Listas Enlazadas

Héctor Ferrada
académico

Instituto de Informática
Universidad Austral de Chile
INFO088 - Estructuras de Datos y Algoritmos

2do Semestre, 2019

Listas Enlazadas en C++

Recordemos el Concepto de Puntero

Considere la variable $int\ var = 17;$

- Esta variable, que usamos para simples cómputos, la entendemos como una localización de memoria para trabajar con un valor entero (típicamente 32 bits).

Recordemos el Concepto de Puntero

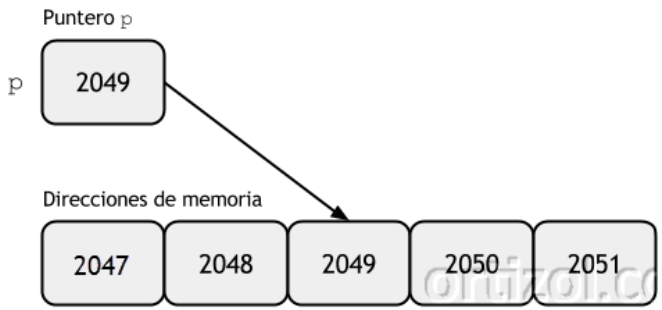
Considere la variable $int\ var = 17;$

- Esta variable, que usamos para simples cómputos, la entendemos como una localización de memoria para trabajar con un valor entero (típicamente 32 bits).
- En nuestro código no nos intereza su dirección física, simplemente usamos su identificador (el nombre de la variable: *var*) para trabajar con lo que hay en dicha localidad.

Recordemos el Concepto de Puntero

Considere la variable $int\ var = 17$;

- Esta variable, que usamos para simples cómputos, la entendemos como una localización de memoria para trabajar con un valor entero (típicamente 32 bits).
- En nuestro código no nos intereza su dirección física, simplemente usamos su identificador (el nombre de la variable: *var*) para trabajar con lo que hay en dicha localidad.
- Con cada variable *var*, podemos obtener su dirección física de memoria con $\&var$:
  - Así, podemos imprimir su dirección con: cout « "$\&var =$" « $\&var$;
  - Ó usanr un "**puntero**" a dicha localidad: int $*p = \&var$;



$\&var = 2049$

Ejemplo en C++

Escriba el sgte. código y ejecútelo:

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char** argv){
    int var;
    int *p = &var; // p apunta a la posicion de memoria de var
    *p = 20;       // asigna un valor en la posicion de memoria.
    cout « "Direccion de var: " « &var « " y var = " « var « endl;
    if ((p != nullptr))
      cout«"p apunta a " « p « " y guarda *p = " « *p « endl;
    p = NULL;
    if ((p == nullptr))
      cout « "p apunta a nada !! " « endl;
    return 0
}
```

¿Qué son la Listas Enlazadas?

- Una lista enlazada es una estructura de datos construida a partir de struct's y punteros.

¿Qué son la Listas Enlazadas?

- Una lista enlazada es una estructura de datos construida a partir de struct's y punteros.
- Su forma se asemeja a una "cadena" de *Nodos*, donde los punteros son el enlace entre los nodos.

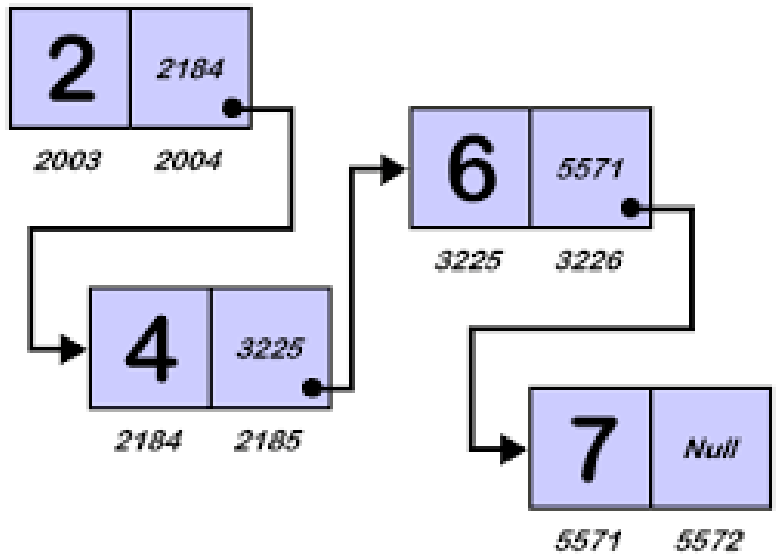
- Debe existir una variable que sea la cabeza de la lista (*head*).
- NULL representa que el puntero no tiene ninguna dirección de memoria (apunta a nada) y en el diagrama marca el fin de la lista.

Nodos en Cualquier Lugar

- Una lista enlazada es una estructura llamada "lineal", ya que es una estructura secuencial donde, a lo más, existe un solo predecesor y un solo sucesor para cada elemento de la lista.

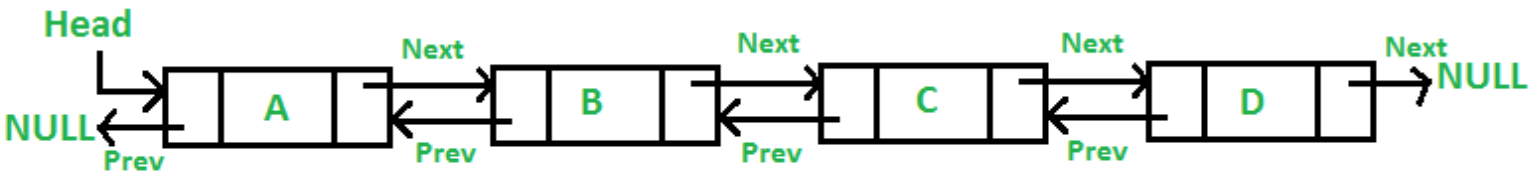
Nodos en Cualquier Lugar

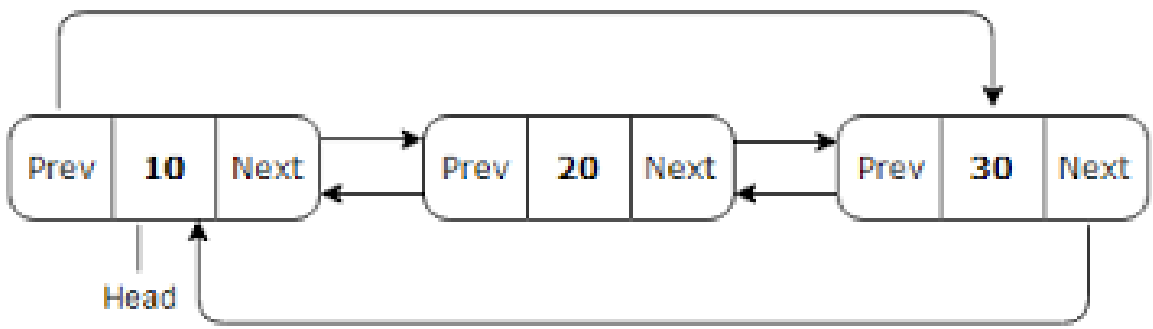
- Una lista enlazada es una estructura llamada "lineal", ya que es una estructura secuencial donde, a lo más, existe un solo predecesor y un solo sucesor para cada elemento de la lista.
- Una estructura lineal, no significa que sus nodos se encuentren almacenados consecutivamente en espacios de memoria.



Variedad de Listas Enlazadas

- Puedemos crear listas doblemente enlazadas, listas circulares, etc... todo depende de nuestra creatividad y de lo que se necesite





Algunas Preguntas Sobre las Listas Enlazadas...

- ¿Cuanta es la memoria que se necesita por cada nodo? ¿Suma el puntero también a los datos del nodo?

Algunas Preguntas Sobre las Listas Enlazadas...

- ¿Cuanta es la memoria que se necesita por cada nodo? ¿Suma el puntero también a los datos del nodo?
- Por tanto, ¿Que pasa con los tamaños si la comparamos con un arreglo del mismo largo que almacena los mismos datos?

Algunas Preguntas Sobre las Listas Enlazadas...

- ¿Cuanta es la memoria que se necesita por cada nodo? ¿Suma el puntero también a los datos del nodo?
- Por tanto, ¿Que pasa con los tamaños si la comparamos con un arreglo del mismo largo que almacena los mismos datos?
- ¿Puedo acceder a elementos en posiciones aleatorias como en un arreglo A, p.ej. a la í-sesima posición del array $A[i]$?

Algunas Preguntas Sobre las Listas Enlazadas...

- ¿Cuanta es la memoria que se necesita por cada nodo? ¿Suma el puntero también a los datos del nodo?
- Por tanto, ¿Que pasa con los tamaños si la comparamos con un arreglo del mismo largo que almacena los mismos datos?
- ¿Puedo acceder a elementos en posiciones aleatorias como en un arreglo A, p.ej. a la í-sesima posición del array $A[i]$?
- Si una lista de n nodos está totalmente ordenada, ¿Puedo implementar una búsqueda binaria sobre ella?

Algunas Preguntas Sobre las Listas Enlazadas...

- ¿Cuanta es la memoria que se necesita por cada nodo? ¿Suma el puntero también a los datos del nodo?
- Por tanto, ¿Que pasa con los tamaños si la comparamos con un arreglo del mismo largo que almacena los mismos datos?
- ¿Puedo acceder a elementos en posiciones aleatorias como en un arreglo A, p.ej. a la í-sesima posición del array $A[i]$?
- Si una lista de n nodos está totalmente ordenada, ¿Puedo implementar una búsqueda binaria sobre ella?
- Entonces, ¿Bajo que situaciones ud. usaria un arreglo o una lista enlazada?

Listas Enlazadas Simples en C++

- A partir de ahora, para las definiciones de nuevos tipos de datos usaremos la palabra reservada ***typedef***.

Listas Enlazadas Simples en C++

- A partir de ahora, para las definiciones de nuevos tipos de datos usaremos la palabra reservada ***typedef***.
- Definiremos un nuevo tipo de datos: **nodo**. Este será un struct, **nodeList**, que almacenara un entero (su data) y un puntero a otro nodeList:

```
struct nodeList {
    int val;
    nodeList *next;
};
typedef struct NodeList nodo;
```

- Ahora existe un nuevo tipo de datos llamado **nodo**, y lo podemos usar para crear nuestra lista vacia.

```
nodo *miLista = NULL;
```

Creando Nuestra Lista Enlazada

En una carpeta vacia "listaEnlazada" cree el archivo listas.cpp con:

```
#include <iostream>
#include <cstdlib>
using namespace std;

struct nodeList {
    int val;
    nodeList *next;
};
typedef struct nodeList nodo;

void appendToListL(nodo **ℓ, int num);      // note que hay doble *
void printList(nodo *ℓ);                    // aca solo hay un *

int main(int argc, char **argv){
    nodo *miLista = NULL;

    return EXIT_SUCCESS
}
```

Compile: g++ -std=c++14 -o listas listas.cpp

Agregando Nodos a la Lista

Tomaremos la siguiente convención al insertar un valor *num* a la lista ℓ:

- llamaremos appentToListL(ℓ, *num*), método que agrega un nodo con valor *num* **al inicio de la lista**; es decir por la izquierda (L de Left).
- llamaremos appentToListR(ℓ, *num*), método que agrega un nodo con valor *num* **al final de la lista**; es decir por la derecha (R de Right).

Note que es mucho más sencillo añadir un nodo por la izquierda si la cabeza de la lista esta apuntando al nodo de la extrema izquierda. Así, se crea un nuevo nodo: *nuevo* con el valor *num*, se hace apuntar a ℓ y se hace apuntar la cabeza de la lista a *nuevo*.
Codifique ambos métodos.

Creando Nuestra Lista Enlazada

En el código anterior, crearemos nuestra lista vacia y luego añadiremos los nodos por la izquieda de la lista, invocando a appendToListL(). Añadiremos los valores: $\{50, 40, 30, 20, 10\}$.
Cada vez que ingresemos un valor, imprimeremos la lista completa.

Creando Nuestra Lista Enlazada

En el código anterior, crearemos nuestra lista vacia y luego añadiremos los nodos por la izquieda de la lista, invocando a appendToListL(). Añadiremos los valores: $\{50, 40, 30, 20, 10\}$.
Cada vez que ingresemos un valor, imprimeremos la lista completa.
Edite el método main() y escriba:

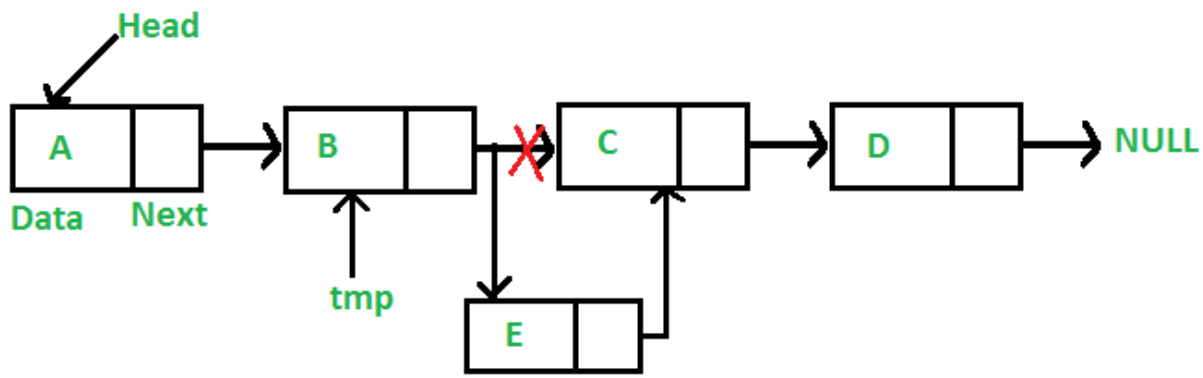
```
int main(int argc, char **argv){
    nodo *miLista = NULL;

    for (int i = 50; i >= 10; i- = 10){
        appendToListL(&miLista, i);
        printList(miLista);
    }
    return EXIT_SUCCESS
}
```

Trabaje en los métodos appendToListL() y printList().

Insertando nodos en el medio de la Lista

Suponga que deseamos insertar un nodo con un valor "E" después de "B":

- Debemos buscar al nodo *tmp* quién será el **antecesor** de "E" (tmp="B").
- Luego, hacemos apuntar el nuevo nodo a donde apunta *tmp*
- Y finalmente redireccionar el puntero de *tmp* hacia el nuevo nodo "E".

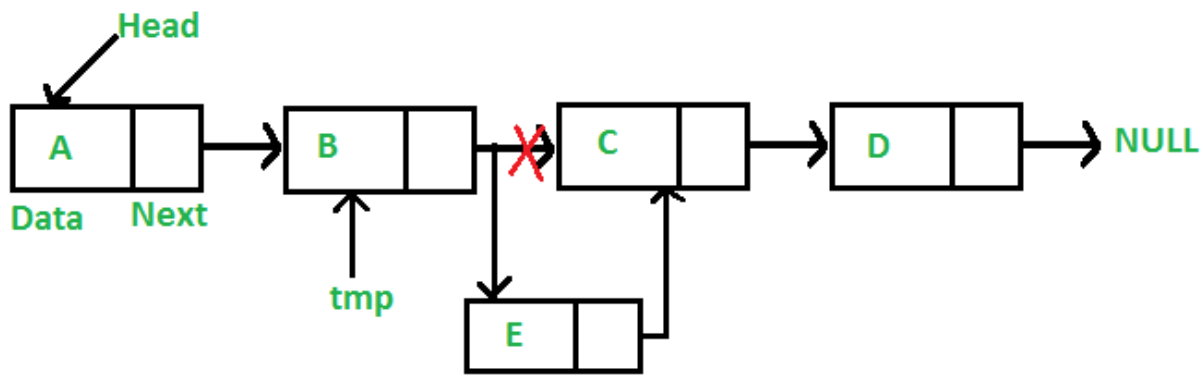


El antecesor *tmp* es "B"

Insertando nodos en el medio de la Lista

Suponga que deseamos insertar un nodo con un valor "E" después de "B":

- Debemos buscar al nodo *tmp* quién será el **antecesor** de "E" (tmp="B").
- Luego, hacemos apuntar el nuevo nodo a donde apunta *tmp*
- Y finalmente redireccionar el puntero de *tmp* hacia el nuevo nodo "E".



El antecesor *tmp* es "B"

- Cree la función void inserInList(*nodo* **ℓ, int *num*); que inserte el nodo en su posición correcta.
- Note que si inserta al comienzo, la cabecera de la lista tendrá una nueva dirección (por eso usamos *nodo* **ℓ)

Programando en clases...

En base a al fuente listas.cpp, realice los siguentes ejercicios:

1.- Cree las funciones float mediaList(*nodo** ℓ); int minList(*nodo** ℓ) e int maxList(*node** ℓ) que recorra la lista ℓ y entregue la media, el mínimo y el máximo respectivamente.

2.- Cree un procedimiento appendList(*nodo*** ℓ, *node** q), que añadirá la lista q al final de ${}^*\ell$.

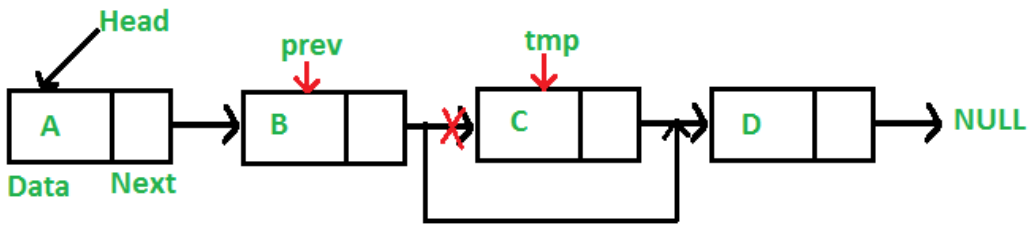
3.- Cree un procedimiento addPrevVal(*nodo* *ℓ), que recorra la lista ℓ y a cada nodo sume acumulativamente el valor del predecesor.
Ej: $\ell = 6 \rightarrow 19 \rightarrow 2 \rightarrow 8 \Rightarrow \ell = 6 \rightarrow 25 \rightarrow 27 \rightarrow 35$ (note que la lista queda creciente)

4.- Cree un procedimiento insertList(*nodo*** ℓ, *nodo* *q), donde *ℓ y q están ordenadas ascendentemente (luego de invocar al método anterior). Inserté a la lista *ℓ todos los nodos de la lista q en sus posiciones correctas.

Eliminando nodos en la Lista

Suponga que deseamos eliminar el nodo de valor "C":

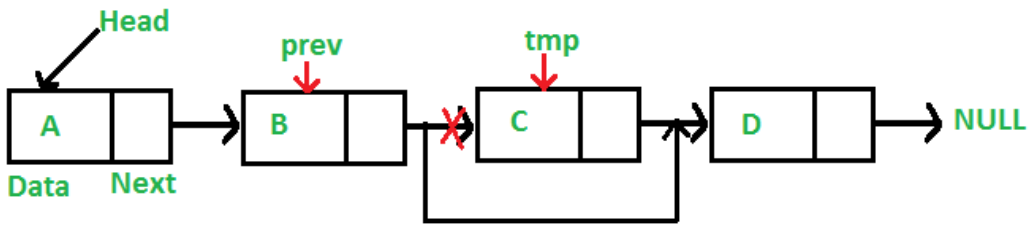
- Nuevamente, buscamos al nodo ***prev*** que es **antecesor** de "C".
- Luego, hacemos apuntar *prev* al **sucesor** de "C": *tmp->next*.
- Y finalmente eliminar el nodo *tmp* que contiene el valor "C".



Eliminando nodos en la Lista

Suponga que deseamos eliminar el nodo de valor "C":

- Nuevamente, buscamos al nodo ***prev*** que es **antecesor** de "C".
- Luego, hacemos apuntar *prev* al **sucesor** de "C": *tmp->next*.
- Y finalmente eliminar el nodo *tmp* que contiene el valor "C".

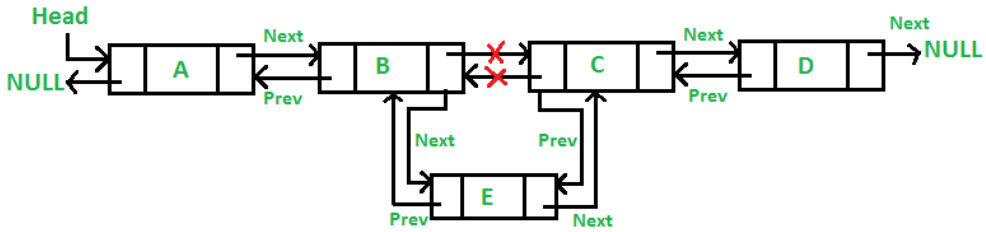


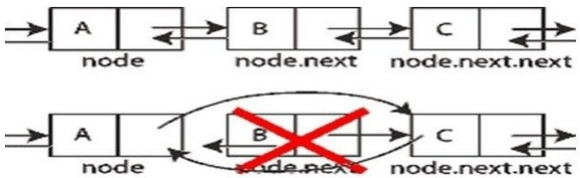
- Cree la función bool removeFromList(*nodo* **ℓ, int *num*); esta debe retornar verdadero si encontró y eliminó el nodo con valor *num*.

Listas Doblemente Enlazadas

La idea es la misma que en el caso simple, salvo que ahora cada nodo posee dos punteros: su antecesor (prev) y a su sucesor (next):

```
struct nodeDoubleList {
    int val;
    nodeDoubleList *prev;
    nodeDoubleList *next;
};
typedef struct nodeDoubleLis
```





Cree una copia de listas.cpp → listasDobles.cpp y modifique las funciones anteriores para que opere sobre una lista doblemente enlazada.

Ejercicios Propuestos

1.- Cree una lista doblemente enlazada y circular, LDC, con n enteros en el intervalo $[-n, n]$. Pida el ingreso de un entero num, búsquele en LDC, si $num \in LDC$, entonces pida el ingreso de un entero k, e imprima los k nodos previos y los k nodos siguientes al nodo num.

2.- Una permutación de n es una secuencia de n enteros $\{x_1, x_2, \ldots, x_n\}$ tal que $1 \leq x_i \leq n$, $\forall i \in [1 \ldots n]$ y $x_i \neq x_j$ $\forall i \in [1 \ldots n]$.
Ejemplo, permitaciones de $n = 10$ son: $\{3, 8, 2, 6, 9, 1, 5, 10, 2, 7, 4\}$, $\{4, 9, 5, 2, 1, 6, 3, 10, 7, 8\}$, $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, etc.
Lea n desde loa argumentos del programa y cree una lista enlazada simple que almacene una permutación aleatoria de n. Luego cree un método que imprima todas las subsecuencias crecientes y decrecientes que posee. Ejemplo, para $\{3, 8, 2, 6, 9, 1, 5, 10, 2, 7, 4\}$, tenemos las siguientes subsecuencias crecientes: $\{3, 8\}, \{2, 6, 9\}, \{1, 5, 10\}, \{2, 7\}$ y $\{4\}$. Y las siguientes decrecientes: $\{3\}, \{8, 2\}, \{6\}, \{9, 1\}, \{5\}, \{10, 2\}$ y $\{7, 4\}$