Universidad de Guanajuato División de Ingenierías
Campus Irapuato Salamanca (DICIS)

Algoritmos y estructura de datos

Carlos Hugo García Capulín

Tarea No. 11

Reporte Lista Enlazada Simple

Jair Chávez Islas

01/Noviembre/2021

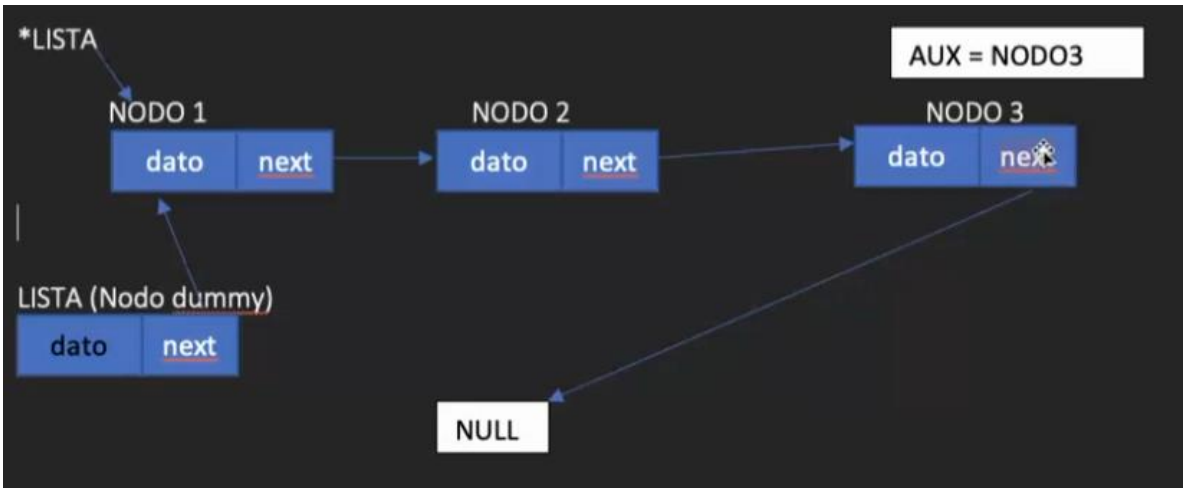
Problema

Una lista enlazada es una estructura de datos abstracta, dinámica y lineal. Consta de un elemento base llamado “NODO”, el cual consta de un dato y un apuntador al siguiente NODO.

Se dice que es dinámica porque no tiene un tamaño fijo, su tamaño crece conforme se agregan elementos a la lista.

Se dice que es lineal dado que todos los elementos están enlazados de manera consecutiva, es decir un NODO solamente puede apuntar al NODO siguiente.

De manera gráfica la podemos representar de la siguiente manera:



Supongamos que agregamos 9 nodos los cuales son los siguientes: 10,20,30,40,50,60,70,80 y 90, luego agregamos 4 nodos al principio los cuales son 6,7,8,9 y eliminamos un nodo del final, así la lista queda de la siguiente manera:

6
7
8
9
10
20
30
40
50
60
70
80

Luego eliminamos 3 nodos más del final, quedaría así:

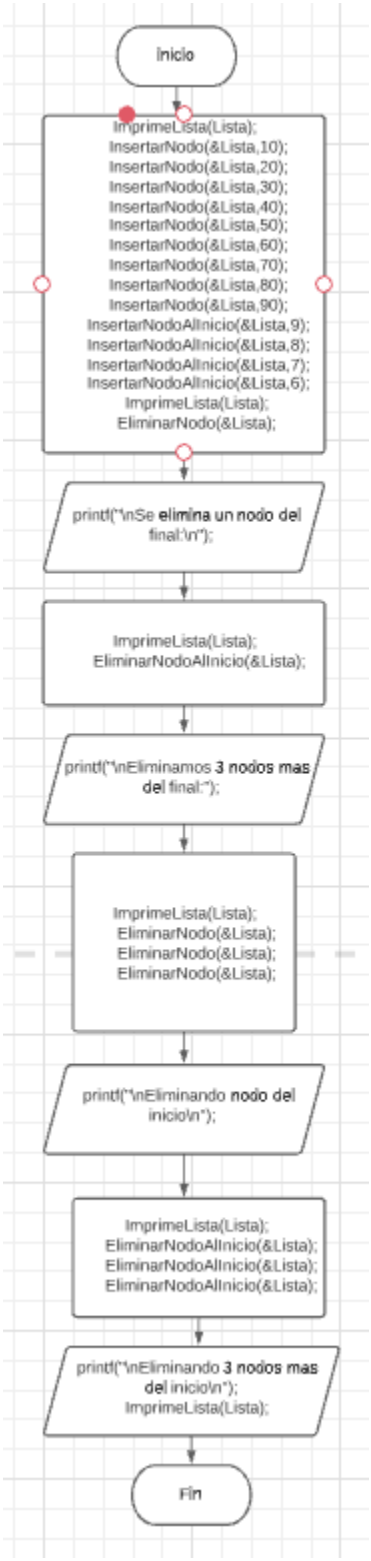
6
7
8
9
10
20
30
40
50

Pero al final eliminamos los nodos del inicio, queda de la siguiente manera:

10
20
30
40
50

Solución implementada

Diagrama del programa



Código comentado del programa

```c
//Agregamos las librerias necesarias para las funciones que necesitamos
#include <stdio.h>
#include <stdlib.h>

struct NODO;

//Declaramos una estructura
typedef struct
{
    //Miembros de la estructura
    int dato;
    struct NODO *next;
}NODO;//Nombre de la estructura

//Aqui ponemos los prototipos de las funciones utilizadas
NODO *CrearNodo(int dato);
void InsertarNodo(NODO **list, int dato);
void ImprimeLista(NODO *list);
void EliminarNodo(NODO **list);
void InsertarNodoAlInicio(NODO **list, int dato);
void EliminarNodoAlInicio(NODO **list);

//Inicializamos la funcion principal
int main()
{
    //Declaracion de variables de esta funcion
    NODO *Lista=NULL;
    int num;

    //Imprimimos la lista
    ImprimeLista(Lista);
    //Insertamos un nodo 10
    InsertarNodo(&Lista,10);
    //Insertamos un nodo 20
    InsertarNodo(&Lista,20);
    //Insertamos un nodo 30
    InsertarNodo(&Lista,30);
    //Insertamos un nodo 40
    InsertarNodo(&Lista,40);
    //Insertamos un nodo 50
    InsertarNodo(&Lista,50);
    //Insertamos un nodo 60
    InsertarNodo(&Lista,60);
    //Insertamos un nodo 70
    InsertarNodo(&Lista,70);
    //Insertamos un nodo 80
    InsertarNodo(&Lista,80);
    //Insertamos un nodo 90
    InsertarNodo(&Lista,90);
    //Insertamos nodo al inicio 9
    InsertarNodoAlInicio(&Lista,9);
```

```
    InsertarNodoAlInicio(&Lista,9);
    //Insertamos nodo al inicio 8
    InsertarNodoAlInicio(&Lista,8);
    //Insertamos nodo al inicio 7
    InsertarNodoAlInicio(&Lista,7);
    //Insertamos nodo al inicio 6
    InsertarNodoAlInicio(&Lista,6);

    //Imprimimos la lista
    ImprimeLista(Lista);

    //Eliminamos un nodo del final
    EliminarNodo(&Lista);
    printf("\nSe elimina un nodo del final:\n");
    //Imprimimos la lista
    ImprimeLista(Lista);

    //Eliminamos un nodo del final
    EliminarNodo(&Lista);
    //Eliminamos un nodo del final
    EliminarNodo(&Lista);
    //Eliminamos un nodo del final
    EliminarNodo(&Lista);
    printf("\nEliminamos 3 nodos mas del final:");
    //Imprimimos la lista
    ImprimeLista(Lista);

    EliminarNodoAlInicio(&Lista);
    printf("\nEliminando nodo del inicio\n");
    //Imprimimos la lista
    ImprimeLista(Lista);
    //Eliminamos un nodo del inicio
    EliminarNodoAlInicio(&Lista);
    //Eliminamos un nodo del inicio
    EliminarNodoAlInicio(&Lista);
    //Eliminamos un nodo del inicio
    EliminarNodoAlInicio(&Lista);
    printf("\nEliminando 3 nodos mas del inicio\n");
    //Imprimimos la lista
    ImprimeLista(Lista);

    printf(" \n");
    return 0;//se retorna el valor de 0 para verificar que haya finalizado sin problemas
}

void InsertarNodoAlInicio(NODO **list, int dato)
{
    NODO *NewNodo;

    NewNodo = CrearNodo(dato);
```

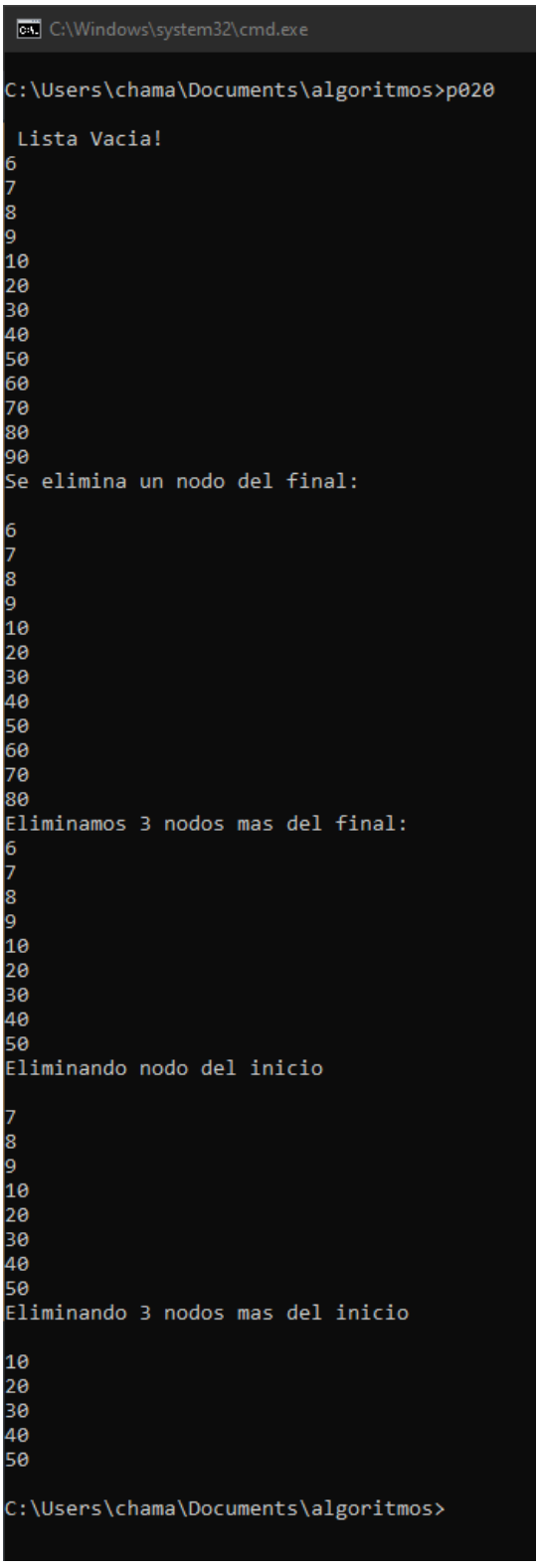
```
102     if(*list==NULL)
103     {
104         //Lista vacia
105         *list = NewNodo; // Se asigna la direcion del nuevo nodo a list, entonces se convierte en el nodo1
106     }
107     else
108     {
109         //La lista contiene al menos un nodo insertado
110         NewNodo->next = (struct NODO *)*list;
111         *list = NewNodo;
112     }
113 }
114 
115 void EliminarNodoAlInicio(NODO **list)
116 {
117     NODO *aux;
118 
119     if(*list==NULL)
120     {
121         //Lista vacia
122         printf("\nNo se puede eliminar Nodo, Lista Vacia!.");
123     }
124     else
125     {
126         aux = *list;
127         *list = (NODO *)aux->next;
128         free(aux);
129 
130     }
131 }
132 
133 void EliminarNodo(NODO **list)
134 {
135     NODO *ultimo;
136     NODO *penultimo;
137 
138     if(*list==NULL)
139     {
140         //Lista vacia
141         printf("\nNo se puede eliminar Nodo, Lista Vacia!.");
142     }
143     else
144     {
145         //La lista contiene al menos un nodo insertado
146         ultimo = *list;
147         if(ultimo->next==NULL)
148         {
149             free(ultimo);
150             *list = NULL;
151         }
152         else
```

```
153         {
154             //Buscar el nodo que apunta a NULL(El nodo final)
155             while(ultimo->next!=NULL)
156             {
157                 penultimo = ultimo;
158                 ultimo =(NODO *) ultimo->next;
159             }
160             penultimo->next = NULL;
161             free(ultimo);
162         }
163 
164     }
165 }
166 
167 void ImprimeLista(NODO *list)
168 {
169     NODO *aux;
170     if(list==NULL)
171     {
172         //Lista vacia
173         printf("\n Lista Vacia!");
174     }
175     else
176     {
177         //La lista contiene al menos un nodo insertado
178         aux = list;
179         //Imprimir el nodo apuntado por aux
180         do
181         {
182             printf("\n%i",aux->dato);
183             aux =(NODO *) aux->next;
184 
185         } while(aux!=NULL);
186     }
187 }
188 
189 NODO *CrearNodo(int dato)
190 {
191     NODO *ptr;
192 
193     ptr = (NODO *)malloc(sizeof(NODO));
194 
195     if(ptr==NULL)
196     {
197         printf("\n Error al reservar la memroia para el nodo.");
198         exit(0);
199     }
200     ptr->dato = dato;
201     ptr->next = NULL;
202 
203     return ptr;
```

```
203     return ptr;
204 }
205 void InsertarNodo(NODO **list, int dato)
206 {
207     NODO *aux;
208     NODO *NewNodo;
209 
210     NewNodo = CrearNodo(dato);
211 
212     if(*list==NULL)
213     {
214         //Lista vacia
215         *list = NewNodo; // Se asigna la direcion del nuevo nodo a list, entonces se convierte en el nodo1
216     }
217     else
218     {
219         //La lista contiene al menos un nodo insertado
220         aux = *list;
221 
222         // Buscar el nodo que paunta a NULL (El nodo final)
223         while(aux->next!=NULL)
224             aux = (NODO *)aux->next;
225 
226         aux->next = (struct NODO *)NewNodo;
227     }
228 }
```

Pruebas y resultados

Evidencia del programa



Como vimos y habíamos previsto en el planteamiento del problema, la lista quedó con los nodos exactamente como lo habíamos previsto con los cambios que hicimos quedando al final solo la siguiente lista:

10
20
30
40
50