

UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA
IE0117 PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

Reporte

Laboratorio 4

Prof. Carolina Trejos Quirós

Estudiante: Diego Pereira¹, B85938

23 de octubre de 2025

ÍNDICE

1. INTRODUCCIÓN.....	3
2. IMPLEMENTACIÓN.....	3
2.1. EJERCICIO1: ORDENAR ARREGLOS Y MATRICES	3
<i>Figura 1. Elaboración propia. Algoritmo que ordena los valores de un arreglo (Ascendente/Descendente), 2025.....</i>	<i>3</i>
2.2. EJERCICIO2: REEMPLAZAR PALABRAS DE UN ARCHIVO.....	4
2.2.1. Chequear letras/vocales con acento	4
<i>Figura 2. Elaboración propia. Funciones para chequear si un símbolo se encuentra entre los permitidos, 2025.</i>	<i>4</i>
2.2.2. Buscar y reemplazar coincidencias	5
<i>Figura 3. Elaboración propia. Función para reemplazar coincidencias sólo si los extremos son palabras, 2025.</i>	<i>5</i>
3. RESULTADOS	6
3.1. EJERCICIO1: ORDENAR ARREGLOS Y MATRICES	6
<i>Figura 4. Elaboración propia. Ejemplo de matriz aleatoria ordenada (Descendente), 2025.....</i>	<i>6</i>
<i>Figura 5. Elaboración propia. Ejemplo de matriz aleatoria ordenada (Ascendente), 2025.</i>	<i>7</i>
3.2. EJERCICIO2: REEMPLAZAR PALABRAS DE UN ARCHIVO.....	7
<i>Figura 6. Elaboración propia. Archivo de prueba autogenerado para buscar coincidencias, 2025.</i>	<i>7</i>
<i>Figura 7. Elaboración propia. Resultado de valgrind y coincidencias reemplazadas, 2025.</i>	<i>8</i>
<i>Figura 8. Elaboración propia. Muestra del contenido del archivo de prueba autogenerado, 2025.</i>	<i>9</i>
<i>Figura 9. Elaboración propia. Muestra del contenido del archivo nuevo luego del reemplazo, 2025.....</i>	<i>9</i>
4. ANÁLISIS DE RESULTADOS.....	10
5. CONCLUSIONES Y RECOMENDACIONES	10

1. INTRODUCCIÓN

Con conocimientos un poco más avanzados en el lenguaje C, en este laboratorio de programación bajo plataformas abiertas no sólo se reafirma la parte introductoria de memoria estática usada en el primer ejercicio para ordenar arreglos o matrices, sino que además se pone en práctica tanto memoria dinámica como punteros para recorrer líneas contenidas en un archivo, las cuales son revisadas para buscar coincidencias y hacer los reemplazos respectivos cuando se trata únicamente de palabras completas.

2. IMPLEMENTACIÓN

2.1. Ejercicio1: Ordenar arreglos y matrices

El primer ejercicio consiste en un algoritmo de ordenamiento tipo burbuja (del inglés “bubble”), que al ejecutar el programa crea 3 matrices de tamaño aleatorio con valores también aleatorios sin ordenar, los cuales son ordenados luego usando la función que se muestra en la Figura 1, donde se recibe por parámetro una variable tipo “macro” que indica el tipo de ordenamiento (Ascendente/Descendente), y que para efectos de este laboratorio siempre se ordenan de forma ascendente.

```
280 // function to swap two integers
281 void swap_int(int *xp, int *yp) {
282     int temp = *xp;
283     *xp = *yp;
284     *yp = temp;
285 }
286 // function to perform bubble sort on an array
287 void sort_bubble(int length, int array[(length)], int order) {
288     int i, j;
289     bool swapped;
290     for (i = 0; i < (length - 1); i++) {
291         swapped = false; // flag to check if any swap occurred in this
292         // pass
293         for (j = 0; j < (length - i - 1); j++) {
294             if ((order == ASCENDING && array[j + 1] < array[j]) || (
295                 array[j] < array[j + 1] && order == DESCENDING)) {
296                 //int temp = array[j];
297                 //array[j] = array[j + 1];
298                 //array[j + 1] = temp;
299                 // swap function replaces previous lines using pointers
300                 swap_int(&array[j], &array[j + 1]);
301                 swapped = true; // set flag to true if a swap occurred
302             }
303         }
304         // if no elements were swapped, then the array is sorted
305         if (swapped == false) break;
306     }
}
```

Figura 1. Elaboración propia. Algoritmo que ordena los valores de un arreglo (Ascendente/Descendente), 2025.

2.2. Ejercicio2: Reemplazar palabras de un archivo

El segundo ejercicio consiste en leer el contenido de un archivo, en este caso se realiza línea por línea para buscar coincidencias de palabras ingresadas por el usuario, las cuales serán reemplazadas luego por un nuevo texto, cuyo resultado es almacenado en un nuevo archivo con el contenido original modificado para guardar los cambios realizados.

2.2.1. Chequear letras/vocales con acento

Para validar las coincidencias encontradas se revisan los extremos con el objetivo de averiguar si se trata de palabras completas, las cuales pueden contener únicamente letras, números, guion bajo (del inglés “*underscore*”), y algunos símbolos especiales agregados para permitir vocales con acento o el caso de la letra “*eñe*” que también tiene un acento. Cualquier otro símbolo (como el espacio en blanco) no mencionado como parte de una palabra y que se encuentre en los extremos, permitirá realizar el reemplazo por el nuevo texto ingresado por el usuario.

```

25 ifndef ALLOWED
26     // ANSI/ASCII 192 <= symbol <= 255
27     // they are 50 symbols, but each one occupies 2 bytes, so strlen is
28     // 100
29     #define ALLOWED
30         "ÀÁÂÃÄÈÉÎÏÏÑÒÓÔÔÛÛÙÛàáâãäèéêîïïñòóôôûûüýý";
31     // #define ALLOWED "";
32 #endif
33
34 bool last_next(const char *p, int b, int n) {
35     char c = '\x00';
36     return (isalpha(p[b+1]) || isdigit(p[b+1]) || p[b+1] == '_' || 
37             alphas(p[b+1], (b+2)<n?p[b+2]:c, (b+3)<n?p[b+3]:c, (b+4)<n?p[b+4]:c,
38             FORWARD));
39 }
40
41 bool last(const char *p, int b) {
42     char c = '\x00';
43     return (isalpha(p[b]) || isdigit(p[b]) || p[b] == '_' || alphas(p[b],
44             (b-1)>=0?p[b-1]:c, (b-2)>=0?p[b-2]:c, (b-3)>=0?p[b-3]:c, REVERSE));
45 }
46
47 bool first_prev(const char *p, int a) {
48     char c = '\x00';
49     return (isalpha(p[a-1]) || isdigit(p[a-1]) || p[a-1] == '_' || 
50             alphas(p[a-1], (a-2)>=0?p[a-2]:c, (a-3)>=0?p[a-3]:c, (a-4)>=0?p[a-4]:c,
51             REVERSE));
52 }
53
54 bool first(const char *p, int a, int n) {
55     char c = '\x00';
56     return (isalpha(p[a]) || isdigit(p[a]) || p[a] == '_' || alphas(p[a],
57             (a+1)<n?p[a+1]:c, (a+2)<n?p[a+2]:c, (a+3)<n?p[a+3]:c, FORWARD));
58 }
```

Figura 2. Elaboración propia. Funciones para chequear si un símbolo se encuentra entre los permitidos, 2025.

2.2.2. Buscar y reemplazar coincidencias

La Figura 3 muestra cómo se realiza un recorrido símbolo por símbolo mediante el uso de un puntero al buffer que contiene la línea de entrada, donde la función “`strstr`” chequea si hay coincidencia, pero además se revisa si se trata de una palabra completa antes de realizar el reemplazo por la nueva, ya que en caso de no serlo se deja la palabra vieja. Si el símbolo no coincide, se copia al nuevo buffer de salida.

```

294 // function to replace only complete words in file content
295 long double replace_word(const char *word_old, const char *word_new,
296                           const char *p, char **buffer_output, long double *matches) {
297
298     // perform replacement
299     i = 0, j = 0;
300     // iterate through buffer_input to replace
301     while (*buffer_input) {
302         if (strstr(buffer_input, word_old) == buffer_input) {
303             //a = 0; // first starting index
304             //b = (size_old - 1); // last ending index
305             a = j; // first starting index
306             b = (j + size_old - 1); // last ending index
307             is_word = true;
308             //is_word = ((isalpha(p[b+1]) || isdigit(p[b+1]) || p[b+1]
309             == '_') && (isalpha(p[b]) || isdigit(p[b]) || p[b] == '_'))
310             ? false : !(0 < i && (isalpha(p[a-1]) || isdigit(p[a-1]) || p[a-1] == '_') && (isalpha(p[a]) || isdigit(p[a]) || p[a] == '_'));
311             is_word = (last_next(p, b, n) && last(p, b)) ? false : !(0 <
312                 j && first_prev(p, a) && first(p, a, n)); // with accent
313             check
314             // when an instance of word_old is found, copy word_new
315             // into buffer_new
316             // strcpy or strcat functions can be helpful here
317             // must replace only if the match is a complete word
318             if (is_word) {
319                 strcpy(&buffer_new[i], word_new);
320                 i += (size_new);
321             } else {
322                 strcpy(&buffer_new[i], word_old);
323                 i += (size_old);
324             }
325             j += (size_old);
326             buffer_input += size_old;
327             //printf("%2d buffer '%s'\n", i, buffer_new);
328         } else {
329             // otherwise, manual character copying from the buffer_input
330             buffer_new[i++] = *buffer_input++;
331             j++;
332         }
333         // null terminator (\0)
334         // ensure the output buffer is null-terminated
335         buffer_new[i] = '\0';
336         // update the pointer to the output buffer
337         *buffer_output = buffer_new;
338     }
339 }
```

Figura 3. Elaboración propia. Función para reemplazar coincidencias sólo si los extremos son palabras, 2025.

3. RESULTADOS

3.1. Ejercicio1: Ordenar arreglos y matrices

```
[diego@pereira Shared]$ ./lab04e1sort_gcc
----- Matriz 1 Sin Orden -----
----- Filas 15 x Columnas 13 -----
 37  22  27  168  85  88  23   5  12  164  162  82  126
 13  10  170  151  59  103  159  72  83  75  41  65  94
181 148  20  185  11  16  119  28  42  78  120  15  157
189   9  131  141  44  121  174 110  192  163  38  48  111
 33  51  183  156  56  191  135  95  63  138  62  161  90
 34 106  178  173 152  123  97 140  154  100  177  18  64
145  55  172   4  17  93  113  52  73  102  26  70  50
 67   6  124  61  40  171  136  89  127  194   1  137  144
   3 195  153  190 129  66  160 107  108  43  54  104  45
122 130  39  53  96  150  115  68  125  133  31  158  169
180  60  91   8  2  149  143 132   7  193  80  24  134
 25 166  35  165  84  98  36 128  118  139  32  49  92
188 184  146  147  57  77  112 186  142  47  175  176  155
 74 179  86  99 105  76  101  30  14  69  87  29  117
182 114  109  58  81  46 116  19  167  71  187  21  79

----- Matriz 1 Con Orden -----
----- Filas 15 x Columnas 13 -----
195 194 193 192 191 190 189 188 187 186 185 184 183
182 181 180 179 178 177 176 175 174 173 172 171 170
169 168 167 166 165 164 163 162 161 160 159 158 157
156 155 154 153 152 151 150 149 148 147 146 145 144
143 142 141 140 139 138 137 136 135 134 133 132 131
130 129 128 127 126 125 124 123 122 121 120 119 118
117 116 115 114 113 112 111 110 109 108 107 106 105
104 103 102 101 100  99  98  97  96  95  94  93  92
 91  90  89  88  87  86  85  84  83  82  81  80  79
 78  77  76  75  74  73  72  71  70  69  68  67  66
 65  64  63  62  61  60  59  58  57  56  55  54  53
 52  51  50  49  48  47  46  45  44  43  42  41  40
 39  38  37  36  35  34  33  32  31  30  29  28  27
 26  25  24  23  22  21  20  19  18  17  16  15  14
 13  12  11  10   9   8   7   6   5   4   3   2   1
```

Figura 4. Elaboración propia. *Ejemplo de matriz aleatoria ordenada (Descendente), 2025.*

```
----- Matriz 3 Sin Orden -----
----- Filas 9 x Columnas 6 -----
 47   2   11   16   23   14
 27   7   51   45   42   34
 31   1   21   20   52   41
 54   32  12   44   22    8
 39   6   38   43   48   53
 36   26    3   24   30   46
 19   37   25   15   28   49
 29   50   35    4   17   33
   5   18    9   10   13   40
----- Matriz 3 Con Orden -----
----- Filas 9 x Columnas 6 -----
   1   2   3   4   5   6
   7   8   9   10  11  12
  13  14  15  16  17  18
  19  20  21  22  23  24
  25  26  27  28  29  30
  31  32  33  34  35  36
  37  38  39  40  41  42
  43  44  45  46  47  48
  49  50  51  52  53  54
[diego@pereira Shared]$ []
```

Figura 5. Elaboración propia. *Ejemplo de matriz aleatoria ordenada (Ascendente)*, 2025.

3.2. Ejercicio2: Reemplazar palabras de un archivo

```
[diego@pereira Shared]$ ./lab04e2replace_gcc "/home/diego/Desktop/prueba con espacios.txt" "índice i"
==1828== Memcheck, a memory error detector
==1828== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==1828== Using Valgrind-3.25.1 and LibVEX; rerun with -h for copyright info
==1828== Command: ./lab04e2replace /home/diego/Desktop/prueba\ con\ espacio s.txt __ndice\ i
==1828==
No se pudo abrir el archivo "/home/diego/Desktop/prueba con espacios.txt".
Se usará un archivo de prueba... Autogenerar ? (si/no) : y
Opción aceptada...
----- Creando Archivo -----

Fin de Línea sin CR (Carro de Retorno)
Creados          0 Bytes en Línea      1
índice i*borrado del inicio.
Fin de Línea sin CR (Carro de Retorno)
Creados          31 Bytes en Línea      2
índice i no es borrado.
Fin de Línea sin CR (Carro de Retorno)
Creados          25 Bytes en Línea      3
*borrado del final.índice i
Fin de Línea sin CR (Carro de Retorno)
Creados          30 Bytes en Línea      4
----- Final de Archivo -----
```

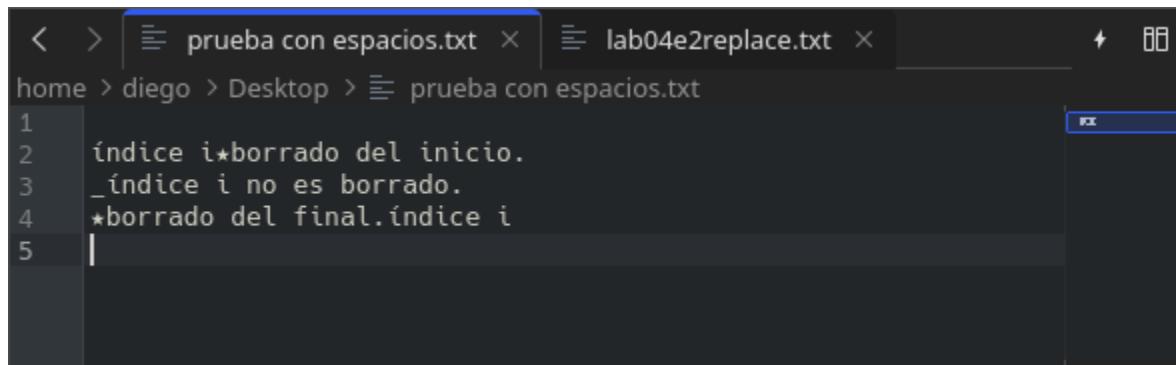
Figura 6. Elaboración propia. *Archivo de prueba autogenerado para buscar coincidencias*, 2025.

```
----- Procesando Archivo -----
Nombre del leido : "/home/diego/Desktop/prueba con espacios.txt"
Nombre del nuevo : "./lab04e2replace.txt"
Texto que se busca : "índice i"
El reemplazo nuevo : ""
----- Línea      1 -----

Fin de Línea tipo Windows (CR LF)
Leídos      0 Bytes en Línea      1

Fin de Línea tipo Windows (CR LF)
Reemplazados  0 Bytes en Línea      1
----- Línea      2 -----
índice i*borrado del inicio.
Fin de Línea tipo Windows (CR LF)
Leídos      31 Bytes en Línea      2
*borrado del inicio.
Fin de Línea tipo Windows (CR LF)
Reemplazados  22 Bytes en Línea      2
----- Línea      3 -----
índice i no es borrado.
Fin de Línea tipo Windows (CR LF)
Leídos      25 Bytes en Línea      3
índice i no es borrado.
Fin de Línea tipo Windows (CR LF)
Reemplazados  25 Bytes en Línea      3
----- Línea      4 -----
*borrado del final.índice i
Fin de Línea tipo Windows (CR LF)
Leídos      30 Bytes en Línea      4
*borrado del final.
Fin de Línea tipo Windows (CR LF)
Reemplazados  21 Bytes en Línea      4
----- Línea      5 -----
Coincidencias Encontradas      3
Coincidencias Reemplazadas      2
==1828==
==1828== HEAP SUMMARY:
==1828==     in use at exit: 0 bytes in 0 blocks
==1828==   total heap usage: 60 allocs, 60 frees, 17,901 bytes allocated
==1828==
==1828== All heap blocks were freed -- no leaks are possible
==1828==
==1828== For lists of detected and suppressed errors, rerun with: -s
==1828== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[diego@pereira Shared]$ ]
```

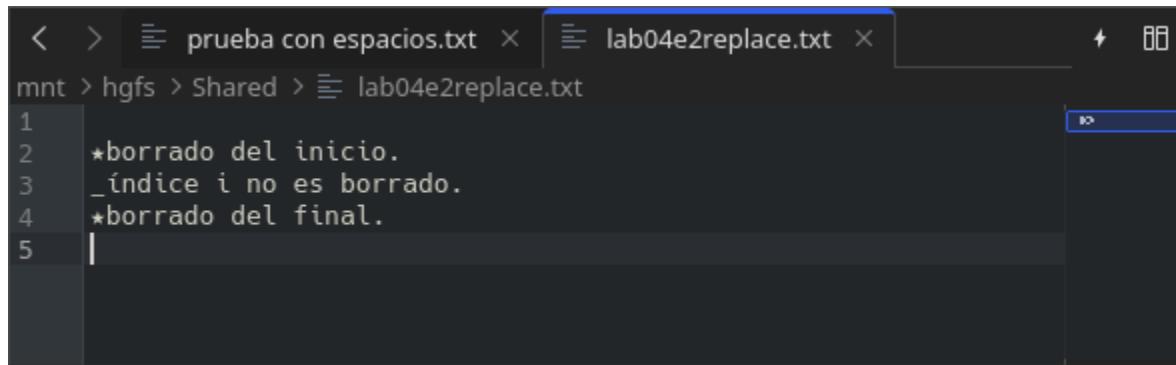
Figura 7. Elaboración propia. Resultado de valgrind y coincidencias reemplazadas, 2025.



A screenshot of a terminal window with two tabs: 'prueba con espacios.txt' and 'lab04e2replace.txt'. The current tab, 'prueba con espacios.txt', shows the following directory path: home > diego > Desktop > prueba con espacios.txt. The content of the file is:

```
1 índice i★borrado del inicio.  
2 _índice i no es borrado.  
3 *borrado del final.índice i  
4  
5 |
```

Figura 8. Elaboración propia. *Muestra del contenido del archivo de prueba autogenerado, 2025.*



A screenshot of a terminal window with two tabs: 'prueba con espacios.txt' and 'lab04e2replace.txt'. The current tab, 'prueba con espacios.txt', shows the following directory path: mnt > hgfs > Shared > lab04e2replace.txt. The content of the file is:

```
1 *borrado del inicio.  
2 _índice i no es borrado.  
3 *borrado del final.  
4  
5 |
```

Figura 9. Elaboración propia. *Muestra del contenido del archivo nuevo luego del reemplazo, 2025.*

4. ANÁLISIS DE RESULTADOS

Para la sección 3.1 acerca del ordenamiento de arreglos y matrices, la Figura 4 muestra el ejemplo de una matriz ordenada de mayor a menor (descendente), mientras que la Figura 5 muestra el ejemplo de una matriz ordenada de menor a mayor (ascendente). Aunque el informe muestra ambas formas de ordenamiento, el enunciado de este laboratorio solicita 3 ejemplos de orden ascendente, y aunque en este caso sólo se muestra uno para no sobrecargar el documento con imágenes similares, al ejecutar el programa sí se imprimen en pantalla las 3 matrices solicitadas.

Para la sección 3.2 acerca del reemplazo de palabras de un archivo, la Figura 6 muestra el ejemplo de un archivo autogenerado por el programa en caso de no poder leer el indicado por el usuario. La Figura 7 muestra línea por línea del archivo leído, el resultado obtenido en caso de haber encontrado o no coincidencias, además de la salida que produce el comando “*valgrind*” con estadísticas acerca del uso de memoria del programa, mostrando que toda la memoria utilizada es liberada correctamente.

Adicional para la sección 3.2, la Figura 8 muestra el contenido del archivo de prueba autogenerado para realizar la búsqueda de coincidencias, y finalmente la Figura 9 muestra el contenido del archivo nuevo luego de realizar los reemplazos solicitados.

5. CONCLUSIONES Y RECOMENDACIONES

Se comprueba que el comando “*valgrind*” es una buena herramienta para 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.