Universidad Nacional Autónoma de Nicaragua, León

Facultad de Ciencias y Tecnología

Departamento de Computación

Componente: Aplicaciones de Estructuras de Datos

Título de la Guía: Listas dobles y operaciones básicas

Objetivos:

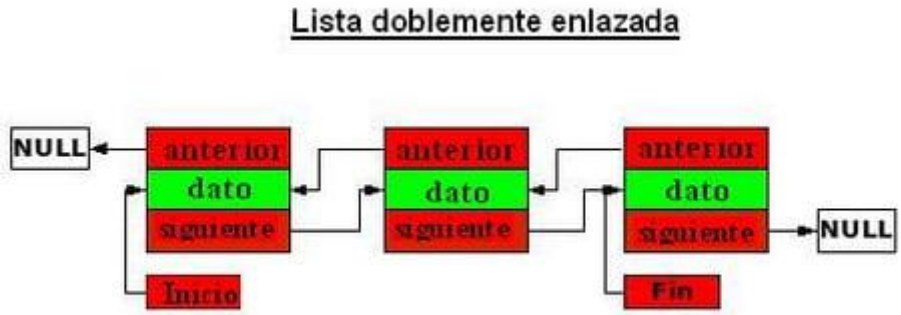
- Definir y analizar el uso de listas dobles, sus variantes y operaciones básicas para el acceso a datos.

"A la libertad por la Universidad"

1. Listas dobles

Las listas doblemente enlazadas son estructuras de datos semejantes a las listas enlazadas simples, pero algo más complejas. La asignación de memoria se realiza en el momento de la ejecución.

En cambio, en relación a las listas enlazadas simple el enlace entre los elementos se hace gracias a dos punteros (uno que apunta hacia el elemento anterior y otro que apunta hacia el elemento siguiente):



Operaciones sobre la lista doblemente enlazadas

- **Inicialización**: void inicialización (Lista *lista);

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

- **Inserción de un elemento en la lista**

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

Insertar en lista vacia

```
int insercion_en_lista_vacia (dl_Lista * lista, char *dato){
  dl_Elemento *nuevo_elemento;
  if ((nuevo_elemento = alloc (nuevo_elemento)) == NULL)
    return -1;
  strcpy (nuevo_elemento->dato, dato);
  nuevo_elemento->anterior = lista->inicio;
  nuevo_elemento->siguiente = lista->fin;
  lista->inicio = nuevo_elemento;
  lista->fin = nuevo_elemento;
```

```
  lista->tamaño++;
  return 0;
}
```

Inserción al inicio de la lista

```
int ins_inicio_lista(dl_Lista * lista, char *dato){
  dl_Elemento *nuevo_elemento;
  if ((nuevo_elemento = alloc (nuevo_elemento)) == NULL)
    return -1;
  strcpy (nuevo_elemento->dato, dato);
  nuevo_elemento->anterior = NULL;
  nuevo_elemento->siguiente = lista->inicio;
  lista->inicio->anterior = nuevo_elemento;
  lista->inicio = nuevo_elemento;
  lista->tamaño++;
  return 0;
}
```

Inserción al final de la lista

```
int ins_fin_lista(dl_Lista * lista, char *dato){
  dl_Elemento *nuevo_elemento;
  if ((nuevo_elemento = alloc (nuevo_elemento)) == NULL)
    return -1;
  strcpy (nuevo_elemento->dato, dato);
  nuevo_elemento->siguiente = NULL;
  nuevo_elemento->anterior = lista->fin;
  lista->fin->siguiente = nuevo_elemento;
  lista->fin = nuevo_elemento;
  lista->tamaño++;
  return 0;
}
```

Inserción antes de un elemento de la lista

```
int ins_antes (dl_Lista * lista, char *dato, int pos){
  int i;
  dl_Elemento *nuevo_elemento, *actual;
```

```
  if ((nuevo_elemento = alloc (nuevo_elemento)) == NULL)
    return -1;
  strcpy (nuevo_elemento->dato, dato);
  actual = lista->inicio;
  for (i = 1; i < pos; ++i)
    actual = actual->siguiente;
  nuevo_elemento->siguiente = actual;
  nuevo_elemento-> anterior = actual->anterior;
  if(actual->anterior == NULL)
    lista->inicio = nuevo_elemento;
  else
    actual->anterior->siguiente = nuevo_elemento;
  actual->anterior = nuevo_elemento;
  lista->tamaño++;
  return 0;
}
```

Inserción después de un elemento de la lista

```
int ins_después (dl_Lista * lista, char *dato, int pos){
  int i;
  dl_Elemento *nuevo_elemento, *actual;
  if ((nuevo_elemento = alloc (nuevo_elemento)) == NULL)
    return -1;
  strcpy (nuevo_elemento->dato, dato);
  actual = lista->inicio;
  for (i = 1; i < pos; ++i)
    actual = actual->siguiente;
  nuevo_elemento->siguiente = actual->siguiente;
  nuevo_elemento->anterior = actual;
  if(actual->siguiente == NULL)
    lista->fin = nuevo_elemento;
  else
    actual->siguiente->anterior = nuevo_elemento;
  actual->siguiente = nuevo_elemento;
  lista->tamaño++;
```

```
  return 0;
}
```

- **Eliminación de un elemento de la lista**

```
int supp(dl_Lista *lista, int pos){
  int i;
  dl_Elemento *sup_elemento,*actual;
  if(lista->tamaño == 0)
    return -1;
  if(pos == 1){ /* eliminación del 1er elemento */
    sup_elemento = lista->inicio;
    lista->inicio = lista->inicio->siguiente;
    if(lista->inicio == NULL)
      lista->fin = NULL;
    else
      lista->inicio->anterior == NULL;
  }else if(pos == lista->tamaño){ /* eliminación del último elemento */
    sup_elemento = lista->fin;
    lista->fin->anterior->siguiente = NULL;
    lista->fin = lista->fin->anterior;
  }else { /* eliminación en otra parte */
    actual = lista->inicio;
      for(i=1;i<pos;++i)
        actual = actual->siguiente;
    sup_elemento = actual;
    actual->anterior->siguiente = actual->siguiente;
    actual->siguiente->anterior = actual->anterior;
  }
  free(sup_elemento->dato);
  free(sup_elemento);
  lista->tamaño--;
  return 0;
}
```

- **Visualización de la lista**

```
void affiche(dl_Lista *lista){ /* visualización hacia adelante */
  dl_Elemento *actual;
  actual = lista->inicio; /* punto de inicio el 1er elemento */
  printf("[ ");
  while(actual != NULL){
    printf("%s ",actual->dato);
    actual = actual->siguiente;
  }
  printf("]\n");
}

void mustra_inv(dl_Lista *lista){ /* visualización hacia atrás */
  dl_Elemento *actual;
  actual = lista->fin; /* punto de inicio el ultimo elemento */
  printf("[ ");
  while(actual != NULL){
    printf("%s : ",actual->dato);
    actual = actual->anterior;
  }
  printf("]\n");
}
```

- **Destrucción de la lista**

```
void destruir(dl_Lista *lista){
  while(lista->tamaño > 0)
    sup(lista,1);
}
```

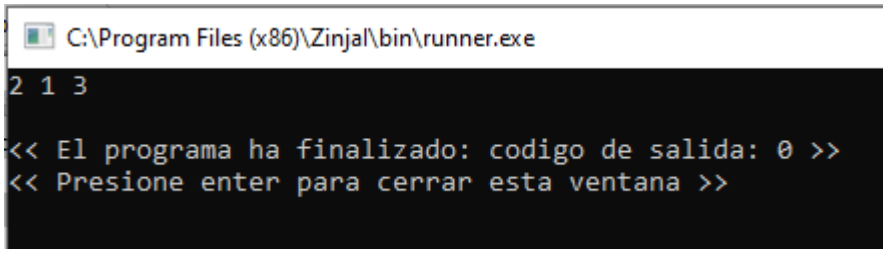
1. Ejercicios Resueltos

a. Ejercicios Resuelto 1:

```
#include <stdio.h>
#include <stdlib.h>

//Estructura del nodo
typedef struct nodo
{
   int dato;
   struct nodo *siguiente;
   struct nodo *anterior;
} NODO;

NODO *CrearNodo(int dato);
int InsertarInicio(NODO **cabeza, int dato);
int InsertarFinal(NODO **cabeza, int dato);
void ImprimirLista(NODO *cabeza);
int EliminarNodo(NODO **cabeza, int dato);

int main()
{
   NODO *cabeza = NULL;
   InsertarInicio(&cabeza, 1);
   InsertarInicio(&cabeza, 2);
   InsertarFinal(&cabeza, 3);
   InsertarFinal(&cabeza, 4);
   EliminarNodo(&cabeza, 4);
   ImprimirLista(cabeza);
   system("pause>nul");
   return 0;
}

//Función para eliminar un nodo de la lista
int EliminarNodo(NODO **cabeza, int dato)
{
   NODO *actual = *cabeza, *ant = NULL, *sig = NULL;
   while(actual != NULL)
   {
      if(actual->dato == dato)
      {
         if( actual == *cabeza)
         {
            *cabeza = actual->siguiente;
```

```
            if( actual->siguiente != NULL)
               actual->siguiente->anterior = NULL;
         }
         else if( actual->siguiente == NULL)
         {
            ant = actual->anterior;
            actual->anterior = NULL;
            ant->siguiente = NULL;
         }
         else
         {
            ant = actual->anterior;
            actual->anterior = NULL;
            sig = actual->siguiente;
            actual->siguiente = NULL;
            ant->siguiente = sig;
            sig->anterior = ant;
         }
         free(actual);
         return 1;
      }
      actual = actual->siguiente;
   }
   return 0;
}

//Función para insertar al final de la lista
int InsertarFinal(NODO **cabeza, int dato)
{
   NODO *nuevo = NULL, *nAux = *cabeza;
   nuevo = CrearNodo(dato);
   if (nuevo != NULL)
   {
      while(nAux->siguiente != NULL){ nAux = nAux->siguiente;}
      nuevo->anterior = nAux;
      nAux->siguiente = nuevo;
      return 1;
   }
   return 0;
}

//Función para imprimir la lista
void ImprimirLista(NODO *cabeza)
{
```

```
    NODO *nAux = cabeza;
    while(nAux != NULL)
    {
        printf("%d ", nAux->dato);
        nAux = nAux->siguiente;
    }
}

//Función para insertar al inicio de la lista
int InsertarInicio(NODO **cabeza, int dato)
{
    NODO *nuevo = NULL;
    nuevo = CrearNodo(dato);
    if (nuevo != NULL)
    {
        nuevo->siguiente = *cabeza;
        nuevo->anterior = NULL;
        if( *cabeza != NULL)
            (*cabeza)->anterior = nuevo;
        *cabeza = nuevo;
        return 1;
    }
    return 0;
}

//Función para crear un nuevo nodo
NODO *CrearNodo(int dato)
{
    NODO* nuevo = NULL;
    nuevo = (NODO*)malloc(sizeof(NODO));
    if( nuevo != NULL)
    {
        nuevo->dato = dato;
        nuevo->siguiente = NULL;
        nuevo->anterior = NULL;
    }
    return nuevo;
}
```



b. Ejercicios Resuelto 2:

```
#include <stdlib.h>
#include <stdio.h>

struct Node {
	int data;
	struct Node* next;
	struct Node* previous;
};

void print_list_backwards(struct Node *headNode)
{
	if (NULL == headNode)
	{
		return;
	}
	struct Node *i = headNode;
	while (i->next != NULL) {
		i = i->next; /* Move to the end of the list */

	}
	while (i != NULL) {
		printf("Value: %d\n", i->data);
		i = i->previous;
	}
}

void print_list(struct Node *headNode)
{
	struct Node *i;
	for (i = headNode; i != NULL; i = i->next) {
		printf("Value: %d\n", i->data);
	}
}

void insert_at_beginning(struct Node **pheadNode, int value)
{
	struct Node *currentNode;

	if (NULL == pheadNode)
	{
		return;
	}

	currentNode = malloc(sizeof *currentNode);
	currentNode->next = NULL;
	currentNode->previous = NULL;
	currentNode->data = value;

	if (*pheadNode == NULL) { /* The list is empty */
		*pheadNode = currentNode;
```

```
                return;
        }

        currentNode->next = *pheadNode;
        (*pheadNode)->previous = currentNode;
        *pheadNode = currentNode;
}

void insert_at_end(struct Node **pheadNode, int value) {
        struct Node *currentNode;
        if (NULL == pheadNode)
        {
                return;
        }

        currentNode = malloc(sizeof *currentNode);
        struct Node *i = *pheadNode;

        currentNode->data = value;
        currentNode->next = NULL;
        currentNode->previous = NULL;

        if (*pheadNode == NULL) {
                *pheadNode = currentNode;
                return;
        }
        while (i->next != NULL) { /* Go to the end of the list */
                i = i->next;
        }

        i->next = currentNode;
        currentNode->previous = i;
}

void free_list(struct Node *node) {
        while (node != NULL) {
                struct Node *next = node->next;
                free(node);
                node = next;
        }

}

int main(void) {
        /* Sometimes in a doubly linked list the last node is also stored */
        struct Node *head = NULL;

        printf("Insert a node at the beginning of the list.\n");
        insert_at_beginning(&head, 5);
        print_list(head);
```

```
    printf("Insert a node at the beginning, and then print the list backwards\n");
    insert_at_beginning(&head, 10);
    print_list_backwards(head);

    printf("Insert a node at the end, and then print the list forwards.\n");

    insert_at_end(&head, 15);
    print_list(head);

    free_list(head);

    return 0;
}
```

```
C:\Program Files (x86)\Zinjal\bin\runner.exe
Insert a node at the beginning of the list.
Value: 5
Insert a node at the beginning, and then print the list backwards
Value: 5
Value: 10
Insert a node at the end, and then print the list forwards.
Value: 10
Value: 5
Value: 15


<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

2. Ejercicios propuestos

a. Ejercicios Propuesto 1:

Escribir un programa que permita invertir los datos almacenados en una lista doblemente enlazada, es decir que el primer elemento pase a ser el último elemento y el último pase a ser el primer elemento, que el segundo sea el penúltimo y el penúltimo pase a ser el segundo y así sucesivamente.

b. Ejercicios Propuesto 2:

Escribir un programa que retorne el número de veces que se encuentra el dato dentro de la lista doble. En caso de no encontrarse, se debe mostrar un mensaje indicando que el dato no fue encontrado. Se debe ingresar el valor que se desea buscar.

c. Ejercicios Propuesto 3:

Crear un programa que maneje el registro de los estudiantes, utilizando listas doblemente enlazadas. Los estudiantes aprobados deben insertarse al inicio y los reprobados por el final de la lista. Los datos requeridos por cada estudiante son los siguientes:

- Código
- Nombre
- Apellido
- Correo
- Nota

El programa debe permitir realizar las operaciones de:

1. Agregar un estudiante
2. Buscar un estudiante por código
3. Eliminar un estudiante
4. Total, de estudiantes aprobados
5. Total, de estudiantes reprobados

3. Bibliografía

- Ceballos Francisco Javier. Curso de Programación C/C++. Segunda Edición. Editorial RA-MA, Madrid, 2002.
- Joyanes Aguilar Luis, Zahonero Martínez Ignacio. Programación en C. Metodología, estruc-tura de datos y objetos. Mc Graw Hill.
- Byron S. Gottfried . Programación en C. Mc Graw Hil
- H.M. Dietel – P. J. Dietel . Como Programar en C/C++. Segunda Edición. Prentice Hall