

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Estructure "Tuple". Es el tipo de estructura del cual
7  * están hechos los arreglos dinámicos.
8  */
9
10 typedef struct Tuple {
11     int keyN;
12     char *keyA;
13     char *name;
14     int number;
15 }
16 Tuple;
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Estructura "Array" utilizada para la creación de un arreglo dinámico.
7  */
8
9 typedef struct Array {
10     int size;
11     int used;
12     struct Tuple* data;
13 }
14 Array;
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Procedimientos auxiliares para la creación, inserción y
7  * eliminación de arreglos dinámicos.
8  */
9
10 #ifndef __ARRAYFUNC_H__
11 #define __ARRAYFUNC_H__
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include <time.h>
17
18 void initArray(Array *array, int initSize);
19 void insertArray(Array *array, struct Tuple* value);
20 void freeArray(Array *array);
21
22 #endif
```

```
1  /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Funciones y estructuras auxiliares que ayudan
7  * al manejo de los registros y claves proporcionados
8  * por el usuario.
9  */
10
11 #ifndef __RECORDS_H__
12 #define __RECORDS_H__
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17 #include <time.h>
18
19 char* cutLine(char *file, int i, int n);
20 Tuple* createTuple(char* key, char* name, int number, char* type);
21 void createRecords(char filename[], struct Array* a, struct Array* b, char*
type);
22 void searchKeys(char filename[], char file_binary[], char file_linear[], struct
Array* a, struct Array* b, char* type);
23
24 #endif
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Funciones auxiliares para la organización lineal
7  * ascendente de arreglos dinámicos.
8  */
9
10 #ifndef __ORDER_H__
11 #define __ORDER_H__
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16
17 int compareN(const void *a, const void *b);
18 int compareA(const void *a, const void *b);
19 void organizePairs(Array *a, char* type);
20
21 #endif
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Implementación de algoritmos de búsqueda linear
7  * y binaria.
8  */
9
10 #ifndef __SEARCHALG_H__
11 #define __SEARCHALG_H__
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include <time.h>
17
18 void SearchAlg(char* key, char* type, Array *a, Array *b, FILE *fp_binary,
19               FILE *fp_linear, char filename_binary[], char filename_linear[]);
20 void BinarySearch(Array* a, Array* b, char* key, char* type, FILE *fp, char
21 filename[]);
22 void LinearSearch(Array *a, char* key, char* type, FILE *fp, char filename[]);
23 #endif
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera Herrera
4  * 16-11233
5  *
6  * Procedimientos auxiliares para la escritura en archivos
7  * de los resultados de los algoritmos de búsqueda.
8  */
9
10 #ifndef __RESULTS_H__
11 #define __RESULTS_H__
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include <time.h>
17
18 void BinaryPrint(Array* a, char* keyA, int keyN, int position, char* type,
19                 double time, int found, FILE* fp, char filename[]);
20
21 void LinearPrint(Array* a, char* keyA, int keyN, char* type, int i,
22                 double time, int found, FILE *fp, char filename[]);
23
24 #endif
```

```
1  /**
2   * CI3825: Sistemas de Operación
3   * Valeria Vera
4   * 16-11233
5   *
6   * - arrayfunc.h: Funciones necesarias para el manejo de arreglos dinámicos.
7  */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <time.h>
13
14 #include "Array.h"
15 #include "Tuple.h"
16
17 #include "arrayfunc.h"
18 #include "records.h"
19 #include "order.h"
20 #include "searchalg.h"
21 #include "results.h"
22
23 /**
24  * Inicializa un arreglo dinámico.
25  *
26  * Entrada:
27  * - array: Apuntador al arreglo dinámico a inicializar.
28  * - initSize: Tamaño inicial del arreglo.
29  */
30 void initArray(Array *array, int initSize) {
31     array->size = initSize;
32     array->used = 0;
33     array->data = malloc(10 * sizeof(Tuple));
34
35     if (!array->data) {
36         printf("Error: No se pudo reservar memoria para el arreglo
37 dinamico.");
38         exit(1);
39     }
40 }
41
42 /**
43  * Inserta un elemento al arreglo dinamico y
44  * reserva más memoria en caso de que este esté
45  * lleno.
46  *
47  * Entrada:
48  * - array: arreglo dinamico.
49  * - value: elemento a insertar.
50  */
51 void insertArray(Array* array, struct Tuple* value) {
52     if (array->size == array->used) {
53         array->size += array->size;
54         array->data = realloc(array->data, array->size * sizeof(Tuple));
55
56         if (!array->data) {
57             printf("Error: No se pudo reservar más memoria para el arreglo
58 dinamico.");
59             exit(1);
60         }
61     }
62 }
```



```
58     }
59 }
60     array->data[array->used++] = *value;
61 }
62
63 /**
64  * Libera espacio de memoria reservado para
65  * el arreglo dinámico.
66  *
67  * Entrada:
68  * - array: Apuntador al arreglo dinámico.
69  */
70 void freeArray(Array* array) {
71     free(array->data);
72     array->data = NULL;
73     array->used = array->size = 0;
74 }
```

```
1  /**
2   * CI3825: Sistemas de Operación
3   * Valeria Vera
4   * 16-11233
5   *
6   * - records.h: Funciones auxiliares para la lectura de datos,
7   *   creación de tuplas y guardado de ellas en arreglos dinámicos.
8  */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <time.h>
14
15 #include "Array.h"
16 #include "Tuple.h"
17
18 #include "arrayfunc.h"
19 #include "records.h"
20 #include "order.h"
21 #include "searchalg.h"
22 #include "results.h"
23
24 void insertArray(Array *array, struct Tuple* value);
25
26 /**
27  * Corta una sección de un string con ayuda de strncpy.
28  *
29  * Entrada:
30  * - line: cadena de caracteres a ser cortada.
31  * - i: posición de la cadena desde donde se
32  *   cortará.
33  * - n: cantidad de caracteres a cortar.
34  *
35  * Salida:
36  *   Apuntador a cadena de caracteres.
37  */
38 char* cutLine(char* line, int i, int n) {
39     char* str = malloc((n + 1) * sizeof(char));
40
41     if (!str) {
42         printf("Error: No se pudo reservar memoria para el arreglo.");
43         exit(1);
44     }
45
46     strncpy(str, &line[i], n);
47     str[n] = 0;
48
49     if (!str) {
50         exit(1);
51     }
52
53     return str;
54 }
55
56 /**
57  * Crea las tuplas que se insertarán en los
58  * arreglos dinámicos.
59  */
```

```
60  * Entrada:
61  * - key: clave de la tupla
62  * - name: nombre de la tupla en caso de poseerla
63  * - number: valor numérico de la tupla, puede ser
64  *         tanto el dato de edad como la posición.
65  * - type: tipo de clave que se está manejando
66  *
67  * Salida: Apuntador a tupla con los datos de entrada.
68  */
69  Tuple* createTuple(char* key, char* name, int number, char* type) {
70      Tuple* newTuple = malloc(sizeof(Tuple));
71
72      if (!newTuple) {
73          printf("Error: No se pudo reservar memoria para la tupla");
74          exit(1);
75      }
76
77      /* Define cómo guardar la tupla de acuerdo
78      al tipo de clave. En caso alfanumérico, el
79      valor de la clave keyN es el mayor acercamiento
80      al valor infinito en C. */
81      if (strcmp(type, "a") == 0) {
82          newTuple->keyA = key;
83          newTuple->keyN = 2147483647;
84          newTuple->name = name;
85          newTuple->number = number;
86      } else if (strcmp(type, "n") == 0) {
87
88          newTuple->keyN = atoi(key);
89          newTuple->keyA = "";
90          newTuple->name = name;
91          newTuple->number = number;
92      }
93
94      return newTuple;
95  }
96
97  /**
98  * Lee las líneas de un archivo, las divide y las
99  * agrega en un arreglo dinámico. A su vez agrega
100  * en otro array dinámico de tuplas las claves y
101  * sus posiciones en el primer arreglo.
102  *
103  * Entrada:
104  * - filename: nombre del archivo a leer.
105  * - a: array dinámico donde se agregan los registros.
106  * - b: array dinámico donde se agregan las claves.
107  *     y sus posiciones.
108  * - type: tipo de clave a usar (numérica o alfanumérica).
109  */
110  void createRecords(char filename[], struct Array* a, struct Array* b, char*
type) {
111      FILE *fp;
112      char linea[29];
113      int i = 0;
114
115      fp = fopen(filename, "r");
116
117      if (fp == NULL) {
118          printf("Error al abrir el archivo de datos.");
```

```
119     exit(1);
120 }
121
122 while(fgets(linea, sizeof(linea) + 1, fp)) {
123     char* key = cutLine(linea, 0, 6);
124     char* name = cutLine(linea, 6, 20);
125     char* number = cutLine(linea, 26, 2);
126
127     Tuple* three = createTuple(key, name, atoi(number), type);
128     Tuple* two = createTuple(key, "", i, type);
129
130     insertArray(a, three);
131     insertArray(b, two);
132
133     i++;
134 }
135
136 organizePairs(b, type);
137
138 fclose(fp);
139 }
140
141 /**
142  * Lee las claves a buscar a partir de dividir
143  * las líneas de un archivo, luego llama a los
144  * algoritmos de búsqueda.
145  *
146  * Entrada:
147  * - dilename: nombre del archivo donde se encuentran las
148  *   claves.
149  * - file_binary: nombre del archivo a crear donde se
150  *   guardarán los resultados de la búsqueda binaria.
151  * - file_linear: nombre del archivo a crear donde se
152  *   guardarán los resultados de la búsqueda secuencial.
153  * - a: array dinámico donde se encuentran los registros.
154  * - b: array dinámico donde se encuentran las claves y
155  *   sus posiciones.
156  * - type: tipo de clave de los registros.
157  */
158 void searchKeys(char filename[], char file_binary[], char file_linear[],
159                 struct Array* a, struct Array* b, char* type) {
160     FILE *fp;
161     FILE *fp_binary;
162     FILE *fp_linear;
163
164     char linea[6];
165     fp = fopen(filename, "r");
166     fp_binary = fopen(file_binary, "w");
167     fp_linear = fopen(file_linear, "w");
168
169     if (fp == NULL) {
170         printf("Error al abrir el archivo de claves.");
171         exit(1);
172     }
173
174     while(fgets(linea, sizeof(linea) + 1, fp)) {
175         char* key = cutLine(linea, 0, 6);
176         if (strcmp(key, "\n") > 0 || strcmp(key, "\n") < 0) {
177             SearchAlg(key, type, a, b, fp_binary, fp_linear, file_binary,
178                       file_linear);
179         }
180     }
181 }
```

```
178         }  
179     }  
180  
181     fclose(fp);  
182 }
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera
4  * 16-11233
5  *
6  * - order.h: Funciones utilizadas para organizar
7  *   ascendentemente el arreglo dinámico de claves y posiciones.
8  */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <time.h>
14
15 #include "Array.h"
16 #include "Tuple.h"
17
18 #include "arrayfunc.h"
19 #include "records.h"
20 #include "order.h"
21 #include "searchalg.h"
22 #include "results.h"
23
24 /** ORDENAMIENTO ACORDE A LA CLAVE**/
25 /**
26  * Función de comparación para qsort en el caso de
27  * clave numérica.
28  *
29  * Entrada:
30  * - Dos apuntadores void que tomarán los valores
31  *   de las claves numéricas a comparar.
32  *
33  * Salida: Entero que indica si la primera
34  *   clave es menor, igual o mayor a la segunda.
35  */
36 int compareN(const void *a, const void *b) {
37     Tuple *ia = (Tuple *)a;
38     Tuple *ib = (Tuple *)b;
39
40     return ia->keyN - ib->keyN;
41 }
42
43 /**
44  * Función de comparación para qsort en el caso de
45  * clave alfanumérica.
46  *
47  * Entrada:
48  * - Dos apuntadores void que tomarán los valores
49  *   de las claves alfanuméricas a comparar.
50  *
51  * Salida: Entero que indica si la primera
52  *   clave es menor, igual o mayor a la
53  *   segunda.
54  */
55 int compareA(const void *a, const void *b) {
56     Tuple *ia = (Tuple *)a;
57     Tuple *ib = (Tuple *)b;
58
59     return strcmp(ia->keyA, ib->keyA);
```

```
60 }
61
62 /**
63  * Función que organiza el arreglo dinámico de claves
64  * y posiciones con ayuda de qsort de acuerdo al valor
65  * de las claves de c/u.
66  *
67  * Entrada:
68  * - a: arreglo dinámico a organizar.
69  * - type: tipo de clave del arreglo a ordenar.
70  */
71 void organizePairs(Array *a, char* type) {
72     if (strcmp(type, "a") == 0) {
73         qsort(a->data, a->used, sizeof(Tuple), compareA);
74     } else if (strcmp(type, "n") == 0) {
75         qsort(a->data, a->used, sizeof(Tuple), compareN);
76     }
77 }
78
```

```
1  /**
2   * CI3825: Sistemas de Operación
3   * Valeria Vera
4   * 16-11233
5   *
6   * - searchalg.h: Algoritmos de búsqueda.
7  */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <time.h>
13
14 #include "Array.h"
15 #include "Tuple.h"
16
17 #include "arrayfunc.h"
18 #include "records.h"
19 #include "order.h"
20 #include "searchalg.h"
21 #include "results.h"
22
23 /**
24  * Función que llama a los algoritmos de búsqueda.
25  *
26  * Entrada:
27  * - key: clave a buscar.
28  * - type: tipo de clave a buscar.
29  * - a: arreglo dinámico donde se implementará la
30  *     búsqueda lineal y en el cual se hallarán las
31  *     posiciones del resultado de la búsqueda binaria.
32  * - b: arreglo dinámico donde se implementará la búsqueda
33  *     binaria.
34  * - fp_binary: apuntador al archivo donde se escribirá el
35  *     resultado de la búsqueda binaria.
36  * - fp_linear: apuntador al archivo donde se escribirá el
37  *     resultado de la búsqueda lineal.
38  * - filename_binary: nombre del archivo donde se escribirá
39  *     el resultado de la búsqueda binaria.
40  * - filename_linear: nombre del archivo donde se escribirá
41  *     el resultado de la búsqueda secuencial.
42 */
43 void SearchAlg(char* key, char* type, Array *a, Array *b,
44               FILE *fp_binary, FILE *fp_linear, char filename_binary[],
45               char filename_linear[]) {
46     BinarySearch(a, b, key, type, fp_binary, filename_binary);
47     LinearSearch(a, key, type, fp_linear, filename_linear);
48 }
49
50 /**
51  * Algoritmo de búsqueda binaria.
52  *
53  * Entrada:
54  * - a: arreglo de valores donde se se verá el
55  *     resultado en caso de ser hallado.
56  * - b: arreglo de claves y sus posiciones en el
57  *     arreglo a.
58  * - key: clave a buscar.
```



```

59  * - type: tipo de clave a buscar.
60  * - fp: apuntador al archivo donde se escribirá
61  *   el resultado de la búsqueda.
62  * - filename: nombre del archivo donde se escribirá
63  *   el resultado de la búsqueda.
64  */
65 void BinarySearch(Array* a, Array* b, char* key, char* type, FILE *fp, char
filename[]) {
66     int i = 0;
67     int j = b->used - 1;
68     int m = 0;
69     int found = 0;
70
71     clock_t t;
72     double time_taken;
73     t = clock();
74
75     while (i <= j && !found) {
76         m = (i + j) / 2;
77         if ((strcmp(type, "a") == 0 && strcmp(b->data[m].keyA, key) == 0) ||
78             (strcmp(type, "n") == 0 && b->data[m].keyN == atoi(key))) {
79             found = 1;
80
81             t = clock() - t;
82             time_taken = ((double)t)/CLOCKS_PER_SEC;
83             BinaryPrint(a, b->data[m].keyA, b->data[m].keyN, b-
>data[m].number, type, time_taken, found, fp, filename);
84         } else if (((strcmp(type, "a") == 0 && strcmp(b->data[m].keyA, key) <
0) ||
85             (strcmp(type, "n") == 0 && b->data[m].keyN < atoi(key)))) {
86             i = m + 1;
87         } else {
88             j = m - 1;
89         }
90     }
91
92     if (!found) {
93         t = clock() - t;
94         time_taken = ((double)t)/CLOCKS_PER_SEC;
95         BinaryPrint(a, key, atoi(key), b->data[m].number, type, time_taken,
found, fp, filename);
96     }
97 }
98
99 /**
100  * Algoritmo de búsqueda lineal.
101  *
102  * Entrada:
103  * - a: arreglo de valores donde se implementará
104  *   la búsqueda.
105  * - key: clave a buscar.
106  * - type: tipo de clave a buscar.
107  * - fp: apuntador al archivo donde se escribirá
108  *   el resultado de la búsqueda.
109  * - filename: nombre del archivo donde se escribirá
110  *   el resultado de la búsqueda.
111  */
112 void LinearSearch(Array *a, char* key, char* type, FILE *fp, char filename[])
{
113     int i = 0;

```

```
114     int found = 0;
115
116     clock_t t;
117     double time_taken;
118     t = clock();
119
120     while (i < a->used && !found) {
121         if ((strcmp(type, "a") == 0 && strcmp(a->data[i].keyA, key) == 0) ||
122             (strcmp(type, "n") == 0 && a->data[i].keyN == atoi(key))) {
123             found = 1;
124             t = clock() - t;
125             time_taken = ((double)t)/CLOCKS_PER_SEC;
126             LinearPrint(a, a->data[i].keyA, a->data[i].keyN, type, i,
time_taken, found, fp, filename);
127         }
128         i++;
129     }
130
131     if (!found) {
132         t = clock() - t;
133         time_taken = ((double)t)/CLOCKS_PER_SEC;
134         LinearPrint(a, key, atoi(key), type, i, time_taken, found, fp,
filename);
135     }
136 }
```

```
1  /**
2   * CI3825: Sistemas de Operación
3   * Valeria Vera
4   * 16-11233
5   *
6   * - results.h: Funciones utilizadas para captar y
7   *   guardar los resultados de las búsquedas en un archivo.
8  */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <time.h>
14
15 #include "Array.h"
16 #include "Tuple.h"
17
18 #include "arrayfunc.h"
19 #include "records.h"
20 #include "order.h"
21 #include "searchalg.h"
22 #include "results.h"
23
24 /**
25  * Función que imprime el resultado de la búsqueda binaria.
26  *
27  * Entrada:
28  * - a: arreglo dinámico de valores.
29  * - keyA: clave alfanumérica a buscar en caso de ser claves
30  *   de tipo alfanumérica.
31  * - keyN: clave numérica a buscar en caso de ser claves de
32  *   tipo numérica.
33  * - position: posición en el arreglo a donde se encuentra
34  *   la clave buscada.
35  * - type: tipo de clave.
36  * - time: tiempo en segundos que tomó la búsqueda.
37  * - found: variable que indica si se encontró o no la clave
38  *   buscada.
39  * - fp: apuntador del archivo donde se escribirá el resultado
40  *   de la búsqueda.
41  * - filename: nombre del archivo donde se escribirá el resultado.
42  */
43 void BinaryPrint(Array* a, char* keyA, int keyN, int position, char* type,
44                 double time, int found, FILE* fp, char filename[]) {
45     double ms = time * 1000;
46     fp = fopen(filename, "a+");
47
48     if (found) {
49         if (strcmp(type, "a") == 0) {
50             fprintf(fp, "%s: (%s, %s, %d) %gms\n", keyA, a-
51 >data[position].keyA,
52                 a->data[position].name, a->data[position].number, ms);
53         } else if (strcmp(type, "n") == 0) {
54             fprintf(fp, "%d: (%d, %s, %d) %gms\n", keyN, a-
55 >data[position].keyN,
56                 a->data[position].name, a->data[position].number, ms);
57         }
58     } else if (!found) {
```

```
58     if (strcmp(type, "a") == 0) {
59         fprintf(fp, "%s: (No existe) %gms\n", keyA, ms);
60     } else if (strcmp(type, "n") == 0) {
61         fprintf(fp, "%d: (No existe) %gms\n", keyN, ms);
62     }
63 }
64
65 fclose(fp);
66 }
67
68 /**
69  * Función que imprime el resultado de la búsqueda lineal.
70  *
71  * Entrada:
72  * - a: array donde se implementó la búsqueda.
73  * - keyA: clave alfanumérica a buscar en caso de ser claves
74  *   de tipo alfanumérica.
75  * - keyN: clave numérica a buscar en caso de ser claves de
76  *   tipo numérica.
77  * - type: tipo de clave.
78  * - position: posición en el arreglo a donde se encuentra la
79  *   clave buscada.
80  * - time: tiempo en segundos que tomó la búsqueda.
81  * - found: variable que indica si se encontró o no la clave buscada.
82  * - filename: nombre archivo donde se escribirá el resultado de la búsqueda.
83  */
84 void LinearPrint(Array* a, char* keyA, int keyN, char* type, int i,
85                 double time, int found, FILE* fp, char filename[]) {
86
87     double ms = time * 1000;
88     fp = fopen(filename, "a+");
89
90     if (found) {
91         if (strcmp(type, "a") == 0) {
92             fprintf(fp, "%s: (%s, %s, %d) %gms\n", keyA, a->data[i].keyA,
93                 a->data[i].name, a->data[i].number, ms);
94         } else if (strcmp(type, "n") == 0) {
95             fprintf(fp, "%d: (%d, %s, %d) %gms\n", keyN, a->data[i].keyN,
96                 a->data[i].name, a->data[i].number, ms);
97         }
98     }
99     else if (!found) {
100         if (strcmp(type, "a") == 0) {
101             fprintf(fp, "%s: (No existe) %gms\n", keyA, ms);
102         } else if (strcmp(type, "n") == 0) {
103             fprintf(fp, "%d: (No existe) %gms\n", keyN, ms);
104         }
105     }
106
107     fclose(fp);
108 }
```

```
1 /**
2  * CI3825: Sistemas de Operación
3  * Valeria Vera
4  * 16-11233
5  *
6  * - Array.h: Estructura de arreglo dinámico.
7  *
8  * - Tuple.h: Estructura de tupla.
9  *
10 * - arrayfunc.h: Funciones necesarias para el manejo de arreglos dinámicos.
11 *
12 * - records.h: Funciones auxiliares para la lectura de datos,
13 *   creación de tuplas y guardado de ellas en arreglos dinámicos.
14 *
15 * - order.h: Funciones utilizadas para organizar
16 *   ascendentemente el arreglo dinámico de claves y posiciones.
17 *
18 * - searchalg.h: Algoritmos de búsqueda.
19 *
20 * - results.h: Funciones utilizadas para captar y
21 *   guardar los resultados de las búsquedas en un archivo.
22 */
23
24 #include <stdio.h>
25 #include <stdlib.h>
26 #include <string.h>
27 #include <time.h>
28
29 #include "Array.h"
30 #include "Tuple.h"
31
32 #include "arrayfunc.h"
33 #include "records.h"
34 #include "order.h"
35 #include "searchalg.h"
36 #include "results.h"
37
38 /**
39  * Crea dos arreglos dinámicos, los llena
40  * y comienza la búsqueda de las claves.
41  * Después de todo el proceso, libera el espacio
42  * de memoria utilizado.
43 */
44 int main(int argc, char **argv) {
45     Array a;
46     Array b;
47     char* type = argv[1];
48
49     initArray(&a, 5);
50     initArray(&b, 5);
51
52     createRecords(argv[2], &a, &b, type);
53     searchKeys(argv[3], argv[4], argv[5], &a, &b, type);
54
55     freeArray(&a);
56
57     return 0;
58 }
```