Clase 7. Listas doblemente ligadas, algoritmos recursivos para listas.

Tarea 4

Tarea 4

4.4 Genere una lista de forma recursiva con datos leidos desde la entrada estándar o archivo (No importa el tipo de dato, pueden ser numeros o strings). Un dato se separa de otro dato por espacios o tabuladores, y el fin de los datos se indica con un salto de línea. Imprima la lista hacia adelante y hacia atras. Devuelva la memoria de la lista, todo en forma recursiva.

Crear listas de forma recursiva

Recordatorio: algoritmos recursivos

Comentamos que los elementos para generar un algoritmo recursivo son:

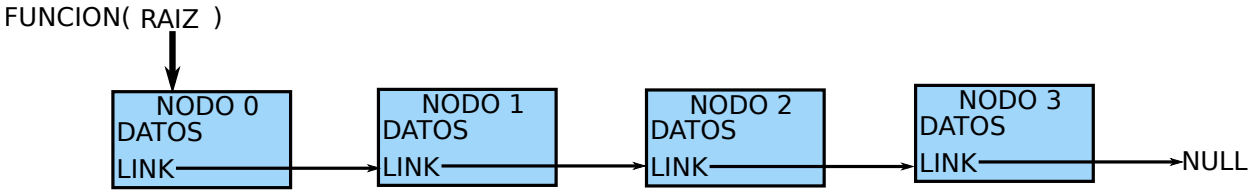
1. Un caso base.
2. Una llamada a si mismo (con un caso mas simple).
3. Cambiar el estado y moverse en direccion del caso base.

Impresion de una lista de números

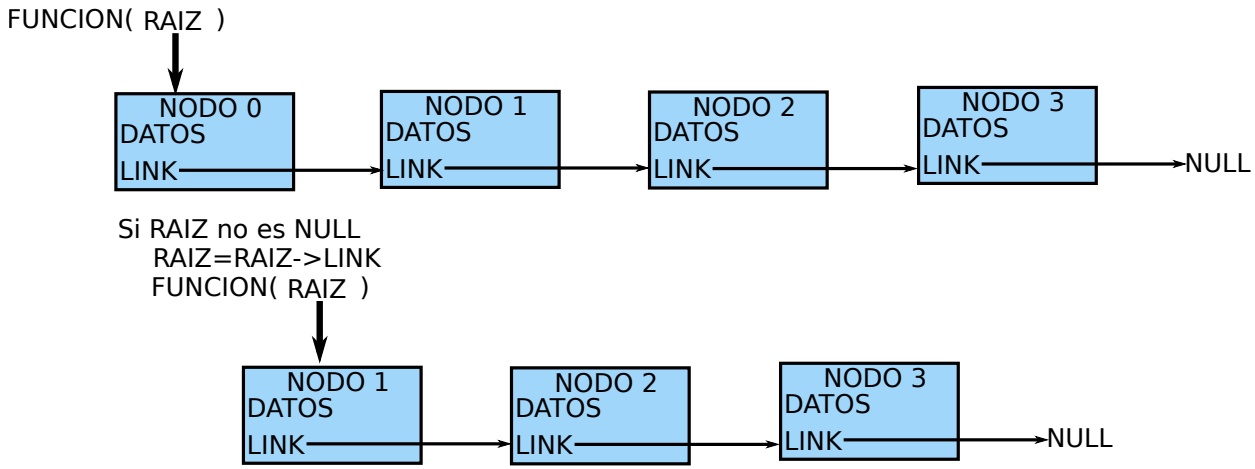
Supongamos que tenemos una lista simplemente ligada de números (enteros). Los elementos de un método de impresión de la lista sería:

1. **Caso base:** lista vacía (apuntador en NULL).
2. **Cambiar el estado y moverse en dirección de caso base: genero un lista mas pequeña.**
3. **Una llamada a si mismo:** si la lista no esta vacía el método se debe de llamar a si mismo con la lista más pequeña.

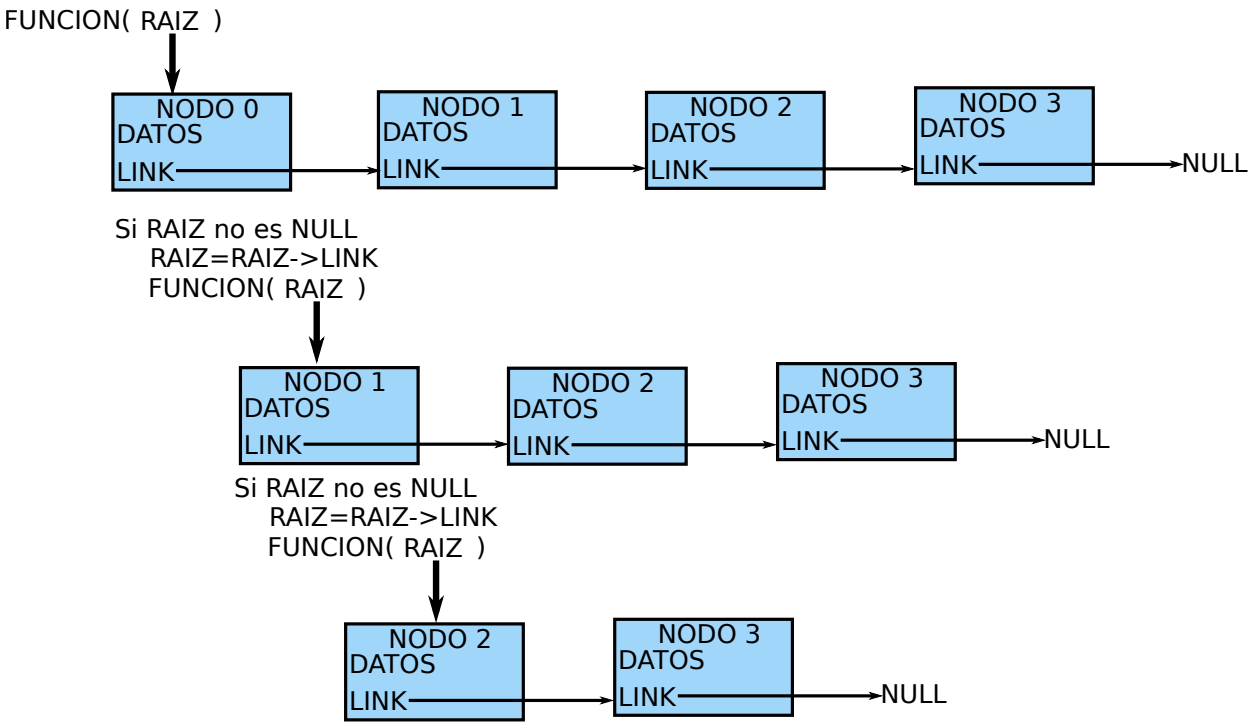
struct



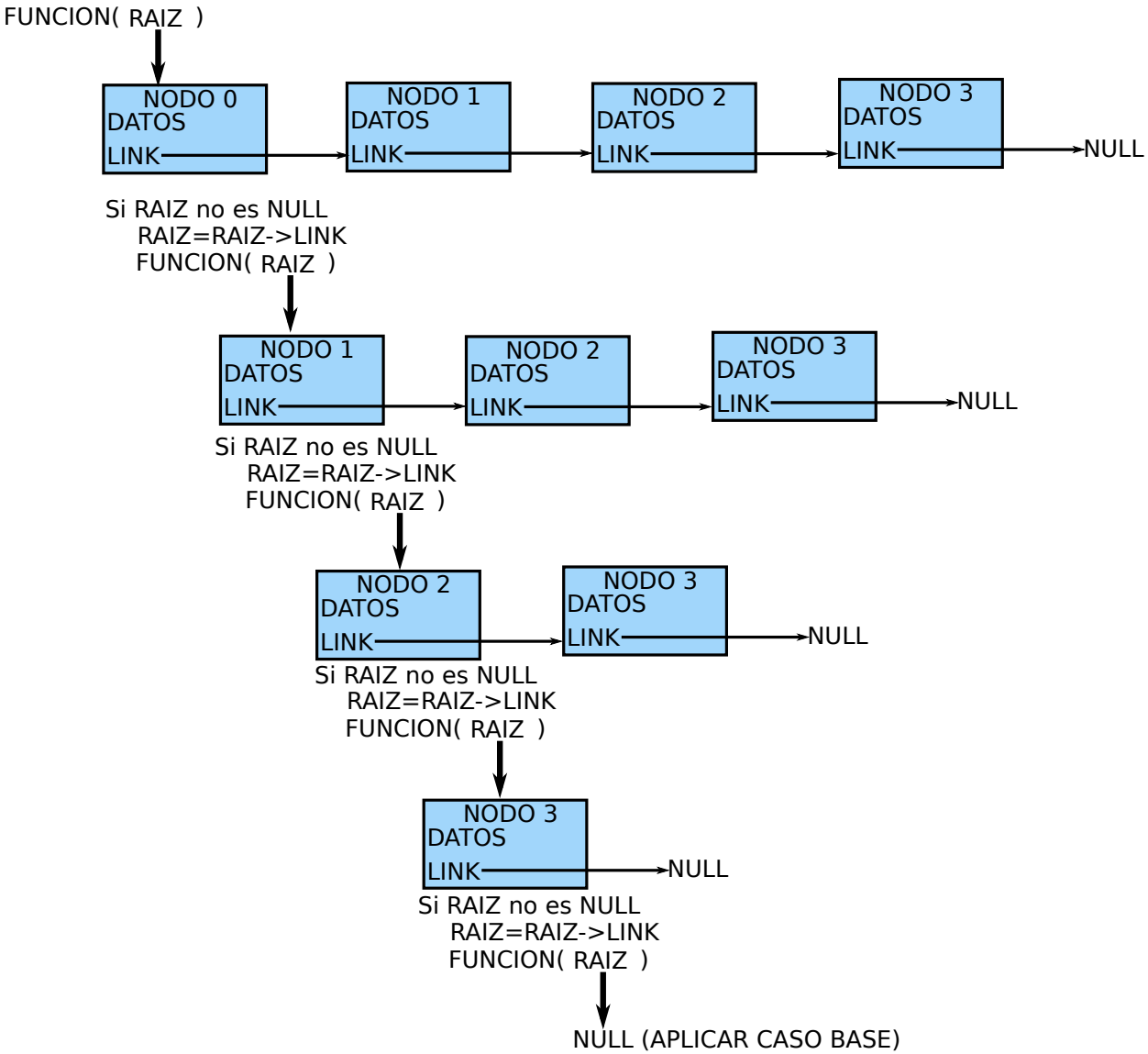
struct



struct



struct



Ejemplo función recursiva para sumar


```
1  typedef struct ESTRUCTURA{
   int dato;
3  struct ESTRUCTURA *ptr;
   }DATO;
5  typedef DATO* PTRDATO;
   int suma(PTRDATO root)
7  {
       int sum=0;
9      if (root!=NULL){
           sum=suma(root->ptr)+root->dato;
11         return sum;
       }
13 return sum;
   }
15
   int main()
17 {
    PTRDATO root=lee_lista();
19  printf("Suma de los elementos=%d\n\n",suma(root));
    borra_lista(root);
21 return 0;
   }
```

Tarea 4

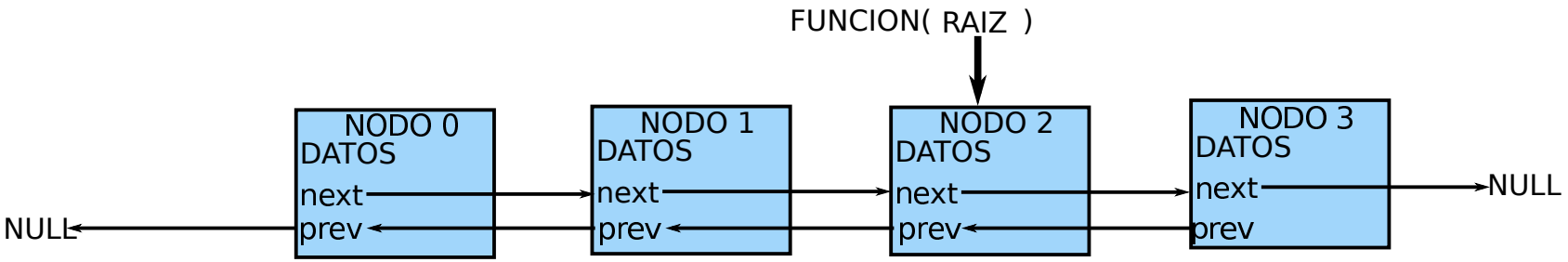
4.5 Generé una lista doblemente ligada. Los datos de la lista son números enteros de -100 a 100. La lista comienza en -100 y termina en 100 (es decir estos nodos apuntan a NULL en el antecesor y sucesor respectivamente) almacene el apuntador del nodo que contiene el 0.

1. Utilizando una función recursiva, que recibe el **solo** el apuntador a un nodo cualquiera en la lista y un valor generado aleatoriamente (pseudo-aleatorio) entre -100 y 100 (flotante), busque el nodo mas cercano por arriba si es positivo o por abajo si es negativo. Ejemplo:
   Si el aleatorio es -35.1264 encuentra el nodo con -36.
   Si el aleatorio es 12.34 encuentra el nodo con 13.
   La función devuelve el apuntador al nodo más cercano o NULL si no lo encuentra.
2. Repita este procedimiento para 100000 números y repita 30 veces. Es decir 30 veces, se generan 100000 números cada vez, se busca el nodo, y al final de los 100000 números se acumula el tiempo que se tardo el programa en hacer la búsqueda.
3. Calcule el tiempo promedio: a) Comenzando del nodo que tiene el -100, b) del que tiene el 100, c) del que tiene 0, y d) comenzando del último nodo encontrado.

Listas doblemente ligadas

Una lista doblemente ligada tiene una referencia hacia el nodo posterior y una hacia el previo. La ventaja es el acceso a cualquier nodo desde cualquier otro sin tener que comenzar siempre desde la raíz.

```
typedef struct ESTRUCTURA{
int dato;
struct ESTRUCTURA *next;
struct ESTRUCTURA *prev;
}DATO;
```



Generador de números pseudo-aleatorios

Las funciones **void srand (unsigned int seed);** y **int rand (void);** se encuentran en la libreria **stdlib.h**.
Las funciones **time_t time (time_t* timer);** y **clock_t clock (void);** están en la libreria **time.h**.


```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int mani(){
  time_t init,fint;
  clock_t inic,finc;
  srand(time(NULL)); //inicializa semilla
  init=time(NULL);
  inic=clock();
  for (int i=0; i<1000000; i++)
     int k=rand()%100;
  fint=time(NULL);
  finc=clock();
  printf("Tiempo time=%d seg, clock=%lf seg",fint-init,
     ((double)finc-(double)inic)/(double)CLOCKS_PER_SEC
    );
}
```

Salida:

```
Tiempo procesamiento time=7 seg, clock=7.320000
```

Algoritmo de Shunting-Yard

Se leen las entradas en orden:

1. Si es un número se hace un push a la pila de números.
2. Si es un operador y es menor que el top de la pila de operadores entonces, se hace pop a la pila de operadores y a la pila de números (dos números y un operador cada vez) y se aplica la operación. Al resultado se le hace push a la pila de números.
3. Si se ha terminado de leer se hace pop a la pila de operadores (y números) hasta que se vacíe la pila de operadores. El resultado queda en la único nodo restante de la pila de números.

Tarea 4

BONUS

Programe un evaluador de expresiones aritmeticas simple, que puede evaluar números y operadores tales como:
5.6+4.0*3/2.5-8.74*3^2
Utilice un tipo similar al tipo **dato** del programa 4.1, pero ahora los posibles tipos son: flotante y operador_binario. Y se necesita ademas tener la precedencia de los operadores (que se puede almacenar en donde se guarda el número flotante). Se utilizará el algoritmo de Shunting-Yard. Se guardará la dirección del top de cada una de las pilas, los nuevos datos se insertan en el top y se ligan con la pila.

EXTRA BONUS

Programe el evaluador anterior con funciones recursivas y con funciones iterativas.

Restricciones

La estructura para almacenar un dato o *token* **solo** debe de tener una variable para el valor númerico y una variable para el operador, y un indicador de si es número u operador. Para indicar un número negativo se utilizará guión bajo en lugar del signo (para que no haya confusión con el operador menos). Para indicar la precedencia del operador se utilizará la variable numérica. Solo se deben utilizar listas simplemente ligadas con operaciones de pila: push, pop, etc. Considerá que todas las expresiones que le den son validas. No utilice parentesis u otro operador diferente de los listados en la tabla de precedencia.

Algoritmo de Shunting-Yard

En pizarrón. Notas:

- El algoritmo (que implementarán) no considera parentesis, ni otros símbolos además de +,-,*,/,^y números flotantes o enteros.
- Los números enteros también debe de ser procesados como flotantes.
- Se debe de programar utilizando dos listas simplemente ligadas para el stack, de las que se almacenará solo la dirección del top.

Expresión de ejemplo: 5.6+4.0*3/2.5-8.74*3^2

Precedencia de operadores:

+	1
-	1
*	2
/	2
^	30

Referencias

time_t
http://www.cplusplus.com/reference/ctime/time/
Listas:
http://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture12.pdf
http://cslibrary.stanford.edu/105/LinkedListProblems.pdf
https://www.cl.cam.ac.uk/teaching/2001/DSAlgs/dsa.pdf