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

Algoritmos y estructura de datos

Carlos Hugo García Capulín

Tarea No. 12

Reporte Lista Enlazada Doble

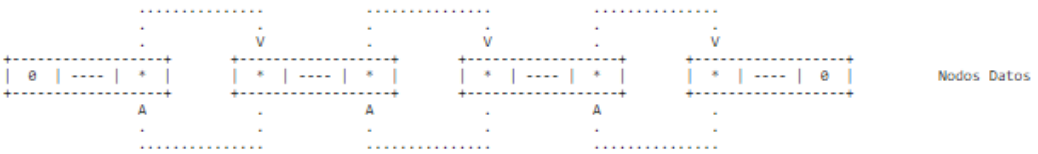
Jair Chávez Islas

03/Diciembre/2021

Problema

Es un tipo de lista enlazada que permite moverse hacia delante y hacia atrás; Cada nodo de una lista doblemente enlazada tiene dos enlaces, ademas de los campos de datos. Un enlace, el derecho, se utiliza para navegar la lista hacia delante. El otro enlace, el isquierdo, se utiliza para navegar la lista hacia atrás; Las Listas pueden navegarse hacia delante y hacia atrás; Las Listas pueden crear, actualizar y eliminar elementos; En las Listas la posicion de los elementos es relevante; Las Listas admiten elementos duplicados; Las Listas tienen dos protocolos, uno secuencial y el otro directo.

Representación Enlazada



* Significa un enlace v lido a un nodo

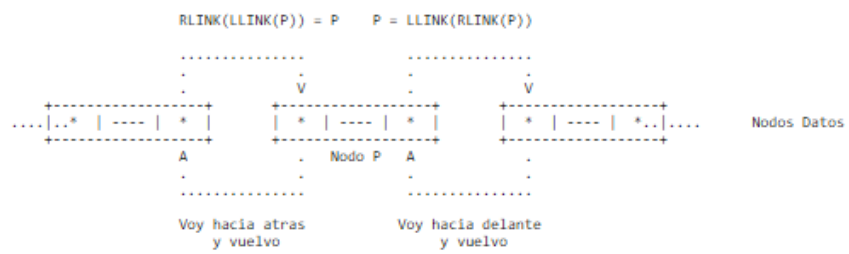
0 Significa el final de la lista o enlace nulo

Representacion Secuencial

Representacion Lineal

La escencia de las Lista Enlazada Doble

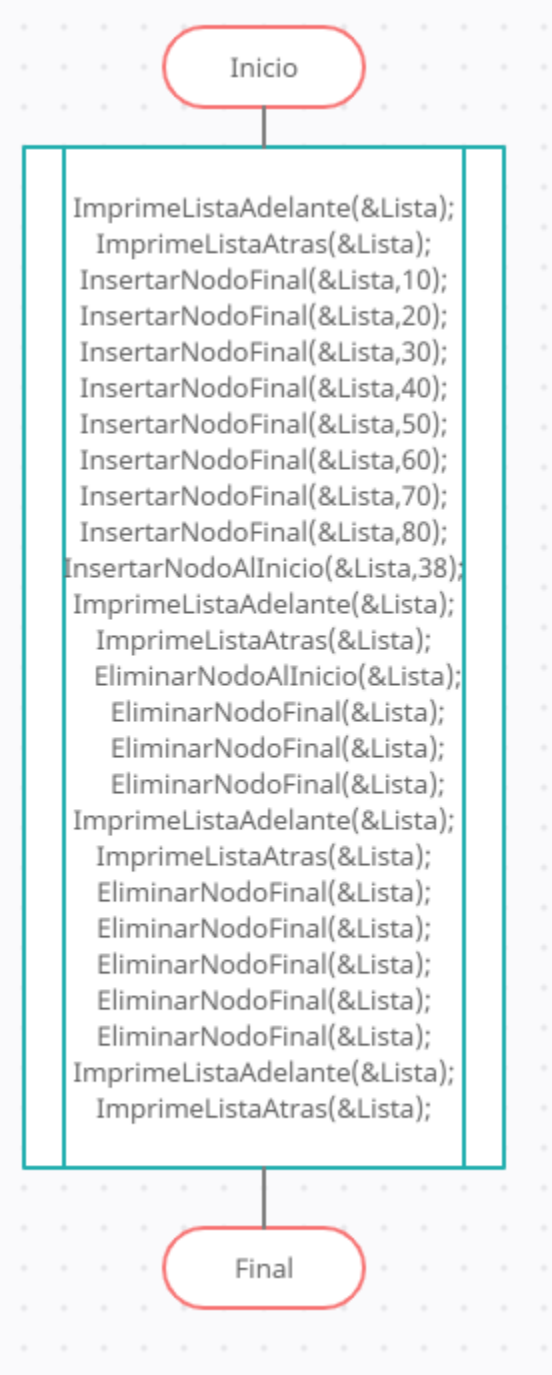
Asumamos a P ser un pointer, o sea, una variable que solo tiene referencias a nodos.



Poniendo como ejemplo el utilizado en clase, Si intentamos imprimir la lista de nodos sol empezar el programa, no podemos, ay que la lista está vacía aún, peroluego creamos ocho nodo y les agregamos lso valores 10, 20, 30 , 40, 50, 60, 70 y 80, quedando así la lista impresa ahcia adelante e imrpesa hacia atrás quedaría de la siguiente manera: 80, 70, 60, 50, 40, 30, 20, 10, y luego agregamos un nodo más pero esta vez al inicio con el valor de 38, tenemos la siguiente lista 38, 10, 20, 30 , 40, 50, 60, 70 y 80, y podemos también eliminar el primer nodo quedando como antes, y al momento de eliminar todos los nodos, la lista vuelve a estar vacía

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;
    struct NODO *prev;
}NODO;//Nombre de la estructura

//Aqui ponemos los prototipos de las funciones utilizadas
NODO* CrearNodo(int dato);
void InsertarNodoFinal(NODO *list, int dato);
void ImprimeListaAdelante(NODO *list);
void ImprimeListaAtras(NODO *list);
void EliminarNodoFinal(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;
    Lista.next=NULL;
    Lista.prev=NULL;

    //Imprimimos la lista hacia adelante
    ImprimeListaAdelante(&Lista);
    //Imprimimos la lista hacia atras
    ImprimeListaAtras(&Lista);
    //Insertamos un nodo al final con el valor de 10
    InsertarNodoFinal(&Lista,10);
    //Insertamos un nodo al final con el valor de 20
    InsertarNodoFinal(&Lista,20);
    //Insertamos un nodo al final con el valor de 30
    InsertarNodoFinal(&Lista,30);
    //Insertamos un nodo al final con el valor de 40
    InsertarNodoFinal(&Lista,40);
    //Insertamos un nodo al final con el valor de 50
    InsertarNodoFinal(&Lista,50);
    //Insertamos un nodo al final con el valor de 60
    InsertarNodoFinal(&Lista,60);
    //Insertamos un nodo al final con el valor de 70
    InsertarNodoFinal(&Lista,70);
    //Insertamos un nodo al final con el valor de 80
    InsertarNodoFinal(&Lista,80);
    //Insertamos un nodo al Inicio con el valor de 38
```

```c
    //Insertamos un nodo al Inicio con el valor de 38
    InsertarNodoAlInicio(&Lista,38);
    printf("\nLa lista de inicio a fin: ");
    //Imprimimos la lista hacia adelante
    ImprimeListaAdelante(&Lista);
    printf("\nLa lista de fin a inicio: ");
    //Imprimimos la lista hacia atras
    ImprimeListaAtras(&Lista);
    EliminarNodoAlInicio(&Lista);
    //Eliminamos 3 nodos
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    printf("\nLa lista de inicio a fin despues de eliminar 3 nodos: ");
    //Imprimimos la lista hacia adelante
    ImprimeListaAdelante(&Lista);
    printf("\nLa lista de fin a inicio despues de eliminar 3 nodos: ");
    //Imprimimos la lista hacia atras
    ImprimeListaAtras(&Lista);
    //Eliminamos 5 nodos mas
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    EliminarNodoFinal(&Lista);
    printf("\nLa lista de inicio a fin despues de eliminar 8 nodos: ");
    //Imprimimos la lista hacia adelante
    ImprimeListaAdelante(&Lista);
    printf("\nLa lista de fin a inicio despues de eliminar 8 nodos: ");
    //Imprimimos la lista hacia atras
    ImprimeListaAtras(&Lista);

    printf("\n");
    return 0;
}

//Inicializamos la funcion Insertar nodo al inicio
void InsertarNodoAlInicio(NODO *list, int dato)
{
    NODO* aux;
    NODO* newNodo;
    newNodo = CrearNodo(dato);

    if((list->next == NULL) && (list->prev == NULL))
    { //Lista vacia
        list->next = (struct NODO*)newNodo;
        list->prev = (struct NODO*)newNodo;
        //Se asigna la dirección del nuevo nodo al apuntador del nodo inicial y nodo final
    }
    else
    { //La lista contiene al menos un nodo inicializado
        aux =(NODO *)list->next;
        //Buscar el nodo de inicio
```

```
106             while(aux->prev != NULL)
107                 aux = (NODO*)aux->prev;
108 
109             aux ->prev= (struct NODO*)newNodo;
110             newNodo->next= (struct NODO*)aux;
111             list->next=(struct NODO*) newNodo;
112         }
113     }
114 
115 
116     void EliminarNodoAlInicio(NODO *list)
117     {
118         NODO* inicio;
119         NODO* segundo;
120         if((list->next == NULL) && (list->prev == NULL))
121         {
122         //Lista vacia
123         printf("\nNo se puede eliminar Nodo, lista vacía\n");
124         }
125         else
126         {
127         //La lista contiene al menos un nodo inicializado
128         inicio = (NODO*)list->prev;
129         if(list->next==list->prev)
130             {
131                 free(inicio);
132                 list->next=NULL; //hacer la lista vacia
133                 list->prev=NULL; //hacer la lista vacia
134             }
135             else
136             {
137             //Buscar el nodofinal
138                 while(inicio->prev != NULL)
139                     inicio = (NODO*)inicio->prev;
140                 segundo=(NODO*)inicio->next;
141                 segundo->prev= NULL;
142                 list->next=(struct NODO*)segundo;
143                 free(inicio); //liberar la memoria del ultimo nodo
144             }
145         }
146     }
147     void InsertarNodoFinal(NODO *list, int dato)
148     {
149         //Se declaran las variables locales de la funcion
150         NODO *aux;
151         NODO *Newnodo;
152 
153         //Añade un nodo al final de la lista
154         Newnodo=CrearNodo(dato);
155         if((list->next==NULL)&&(list->prev==NULL))
156         {
157             list->next=(struct NODO *)Newnodo;
158             list->prev=(struct NODO *)Newnodo;
159         }
```

```
160         else
161         {
162             aux=list;
163 
164             while(aux->next!=NULL)
165                 aux=(NODO *)aux->next;
166             aux->next=(struct NODO *)Newnodo;
167             Newnodo->prev=(struct NODO *)aux;
168             list->prev=(struct NODO *)Newnodo;
169         }
170     }
171 
172     void EliminarNodoFinal(NODO *list)
173     {
174         //Se declaran las variables locales de la funcion
175         NODO *ultimo;
176         NODO *penultimo;
177 
178         if((list->next==NULL)&&(list->prev==NULL))
179         {
180             printf("\nNo se puede eliminar nodo, lista vacia!");
181         }
182         else
183         {
184             ultimo = (NODO *)list->next;
185             if (list -> next==list->prev)
186             {
187                 free(ultimo);
188                 list->next=NULL;
189                 list->prev=NULL;
190             }
191             else
192             {
193                 while(ultimo->next!=NULL)
194                     ultimo =(NODO *)ultimo->next;
195 
196                 penultimo=(NODO *)ultimo->prev;
197                 penultimo->next=NULL;
198                 list->prev=(NODO *) penultimo;
199                 free(ultimo);
200             }
201         }
202     }
203 
204     void ImprimeListaAdelante(NODO *list)
205     {
206         //Se declaran las variables locales de la funcion
207         NODO* aux;
208 
209         if(list->next == NULL)
210         {
```

```
210     {
211         //Lista vacia
212         printf("Lista vacía\n");
213     }
214 
215     else
216     {
217         //La lista contiene al menos un nodo inicializado
218         aux = (NODO*)list->next;
219         //Imprimimr el nodo apuntado por aux
220         do
221         {
222             printf("\n%i ", aux->dato);
223             aux = (NODO*)aux ->next;
224         }
225         while(aux != NULL);
226     }
227 }
228 
229 //Se inicializa la funcion imprimelistaatras
230 void ImprimeListaAtras(NODO *list)
231 {
232     //Se declaran las variables locales de la funcion
233         NODO* aux;
234 
235     if(list->prev == NULL)
236     {
237         //Lista vacia
238         printf("Lista vacía\n");
239     }
240 
241     else
242     {
243         //La lista contiene al menos un nodo inicializado
244         aux = (NODO*)list->prev;
245         //Imprimimr el nodo apuntado por aux
246         do
247         {
248             printf("\n%i ", aux->dato);
249             aux = (NODO*)aux ->prev;
250         }
251         while(aux != NULL);
252     }
253 }
254 
255 //Se inicializa la funcion Crear nodo
256 NODO* CrearNodo(int dato)
257 {
258     //Se declaran las variables locales de la funcion
259     NODO *ptr;
260     ptr = (NODO *)malloc(sizeof(NODO));
261     if (ptr==NULL)
```

```
261         if (ptr==NULL)
262         {
263             printf("Error al reservar memoria para el nodo");
264             exit(0);
265         }
266         ptr->dato=dato;
267         ptr->next=NULL;
268         ptr->prev=NULL;
269         return ptr;
270     }
```

Pruebas y resultados

Evidencia del programa

Los resultados son los mimos que habíamos previsto en la introducción de este reporte.