UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

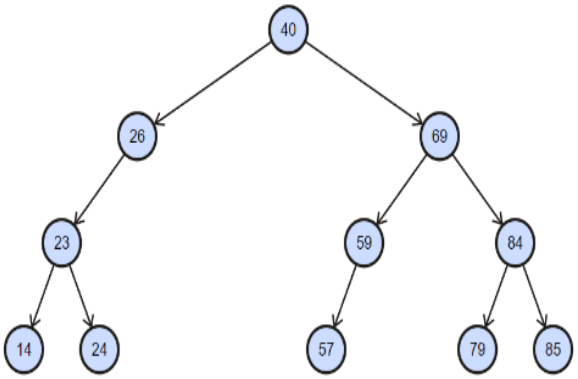
Ejercicio 1

Sea el siguiente código:

```
typedef struct nodoA {
    int dato;
   struct nodoA* izq;
  struct nodoA* der;
}nodoA;


int hacerAlgo(nodoA* A)
{
 if (A)
  {
    if (A->izq != NULL && A->der == NULL)
      return (A->dato %2 == 0) + hacerAlgo(A->izq);
   else
       if (A->der != NULL && A->izq == NULL)
            return (A->dato %2 == 0 ) + hacerAlgo(A->der);
       else
            return hacerAlgo(A->izq) + hacerAlgo(A->der);
  }
else
  return 0;
}
```

y el siguiente Árbol:



Responda EXACTAMENTE el valor que devuelve la función

(SOLO EL NUMERO) : …4…. (No cuenta el nodo root)

Ejercicio 2

A) Explique con sus propias palabras el concepto y DIFERENCIAS entre Árbol, Árbol Binario y Árbol Binario de Búsqueda.

Árbol: Un árbol en estructuras de datos es una colección de nodos que están organizados jerárquicamente. Cada nodo tiene un valor y cero o más nodos hijos, que están conectados mediante enlaces o aristas. El nodo en la parte superior se llama "raíz", y los nodos que no tienen hijos se llaman "hojas". Los nodos que comparten un ancestro común forman una "subárbol". Los árboles se utilizan para representar estructuras jerárquicas de manera eficiente.

Árbol Binario: Un árbol binario es un tipo especial de árbol en el que cada nodo tiene como máximo dos nodos hijos, generalmente denominados "izquierdo" y "derecho". Cada nodo puede tener, a lo sumo, un nodo padre. Este tipo de estructura permite una fácil implementación y búsqueda eficiente.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Árbol Binario de Búsqueda (BST - Binary Search Tree): Un árbol binario de búsqueda es un tipo específico de árbol binario en el que para cada nodo, todos los nodos en su subárbol izquierdo tienen valores menores que el nodo, y todos los nodos en su subárbol derecho tienen valores mayores. Esta propiedad facilita la búsqueda eficiente de elementos en el árbol, ya que se puede realizar una búsqueda binaria. Los árboles binarios de búsqueda son útiles en la implementación de estructuras de datos como conjuntos y mapas.

En resumen, la diferencia clave radica en las restricciones impuestas en la relación entre los valores de los nodos y sus posiciones relativas en el árbol:

- **Árbol general:** Cualquier número de nodos hijos.
- **Árbol Binario:** Máximo dos nodos hijos por nodo.
- **Árbol Binario de Búsqueda:** Estructura organizada de manera que los nodos en el subárbol izquierdo tienen valores menores, los nodos en el subárbol derecho tienen valores mayores.

B) Explique con sus propias palabras cuál es el criterio de inserción de los datos en un árbol Binario de Búsqueda

En un Árbol Binario de Búsqueda (BST), cuando se quiere agregar un nuevo número:

1. Comparamos el número con el que está en la raíz.
2. Si es menor, vamos al lado izquierdo; si es mayor, vamos al lado derecho.
3. Repetimos el proceso hasta encontrar un espacio donde no haya otro número en la dirección que estamos mirando.
4. Colocamos el nuevo número allí.

La idea es organizar el árbol de manera que los números más pequeños estén a la izquierda y los más grandes a la derecha, facilitando búsquedas rápidas.

C) ¿Cuál es el beneficio de trabajar con Arboles Binarios de Búsqueda?

Trabajar con Árboles Binarios de Búsqueda (BST) tiene varios beneficios, y uno de los más importantes es facilitar y acelerar la búsqueda de datos. Aquí hay una explicación sencilla:

Imagina que tienes una lista de números organizados de una manera especial. En esta lista, si estás buscando un número, puedes decidir rápidamente si debes buscar a la izquierda o a la derecha en función de si el número es más pequeño o más grande que el número en el medio. Luego, repites este proceso en la sección más pequeña.

En los BST, cada nodo tiene un "número" y sigue esta regla. Todos los números a la izquierda son más pequeños, y todos los números a la derecha son más grandes. Esto hace que buscar un número sea como encontrar rápidamente un lugar en un libro sabiendo si estás en la mitad correcta.

En resumen, los Árboles Binarios de Búsqueda nos permiten organizar datos de una manera que facilita encontrar información rápidamente, lo que es muy útil cuando estamos buscando cosas en grandes conjuntos de datos.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 3

Tenemos una lista de listas en la cual:

- en la lista principal hay equipos de trabajo
- cada equipo tiene una lista de personas que lo integran

Dada la siguiente función, responda:

a)qué hace cada una de sus líneas.

Explicación de cada línea:

1. `nodoSubLista *nuevoNodo = crearNodo(p);`: Se crea un nuevo nodo para la lista de personas (`nodoSubLista`) utilizando la función `crearNodo` y se le asigna el valor de la persona `p`.
2. `nodoListaPpal *eqEncontrado = buscar(listaPpal, e);`: Se busca en la lista principal (`listaPpal`) el equipo (`e`) utilizando la función `buscar`. El resultado se almacena en `eqEncontrado`.
3. `if(eqEncontrado == NULL) {...}`: Se verifica si el equipo fue encontrado. Si `eqEncontrado` es `NULL`, significa que el equipo no está en la lista principal.
4. `nodoListaPpal *nuevoNodoppal = crearNodoEquipo(e);`: Se crea un nuevo nodo para el equipo (`nodoListaPpal`) utilizando la función `crearNodoEquipo` y se le asigna el valor del equipo `e`.
5. `listaPpal = agregarEqPpio(listaPpal, nuevoNodoppal);`: Se agrega el nuevo nodo del equipo al principio de la lista principal utilizando la función `agregarEqPpio`.
6. `eqEncontrado->subLista = agregarPersPpio(eqEncontrado->subLista, nuevoNodo);`: Se agrega el nuevo nodo de la persona al principio de la sublista del equipo encontrado utilizando la función `agregarPersPpio`.
7. `return listaPpal;`: Se devuelve la lista principal actualizada.

b)si funciona correctamente o no, y en caso de haber algún error, explique cuál es.

Evaluación de su funcionamiento:

La función parece diseñada para agregar una persona a un equipo en una estructura de lista de listas. Sin embargo, hay un problema si el equipo no se encuentra en la lista principal (`eqEncontrado` es `NULL`). En ese caso, se crea un nuevo nodo para el equipo y se agrega al principio de la lista principal, pero luego se intenta acceder a la sublista del equipo (`eqEncontrado->subLista`) y si antes fue NULL, entonces se esta queriendo acceder a algo NULL, habría que cambiar dentro del if y en vez de crear otro nodoListaPpal, trabajar con el mismo eqEncontrado para que luego se actualice y se pueda acceder.

```
nodoListaPpal* alta (nodoListaPpal*listaPpal, stEquipo e, stPersona p){
    nodoSubLista* nuevoNodo = crearNodo(p);
    nodoListaPpal* eqEncontrado = buscar(listaPpal,e);

    if(eqEncontrado==NULL){
        nodoListaPpal*nuevoNodoppal = crearNodoEquipo(e);
        listaPpal = agregarEqPpio(listaPpal, nuevoNodoppal);
    }

    eqEncontrado->subLista = agregarPersPpio(eqEncontrado->subLista, nuevoNodo);

return listaPpal;
}
```

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 4

Suponiendo que las funciones crearNodo() y agregarAlPrincipio() ya se encuentran codificadas, y dada la siguiente función:

```
void misterio (nodoA* a, nodoL* l)
{
    nodoL* aux= NULL;

    if (a != NULL) {
        l= misterio(a->izq, l);

        aux= crearNodo(a->dato);
        l= agregarAlPrincipio(l, aux);

        l= misterio(a->der, l);
    }
}
```

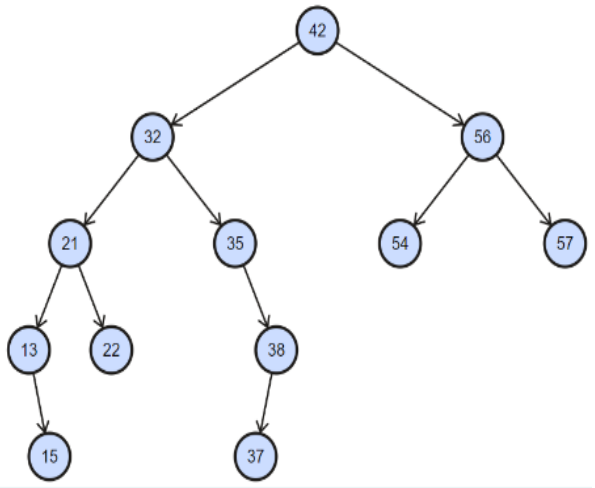
Elegir TODAS las opciones correctas (pensar bien, las respuestas incorrectas restan puntos).

a. La función pasa datos de un árbol a una lista agregándolos al principio de la lista. **X**
b. La función debería ser de tipo nodoL y retornar la lista. **X**
c. La función pasa datos de un árbol a una lista agregándolos al principio de la lista y generando así una lista ordenada en orden creciente.
d. Ninguna de las opciones restantes es correcta.
e. La función pasa datos de una lista a un árbol.
f. La función debería ser de tipo nodoA y retornar el árbol.
g. La función pasa datos de un árbol a una lista agregándolos al principio de la lista y generando así una lista ordenada en orden decreciente.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 5

Analice el siguiente árbol binario de búsqueda y responda:



Nivel en el que se encuentra situado el valor 22	3
Altura del árbol	5
Si se inserta el valor '20', que clave contiene el nodo que seria su padre?	15
Cantidad de nodos de grado 0	5
Cantidad de niveles:	4
Nivel máximo del subarbol que tiene a 35 como raíz	2
altura del subarbol derecho a la raíz	2

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 6

Tenemos una lista de árboles en la cual:

- en la lista hay géneros de películas
- cada género tiene un árbol de películas que pertenecen al mismo

Dada la siguiente función, responda (puede haber más de una respuesta correcta, pensar bien ya que las respuestas incorrectas restan puntos)

```
nodoLista* alta (nodoLista *lista, stGenero g, stPeli p){
    nodoArbol* nuevo = crearNodoA(p);
    nodoLista* encontrado = buscar(lista,g);

    if(encontrado==NULL){
        nodoLista* nuevo = crearNodoL(g);
        lista = agregarAlPrincipio(lista, nuevo);
        lista->arbol = insertar(lista->arbol, nuevo);
    }
    else{
        encontrado->arbol = insertar(encontrado->arbol, nuevo);
    }

return lista;
}
```

a. La función compila pero fallará en su ejecución.

b. La función no compila. **X**

c. La función tiene errores al momento de agregar las películas. **X**

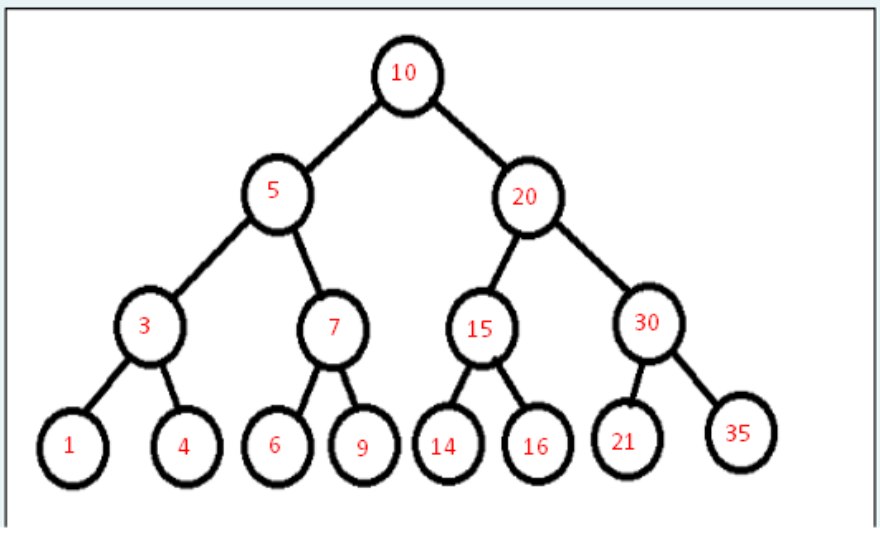
d. La función es correcta, no tiene ningún error.

e. La función tiene errores al momento de agregar los géneros.

Ejercicio 7

Inserte los siguientes datos en el nodo que corresponda del siguiente Árbol Binario de Búsqueda (respetando el orden de aparición de los datos en el siguiente listado de números).

10 - 5 - 7 - 20 - 3 - 6 - 15 - 4 - 1- 30 - 14 - 21 - 16 - 35 - 9



UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 8

Dada la siguiente función…

```
nodoArbol* haceAlgo(nodoArbol* arbol)
{
    if(arbol)
    {
        if(arbol->der != NULL)
        {
            arbol = haceAlgo(arbol->der);
        }
    }
    return arbol;
}
```

… seleccionar LA respuesta correcta.

a. La función busca y retorna el nodo que se encuentra más a la derecha del árbol. **X**
b. La función busca y retorna el nodo con el dato mayor que se encuentra en el árbol.
c. Es un recursividad infinita
d. Ninguna de las restantes respuestas es correcta.

Explicación:

- La función verifica si el nodo actual (`arbol`) existe (`if(arbol)`).
- Luego, comprueba si hay un hijo derecho (`if(arbol->der != NULL)`).
- Si hay un hijo derecho, se llama recursivamente a `haceAlgo(arbol->der)`, lo que significa que la función se mueve hacia el hijo derecho.
- Esto se repite hasta que no hay más hijos derechos, y en ese caso, el último nodo visitado (el más a la derecha) se devuelve.

Entonces, la función busca y retorna el nodo que se encuentra más a la derecha del árbol.

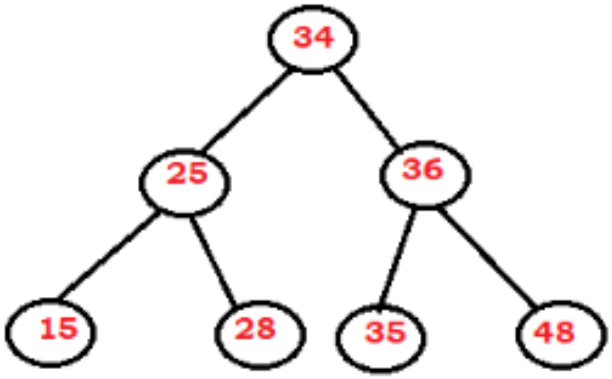
Las otras opciones no son correctas:

- La opción b no es precisa, ya que la función no retorna el nodo con el dato mayor, sino el nodo más a la derecha.
- La opción c es incorrecta, ya que la función no presenta recursión infinita; eventualmente, llegará al nodo más a la derecha y finalizará.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 9

Dado el siguiente Árbol Binario de Búsqueda:



Escribir el orden en que se mostrarían los datos en cada uno de los tres tipos de recorrido de un árbol.

PREORDER: **34**-25-15-28-36-35-48 *(**RAIZ**-IZQ-DER)*
INORDER: 15-25-28-**34**-35-36-48 *(IZQ-**RAIZ**-DER)*
POSTORDER: 15-28-25-35-48-36-**34** *(IZQ-DER-**RAIZ**)*

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 10

Dado el siguiente código:

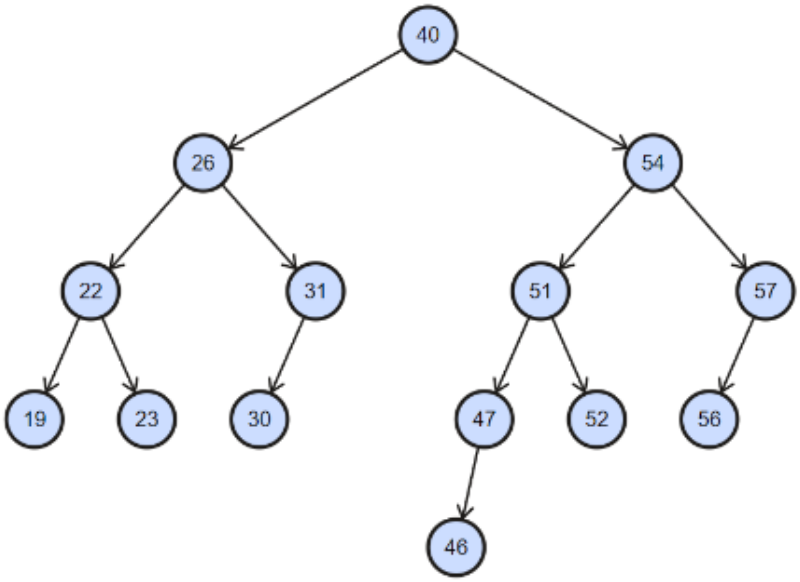
```
int algo(Arbol A)

{
    if (A)
    {
        return (A->der!=NULL) + algo(A->izq) + algo(A->der);
    }

    return 0;

}
```

el siguiente árbol:



indique que resultado retornara la función: ….4……

La lógica parece estar relacionada con contar la cantidad de nodos en el árbol que tienen un hijo derecho no nulo. El valor devuelto por la función sería la suma de:

- 1 si el nodo actual tiene un hijo derecho no nulo (`A->der != NULL`)
- La cantidad de nodos en el subárbol izquierdo (`algo(A->izq)`)
- La cantidad de nodos en el subárbol derecho (`algo(A->der)`)

En resumen, la función parece calcular algún tipo de recuento relacionado con la presencia de hijos derechos no nulos en un árbol.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 11: Indica si las siguientes afirmaciones son Verdaderas (V) o Falsas (F) en relación a las estructuras de filas

- Una fila es una estructura de datos que almacena elementos de manera lineal y permite el acceso a sus elementos mediante un índice. **V**
- En una fila, el primer elemento añadido es el primero en ser eliminado, siguiendo el principio "Primero en entrar, primero en salir" (FIFO). **V**
- Para una fila normalmente se elije su implementación mediante listas dobles ya que se puede eliminar un nodo cualquiera de manera mas sencilla que si estuviese implementada con listas simples. **V**
- Para implementar una fila en C, se puede utilizar una lista enlazada simple o doble, pero no arreglos. **F**

Falso (F): Para implementar una fila en C, se puede utilizar tanto una lista enlazada simple como doble, pero también es posible implementar una fila utilizando arreglos. Sin embargo, la implementación con listas enlazadas suele ser más flexible en términos de manejo dinámico de la memoria.

UTN – FR Mar del Plata – TUP ***Programación II – Segundo Parcial*** ***Comisión …. - Tema B*** Fecha:	***Nombre y Apellido***	***Nota***

Ejercicio 12: Dadas las siguientes funciones, si lo considera necesario, corrija su codigo.

```
struct Nodo {
   int dato;
   struct Nodo* siguiente;
};

struct Fila {
   struct Nodo* frente;
   struct Nodo* final;
};

void encolar(struct Fila* fila, int dato) {
   struct Nodo* nuevoNodo = (struct Nodo*)malloc(sizeof(struct Nodo));
   if (nuevoNodo == NULL) {
      printf("Error: no se pudo asignar memoria para el nuevo nodo.\n");
      exit(1);
   }
   nuevoNodo->dato = dato;
   nuevoNodo->siguiente = NULL;

   if (fila->final == NULL) {
      fila->frente = nuevoNodo;
      fila->final = nuevoNodo;
   } else {
      fila->final->siguiente = nuevoNodo;
      fila->final = nuevoNodo;
   }
}

int desencolar(struct Fila* fila) {
   if (fila->frente == NULL) {
      printf("La fila está vacía\n");
      return -1;
   }

   struct Nodo* temp = fila->frente;
   int dato = temp->dato;
   fila->frente = temp->siguiente;

   if (fila->frente == NULL) {
      fila->final = NULL;
   }

   free(temp);
   return dato;
}

int buscar(struct Fila* fila, int objetivo) {
   struct Nodo* actual = fila->frente;
   while (actual != NULL) {
      if (actual->dato == objetivo) {
         return 1;  /// Se encontró el elemento en la fila.
      }
      actual = actual->siguiente;
   }
   return 0;  /// El elemento no se encontró en la fila.
}
```

No veo necesidad de hacer cambios ni modificaciones.