Instituto tecnológico de Costa Rica
Sede central de Cartago
1er semestre, 2020
Prof. Esteban Arias M

Daniel Cornejo Granados
Tarea #3
10/3/2020
Estructuras de Datos

Tarea 3

Punto 7:

a) Aunque el numero de registro sea único la cantidad de nodos máximos que una lista puede tener son tantos como la memoria principal del dispositivo lo permita, tanto en la lista simple como en una lista doble.
b) Se tiene que validar nodo a nodo que los datos no se encuentren ya en la lista.
c) Si se pueden repetir los nodos para las validaciones y diferenciación de nodos se tendría que tomar en cuenta sus direcciones de memoria como factor diferenciante.
d) Se debe de considerar el numero de registro en caso de que deban de ordenar de alguna forma en especial

Explicación del código:

Comenzando con la lista simple en la línea 2 se importó la librería stdlib.h por sugerencia del compilador

Para las consultas de registro se agregó un como parámetro un int llamado id el cual es el número de identificación del nodo, la consulta se hace pidiéndole un numero de registro al usuario y si existe algún nodo que tenga el mismo id este es mostrado al usuario.

Al tener que hacer búsquedas con otro atributo que no fuera el numero de registro se decidió hacerlas con el atributo ‘nombre’, para esto fue utilizada la función strcmp() la cual es parte de la biblioteca string.h la cual fue importada al inicio del programa.

Para eliminar un registro hay que tener en cuenta varios casos, el de eliminar un elemento al inicio, uno en medio de la lista, y otro al final de la lista. Para la eliminación de un nodo en el medio de la lista se tiene que utilizar un puntero temporal que guarde la dirección del nodo a eliminar para después poder liberar la memoria en la cual está alojado.

ListaSimple.c

```c
#include <stdio.h>
#include <stdlib.h> //sugerencia del compilador
#include <string.h>

typedef struct _agenda {

    int id;
    float height;
    char dir[50];
    char nombre[20];
    char telefono[12];
    struct _agenda *siguiente;

}_agenda; //definicion del tipo de dato _agenda

struct _agenda *primero, *ultimo;


void mostrar_menu() {

    printf("\n\nMenú:\n=====\n\n");
    printf("1.- Añadir elementos\n");
    printf("2.- Mostrar lista\n");
    printf("3.- Consultar Registro\n");
    printf("4.- Consultar Nombre\n");
    printf("5.- Eliminar Registro\n");
    printf("6.- Salir\n\n");
    printf("Escoge una opción: ");

    fflush(stdout);
}

/* Con esta función añadimos un elemento al final de la lista */
void anadir_elemento() {

    struct _agenda *nuevo;

    /* reservamos memoria para el nuevo elemento */
    nuevo = ( _agenda *) malloc (sizeof( _agenda));

    if (nuevo==NULL){
        printf( "No hay memoria disponible!\n");
        return; // se le agrega return a la validacion porque sino se cae el programa en caso de no haber memoria
    }

    printf("\nNuevo elemento:\n");

    printf("Altura: ");
    scanf("%f", &nuevo->height);
    while ((getchar()) != '\n');

    printf("Direccion: ");
    fgets(nuevo->dir,50,stdin);
    for (int i = 0; i < 50; i++){
        if(nuevo->dir[i] == '\n'){
            nuevo->dir[i] = '\000';
```

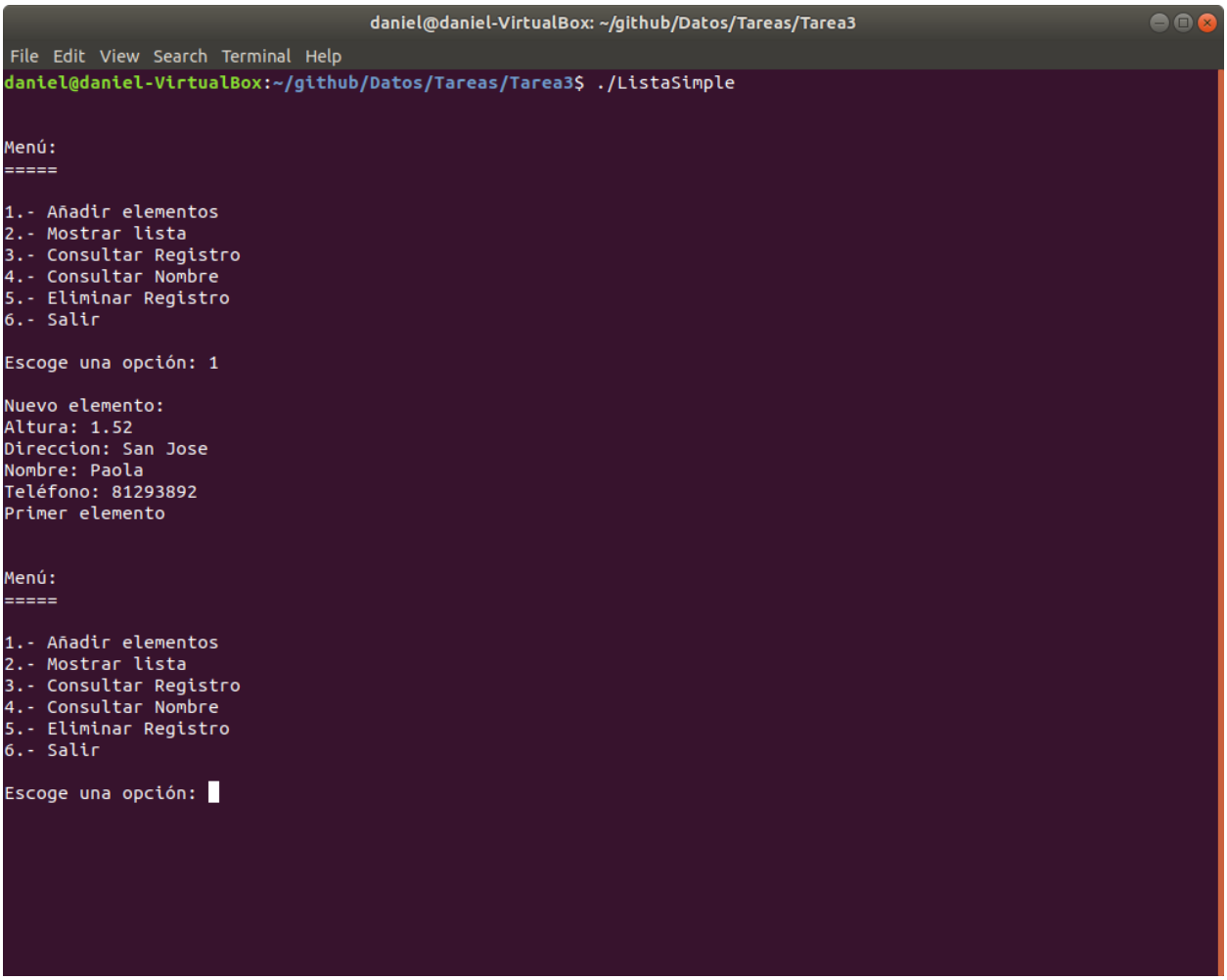
```
        }
    }


    printf("Nombre: ");
    fflush(stdout);
    scanf("%s", nuevo->nombre); // cambio de gets() a fgets() por sugerencia del compilador
    while ((getchar()) != '\n');

    printf("Teléfono: ");
    fflush(stdout);
    scanf("%s", nuevo->telefono); // cambio de gets() a fgets() por sugerencia del compilador
    while ((getchar()) != '\n');

    /* el campo siguiente va a ser NULL por ser el último
    elemento de la lista */
    nuevo->siguiente = NULL;

    /* ahora metemos el nuevo elemento en la lista. lo situamos
    al final de la lista, se comprueba si la lista está vacía.
    si primero==NULL es que no hay ningún elemento en la lista.
    también vale ultimo==NULL */
    if (primero==NULL) {

    printf( "Primer elemento\n");
    nuevo->id = 0;
    primero = nuevo;
    ultimo = nuevo;

    }
    else {

    /* el hasta ahora último apuntará al nuevo */
    nuevo->id = ultimo->id+1;
    ultimo->siguiente = nuevo;

    /* hacemos que el nuevo sea ahora el último */
    ultimo = nuevo;

    }

}
void organizarRegistros(){

            _agenda *aux = primero;
            int registro = 0;

            while(aux!= NULL) {
                aux->id = registro;
                registro++;
                aux = aux->siguiente;
            }
        }

        void EliminarElemento(){

            int registro;
            printf("Ingrese el nummero de registro a eliminar: ");
            scanf("%d", &registro);
            while((getchar()) != '\n');

```

```
118.             _agenda *aux1 = primero;
119.
120.
121.             while(aux1 != NULL){
122.                 if(registro == primero->id && registro == ultimo->id){
123.
124.                     free(primero);
125.                     free(ultimo);
126.                     primero = NULL;
127.                     ultimo = NULL;
128.
129.                 }
130.
131.                 else if (registro == ultimo->id){
132.                     free(ultimo);
133.                     aux1->siguiente = NULL;
134.                     ultimo = aux1;
135.                     break;
136.                 }
137.                 else if(aux1->siguiente->id == registro){
138.                     _agenda *borrar = aux1->siguiente;
139.                     aux1->siguiente = borrar->siguiente;
140.                     free(borrar);
141.                     organizarRegistros();
142.                     break;
143.                 }
144.                 else if(registro == primero->id){
145.                     _agenda *borrar = primero;
146.                     primero = aux1->siguiente;
147.                     free(borrar);
148.                     organizarRegistros();
149.                     break;
150.                 }
151.
152.                 aux1 = aux1->siguiente;
153.
154.             };
155.
156.         }
157.
158.         void mostrar_lista() {
159.
160.             struct _agenda *auxiliar; /* para recorrer la lista */
161.
162.             int i; i=0;
163.             auxiliar = primero;
164.
165.             printf("\nMostrando la lista completa:\n");
166.
167.             while (auxiliar!=NULL) {
168.                 printf("------------------------\n");
169.                 printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
170.                 auxiliar->nombre, auxiliar->telefono, auxiliar->height, auxiliar-
    >dir, auxiliar->id);
171.                 printf("------------------------\n");
172.                 auxiliar = auxiliar->siguiente;
173.
174.                 i++;
175.
176.             }
177.
```

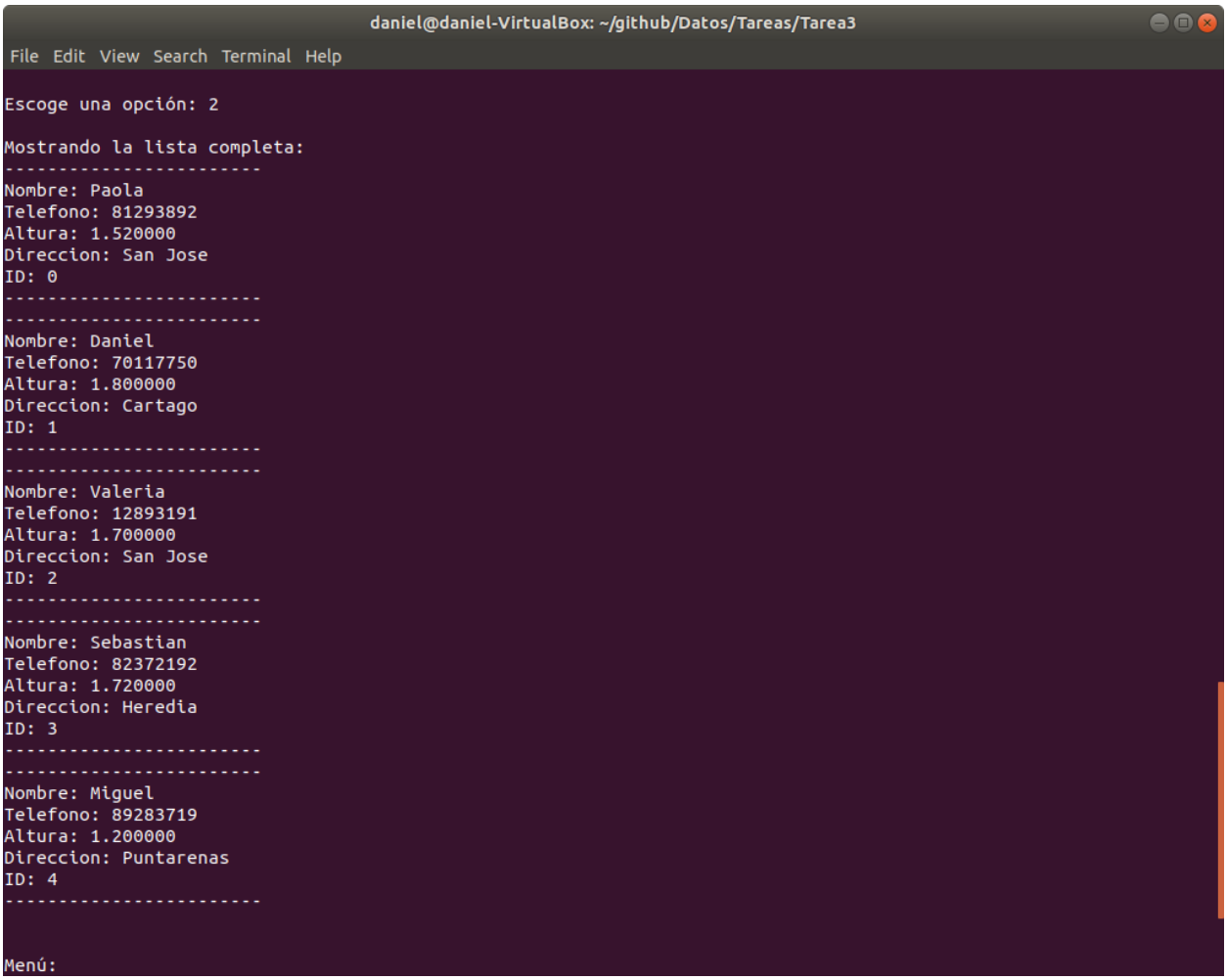
```
178.             if (i==0){
179.                 printf( "\nLa lista está vacía!!\n" );
180.             }
181. 
182.         }
183. 
184.         void ConsultarRegistro(){
185.             int registro;
186.             printf("Ingrese numero de registro a consultar: ");
187.             scanf("%d", &registro);
188.             while ((getchar()) != '\n');
189. 
190.             _agenda *aux;
191.             aux = primero;
192. 
193.             while (aux != NULL){
194.                 if(registro == aux->id){
195.                     printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
196.                             aux->nombre, aux->telefono, aux->height, aux->dir, aux->id);
197.                     break;
198.                 }
199.                 else{
200.                     aux = aux->siguiente;
201.                 }
202.             }
203. 
204. 
205.         }
206. 
207.         void ConsultarNombre(){
208.             char nombre[20] = {};
209.             printf("Ingrese nombre a consultar: ");
210.             scanf("%s", nombre);
211.             while((getchar()) != '\n');
212. 
213.             _agenda *aux;
214.             aux = primero;
215. 
216. 
217. 
218.             while (aux != NULL){
219.                 int comp = strcmp(nombre, aux->nombre);
220.                 if(comp == 0){
221.                     printf("\n");
222.                     printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
223.                             aux->nombre, aux->telefono, aux->height, aux->dir, aux->id);
224.                     break;
225.                 }
226.                 else{
227.                     aux = aux->siguiente;
228.                 }
229.             }
230. 
231.             if(aux == NULL){
232.                 printf("No existe el elemento buscado\n");
233.             }
234.         }
235. 
236.         int main() {
237. 
238.             char opcion;
```

```
239.
240.            primero = (_agenda *) NULL;
241.            ultimo = (_agenda *) NULL;
242.
243.            do {
244.
245.                mostrar_menu();
246.                scanf("%c", &opcion);// cambio de getch() a scanf() para poder limpiar el
    buffer de entrada y eliminar el \n
247.                while ((getchar()) != '\n');// elimina el \n en el buffer de entrada
248.
249.                switch ( opcion ) {
250.
251.                    case '1':
252.                        anadir_elemento(); break;
253.
254.                    case '2':
255.                        mostrar_lista(primero); break;
256.
257.                    case '3':
258.                        ConsultarRegistro(); break;
259.
260.                    case '4':
261.                        ConsultarNombre(); break;
262.
263.                    case '5':
264.                        EliminarElemento(); break;
265.
266.                    case '6':
267.                        exit( 1);
268.
269.                    default:
270.
271.                    printf( "Opción no válida\n" );
272.
273.                    break;
274.
275.                }
276.
277.            } while (opcion!='6');
278.
279.            return 0;
280.        }
```

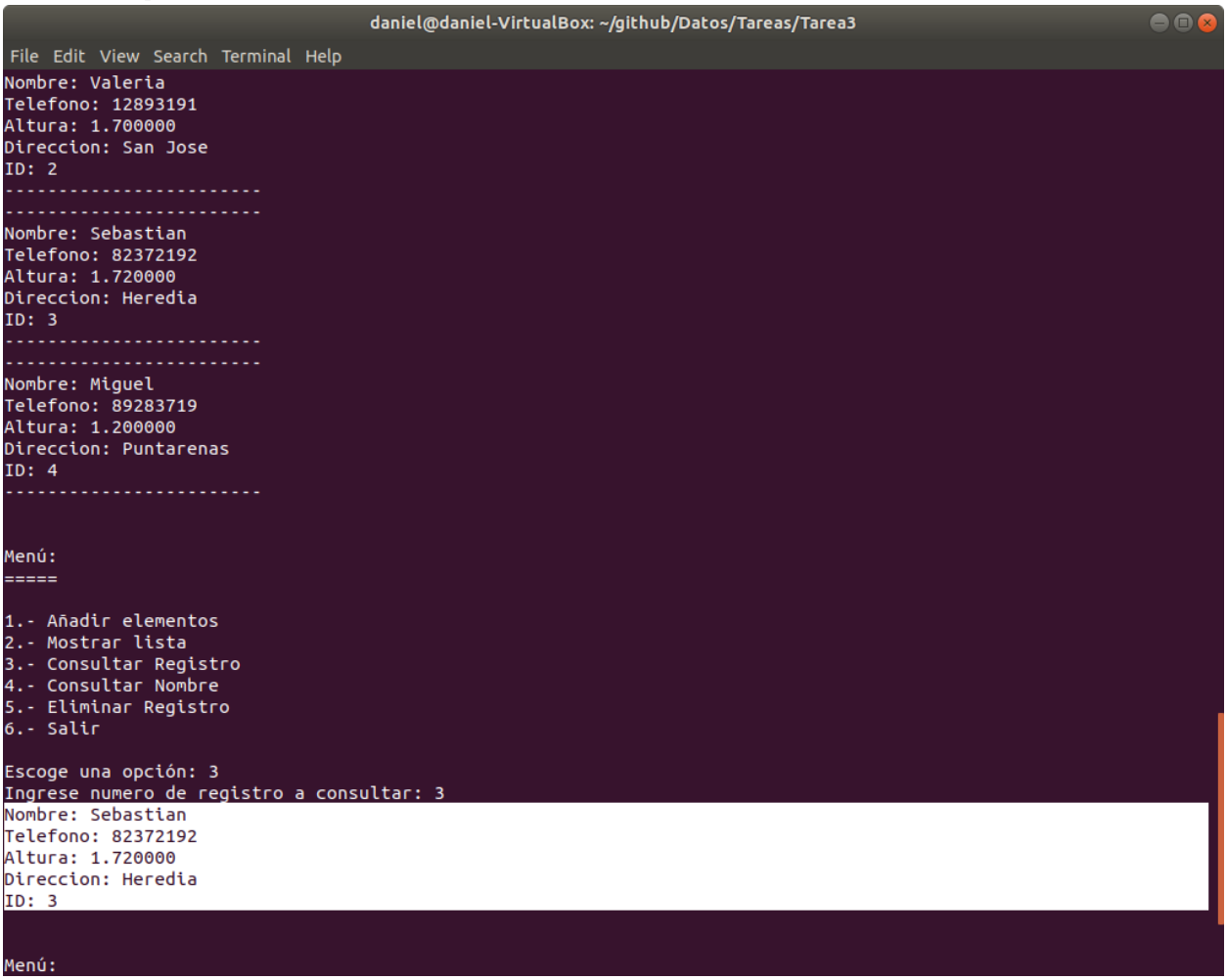
Inserción de un elemento



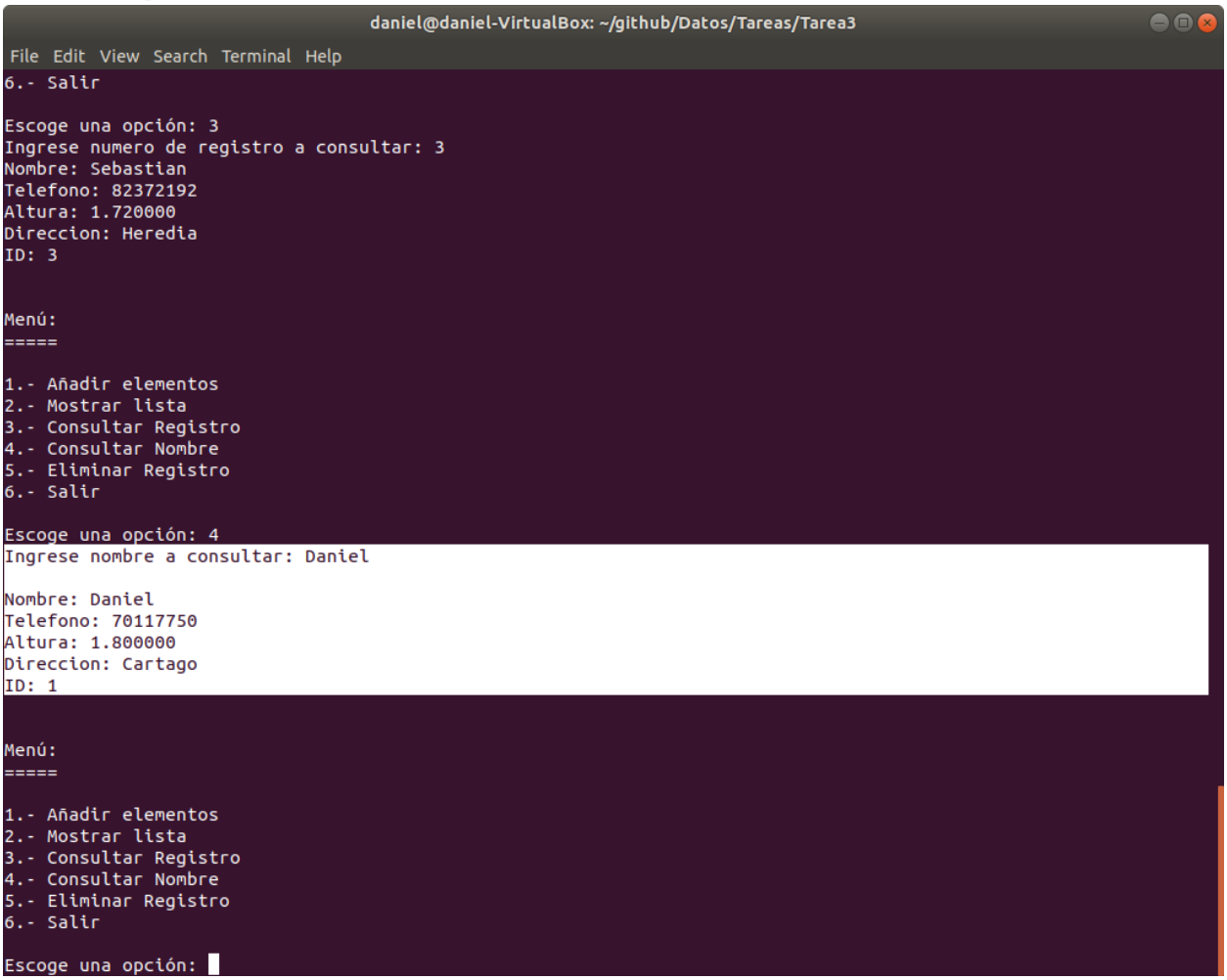
Muestra de la lista



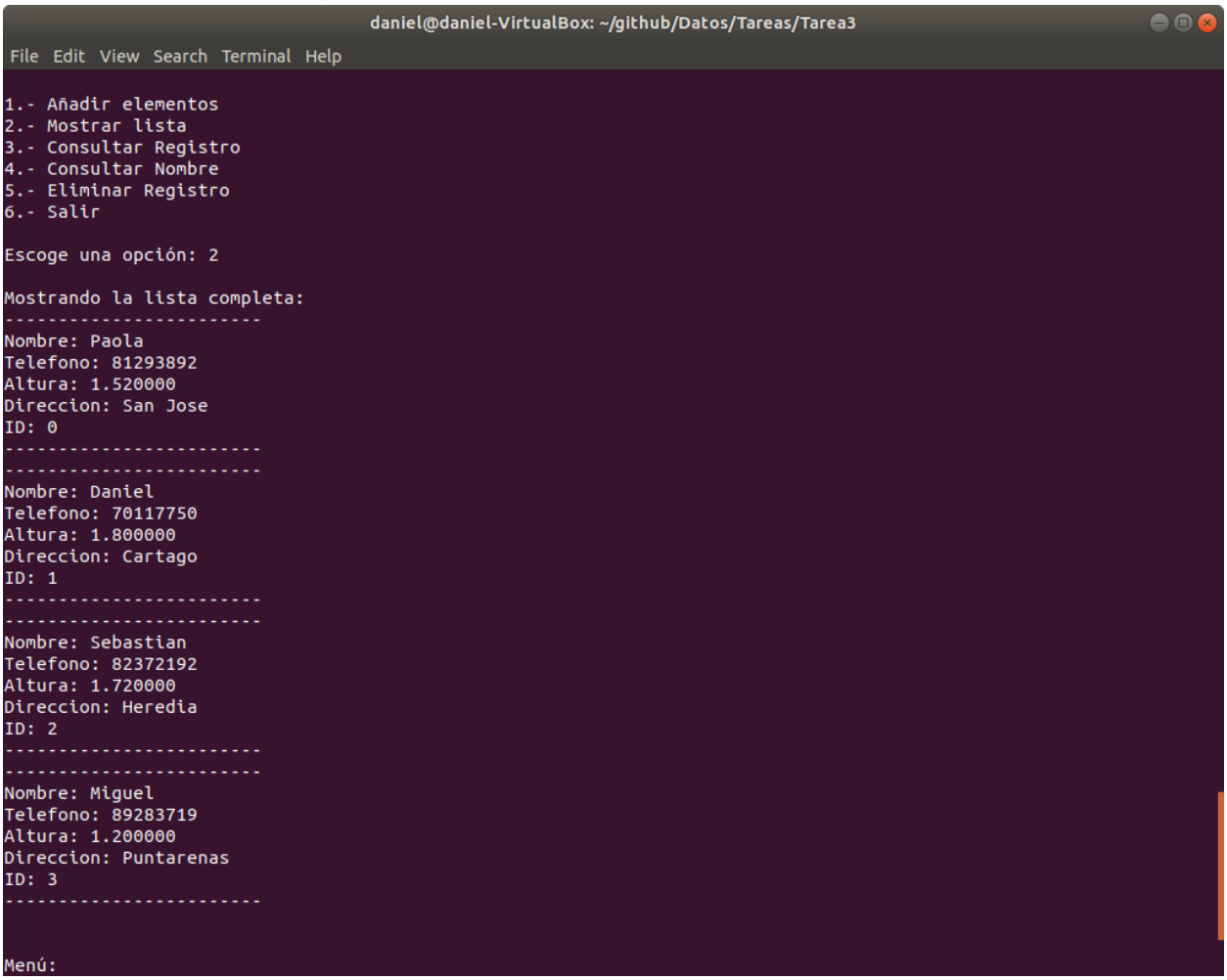
Consulta por indice



Consulta por nombre en vez de inidice



eliminación de un registro



La mayor diferencia que hay en el código de la lista doble con relación al código de la lista simple es que en este al insertar un nuevo nodo se asigna el nodo anterior al puntero de ´anterior´ del nuevo elemento.

Al eliminar un nodo ahora es importante tener una buena idea de la forma de la lista y de la dirección de los punteros puesto que cuando se quiere eliminar un nodo en medio de la lista se debe de tener un buen control sobre los punteros y sus respectivos contenidos.

ListaDoble.c

```c
#include <stdio.h>
#include <stdlib.h> //sugerencia del compilador
#include <string.h>

typedef struct _agenda {

    int id;
    float height;
    char dir[50];
    char nombre[20];
    char telefono[12];
    struct _agenda *siguiente;
    struct _agenda *anterior;

}_agenda; //definicion del tipo de dato _agenda

struct _agenda *primero, *ultimo;


void mostrar_menu() {

    printf("\n\nMenú:\n=====\n\n");
    printf("1.- Añadir elementos\n");
    printf("2.- Mostrar lista\n");
    printf("3.- Consultar Registro\n");
    printf("4.- Consultar Nombre\n");
    printf("5.- Eliminar Registro\n");
    printf("6.- Navegar\n");
    printf("7.- Salir\n\n");
    printf("Escoge una opción: ");

    fflush(stdout);
}

/* Con esta función añadimos un elemento al final de la lista */
void anadir_elemento() {

    struct _agenda *nuevo;

    /* reservamos memoria para el nuevo elemento */
    nuevo = ( _agenda *) malloc (sizeof( _agenda));

    if (nuevo==NULL){
        printf( "No hay memoria disponible!\n");
        return; // se le agrega return a la validacion porque sino se cae el programa en caso de no haber memoria
    }

    printf("\nNuevo elemento:\n");

    printf("Altura: ");
    scanf("%f", &nuevo->height);
    while ((getchar()) != '\n');

    printf("Direccion: ");
    fgets(nuevo->dir,50,stdin);
    for (int i = 0; i < 50; i++){
```

```
57.         if(nuevo->dir[i] == '\n'){
58.             nuevo->dir[i] = '\000';
59.         }
60.     }
61.
62.
63.     printf("Nombre: ");
64.     fflush(stdout);
65.     scanf("%s", nuevo->nombre); // cambio de gets() a fgets() por sugerencia del compilador
66.     while ((getchar()) != '\n');
67.
68.     printf("Teléfono: ");
69.     fflush(stdout);
70.     scanf("%s", nuevo->telefono); // cambio de gets() a fgets() por sugerencia del compilador
71.     while ((getchar()) != '\n');
72.
73.     /* el campo siguiente va a ser NULL por ser el último
74.     elemento de la lista */
75.     nuevo->siguiente = NULL;
76.
77.     /* ahora metemos el nuevo elemento en la lista. lo situamos
78.     al final de la lista, se comprueba si la lista está vacía.
79.     si primero==NULL es que no hay ningún elemento en la lista.
80.     también vale ultimo==NULL */
81.     if (primero==NULL) {
82.
83.         printf( "Primer elemento\n");
84.         nuevo->id = 0;
85.         nuevo->anterior = NULL;
86.
87.         primero = nuevo;
88.         ultimo = nuevo;
89.
90.     }
91.     else {
92.
93.         /* el hasta ahora último apuntará al nuevo */
94.         nuevo->id = ultimo->id+1;
95.
96.         ultimo->siguiente = nuevo;
97.         nuevo->anterior = ultimo;
98.
99.         /* hacemos que el nuevo sea ahora el último */
100.                 ultimo = nuevo;
101.
102.             }
103.
104.         }
105.         void organizarRegistros(){
106.
107.             _agenda *aux = primero;
108.             int registro = 0;
109.
110.             while(aux!= NULL) {
111.                 aux->id = registro;
112.                 registro++;
113.                 aux = aux->siguiente;
114.             }
115.         }
116.
117.         void EliminarElemento(){
```

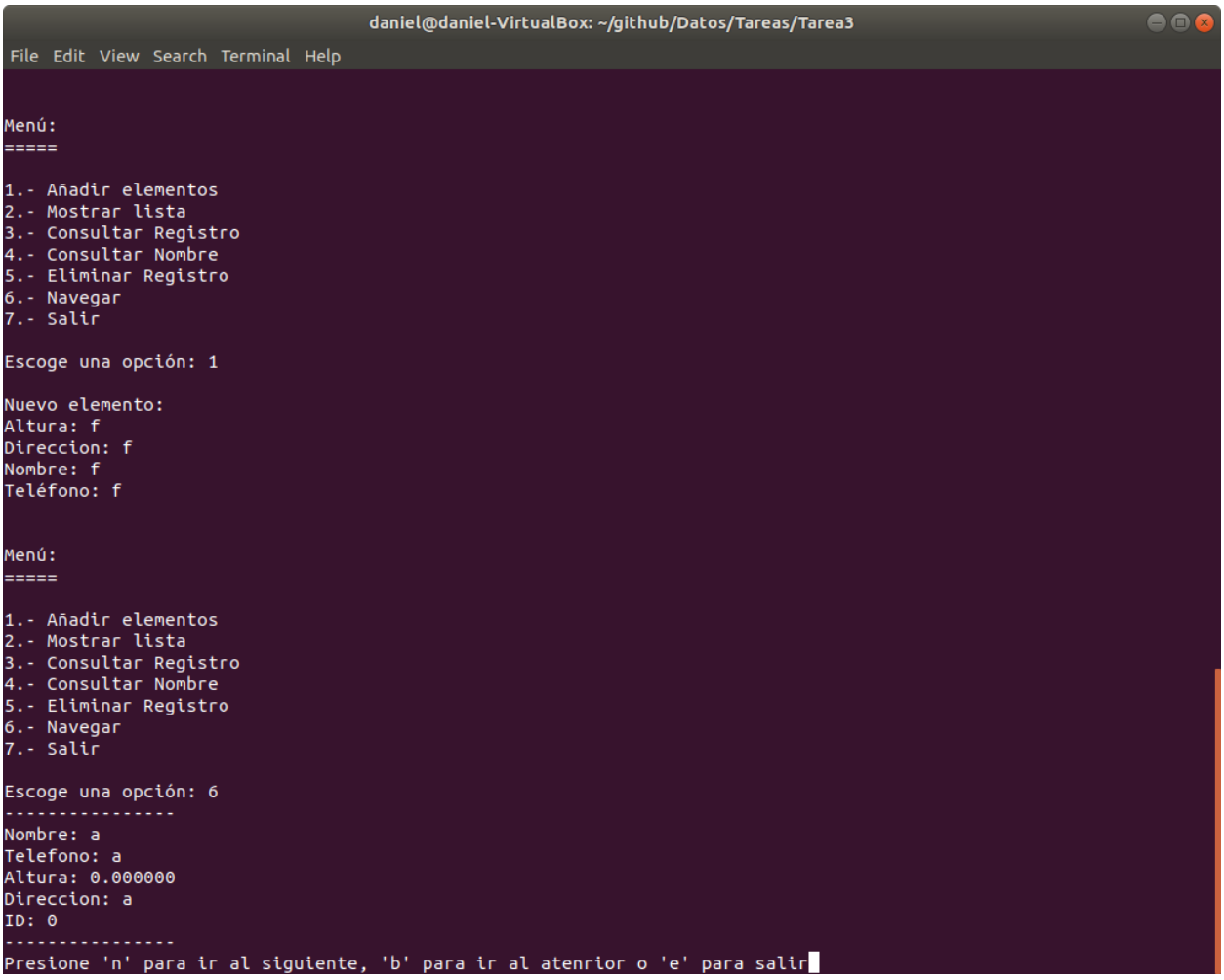
```
118.
119.             int registro;
120.             printf("Ingrese el nummero de registro a eliminar: ");
121.             scanf("%d", &registro);
122.             while((getchar()) != '\n');
123.
124.             _agenda *aux1 = primero;
125.
126.
127.             while(aux1 != NULL){
128.                 if(registro == primero->id && registro == ultimo->id){
129.
130.                     free(primero);
131.                     free(ultimo);
132.                     primero = NULL;
133.                     ultimo = NULL;
134.
135.                 }
136.
137.                 else if (registro == ultimo->id && aux1->siguiente == ultimo){
138.                     free(ultimo);
139.                     aux1->siguiente = NULL;
140.                     ultimo = aux1;
141.                     break;
142.                 }
143.                 else if(aux1->id == registro){
144.
145.                     aux1->anterior->siguiente = aux1->siguiente;
146.                     aux1->siguiente->anterior = aux1->anterior;
147.                     free(aux1);
148.                     organizarRegistros();
149.                     break;
150.                 }
151.                 else if(registro == primero->id){
152.                     _agenda *borrar = primero;
153.                     primero = aux1->siguiente;
154.                     primero->anterior = NULL;
155.                     free(borrar);
156.                     organizarRegistros();
157.                     break;
158.                 }
159.
160.                 aux1 = aux1->siguiente;
161.
162.             };
163.
164.         }
165.
166.         void mostrar_lista() {
167.
168.             struct _agenda *auxiliar; /* para recorrer la lista */
169.
170.             int i; i=0;
171.             auxiliar = primero;
172.
173.             printf("\nMostrando la lista completa:\n");
174.
175.             while (auxiliar!=NULL) {
176.                 printf("------------------------\n");
177.                 printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
```

```
                        auxiliar->nombre, auxiliar->telefono, auxiliar->height, auxiliar->dir, auxiliar->id);
                printf("------------------------\n");
                auxiliar = auxiliar->siguiente;

                i++;

            }

            if (i==0){
                printf( "\nLa lista está vacía!!\n" );
            }

        }

        void ConsultarRegistro(){
            int registro;
            printf("Ingrese numero de registro a consultar: ");
            scanf("%d", &registro);
            while ((getchar()) != '\n');

            _agenda *aux;
            aux = primero;

            while (aux != NULL){
                if(registro == aux->id){
                    printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
                            aux->nombre, aux->telefono, aux->height, aux->dir, aux->id);
                    break;
                }
                else{
                    aux = aux->siguiente;
                }
            }


        }

        void ConsultarNombre(){
            char nombre[20] = {};
            printf("Ingrese nombre a consultar: ");
            scanf("%s", nombre);
            while((getchar()) != '\n');

            _agenda *aux;
            aux = primero;



            while (aux != NULL){
                int comp = strcmp(nombre, aux->nombre);
                if(comp == 0){
                    printf("\n");
                    printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
                            aux->nombre, aux->telefono, aux->height, aux->dir, aux->id);
                    break;
                }
                else{
                    aux = aux->siguiente;
                }
            }
```

```
238.
239.             if(aux == NULL){
240.                 printf("No existe el elemento buscado\n");
241.             }
242.         }
243.
244.
245.         void Navegar(){
246.
247.             _agenda *aux = primero;
248.
249.             while (1){
250.
251.                 printf("----------------\n");
252.                 printf( "Nombre: %s\nTelefono: %s\nAltura: %f\nDireccion: %s\nID: %d\n",
253.                         aux->nombre, aux->telefono, aux->height, aux->dir, aux->id);
254.                 printf("----------------\n");
255.                 printf("Presione 'n' para ir al siguiente, 'b' para ir al atenrior o 'e' para
    salir");
256.
257.
258.                 char dir;
259.                 scanf("%c", &dir);
260.                 while((getchar()) != '\n');
261.
262.                 if (dir == 'n'){
263.                     if( aux->siguiente == NULL){
264.                         printf("No hay mas elementos hacia adelante\n");
265.                     }
266.                     else{
267.                         aux = aux->siguiente;
268.                     }
269.
270.                 }
271.                 else if( dir == 'b'){
272.                     if(aux->anterior == NULL){
273.                         printf("No hay mas elementos hacia atras\n");
274.                     }
275.                     else{
276.                         aux = aux->anterior;
277.                     }
278.                 }
279.                 else if( dir == 'e'){
280.                     break;
281.                 }
282.             }
283.
284.
285.         }
286.
287.
288.         int main() {
289.
290.             char opcion;
291.
292.             primero = (_agenda *) NULL;
293.             ultimo = (_agenda *) NULL;
294.
295.             do {
296.
297.                 mostrar_menu();
```

```
298.                 scanf("%c", &opcion);// cambio de getch() a scanf() para poder limpiar el
    buffer de entrada y eliminar el \n
299.                 while ((getchar()) != '\n');// elimina el \n en el buffer de entrada
300.
301.                 switch ( opcion ) {
302.
303.                     case '1':
304.                         anadir_elemento(); break;
305.
306.                     case '2':
307.                         mostrar_lista(primero); break;
308.
309.                     case '3':
310.                         ConsultarRegistro(); break;
311.
312.                     case '4':
313.                         ConsultarNombre(); break;
314.
315.                     case '5':
316.                         EliminarElemento(); break;
317.
318.                     case '6':
319.                         Navegar(); break;
320.
321.                     case '7':
322.                         exit( 1);
323.
324.                     default:
325.
326.                         printf( "Opción no válida\n" );
327.
328.                         break;
329.
330.                 }
331.
332.             } while (opcion!='6');
333.
334.             return 0;
335.         }
```

Llenado de lista e inserción de elementos



Navegacion completa