Reto 5: Aplicacion de bintrees

Generated by Doxygen 1.9.2

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

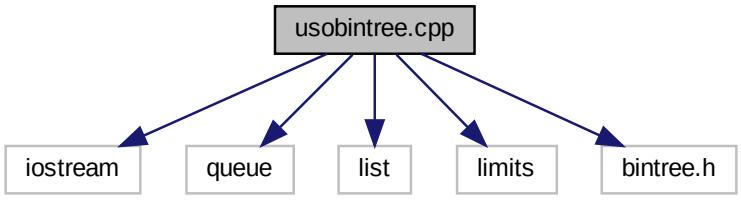
File Documentation

2.1 usobintree.cpp File Reference

archivo de pruebas de funciones de bintree

```
#include <iostream>
#include <queue>
#include <list>
#include <limits>
#include "bintree.h"
```

Include dependency graph for usobintree.cpp:



Functions

- template<class T >
  bool esHoja (const bintree< T > &A, const typename bintree< T >::node &v)
- template<class T >
  bool esInterno (const bintree< T > &A, const typename bintree< T >::node &v)
- template<class T >
  void PreordenBinario (const bintree< T > &A, typename bintree< T >::node v)
- template<class T >
  void InordenBinario (const bintree< T > &A, typename bintree< T >::node v)
- template<class T >
  void PostordenBinario (const bintree< T > &A, typename bintree< T >::node v)

- template<class T >
  void ListarPostNiveles (const bintree< T > &A, typename bintree< T >::node n)
- int numerocaminos (bintree< int > &ab, int k, bintree< int >::node n)
- pair< int, list< int > > caminomen_nodo (bintree< int >::node n)
- list< int > caminodemenores (bintree< int > &ab)
- bool pesointerior_nodo (bintree< int >::node n)

  funcion auxiliar a pesointerior para trabajar con nodos de forma recursiva
- bool pesointerior (bintree< int > a)

  comprueba si un arbol es pesointerior, es decir, la suma de los hijos de cada nodo es el padre Usa la funcion auxiliar pesointerior_nodo para trabajar con nodos de forma recursiva
- int sumaparantec (bintree< int > &a, bintree< int >::node n)

  devuelve la suma de los caminos cuyos nodos tienen valor par
- bool camino_ord (bintree< int > A, bintree< int >::node n)

  comprueba si un arbol dado contiene caminos ordenador de menor a mayor
- template<class T >
  ostream & operator<< (ostream &os, bintree< T > &arb)
- void separador ()
- int main ()

2.1.1 Detailed Description

archivo de pruebas de funciones de bintree

Author

Yeray Lopez Ramirez

Date

diciembre de 2021

Definition in file usobintree.cpp.

2.1.2 Function Documentation

2.1.2.1 camino_ord()

```
bool camino_ord (
            bintree< int > A,
            bintree< int >::node n )
```

comprueba si un arbol dado contiene caminos ordenador de menor a mayor

Parameters

a	arbol a comprobar
n	node raiz del arbol

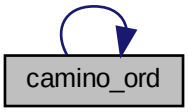
Returns

true si existe el camino, false en otro caso

Definition at line 184 of file usobintree.cpp.

```
00184                                                          {
00185    if(n.left().null() && n.right().null())
00186       if(*n > *n.parent())
00187          return true;
00188       else
00189          return false;
00190    else{
00191       if(n.parent().null()){
00192          if(camino_ord(A, n.left()))
00193             return true;
00194          if(camino_ord(A, n.right()))
00195             return true;
00196
00197          return false;
00198       }
00199       else if( *n > *n.parent()){
00200          return camino_ord(A, n.left());
00201          return camino_ord(A, n.right());
00202       }
00203    }
00204
00205    return false; //false en otro caso;
00206 }
```

Here is the call graph for this function:



2.1.2.2 caminodemenores()

```
list< int > caminodemenores (
              bintree< int > & ab )
```

Definition at line 111 of file usobintree.cpp.

```
00111                                                  {
00112    return caminomen_nodo(ab.root()).second;
00113 }
```

2.1.2.3 caminomen_nodo()

```
pair< int, list< int > > caminomen_nodo (
            bintree< int >::node n )
```

Definition at line 89 of file usobintree.cpp.

```
00089                                                       {
00090    if(n.null())
00091       return pair<int,list<int»(numeric_limits<int>::max(), list<int>());
00092    else
00093       if(n.left().null() && n.right().null()) //Paramos en la hoja
00094          return pair<int,list<int»(*n,list<int>(1,*n));
00095       else{
00096          pair<int,list<int» li=caminomen_nodo(n.left());
00097          pair<int,list<int» ld=caminomen_nodo(n.right());
00098          if(li.first<ld.first){
00099             li.second.push_front(*n);
00100             li.first += *n;
00101             return li;
00102          }
00103          else{
00104             ld.second.push_front(*n);
00105             ld.first += *n;
00106             return ld;
00107          }
00108       }
00109 }
```

2.1.2.4 esHoja()

```
template<class T >
bool esHoja (
            const bintree< T > & A,
            const typename bintree< T >::node & v )
```

Definition at line 17 of file usobintree.cpp.

```
00018 {
00019   return ( v.left().null()  && v.right().null() );
00020 }
```

2.1.2.5 esInterno()

```
template<class T >
bool esInterno (
            const bintree< T > & A,
            const typename bintree< T >::node & v )
```

Definition at line 23 of file usobintree.cpp.

```
00024 {
00025   return ( !v.left().null() || !v.right().null() );
00026 }
```

2.1.2.6 InordenBinario()

```
template<class T >
void InordenBinario (
             const bintree< T > & A,
             typename bintree< T >::node v )
```

Definition at line 39 of file usobintree.cpp.

```
00041 {
00042   if (!v.null()) {
00043     InordenBinario(A, v.left());
00044     cout « *v; //acción sobre el nodo v.
00045     InordenBinario(A, v.right());
00046   }
00047 }
```

2.1.2.7 ListarPostNiveles()

```
template<class T >
void ListarPostNiveles (
             const bintree< T > & A,
             typename bintree< T >::node n )
```

Definition at line 61 of file usobintree.cpp.

```
00061                                                                    {
00062   queue<typename bintree<T>::node> nodos;
00063   if (!n.null()) {
00064     nodos.push(n);
00065     while (!nodos.empty()) {
00066       n = nodos.front(); nodos.pop();
00067       cout « *n;
00068       if (!n.left().null()) nodos.push(n.left());
00069       if (!n.right().null())
00070         nodos.push(n.right());
00071     }
00072   }
00073 }
```

2.1.2.8 main()

```
int main ( )
```

Definition at line 227 of file usobintree.cpp.

```
00228 {
00229   /*list<int> camino = caminodemenores(Arb);
00230   cout « "El camino menores es: ";
00231   for (auto const &i: camino) {
00232         cout « i « " ";
00233   }
00234   cout « endl;*/
00235
00237
00238   separador();
00239   cout « "Funcion 21: pesointerior:" « endl;
00240   //       30
00241   //     /    \
00242   //    21     9
00243   //  /  \    /
00244   // 7   14  9
00245   //        / \
00246   //       1  13
00247   bintree<int> pesoArb(30);
00248   pesoArb.insert_left(pesoArb.root(), 21);
00249   pesoArb.insert_right(pesoArb.root(), 9);
```

```
00250
00251   pesoArb.insert_left(pesoArb.root().left(), 7);
00252   pesoArb.insert_right(pesoArb.root().left(), 14);
00253
00254   pesoArb.insert_left(pesoArb.root().right(), 9);
00255
00256   pesoArb.insert_left(pesoArb.root().right().left(), 1);
00257   pesoArb.insert_right(pesoArb.root().right().left(), 8);
00258
00259   cout « "Preorden:";
00260
00261   for (bintree<int>::preorder_iterator i = pesoArb.begin_preorder(); i!=pesoArb.end_preorder(); ++i)
00262     cout « *i « " ";
00263
00264   cout « endl;
00265
00266
00267   cout « "Inorden:";
00268
00269   for (bintree<int>::inorder_iterator i = pesoArb.begin_inorder(); i!=pesoArb.end_inorder(); ++i)
00270     cout « *i « " ";
00271
00272   cout « endl;
00273
00274
00275   cout « "Postorden:";
00276
00277   for (bintree<int>::postorder_iterator i = pesoArb.begin_postorder(); i!=pesoArb.end_postorder();
      ++i)
00278     cout « *i « " ";
00279
00280   cout « endl;
00281
00282   cout « "Por Niveles:";
00283
00284   for (bintree<int>::level_iterator i = pesoArb.begin_level(); i!=pesoArb.end_level(); ++i)
00285     cout « *i « " ";
00286   cout « endl;
00287   if(pesointerior(pesoArb))
00288      cout « "Es pesointerior" « endl;
00289   else
00290      cout « "No es pesointerior" « endl;
00291
00292   separador();
00293
00295
00296   cout « "Funcion 24: sumaparantec:" « endl;
00297   //       10
00298   //      /  \
00299   //     21   6
00300   //         / \
00301   //        8  11
00302   //          / \
00303   //         12 6
00304   bintree<int> sumaArb(10);
00305   sumaArb.insert_left(sumaArb.root(), 21);
00306   sumaArb.insert_right(sumaArb.root(), 6);
00307
00308   sumaArb.insert_left(sumaArb.root().right(), 8);
00309   sumaArb.insert_right(sumaArb.root().right(), 11);
00310
00311   sumaArb.insert_left(sumaArb.root().right().right(), 12);
00312   sumaArb.insert_right(sumaArb.root().right().right(),6);
00313
00314   cout « "Preorden:";
00315
00316   for (bintree<int>::preorder_iterator i = sumaArb.begin_preorder(); i!=sumaArb.end_preorder(); ++i)
00317     cout « *i « " ";
00318
00319   cout « endl;
00320
00321
00322   cout « "Inorden:";
00323
00324   for (bintree<int>::inorder_iterator i = sumaArb.begin_inorder(); i!=sumaArb.end_inorder(); ++i)
00325     cout « *i « " ";
00326
00327   cout « endl;
00328
00329
00330   cout « "Postorden:";
00331
00332   for (bintree<int>::postorder_iterator i = sumaArb.begin_postorder(); i!=sumaArb.end_postorder();
      ++i)
00333     cout « *i « " ";
00334
00335   cout « endl;
```

```
00336
00337   cout « "Por Niveles:";
00338
00339   for (bintree<int>::level_iterator i = sumaArb.begin_level(); i!=sumaArb.end_level(); ++i)
00340     cout « *i « " ";
00341   cout « endl;
00342   bintree<int>::node n = sumaArb.root();
00343   cout « "La suma de parantec es " « sumaparantec(sumaArb,n) « endl;
00344
00345   separador();
00346
00348
00349   cout « "Funcion 26: camino_ord:" « endl;
00350   //        3
00351   //       /   \
00352   //      6     7
00353   //     / \   / \
00354   //    1   3 9   5
00355   bintree<int> caminoOrdArb(3);
00356   caminoOrdArb.insert_left(caminoOrdArb.root(), 6);
00357   caminoOrdArb.insert_right(caminoOrdArb.root(), 7);
00358
00359   caminoOrdArb.insert_left(caminoOrdArb.root().left(), 1);
00360   caminoOrdArb.insert_right(caminoOrdArb.root().left(), 3);
00361
00362   caminoOrdArb.insert_left(caminoOrdArb.root().right(), 9);
00363   caminoOrdArb.insert_right(caminoOrdArb.root().right(), 5);
00364
00365   cout « "Preorden:";
00366
00367   for (bintree<int>::preorder_iterator i = caminoOrdArb.begin_preorder();
      i!=caminoOrdArb.end_preorder(); ++i)
00368     cout « *i « " ";
00369
00370   cout « endl;
00371
00372
00373   cout « "Inorden:";
00374
00375   for (bintree<int>::inorder_iterator i = caminoOrdArb.begin_inorder(); i!=caminoOrdArb.end_inorder();
      ++i)
00376     cout « *i « " ";
00377
00378   cout « endl;
00379
00380
00381   cout « "Postorden:";
00382
00383   for (bintree<int>::postorder_iterator i = caminoOrdArb.begin_postorder(); i !=
      caminoOrdArb.end_postorder(); ++i)
00384     cout « *i « " ";
00385
00386   cout « endl;
00387
00388   cout « "Por Niveles:";
00389
00390   for (bintree<int>::level_iterator i = caminoOrdArb.begin_level(); i!=caminoOrdArb.end_level(); ++i)
00391     cout « *i « " ";
00392   cout « endl;
00393   bintree<int>::node nOrdAbr = caminoOrdArb.root();
00394   if(camino_ord(caminoOrdArb, nOrdAbr))
00395      cout « "Existe un camino ordenado" « endl;
00396   else
00397      cout « "No existe camino ordenado" « endl;
00398
00399   separador();
00400   return 0;
00401 }
```

2.1.2.9 numerocaminos()

```
int numerocaminos (
            bintree< int > & ab,
            int k,
            bintree< int >::node n )
```

Definition at line 75 of file usobintree.cpp.

```
00075                                                                    {
00076    if(n.left().null() && n.right().null()) //es una hoja?
00077       if(*n==k) return 1;
00078       else return 0;
00079    else{
00080       int contador = 0;
00081       if(!n.left().null())
00082          contador+=numerocaminos(ab, k - *n, n.left());
00083       if(!n.right().null())
00084          contador+=numerocaminos(ab, k - *n, n.right());
00085       return contador;
00086    }
00087 }
```

2.1.2.10 operator<<()

```
template<class T >
ostream & operator<< (
             ostream & os,
             bintree< T > & arb )
```

Definition at line 209 of file usobintree.cpp.

```
00210 {
00211
00212   cout « "Preorden:";
00213
00214   for (typename bintree<T>::preorder_iterator i = arb.begin_preorder(); i!=arb.end_preorder(); ++i)
00215     cout « *i « " ";
00216
00217   cout « endl;
00218   return os;
00219 }
```

2.1.2.11 pesointerior()

```
bool pesointerior (
             bintree< int > a )
```

comprueba si un arbol es pesointerior, es decir, la suma de los hijos de cada nodo es el padre Usa la funcion auxiliar pesointerior_nodo para trabajar con nodos de forma recursiva

Parameters

a	arbol a comprobar

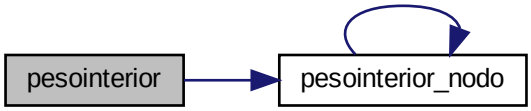
Returns

true si la suma de los hijos de los nodos es igual al padre, false en otro caso

Definition at line 153 of file usobintree.cpp.

```
00153                                    {
00154    return pesointerior_nodo(a.root());
00155 }
```

Here is the call graph for this function:



2.1.2.12 pesointerior_nodo()

```
bool pesointerior_nodo (
            bintree< int >::node n )
```

funcion auxiliar a pesointerior para trabajar con nodos de forma recursiva

Parameters

n	node a iterar del arbol

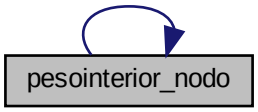
Returns

true si la suma de los hijos es igual al padre, false en otro caso

Definition at line 120 of file usobintree.cpp.

```
00120                                           {
00121    if(n.left().null() && n.right().null())
00122       return true; //Si llega aqui, ha pasado todas las comprobaciones anteriores, por tanto, devuelve
       true
00123    else
00124       if(n.parent().null()){ //si es el nodo raiz, controla el resultado final
00125          if(*n == *n.left() + *n.right()){
00126             if(!pesointerior_nodo(n.left()))
00127                return false;
00128             if(!pesointerior_nodo(n.right()))
00129                return false;
00130          return true; //si todos los nodos cumplen la condicion, entonces devuelve true
00131          }
00132       }
00133       else //si es otro nodo
00134          if(n.left().null() && *n == *n.right())
00135             return pesointerior_nodo(n.right());
00136          else if(n.right().null() && *n == *n.left())
00137             return pesointerior_nodo(n.left());
00138          else
00139             if(*n == *n.left() + *n.right()){
00140                return pesointerior_nodo(n.left());
00141                return pesointerior_nodo(n.right());
00142
00143             }
00144    return false; //false en otro caso
00145 }
```

Here is the call graph for this function:



2.1.2.13 PostordenBinario()

```
template<class T >
void PostordenBinario (
             const bintree< T > & A,
             typename bintree< T >::node v )
```

Definition at line 50 of file usobintree.cpp.

```
00052 {
00053   if (!v.null()) {
00054     PostordenBinario(A, v.left());
00055     PostordenBinario(A, v.right());
00056     cout « *v; // acción sobre el nodo v.
00057   }
00058 }
```

2.1.2.14 PreordenBinario()

```
template<class T >
void PreordenBinario (
             const bintree< T > & A,
             typename bintree< T >::node v )
```

Definition at line 29 of file usobintree.cpp.

```
00030                                                {
00031   if (!v.null()) {
00032     cout « *v; // acción sobre el nodo v.
00033     PreordenBinario(A, v.left());
00034     PreordenBinario(A, v.right());
00035   }
00036 }
```

2.1.2.15 separador()

```
void separador ( )
```

Definition at line 221 of file usobintree.cpp.

```
00221                {
00222    for(int i=0; i < 100; i++)
00223       cout « "=";
00224    cout « endl;
00225 }
```

2.1.2.16 sumaparantec()

```
int sumaparantec (
            bintree< int > & a,
            bintree< int >::node n )
```

devuelve la suma de los caminos cuyos nodos tienen valor par

Parameters

a	arbol a comprobar
n	node raiz del arbol

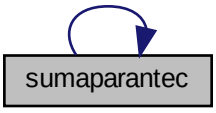
Returns

contador total de los valores de los nodos con valor par(siguiendo un camino)

Definition at line 163 of file usobintree.cpp.

```
00163                                                          {
00164    if(n.left().null() && n.right().null()) //es una hoja?
00165       if(*n%2 == 0){return *n;}
00166       else return 0;
00167    else{
00168       int contador = 0;
00169       if(*n%2 == 0){
00170          contador+=sumaparantec(a, n.left());
00171          contador+=sumaparantec(a, n.right());
00172          contador+=*n;
00173       }
00174          return contador;
00175    }
00176 }
```

Here is the call graph for this function:



2.2 usobintree.cpp

Go to the documentation of this file.

```
00001
00008 #include <iostream>
00009 #include<queue>
00010 #include <list>
00011 #include <limits>
00012 #include "bintree.h"
00013
00014 using namespace std;
00015
00016 template <class T>
00017 bool esHoja(const bintree<T> & A, const typename bintree<T>::node &v)
00018 {
```

```
00019   return ( v.left().null()  && v.right().null() );
00020 }
00021
00022 template <class T>
00023 bool esInterno(const bintree<T> & A, const typename bintree<T>::node &v)
00024 {
00025   return ( !v.left().null() || !v.right().null() );
00026 }
00027
00028 template <class T>
00029 void PreordenBinario(const bintree<T> & A,
00030                      typename bintree<T>::node v) {
00031   if (!v.null()) {
00032     cout « *v; // acción sobre el nodo v.
00033     PreordenBinario(A, v.left());
00034     PreordenBinario(A, v.right());
00035   }
00036 }
00037
00038 template <class T>
00039 void InordenBinario(const bintree<T> & A,
00040                     typename bintree<T>::node v)
00041 {
00042   if (!v.null()) {
00043     InordenBinario(A, v.left());
00044     cout « *v; //acción sobre el nodo v.
00045     InordenBinario(A, v.right());
00046   }
00047 }
00048
00049 template <class T>
00050 void PostordenBinario(const bintree<T> & A,
00051                       typename bintree<T>::node v)
00052 {
00053   if (!v.null()) {
00054     PostordenBinario(A, v.left());
00055     PostordenBinario(A, v.right());
00056     cout « *v; // acción sobre el nodo v.
00057   }
00058 }
00059
00060 template <class T>
00061 void ListarPostNiveles(const bintree<T> &A, typename bintree<T>::node n) {
00062   queue<typename bintree<T>::node> nodos;
00063   if (!n.null()) {
00064     nodos.push(n);
00065     while (!nodos.empty()) {
00066       n = nodos.front(); nodos.pop();
00067       cout « *n;
00068       if (!n.left().null()) nodos.push(n.left());
00069       if (!n.right().null())
00070         nodos.push(n.right());
00071     }
00072   }
00073 }
00074
00075 int numerocaminos(bintree<int> &ab,int k, bintree<int>::node n){
00076    if(n.left().null() && n.right().null()) //es una hoja?
00077       if(*n==k) return 1;
00078       else return 0;
00079    else{
00080       int contador = 0;
00081       if(!n.left().null())
00082          contador+=numerocaminos(ab, k - *n, n.left());
00083       if(!n.right().null())
00084          contador+=numerocaminos(ab, k - *n, n.right());
00085       return contador;
00086    }
00087 }
00088
00089 pair<int,list<int» caminomen_nodo(bintree<int>::node n){
00090    if(n.null())
00091       return pair<int,list<int»(numeric_limits<int>::max(), list<int>());
00092    else
00093       if(n.left().null() && n.right().null()) //Paramos en la hoja
00094          return pair<int,list<int»(*n,list<int>(1,*n));
00095       else{
00096          pair<int,list<int» li=caminomen_nodo(n.left());
00097          pair<int,list<int» ld=caminomen_nodo(n.right());
00098          if(li.first<ld.first){
00099             li.second.push_front(*n);
00100             li.first += *n;
00101             return li;
00102          }
00103          else{
00104             ld.second.push_front(*n);
00105             ld.first += *n;
```

```
00106             return ld;
00107          }
00108       }
00109 }
00110
00111 list<int> caminodemenores(bintree<int> &ab){
00112    return caminomen_nodo(ab.root()).second;
00113 }
00114
00120 bool pesointerior_nodo(bintree<int>::node n){
00121    if(n.left().null() && n.right().null())
00122       return true; //Si llega aqui, ha pasado todas las comprobaciones anteriores, por tanto, devuelve
       true
00123    else
00124       if(n.parent().null()){ //si es el nodo raiz, controla el resultado final
00125          if(*n == *n.left() + *n.right()){
00126             if(!pesointerior_nodo(n.left()))
00127                return false;
00128             if(!pesointerior_nodo(n.right()))
00129                return false;
00130          return true; //si todos los nodos cumplen la condicion, entonces devuelve true
00131          }
00132       }
00133       else //si es otro nodo
00134          if(n.left().null() && *n == *n.right())
00135             return pesointerior_nodo(n.right());
00136          else if(n.right().null() && *n == *n.left())
00137             return pesointerior_nodo(n.left());
00138          else
00139             if(*n == *n.left() + *n.right()){
00140                return pesointerior_nodo(n.left());
00141                return pesointerior_nodo(n.right());
00142
00143             }
00144    return false; //false en otro caso
00145 }
00146
00153 bool pesointerior(bintree<int> a){
00154    return pesointerior_nodo(a.root());
00155 }
00156
00163 int sumaparantec(bintree<int> & a, bintree<int>::node n){
00164    if(n.left().null() && n.right().null()) //es una hoja?
00165       if(*n%2 == 0){return *n;}
00166       else return 0;
00167    else{
00168       int contador = 0;
00169       if(*n%2 == 0){
00170          contador+=sumaparantec(a, n.left());
00171          contador+=sumaparantec(a, n.right());
00172          contador+=*n;
00173       }
00174          return contador;
00175    }
00176 }
00177
00184 bool camino_ord(bintree<int> A, bintree<int>::node n){
00185    if(n.left().null() && n.right().null())
00186       if(*n > *n.parent())
00187          return true;
00188       else
00189          return false;
00190    else{
00191       if(n.parent().null()){
00192          if(camino_ord(A, n.left()))
00193             return true;
00194          if(camino_ord(A, n.right()))
00195             return true;
00196
00197          return false;
00198       }
00199       else if( *n > *n.parent()){
00200          return camino_ord(A, n.left());
00201          return camino_ord(A, n.right());
00202       }
00203    }
00204
00205    return false; //false en otro caso;
00206 }
00207
00208 template <class T>
00209 ostream & operator « (ostream & os, bintree<T> &arb)
00210 {
00211
00212   cout « "Preorden:";
00213
00214   for (typename bintree<T>::preorder_iterator i = arb.begin_preorder(); i!=arb.end_preorder(); ++i)
```

```
00215     cout « *i « " ";
00216
00217   cout « endl;
00218   return os;
00219 }
00220
00221 void separador(){
00222    for(int i=0; i < 100; i++)
00223       cout « "=";
00224    cout « endl;
00225 }
00226
00227 int main()
00228 {
00229   /*list<int> camino = caminodemenores(Arb);
00230   cout « "El camino menores es: ";
00231   for (auto const &i: camino) {
00232         cout « i « " ";
00233   }
00234   cout « endl;*/
00235
00237
00238   separador();
00239   cout « "Funcion 21: pesointerior:" « endl;
00240   //       30
00241   //     /    \
00242   //    21     9
00243   //  /  \    /
00244   // 7   14  9
00245   //        / \
00246   //       1  13
00247   bintree<int> pesoArb(30);
00248   pesoArb.insert_left(pesoArb.root(), 21);
00249   pesoArb.insert_right(pesoArb.root(), 9);
00250
00251   pesoArb.insert_left(pesoArb.root().left(), 7);
00252   pesoArb.insert_right(pesoArb.root().left(), 14);
00253
00254   pesoArb.insert_left(pesoArb.root().right(), 9);
00255
00256   pesoArb.insert_left(pesoArb.root().right().left(), 1);
00257   pesoArb.insert_right(pesoArb.root().right().left(), 8);
00258
00259   cout « "Preorden:";
00260
00261   for (bintree<int>::preorder_iterator i = pesoArb.begin_preorder(); i!=pesoArb.end_preorder(); ++i)
00262     cout « *i « " ";
00263
00264   cout « endl;
00265
00266
00267   cout « "Inorden:";
00268
00269   for (bintree<int>::inorder_iterator i = pesoArb.begin_inorder(); i!=pesoArb.end_inorder(); ++i)
00270     cout « *i « " ";
00271
00272   cout « endl;
00273
00274
00275   cout « "Postorden:";
00276
00277   for (bintree<int>::postorder_iterator i = pesoArb.begin_postorder(); i!=pesoArb.end_postorder();
      ++i)
00278     cout « *i « " ";
00279
00280   cout « endl;
00281
00282   cout « "Por Niveles:";
00283
00284   for (bintree<int>::level_iterator i = pesoArb.begin_level(); i!=pesoArb.end_level(); ++i)
00285     cout « *i « " ";
00286   cout « endl;
00287   if(pesointerior(pesoArb))
00288      cout « "Es pesointerior" « endl;
00289   else
00290      cout « "No es pesointerior" « endl;
00291
00292   separador();
00293
00295
00296   cout « "Funcion 24: sumaparantec:" « endl;
00297   //       10
00298   //      /  \
00299   //     21   6
00300   //         / \
00301   //        8  11
00302   //          / \
```

```
00303   //         12 6
00304   bintree<int> sumaArb(10);
00305   sumaArb.insert_left(sumaArb.root(), 21);
00306   sumaArb.insert_right(sumaArb.root(), 6);
00307
00308   sumaArb.insert_left(sumaArb.root().right(), 8);
00309   sumaArb.insert_right(sumaArb.root().right(), 11);
00310
00311   sumaArb.insert_left(sumaArb.root().right().right(), 12);
00312   sumaArb.insert_right(sumaArb.root().right().right(),6);
00313
00314   cout « "Preorden:";
00315
00316   for (bintree<int>::preorder_iterator i = sumaArb.begin_preorder(); i!=sumaArb.end_preorder(); ++i)
00317     cout « *i « " ";
00318
00319   cout « endl;
00320
00321
00322   cout « "Inorden:";
00323
00324   for (bintree<int>::inorder_iterator i = sumaArb.begin_inorder(); i!=sumaArb.end_inorder(); ++i)
00325     cout « *i « " ";
00326
00327   cout « endl;
00328
00329
00330   cout « "Postorden:";
00331
00332   for (bintree<int>::postorder_iterator i = sumaArb.begin_postorder(); i!=sumaArb.end_postorder();
      ++i)
00333     cout « *i « " ";
00334
00335   cout « endl;
00336
00337   cout « "Por Niveles:";
00338
00339   for (bintree<int>::level_iterator i = sumaArb.begin_level(); i!=sumaArb.end_level(); ++i)
00340     cout « *i « " ";
00341   cout « endl;
00342   bintree<int>::node n = sumaArb.root();
00343   cout « "La suma de parantec es " « sumaparantec(sumaArb,n) « endl;
00344
00345   separador();
00346
00348
00349   cout « "Funcion 26: camino_ord:" « endl;
00350   //        3
00351   //      /   \
00352   //     6     7
00353   //    / \   / \
00354   //   1   3 9   5
00355   bintree<int> caminoOrdArb(3);
00356   caminoOrdArb.insert_left(caminoOrdArb.root(), 6);
00357   caminoOrdArb.insert_right(caminoOrdArb.root(), 7);
00358
00359   caminoOrdArb.insert_left(caminoOrdArb.root().left(), 1);
00360   caminoOrdArb.insert_right(caminoOrdArb.root().left(), 3);
00361
00362   caminoOrdArb.insert_left(caminoOrdArb.root().right(), 9);
00363   caminoOrdArb.insert_right(caminoOrdArb.root().right(), 5);
00364
00365   cout « "Preorden:";
00366
00367   for (bintree<int>::preorder_iterator i = caminoOrdArb.begin_preorder();
      i!=caminoOrdArb.end_preorder(); ++i)
00368     cout « *i « " ";
00369
00370   cout « endl;
00371
00372
00373   cout « "Inorden:";
00374
00375   for (bintree<int>::inorder_iterator i = caminoOrdArb.begin_inorder(); i!=caminoOrdArb.end_inorder();
      ++i)
00376     cout « *i « " ";
00377
00378   cout « endl;
00379
00380
00381   cout « "Postorden:";
00382
00383   for (bintree<int>::postorder_iterator i = caminoOrdArb.begin_postorder(); i !=
      caminoOrdArb.end_postorder(); ++i)
00384     cout « *i « " ";
00385
00386   cout « endl;
```

```
00387
00388   cout « "Por Niveles:";
00389
00390   for (bintree<int>::level_iterator i = caminoOrdArb.begin_level(); i!=caminoOrdArb.end_level(); ++i)
00391     cout « *i « " ";
00392   cout « endl;
00393   bintree<int>::node nOrdAbr = caminoOrdArb.root();
00394   if(camino_ord(caminoOrdArb, nOrdAbr))
00395      cout « "Existe un camino ordenado" « endl;
00396   else
00397      cout « "No existe camino ordenado" « endl;
00398
00399   separador();
00400   return 0;
00401 }
00402
```

Index