```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Estructure "Tuple". Es el tipo de estructura del cual
* están hechos los arreglos dinámicos.
*/

typedef struct Tuple {
    int keyN;
    char *keyA;
    char *name;
    int number;
}
Tuple;
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Estructura "Array" utilizada para la creación de un arreglo dinámico.
*/

typedef struct Array {
    int size;
    int used;
    struct Tuple* data;
}
Array;
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Procedimientos auxiliares para la creación, inserción y
* eliminación de arreglos dinámicos.
*/

#ifndef __ARRAYFUNC_H__
#define __ARRAYFUNC_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void initArray(Array *array, int initSize);
void insertArray(Array *array, struct Tuple* value);
void freeArray(Array *aray);

#endif
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Funciones y estructuras auxiliares que ayudan
* al manejo de los registros y claves proporcionados
* por el usuario.
*/

#ifndef __RECORDS_H__
#define __RECORDS_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

char* cutLine(char *file, int i, int n);
Tuple* createTuple(char* key, char* name, int number, char* type);
void createRecords(char filename[], struct Array* a, struct Array* b, char* type);
void searchKeys(char filename[], char file_binary[], char file_linear[],struct Array* a, struct Array* b, char* type);

#endif
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Funciones auxiliares para la organización lineal
* ascendente de arreglos dinámicos.
*/

#ifndef __ORDER_H__
#define __ORDER_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int compareN(const void *a, const void *b);
int compareA(const void *a, const void *b);
void organizePairs(Array *a, char* type);

#endif
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Implementación de algoritmos de búsqueda linear
* y binaria.
*/

#ifndef __SEARCHALG_H__
#define __SEARCHALG_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void SearchAlg(char* key, char* type, Array *a, Array *b, FILE *fp_binary,
            FILE *fp_linear, char filename_binary[], char filename_linear[]);
void BinarySearch(Array* a, Array* b, char* key, char* type, FILE *fp, char
filename[]);
void LinearSearch(Array *a, char* key, char* type, FILE *fp, char filename[]);

#endif
```

```c
/**
* CI3825: Sistemas de Operación
* Valeria Vera Herrera
* 16-11233
*
* Procedimientos auxiliares para la escritura en archivos
* de los resultados de los algoritmos de búsqueda.
*/

#ifndef __RESULTS_H__
#define __RESULTS_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void BinaryPrint(Array* a, char* keyA, int keyN, int position, char* type,
                 double time, int found, FILE* fp, char filename[]);

void LinearPrint(Array* a, char* keyA, int keyN, char* type, int i,
                 double time, int found, FILE *fp, char filename[]);

#endif
```

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - arrayfunc.h: Funciones necesarias para el manejo de arreglos dinámicos.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

/**
 * Inicializa un arreglo dinámico.
 *
 * Entrada:
 *  - array: Apuntador al arreglo dinámico a inicializar.
 *  - initSize: Tamaño inicial del arreglo.
*/
void initArray(Array *array, int initSize) {
    array->size = initSize;
    array->used = 0;
    array->data = malloc(10 * sizeof(Tuple));

    if (!array->data) {
        printf("Error: No se pudo reservar memoria para el arreglo dinamico.");
        exit(1);
    }
}

/**
 * Inserta un elemento al arreglo dinamico y
 * reserva más memoria en caso de que este esté
 * lleno.
 *
 * Entrada:
 * - array: arreglo dinamico.
 * - value: elemento a insertar.
 */
void insertArray(Array* array, struct Tuple* value) {
    if (array->size == array->used) {
        array->size += array->size;
        array->data = realloc(array->data, array->size * sizeof(Tuple));

        if (!array->data) {
            printf("Error: No se pudo reservar más memoria para el arreglo dinamico.");
            exit(1);
```

```
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  }
```

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - records.h: Funciones auxiliares para la lectura de datos,
 *   creación de tuplas y guardado de ellas en arreglos dinámicos.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

void insertArray(Array *array, struct Tuple* value);

/**
 * Corta una sección de un string con ayuda de strncpy.
 *
 * Entrada:
 * - line: cadena de caracteres a ser cortada.
 * - i: posición de la cadena desde donde se
 *      cortará.
 * - n: cantidad de caracteres a cortar.
 *
 * Salida:
 *      Apuntador a cadena de caracteres.
 */
char* cutLine(char* line, int i, int n) {
    char* str = malloc((n + 1) * sizeof(char));

    if (!str) {
        printf("Error: No se pudo reservar memoria para el arreglo.");
        exit(1);
    }

    strncpy(str, &line[i], n);
    str[n] = 0;

    if (!str) {
        exit(1);
    }

    return str;
}

/**
 * Crea las tuplas que se insertarán en los
 * arreglos dinámicos.
 *
```

```
 * Entrada:
 * - key: clave de la tupla
 * - name: nombre de la tupla en caso de poseerla
 * - number: valor numérico de la tupla, puede ser
 *          tanto el dato de edad como la posición.
 * - type: tipo de clave que se está manejando
 *
 * Salida: Apuntador a tupla con los datos de entrada.
*/
Tuple* createTuple(char* key, char* name, int number, char* type) {
    Tuple* newTuple = malloc(sizeof(Tuple));

    if (!newTuple) {
        printf("Error: No se pudo reservar memoria para la tupla");
        exit(1);
    }

    /* Define cómo guardar la tupla de acuerdo
    al tipo de clave. En caso alfanumérico, el
    valor de la clave keyN es el mayor acercamiento
    al valor infinito en C. */
    if (strcmp(type, "a") == 0) {
        newTuple->keyA = key;
        newTuple->keyN = 2147483647;
        newTuple->name = name;
        newTuple->number = number;
    } else if (strcmp(type, "n") == 0) {

        newTuple->keyN = atoi(key);
        newTuple->keyA = "";
        newTuple->name = name;
        newTuple->number = number;
    }

    return newTuple;
}

/**
 * Lee las líneas de un archivo, las divide y las
 * agrega en un arreglo dinámico. A su vez agrega
 * en otro array dinámico de tuplas las claves y
 * sus posiciones en el primer arreglo.
 *
 * Entrada:
 * - filename: nombre del archivo a leer.
 * - a: array dinámico donde se agregan los registros.
 * - b: array dinámico donde se agregan las claves.
 *      y sus posiciones.
 * - type: tipo de clave a usar (numérica o alfanumérica).
*/
void createRecords(char filename[], struct Array* a, struct Array* b, char*
type) {
    FILE *fp;
    char linea[29];
    int i = 0;

    fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("Error al abrir el archivo de datos.");
```

```
        exit(1);
    }

    while(fgets(linea, sizeof(linea) + 1, fp)) {
        char* key = cutLine(linea, 0, 6);
        char* name = cutLine(linea, 6, 20);
        char* number = cutLine(linea, 26, 2);

        Tuple* three = createTuple(key, name, atoi(number), type);
        Tuple* two = createTuple(key, "", i, type);

        insertArray(a, three);
        insertArray(b, two);

        i++;
    }

    organizePairs(b, type);

    fclose(fp);
}

/**
 * Lee las claves a buscar a partir de dividir
 * las líneas de un archivo, luego llama a los
 * algoritmos de búsqueda.
 *
 * Entrada:
 * - dilename: nombre del archivo donde se encuentran las
    claves.
 * - file_binary: nombre del archivo a crear donde se
 *  guardarán los resultados de la búsqueda binaria.
* - file_linear: nombre del archivo a crear donde se
 *  guardarán los resultados de la búsqueda secuencial.
 * - a: array dinámico donde se encuentran los registros.
 * - b: array dinámico donde se encuentran las claves y
 *      sus posiciones.
 * - type: tipo de clave de los registros.
*/
void searchKeys(char filename[], char file_binary[], char file_linear[],
                struct Array* a, struct Array* b, char* type) {
    FILE *fp;
    FILE *fp_binary;
    FILE *fp_linear;

    char linea[6];
    fp = fopen(filename, "r");
    fp_binary = fopen(file_binary, "w");
    fp_linear = fopen(file_linear, "w");

    if (fp == NULL) {
        printf("Error al abrir el archivo de claves.");
        exit(1);
    }

    while(fgets(linea, sizeof(linea) + 1, fp)) {
        char* key = cutLine(linea, 0, 6);
        if (strcmp(key, "\n") > 0 || strcmp(key, "\n") < 0) {
            SearchAlg(key, type, a, b, fp_binary, fp_linear, file_binary,
file_linear);
```

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

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - order.h: Funciones utilizadas para organizar
 *   ascendentemente el arreglo dinámico de claves y posiciones.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

/** ORDENAMIENTO ACORDE A LA CLAVE**/
/**
 * Función de comparación para qsort en el caso de
 * clave numérica.
 *
 * Entrada:
 *  - Dos apuntadores void que tomarán los valores
 *    de las claves numéricas a comparar.
 *
 * Salida: Entero que indica si la primera
 *  clave es menor, igual o mayor a la segunda.
*/
int compareN(const void *a, const void *b) {
    Tuple *ia = (Tuple *)a;
    Tuple *ib = (Tuple *)b;

    return ia->keyN - ib->keyN;
}

/**
 * Función de comparación para qsort en el caso de
 * clave alfanumérica.
 *
 * Entrada:
 * - Dos apuntadores void que tomarán los valores
 *   de las claves alfanuméricas a comparar.
 *
 * Salida: Entero que indica si la primera
 *  clave es menor, igual o mayor a la
 *  segunda.
*/
int compareA(const void *a, const void *b) {
    Tuple *ia = (Tuple *)a;
    Tuple *ib = (Tuple *)b;

    return strcmp(ia->keyA, ib->keyA);
```

```c
}

/**
 * Función que organiza el arreglo dinámico de claves
 * y posiciones con ayuda de qsort de acuerdo al valor
 * de las claves de c/u.
 *
 * Entrada:
 * - a: arreglo dinámico a organizar.
 * - type: tipo de clave del arreglo a ordenar.
*/
void organizePairs(Array *a, char* type) {
    if (strcmp(type, "a") == 0) {
        qsort(a->data, a->used, sizeof(Tuple), compareA);
    } else if (strcmp(type, "n") == 0) {
        qsort(a->data, a->used, sizeof(Tuple), compareN);
    }
}

```

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - searchalg.h: Algoritmos de búsqueda.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

/**
 * Función que llama a los algoritmos de búsqueda.
 *
 * Entrada:
 * - key: clave a buscar.
 * - type: tipo de clave a buscar.
 * - a: arreglo dinámico donde se implementará la
 *      búsqueda lineal y en el cual se hallarán las
 *      posiciones del resultado de la búsqueda binaria.
 * - b: arreglo dinámico donde se implementará la búsqueda
 *      binaria.
 * - fp_binary: apuntador al archivo donde se escribirá el
 *  resultado de la búsqueda binaria.
 * - fp_linear: apuntador al archivo donde se escribirá el
 *   resultado de la búsqueda lineal.
 * - filename_binary: nombre del archivo donde se escribirá
 *   el resultado de la búsqueda binaria.
 * - filename_linear: nombre del archivo donde se escribirá
 *   el resultado de la búsqueda secuencial.
*/
void SearchAlg(char* key, char* type, Array *a, Array *b,
                FILE *fp_binary, FILE *fp_linear, char filename_binary[],
char filename_linear[]) {

    BinarySearch(a, b, key, type, fp_binary, filename_binary);
    LinearSearch(a, key, type, fp_linear, filename_linear);
}

/**
 * Algoritmo de búsqueda binaria.
 *
 * Entrada:
 * - a: arreglo de valores donde se se verá el
 *      resultado en caso de ser hallado.
 * - b: arreglo de claves y sus posiciones en el
 *      arreglo a.
 * - key: clave a buscar.
```

```
 * - type: tipo de clave a buscar.
 * - fp: apuntador al archivo donde se escribirá
 *   el resultado de la búsqueda.
 * - filename: nombre del archivo donde se escribirá
 *   el resultado de la búsqueda.
*/
void BinarySearch(Array* a, Array* b, char* key, char* type, FILE *fp, char filename[]) {
    int i = 0;
    int j = b->used - 1;
    int m = 0;
    int found = 0;

    clock_t t;
    double time_taken;
    t = clock();

    while (i <= j && !found) {
        m = (i + j) / 2;
        if ((strcmp(type, "a") == 0 && strcmp(b->data[m].keyA, key) == 0) ||
            (strcmp(type, "n") == 0 && b->data[m].keyN == atoi(key))) {
            found = 1;

            t = clock() - t;
            time_taken = ((double)t)/CLOCKS_PER_SEC;
            BinaryPrint(a, b->data[m].keyA, b->data[m].keyN, b->data[m].number, type, time_taken, found, fp, filename);
        } else if (((strcmp(type, "a") == 0 && strcmp(b->data[m].keyA, key) < 0) ||
            (strcmp(type, "n") == 0 && b->data[m].keyN < atoi(key)))) {
            i = m + 1;
        } else {
            j = m - 1;
        }
    }

    if (!found) {
        t = clock() - t;
        time_taken = ((double)t)/CLOCKS_PER_SEC;
        BinaryPrint(a, key, atoi(key), b->data[m].number, type, time_taken, found, fp, filename);
    }
}

/**
 * Algoritmo de búsqueda lineal.
 *
 * Entrada:
 * - a: arreglo de valores donde se implementará
 *      la búsqueda.
 * - key: clave a buscar.
 * - type: tipo de clave a buscar.
 * - fp: apuntador al archivo donde se escribirá
 *   el resultado de la búsqueda.
 * - filename: nombre del archivo donde se escribirá
 *   el resultado de la búsqueda.
*/
void LinearSearch(Array *a, char* key, char* type, FILE *fp, char filename[])
{
    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 }
```

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - results.h: Funciones utilizadas para captar y
 *    guardar los resultados de las búsquedas en un archivo.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

/**
 * Función que imprime el resultado de la búsqueda binaria.
 *
 * Entrada:
 * - a: arreglo dinámico de valores.
 * - keyA: clave alfanumérica a buscar en caso de ser claves
 *     de tipo alfanumérica.
 * - keyN: clave numérica a buscar en caso de ser claves de
 *     tipo numérica.
 * - position: posición en el arreglo a donde se encuentra
 *     la clave buscada.
 * - type: tipo de clave.
 * - time: tiempo en segundos que tomó la búsqueda.
 * - found: variable que indica si se encontró o no la clave
 *     buscada.
 * - fp: apuntador del archivo donde se escribirá el resultado
 *     de la búsqueda.
 * - filename: nombre del archivo donde se escribirá el resultado.
*/
void BinaryPrint(Array* a, char* keyA, int keyN, int position, char* type,
                 double time, int found, FILE* fp, char filename[]) {

    double ms = time * 1000;
    fp = fopen(filename, "a+");

    if (found) {
        if (strcmp(type, "a") == 0) {
            fprintf(fp, "%s: (%s, %s, %d) %gms\n", keyA, a->data[position].keyA,
                    a->data[position].name, a->data[position].number, ms);
        } else if (strcmp(type, "n") == 0) {
            fprintf(fp, "%d: (%d, %s, %d) %gms\n", keyN, a->data[position].keyN,
                    a->data[position].name, a->data[position].number, ms);
        }
    } else if (!found) {
```

```c
        if (strcmp(type, "a") == 0) {
            fprintf(fp, "%s: (No existe) %gms\n", keyA, ms);
        } else if (strcmp(type, "n") == 0) {
            fprintf(fp, "%d: (No existe) %gms\n", keyN, ms);
        }
    }

    fclose(fp);
}

/**
 * Función que imprime el resultado de la busqueda lineal.
 *
 * Entrada:
 * - a: array donde se implementó la búsqueda.
 * - keyA: clave alfanumérica a buscar en caso de ser claves
 *     de tipo alfanumérica.
 * - keyN: clave numérica a buscar en caso  de ser claves de
 *     tipo numérica.
 * - type: tipo de clave.
 * - position: posición en el arreglo a donde se encuentra la
 *     clave buscada.
 * - time: tiempo en segundos que tomó la búsqueda.
 * - found: variable que indica si se encontró o no la clave buscada.
 * - filename: nombre archivo donde se escribirá el resultado de la búsqueda.
*/
void LinearPrint(Array* a, char* keyA, int keyN, char* type, int i,
                 double time, int found, FILE* fp, char filename[]) {

    double ms = time * 1000;
    fp = fopen(filename, "a+");

    if (found) {
        if (strcmp(type, "a") == 0) {
            fprintf(fp, "%s: (%s, %s, %d) %gms\n", keyA, a->data[i].keyA,
                    a->data[i].name, a->data[i].number, ms);
        } else if (strcmp(type, "n") == 0) {
            fprintf(fp, "%d: (%d, %s, %d) %gms\n", keyN, a->data[i].keyN,
                    a->data[i].name, a->data[i].number, ms);

        }
    } else if (!found) {
        if (strcmp(type, "a") == 0) {
            fprintf(fp, "%s: (No existe) %gms\n", keyA, ms);
        } else if (strcmp(type, "n") == 0) {
            fprintf(fp, "%d: (No existe) %gms\n", keyN, ms);
        }
    }

    fclose(fp);
}
```

```c
/**
 * CI3825: Sistemas de Operación
 * Valeria Vera
 * 16-11233
 *
 * - Array.h: Estructura de arreglo dinámico.
 *
 * - Tuple.h: Estructura de tupla.
 *
 * - arrayfunc.h: Funciones necesarias para el manejo de arreglos dinámicos.
 *
 * - records.h: Funciones auxiliares para la lectura de datos,
 *   creación de tuplas y guardado de ellas en arreglos dinámicos.
 *
 * - order.h: Funciones utilizadas para organizar
 *   ascendentemente el arreglo dinámico de claves y posiciones.
 *
 * - searchalg.h: Algoritmos de búsqueda.
 *
 * - results.h: Funciones utilizadas para captar y
 *   guardar los resultados de las búsquedas en un archivo.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Array.h"
#include "Tuple.h"

#include "arrayfunc.h"
#include "records.h"
#include "order.h"
#include "searchalg.h"
#include "results.h"

/**
 * Crea dos arreglos dinámicos, los llena
 * y comienza la búsqueda de las claves.
 * Después de todo el proceso, libera el espacio
 * de memoria utilizado.
*/
int main(int argc, char **argv) {
    Array a;
    Array b;
    char* type = argv[1];

    initArray(&a, 5);
    initArray(&b, 5);

    createRecords(argv[2], &a, &b, type);
    searchKeys(argv[3], argv[4], argv[5], &a, &b, type);

    freeArray(&a);

    return 0;
}
```