

Recorrer Estructuras Recursivas

Organización del Computador II

Mariano Cerrutti

4 de abril de 2017

Qué vimos

- Punteros.
- Vectores y Matrices.
- Structs.
- Memoria Dinámica.
- TP1

Qué vamos a ver hoy

- Recorriendo Estructuras Recursivas.
- Ejercicio

¿Qué es una estructura recursiva?

Una **estructura recursiva** es una estructura que hace referencia al menos a otra de su mismo tipo en uno de sus atributos.

Ejemplo de árbol binario

Supongamos que queremos representar un árbol binario donde cada nodo tenga un literal.

Ejemplo de árbol binario

Supongamos que queremos representar un árbol binario donde cada nodo tenga un literal.

```
struct nodo{  
    char* val;  
    nodo* hijoIzquierdo;  
    nodo* hijoDerecho;  
}
```

Ejemplo de árbol binario

Supongamos que queremos representar un árbol binario donde cada nodo tenga un literal.

```
struct nodo{  
    char* val;  
    nodo* hijoIzquierdo;  
    nodo* hijoDerecho;  
}
```

¿Y dónde está el árbol?

¿Cómo agregamos un elemento?

Ahora supongamos que tenemos un árbol binario de búsqueda (ABB), es decir, que todos los nodos que están del lado del hijo izquierdo son menores, y todos los demás están del lado del hijo derecho.

```
struct nodo{  
    char* val;  
    nodo* hijoIzquierdo;  
    nodo* hijoDerecho;  
}
```

```
struct abb{  
    nodo* raiz;  
}
```

¿Cómo agregamos un elemento?

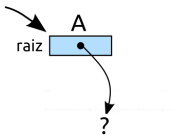
Paso a paso...

Cuando armamos estructuras recursivas a bajo nivel necesitamos manejar distintos niveles de abstracción y pensar bien los detalles.

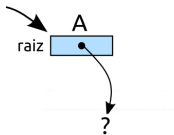
Antes de echarnos a escribir recomendamos:

- 1 Dibujar la estructura y su grafo de relaciones.
- 2 Hacer (pseudo)código en C.
- 3 Implementar en ASM siguiendo el (pseudo)código anterior.

Insertar

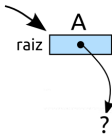


Insertar



```
void insertar( abb* A, char* valor ){  
    if( A->raiz == NULL )  
        A->raiz = nodo_crear( valor );  
    else  
        insertarNodo( A->raiz, valor );  
}
```

Insertar

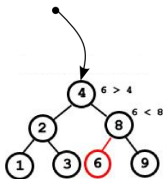


```
void insertar( abb* A, char* valor ){  
    if( A->raiz == NULL )  
        A->raiz = nodo_crear( valor );  
    else  
        insertarNodo( A->raiz, valor );  
}
```

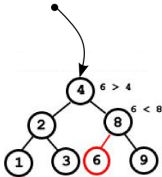
```
nodo* nodo_crear( char* valor ){  
    nodo* N = malloc( sizeof( nodo ) );  
    N->val = valor;  
    N->hijoIzquierdo = NULL;  
    N->hijoDerecho = NULL;  
    return N;  
}
```

Nota: (*A).raiz == A->raiz

¿Por dónde arrancamos? (insertarNodo)



¿Por dónde arrancamos? (insertarNodo)



```
void insertarNodo( nodo* N, char* valor ){
    nodo* actual = N;
    bool termine = false;
    while(!termine)
        if( compararStrings(valor, actual->val) < 0){
            if( actual->hijoIzquierdo == NULL )
                actual->hijoIzquierdo = nodo_crear( valor );
            termine = true;
        }
        else
            actual = actual->hijoIzquierdo;
    }
    else {
        if( actual->hijoDerecho == NULL )
            actual->hijoDerecho = nodo_crear( valor );
            termine = true;
        else
            actual = actual->hijoDerecho;
    }
}
```

Ejercicio 1:

Hacer la función insertarNodo vista anteriormente en lenguaje ensamblador

Ejercicio 1:

A tener en cuenta que las estructuras de control de flujo se fueron al cuerno. Vale hacer un control flow graph.


```
;EJERCICIO ESTRUCTURAS RECURSIVAS: ABB
```

```
;struct nodo{
;   char* valor
;   nodo* hijoDerecho
;   nodo* hijoIzquierdo
;}
```

```
OFFSET_VALOR equ 0
```

```
OFFSET_HIJODERECHO equ 8
```

```
OFFSET_HIJOIZQUIERDO equ 16
```

```
NULL equ 0
```

```
;void insertar(nodo* raiz, char* palabra)
```

```
;RDI: puntero a raiz, RSI: puntero a palabra a insertar
_inserir:
```

```
    PUSH RBP;           alineada
    MOV RBP, RSP
    PUSH R14;           desalineada
    PUSH R15;           alineada
```

```
    MOV R15, RDI        ;R15 es puntero a nodo actual
    MOV R14, RSI        ;R14 puntero a palabra por insertar
```

```
.ciclo:
```

```
    MOV RDI, R14
    MOV RSI, [R15 + OFFSET_VALOR]
    call compararStrings
```

```
JE .menor
```

```
    CMP [R15 + OFFSET_HIJODERECHO], NULL
    JE .agregar_y_terminar_derecha
```

```
    MOV R15, [R15 + OFFSET_HIJODERECHO]
    JMP .ciclo
```

```
.menor:
```

```
    CMP [R15 + OFFSET_HIJOIZQUIERDO], NULL
    JE .agregar_y_terminar_izquierda
```

```
    MOV R15, [R15 + OFFSET_HIJOIZQUIERDO]
    JMP .ciclo
```

```
.agregar_y_terminar_derecha:
```

```
    call _dameNodo
    MOV [RAX + OFFSET_VALOR], R14
    MOV [R15 + OFFSET_HIJODERECHO], RAX
    JMP .terminar
```

```
.agregar_y_terminar_izquierda:
```

```
    call _dameNodo
    MOV [RAX + OFFSET_VALOR], R14
    MOV [R15 + OFFSET_HIJODERECHO], RAX
```

```
.terminar:
```

```
    POP R15
    POP R14
    POP RBP
```

```
_dameNodo:
    PUSH RBP;           Pila alineada
    MOV RBP, RSP

    MOV RDI, 24
    call malloc
    MOV [RAX + OFFSET_HIJODERECHO], NULL
    MOV [RAX + OFFSET_HIJOIZQUIERDO], NULL

    POP RBP
    RET
```

compararStrings:

```
    PUSH RBP
    MOV RBP, RSP

.ciclo:
    MOV DL, [RDI]
    MOV CL, [RSI]

    CMP DL, CL
    JNE .seguir
    INC RDI
    INC RSI
    JMP .ciclo

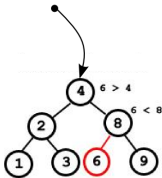
.seguir:
    CMP DL, CL
    JL .devuelvePositivo
    JMP .devuelveNegativo

.devuelvePositivo:
    MOV RAX, 1
    JMP .terminar

.devuelveNegativo:
    MOV RAX, -1

.terminar:
    POP RBP
    RET
```

El doble puntero



```
void insertarNodo( nodo* N, int valor ){
    nodo* actual = N;
    nodo **siguiente;

    while(actual != NULL){
        if( valor < actual->val )
            siguiente = &actual->hijoIzquierdo;
        else
            siguiente = &actual->hijoDerecho;

        actual = *siguiente;
    }

    *siguiente = nodo_crear( valor );
}
```

```
;EJERCICIO ESTRUCTURAS RECURSIVAS: ABB
```

```
;struct nodo{  
;   char* valor  
;   nodo* hijoDerecho  
;   nodo* hijoIzquierdo  
;}
```

```
OFFSET_VALOR equ 0  
OFFSET_HIJODERECHO equ 8  
OFFSET_HIJOIZQUIERDO equ 16  
NULL equ 0
```

```
;void insertar(nodo* raiz, char* palabra)
```

```
;RDI: puntero a raiz, RSI: puntero a palabra a insertar  
_insertar:
```

```
    PUSH RBP;           alineada  
    MOV RBP, RSP  
    PUSH R14;           desalineada  
    PUSH R15;           alineada  
    PUSH R13;           desalineada  
    SUB RSP, 8;         alineada
```

```
    MOV R15, RDI        ;R15 es puntero a nodo actual  
    MOV R14, RSI        ;R14 puntero a palabra por insertar  
    MOV R13, NULL       ;R13 puntero a puntero a siguiente
```

```
.ciclo:
```

```
    CMP R15, NULL  
    JE .agregar_y_terminar
```

```
    MOV RDI, R14  
    MOV RSI, [R15 + OFFSET_VALOR]  
    call compararStrings  
    CMP RAX, -1
```

```
    JE .menor  
    LEA R13, [R15 + OFFSET_HIJODERECHO]  
    JMP .seguir
```

```
.menor:  
    LEA R13, [R15 + OFFSET_HIJOIZQUIERDO]
```

```
.seguir:  
    MOV R15, [R13]  
    JMP .ciclo
```

```
.agregar_y_terminar:  
    call dameNodo  
    MOV [R13], RAX
```

```
.terminar:  
    ADD RSP, 8  
    POP R13  
    POP R15  
    POP R14  
    POP RBP  
    RET
```

Para pensar en casa...

La Navidad esta cerca y como todos los años, el Grinch se dispone a organizar un recorrido de la ciudad para robar regalos. El año pasado su plan se vio truncado cuando un perro le mordió su rabo, lo cual lo llevó a pasar todo el 25 con su cola vendada y recibiendo vacunas antirrábicas, por lo que este año nos pidió que lo ayudáramos con su tarea.

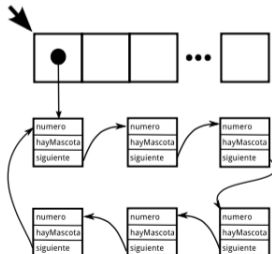
La estructura que representa una GuiaT es una arreglo de punteros a manzanas. Una manzana no es más que un grupo de casas que se apuntan circularmente.

La estructura de una casa es:

```
typedef struct casa_t {  
    short numero;  
    boolean hayMascotaGuardiana;  
    struct casa* siguiente;  
} __attribute__((__packed__)) casa;
```

Donde:

```
typedef enum boolean_e { no=0, si=1 } boolean;
```



Se pide escribir una función que dada una GuiaT elimine de las manzanas aquellas casas que tienen una mascota guardiana. El prototipo de la función es: `void filtrarGuia(casa** guiat, short n);`

- (10p) Escribir en C la función `filtrarGuia`.
- (30p) Implementar en ASM de 64 bits la función `filtrarGuia`.