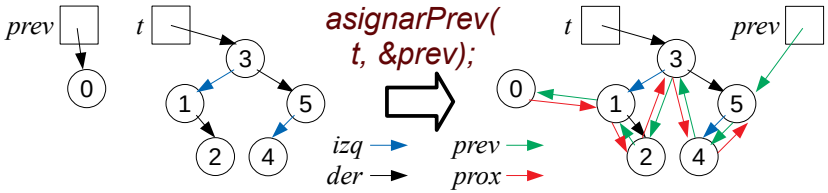
En un *recorrido en orden* de un árbol binario, se visita recursivamente primero el subárbol izquierdo, luego se visita la raíz y finalmente se visita recursivamente el subárbol derecho. Considere que se está visitando un nodo T al recorrer un árbol binario en orden. Se define como *previo* a T el nodo que se visitó anteriormente, y como *próximo* el nodo que se visitará a continuación. Estudie el lado derecho de la figura de ejemplo. Programe en el archivo *prev.c* la función *asignarPrev* que asigna los campos *prev* y *prox* agregados a la estructura de los nodos de un árbol *t*. El encabezado de la función se muestra a la derecha. El parámetro **pprev* es de entrada y salida. El nodo previo del primer nodo visitado (el nodo 1 en el ejemplo) debe ser el nodo apuntado inicialmente por **pprev* (nodo 0) y el nodo próximo del último nodo en ser visitado (nodo 5) debe ser NULL. En **pprev* debe quedar finalmente la dirección del último nodo visitado (nodo 5). En el siguiente ejemplo de uso las variables *t* y *prev* son de tipo Nodo *.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
  struct nodo *prev, *prox;
} Nodo;
void asignarPrev(Nodo t,
                 Nodo **pprev);
```



Restricción: Su solución debe tomar tiempo linealmente proporcional al número de nodos en el árbol *t*.

Ayuda: Cuando visite el nodo T, su nodo previo es **pprev*. Asigne NULL a su nodo próximo por ahora. Si el nodo previo a T no es NULL, T es el nodo próximo del nodo previo a T. Antes de continuar el recorrido, asigne T a **pprev*.

Instrucciones

Descargue *t3.zip* de U-cursos y descomprímalo. El directorio *T3* contiene los archivos (a) *test-prev.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *prev.h* que incluye los encabezados de las funcion pedidas, (d) *Makefile* que le servirá para compilar y ejecutar su tarea, y (e) *prev.cbf* para que pueda probar su tarea con *codeblocks*. **Ejecute en un terminal el comando *make*** para recibir instrucciones adicionales. Estos son los requerimientos para aprobar su tarea.

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo *heap-buffer-overflow*.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *prev.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.