Resumen

Se debe desarrollar un programa en C++ que realice la funcionalidad de un proceso de simulación de encuestas con las siguientes operaciones:

- Información del padrón: Carga de información de los registros desde archivos de texto del TSE hacia los nodos de las listas enlazadas.
- Colección Votantes con su información relevante (archivo: padrón completo)
- Colección Distrito Electoral (archivo: Distelec)
- Colección Provincias
- Información de las encuestas: Generando una "lista de preguntas" con su respectiva lista de respuestas y pesos(valores)
- Lista de encuestas: Una encuesta está conformada por una serie de preguntas.
- Simulación de encuestas al padrón electoral: usando la información cargada en las listas enlazadas anteriores Pasada total sobre el padrón de CR en forma Random algunos votantes no van a responder del todo.

La lista enlazada simple

- Las listas enlazadas son Arreglos con acceso mediante un puntero.
- La asignación de memoria es hecha durante la ejecución.
- Las listas enlazadas pueden ser utilizadas cuando se necesitan hacer varias operaciones de inserción y eliminación de elementos.

Lista simplemente enlazada

- El puntero **siguiente** del último elemento apunta hacia NULL.
- Para acceder a un elemento, la lista es recorrida comenzando por el inicio, el puntero **Siguiente** permite el cambio hacia el próximo elemento.
- El desplazamiento se hace en una sola dirección, del primer al último elemento. Si se desea desplazar en las dos direcciones (hacia delante y hacia atrás) se tiene que utilizar las listas doblemente enlazadas. Las cuales funcionan a partir de 2 punteros Auxiliares.
- Para establecer un elemento de la lista, será utilizado el tipo *struct*. El elemento de la lista tendrá un campo **dato** y un puntero **siguiente**.
- El puntero **siguiente** tiene que ser del mismo tipo que el elemento, si no, no podrá apuntar hacia el elemento. El puntero **siguiente** permitirá el acceso al próximo elemento.

```
typedef struct ElementoLista {
char *dato;
struct ElementoLista *siguiente;
}Elemento;
```

Operaciones con listas

1. Inicialización

- Se debe inicializar la lista para que funcione el arreglo.
- Esta operación debe estar antes de otra operación sobre la lista.
- Esta comienza el puntero inicio y el puntero fin con el puntero NULL, y el tamaño con el valor 0.

```
void inicializacion (Lista *lista){
lista->inicio = NULL;
lista->fin = NULL;
tamaño = 0;
}
```

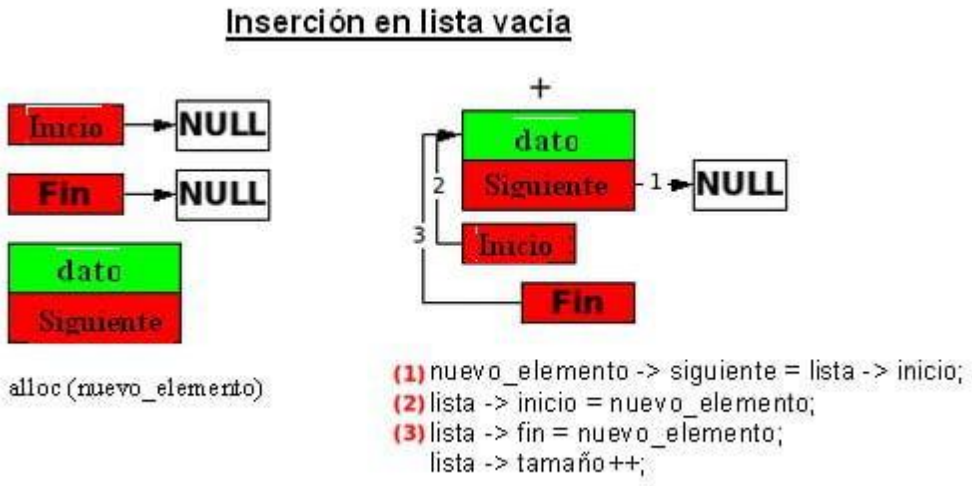
2. Inserción de un elemento en la lista

- A continuación, el algoritmo de inserción y el registro de los elementos: se declara el elemento que se va a insertar, y se realiza una asignación de la memoria para el nuevo elemento, llenado el contenido del campo de datos, actualización de los punteros hacia el primer y último elemento si es necesario.
- Caso particular: en una lista con un único elemento, el primero es al mismo tiempo el último.
- Para añadir un elemento a la lista se presentan varios casos: la inserción en una lista vacía, la inserción al inicio de la lista, la inserción al final de la lista y la inserción en otra parte de la lista.

a. Inserción en una lista vacía

```
int ins_en_lista_vacia (Lista *lista, char *dato);
```

- La función retorna 1 en caso de error, si no devuelve 0.
- Etapas:
  - Se asigna memoria para el nuevo elemento
  - el puntero siguiente de este nuevo elemento apuntará hacia NULL (ya que la inserción es realizada en una lista vacía
  - se utiliza la dirección del puntero inicio que vale NULL)
  - los punteros inicio y fin apuntaran hacia el nuevo elemento y el tamaño es actualizado.



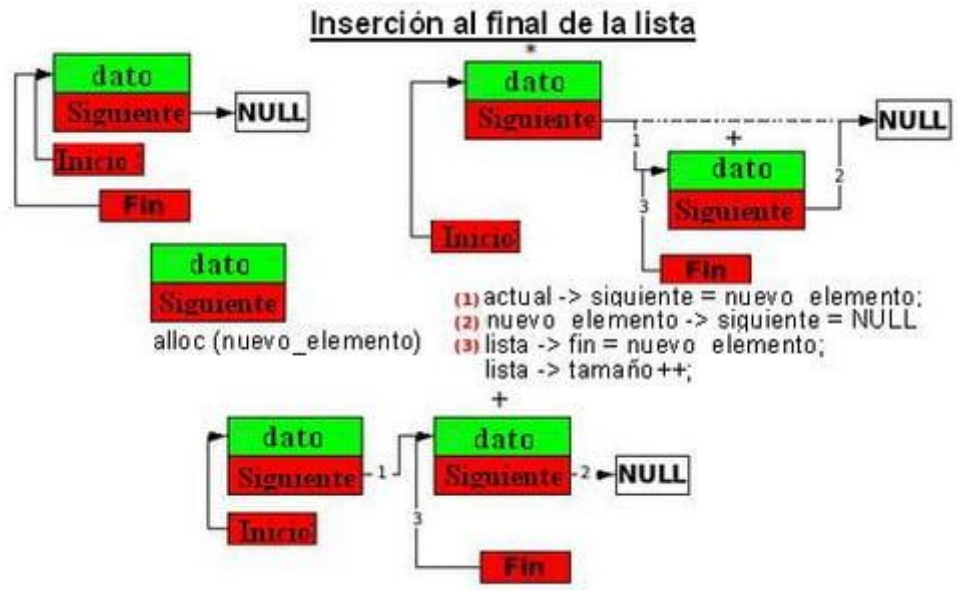

```
/* inserción al inicio de la lista */
int ins_inicio_lista (Lista * lista, char *dato){
Element *nuevo_elemento;
if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

nuevo_elemento->siguiente = lista->inicio
lista->inicio = nuevo_elemento;
lista->tamaño++;
return 0;
}
```

b. Inserción al final de la lista

```
int ins_fin_lista (Lista *lista, Element *actual, char *dato);
```

- La función da **-1** en caso de error, si no arroja **0**.
- Etapas:
  - proporcionar memoria al nuevo elemento
  - rellenar el campo de datos del nuevo elemento
  - el puntero **siguiente** del último elemento apunta hacia el nuevo elemento
  - el puntero **fin** apunta al nuevo elemento
  - el puntero **inicio** no varía, el tamaño es incrementado:



```
/*inserción al final de la lista */
int ins_fin_lista (Lista * lista, Element * actual, char *dato){
Element *nuevo_elemento;
if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

actual->siguiente = nuevo_elemento;
nuevo_elemento->siguiente = NULL;

lista->fin = nuevo_elemento;

lista->tamaño++;
return 0;
}
```

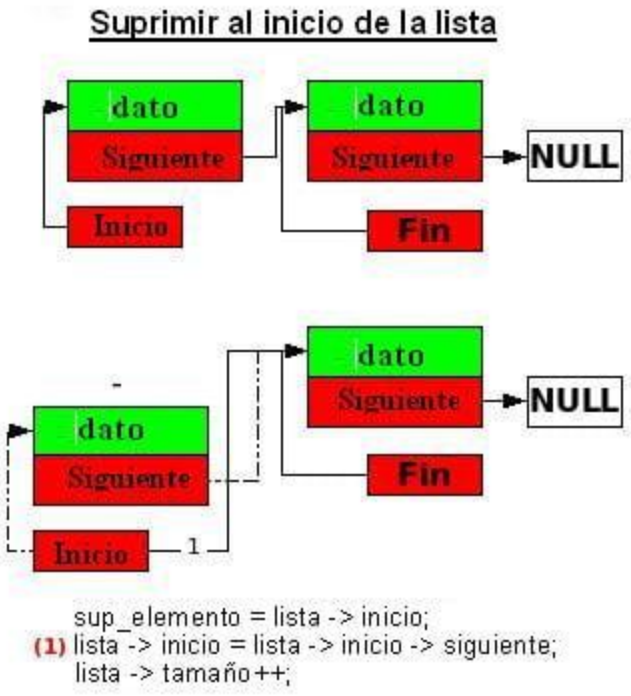
3. Eliminación de un elemento de la lista

- A continuación, un algoritmo para eliminar un elemento de la lista: uso de un puntero temporal para almacenar la dirección de los elementos a borrar, el elemento a eliminar se encuentra después del elemento actual, apuntar el puntero **siguiente** del elemento actual en dirección del puntero siguiente del elemento a eliminar, liberar la memoria ocupada por el elemento borrado, actualizar el tamaño de la lista.
- Para eliminar un elemento de la lista hay varios casos: eliminación al inicio de la lista y eliminación en otra parte de la lista.

a. Eliminación al inicio de la lista

```
int sup_inicio (Lista *lista);
```

- La función devolverá **-1** en caso de equivocación, de lo contrario da **0**.
- Etapas:
    - el puntero **sup_elem** contendrá la dirección del 1er elemento, el puntero **inicio** apuntara hacia el segundo elemento, el tamaño de la lista disminuirá un elemento:

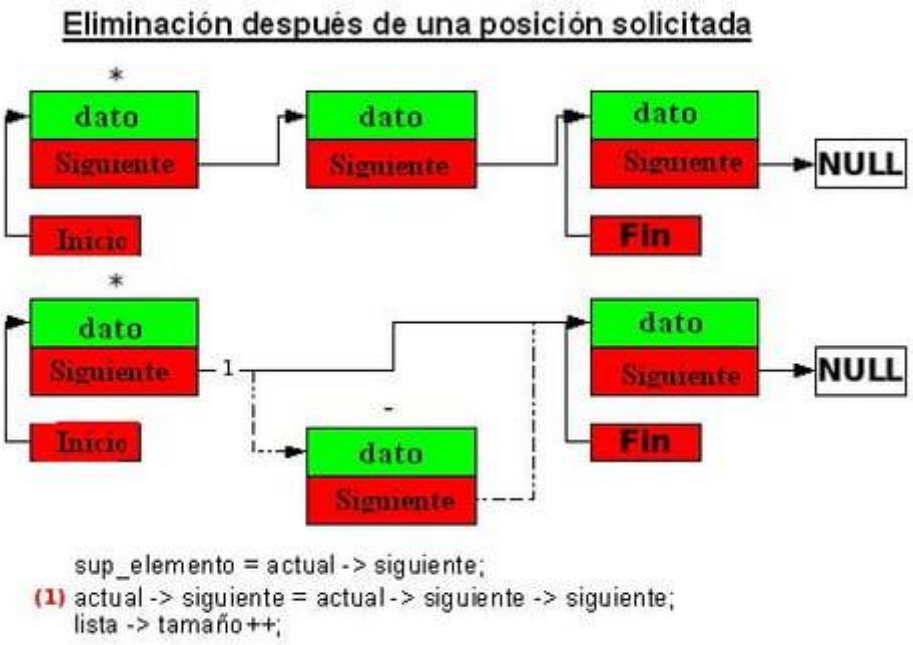


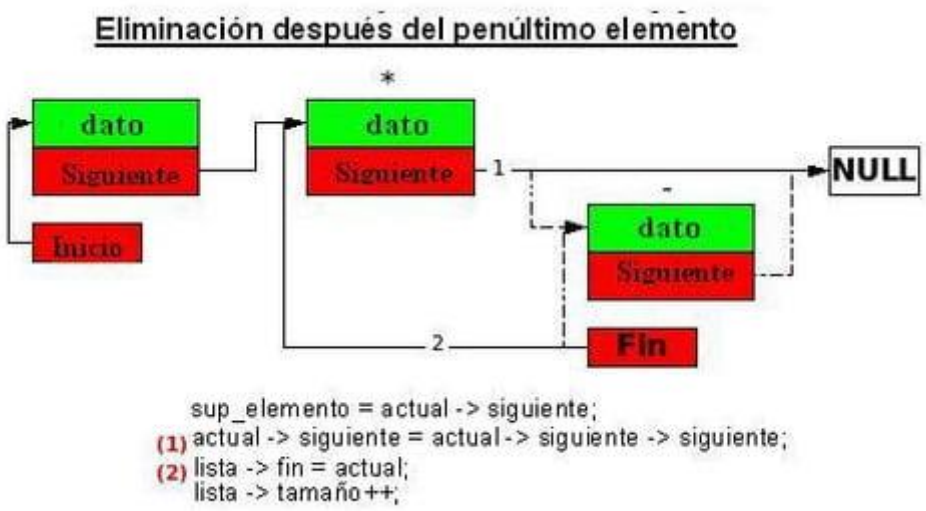

```
/* eliminación al inicio de la lista */
int sup_inicio (Lista * lista){
if (lista->tamaño == 0)
return -1;
Element *sup_elemento;
sup_element = lista->inicio;
lista->inicio = lista->inicio->siguiente;
if (lista->tamaño == 1)
lista->fin = NULL;
free (sup_elemento->dato);
free (sup_elemento);
lista->tamaño--;
return 0;
}
```

b. Eliminación en otra parte de la lista

```
int sup_en_lista (Lista *lista, int pos);
```

- La función da **-1** en caso de error, si no devuelve **0**.
- Etapas:
    - El puntero **sup_elem** contendrá la dirección hacia la que apunta el puntero **siguiente** del elemento **actual**
    - El puntero **siguiente** del elemento actual apuntará hacia el elemento al que apunta el puntero **siguiente** del elemento que sigue al elemento **actual** en la lista.
    - Si el elemento **actual** es el penúltimo elemento, el puntero **fin** debe ser actualizado. El tamaño de la lista será disminuido en un elemento:






```
/* eliminar un elemento después de la posición solicitada */
int sup_en_lista (Lista * lista, int pos){
if (lista->tamaño <= 1 || pos < 1 || pos >= lista->tamaño)
return -1;
int i;
Element *actual;
Element *sup_elemento;
actual = lista->inicio;

for (i = 1; i < pos; ++i)
actual = actual->siguiente;

sup_elemento = actual->siguiente;
actual->siguiente = actual->siguiente->siguiente;
if(actual->siguiente == NULL)
lista->fin = actual;
free (sup_elemento->dato);
free (sup_elemento);
lista->tamaño--;
return 0;
}
```

4. Visualización de la lista

- Para mostrar la lista entera hay que posicionarse al inicio de la lista (el puntero **inicio** lo permitirá).
- Luego usando el puntero **siguiente** de cada elemento la lista es recorrida del primero al último elemento. La condición para detener es proporcionada por el puntero **siguiente** del último elemento valga NULL.

```
/* visualización de la lista */
void visualización (Lista * lista){
Element *actual;
actual = lista->inicio;
while (actual != NULL){
printf ("%p - %s\n", actual, actual->dato);
actual = actual->siguiente;
}
}
```

5. Destrucción de la lista

```
/* destruir la lista */
void destruir (Lista *lista){
while (lista->tamaño > 0)
sup_inicio (lista);
}
```

Manual:

Se debe descargar el Código electoral y el Padrón de la siguiente página del TSE http://www.tse.go.cr/descarga_padron.htm .

Después de descomprimir el archivo se debe agregar a ala raíz del Proyecto en el cual se copiará él .cpp adjunto con este documento.

Al ser ejecutado se desplegará un menú el cual nos mostrará lo siguiente:

*El código No logra correr

Explicación del Código:

- Este Tendrá un struct que funcionara como un arreglo que define todas las características de Cada Votante como lo son Nombre, Cedula, etc.
- Este arreglo contiene una función el agrega a cada uno de los votantes
- Luego tiene función para cargar los votantes, se encarga de tomar cada una de las líneas directo del archivo de texto y enviarlo a la función encargada de extraer cada uno de los datos e incorporarlos en un nodo de la lista. Esta cuenta con un contador el cual nos indica cuánto dura la Carga.
- Tenemos una función para liberar los votantes en estructuras de memoria dinámica de la lista enlazada hasta dejar la lista en NULL.
- Luego cuenta con una función que sirve para encontrar algún votante a partir de una Cedula o Nombre.
- Contamos con Una función la cual nos inicia un Arreglo de preguntas y en un Struct aparte definimos el Arreglo de Preguntas en otro apartado generamos un random que nos elegirá 3 preguntas de las 5 y en la siguiente función nos permite realizar las encuestas.
- Además, contamos con una función la cual carga el Distelec. (Distrito Electoral)

Bibliografía

[1]Villagomez C. (2004). La Lista Enlazada Simple. Recuperado de: https://es.ccm.net/faq/2842-la-lista-enlazada-simple