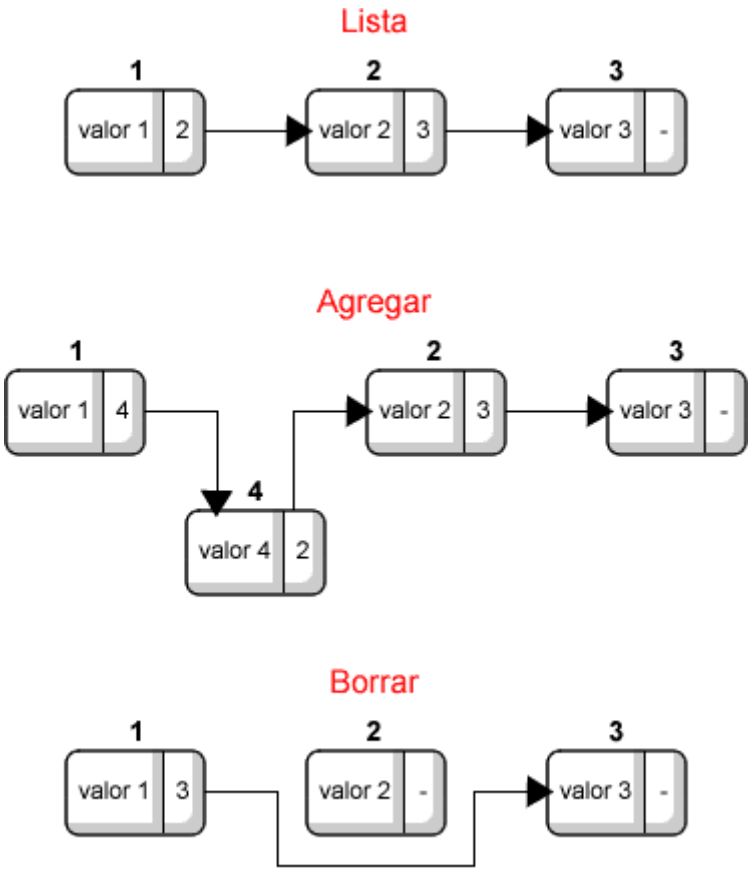
UNIDAD CURRICULAR: ALGORITMICA Y PROGRAMACIÓN

UNIDAD XII. LISTAS ENLAZADAS

CONTENIDO:



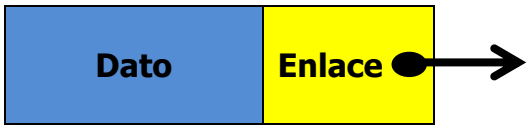
UNIDAD XII

LISTAS ENLAZADAS

CONCEPTO

Una lista es una secuencia de 0 o más elementos de un tipo dado almacenados en memoria. Son estructuras lineales formadas por una secuencia de ***nodos***, en los que se guardan campos de datos arbitrarios y una o dos referencias (punteros) al nodo anterior o posterior. Si una lista tiene 0 elementos se dice que la lista está vacía.

Estructura de un nodo simple

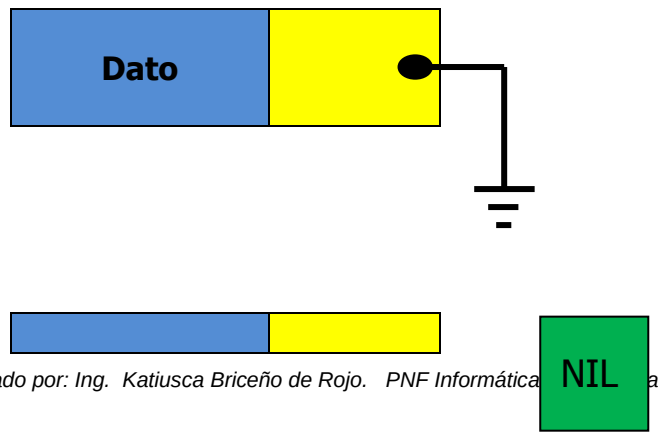


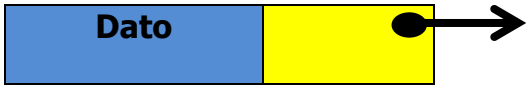
Como puede observarse, un nodo simple se representa con una caja rectangular dividida en dos secciones: Dato o información y enlace o puntero.

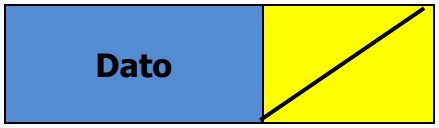
Dato: Representa el dato o los datos a almacenar. Puede ser de cualquier tipo.

Enlace: es un puntero al siguiente elemento de la lista y se representa con una flecha para facilitar la comprensión de la conexión de los nodos. Este puntero contiene la dirección del nodo sucesor, de esta forma se enlazan y podemos construir la lista. El último nodo no enlaza con ningún otro, por lo tanto, el puntero contiene un valor nulo que en el caso de C++ se denomina ***nil.***

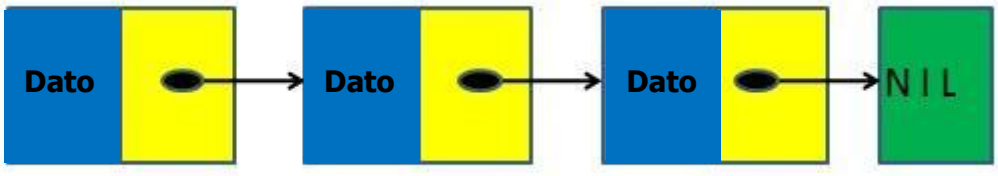
Representaciones gráficas de un nodo con enlace apuntando a *nil*:







Representación gráfica de la lista enlazada



CLASIFICACIÓN DE LISTAS

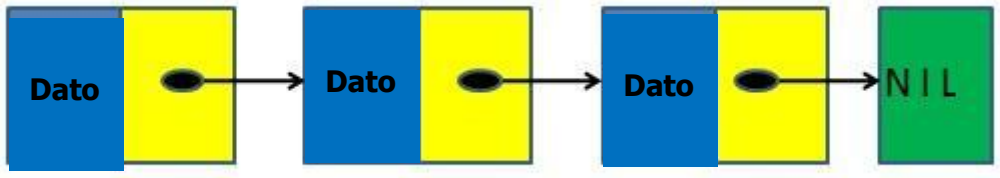
Existen varios tipos de listas:

- Listas simples o enlazadas
- Listas circulares
- Listas doblemente enlazadas
- Listas circulares doblemente enlazadas.

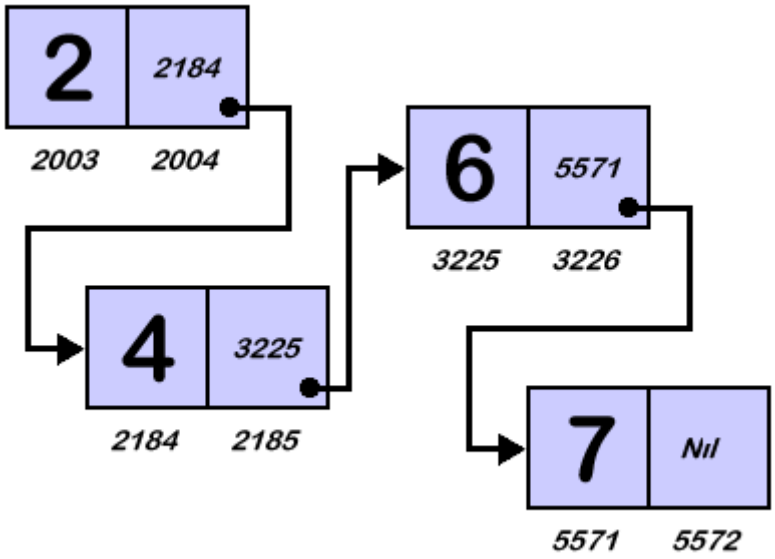
LISTAS SIMPLES O ENLAZADAS

En una lista enlazada, cada elemento apunta al siguiente excepto el último que no tiene sucesor y el valor del enlace es ***nil.*** Por ello los elementos son registros que contienen el dato a almacenar y un enlace al siguiente elemento.

Representación Gráfica



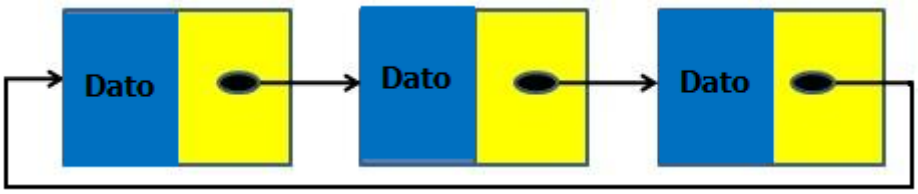
A continuación se presenta un ejemplo donde se observa que los enlaces contienen la dirección de memoria del siguiente nodo:



LISTAS CIRCULARES

Es una modificación de la lista enlazada, en las que el puntero del último elemento apunta al primero de la lista. Es decir que no hay ningún enlace a ***nil.***

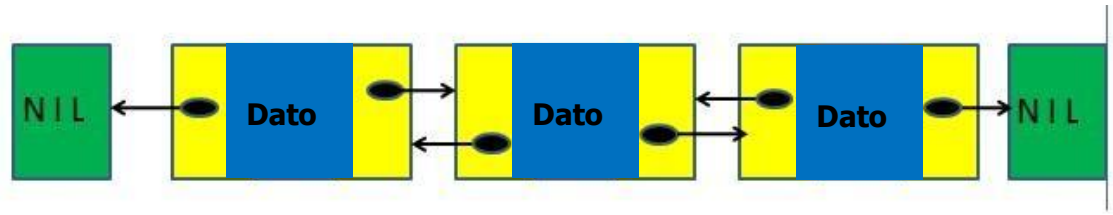
Representación Gráfica



LISTAS DOBLEMENTE ENLAZADAS

En este tipo de lista, cada nodo consta de un campo con información y dos punteros: un puntero que apunta al nodo anterior y otro que apunta al nodo siguiente. En el caso del primer nodo, como no tiene nodo anterior, el puntero que apunta a anterior apuntará a ***nil***. En el caso del último nodo, el puntero que apunta al siguiente también apuntará a ***nil***.

Representación Gráfica



LISTAS DOBLEMENTE ENLAZADAS CIRCULARES

En este tipo de lista, cada nodo consta de un campo con información y dos punteros: un puntero que apunta al nodo anterior y otro que apunta al nodo siguiente. En el caso del primer nodo, como no tiene nodo anterior, el puntero que apunta a anterior apuntará a último nodo. En el caso del último nodo, el puntero que apunta al siguiente apuntará al primer nodo.

Representación Gráfica

A continuación se muestran dos ejemplos:

OPERACIONES CON LISTAS ENLAZADAS

Las operaciones que podemos realizar sobre una lista enlazada son las siguientes:

- ***Recorrido:*** Esta operación consiste en visitar cada uno de los nodos que forman la lista. Para recorrer todos los nodos de la lista, se comienza con el primero, luego se toma como referencia el puntero que apunta al siguiente nodo para avanzar al segundo nodo, luego el campo puntero de este nodo nos dará la dirección del tercer nodo, y así sucesivamente.

- ***Inserción:*** Esta operación consiste en agregar un nuevo nodo a la lista. Para esta operación se pueden considerar tres casos:
  - Insertar un nodo al inicio.
  - Insertar un nodo antes o después de cierto nodo.
  - Insertar un nodo al final.

- ***Eliminación o Borrado:*** La operación de borrado consiste en quitar un nodo de la lista, redefiniendo las direcciones de los punteros que correspondan. Se pueden presentar cuatro casos:
  - Eliminar el primer nodo.
  - Eliminar el último nodo.
  - Eliminar un nodo antes o después de cierto nodo.

- ***Búsqueda:*** Esta operación consiste en visitar cada uno de los nodos, tomando al campo puntero como la dirección al siguiente nodo a visitar.

EJERCICIOS RESUELTOS

A continuación se presenta un miniproyecto desarrollado en C++ que contiene las operaciones básicas de una lista enlazada simple.

Consiste en crear una lista enlazada cuyos nodos guarden la información de los empleados: cedula, nombre, edad y sueldo. El programa debe tener las siguientes opciones:

1.- Añadir empleado a la lista
2.- Borrar empleado;
3.- Mostrar listado de empleados
4.- Calcular Promedio de Sueldos
5.- Borrar lista Completa

La opción *Añadir empleado a la lista* consiste en insertar un nodo a la lista de empleados.

La opción *Borrar empleado* consiste en realizar la búsqueda de un empleado en la lista de empleados y eliminar el nodo encontrado.

La opción *Mostrar listado de empleados* consiste en mostrar por pantalla la lista de empleados.

La opción *Calcular Promedio de Sueldos* consiste en recorrer toda la lista para calcular el promedio de sueldos.

La opción *Borrar lista completa* consiste en eliminar todos los nodos de la lista.

```
// programa con operaciones básicas de listas simples o enlazadas.
#include <iostream>
#include <stdlib.h>
#include <string.h>
using namespace std;
typedef struct _nodoempleado
  {char cedula [10];  //cedula del empleado
   char nombre[20];   //nombre del empleado
   int edad;          // edad del empleado
   float sueldo;     //sueldo del empleado
   _nodoempleado *siguiente;   //puntero que apunta a otro elemento o
    };                         //nodo con esta misma estructura
_nodoempleado *primero, *ultimo;   //declaramos dos punteros para guardar la direccion del primer y del último elemento de la lista

int opcion;

int mostrar_menu() {  //función que muestra un menu y retorna la opcion seleccionada
cout<<"Menú de Listas Enlazadas:"<<endl;
cout<<"1.- Añadir empleado a la lista"<< endl;
cout <<"2.- Borrar empleado"<<endl;
cout <<"3.- Mostrar listado de empleados"<<endl;
```

```
cout <<"4.- Calcular Promedio de Sueldos"<<endl;
cout <<"5.- Borrar lista Completa"<<endl;
cout <<"6.- Salir"<<endl;
cout <<"Escoge una opción: ";
cin >> opcion;
return opcion;
}


void anadir_elemento() {  //añadir un nodo a la lista

_nodoempleado *nuevonodo;   //puntero del nodo que guardara la informacion del nuevo empleado
nuevonodo = (_nodoempleado *) malloc (sizeof(_nodoempleado)); //malloc reserva la memoria del
                                                              //nuevo nodo y devuelve la direccion
                                                              //de memoria donde se guardará el nodo o retorna NULL si no hay memoria
if (nuevonodo==NULL)   //verificamos si malloc retornó NULL, es decir no hay memoria.
   {cout << "No hay memoria disponible...";}
else  //si hay memoria, malloc si retornó una direccion de memoria disponible para el nodo
  { //a continuación pedimos los datos y los guardamos en el registro (nodo)
  cout <<" Ingreso del Nuevo empleado:"<<endl;
  cout <<"C.I.: ";
  cin >>nuevonodo->cedula;
  cout <<"Nombre: ";
  cin >>nuevonodo->nombre;
  cout <<"Edad: "<<endl;
  cin>>nuevonodo->edad;
  cout <<"Sueldo: "<<endl;
  cin >>nuevonodo->sueldo;
  nuevonodo->siguiente = NULL; //el nuevo nodo debe apuntar a Null por ser
                               //el último nodo. El último nodo siempre debe apuntar a NULL en una lista simple
  if (primero==NULL) //el puntero primero guarda la dirección del primer elemento de la lista. Esto se hace con
                     //la finalidad de saber donde comienza la lista.
    {cout << "Añadiendo el primer elemento" << endl;
      primero = nuevonodo; //como es el primer nodo, el puntero primero debe apuntar a este nodo.
      ultimo = nuevonodo; //ultimo siempre debe apuntar al nuevo nodo.
    }
  else { //si el nodo a agregar no es el primero se hace lo siguiente:
      ultimo->siguiente = nuevonodo; //el último actual pasa a ser penúltimo, ya que deja de apuntar a NULL y ahora apunta al nuevo nodo.
      ultimo = nuevonodo; //el puntero ultimo siempre debe apuntar al nuevo nodo que se incluya
       }
  }
}



void eliminar_de_la_lista()
{
 _nodoempleado *auxnodo,*auxsig;
 int i;
 char cedulaabuscar[10];
 bool eliminado;
```

```
 i=0;
 auxnodo = primero;
 if (auxnodo==NULL)
   {cout <<"La lista está vacía"<<endl;}
 else
   {cout<<"Ingrese el numero de cedula del empleado a eliminar:";
    cin >>cedulaabuscar;
    eliminado=false;
       if (strcmp(primero->cedula,cedulaabuscar)==0)
       {cout<<"Eliminando el primer empleado"<<endl;
         auxnodo=primero;
         primero=primero->siguiente;
         free(auxnodo);
         eliminado=true;
        }
     else
        {auxnodo=primero;
         auxsig=primero->siguiente;
         while (auxnodo->siguiente !=NULL && eliminado==false)
           if (strcmp(auxsig->cedula,cedulaabuscar)==0)
            {auxnodo->siguiente=auxsig->siguiente;
             free(auxsig);
             cout<<"eliminacion realizada"<<endl;
             eliminado=true;
             }
           else
             {cout <<primero->cedula<<cedulaabuscar;
              cout<<strcmp(primero->cedula,cedulaabuscar);
              auxnodo=auxnodo->siguiente;
              auxsig=auxsig->siguiente;
              }
        }//else
     if (eliminado==false)
       {cout<<"La cedula suministrada no se encuentra registrada"<<endl;}
}

}

void promedio_sueldos()
{
_nodoempleado *auxnodo;
int i;
float prom;
i=0;
prom=0;
auxnodo = primero;
if (auxnodo==NULL)
  {cout <<"La lista está vacía"<<endl;}
else
 {
  cout <<"Calculando el Promedio de Sueldos:"<<endl;
  while (auxnodo!=NULL)
       {
        prom = prom + auxnodo->sueldo;
        i++;
        auxnodo = auxnodo->siguiente;
       }
  prom = prom / i ;
```

```
   cout <<"El Número Total de Empleados es: "<<i<<endl;
   cout <<"El Promedio de Sueldo para cada uno de ellos es: "<<prom<<endl;
  }
}


void eliminar_lista_completa()
{
  _nodoempleado *auxnodo;

  if (primero==NULL)
     {cout <<"La lista está vacía"<<endl;}
  else
     {
      while (primero != NULL)
           {
            auxnodo=primero;
            primero=primero->siguiente;
            free(auxnodo);
           }
      cout <<"Se han eliminado todos los elementos de la Lista"<<endl;
     }
}



void mostrar_lista() {
_nodoempleado *auxnodo;
int i;
i=0;
auxnodo = primero;
if (auxnodo==NULL)
{cout <<"La lista está vacía"<<endl;}
else
{
cout <<"Mostrando la lista de empleados:"<<endl;
while (auxnodo!=NULL)
  {i++;
   cout<<"Datos del empleado N°: "<<i<<endl;
   cout << "Cedula: "<<auxnodo->cedula<<endl;
   cout << "Nombre: "<<auxnodo->nombre<<endl;
   cout <<"Edad: "<<  auxnodo->edad<<endl;
   cout <<"Sueldo:  "<< auxnodo->sueldo<<endl;
   auxnodo = auxnodo->siguiente;
   }
}
}

main()
{
primero = (_nodoempleado *) NULL;
ultimo = (_nodoempleado *) NULL;
do {
   opcion= mostrar_menu();
   switch ( opcion )
   {
      case 1: anadir_elemento();
      break;
      case 2:  eliminar_de_la_lista();
```

```
      break;
      case 3: mostrar_lista();
      break;
      case 4: promedio_sueldos();
      break;
      case 5: eliminar_lista_completa();
      break;
      default: cout <<( "Opción no válida\n" );
      break;
      }
   } while (opcion!=6);
}
```

REFERENCIAS BIBLIOGRÁFICAS

Joyanes, L. (2008). Fundamentos de programación. Algoritmos , Estructuras de datos y objetos. Mc Graw Hill. Tercera edición.

Joyanes, L. y Zahonero, I. (2005). Programación en C. Metodología, algoritmos y Estructura de datos. Mc Graw Hill. Segunda Edición

Martí, N. y Ortega, Y. (2004). Estructuras de datos y Métodos Algorítmicos. Ejercicios Resueltos. Pearson Education.