Clase 10. Union. Árboles binarios. Clases de almacenamiento.

Tipos datos avanzados

union

Una **union** es un tipo de dato que puede almacenar datos de diferente tipo en la misma locación de memoria.

```
1  union dato{
     int entero;
3    float flotante;
     char caracter;
5  };
```

1. El tamaño de la unión es igual al tamaño del mayor (en memoria) de sus elementos.
2. No se verifica que se utilice el tipo correcto.

Ejemplo 1, union


```c
#include <stdlib.h>
#include <stdio.h>
union DATO{
    int entero;
    double flotante;
    char caracter;
};
int main(int argc, char *argv[])
{
  union DATO x;
  printf("Tamanio memoria=%d\n",sizeof(x));
  x.entero=5;
  printf("1 Entero=%d flotante=%f char=%c\n",x.entero,x
    .flotante,x.caracter);
  x.flotante=5.0;
  printf("2 Entero=%d flotante=%f char=%c\n",x.entero,x
    .flotante,x.caracter);
  x.caracter='5';
  printf("3 Entero=%d flotante=%f char=%c\n",x.entero,x
    .flotante,x.caracter);
return 0;
}
```

Ejemplo 2, union y typedef


```c
#include <stdlib.h>
#include <stdio.h>
typedef union{
    int entero;
    double flotante;
    char caracter;
}DATO;
int main(int argc, char *argv[])
{
  DATO x;

return 0;
}
```

Continuación estructuras de datos

Ejemplo 3, estructuras y uniones

- Note que una unión es miembro de la estructura.

Ejemplo 3, estructuras y uniones

- Note que una unión es miembro de la estructura.
- En la inicialización de la estructura se accede a la unión para inicializarlo, la unión se copia byte por byte (como la funcion memcpy), por lo tanto no importa que tipo de dato haya guardado.


```
1  typedef union{
       int entero;
3      double flotante;
       char caracter;
5  }DATO;

7  typedef struct Enodo{
       DATO x;
9      struct Enodo* sig;
   }NODO;
11

   NODO *creaNODO(DATO x){
13   NODO *nnodo=(NODO*)malloc(sizeof(NODO));
     if (nnodo){
15     nnodo->x=x;
       nnodo->sig=NULL;
17   }
   return nnodo;
```

Ejemplo 3, estructuras y uniones, continuación

- Note que se hace un cast de un entero a un tipo de la unión.

Ejemplo 3, estructuras y uniones, continuación

- Note que se hace un cast de un entero a un tipo de la unión.
- El cast inicializa debidamente (en caso de tipos básicos) la unión (pruebe con otros tipos de datos).


```
NODO *creaListaCiclo(int n){
  NODO *raiz=NULL,*tmp;
   for (int i=0; i<n; i++)
       if (raiz==NULL){
           tmp=raiz=creaNODO((DATO)i);
       }
       else{
          tmp->sig=creaNODO((DATO)i);
          tmp=tmp->sig;
       }
return raiz;
}

NODO *rmListaCiclo(NODO *raiz){
  NODO* tmp;
   for (;raiz!=NULL;tmp=raiz,raiz=raiz->sig,free(tmp));
return raiz;
}
```

Ejemplo 3, estructuras y uniones, continuación 2

- Vemos la liberación de memoria adecuadamente.

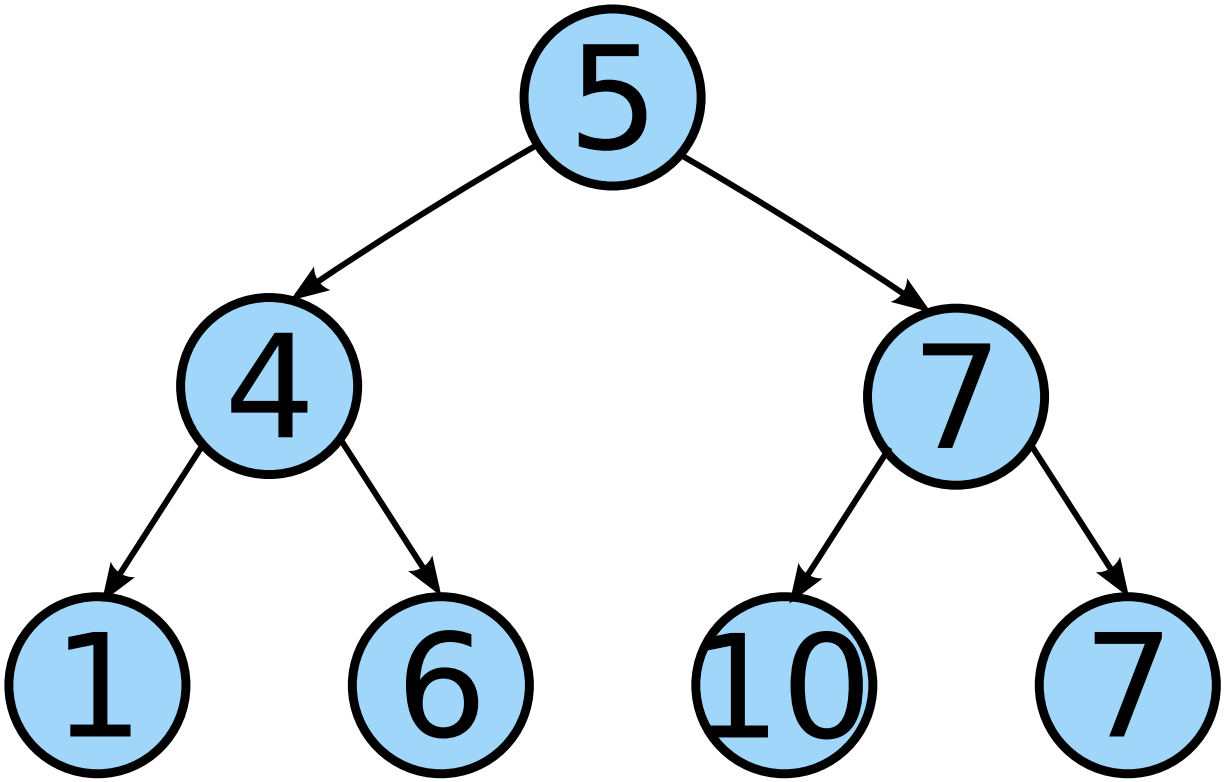

```
void printLista(NODO* raiz){
  for (;raiz!=NULL;printf("entero=%d double=%lf char=%c
    \n",raiz->x.entero, raiz->x.flotante, raiz->x.
    caracter), raiz=raiz->sig);
}

int main(int argc, char *argv[]){
  NODO *raiz=creaListaCiclo(10);
   printLista(raiz);
   raiz=rmListaCiclo(raiz);
return 0;
}
```

```
Ejecucion con valgrind:
valgrind --tool=memcheck ./eje1
Salida de Valgrind:
==29961== HEAP SUMMARY:
==29961==     in use at exit: 0 bytes in 0 blocks
==29961==   total heap usage: 10 allocs, 10 frees, 160 bytes alloca
==29961==
==29961== All heap blocks were freed -- no leaks are possible
==29961==
==29961== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 fr
```

Estructuras de datos dinámicas (continuación). Arboles binarios

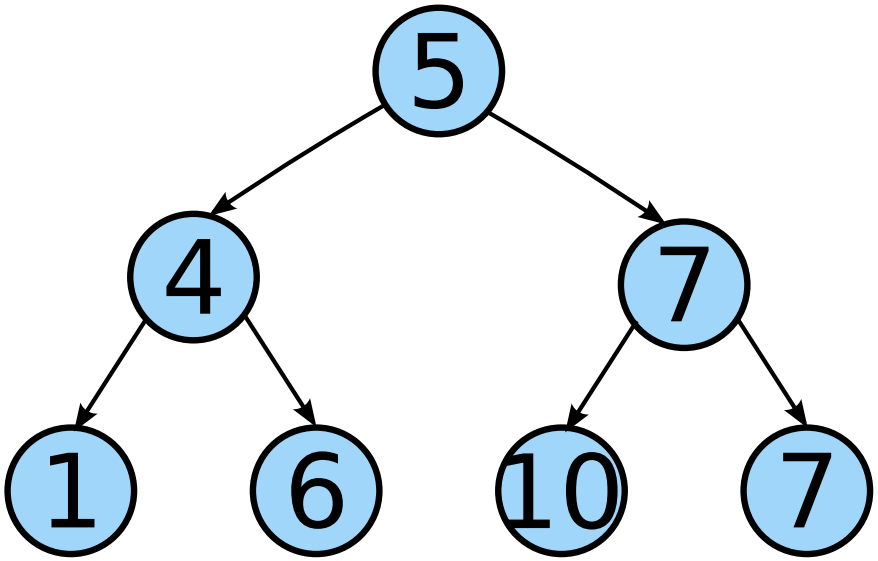
Arboles binarios, definiciones

- Un arbol binario es una estructura de datos donde cada nodo tiene a lo mx́imo dos referencias a otros nodos del mismo tipo de dato.



Árbol binario ordenado (ABO, binary search algoritm)

- En el arbol binario ordenado un nodo siempre tiene (en caso de tener) un nodo de menor valor a la izquierda y uno de mayor valor a la derecha.
- Los valores no se repiten.



Algoritmos para insertar nodos en un ABO

Algoritmo iterativo cada que se lee un dato:

1. Si la raíz está vacía insertar el nuevo dato como la raíz y **salir**.

Algoritmos para insertar nodos en un ABO

Algoritmo iterativo cada que se lee un dato:

1. Si la raíz está vacía insertar el nuevo dato como la raíz y **salir**.
2. El dato es menor que la raíz:
   (a) Si el lado izquierdo está vacío insertar el dato en el lado izquierdo.
   (b) raiz=raiz$->$izquierdo. Y regresar al paso 2. Note que está perdiendo el valor original en raíz (hay que almacenarlo en otro lado).

Algoritmos para insertar nodos en un ABO

Algoritmo iterativo cada que se lee un dato:

1. Si la raíz está vacía insertar el nuevo dato como la raíz y **salir**.
2. El dato es menor que la raíz:
   (a) Si el lado izquierdo está vacío insertar el dato en el lado izquierdo.
   (b) raiz=raiz$->$izquierdo. Y regresar al paso 2. Note que está perdiendo el valor original en raíz (hay que almacenarlo en otro lado).
3. El dato es mayor que la raíz:
   (a) Si el lado derecho está vacío insertar el dato en el lado derecho.
   (b) raiz=raiz$->$derecho. Y regresa al paso 2.

Algoritmos para insertar nodos en un ABO

Algoritmo recursivo cada que lee un dato (recibe la raiz y devuelve un apuntador):

1. Si la raíz está vacía regresar el nuevo nodo y salir.

Algoritmos para insertar nodos en un ABO

Algoritmo recursivo cada que lee un dato (recibe la raiz y devuelve un apuntador):

1. Si la raíz está vacía regresar el nuevo nodo y salir.
2. Si el dato es menor que la raíz, llamar a la misma función con raiz$->$izq como argumento y recibir el resultado en raiz$->$izq.

Algoritmos para insertar nodos en un ABO

Algoritmo recursivo cada que lee un dato (recibe la raiz y devuelve un apuntador):

1. Si la raíz está vacía regresar el nuevo nodo y salir.
2. Si el dato es menor que la raíz, llamar a la misma función con raiz$->$izq como argumento y recibir el resultado en raiz$->$izq.
3. Si el dato es mayor que la raíz, llamar a la misma función con raiz$->$der como argumento y recibir el resultado en raiz$->$der.

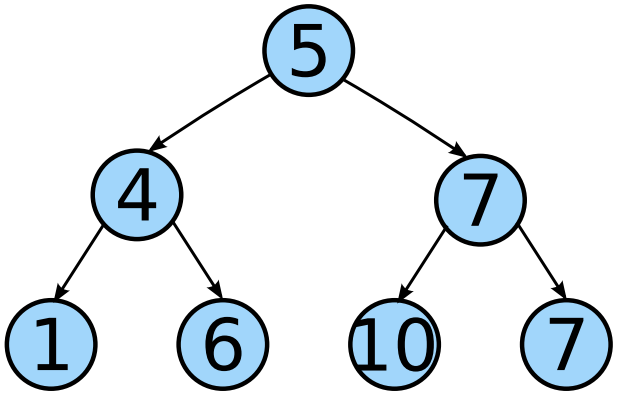
Algoritmos para insertar nodos en un ABO

Algoritmo recursivo cada que lee un dato (recibe la raiz y devuelve un apuntador):

1. Si la raíz está vacía regresar el nuevo nodo y salir.
2. Si el dato es menor que la raíz, llamar a la misma función con raiz$->$izq como argumento y recibir el resultado en raiz$->$izq.
3. Si el dato es mayor que la raíz, llamar a la misma función con raiz$->$der como argumento y recibir el resultado en raiz$->$der.
4. Salir y regresar raiz.

Tarea 5

BONUS Programar un método (iterativo o recursivo) cree un arbol insertando nodos con números aleatorios, con (alguno) de los algoritmos vistos en esta clase (iterativo o recursivo) y que dado un arbol binario, eje:



Imprima en la consola algo como:

```
      5
   4      7
 1  6   10  7
```

Nota: requerir y devolver la memoria adecuadamente.

EXTRABONUS Programar tanto el iterativo como el recursivo y programar una función que devuelva la altura del arbol.

Palabras reservadas: Clase de almacenamiento

auto, register, static, extern

- **auto** El especificador de clase de almacenamiento **auto** es el default para todas las variables locales, lo único que indica es que la variable se alloca automáticamente e igual se dealloca (requerir y devolver memoria) automáticamente.
- **register** Es un especificador que “sugiere” al sistema almacenar este dato en los registros del procesador, que es una memoria muy limitada pero de acceso muy rápido.

- **static** El especificador índica que la variable local de este tipo almacena el último valor de la variable tomado en la función y lo recupera en la siguiente llamada.
- **extern** Indica que una variable declarada en un archivo .c será utilizada en el archivo donde se declare extern.

Ejemplo register

Como los registros son pocos, y lo que se quiere minimizar es el acceso a la RAM o cache, los register se utilizan en contadores, o variables muy utilizadas en una sección de código.

```
for (register int i=0; i<10; i++);
```

Ejemplo extern

Puedo declarar variables globales en el main, y utilizarlas en otro archivo .c diferente sabiendo que es la misma variable.

```c
#include "external.h"
int var=8;
int main(int argc, char *argv[])
{
    var=5;
    cambiaValor();
    printf("var=%d\n",var);
return 0;
}
```

En otro archivo .c

```c
#include "external.h"
extern int var;
int cambiaValor(){
  printf("Valor actual=%d\n",var);
  var=19;
}
```

Ejemplo static

La inicialización solo se aplica la primera vez que entra a la función, de la segunda en adelante, la variable ya se ha declarado e inicializado, solo se aumenta su valor.

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

typedef struct Enode{
  int count;
  float x, y;
}NODO;

int funcion(){
    static NODO x={0,3.5,-6.8};
    printf("Ha entrado %d veces, x=%lf y=%lf\n",x.count
    , x.x,x.y);
return 0;
}
int main(int argc, char *argv[])
{
    for (int i=0; i<10; i++)
        funcion();
return 0;
}
```