Algoritmos y Estructura de Datos

Práctica 4:
LISTAS ENLAZADAS

Grupo: **2CM2**

Profra. Cecilia Alborante Morato

Integrantes:

- Guevara Badillo Areli Alejandra
- Hernández Simón Rebeca

04 de Abril, 2023

INTRODUCCIÓN

Las listas estáticas y las listas enlazadas son dos tipos de estructuras de datos utilizadas en programación para almacenar y manipular colecciones de elementos. Aunque ambos tipos de estructuras pueden contener datos del mismo tipo, cada uno tiene sus propias características y se adapta mejor a diferentes situaciones.

Las listas estáticas, también conocidas como arreglos, son una estructura de datos en la que los elementos se almacenan en una ubicación de memoria contigua y se acceden mediante un índice numérico que representa su posición en la lista. La longitud de la lista estática se establece en el momento de la creación y no se puede cambiar durante la ejecución del programa. A diferencia de las listas enlazadas, las listas estáticas tienen una asignación de memoria fija y un tamaño fijo.

Una de las ventajas de las listas estáticas es que el acceso a los elementos es rápido y eficiente. Al ser una estructura de datos en memoria contigua, los elementos pueden ser accedidos directamente por su índice, lo que significa que la búsqueda y acceso de elementos es muy rápida. Además, las listas estáticas son eficientes en términos de uso de memoria, ya que no necesitan estructuras adicionales de enlace para conectarse entre sí.

Sin embargo, una de las desventajas de las listas estáticas es que su longitud es fija, lo que significa que no se pueden agregar o eliminar elementos de la lista de forma dinámica. En cambio, para agregar o eliminar elementos, se debe crear una nueva lista estática y copiar los elementos de la lista original a la nueva lista. Este proceso puede ser ineficiente y puede llevar mucho tiempo en listas grandes.

Las listas enlazadas, por otro lado, son una estructura de datos que utiliza nodos para almacenar elementos enlazados entre sí. Cada nodo contiene un elemento y una referencia al siguiente nodo en la lista. La lista enlazada puede crecer o disminuir en longitud dinámicamente, lo que significa que es posible agregar o eliminar elementos de la lista en tiempo de ejecución.

Una de las ventajas de las listas enlazadas es que la inserción y eliminación de elementos es eficiente. Si se elimina un elemento, el nodo simplemente se elimina de la lista y se actualiza la referencia del nodo anterior para que apunte al siguiente nodo en la lista. Si se agrega un elemento, se crea un nuevo nodo y se actualiza la referencia del nodo anterior para que apunte al nuevo nodo.

Sin embargo, una de las desventajas de las listas enlazadas es que el acceso a los elementos no es tan rápido como en las listas estáticas, ya que cada nodo en la lista debe ser seguido a través de sus referencias para acceder al elemento. Además, las listas enlazadas pueden requerir más memoria que las listas estáticas, ya que cada nodo debe contener información adicional sobre el enlace a su nodo siguiente.

En resumen, tanto las listas estáticas como las listas enlazadas son estructuras de datos comunes en programación. Las listas estáticas son eficientes en términos de acceso y uso de memoria, pero no son dinámicas en cuanto a su longitud. Las listas enlazadas, por otro lado, son dinámicas y eficientes en términos de inserción y eliminación de elementos, pero pueden requerir más memoria y el acceso a los elementos no es tan rápido como en las listas estáticas. La elección de la estructura de datos depende del uso específico que se le quiera dar, y es importante tener en cuenta las ventajas y desventajas de cada una.

Programa 1: LISTAS ESTÁTICAS

Planteamiento del problema:
Implementar las funciones del manejo de listas estáticas (insertar, buscar, borrar, mostrar y obtener) y realizar un menú para que el usuario decida qué operación quiere realizar.

Se empieza declarando las librerías <stdio.h> que es la librería estándar y <string.h> para poder ocupar las funciones que esta nos brinda como strcmp.

```
1   #include <stdio.h>
2   #include <string.h>
```

Definimos la estructura indicada en la práctica con sus campos y la variable cantidad definida como global que se ocupará para saber la cantidad de personas que hay en la lista.

```
typedef struct {
    char nombre[50];
    char fecha[20];
    char rfc[20];
}Persona;

int cantidad = 0;
```

Se declara la lista estática mediante un arreglo de estructuras de tipo Persona en la función main y la variable opción para el menú.

```
11
12  int main() {
13      Persona lista[100]; //lista
14      int opcion;
15
```

Después, se encuentra el menú de opciones. En cada opción se manda a llamar a las funciones con las que se trabajará. La opción cinco es para salir del programa y default por si el usuario no ingresa un número válido.

```
do{ //menú de opciones
    printf("\n << >> Menu de Opciones << >> \n");
    printf("1) Agregar persona\n");
    printf("2) Buscar persona mediante RFC\n");
    printf("3) Eliminar persona mediante posicion\n");
    printf("4) Mostrar personas\n");
    printf("5) Salir del programa\n");
    printf("Elige una opcion: ");
    scanf("%d", &opcion);

    system("cls");

    switch(opcion) {
        case 1:
            printf("\n1) Agregar persona\n\n");
            insertar(lista); //la función insertar agrega personas a la lista.
            break;
        case 2:
            printf("\n2) Buscar persona mediante RFC\n\n");
            buscar(lista); //la función buscar busca personas en la lista.
            break;
        case 3:
            printf("\n3) Eliminar persona mediante posicion\n\n");
            borrar(lista); //la función eliminar elimina personas de la lista.
            break;
        case 4:
            printf("\n4) Mostrar personas\n\n");
            mostrar(lista); //la función mostrar muestra a las personas que se encuentran dentro de la lista.
            break;

        case 5:
            printf("\n\nGracias por usar nuestro programa!!\n\n");
            exit(1); //sale del programa
            break;

        default: printf("\nOpcion no valida, elija de nuevo!!\n");
                 system("pause");
                 system("cls"); //opcion por si el usuario teclea un numero que no esté dentro del menú.
    }
}while (opcion > 0 || opcion < 6); //el menú se repite siempre y cuando esta condicion se cumpla.
return 0;
```

Esta función tiene como objetivo agregar personas a la lista. Empezamos declarando una nueva persona y limpiamos el buffer del teclado. A la nueva persona que vamos a agregar añadimos su nombre, fecha de nacimiento y RFC. Con la variable global 'cantidad' vamos añadiendo a esa persona a la lista y después la incrementamos. Para finalizar se manda un mensaje diciendo que la personas se ha agregado con éxito a la lista.

```
}

void insertar(Persona lista[]) {
    Persona p; //declaramos la nueva persona.
    fflush(stdin); //limpia el buffer del teclado

    printf("Ingrese el nombre de la persona: ");
    gets(p.nombre); //se le añade nombre a la nueva persona
    printf("Ingrese la fecha de nacimiento de la persona: ");
    scanf("%s", p.fecha); //se le añade fecha de nacimiento a la nueva persona
    printf("Ingrese el RFC de la persona: ");
    scanf("%s", p.rfc); //se le añade rfc a la nueva persona

    lista[cantidad] = p; //sirve para añadirla a la lista
    (cantidad)++; //se aumenta el numero de personas que hay en la lista.

    printf("\nLa persona ha sido agregada a la lista.\n");
    system("pause");
    system("cls");
}
```

La función buscar pide el RFC de la persona para poder hacer una búsqueda y el for que se tiene ahí dice que si el contador es menor que la cantidad de personas que hay en la lista entra al for y compara, con la función "strcmp" que se encuentra en la librería string.h, si el dato que se ingresó es igual al dato que está en la lista y si es igual, lo imprime.

```
int buscar(Persona lista[]) {
    char com[20];
    printf("Ingrese el RFC de la persona: ");
    scanf("%s", com); //se guarda el valor del rfc

    for (int i = 0; i < cantidad; i++) {//si el contador es menor que la cantidad de personas que hay en la
                                        //lista entra al for y compara, con la funcion "strcmp" que se encuentra
                                       //en la libreria string.h, si el dato que ingresaste es igual a un dato
                                      // que está en la lista. Si es igual, lo imprime.
        if (strcmp(lista[i].rfc, com) == 0) {
            printf("\n%d)Nombre:%s\n  Fecha de Nacimiento:%s\n  RFC:%s\t\n", i + 1, lista[i].nombre, lista[i].fecha, lis
        }
    }
    system("pause");
    system("cls");
}
```

La función borrar se encarga de buscar la posición que ingresó el usuario para poder eliminarla y recorrer el arreglo.

```
int borrar(Persona lista[]) {
    int pos;
    printf("Que posicion deseas eliminar: ");
    scanf("%d", &pos); //ingresas la posicion que se desea elimnar y se guarda en la variable pos.

    for (int i = pos - 1 ; i < cantidad; i++) { //pos -1 porque se cuenta desde 0
        lista[i] = lista[i+1]; //se eliminan y se recorren los datos.
    }
    cantidad--; //se disminuye la cantidad de personas (contador)

    printf("\n\nLos datos han sido eliminados exitosamente!!!! :)\n");
    system("pause");
    system("cls");
}
```

Y por último, la función mostrar imprime todas las personas (con sus respectivos datos) que se encuentran en la lista.

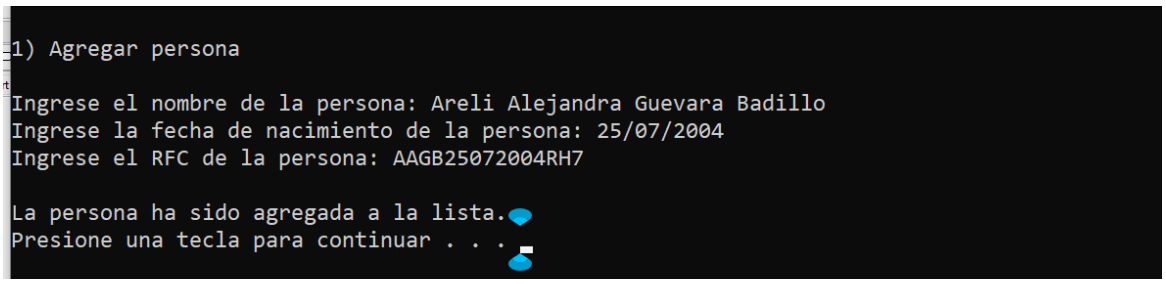
```
void mostrar(Persona lista[]) {
    if (cantidad == 0) printf("No hay personas en la lista.\n");

    else {
        printf("Lista de personas:\n"); //este for recorre la lista e imprime los datos que se encuentran dentro.
        for (int i = 0; i < cantidad; i++) {
            printf("%d)Nombre:%s\n  Fecha de Nacimiento:%s\n  RFC:%s\t\n\n", i + 1, lista[i].nombre, lista[i].fecha, lis
        }
     }
    system("pause");
    system("cls");
}
```

RESULTADOS

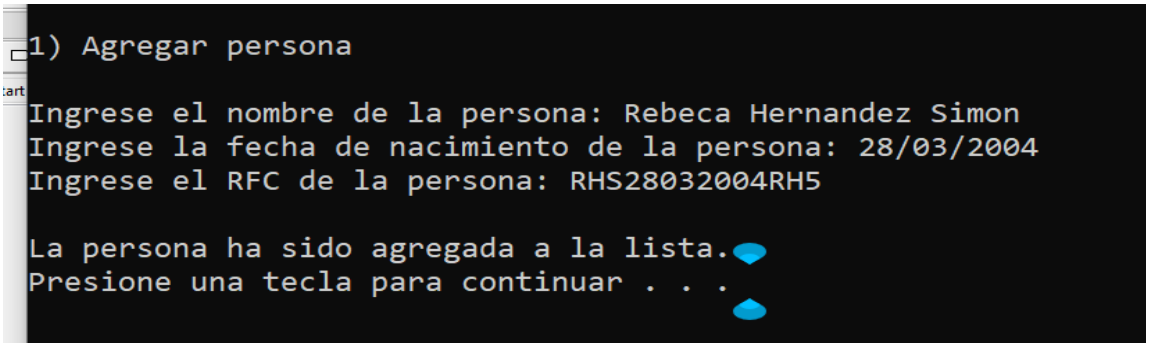
Muestra el menú de opciones en un primer momento.

Opción 1: Se ingresan los datos que pide el programa para que se añadan una o más personas a la lista.

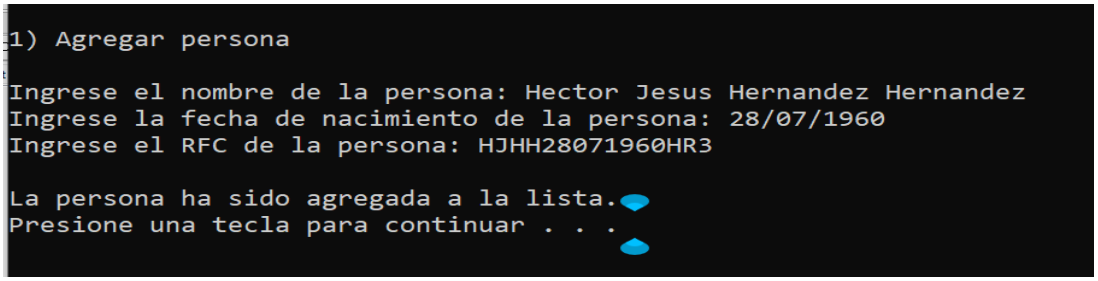
Primera persona:



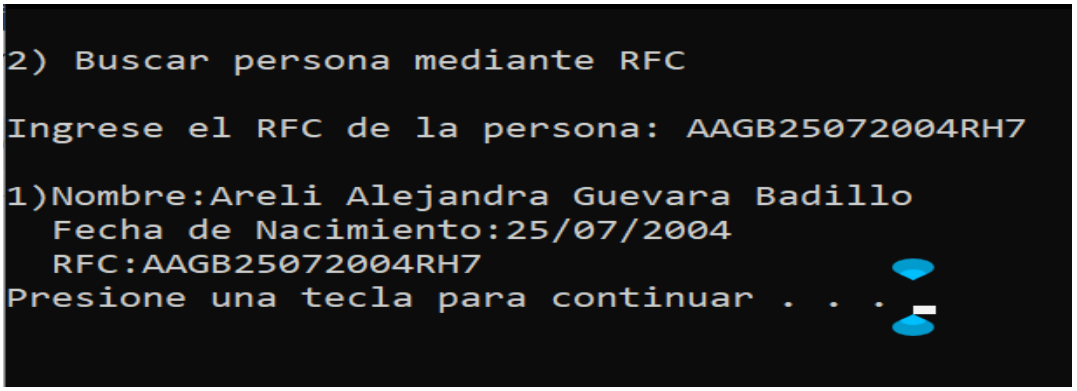
Segunda persona:



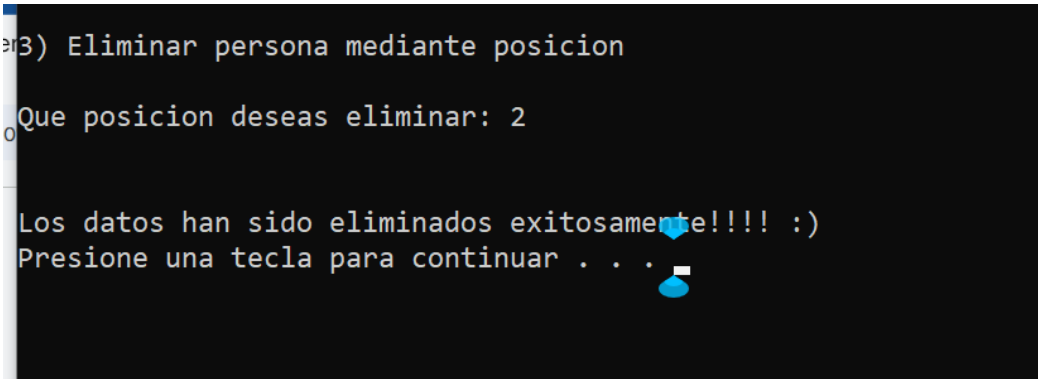
Tercera persona:



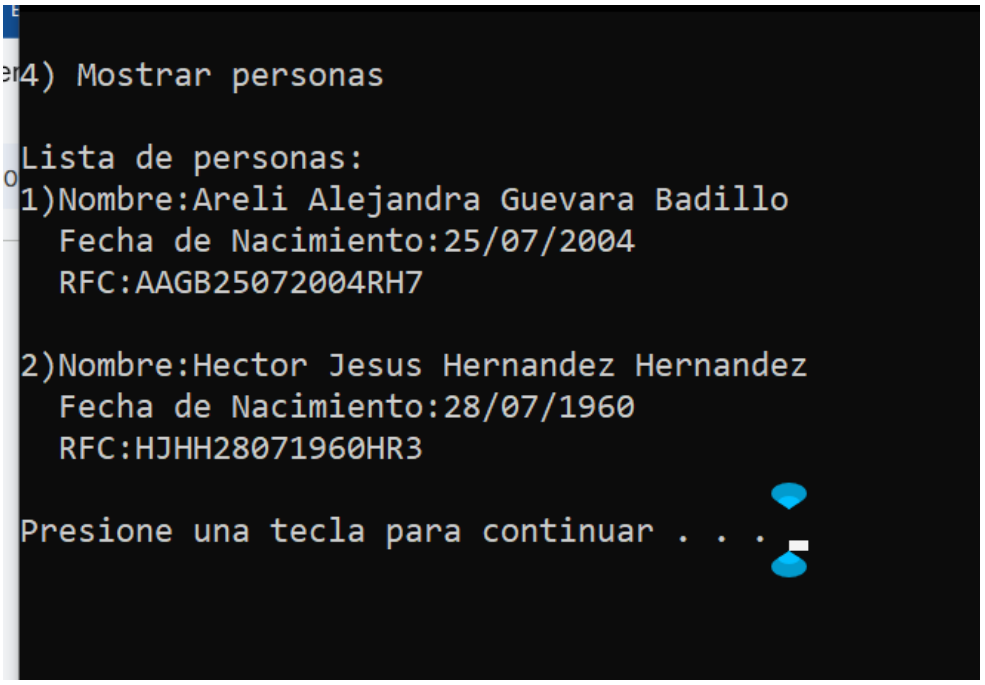
Opción 2: Se ingresa el RFC de la persona que se desea buscar y se imprime si lo encuentra.



Opción 3: Se le pide al usuario la posición que se desea borrar, esta función manda un mensaje diciendo que la persona se ha borrado con éxito.

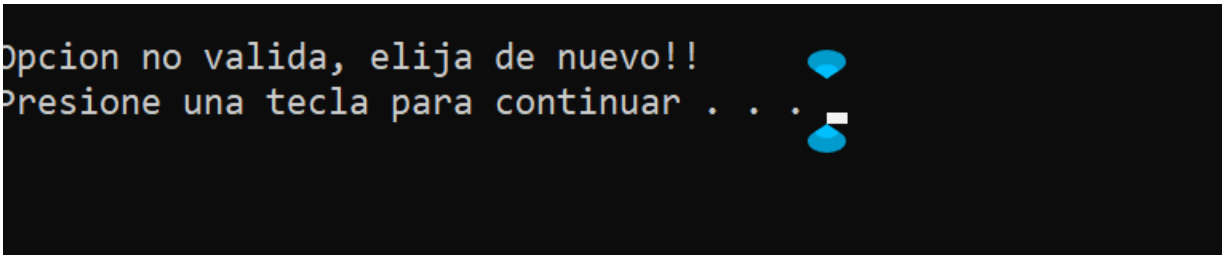


Opción 4: Para poder visualizar a las personas que se encuentran en la lista ocupamos esta función.



Borró a quien se encontraba en la posición 2 que era ‘Rebeca Hernandez Simon’ y los recorrió.

Este mensaje lo arroja el programa por si el usuario se equivocó.



Programa 2: LISTAS DINÁMICAS

Planteamiento del problema:
Permitir al usuario ingresar los valores para dos listas, de las cuales se regresa la cantidad de elementos y el numero de elementos pares que hay en cada lista. Tambien dira que elementos del conjunto dos pertenecen tambien al conjunto uno.

- Se crea la estructura del nodo

```
5   typedef struct nodo{
6       int valor;
7       struct nodo* sig;
8   }nodo;
```

- Se definen las funciones que se necesitaran en el programa

```
10  void insertar(nodo **, int);//inserta nuevos elementos
11  void imprimir(nodo **, int);//imprime la lista
12  int cantidad(nodo **);//devuelve la cantidad de elementos de la lista
13  int pares(nodo **);//devuelve la cantidad de elementos pares de la lista
14  int buscar(nodo **, nodo **);//imprime si esta en la lista uno
15  int coin(nodo **, int);//compara los elementos de la lista 1 con la lista 2
```

- En el *main* se llamara a todas las funciones conforme estas se necesiten, estas enviaran el apuntador al nodo y en algunos casos un valor de tipo entero.

```
17  void main(){
18      printf("Practica 4:\n");
19      printf("Parte 2: Listas dinamicas.\n\n");
20
21      int x;
22      nodo* p1 = NULL;//inicializa lista 1
23      nodo* p2 = NULL;//inicializa lista 2
24
25      printf(">Ingresa los valores para Lista 1:\n");
26      printf("  *0: final de la lista\n");
27      do{
28          scanf("%d",&x);
29          insertar(&p1, x);
30      }while(x!=0);//registra valores hasta que el usuario teclee 0
31
32      printf("\n>Ingresa los valores para Lista 2:\n");
33      printf("  *0: final de la lista\n");
34      do{
35          scanf("%d",&x);
36          insertar(&p2,x);
37      }while(x!=0);//registra valores hasta que el usuario teclee 0
```

```
38
39          printf("\n");
40          system("pause");
41          system("cls");
42
43          printf("\n DATOS DE LAS LISTAS:\n");
44
45          imprimir(&p1, 1);//imprime la lista 1
46          printf("Cantidad de elementos: %d\n",cantidad(&p1));
47          printf("Cantidad de elementos pares: %d\n",pares(&p1));
48
49          imprimir(&p2, 2);//imprime la lista 2
50          printf("Cantidad de elementos: %d\n",cantidad(&p2));
51          printf("Cantidad de elementos pares: %d\n",pares(&p2));
52
53          buscar(&p1, &p2);//llama a buscar
54
55      }
```

- En la funcion *insertar* se insertan al final de cada lista el nuevo elemento que ingrese al usuario.

```
57  void insertar(nodo** p,int x){
58      struct nodo *nuevo = malloc(sizeof(struct nodo));
59      nuevo->valor = x;//iguala el valor del nuevo dato al dato ingresado
60      nuevo->sig = NULL;//al ser el ultimo elemento se inicializa a nulo
61
62      if (*p == NULL)*p = nuevo;// si la lista esta vacia se coloca en p
63      else{// si no esta vacia recorremos la lista hasta llegar al ultimo nodo y lo colocamos ahi
64          nodo* aux = *p;
65
66          while( aux->sig != NULL ){
67              aux = aux->sig;
68          }
69          aux->sig = nuevo;
70      }
71  }
```

- En la funcion *cantidad* se recorre toda la lista hasta llegar al final de la lista (encontrar el valor nulo en reco→sig) y va contando cada elemento.

```
73  int cantidad(nodo** p){
74      struct nodo *reco = *p;
75      int cant = 0;
76
77      while (reco != NULL && reco->valor != 0){//recoremos todos los nodos hasta el final
78          cant++;//contamos el numero de nodos
79          reco = reco->sig;
80      }
81      return cant;//devuelve la cantidad
82  }
```

- La funcion *pares* sigue la misma logica que la funcion *cantidad* pero solo contara los elementos los cuales el modulo con dos sea igual a 0.

```
84  int pares(nodo** p){
85      struct nodo *reco = *p;
86      int par = 0;
87
88      while (reco != NULL && reco->valor != 0){//recoremos todos los nodos hasta el final
89          if(reco->valor%2 == 0)par++;//contamos el numero de valores que sean pares
90          reco = reco->sig;
91      }
92      return par;//devuelve la cantidad de numeros pares
93  }
```

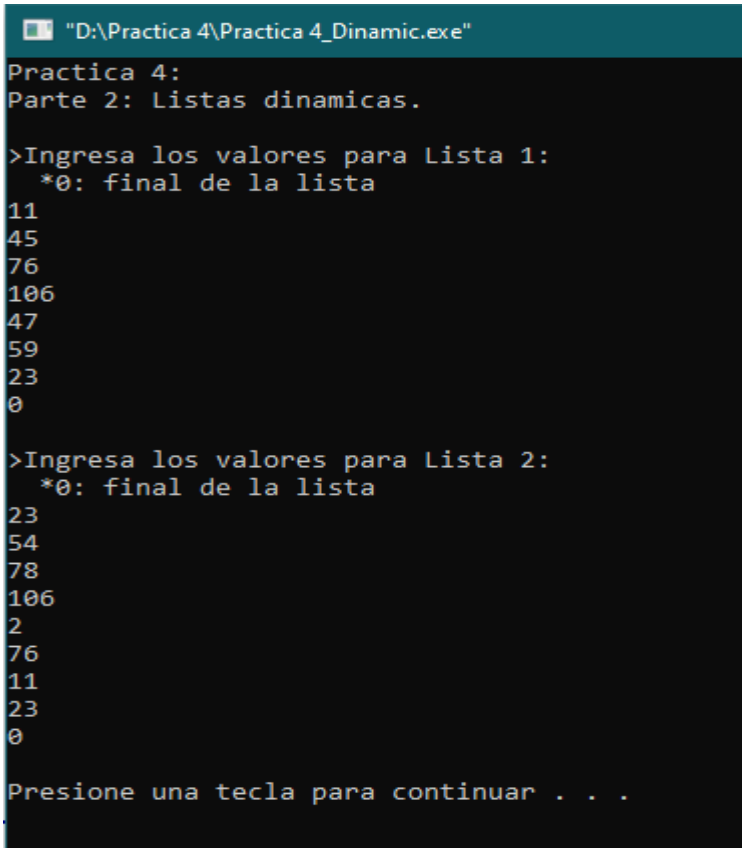
- La funcion *imprimir* recorre la lista e imprime la seccion de valor de todos los nodos hasta terminar la lista.

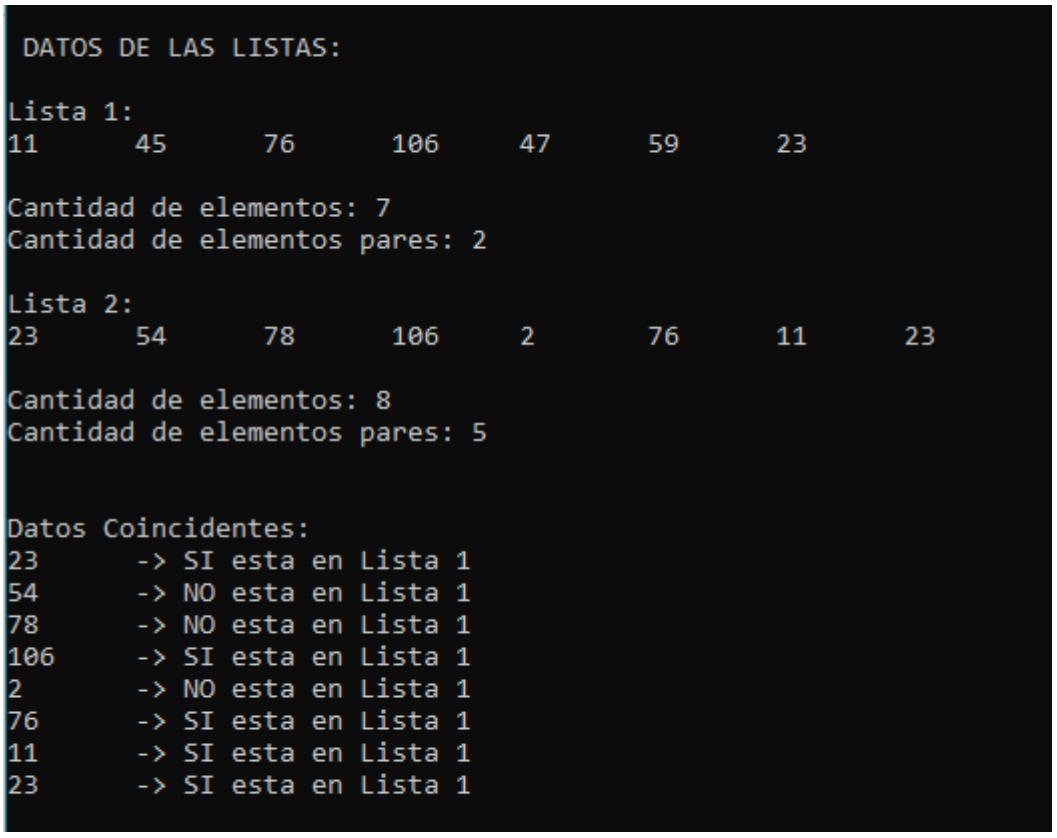
```
95  void imprimir(nodo** p,int n){
96      struct nodo *reco = *p;
97
98      printf("\nLista %d:\n",n);
99
100     while (reco!=NULL && reco->valor != 0){//recorre la lista
101         printf("%d\t",reco->valor);//imprime la seccion de valor del nodo
102         reco = reco->sig;
103     }
104     printf("\n\n");
105 }
```

- La funcion de *buscar* tomara cada uno de los valores de la lista dos y los enviara a la funcion *coin* la cual se encargara de buscar ese valor en la lista 1, si lo encuentra devolvera 1 y si no lo encuentra devolvera 0, y dependiendo de lo que devuelva se imprimira si encontro el valor o no.

```
107 int buscar(nodo **p1, nodo **p2){
108     struct nodo *datos = *p2;
109     struct nodo *p = *p1;
110     int x, e = 0;
111
112     printf("\n\nDatos Coincidentes:\n");
113
114     while (datos!=NULL && datos->valor != 0){
115         x = datos->valor;//guarda el valor que se va a buscar en la lista 1
116         e = coin(&p, x);//devuelve 1 si esta en la lista 1 y 0 si no esta
117
118         if(e == 1)printf("%d\t-> SI esta en Lista 1\n", x);
119         else printf("%d\t-> NO esta en Lista 1\n", x);
120
121         datos = datos->sig;
122     }
123     printf("\n");
124 }
126 int coin(nodo** p, int x){
127     struct nodo* aux = *p;
128     int n = 0;
129
130     while(aux != NULL ){
131         if(aux->valor == x) n = 1;//si encuentra el valor en la lista n vale 1
132         aux = aux->sig;
133     }
134     return n;//devuelve si encontro o no el valor
135 }
```

RESULTADOS


"D:\Practica 4\Practica 4_Dinamic.exe"
Practica 4:
Parte 2: Listas dinamicas.
>Ingresa los valores para Lista 1:
*0: final de la lista
11
45
76
106
47
59
23
0
>Ingresa los valores para Lista 2:
*0: final de la lista
23
54
78
106
2
76
11
23
0
Presione una tecla para continuar . . .


DATOS DE LAS LISTAS:
Lista 1:
11 45 76 106 47 59 23
Cantidad de elementos: 7
Cantidad de elementos pares: 2
Lista 2:
23 54 78 106 2 76 11 23
Cantidad de elementos: 8
Cantidad de elementos pares: 5
Datos Coincidentes:
23 -> SI esta en Lista 1
54 -> NO esta en Lista 1
78 -> NO esta en Lista 1
106 -> SI esta en Lista 1
2 -> NO esta en Lista 1
76 -> SI esta en Lista 1
11 -> SI esta en Lista 1
23 -> SI esta en Lista 1