```
34  }
```

□

Ejemplo 6.3.4

Usando el iterador preorden_iterator y postorden_iterator implementar una función para deducir si dos árboles binarios son uno el reflejado del otro

Para implementar esta función usaremos el operador ++ de iterador postorden_iterator y el operador – de un iterador preorden_iterator.

```
1     typename <class T>
2     bool Reflejados (ArbolBinario<T> &a1, ArbolBinario<T> &a2){
3         typename ArbolBinario<T>::iterator_preorden itpre=a2.end();
4         //retrocedemos al ultimo
5         --itpre;
6         typename ArbolBinario<T>::iterator_preorden itpost=a1.begin();
7
8        while (itpre!=a2.end() && itpost!=a1.end() && *itpre==*itpost){
9         --itpre; ++itpost
10       }
11       //si los dos no han llegado al final
12       if (itpre!=a2.end() && itpost!=a1.end()) return false;
13       else
14       return true;
15    }
```

□

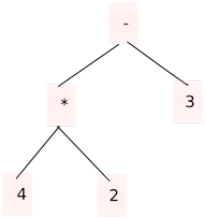
Ejercicio 6.3

Sobrecargar los operadores – para la clase postorden_iterator e inorden_iterator.

□

6.3.3 Expresiones Algebraicas

*Ya vimos en el capítulo de estructuras lineales las expresiones algebráicas como una aplicación de uso de la pila, para pasar una expresión algebraica a notación Polaca o notación inorden. En esta sección veremos que se puede usar un árbol binario para almacenar una expresión algebraica (compuesta de operadores: +, -, /, *) en notación inorden (con la que estamos acostumbrados a trabajar p.e 4*2-3) o a notación prefijo o polaca (p.e - * 4 2 3), o notación postfijo (p.e 4 2 * 3 -).*



Para ello vamos a implementar el TDA Expresión que contiene una expresión en notación inorden, usando para su representación un ArbolBinario. A este TDA le añadiremos métodos para poder evaluar la expresión dando un resultado escalar, obtener la expresión Polaca o prefijo equivalente y obtener la expresión postfijo. De esta forma el TDA Expresion sería el siguiente:

```
class Expresion{
private:
     ArbolBinario<string> datos;

public:
  Expresion(){}
  /**
   * @brief Inicia una expresion con la cadena de entrada
   * @note si la cadena no tiene valores correctos la expresion se inicia a vacio
   */
  Expresion(const string &e);

  /**
   * * @brief Evalua la expresion deovolviendo el resultado
   */
  float Evalua()const;

  /**
   * * @brief Obtiene la notacion en prefijo
   *
   */
  string Expresion_Prefijo()const;


  /**
   * * @brief Obtiene la notacion en postfijo
   *
   */
  string Expresion_Postfijo()const;

};
```

Antes de ver la implementación de estas funciones nos hará falta algunas funciones que nos diga si dada una expresión lo que viene a continuación es un operador o un operando y obtenerlo en caso afirmativo. Además tendremos una función QuitarBlancos que nos elimina todos los espacios que haya al principio de una expresión.

```
#include "expresion.h"
#include <sstream>
```

```
3
4  /*************************************************************/
5  void QuitarBlancos(string &expresion){
6    while (expresion.size()>0 && expresion[0]==' '){
7       expresion= expresion.substr(1,string::npos);
8    }
9  }
10 /*************************************************************/
11  bool Operador(string &expresion, char &operador){
12
13       QuitarBlancos(expresion);
14       if (expresion.size()>0){
15         if (expresion[0]=='+' || expresion[0]=='-' ||
16           expresion[0]=='*' || expresion[0]=='/'){
17             operador = expresion[0];
18             expresion= expresion.substr(1,string::npos);
19             return true;
20         }
21       }
22       return false;
23  }
24 /*************************************************************/
25 template <class T>
26 void GetOperando(string & expresion,T &operando){
27       QuitarBlancos(expresion);
28       if (expresion.size()>0){
29           stringstream ss;
30           string aux;
31           ss.str(expresion);
32           ss>>aux; //hasta el primer separador
33           //le quitamos a expresion lo leido en aux
34           expresion=expresion.substr(aux.size(),string::npos);
35           //convertimos de string a int
36
37           ss.clear();
38           ss.str(aux);
39           ss>>operando;
40       }
41 }
42 /*************************************************************/
43 bool isOperator(string expresion){
44           if (expresion[0]=='+' || expresion[0]=='-' ||
45           expresion[0]=='*' || expresion[0]=='/'){
```

```
46            return true;
47          }
48          else return false;
49  }
```

Ahora si veamosla implementación de los métodos:

```
1   Expresion::Expresion(const string &e){
2     string expresion = e;
3
4     QuitarBlancos(expresion);
5     //inicializamos el arbol con los tres primeros elementos
6     string op1;
7     GetOperando(expresion, op1);//el operando izquierdo
8     QuitarBlancos(expresion);
9
10    char operacion;
11    if (Operador(expresion,operacion)){//la operacion
12      string op2;
13
14      QuitarBlancos(expresion);
15      GetOperando(expresion, op2);//el operando derecho
16      //inicializamos el arbol
17      string oper; oper.push_back(operacion);
18      datos=ArbolBinario<string>(oper);
19      datos.Insertar_Hi(datos.getRaiz(),op1);
20      datos.Insertar_Hd(datos.getRaiz(),op2);
21
22      //ahora vamos leyendo de dos en dos:operador operando derecho
23      while (expresion.size()>0){
24        QuitarBlancos(expresion);
25        string op;
26        if (Operador(expresion,operacion)){
27          QuitarBlancos(expresion);
28          GetOperando(expresion, op2);
29          string oper; oper.push_back(operacion);
30          ArbolBinario<string> aux(oper);
31          aux.Insertar_Hd(aux.getRaiz(),op2);
32          aux.Insertar_Hi(aux.getRaiz(),datos);
33          datos=aux;
34
35        }
36        else {
37          datos=ArbolBinario<string>();
38          return ;
```

```
39          }
40
41        }
42
43     }
44     else return;
45   }
46   /*************************************************************/
47   float Expresion::Evalua()const{
48     float res=0.0;
49     ArbolBinario<string>::inorden_iterador in=datos.begininorden();
50     float left_op,right_op;
51     string op;
52     while (in!=datos.endinorden()){
53      if (isOperator(*in)){
54        //leemos el siguiente en inorden
55        op=*in;
56        ++in;
57        string aux =*in;
58        GetOperando(aux,right_op);
59        switch (op[0]){
60          case '+':
61              res = left_op+right_op;
62              break;
63          case '-':
64              res = left_op-right_op;
65              break;
66          case '*':
67              res = left_op*right_op;
68              break;
69          case '/':
70              res = left_op/right_op;
71              break;
72
73        }
74        left_op=res;
75
76        ++in;
77      }
78
79      else{
80         string aux =*in;
81         GetOperando(aux,left_op);
```

```
82          ++in;
83       }
84
85     }
86     return res;
87   }
88   /*************************************************************/
89
90   string Expresion::Expresion_Prefijo()const{
91     ArbolBinario<string>::preorden_iterador pre=datos.beginpreorden();
92     string salida="";
93     for(;pre!=datos.endpreorden();++pre){
94        salida=salida+*pre+ " ";
95     }
96     return salida;
97
98   }
99   /*************************************************************/
100  string Expresion::Expresion_Postfijo()const{
101    ArbolBinario<string>::postorden_iterador post=datos.beginpostorden();
102    string salida="";
103    for(;post!=datos.endpostorden();++post){
104       salida=salida +*post+ " ";
105    }
106    return salida;
107
108  }
```

Con respecto a las funciones Expresion_Prefijo y Expresion_Posfijo, simplemente hace falta usar un iterador y recorrer el árbol en el sentido de dicho iterador.

6.4 Arboles generales

Para representar un árbol general, cada nodo contendrá su etiqueta y punteros al padre, al hijo a la izquierda y al hermano a la derecha:

```
1  template <class T>
2  struct info_nodo {
3        T et;
4        info_nodo<T> * padre, * hijoizq, * hermanodcha;
5        info_nodo() {
6              padre = hijoizq = hermanodcha = 0;
7        }
8        infonodo(const T & e) {
9              et = e;
```