UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE0117 PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

Reporte

Laboratorio 6

Prof. Carolina Trejos Quirós

Estudiante: Diego Pereira[1], B85938

02 de diciembre de 2025

ÍNDICE

1. Introducción

Este laboratorio de programación bajo plataformas abiertas hace uso de memoria tanto estática como dinámica para la creación de arreglos con estructura "*struct*" de tipo "*Persona*", donde cada posición (*Persona*) en el arreglo tiene un nombre, edad, y altura en centímetros. Se crean dos arreglos con la misma información de personas, sin embargo, un arreglo es creado de manera estática y el otro de forma dinámica, ello con el fin de probar los algoritmos existentes de ordenamiento como lo es "*qsort*".

2. Implementación

2.1. Ejercicio1: Ordenamiento de estructuras de datos

La función "*qsort*" en C es parte de la librería estándar "*stdlib.h*", usada para ordenar arreglos de datos arbitrarios o no ordenados, y es una variante del algoritmo "*quick sort*" u ordenamiento rápido, la cual requiere recibir como parámetro un puntero a la función que indica el tipo de ordenamiento al realizar la comparación de dos valores.

2.1.1. Orden por nombre ascendente

La Figura 1 muestra la función que compara los nombres tipo "*char*", donde para un ordenamiento ascendente se debe retornar -1 si el primero es menor que el segundo, +1 si el primero es mayor que el segundo, y 0 si ambos son iguales. Para el caso de un orden descendente, los valores de retorno se deben invertir para las condiciones -1 y +1.

```
273  int compare_by_name(const void *left, const void *right) {
274      // cast void pointers to Person* (pointers to struct)
275      const Person *personL = (const Person*)left;
276      const Person *personR = (const Person*)right;
277      // result -1 for Left < Right, +1 for Left > Right, or 0 Left = Right
278      // use strcmp to compare the strings (case-sensitive)
279      //int result = strcmp(personL->name, personR->name);
280      // use strcasecmp to compare strings (case-insensitive)
281      int result = strcasecmp(personL->name, personR->name);
282      return (result);
283  }
```

Figura 1. Elaboración propia. *Función que realiza ordenamiento por nombre ascendente*, 2025.

2.1.2. Orden por edad ascendente

La Figura 2 muestra la función que compara las edades tipo "*int*" para orden también ascendente, donde si la primera es menor que la segunda se retorna -1, +1 si la primera es mayor que la segunda, y 0 si ambas son iguales. Cuyos valores de retorno -1 y +1 también se debe invertir si se desea un orden descendente.

```
285 int compare_by_age(const void *left, const void *right) {
286     // cast void pointers to Person* (pointers to struct)
287     const Person *personL = (const Person*)left;
288     const Person *personR = (const Person*)right;
289     if (personL->age < personR->age) {
290         return (-1);
291     } else if (personL->age > personR->age) {
292         return (1);
293     } else {
294         return (0); // are equal
295     }
296 }
```

Figura 2. Elaboración propia. *Función que realiza ordenamiento por edad ascendente*, 2025.

2.1.3. Orden por altura ascendente

La Figura 3 muestra la función que compara las alturas tipo "*double*" para orden también ascendente, donde si la primera es menor que la segunda se retorna -1, +1 si la primera es mayor que la segunda, y 0 si ambas son iguales. Cuyos valores de retorno -1 y +1 también se debe invertir si se desea un orden descendente.

```
298 int compare_by_height(const void *left, const void *right) {
299     // cast void pointers to Person* (pointers to struct)
300     const Person *personL = (const Person*)left;
301     const Person *personR = (const Person*)right;
302     if (personL->height < personR->height) {
303         return (-1);
304     } else if (personL->height > personR->height) {
305         return (1);
306     } else {
307         return (0); // are equal
308     }
309 }
```

Figura 3. Elaboración propia. *Función que realiza ordenamiento por altura ascendente*, 2025.

2.2. **Ejercicio2**: Repositorio

Dado que el ejercicio 2 se refiere a la estructura de directorios y archivos en el repositorio, y la nueva forma de compilación y ejecución mediante el uso del comando "*make*", para más información acerca de esta sección se puede consultar el archivo README o la sección de resultados.

3. RESULTADOS

3.1. **Ejercicio1**: Ordenamiento de estructuras de datos

Esta sección muestra los resultados obtenidos para el ordenamiento de una estructura "*struct*" de tipo "*Persona*" guardada en un arreglo, y ordenada ascendentemente mediante tres tipos de datos distintos, los cuales son el nombre, la edad, y la altura.

3.1.1. Orden por nombre ascendente

La Figura 5 muestra el resultado de un ordenamiento ascendente según el dato de nombre.

```
---------------- NOMBRE ASCENDENTE ----------------
N# Edad Altura Nombre
 1   24    164 Adrián Méndez Chavarría
 2   21    168 allan Castro Acosta
 3   19    170 Andrés Zumbado Moreira
 4   21    166 Angelica Isabel Aguilar Jiménez
 5   22    165 Emily Maryan Flores Rojas
 6   21    161 Felipe Alberto Mata Mata
 7   19    173 George Brian Morison Pallais
 8   19    171 Javier Fernando Bolaños Castellón
 9   19    176 Jonatan Hidalgo Morales
10   22    174 José Manuel Solís Quesada
11   29    172 José Mario Monge Guerrero
12   22    169 Livia Valentina Corrales Madrigal
13   23    177 Melany Dayana Rosales Montiel
14   22    175 Pablo Andrés Bermudez Duarte
15   20    167 Randall Alonso Méndez Blanco
16   22    163 Randy Arturo Barrantes Arroyo
17   20    162 Sebastián Alejandro Araya Fuks
```

Figura 5. Elaboración propia. *Resultado del ordenamiento por nombre ascendente*, 2025.

3.1.2. Orden por edad ascendente

La Figura 6 muestra el resultado de un ordenamiento ascendente según el dato de edad.

```
----------------- EDAD ASCENDENTE -----------------
N# Edad Altura Nombre
 1   19    170 Andrés Zumbado Moreira
 2   19    173 George Brian Morison Pallais
 3   19    171 Javier Fernando Bolaños Castellón
 4   19    176 Jonatan Hidalgo Morales
 5   20    167 Randall Alonso Méndez Blanco
 6   20    162 Sebastián Alejandro Araya Fuks
 7   21    168 allan Castro Acosta
 8   21    166 Angelica Isabel Aguilar Jiménez
 9   21    161 Felipe Alberto Mata Mata
10   22    165 Emily Maryan Flores Rojas
11   22    174 José Manuel Solís Quesada
12   22    169 Livia Valentina Corrales Madrigal
13   22    175 Pablo Andrés Bermudez Duarte
14   22    163 Randy Arturo Barrantes Arroyo
15   23    177 Melany Dayana Rosales Montiel
16   24    164 Adrián Méndez Chavarría
17   29    172 José Mario Monge Guerrero
```

Figura 6. Elaboración propia. *Resultado del ordenamiento por edad ascendente*, 2025.

3.1.3. Orden por altura ascendente

La Figura 7 muestra el resultado de un ordenamiento ascendente según el dato de altura.

```
---------------- ALTURA ASCENDENTE ----------------
N# Edad Altura Nombre
 1   21    161 Felipe Alberto Mata Mata
 2   20    162 Sebastián Alejandro Araya Fuks
 3   22    163 Randy Arturo Barrantes Arroyo
 4   24    164 Adrián Méndez Chavarría
 5   22    165 Emily Maryan Flores Rojas
 6   21    166 Angelica Isabel Aguilar Jiménez
 7   20    167 Randall Alonso Méndez Blanco
 8   21    168 allan Castro Acosta
 9   22    169 Livia Valentina Corrales Madrigal
10   19    170 Andrés Zumbado Moreira
11   19    171 Javier Fernando Bolaños Castellón
12   29    172 José Mario Monge Guerrero
13   19    173 George Brian Morison Pallais
14   22    174 José Manuel Solís Quesada
15   22    175 Pablo Andrés Bermudez Duarte
16   19    176 Jonatan Hidalgo Morales
17   23    177 Melany Dayana Rosales Montiel
```

Figura 7. Elaboración propia. *Resultado del ordenamiento por altura ascendente,* 2025.

3.1.4. Análisis de memoria con "*valgrind*"

La Figura 8 muestra los resultados del uso de memoria mediante el comando *"valgrind"*.

```
diego@pereira:~/lab06$ ./lab06e1fp_run_valgrind
==2474== Memcheck, a memory error detector
==2474== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==2474== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==2474== Command: /home/diego/lab06/bin/lab06e1fp_main
==2474==
################################################
******** QSORT SÓLO ORDENA STRUCT ESTÁTICO ********
################################################
==2474==
==2474== HEAP SUMMARY:
==2474==     in use at exit: 0 bytes in 0 blocks
==2474==   total heap usage: 19 allocs, 19 frees, 1,568 bytes allocated
==2474==
==2474== All heap blocks were freed -- no leaks are possible
==2474==
==2474== For lists of detected and suppressed errors, rerun with: -s
==2474== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
removed '/home/diego/lab06/bin/lab06e1fp_main'
removed directory '/home/diego/lab06/bin'
diego@pereira:~/lab06$
```

Figura 8. Elaboración propia. *Resultado de la revisión de memoria con "valgrind",* 2025.

3.2. **Ejercicio2**: Repositorio

3.2.1. Contenido del repositorio

Se muestra la misma estructura de directorios y archivos que contiene el repositorio en la nube.

```
diego@pereira:~/lab06$ ls -las
total 64
 4 drwxrwxr-x  4 diego diego  4096 Dec  1 06:11 .
 4 drwxr-x--- 16 diego diego  4096 Dec  1 07:02 ..
 4 drwxrwxr-x  2 diego diego  4096 Dec  1 06:10 include
20 -rw-r--r--  1 diego diego 17795 Dec  1 06:10 lab06e1fp_aio.c
 4 -rw-r--r--  1 diego diego  2720 Dec  1 06:10 lab06e1fp_main.c
 8 -rwxrwxr-x  1 diego diego  4385 Dec  1 06:10 lab06e1fp_run_aio
 4 -rwxrwxr-x  1 diego diego  3587 Dec  1 06:10 lab06e1fp_run_gcc
 4 -rwxrwxr-x  1 diego diego  3440 Dec  1 06:10 lab06e1fp_run_make
 4 -rwxrwxr-x  1 diego diego  3586 Dec  1 06:10 lab06e1fp_run_valgrind
 4 -rwxrwxr-x  1 diego diego  3958 Dec  1 06:10 Makefile
 4 drwxrwxr-x  2 diego diego  4096 Dec  1 06:10 src
diego@pereira:~/lab06$ ls -las include/
total 20
4 drwxrwxr-x 2 diego diego 4096 Dec  1 06:10 .
4 drwxrwxr-x 4 diego diego 4096 Dec  1 06:11 ..
4 -rw-r--r-- 1 diego diego 1484 Dec  1 06:10 Person.h
4 -rw-r--r-- 1 diego diego 1663 Dec  1 06:10 umath.h
4 -rw-r--r-- 1 diego diego 1490 Dec  1 06:10 usort.h
diego@pereira:~/lab06$ ls -las src/
total 24
4 drwxrwxr-x 2 oem oem 4096 Dec  1 06:10 .
4 drwxrwxr-x 4 oem oem 4096 Dec  1 06:11 ..
4 -rw-r--r-- 1 oem oem 3622 Dec  1 06:10 Person.c
4 -rw-r--r-- 1 oem oem 3935 Dec  1 06:10 umath.c
8 -rw-r--r-- 1 oem oem 6606 Dec  1 06:10 usort.c
diego@pereira:~/lab06$
```

Figura 9. Elaboración propia. *Estructura de archivos que contiene el repositorio,* 2025.

4. Análisis de Resultados

Para la sección 3.1 correspondiente al ejercicio 1 acerca de una estructura de datos en un arreglo, las Figuras 5 a la 8 muestran los resultados obtenidos tanto con el comando "*make*" para compilar y ejecutar, como con el comando "*valgrind*" para analizar el correcto uso de la memoria y verificar que no haya espacios en memoria sin ser liberados.

La Figura 5 a la 7 es el resultado de ejecución con "*make*", mientras que la Figura 8 es el resultado de la ejecución con "*valgrind*". En ambos casos se llena un arreglo con 17 personas arbitrarias o sin un orden de inserción, el cual es luego ordenado según el tipo de dato deseado, donde la Figura 5 muestra el ordenamiento por nombre, la Figura 6 por edad, y la Figura 7 por altura.

Para la sección 3.2 que corresponde al ejercicio 2, la Figura 9 muestra la misma estructura de directorios y archivos que contiene el repositorio en la nube, la cual tiene dos directorios, uno con los "*header files*" o archivos con la declaración de funciones, y otro con los "*source files*" o archivos fuente con la definición de las funciones escritas en el lenguaje C, así como el "*main*" o archivo principal ubicado fuera de dichos directorios y usado como entrada o punto de partida para ejecutar el programa haciendo uso de los otros archivos.

5. Conclusiones y recomendaciones

Se verifica que el comando "*valgrind*" es una buena herramienta para ayudar a mejorar la administración de memoria en nuestro código, especialmente cuando se usan punteros y memoria dinámica, el cual contribuye a chequear si toda la memoria reservada durante la ejecución del programa es liberada correctamente.

Y como punto adicional, se comprueba que el comando "*make*" también contribuye a agilizar el proceso de compilación y ejecución de programas, especialmente cuando la estructura de directorios y archivos se vuelve más compleja, sin embargo, es importante mencionar que no es el único comando para dicho fin, ya que también existen otras opciones como "*cmake*", "*meson*", "*ninja*", entre otros que pueden realizar la misma tarea con menor cantidad de líneas de código.