instale Valgrind en su máquina (“sudo apt-get install valgrind**”). Haga un ejemplo con una fuga o “leak” de memoria y otro con una lectura inválida de memoria y registre la salida de Valgrind en ambos casos. ¿Cómo valora su utilidad en relación a GDB? (1 p.)**



3.- Introducción al rendimiento de una aplicación

En este apartado conoceremos la influencia que tiene sobre la velocidad de los programas el acceso a memoria. El objetivo de este apartado es conocer la influencia que produce el recorrido erróneo de un bucle en la velocidad de ejecución de un programa. Para ello nos dan una serie de códigos y cuestiones que debemos resolver.

Actividad 1 - ¿Qué realiza siguiente código?

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    int i, j, h;
    double column_sum[1000];
    double b[1000][1000];
    for (i = 0; i < 1000; i++) {
        for (j = 0; j < 1000; j++) {
            b[j][i] = 1.0;
        }
    }
    for (h = 0; h < 50; h++) {
        for (i = 0; i < 1000; i++) {
            column_sum[i] = 0.0;
            for (j = 0; j < 1000; j++) {
                column_sum[i] += b[j][i];
            }
        }
    }
}
```

Figura 1 – Código sobre el que se basa la actividad 1 y 2.

Empecemos por el principio, encontramos las variables *i*, *j* y *h* inicializadas y de tipo *int*. Seguidamente observamos dos datos de tipo *doublé*, uno con espacio de 1000 celdas y otro un bidimensional de 1000 caracteres cada uno. En la siguiente línea encontramos un bucle *for* anidado que recorre la matriz *b* y asigna en cada celda el valor “1.0”. Al terminar estos dos bucles encadenados, nos encontramos con otros 3 bucles *for* encadenados, que su función es repetir 50 veces la suma iterativa de cada celda de la matriz que anteriormente hemos modificado y guardar el valor en cada columna de *column_sum*. Parece un poco lioso, pero para que nos entendamos, pensemos que la variable *b* es una matriz formada por columnas y filas (al fin y al cabo, es así) y lo que hacemos es seleccionar una columna, sumar de forma recursiva todas las filas pertenecientes a dicha columna y almacenar el valor total en la columna de nuestra variable *column_sum*, y este proceso se repite 50 veces.

Actividad 2: Compilar el programa anterior y decir cuál es el tiempo real empleado para distintos valores de h (50,100,500,1000,5000,10000). (0.5 p.)

En este apartado recurriremos al comando **time**, que lo vamos a utilizar para calcular el tiempo que tarda nuestro ordenador en ejecutar el programa de la actividad 1. Para ello, debemos ir cambiando el valor de h. Cada vez que guardemos el archivo con el nuevo valor tenemos que compilarlo de nuevo, no vale guardar simplemente y ejecutar con el comando time. Una vez sabido esto, ejecutaremos de la siguiente manera:

```
[MacBook-Pro-de-Daniel-2:Practical Daniel]$ gcc -o ejemplo1 ejemplo1.c
[MacBook-Pro-de-Daniel-2:Practical Daniel]$ time ./ejemplo1

real    0m0.200s
user    0m0.191s
sys     0m0.004s
```

Figura 2 – El tiempo que tarda para h=50.

A medida que vamos guardando, compilando y ejecutando vamos rellenando la siguiente tabla con todos los valores resultantes.

Valor de h	Real	User	Sys
50	0.200s	0.191s	0.004s
100	0.378s	0.371s	0.004s
500	1.886s	1.876s	0.007s
1000	3.714s	3.702s	0.008s
5000	18.564s	18.542s	0.015s
10000	36.866s	36.830s	0.027s

Como podemos ver a simple vista mirando los datos progresan de manera lineal, aunque para verlo más visual procedemos a crear una gráfica.



Figura 3 – Representación gráfica de los tiempos obtenidos frente a h.



Actividad 3 - ¿Qué realiza siguiente código?

```
#include<stdio.h>
#include<stdlib.h>
int main() {
    int i, j, h;
    double column_sum[1000];
    double b[1000][1000];
    for (i = 0; i < 1000; i++) {
        for (j = 0; j < 1000; j++) {
            b[j][i] = 1.0;
        }
    }
    for (h = 0; h < 50; h++) {
        for (i = 0; i < 1000; i++) {
            column_sum[i] = 0.0;
        }
        for (j = 0; j < 1000; j++) {
            for (i = 0; i < 1000; i++) {
                column_sum[i] += b[j][i];
            }
        }
    }
}
```

Figura 4 – Código sobre el que se basa la actividad 3 y 4.

Para no alargar este apartado, veo conveniente que leamos la actividad 1 y una vez la hayamos entendido, procederemos a leer esta. Las únicas diferencias que encontramos respecto al anterior código es que en el segundo bucle anidado, inicializamos la variable `column_sum` a 0.0 todas sus celdas. Y la segunda diferencia es el orden de escritura en los bucles siguientes, que primero lee el bucle `j` y luego el bucle `i`.

Actividad 2: Compilar el programa anterior y decir cuál es el tiempo real empleado para distintos valores de h (50,100,500,1000,5000,10000). (0.5 p.)

En este apartado recurriremos al comando **time**, que lo vamos a utilizar para calcular el tiempo que tarda nuestro ordenador en ejecutar el programa de la actividad 1. Para ello, debemos ir cambiando el valor de h. Cada vez que guardemos el archivo con el nuevo valor tenemos que compilarlo de nuevo, no vale guardar simplemente y ejecutar con el comando time. Una vez sabido esto, ejecutaremos de la siguiente manera:

```
[MacBook-Pro-de-Daniel-2:Practical Daniel$ gcc -o ejemplo2 ejemplo2.c
[MacBook-Pro-de-Daniel-2:Practical Daniel$ time ./ejemplo2

real    0m0.165s
user    0m0.158s
sys     0m0.004s
```

Figura 5 – El tiempo que tarda para h=50.

A medida que vamos guardando, compilando y ejecutando vamos rellenando la siguiente tabla con todos los valores resultantes.

Valor de h	Real	User	Sys
50	0.165s	0.158s	0.004s
100	0.293s	0.286s	0.004s
500	1.515s	1.500s	0.008s
1000	2.854s	2.845s	0.006s
5000	15.511s	15.423s	0.048s
10000	30.106s	29.914s	0.027s

Como podemos ver a simple vista mirando los datos progresan de manera lineal, aunque para verlo más visual procedemos a crear una gráfica.



Figura 6 – Representación gráfica de los tiempos obtenidos frente a h.



Actividad 5 - Calcular la aceleración del programa 2 respecto del programa 1. Realizar una gráfica de la aceleración para los distintos valores de h. Dar una breve explicación de las diferencias entre los dos programas. ¿Por qué crees que sucede esto?

Para calcular la aceleración en este apartado vamos a coger el valor de tiempo para cada valor de h y lo dividiremos entre ellos. Una vez tengamos todos los valores, procedemos a la creación de una gráfica:



Sucede esto pues es más eficiente recorrer fila a la hora de leer, ya que si vas por columnas tienes que saltar entre arrays que consume más memoria que de la forma como está planteado el ejercicio 3. Por lo que a la hora el orden de lectura o escritura en un array puede marcar la diferencia con valores de h muy altos.



Los ejercicios restantes ya iré a tutoría para que me resuelva algunas dudas que tengo aunque no sirvan para la nota ya, pues me saltaba un error todo el rato y por más que lo mirase no había forma de encontrar el error, así que con más paciencia intentaré resolverlos. Le dejo una foto del ejercicio 6 “mejorado” y del error que me salta pues yo no he sabido solucionarlo de momento. El 6 “peor” sería cambiando el orden de los bucles, tal como ocurre en los ejercicios anteriores y el rendimiento es peor.

```

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main(int argc, char* argv[]){
5     //Declaración de variables
6
7     int tamano = atoi(argv[100]);
8
9     int* p_vector = malloc(sizeof(int)*tamano);    //Reservamos el espacio dinámicamente
10    for(int i=0; i<tamano; i++) {                  //Inicializar a 1 el vector
11        p_vector[i] = 1;
12    }
13
14
15    int** p_matriz = malloc(sizeof(int*)*tamano); //Reservamos el espacio dinámicamente
16    for(int i = 0; i < tamano; i++){
17        p_matriz[i] = malloc(sizeof(int)*tamano);
18    }
19    for(int i = 0; i < tamano; i++){
20        for(int j = 0; j < tamano; j++){           //Inicializar a 2 la matriz
21            p_matriz[i][j]= 2;
22        }
23    }
24
25    int* p_resultado = malloc(sizeof(int)*tamano); //Reservamos el espacio dinámicamente
26    for(int i=0; i<tamano; i++) {                  // Inicializar a 0 el vector donde guardaremos
27        p_resultado[i] = 0;                         // el resultado de la multiplicación
28    }
29
30    for(int i = 0; i<tamano; i++) {
31        for(int j = 0; j<tamano; j++){
32            p_resultado[j] += p_vector[i]*p_matriz[i][j];
33        }
34    }
35
36    // Liberamos memoria
37
38    free(p_vector);
39    for (int i=0; i<tamano; i++){
40        free(p_matriz[i]);
41    }
42    free(p_matriz);
43    free(p_resultado);
44
45 }
46 }
```

```
[MacBook-Pro-de-Daniel-2:Practical Daniel$ time ./ejercicio6
Segmentation fault: 11

real    0m0.005s
user    0m0.001s
sys     0m0.002s
```

He intentado poner los punteros en double, por si fuese por eso, pero tampoco.