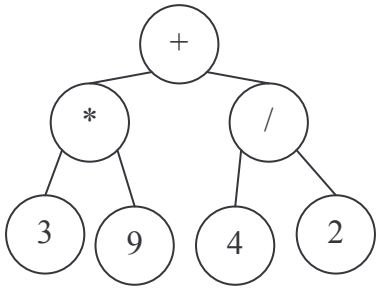
Práctica 7: Árboles Binarios en Turbo Pascal

Estructuras de Datos

Los árboles binarios se utilizan para representar expresiones en memoria; esencialmente, en compiladores de lenguajes de programación. La siguiente figura muestra un árbol para la expresión aritmética 3 * 9 + 4 / 2



El algoritmo aplicado por el compilador para pasar la expresión infija al árbol de expresión es el siguiente:

- Transformación de la expresión de infija a postfija. Dicha transformación se realiza con la ayuda de una pila.

3 9 * 4 2 / +

- Creación del árbol a partir de la expresión postfija.
  - Se necesitará una pila auxiliar de árboles
  - Cada vez que se lea un operando, se crea una hoja con él y se apila
  - Cada vez que se lea un operador, se crea un nuevo árbol, siendo los hijos los dos primeros operandos extraídos de la pila (en orden der, izq) y el operador la raíz.
  - El árbol resultante se almacena en la pila para un uso posterior. El árbol final será el último elemento que quedará en la pila auxiliar, una vez finalizada la expresión postfija.

Crea un programa en Turbo Pascal capaz de implementar en un árbol binario una expresión infija. Para ello, se facilita el código en Turbo Pascal que implementa un árbol binario.

```
UNIT ABinP;

INTERFACE
USES ELEMTAD;
TYPE
 TipoArbolBin = ^TipoNodo;

 TipoNodo = RECORD
  r: TipoElemento;
  izq, der: TipoArbolBin;
 END;

{ CONSTRUCTORAS GENERADORAS }
PROCEDURE CrearArbolVacio(VAR a:TipoArbolBin);
(* POST: CrearArbolVacio= () *)

PROCEDURE ConstruirArbolBin(VAR a: TipoArbolBin; izq: TipoArbolBin; r:TipoElemento; der:
TipoArbolBin);
(* POST: a= (izq) r (der) *)
```

```
{ OBSERVADORAS NO SELECTORAS }

FUNCTION EsArbolBinVacio(a: TipoArbolBin): Boolean;
{ POST: a = ()}

{ OBSERVADORAS SELECTORAS }

PROCEDURE Raiz(a: TipoArbolBin; VAR e: TipoElemento);
{ PRE: a = (izq) r (der)
   POST: e <- r}

PROCEDURE HijoIzq(a: TipoArbolBin; VAR hIzq: TipoArbolBin);
{ PRE: a = (izq) r (der)
   POST: hIzq <- izq }

PROCEDURE HijoDer(a: TipoArbolBin; VAR hDer: TipoArbolBin);
{ PRE: a = (izq) r (der)
   POST: e <- der}

IMPLEMENTATION

{ CONSTRUCTORAS GENERADORAS }
PROCEDURE CrearArbolVacio(VAR a: TipoArbolBin);
{ Complejidad: O(1)}
BEGIN
 a:=NIL
END;

PROCEDURE ConstruirArbolBin(VAR a: TipoArbolBin; izq: TipoArbolBin; r:TipoElemento; der:
TipoArbolBin);
{ Complejidad: O(1)}
BEGIN
 New(a);
 a^.r:=r;
 a^.izq:=izq;
 a^.der:=der
END;
{ OBSERVADORAS NO SELECTORAS }

FUNCTION EsArbolBinVacio(a:TipoArbolBin):Boolean;
{ Complejidad: O(1)}
BEGIN
 EsArbolBinVacio:= (a = NIL)
END;

{ OBSERVADORAS SELECTORAS }
PROCEDURE Raiz(a: TipoArbolBin; VAR e: TipoElemento);
{ Complejidad: O(1)}
BEGIN
 e:= a^.r
END;
PROCEDURE HijoIzq(a: TipoArbolBin; VAR hIzq: TipoArbolBin);
{ Complejidad: O(1)}
BEGIN
 hIzq := a^.izq
END;

PROCEDURE HijoDer(a: TipoArbolBin; VAR hDer: TipoArbolBin);
{ Complejidad: O(1)}
BEGIN
 hDer := a^.der
END;

END.
```

Nota: Haremos uso de las normas de estilo dictadas en clase (cabecera del fichero, interfaz de la unidad con precondiciones, postcondiciones, excepciones, implementaciones con el análisis de complejidad de cada operación, nombres coherentes de variables y operaciones,...)

Plantilla de cabecera del fichero:

```
{*****************************************************************************
*                                                                            *
*      Módulo:                                                               *
*      Tipo:  Programa()     Interfaz-Implementación TAD ()       Otros()     *
*      Autor/es:                                                             *
*      Fecha de actualización:                                               *
*      Descripción:                                                          *
*                                                                            *
*****************************************************************************}
```