(Comisión 3) Programación y Laboratorio 2

Comenzado el	Wednesday, 9 de November de 2022, 08:35
Estado	Finalizado
Finalizado en	Wednesday, 9 de November de 2022, 09:40
Tiempo empleado	1 hora 4 minutos
Calificación	**81,34** de 100,00

Navegación por el cuestionario

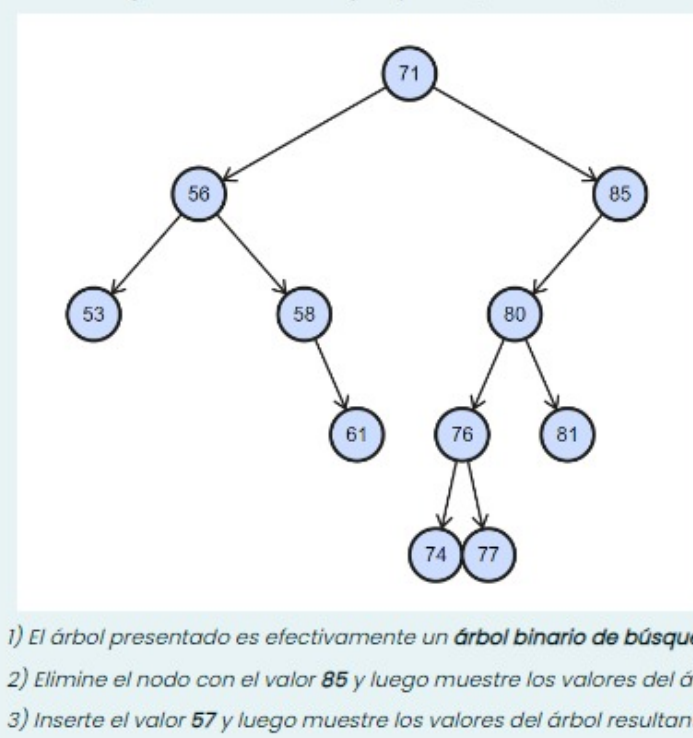
1 2 3 4 5 6 7 8 9 10 11 12 13

Mostrar una página cada vez

Finalizar revisión

Pregunta **1**
Finalizado
Se puntúa 9,00 sobre 10,00
Marcar pregunta

Analice el siguiente arbol binario y responda (Sin codificar):



*1) El árbol presentado es efectivamente un **árbol binario de búsqueda?** Justifique su respuesta.*

*2) Elimine el nodo con el valor **85** y luego muestre los valores del árbol resultante mediante un recorrido **preorden**.*

*3) Inserte el valor **57** y luego muestre los valores del árbol resultante mediante un recorrido* **posorden**. (No tenga en cuenta la eliminación del nodo realizada en el punto 2, utilice el árbol de la imagen)

1) Si, efectivamente es un arbol binario. Se sabe porque cada nodo tiene 0, 1 o 2 hijos: caracteristica fundamental de los arboles binarios.

2) Hago de cuenta que el nodo 85 se borro y que el subarbol derecho de la raiz empieza en el nodo con valor 80.
Valores que mostraria perorden: 71 - 56 - 53 - 58 - 61 - 80 - 76 - 74 - 77 - 81

3) 53 - 57 - 61 - 58 - 56 - 74 - 77 - 76 - 81 - 80 - 85 - 71

1) Si, todos los valores del subarbol izquierdo son menores a la raíz y todos los valores del subarbol derecho son mayores a la raíz, y esto se repite para cada subarbol de éstos sucesivamente.

2) Existe solo 1 criterio para eliminar en este caso.

Recorrido preorden resultante : 71-56-53-58-61-80-76-74-77-81

3) Recorrido posorden resultante : 53-**57**-61-58-56-74-77-76-81-80-85-71

Comentario:
1) incompleto, se preguntaba por ABB, no solo por ABinario
2) y 3) OK

Pregunta **2**
Parcialmente correcta
Se puntúa 2,40 sobre 6,00
Marcar pregunta

Relacione cada uno de los siguientes fundamentos de la Programación Orientada a Objetos con su definición

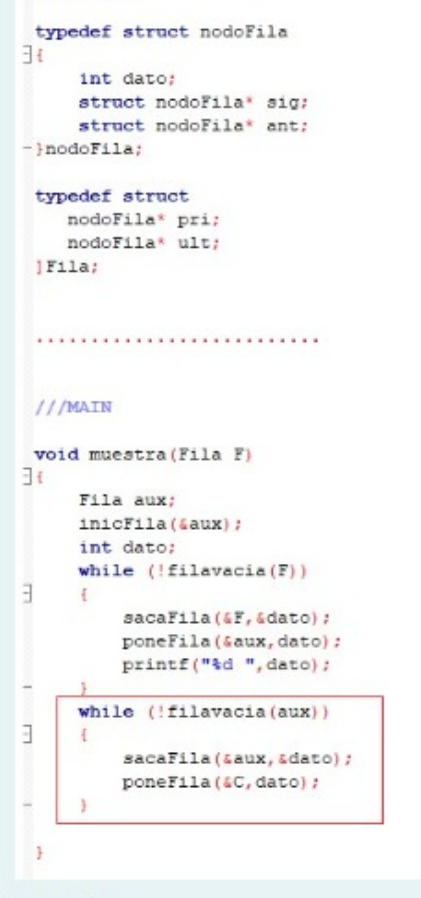
Modularidad	Consiste en dividir un programa en partes, que pueden ser compilados separadamente, pero que tienen conexiones con el exterior.	✓
Polimorfismo	Propiedad que permite que un objeto adquiera múltiples formas.	✗
Abstracción	Es la propiedad del que permite ocultar al mundo exterior la representación interna del objeto.	✗
Encapsulamiento	Propiedad por la cual un objeto puede crearse	✗
Herencia	Proceso mediante el cual un objeto de una clase adquiere propiedades definidas en otra clase que lo preceda en una jerarquía de clasificaciones.	✓

Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.
La respuesta correcta es: Modularidad → Consiste en dividir un programa en partes, que pueden ser compilados separadamente, pero que tienen conexiones con el exterior., Polimorfismo → Permite que un método tenga múltiples implementaciones., Abstracción → Expresa las características esenciales de un objeto, las cuales distinguen al objeto de los demás, dejando de lado detalles., Encapsulamiento → Es la propiedad del que permite ocultar al mundo exterior la representación interna del objeto., Herencia → Proceso mediante el cual un objeto de una clase adquiere propiedades definidas en otra clase que lo preceda en una jerarquía de clasificaciones.

Pregunta **3**
Finalizado
Se puntúa 2,00 sobre 10,00
Marcar pregunta

Dado el siguiente código que corresponde a una porción de código del archivo "Fila.h" y otra que corresponde al "main.c", y suponiendo que todas las funciones relacionadas a la fila implementada con listas fueron desarrolladas en sus librerías correspondientes...



```
///FILA.H

typedef struct nodoFila
{
    int dato;
    struct nodoFila* sig;
    struct nodoFila* ant;
}nodoFila;

typedef struct
   nodoFila* pri;
   nodoFila* ult;
}Fila;

.........................

///MAIN

void muestra(Fila F)
{
    Fila aux;
    inicFila(&aux);
    int dato;
    while (!filavacia(F))
    {
        sacaFila(&F,&dato);
        poneFila(&aux,dato);
        printf("%d ",dato);
    }
    while (!filavacia(aux))
    {
        sacaFila(&aux,&dato);
        poneFila(&C,dato);
    }
}
```

Responder

a) Si luego de invocar a la función "muestra" se desea seguir trabajando con los datos de la fila, y teniendo en cuenta que dicha función no trabaja por referencia, ¿es necesario realizar lo que figura en el recuadro rojo? Justificar la respuesta.

b) ¿Existe alguna diferencia si la fila recibida por parámetro es por referencia o solo por valor? Justificar.

a) No es necesario, la figura en el cuadrado rojo ademas esta cargando una Fila "C" que no existe. Pero incluso si volviera a cargar la fila F, es innecesario. La fila original no es modificada en ningun momento porque se trabaja por copia.

b) Si, si fuera por referencia, al usar este metodo -de vaciar y volver a cargar la fila- (muy inefectivo en mi opinion, seria mejor mostrar la lista partiendo desde el nodo fin), se debe volver a cargar la Fila (recuadro rojo, cargando "F", no "C") para que la Fila no se quede vacia y se pierda.

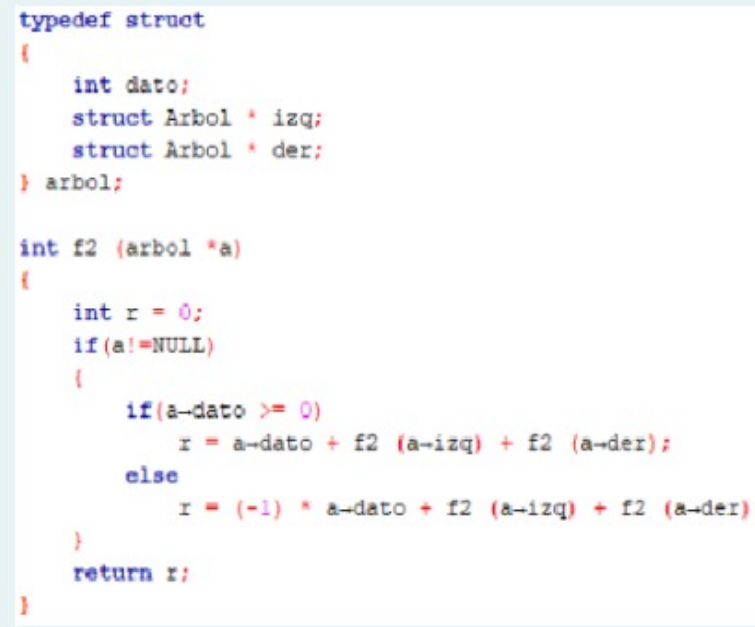
a) Se debe realizar ya que al extraer datos de una fila se eliminan nodos de la lista con la cual está implementada la fila (la función de extraer de la fila elimina el 1er nodo de la lista y retorna el dato que éste contiene), por lo tanto estas referencias ya no existen y deben crearse nuevas (es decir, nuevos nodos). Es decir: aunque la fila como estructura no sea modificada por usar un parámetro por copia, la lista con la cual está implementada la fila utiliza punteros , y las funciones de lista invocadas modifican dicha lista.

b) La diferencia es que cuando se recibe por referencia la fila, los valores de los campos pri y ult (los punteros al inicio y al final de la fila) serán alterados en la fila original, y cuando se recibe por copia no. Pero en ambos casos los nodos de la lista son eliminados, y en ambos casos el while del recuadro es necesario

Comentario:

Pregunta **4**
Correcta
Se puntúa 6,00 sobre 6,00
Marcar pregunta

¿Qué hace la siguiente función?



```
typedef struct
{
    int dato;
    struct Arbol * izq;
    struct Arbol * der;
} arbol;

int f2 (arbol *a)
{
    int r = 0;
    if(a!=NULL)
    {
        if(a→dato >= 0)
            r = a→dato + f2 (a→izq) + f2 (a→der);
        else
            r = (-1) * a→dato + f2 (a→izq) + f2 (a→der);
    }
    return r;
}
```

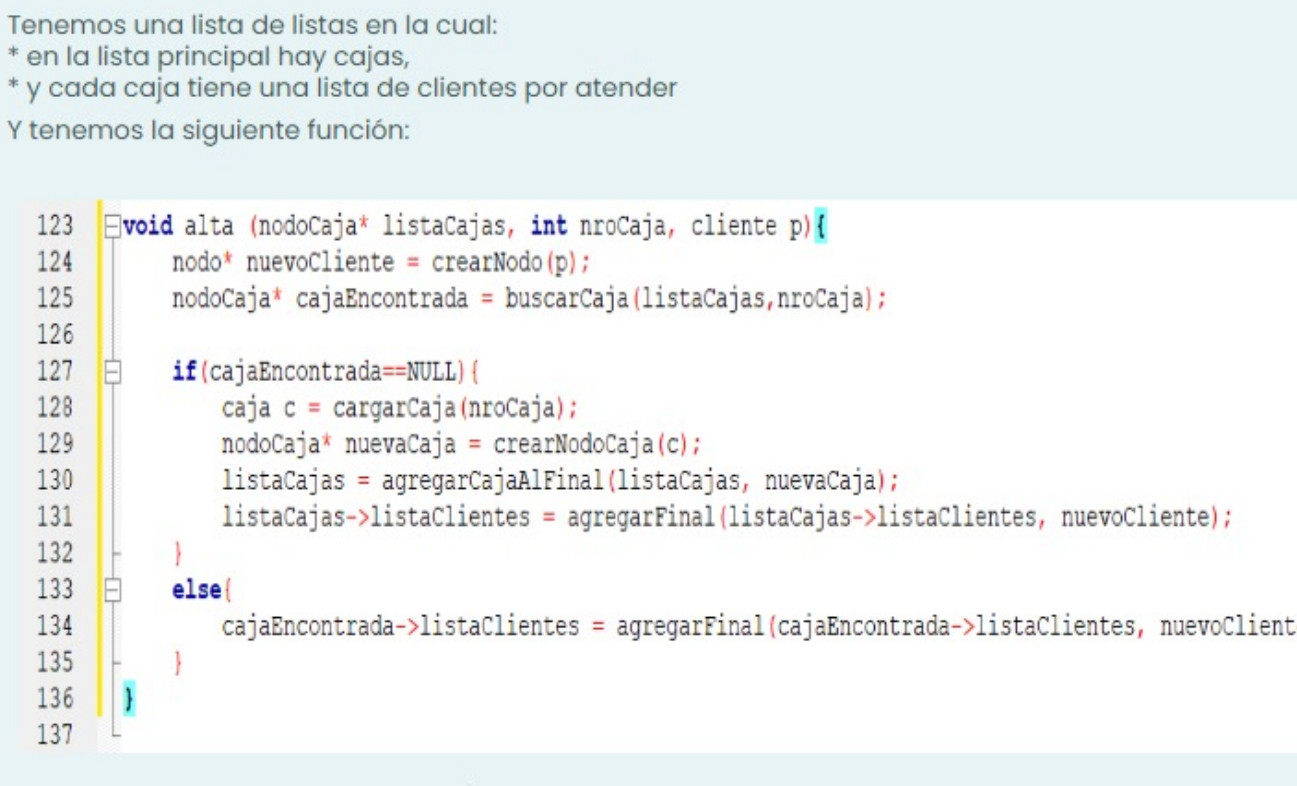
- ○ a. Suma el contenido de todo el árbol.
- ○ b. Suma el contenido de los nodos de un árbol que contengan un dato superior a 0.
- ○ c. Ninguna de las opciones anteriores.
- ○ d. Cuenta la cantidad de nodos de un árbol que contengan un dato superior a 0.
- ○ e. Cuenta la cantidad de nodos de un árbol.
- ● f. Suma el contenido de todo el árbol teniendo en cuenta el valor absoluto de cada dato. ✓

Respuesta correcta
La respuesta correcta es:
Suma el contenido de todo el árbol teniendo en cuenta el valor absoluto de cada dato.

Pregunta **5**
Correcta
Se puntúa 7,00 sobre 7,00
Marcar pregunta

Tenemos una lista de listas en la cual:
* en la lista principal hay cajas,
* y cada caja tiene una lista de clientes por atender

Y tenemos la siguiente función:



```
123  void alta (nodoCaja* listaCajas, int nroCaja, cliente p){
124      nodo* nuevoCliente = crearNodo(p);
125      nodoCaja* cajaEncontrada = buscarCaja(listaCajas,nroCaja);
126
127      if(cajaEncontrada==NULL){
128          caja c = cargarCaja(nroCaja);
129          nodoCaja* nuevaCaja = crearNodoCaja(c);
130          listaCajas = agregarCajaAlFinal(listaCajas, nuevaCaja);
131          listaCajas->listaClientes = agregarFinal(listaCajas->listaClientes, nuevoCliente);
132      }
133      else{
134          cajaEncontrada->listaClientes = agregarFinal(cajaEncontrada->listaClientes, nuevoCliente);
135      }
136  }
137
```

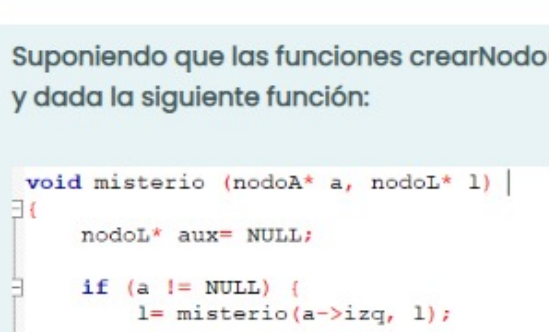
Marcar las afirmaciones correctas (<u>**Ojo! las respuestas incorrectas restan puntos**</u>**, no contestar si no está seguro**)

- ☑ a. La función no agregará correctamente el cliente cuando la caja NO existe en la lista principal. ✓
- ☑ b. La función compila pero no se ejecutará correctamente ✓
- ☐ c. En caso de tener que agregar una nueva caja a la lista porque la caja no existía aún en la lista, la nueva caja siempre debe ser agregada al final.
- ☑ d. La función agregará correctamente el cliente cuando la caja SI existe en la lista principal. ✓
- ☑ e. La función debería retornar listaCajas. ✓
- ☐ f. La función no compila.
- ☐ g. La función no tiene errores, trabaja correctamente

Respuesta correcta
Las respuestas correctas son:
La función compila pero no se ejecutará correctamente,
La función debería retornar listaCajas.,
La función agregará correctamente el cliente cuando la caja SI existe en la lista principal.,
La función no agregará correctamente el cliente cuando la caja NO existe en la lista principal.

Pregunta **6**
Parcialmente correcta
Se puntúa 2,80 sobre 6,00
Marcar pregunta

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, <u>las respuestas incorrectas restan puntos</u>)

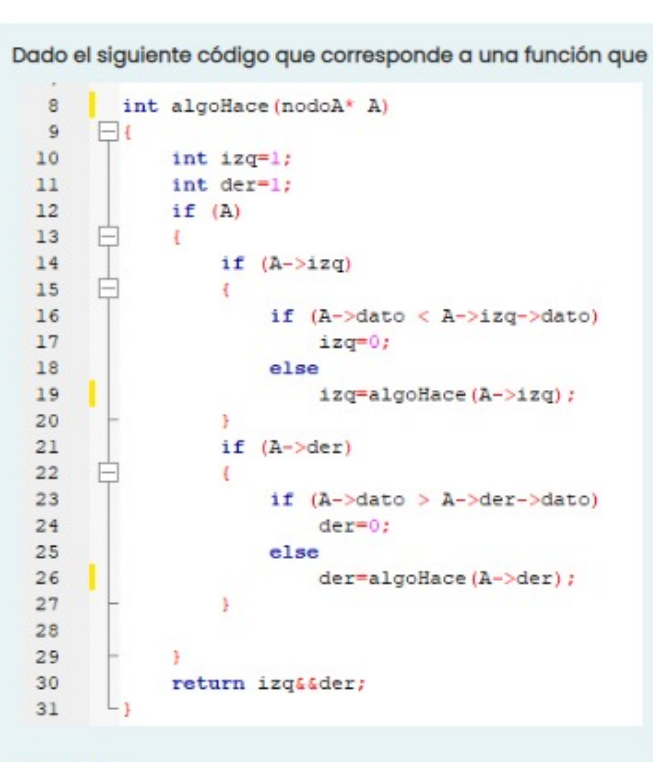
- ☐ a. La función pasa datos de una lista a un árbol.
- ☐ b. La función pasa datos de un árbol a una lista agregándolos al principio de la lista y generando así una lista ordenada en órden decreciente.
- ☑ c. La función pasa datos de un árbol a una lista agregándolos al principio de la lista y generando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

Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.
Las respuestas correctas son:
La función debería ser de tipo nodoL y retornar la lista.,
La función pasa datos de un árbol a una lista agregándolos al principio de la lista.,
La función pasa datos de un árbol a una lista agregándolos al principio de la lista y generando así una lista ordenada en órden decreciente.

Pregunta **7**
Finalizado
Se puntúa 9,00 sobre 10,00
Marcar pregunta

Dado el siguiente código que corresponde a una función que recibe por parámetro un Arbol Binario de enteros:



```
 8   int algoHace(nodoA* A)
 9   {
10       int izq=1;
11       int der=1;
12       if (A)
13       {
14           if (A->izq)
15           {
16               if (A->dato < A->izq->dato)
17                   izq=0;
18               else
19                   izq=algoHace(A->izq);
20           }
21           if (A->der)
22           {
23               if (A->dato > A->der->dato)
24                   der=0;
25               else
26                   der=algoHace(A->der);
27           }
28
29       }
30       return izq&&der;
31   }
```

Resuelva:

1) Explique brevemente el objetivo de la función.

2) Mencione cada linea donde se sitúe alguna/s regla de la recursividad

(indique el Nro de línea y si lo allí presente es: Solucion trivial, Acercamiento a condicion de corte, Llamada recursiva, o Condición de corte).

1) Lo que hace esta funcion es evaluar si el arbol esta cargado de manera ordenada segun las reglas de arboles binarios (subarbol izq de un nodo tiene que ser siempre menor que el nodo en cuestion). Y retorna 1 si esta bien cargado, y 0 si algun subarbol no cumple con esa condicion.

2) Lineas 19 y 26: Llamada a recursividad, acercamiento a la condicion de corte.
Linea 12: Condicion de corte

Lineas 16 y 23: Pueden ser tomadas como condiciones de corte si ambas corren en la misma llamada a la funcion, ya que no se va a llamar a la recursividad. Lo mismo se puede decir si no corren las lineas 21 y 14.

1) El objetivo de la funcion es comprobar si el arbol binario es ABB (Arbol Binario de Búsqueda)

2)

```
int algoHace(nodoA* A)
{
    int izq=1;
    int der=1;
    if (A) --> cond. corte
    {
        if (A->izq) --> condicion de corte
        {
            if (A->dato < A->izq->dato) --> condicion de corte

                izq=0; --> sol trivial
            else
                izq=algoHace(A->izq); --> llamada recursiva, acercamiento a cond. corte

        }
        if (A->der) -->Condicion de corte
        {
            if (A->dato > A->der->dato) -->Condicion de corte

                der=0; --> sol. trivial
            else
                der=algoHace(A->der); --> llamada recursiva, acercamiento a cond. corte
        }
    }
    return izq&&der;
}
```

Comentario:
faltan identificar las soluciones triviales

Pregunta **8**
Finalizado
Se puntúa 11,00 sobre 11,00
Marcar pregunta

A) Explique con sus propias palabras el concepto y diferencias entre Arbol, Arbol Binario y Arbol Binario de Búsqueda.-

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

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

A) Un arbol consiste de una serie de nodos entrelazados entre si secuencialmente; esto quiere decir que el nodo padre se conecta con los hijos pero los hijos, ni los hijos de estos, pueden volver a conectarse con un nodo anterior (sino seria un grafo).

La diferencia principal entre un arbol y un arbol binario es que los arboles normales pueden tener mas de 2 hijos, mientras que los arboles binarios pueden tener hasta 2.

Partiendo de esta diferencia, podremos ver que tenemos dos tipos de arboles binarios: los normales y los de busqueda. Los arboles de busqueda son ordenados con algun criterio/valor para que los nodos hijos de un nodo padre creen un subarbol ordenado. Entonces, por ejemplo, el nodo hijo izq de un nodo raiz sera un valor menor al nodo raiz, mientras que el nodo derecho comprendera un valor mayor: de ahi creando un subarbol.

B) Como explique en el punto A), se inserta segun el valor del nuevo dato a insertar. Se va recorriendo el arbol segun el valor, si el valor del dato a insertar es menor que el del nodo, sigue recorriendo hacia la izquierda. Por el contrario, si llega a ser mayor, sigue por la derecha hasta encontrar un lugar NULL donde insertarse.

C) El gran beneficio de trabajar con arboles binarios de busqueda viene principalmente cuando se tiene que trabajar con datos. Como lo dice el nombre, se pueden buscar datos de una manera muy sencilla porque los arboles estan ordenados, cosa que facilita su recorrido.

Comentario:

Pregunta **9**
Finalizado
Se puntúa 9,00 sobre 10,00
Marcar pregunta

Dado un TDA fila, donde la estructura interna de fila es implementada con una lista doblemente enlazada:

```
typedef struct nodoD {
    int dato;
    nodoD *sig;
    struct nodoD* ant;
}nodoD;

typedef struct {
    nodoD* pri;
    nodoD *ult;
}Fila;
```

Codifique la función de inserción de un elemento en la fila *agregarFila(Fila *F, int dato);*

(Cualquier otra función externa que utilice dentro de ésta, simplemente mencione su cabecera y argumentos, y explique brevemente su objetivo).

```
nodoD* CrearNodo(int dato); //Crea un nodoD (lista) a partir de un dato recibido por parametro y lo retorna.

nodoD* agregarAlFinal(nodoD* lista, nodoD* datoAdd); //Agrega al final de una lista recibida por parametro un nodo, tambien recibido por parametro, y retorna la lista modificada.

void agregarFila(Fila *F, int dato)
{
    nodoD* nuevo = CrearNodo(dato);
   if(F->pri == NULL)
   {
       F->pri = nuevo;
       F->ult = nuevo;
   }
    else
    {
        F->pri = agregarAlFinal(F->pri, nuevo);
        F->ulti = nuevo;
    }
}
```

UNA FORMA DE HACERLO, SIN USAR LIBRERIA DE LISTAS:

```
void agregarFila(Fila* C, int x){
   nodoD* nuevo = crearNodoLDoble(x);
   if (C->ult != NULL){
       C->ult->sig = nuevo;
        nuevo->ante=C->ult
   }
   else
       C->pri = nuevo;
   C->ult = nuevo;
}

nodoD* crearNodoLDoble(x) {
    nodoD* nuevo= nodoD*)malloc(sizeof(nodoC));
    nuevo->sig=NULL;
    nuevo->dato=x;

    return nuevo;
}
```

OTRA FORMA DE HACERLO, USANDO LIBRERIA DE LISTAS:

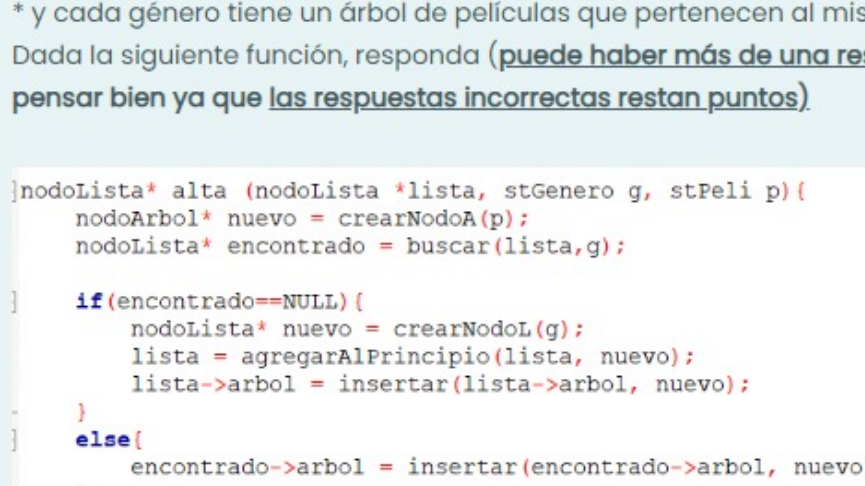
```
void agregar(Fila* fila, int dato)
{
   nodo2* nuevo = crearNodo(dato);

   if(fila->inicio == NULL){
       fila->inicio = nuevo;
   }
   else{
       fila->fin = agregarFinal(fila->fin, nuevo);
   }
   fila->fin = nuevo;
}
```

Comentario:
si ya tenes el puntero al final, es mas efectivo agregar directamente al final y no con el puntero al inicio de la fila

Pregunta **10**
Correcta
Se puntúa 6,00 sobre 6,00
Marcar pregunta

Tenemos una <u>**lista de árboles**</u> en la cual:
* en la lista hay géneros de películas
* y cada género tiene un árbol de películas que pertenecen al mismo

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



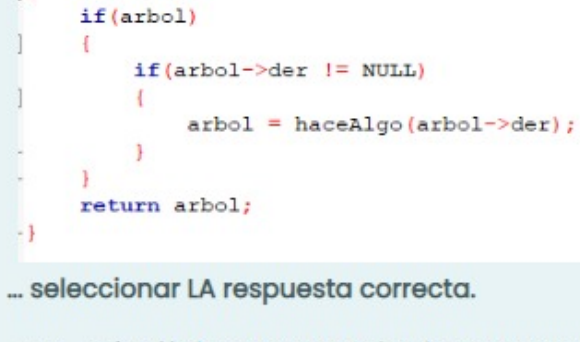
```
nodoLista* alta (nodoLista *lista, stGenero g, stPeli p){
    nodoArbol* nuevo = crearNodoA(p);
    nodoLista* encontrado = buscar(lista,g);

    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 es correcta, no tiene ningún error. ✓
- ☐ c. La función tiene errores al momento de agregar las peliculas.
- ☐ d. La función tiene errores al momento de agregar los géneros.
- ☐ e. La función no compila.

Respuesta correcta
La respuesta correcta es:
La función es correcta, no tiene ningún error.

Pregunta **11**
Correcta
Se puntúa 6,00 sobre 6,00
Marcar pregunta

Dada la siguiente función...



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

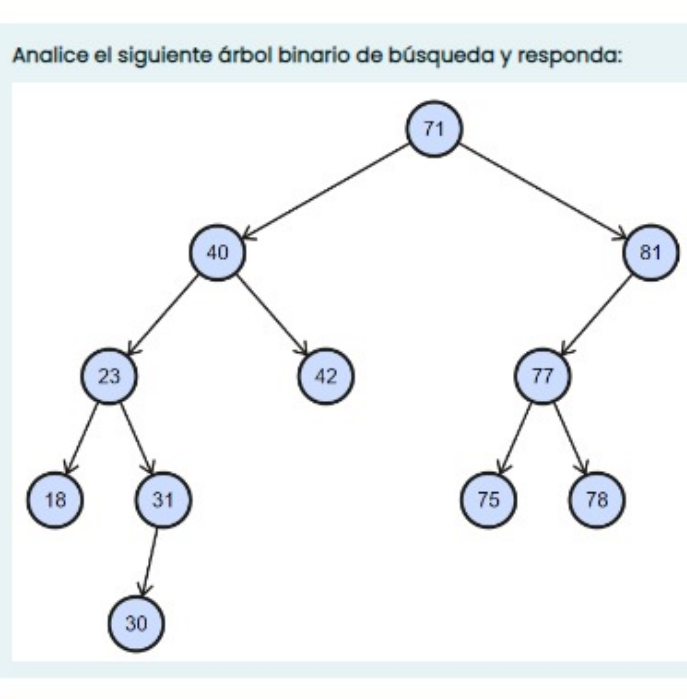
... seleccionar LA respuesta correcta.

- ○ a. La función busca y retorna el nodo que se encuentra más a la derecha del árbol.
- ○ b. La función busca y retorna el nodo con el dato mayor que se encuentra en el árbol.
- ● c. Las dos respuestas anteriores son correctas. ✓
- ○ d. Ninguna de las restantes respuestas es correcta.

Respuesta correcta
La respuesta correcta es:
Las dos respuestas anteriores son correctas.

Pregunta **12**
Parcialmente correcta
Se puntúa 5,14 sobre 6,00
Marcar pregunta

Analice el siguiente árbol binario de búsqueda y responda:



Cantidad de nodos de grado 1	2	✓
Altura del arbol	5	✓
Cantidad de niveles del árbol:	4	✗
Si se inserta el valor '27', que clave contiene el nodo que seria su padre?	30	✓
Altura del subarbol que tiene a **81** como raiz	3	✓
Cantidad de nodos de grado 0	5	✓
Nivel en el que se encuentra situado el valor **31**	3	✓

Respuesta parcialmente correcta.

Ha seleccionado correctamente 6.
La respuesta correcta es: Cantidad de nodos de grado 1 → 2, Altura del arbol → 5, Cantidad de niveles del árbol: → 5, Si se inserta el valor '27', que clave contiene el nodo que seria su padre? → 30, Altura del subarbol que tiene a **81** como raíz → 3, Cantidad de nodos de grado 0 → 5, Nivel en el que se encuentra situado el valor **31** → 3

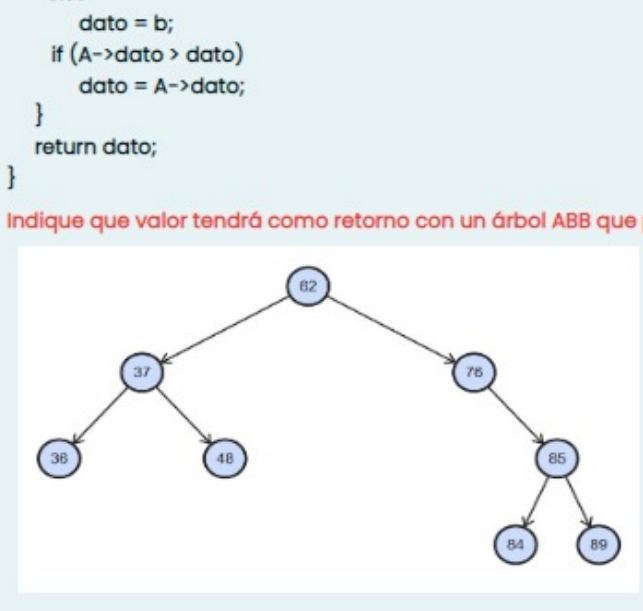
Pregunta **13**
Correcta
Se puntúa 6,00 sobre 6,00
Marcar pregunta

Una elección incorrecta implica un descuento de un porcentaje en el puntaje, si no esta seguro, no seleccione ninguna

Dada la siguiente función:

```
int haceAlgo(nodoArbol* A) {
   int dato=0;
   int a,b;
   if (A)
   {
     a = haceAlgo(A->izq);
     b= haceAlgo(A->der);
     if (a > b)
         dato = a;
     else
         dato = b;
      if (A->dato > dato)
         dato = A->dato;
   }
   return dato;
}
```

Indique que valor tendrá como retorno con un árbol ABB que posee los siguientes datos:



- ○ a. 36
- ● b. 89 ✓
- ○ c. 0
- ○ d. 62
- ○ e. 85
- ○ f. 37
- ○ g. 3
- ○ h. 5
- ○ i. 48
- ○ j. 4
- ○ k. 4
- ○ l. 2
- ○ m. 84
- ○ n. 76

Respuesta correcta
La respuesta correcta es:
89

Finalizar revisión

Actividad previa
◄ Espacio de entrega TP Arboles

Ir a...

Siguiente actividad
FORO DE CREACION DE GRUPOS Y PRESENTACION PROPUESTA DE CADA GRUPO PARA TP FINAL LAB II ►