CONECTA 4

V1.0

Generado por Doxygen 1.8.11

Índice

1. Práctica Final: CONECTA4

Versión

v1

Autor

Luis Baca, Óscar Gómez, Carmen Navarro y Carlos Cano

1.1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan.

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos jerárquicas: Árboles

1.2. Objetivos

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin, el estudiante abordará un problema donde se requieren estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

1.3. Problema

El estudiante debe implementar un programa que simule el juego "Conecta 4" [Conecta4]. El objetivo de Conecta 4 es alinear cuatro fichas sobre un tablero formado por seis filas y siete columnas. Cada jugador dispone de 21 fichas de un color (por lo general, rojas o amarillas). En nuestro caso, las fichas de los jugadores se indicarán con los caracteres "X" y "O". Por turnos, los jugadores deben introducir una ficha en la columna que prefieran (siempre que no esté completa) y ésta caerá a la posición más baja. Gana la partida el primero que consiga alinear cuatro fichas consecutivas de un mismo color en horizontal, vertical o diagonal. Si todas las columnas están llenas pero nadie ha hecho una fila válida, hay empate.

El estudiante debe implementar una versión de este juego en el que los movimientos de uno de los jugadores se efectúan de forma automática. Esta modalidad, que llamaremos "Player VS. Computer" o "1 jugador automático", requiere implementar un mecanismo de decisión automática basado en la disposición actual del tablero. Para implementar este mecanismo de decisión automática, debemos representar todos los posibles movimientos a partir del tablero actual y definir una "métrica" que nos permita "evaluar" cuál de los movimientos posibles deriva un tablero con más posibilidad de éxito para el jugador automático. Cuanto mejor sea esta métrica, más difícil será batir al jugador automático en una partida de Conecta 4.

Por ejemplo, la métrica más sencilla consistiría en evaluar si alguno de los movimientos posibles para el jugador automático produce directamente un tablero donde el jugador automático consigue 4 en línea (y, por tanto, gana la partida) o evita un 4 en línea del rival (y, por tanto, evita perder la partida). Sin embargo, esta métrica sólo nos permitiría evaluar ciertos tableros, dejándonos "a ciegas" para tomar una decisión en tableros que no derivan una victoria o derrota en un sólo movimiento. **Otras métricas más sofisticadas requieren evaluar todos los posibles movimientos a partir de un tablero dado, contemplando varios turnos de los dos jugadores. Una representación natural para almacenar esta información es un Árbol**. El nodo raíz representa el tablero actual y le añadimos un hijo por cada posible tablero que se deriva del padre con un movimiento del jugador que tiene el turno. De este modo, el nivel 1 del árbol representa todos los movimientos posibles de uno de los jugadores, el nivel 2 todos movimientos posibles del otro jugador para cada uno de los tableros de nivel 1, y así sucesivamente. Cada rama del árbol termina en una hoja en la que, o bien se da la victoria de uno de los jugadores, o bien hay empate (no quedan movimientos posibles). Las siguientes figuras ilustran la estructura interna de un TDA Árbol General que almacena los tableros de Conecta 4 con la disposición de la partida en cada momento. La primera figura incide en la estructura interna del árbol y la segunda en posibles tableros almacenados en el árbol.

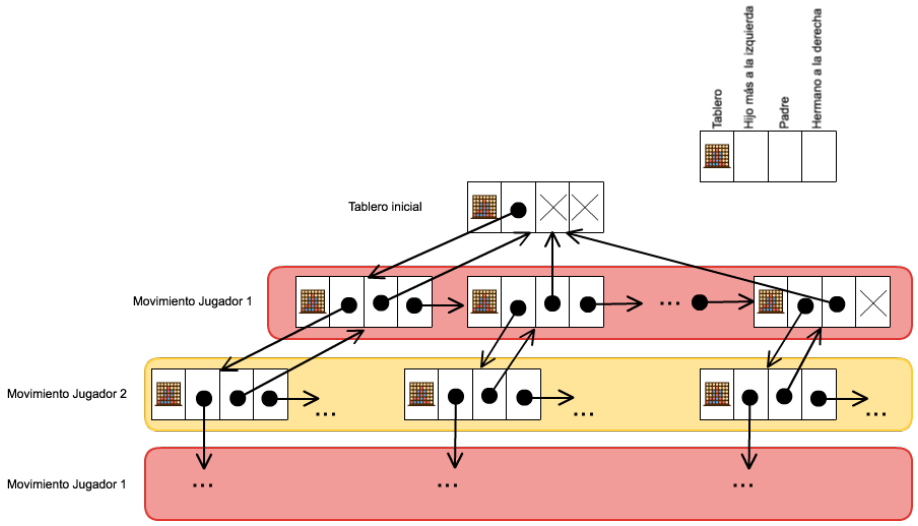

Figura 1 Estructura interna de un TDA Árbol General para representar el espacio de soluciones de Conecta 4

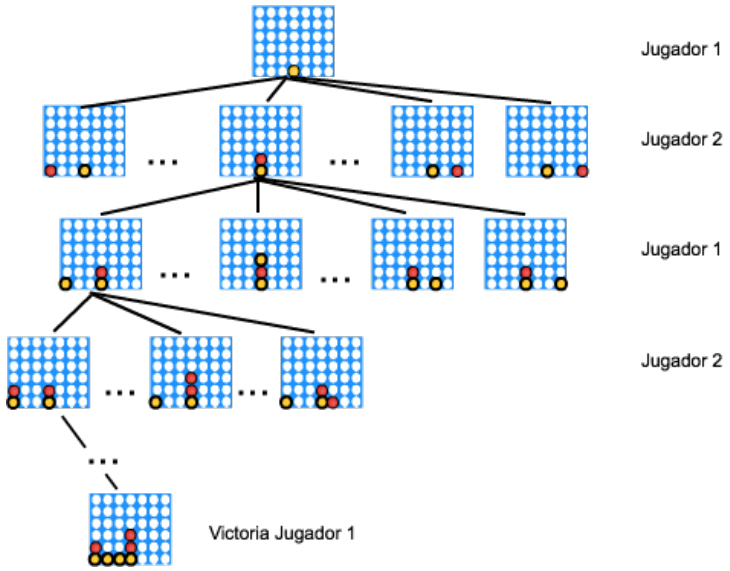

Figura 2 Detalle de posibles movimientos en el espacio de soluciones de Conecta 4

El tamaño real de un tablero de Conecta 4 es de 6 filas x 7 columnas, con lo que cada nodo del árbol tiene hasta 7 hijos. Si una columna del tablero estuviera completa, habría menos movimientos posibles y por tanto menos hijos para ese nodo. Los nodos en los que la partida ya está resuelta (por victoria de uno de los jugadores o por no quedar movimientos posibles) serían los nodos hoja del árbol.

De este modo, en el primer nivel del árbol (considerando 1 turno desde el tablero actual), almacenaríamos hasta 7 jugadas posibles o tableros diferentes. En el segundo nivel (considerando 2 turnos desde el actual), almacenaríamos hasta 7^2 tableros posibles. En el tercero, hasta 7^3. Y así sucesivamente. El estudiante debe ser consciente del coste de almacenar y recorrer este volumen de tableros para implementar métricas de recomendación de movimientos que sean viables (coste razonable en tiempo y memoria). Por ejemplo, se recomienda implementar métricas que utilicen un árbol de movimientos para los próximos N turnos desde el tablero actual, donde N queda a elección del estudiante, para acotar el consumo en tiempo y memoria de la implementación de estas métricas.

Además, a medida que el juego va avanzando, se recomienda reutilizar las ramas del árbol de soluciones que siguen siendo posibles y utilizar mecanismos de poda (ya implementados en el TDA ArbolGeneral) para liberar espacio en memoria eliminando las ramas con movimientos que no se han tomado.

En esta práctica, el estudiante deberá proponer e implementar distintas métricas para guiar la decisión del jugador automático, representando todas las jugadas posibles para los próximos N turnos utilizando un T.D.A. Árbol General.

1.4. Tareas a realizar

Se proprocionan:

1. T.D.A. Tablero: representación del tablero de Conecta 4. Un objeto tablero representa un instante dado de una partida del Conecta 4, es decir, la disposición de las fichas de ambos jugadores sobre el tablero y el turno del jugador al que le corresponde hacer el próximo movimiento.
2. T.D.A. Mando: representación gráfica del tablero de Conecta 4. Incluye la gestión de Entrada/Salida que permite jugar a Conecta 4 de forma interactiva.
3. T.D.A. ArbolGeneral.

Se deberán llevar a cabo las siguientes tareas:

1. Construir el T.D.A. Conecta4, que almacena todos los tableros posibles de Conecta 4 generados en los próximos N turnos a partir de un tablero inicial utilizando las clases T.D.A. ArbolGeneral y T.D.A. Tablero.
2. Definir los métodos del T.D.A. Conecta4 para la solución de los problemas propuestos, en particular, para la implementación de un jugador automático.
3. Probar los módulos con programas test. Se puede usar la STL en todos los módulos excepto en la implementación del TDA ArbolGeneral.

A continuación se detallan los programas que se deberán desarrollar.

1.5. Implementar un jugador automático

Conecta 4 es un juego de estrategia abstracta donde los contrincantes disponen de información perfecta [Conecta4]. Por norma general, el primer jugador tiene más posibilidades de ganar si introduce la primera ficha en la columna central. Si lo hace en las contiguas se puede forzar un empate, mientras que si la mete en las más alejadas del centro su rival puede vencerle con mayor facilidad. Existen libros y webs donde se explican las mejores estrategias para ganar en el Conecta 4, por ejemplo puede consultarse [Allen13].

El estudiante debe implementar distintas "métricas" que permitan al jugador automático decidir qué movimiento realizar dada una disposición del tablero. Las métricas implementadas analizarán el espacio de soluciones posible a partir del tablero actual para evaluar cuál es el movimiento más beneficioso para el jugador automático (esto es, el movimiento que más aumente las expectativas de ganar la partida). La calidad de estas métricas dependerá, en parte, de la cantidad de información de que dispongan. De este modo, una métrica que sólo considere dos niveles en profundidad a partir del tablero actual (esto es, todos los movimientos posibles del jugador y todas las repuestas inmediatas del rival) tendrá más limitaciones que una métrica que disponga de todo el árbol de soluciones a partir del tablero actual. Evidentemente, la contrapartida es que una métrica que dispone de todo el árbol de soluciones requiere de un alto consumo en memoria y en tiempo de cómputo (según el tamaño del tablero, almacenar todo el árbol de soluciones puede ser directamente inviable). De este modo, se propone diseñar distintas métricas

utilizando un árbol de soluciones con N niveles en profundidad a partir del tablero actual, donde N es variable y su elección se deja al estudiante.

El estudiante debe implementar la clase TDA Conecta4 utilizando como tipo rep. el TDA. ArbolGeneral para representar el espacio de soluciones con profundidad N e implementar distintas métricas que, haciendo uso de esta representación del espacio de soluciones, recomienden un movimiento al jugador automático. Anteriormente se ha descrito una posible métrica inicial que sólo requiere explorar un nivel del espacio de soluciones: evaluar si alguno de los movimientos posibles para el jugador automático produce directamente un tablero donde el jugador automático consigue 4 en línea (y, por tanto, gana la partida) o evita un 4 en línea del rival (y, por tanto, evita perder la partida). Otra métrica sencilla que requiere explorar todo el árbol de búsqueda hasta profundidad N consiste en hacer recuento de cuántos nodos a profundidad <=N son favorables/desfavorables (victorias/empates/derrotas) al jugador automático para cada nodo hijo del tablero actual, de modo que el hijo con mejores métricas indicará el mejor movimiento posible. Otras métricas más sofisticadas podrían explorar este espacio de búsqueda para considerar no sólo victorias o derrotas, sino también recontar el número de secuencias de tres/dos fichas del jugador automático no rodeadas por fichas del rival en cada uno de los tableros.

Es importante destacar que, para aliviar el costo de almacenamiento de las métricas basadas en árboles de búsqueda, una vez realizado un movimiento, el árbol debe podarse para eliminar los nodos asociados a movimientos alternativos que finalmente no fueron tomados. El TDA ArbolGeneral incluye métodos de poda facilitan esta tarea.

El estudiante tiene libertad para implementar las métricas sugeridas o cualquier otra para guiar los movimientos del jugador automático. La documentación final a entregar debe recoger una descripción completa de cada métrica implementada y una justificación de sus resultados. Se valorará la dificultad para batir al jugador automático guiado por las métricas implementadas por el estudiante.

En la siguiente sección se detalla cómo implementar este programa principal para jugar a Conecta 4.

1.6. Implementar una partida de Conecta 4

Se adjunta un programa de ejemplo (`src/conecta4.cpp`) que utiliza los TDA Tablero y TDA Mando para implementar una partida de Conecta 4 interactiva entre dos jugadores (no automáticos). Nótese que este programa va pidiendo por teclado sucesivamente cada movimiento de ambos jugadores, y no hace uso del TDA ArbolGeneral (no representa el espacio de soluciones posibles para emitir recomendaciones sobre el mejor movimiento o tomar decisiones automáticas).

```
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <stdio.h>
#include <unistd.h>
#include <termio.h>         // Linux/Windows users
//#include <termios.h>      // Mac OSX users

#include "ArbolGeneral.h"
#include "tablero.h"
#include "mando.h"

using namespace std;

// Captura el caracter pulsado por teclado (sin necesidad de pulsar, a continuación, Enter).
// Devuelve: Caracter pulsado.

char getch() {
 ...
}

// Imprime en pantalla el tablero completo, con el mando y el jugador.
// t : Tablero que se va a imprimir.
// m : Mando indicando la posición del jugador.

void imprimeTablero(Tablero & t, Mando & m){
    cout << m.GetJugador() << endl;
    cout << t ;
    cout << m.GetBase() << endl;
    cout << m.GetMando() << endl;
```

```
}


// Implementa el desarrollo de una partida de Conecta 4 con dos jugadores humanos
// Devuelve: identificador (int) del jugador que gana la partida
int jugar_partida() {

    Tablero tablero(5, 7);      //Tablero 5x7
    Mando mando(tablero);       //Mando para controlar E/S de tablero
    char c = 1;
    int quienGana = tablero.quienGana();
    //mientras no haya ganador y no se pulse tecla de terminación
    while(c != Mando::KB_ESCAPE && quienGana == 0) {
        system("clear");
        mando.actualizarJuego(c, tablero);  // actualiza tablero según comando c
        imprimeTablero(tablero, mando);     // muestra tablero y mando en pantalla
        quienGana = tablero.quienGana();    // hay ganador?
        if(quienGana==0) c = getch();       // Capturamos la tecla pulsada.
    }

    return tablero.quienGana();
}

int main(int argc, char *argv[]){
    int ganador = jugar_partida();
    cout « "Ha ganado el jugador " « ganador « endl;
}
```

El estudiante debe implementar una partida de Conecta 4 en la que uno de los jugadores realiza sus movimientos automáticamente y el otro jugador introduce sus movimientos por teclado. Utilice el código de `src/conecta4.cpp` como base para programar este módulo. El programa debe permitir elegir cuál es el jugador que inicia la partida: el jugador automático o el humano.

Además, en caso de implementar distintas métricas que guían al jugador automático, el programa contará con un argumento entero que permite identificar la métrica utilizada por el jugador automático (métrica 1, métrica 2, y así sucesivamente). La métrica 1 será, por defecto, la que mejor resultados haya obtenido para el estudiante.

Finalmente, el programa debe también permitir definir el tamaño del tablero.

En resumen, el módulo a desarrollar por el estudiante debe tener los siguientes argumentos:

```
prompt %> conecta4 <filas_tablero> <cols_tablero> <metrica> <turno>
```

Donde:

- `<filas_tablero>, <cols_tablero>`: especifica las dimensiones del tablero (por defecto, 4 en ambas opciones para tablero 4x4)
- `<metrica>`: indica la métrica a utilizar para guiar al jugador automático. Comenzar a numerar con 1 y en adelante, en orden de mejores a peores resultados obtenidos con esa métrica para el jugador automático. 1 es la opción por defecto (la métrica con mejores resultados). 0 para que la partida se desarrolle entre dos jugadores humanos (sin jugador automático).
- `<turno>` indica qué jugador tiene el primer turno:
  - 1: primer turno para el jugador humano
  - 2: primer turno para el jugador automático

1.7. Manejo del T.D.A. ArbolGeneral

Se adjunta un programa de ejemplo (`src/arboltablero_test.cpp`) en el que se exhibe alguna funcionalidad asociada a los TDA Tablero y ArbolGeneral. En este programa se crean distintos objetos Tablero con distintas jugadas posibles de Conecta 4 y una estructura ArbolGeneral que almacena jerárquicamente estos tableros.

En particular, se muestra como:

Para TDA Tablero:

- crear de tableros con las dimensiones deseadas
- colocar ficha para el jugador que dispone del turno
- cambiar turno entre jugadores

Para TDA ArbolGeneral:

- crear de árboles con TDA Tablero como nodos
- obtener el nodo raiz de un ArbolGeneral
- insertar nodos como hijomasizquierda
- insertar nodos como hermanoderecha
- imprimir ArbolGeneral en preorden
- podar una rama del ArbolGeneral

```
#include <iostream>
#include "ArbolGeneral.h"
#include "tablero.h"
#include <string>

using namespace std;


int main(int argc, char *argv[]){

  //Tablero vacío 5x7
    Tablero tablero(5, 7);

    //Manualmente se insertan algunos movimientos:
    tablero.colocarFicha(3);  //Jugador 1 inserta ficha en columna 3
    tablero.cambiarTurno();
    tablero.colocarFicha(1);  //Jugador 2 inserta ficha en columna 1
    tablero.cambiarTurno();
    tablero.colocarFicha(3);  //Jugador 1 inserta ficha en columna 3.
    tablero.cambiarTurno();

    //Se muestra el tablero
    cout « "Tablero obtenido tras tres movimientos: \n"«tablero;

    //A partir de la situación actual del tablero, montamos un árbol para estudiar algunas posibilidades.

    // Éste es el árbol que queremos montar:
    //         tablero
    //           |
    //      |---------------|
    //    tablero1       tablero2
    //                       |
    //                   tablero3


    //Árbol 'partida', con 'tablero' como nodo raíz
    ArbolGeneral<Tablero> partida(tablero);

    //Estudio opciones a partir de tablero: Jugador 2 coloca ficha en columna 1. (tablero1)
    Tablero tablero1(tablero);          //tablero queda sin modificar
    tablero1.colocarFicha(1);
    ArbolGeneral<Tablero> arbol1 (tablero1);  //creo árbol con un nodo (tablero1)
```

```
    //Otra opción: Jugador 2 coloca ficha en columna 2. (tablero2)
    Tablero tablero2(tablero);          //tablero queda sin modificar
    tablero2.colocarFicha(2);
    ArbolGeneral<Tablero> arbol2(tablero2);   //creo árbol con un nodo

    // Sobre la última opción, ahora contemplo la posibilidad de que
    //  Jugador 1 coloque ficha también en columna 2.
    tablero2.cambiarTurno();          //modifico tablero2 (esta modificación sería tablero3)
    tablero2.colocarFicha(2);
    ArbolGeneral<Tablero> arbol3 (tablero2);  //creo árbol con un nodo
    arbol2.insertar_hijomasizquierda(arbol2.raiz(), arbol3);  //añado este árbol como hijo de arbol2

    // Inserto arbol1 y arbol2 como hijos de partida.
    // arbol1 es el hijo más a la izquierda y arbol2 es hermano a la derecha de arbol1

    //  Forma de hacerlo A: inserto varios hijomasizquierda en el orden inverso al deseado
    //  partida.insertar_hijomasizquierda(partida.raiz(), arbol2);
    //  partida.insertar_hijomasizquierda(partida.raiz(), arbol1);  //hijomasizquierda desplaza al anterior
       a la derecha

    // Forma de hacerlo B: inserto un hijomasizquierda y hermanoderecha
    partida.insertar_hijomasizquierda(partida.raiz(), arbol1);              //inserto un hijomasizquierda
    partida.insertar_hermanoderecha(partida.hijomasizquierda(partida.raiz()), arbol2);  //le inserto un
       hermanoderecha


    // Recorremos en preorden para comprobar el arbol 'partida' resultante
    cout « "\nÁrbol en preorden: \n"«endl;
    partida.recorrer_preorden();

    // Podamos el hijomasizquierda y recorremos en preorden:
    ArbolGeneral<Tablero> rama_podada;
    partida.podar_hijomasizquierda(partida.raiz(), rama_podada);

    cout « "\nRecorrido preorden después de podar hijomasizquierda: \n"«endl;
    partida.recorrer_preorden();

    return 0;
}
```

El estudiante deber revisar detenidamente este código y la documentación completa asociada a las clases TDA Tablero y TDA ArbolGeneral para familiarizarse con su manejo y sacarles el máximo partido para la implementación del T.D.A. Conecta4.

1.8. Recomendaciones

- Analice con detenimiento la documentación de las clases proporcionadas: T.D.A. ArbolGeneral, T.D.A. Tablero y T.D.A. Mando. Parte importante del esfuerzo que debe realizar en esta práctica es entender cómo debe utilizar estas clases para su proyecto. Esta práctica es habitual en empresas: para el desarrollo de un proyecto se cuenta con parte del código desarrollado previamente por otro equipo.
- Para acotar el espacio de búsqueda y el tamaño de los objetos ArbolGeneral, comience el desarrollo de su proyecto considerando tableros más pequeños (por ejemplo, 4x4). Cuando el funcionamiento sea correcto, amplíe progresivamente el tamaño de los tableros. Del mismo modo, considere con cautela la profundidad máxima N que va a explorar en el árbol de soluciones. Comience con un valor bajo de N y estudie hasta qué valor de N (depende del tamaño del tablero) es viable el cálculo de sus métricas.

1.9. Entrega

El estudiante deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "conecta4.tgz" y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes. El estudiante debe incluir el archivo Makefile para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios:

conecta4

- Makefile
- include – Carpeta con ficheros de cabecera (.h)
- src –Carpeta con código fuente (.cpp)
- doc –Carpeta con Documentación
- obj – Carpeta para código objeto (.o)
- bin – Carpeta para ejecutables
- datos – Carpeta para ficheros de datos

Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "conecta4") para ejecutar:

```
prompt% tar zcv conecta4.tgz conecta4
```

tras lo cual, dispondrá de un nuevo archivo conecta4.tgz que contiene la carpeta conecta4 así como todas las carpetas y archivos que cuelgan de ella.

La fecha límite de entrega es el día 22 de Enero de 2017 a las 23:59.

1.10. Referencias

[Allen13] James D. Allen. "Expert Play in Connect-Four" (en inglés). Archivado desde el original el 4 de noviembre de 2015. http://web.archive.org/web/20141125065033/http://homepages.cwi.↩nl/∼tromp/c4.html

[Conecta4] https://es.wikipedia.org/wiki/Conecta_4

[GAR06b] Garrido, A. Fdez-Valdivia, J. "Abstracción y estructuras de datos en C++". Delta publicaciones, 2006.

2. Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

3. Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

4. Documentación de las clases

4.1. Referencia de la plantilla de la Clase ArbolGeneral< Tbase >

T.D.A. ArbolGeneral.

```
#include <ArbolGeneral.h>
```

Clases

- class inorden_iterador
- struct nodo

  repConjunto Rep del TDA ArbolGeneral
- class postorden_iterador
- class preorden_iterador
- class reverse_preorden_iterador

Tipos públicos

- typedef struct nodo * Nodo

  Tipo Nodo.

Métodos públicos

- ArbolGeneral ()

  Constructor por defecto.
- ArbolGeneral (const Tbase &e)

  Constructor de raíz.
- ArbolGeneral (const ArbolGeneral< Tbase > &v)

  Constructor de copias.
- ~ArbolGeneral ()

  Destructor.
- ArbolGeneral< Tbase > & operator= (const ArbolGeneral< Tbase > &v)

  Operador de asignación.
- void AsignaRaiz (const Tbase &e)

  Asignar nodo raíz.
- Nodo raiz () const

  Raíz del árbol.
- Nodo hijomasizquierda (const Nodo n) const

  Hijo más a la izquierda.
- Nodo hermanoderecha (const Nodo n) const

  Hermano derecha.
- Nodo padre (const Nodo n) const

  Nodo padre.
- Tbase & etiqueta (const Nodo n)

  Etiqueta de un nodo.
- const Tbase & etiqueta (const Nodo n) const

  Etiqueta de un nodo.
- void asignar_subarbol (const ArbolGeneral< Tbase > &orig, const Nodo nod)

  Copia subárbol.
- void podar_hijomasizquierda (Nodo n, ArbolGeneral< Tbase > &dest)

  Podar subárbol hijo más a la izquierda.
- void podar_hermanoderecha (Nodo n, ArbolGeneral< Tbase > &dest)

  Podar subárbol hermano derecha.
- void insertar_hijomasizquierda (Nodo n, ArbolGeneral< Tbase > &rama)

  Insertar subárbol hijo más a la izquierda.
- void insertar_hermanoderecha (Nodo n, ArbolGeneral< Tbase > &rama)

  Insertar subárbol hermano derecha.
- void clear ()

  Borra todos los elementos.
- int size () const

  Número de elementos.
- bool empty () const

  Vacío.
- int altura (Nodo t) const

  Altura de un nodo.
- void reflejado (Nodo t)

  Arbol reflejado.
- bool operator== (const ArbolGeneral< Tbase > &v) const

  Operador de comparación (igualdad)
- bool operator!= (const ArbolGeneral< Tbase > &v) const

  Operador de comparación (diferencia)
- void recuperar_arbol (string preorden, string inorden, string postorden, Nodo nuevo)

Crea un arbol a partir de sus tres recorridos.

- preorden_iterador beginpreorden () const

  Comienzo de un iterador preorden_iterador.
- preorden_iterador endpreorden () const

  Final de un iterador preorden_iterador.
- void recorrer_preorden () const

  Recorrido en preorden. Muestra el recorrido del árbol en preorden usando iteradores.
- void recorrer_preorden2 (Nodo t) const

  Recorrido en preorden.
- void recorrer_preorden_al_reves () const

  Recorrido en preorden al reves. Muestra el recorrido del árbol en preorden, pero al reves. Utilizando iteradores preorden.
- reverse_preorden_iterador beginreverse_preorden () const

  Comienzo de un iterador reverse_preorden_iterador.
- reverse_preorden_iterador endreverse_preorden () const

  Final de un iterador reverse_preorden_iterador.
- void recorrer_reverse_preorden () const

  Recorrido en preorden invertido. Muestra el recorrido del árbol en preorden invertido usando iteradores.
- void recorrer_reverse_preorden_al_reves () const

  Recorrido en preorden invertido, al reves. Recorrido en preorden normal.
- inorden_iterador begininorden () const

  Comienzo de un iterador inorden_iterador.
- inorden_iterador endinorden () const

  Final de un iterador inorden_iterador.
- void recorrer_inorden () const

  Recorrido en inorden. Muestra el árbol recorriendolo en inorden usando iteradores.
- void recorrer_inorden2 (Nodo t) const

  Recorrido en inorden.
- postorden_iterador beginpostorden () const

  Primer elemento en postorden.
- postorden_iterador endpostorden () const

  Último elmento.
- void recorrer_postorden () const

  Recorrido en postorden. Recorre el árbol en postorden. Desde la raíz.
- void recorrer_postorden2 (Nodo t) const

  Recorrido postorden.
- void recorrer_por_niveles (Nodo t) const

  Recorrido por niveles.

Métodos privados

- void destruir (nodo *n)

  Destruye el subárbol.
- void copiar (nodo *&dest, nodo *orig)

  Copia un subárbol.
- int contar (const nodo *n) const

  Cuenta el número de nodos.
- int contar_Hijos (nodo *n) const

  Cuenta el número de hijos.
- bool soniguales (const nodo *n1, const nodo *n2) const

Comprueba igualdad de dos subárboles.

- void escribe_arbol (std::ostream &out, nodo *nod) const

  Escribe un subárbol.

- void lee_arbol (std::istream &in, nodo *&nod)

  Lee un subárbol.

Atributos privados

- struct nodo * laraiz

  Puntero a la raíz.

Amigas

- template<class T >
  std::istream & operator>> (std::istream &in, ArbolGeneral< T > &v)

  Operador de extracción de flujo.

- template<class T >
  ostream & operator<< (ostream &out, const ArbolGeneral< T > &v)

  Operador de inserción en flujo.

4.1.1. Descripción detallada

template<class Tbase>
class ArbolGeneral< Tbase >

T.D.A. ArbolGeneral.

Definición: Una instancia *a* del tipo de dato abstracto ArbolGeneral sobre un dominio *Tbase* se puede construir como

- Un objeto vacío (árbol vacío) si no contiene ningún elemento. Lo denotamos {}.
- Un árbol que contiene un elemento destacado, el nodo raíz, con un valor *e* en el dominio *Tbase* (denominado *etiqueta*), y *k* subárboles $(T_1, \ldots, T_k)$ del T.D.A. ArbolGeneral sobre *Tbase*.

Se establece una relación *padre-hijomasalaizquierda-hermanoaladerecha* entre cada nodo y los nodos raíz de los subárboles (si los hubiera) que cuelgan de él.

Para poder usar el tipo de dato ArbolGeneral se debe incluir el fichero

```
#include ArbolGeneral.h
```

El espacio requerido para el almacenamiento es O(n), donde n es el número de nodos del árbol.

Autor

Luis Baca Ruiz.

Fecha

Diciembre de 2011

4.1.2. Documentación de los 'Typedef' miembros de la clase

4.1.2.1. template<class Tbase> typedef struct nodo* ArbolGeneral< Tbase >::Nodo

Tipo Nodo.

Este tipo nos permite manejar cada uno de los nodos del árbol. Los valores que tomará serán tantos como nodos en el árbol (para poder referirse a cada uno de ellos) y además un valor destacado *nulo* (0), que indica que no se refiere a ninguno de ellos.

Una variable *n* de este tipo se declara

```
ArbolGeneral::Nodo n;
```

Las operaciones válidas sobre el tipo nodo son:

- Operador de Asignación (=).
- Operador de comprobación de igualdad (==).
- Operador de comprobación de desigualdad (!=).

4.1.3. Documentación del constructor y destructor

4.1.3.1. template<class Tbase > ArbolGeneral< Tbase >::ArbolGeneral ( )

Constructor por defecto.

Reserva los recursos e inicializa el árbol a vacío {}. La operación se realiza en tiempo O(1).

4.1.3.2. template<class Tbase > ArbolGeneral< Tbase >::ArbolGeneral (const Tbase & *e*)

Constructor de raíz.

Parámetros

e	Etiqueta de la raíz

Reserva los recursos e inicializa el árbol con un único nodo raíz que tiene la etiqueta *e*, es decir, el árbol {e, {}, {}}. La operación se realiza en tiempo O(1).

4.1.3.3. template<class Tbase > ArbolGeneral< Tbase >::ArbolGeneral (const ArbolGeneral< Tbase > & *v*)

Constructor de copias.

Parámetros

v	ArbolGeneral a copiar

Construye el árbol duplicando el contenido de *v* en el árbol receptor. La operación se realiza en tiempo O(n), donde

n es el número de elementos de *v*.

4.1.3.4. template<class Tbase > ArbolGeneral< Tbase >::~ArbolGeneral ( )

Destructor.

Libera los recursos ocupados por el árbol receptor. La operación se realiza en tiempo O(n), donde n es el número de elementos del árbol receptor.

4.1.4. Documentación de las funciones miembro

4.1.4.1. template<class Tbase > int ArbolGeneral< Tbase >::altura (Nodo *t*) const

Altura de un nodo.

Parámetros

t	Nodo que se calculará la altura.

Devuelve

Devuelve la altura de un nodo.

4.1.4.2. template<class Tbase > void ArbolGeneral< Tbase >::asignar_subarbol (const ArbolGeneral< Tbase > & *orig,* const Nodo *nod*)

Copia subárbol.

Parámetros

orig	Árbol desde el que se va a copiar una rama
nod	Nodo raíz del subárbol que se copia.

Precondición

nod es un nodo del árbol *orig* y no es nulo

El árbol receptor acaba con un valor copia del subárbol que cuelga del nodo *nod* en el árbol *orig*. La operación se realiza en tiempo O(n), donde *n* es el número de nodos del subárbol copiado.

4.1.4.3. template<class Tbase > void ArbolGeneral< Tbase >::AsignaRaiz (const Tbase & *e*)

Asignar nodo raíz.

Parámetros

e	Etiqueta a asignar al nodo raíz

Vacía el árbol receptor y le asigna como valor el árbol de un único nodo cuya etiqueta es *e*.

4.1.4.4. template<class Tbase > ArbolGeneral< Tbase >::inorden_iterador ArbolGeneral< Tbase >::begininorden ( ) const

Comienzo de un iterador inorden_iterador.

Devuelve

un iterador de tipo inorden apuntando a la raiz.

4.1.4.5. template<class Tbase > ArbolGeneral< Tbase >::postorden_iterador ArbolGeneral< Tbase >::beginpostorden ( ) const

Primer elemento en postorden.

Devuelve

Devuelve un iterador al primer elemento en postorden.

4.1.4.6. template<class Tbase > ArbolGeneral< Tbase >::preorden_iterador ArbolGeneral< Tbase >::beginpreorden ( ) const

Comienzo de un iterador preorden_iterador.

Devuelve

un iterador de tipo preorden apuntando a la raiz.

4.1.4.7. template<class Tbase > ArbolGeneral< Tbase >::reverse_preorden_iterador ArbolGeneral< Tbase >::beginreverse_preorden ( ) const

Comienzo de un iterador reverse_preorden_iterador.

Devuelve

un iterador de tipo reverse_preorden apuntando al penultimo elemento.

4.1.4.8. template<class Tbase > void ArbolGeneral< Tbase >::clear ( )

Borra todos los elementos.

Borra todos los elementos del árbol receptor. Cuando termina, el árbol está vacía. La operación se realiza en tiempo O(n), donde *n* es el número de elementos del árbol receptor.

4.1.4.9. template<class Tbase > int ArbolGeneral< Tbase >::contar (const nodo * *n*) const [private]

Cuenta el número de nodos.

Parámetros

n	Nodo del que cuelga el subárbol de nodos a contabilizar.

Cuenta cuántos nodos cuelgan de *n*, incluido éste.

4.1.4.10. template<class Tbase > int ArbolGeneral< Tbase >::contar_Hijos (nodo * *n*) const [private]

Cuenta el número de hijos.

Parámetros

n	Nodo padre del que cuelgan los hijos.

Devuelve

Número de hijos que tiene ese nodo.

4.1.4.11. template<class Tbase > void ArbolGeneral< Tbase >::copiar (nodo *& *dest,* nodo * *orig*) [private]

Copia un subárbol.

Parámetros

dest	Referencia al puntero del que cuelga la copia
orig	Puntero a la raíz del subárbol a copiar

Hace una copia de todo el subárbol que cuelga de *orig* en el puntero *dest*. Es importante ver que en *dest->padre* (si existe) no se asigna ningún valor, pues no se conoce.

4.1.4.12. template<class Tbase > void ArbolGeneral< Tbase >::destruir (nodo * *n*) [private]

Destruye el subárbol.

Parámetros

n	Nodo a destruir, junto con sus descendientes

Libera los recursos que ocupan *n* y sus descendientes.

4.1.4.13. template<class Tbase > bool ArbolGeneral< Tbase >::empty ( ) const

Vacío.

Devuelve

Devuelve *true* si el número de elementos del árbol receptor es cero, *false* en caso contrario.

La operación se realiza en tiempo O(1).

4.1.4.14. template<class Tbase > ArbolGeneral< Tbase >::inorden_iterador ArbolGeneral< Tbase >::endinorden ( ) const

Final de un iterador inorden_iterador.

Devuelve

Un iterador de tipo inorden apuntando al final.

4.1.4.15. template<class Tbase > ArbolGeneral< Tbase >::postorden_iterador ArbolGeneral< Tbase >::endpostorden ( ) const

Último elmento.

Devuelve

Devuelve un iterador al último elemento en postorden.

4.1.4.16. template<class Tbase > ArbolGeneral< Tbase >::preorden_iterador ArbolGeneral< Tbase >::endpreorden ( ) const

Final de un iterador preorden_iterador.

Devuelve

un iterador de tipo preorden_iterador apuntando al final.

4.1.4.17. template<class Tbase > ArbolGeneral< Tbase >::reverse_preorden_iterador ArbolGeneral< Tbase >::endreverse_preorden ( ) const

Final de un iterador reverse_preorden_iterador.

Devuelve

un iterador de tipo preorden_iterador apuntando al primer elemento.

4.1.4.18. template<class Tbase> void ArbolGeneral< Tbase >::escribe_arbol (std::ostream & *out,* nodo * *nod*) const [private]

Escribe un subárbol.

Parámetros

out	Stream de salida donde escribir
nod	Nodo del que cuelga el subárbol a escribir

Escribe en el flujo de salida todos los nodos del subárbol que cuelga del nodo *nod* siguiendo un recorrido en preorden. La forma de impresión de cada nodo es:

- Si el nodo es nulo, imprime el carácter 'x'.

- Si el nodo no es nulo, imprime el carácter 'n' seguido de un espacio, al que sigue la impresión de la etiqueta

4.1.4.19. template<class Tbase > Tbase & ArbolGeneral< Tbase >::etiqueta (const Nodo *n*)

Etiqueta de un nodo.

Parámetros

n	Nodo en el que se encuentra el elemento.

Precondición

n no es nulo

Devuelve

Referencia al elemento del nodo *n*

Devuelve una referencia al elemento del nodo *n* y por tanto se puede modificiar o usar el valor. La operación se realiza en tiempo O(1).

4.1.4.20. template<class Tbase > const Tbase & ArbolGeneral< Tbase >::etiqueta (const Nodo *n*) const

Etiqueta de un nodo.

Parámetros

n	Nodo en el que se encuentra el elemento.

Precondición

n no es nulo

Devuelve

Referencia constante al elemento del nodo *n*.

Devuelve una referencia al elemento del nodo *n*. Es constante y por tanto no se puede modificiar el valor. La operación se realiza en tiempo O(1).

4.1.4.21. template<class Tbase > ArbolGeneral< Tbase >::Nodo ArbolGeneral< Tbase >::hermanoderecha (const Nodo *n*) const

Hermano derecha.

Parámetros

n	Nodo del que se quiere obtener el hermano a la derecha.

Precondición

n no es nulo

Devuelve

Nodo hermano a la derecha

Devuelve el nodo hermano a la derecha de *n*, que valdrá 0 (nulo) si no tiene hermano a la derecha. La operación se realiza en tiempo O(1).

4.1.4.22. template<class Tbase > ArbolGeneral< Tbase >::Nodo ArbolGeneral< Tbase >::hijomasizquierda (const Nodo *n*) const

Hijo más a la izquierda.

Parámetros

n	Nodo del que se quiere obtener el hijo más a la izquierda.

Precondición

n no es nulo

Devuelve

Nodo hijo más a la izquierda

Devuelve el nodo hijo más a la izquierda de *n*, que valdrá 0 (nulo) si no tiene hijo más a la izquierda. La operación se realiza en tiempo O(1).

4.1.4.23. template<class Tbase > void ArbolGeneral< Tbase >::insertar_hermanoderecha (Nodo *n*, ArbolGeneral< Tbase > & *rama*)

Insertar subárbol hermano derecha.

Parámetros

n	Nodo al que se insertará el árbol *rama* como hermano a la derecha.
rama	Árbol que se insertará como hermano derecho.

Precondición

n no es nulo y es un nodo válido del árbol receptor

El árbol *rama* se inserta como hermano derecho del nodo *n* del árbol receptor. El árbol *rama* queda vacío y los nodos que estaban a la derecha del nodo *n* pasan a la derecha del nuevo nodo.

4.1.4.24. template<class Tbase > void ArbolGeneral< Tbase >::insertar_hijomasizquierda (Nodo *n*, ArbolGeneral< Tbase > & *rama*)

Insertar subárbol hijo más a la izquierda.

Parámetros

n	Nodo al que se insertará el árbol *rama* como hijo más a la izquierda.
rama	Árbol que se insertará como hijo más a la izquierda.

Precondición

n no es nulo y es un nodo válido del árbol receptor

El árbol *rama* se inserta como hijo más a la izquierda del nodo *n* del árbol receptor. El árbol *rama* queda vacío y los nodos que estaban en el subárbol hijo más a la izquierda de *n* se desplazan a la derecha, de forma que el anterior hijo más a la izquierda pasa a ser el hermano a la derecha del nuevo hijo más a la izquierda.

4.1.4.25. template<class Tbase> void ArbolGeneral< Tbase >::lee_arbol (std::istream & *in*, nodo *& *nod*) [private]

Lee un subárbol.

Parámetros

in	Stream de entrada desde el que leer
nod	Referencia al nodo que contendrá el subárbol leído

Lee del flujo de entrada *in* los elementos de un árbol según el formato que se presenta en la función de escritura.

Ver también

escribe_arbol

4.1.4.26. template<class Tbase > bool ArbolGeneral< Tbase >::operator!= (const ArbolGeneral< Tbase > & *v*) const

Operador de comparación (diferencia)

Parámetros

v	ArbolGeneral con el que se desea comparar.

Devuelve

Devuelve *true* si el árbol receptor no tiene los mismos elementos y en el mismo orden, *false* en caso contrario.

La operación se realiza en tiempo O(n).

4.1.4.27. template<class Tbase > ArbolGeneral< Tbase > & ArbolGeneral< Tbase >::operator= (const ArbolGeneral< Tbase > & *v*)

Operador de asignación.

Parámetros

v	ArbolGeneral a copiar

Devuelve

Referencia al árbol receptor.

Asigna el valor del árbol duplicando el contenido de *v* en el árbol receptor. La operación se realiza en tiempo O(n), donde *n* es el número de elementos de *v*.

4.1.4.28. template<class Tbase > bool ArbolGeneral< Tbase >::operator== (const ArbolGeneral< Tbase > & *v*) const

Operador de comparación (igualdad)

Parámetros

v	ArbolGeneral con el que se desea comparar.

Devuelve

Devuelve *true* si el árbol receptor tiene los mismos elementos y en el mismo orden, *false* en caso contrario.

La operación se realiza en tiempo O(n).

Ver también

soniguales

4.1.4.29. template<class Tbase > ArbolGeneral< Tbase >::Nodo ArbolGeneral< Tbase >::padre (const Nodo *n*) const

Nodo padre.

Parámetros

n	Nodo del que se quiere obtener el padre.

Precondición

n no es nulo

Devuelve

Nodo padre

Devuelve el nodo padre de *n*, que valdrá 0 (nulo) si es la raíz. La operación se realiza en tiempo O(1).

4.1.4.30. template<class Tbase > void ArbolGeneral< Tbase >::podar_hermanoderecha (Nodo *n,* ArbolGeneral< Tbase > & *dest*)

Podar subárbol hermano derecha.

Parámetros

n	Nodo al que se le podará la rama hermano derecha.
dest	Árbol que recibe la rama cortada

Precondición

n no es nulo y es un nodo válido del árbol receptor.

Asigna un nuevo valor al árbol *dest*, con todos los elementos del subárbol hermano derecho del nodo *n* en el árbol receptor. Éste se queda sin dichos nodos. La operación se realiza en tiempo O(1).

4.1.4.31. template<class Tbase > void ArbolGeneral< Tbase >::podar_hijomasizquierda (Nodo *n,* ArbolGeneral< Tbase > & *dest*)

Podar subárbol hijo más a la izquierda.

Parámetros

n	Nodo al que se le podará la rama hijo más a la izquierda.
dest	Árbol que recibe la rama cortada

Precondición

n no es nulo y es un nodo válido del árbol receptor.

Asigna un nuevo valor al árbol *dest*, con todos los elementos del subárbol izquierdo del nodo *n* en el árbol receptor. Éste se queda sin dichos nodos. La operación se realiza en tiempo O(1).

4.1.4.32. template<class Tbase > ArbolGeneral< Tbase >::Nodo ArbolGeneral< Tbase >::raiz ( ) const

Raíz del árbol.

Devuelve

Nodo raíz del árbol receptor

Devuelve el nodo raíz, que es 0 (nulo) si el árbol está vacío. La operación se realiza en tiempo O(1).

4.1.4.33. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_inorden ( ) const

Recorrido en inorden. Muestra el árbol recorriendolo en inorden usando iteradores.

4.1.4.34. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_inorden2 (Nodo *t*) const

Recorrido en inorden.

Parámetros

t	Nodo a partir del cual se hará el recorrido. Muestra el árbol recorriendolo en inorden sin usar iteradores.

4.1.4.35. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_por_niveles (Nodo *t*) const

Recorrido por niveles.

Parámetros

t	Nodo a partir del cual se va a hacer el recorrido. Muestra el recorrido por niveles del Árbol.

4.1.4.36. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_postorden ( ) const

Recorrido en postorden. Recorre el árbol en postorden. Desde la raíz.

4.1.4.37. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_postorden2 (Nodo *t*) const

Recorrido postorden.

Parámetros

t	Nodo a partir del cual se va a hacer el recorrido. Muestra el recorrido en postorden sin iteradores.

4.1.4.38. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_preorden ( ) const

Recorrido en preorden. Muestra el recorrido del árbol en preorden usando iteradores.

4.1.4.39. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_preorden2 (Nodo *t*) const

Recorrido en preorden.

Parámetros

t	Nodo a partir del cual se va a hacer el recorrido. Muestra el el recorrido del árbol en preorden sin usar iteradores. De forma iterativa.

4.1.4.40. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_preorden_al_reves ( ) const

Recorrido en preorden al reves. Muestra el recorrido del árbol en preorden, pero al reves. Utilizando iteradores preorden.

4.1.4.41. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_reverse_preorden ( ) const

Recorrido en preorden invertido. Muestra el recorrido del árbol en preorden invertido usando iteradores.

4.1.4.42. template<class Tbase > void ArbolGeneral< Tbase >::recorrer_reverse_preorden_al_reves ( ) const

Recorrido en preorden invertido, al reves. Recorrido en preorden normal.

4.1.4.43. template<class Tbase > void ArbolGeneral< Tbase >::recuperar_arbol (string *preorden,* string *inorden,* string *postorden,* Nodo *nuevo*)

Crea un arbol a partir de sus tres recorridos.

Parámetros

preorden	Recorrido en preorden.
inorden	Recorrido en inorden.
postorden	Recorrido en postorden.
nuevo	Árbol que vamos a crear.

Devuelve

Devuelve un arbol cuyos recorridos son esos tres.

Dados tres recorridos exite tan solo 1 árbol que concuerda con esos tres recorridos.

4.1.4.44. template<class Tbase > void ArbolGeneral< Tbase >::reflejado (Nodo *t*)

Arbol reflejado.

Parámetros

t	Nodo a partir del cual se hará su reflejado.

4.1.4.45. template<class Tbase > int ArbolGeneral< Tbase >::size ( ) const

Número de elementos.

Devuelve

El número de elementos del árbol receptor.

La operación se realiza en tiempo O(n).

Ver también

contar

4.1.4.46. template<class Tbase > bool ArbolGeneral< Tbase >::soniguales (const nodo * *n1,* const nodo * *n2*) const [private]

Comprueba igualdad de dos subárboles.

Parámetros

n1	Primer subárbol a comparar
n2	Segundo subárbol a comparar

Comprueba si son iguales los subárboles que cuelgan de *n1* y *n2*. Para ello deberán tener los mismos nodos en las mismas posiciones y con las mismas etiquetas.

4.1.5. Documentación de las funciones relacionadas y clases amigas

4.1.5.1. template<class Tbase> template<class T > ostream& operator<< (ostream & *out,* const ArbolGeneral< T > & *v*) [friend]

Operador de inserción en flujo.

Parámetros

out	Stream de salida
v	Árbol que escribir

Devuelve

Referencia al stream de salida

Escribe en la salida todos los nodos del árbol *v* siguiendo un recorrido en preorden. La forma de impresión de cada nodo es:

- Si el nodo es nulo, imprime el carácter 'x'.
- Si el nodo no es nulo, imprime el carácter 'n' seguido de un espacio, al que sigue la impresión de la etiqueta.

Ver también

escribe_arbol

4.1.5.2. template<class Tbase> template<class T > std::istream& operator>> (std::istream & *in,* ArbolGeneral< T > & *v*) [friend]

Operador de extracción de flujo.

Parámetros

in	Stream de entrada
v	Árbol que leer

Devuelve

Referencia al stream de entrada

Lee de *in* un árbol y lo almacena en *v*. El formato aceptado para la lectura se puede consultar en la función de salida.

Ver también

lee_arbol

4.1.6. Documentación de los datos miembro

4.1.6.1. template<class Tbase> struct nodo* ArbolGeneral< Tbase >::laraiz [private]

Puntero a la raíz.

Este miembro es un puntero al primer nodo, que corresponde a la raíz del árbol. Vale 0 sin el árbol es vacío.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.2. Referencia de la Clase ArbolGeneral< Tbase >::inorden_iterador

`#include <ArbolGeneral.h>`

Métodos públicos

- inorden_iterador ()
  Constructor por defecto.
- inorden_iterador (const Nodo &n)
  Constructor con nodo.
- inorden_iterador (const inorden_iterador &n)
  Constructor de copia.
- const Tbase & operator* () const
  Acceso a la información del nodo.
- Tbase & operator* ()
- bool operator== (const inorden_iterador &n)
  Operación de igualdad entre dos posiciones.
- bool operator!= (const inorden_iterador &n)
  Operación de desigualdad entre dos posiciones.
- inorden_iterador padre ()
  Nodo del padre.
- inorden_iterador izquierda ()
  Nodo hijo.
- inorden_iterador derecha ()
  Nodo hermano.
- bool nulo ()
  Dice si un nodo es nulo.
- inorden_iterador & operator++ ()
  Siguiente elemento.

Atributos privados

- Nodo p

Amigas

- class ArbolGeneral

4.2.1. Documentación del constructor y destructor

4.2.1.1. template<class Tbase> ArbolGeneral< Tbase >::inorden_iterador::inorden_iterador ( ) [inline]

Constructor por defecto.

4.2.1.2. template<class Tbase> ArbolGeneral< Tbase >::inorden_iterador::inorden_iterador (const Nodo & *n*) [inline]

Constructor con nodo.

Parámetros

n	nodo fuente.

4.2.1.3. template<class Tbase> ArbolGeneral< Tbase >::inorden_iterador::inorden_iterador (const inorden_iterador & *n*) [inline]

Constructor de copia.

Parámetros

n	nodo fuente.

4.2.2. Documentación de las funciones miembro

4.2.2.1. template<class Tbase> inorden_iterador ArbolGeneral< Tbase >::inorden_iterador::derecha ( ) [inline]

Nodo hermano.

Devuelve

Devuelve un nodo apuntando al hermano derecha.

4.2.2.2. template<class Tbase> inorden_iterador ArbolGeneral< Tbase >::inorden_iterador::izquierda ( ) [inline]

Nodo hijo.

Devuelve

devuelve un nodo apuntando al hijo más a la izquierda.

4.2.2.3. template<class Tbase> bool ArbolGeneral< Tbase >::inorden_iterador::nulo ( ) [inline]

Dice si un nodo es nulo.

Devuelve

true si es nulo, false en caso contrario.

4.2.2.4. template<class Tbase> bool ArbolGeneral< Tbase >::inorden_iterador::operator!= (const inorden_iterador & *n*) [inline]

Operación de desigualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son desiguales false en caso contrario.

4.2.2.5. template<class Tbase> const Tbase& ArbolGeneral< Tbase >::inorden_iterador::operator* ( ) const [inline]

Acceso a la información del nodo.

4.2.2.6. template<class Tbase> Tbase& ArbolGeneral< Tbase >::inorden_iterador::operator* ( ) [inline]

4.2.2.7. template<class Tbase > ArbolGeneral< Tbase >::inorden_iterador & ArbolGeneral< Tbase >::inorden_iterador::operator++ ( )

Siguiente elemento.

Devuelve

Devuelve un iterador en inorden al siguiente elemento.

4.2.2.8. template<class Tbase> bool ArbolGeneral< Tbase >::inorden_iterador::operator== (const inorden_iterador & *n*) [inline]

Operación de igualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son iguales false en caso opuesto.

4.2.2.9. template<class Tbase> inorden_iterador ArbolGeneral< Tbase >::inorden_iterador::padre ( ) [inline]

Nodo del padre.

Devuelve

devuelve un nodo apuntando al padre.

4.2.3. Documentación de las funciones relacionadas y clases amigas

4.2.3.1. template<class Tbase> friend class ArbolGeneral [friend]

4.2.4. Documentación de los datos miembro

4.2.4.1. template<class Tbase> Nodo ArbolGeneral< Tbase >::inorden_iterador::p [private]

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.3. Referencia de la Clase Mando

TDA Mando.

```
#include <mando.h>
```

Métodos públicos

- Mando (Tablero t)

  Constructor. Inicializa las variables dado un tablero. No se puede generar una Mando sin un tablero. Para que no cause problemas de incongruencias de tamaño.
- bool actualizarJuego (char c, Tablero &t)

  Actualiza el tablero del juego: la posición del jugador y si se ha introducido alguna pieza en el tablero. Esta función es la encargada de controlar la entrada del teclado.
- string GetBase () const

  Representación gráfica de la base.
- string GetJugador () const

  Representación gráfica del jugador.
- string GetMando () const

  Representación gráfica del mando.

Atributos públicos estáticos

- static const char KB_LEFT = 'a'
- static const char KB_RIGHT = 'd'
- static const char KB_SPACE = ' '
- static const char KB_ESCAPE = 27

Atributos privados

- string mando

  Señala dónde está colocado el jugador. Representada con '^'.
- string jugador

  Indica la ficha del jugador.
- string base

  Base del tablero.
- int posicion

  Indica la posición actual en la que está el jugador.

4.3.1. Descripción detallada

TDA Mando.

Este TDA sirve para representar la parte gráfica del juego.

4.3.2. Documentación del constructor y destructor

4.3.2.1. Mando::Mando (Tablero *t*)

Constructor. Inicializa las variables dado un tablero. No se puede generar una Mando sin un tablero. Para que no cause problemas de incongruencias de tamaño.

Parámetros

t	: Tablero sobre el que se va a inicialiar la parte gráfica. El mando, el indicador de la ficha del jugador y la base del tablero.

4.3.3. Documentación de las funciones miembro

4.3.3.1. bool Mando::actualizarJuego (char *c*, Tablero & *t*)

Actualiza el tablero del juego: la posición del jugador y si se ha introducido alguna pieza en el tablero. Esta función es la encargada de controlar la entrada del teclado.

Parámetros

c	: Caracter leído por el teclado.
t	: Tablero. Si el caracter leido es el de colocar ficha y hay hueco modifica el tablero.

Devuelve

Devuelve true si se ha colocado una ficha en la actualización del juego.

4.3.3.2. string Mando::GetBase ( ) const [inline]

Representación gráfica de la base.

Devuelve

Devuelve una cadena representado la base del tablero.

4.3.3.3. string Mando::GetJugador ( ) const [inline]

Representación gráfica del jugador.

Devuelve

Devuelve una cadena representado del jugador.

4.3.3.4. string Mando::GetMando ( ) const [inline]

Representación gráfica del mando.

Devuelve

Devuelve la representación gráfica del mando.

4.3.4. Documentación de los datos miembro

4.3.4.1. string Mando::base [private]

Base del tablero.

4.3.4.2. string Mando::jugador [private]

Indica la ficha del jugador.

4.3.4.3. const char Mando::KB_ESCAPE = 27 [static]

4.3.4.4. const char Mando::KB_LEFT = 'a' [static]

4.3.4.5. const char Mando::KB_RIGHT = 'd' [static]

4.3.4.6. const char Mando::KB_SPACE = ' ' [static]

4.3.4.7. string Mando::mando [private]

Señala dónde está colocado el jugador. Representada con '^'.

4.3.4.8. int Mando::posicion [private]

Indica la posición actual en la que está el jugador.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/mando.h

4.4. Referencia de la Estructura ArbolGeneral< Tbase >::nodo

repConjunto Rep del TDA ArbolGeneral

Métodos públicos

- nodo ()
  Constructor. Crea un nodo vacio.
- nodo (const Tbase &elemento)
  Constructor con parametros.

Atributos públicos

- Tbase etiqueta
  Elemento almacenado.
- nodo * izqda
  Puntero al hijo más a la izquierda.
- nodo * drcha
  Puntero al hermano derecho.
- nodo * padre
  Puntero al padre.

4.4.1. Descripción detallada

template<class Tbase>
struct ArbolGeneral< Tbase >::nodo

repConjunto Rep del TDA ArbolGeneral

|b Invariante de la representación

Sea *T* un Árbol General sobre el tipo *Tbase*. Entonces el invariante de la representación es

Si *T* es vacío, entonces T.laraiz vale 0. Si no:

- T.laraiz->padre = 0 y
- $\forall$ n nodo de *T*, n->izqda $\neq$ n->drch y
- $\forall$ n, m nodos de *T*, si n->izqda = m, entonces m->padre = n y
- Número de elementos = número elementos de la raiz, donde N(n) = 1 + N(n->izqda) + (N->drcha), con N(0) = 0.

Función de abstracción

Sea *T* un Árbol General sobre el tipo *Tbase*. Entonces, si lo denotamos también Árbol(T.laraiz), es decir, como el árbol que cuelga de su raíz, entonces este árbol del conjunto de valores en la representación se aplica al árbol.

{ T.laraiz->etiqueta, {Arbol(T.laraiz->izqda)}, {Arbol(T.laraiz->drcha)} }

donde {0} es el árbol vacío. nodo

En cada estructura *nodo* se almacena una etiqueta del árbol, que se implementa como un conjunto de nodos enlazados según la relación padre-hijo más a la izquierda-hermano derecha.

4.4.2. Documentación del constructor y destructor

4.4.2.1. template<class Tbase> ArbolGeneral< Tbase >::nodo::nodo ( ) [inline]

Constructor. Crea un nodo vacio.

4.4.2.2. template<class Tbase> ArbolGeneral< Tbase >::nodo::nodo (const Tbase & *elemento*) [inline]

Constructor con parametros.

Parámetros

e	elemento que se le va a asignar. Crea un nodo a con un elemento.

4.4.3. Documentación de los datos miembro

4.4.3.1. template<class Tbase> nodo* ArbolGeneral< Tbase >::nodo::drcha

Puntero al hermano derecho.

En este campo se almacena un puntero al nodo raíz del subárbol hermano derecho, o el valor 0 si no tiene.

4.4.3.2. template<class Tbase> Tbase ArbolGeneral< Tbase >::nodo::etiqueta

Elemento almacenado.

En este campo se almacena la etiqueta que corresponde a este nodo.

4.4.3.3. template<class Tbase> nodo* ArbolGeneral< Tbase >::nodo::izqda

Puntero al hijo más a la izquierda.

En este campo se almacena un puntero al nodo raíz del subárbol más a la izquierda, o el valor 0 si no tiene.

4.4.3.4. template<class Tbase> nodo* ArbolGeneral< Tbase >::nodo::padre

Puntero al padre.

En este campo se almacena un puntero al nodo padre, o el valor 0 si es la raíz.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.5. Referencia de la Clase ArbolGeneral< Tbase >::postorden_iterador

```
#include <ArbolGeneral.h>
```

Métodos privados

- postorden_iterador ()

  Constructor por defecto.
- postorden_iterador (const Nodo &n)

  Constructor con nodo.
- postorden_iterador (const postorden_iterador &n)

  Constructor de copia.
- const Tbase & operator∗ () const

  Acceso a la información del nodo.
- Tbase & operator∗ ()
- bool operator== (const postorden_iterador &n)

  Operación de igualdad entre dos posiciones.
- bool operator!= (const postorden_iterador &n)

  Operación de desigualdad entre dos posiciones.
- postorden_iterador padre ()

  Nodo del padre.
- postorden_iterador izquierda ()

  Nodo hijo.
- postorden_iterador derecha ()

  Nodo hermano.
- bool nulo ()

  Dice si un nodo es nulo.
- postorden_iterador & operator++ ()

  Siguiente elemento en postorden.

Atributos privados

- Nodo p

Amigas

- class ArbolGeneral

4.5.1. Documentación del constructor y destructor

4.5.1.1. template<class Tbase> ArbolGeneral< Tbase >::postorden_iterador::postorden_iterador ( ) [inline], [private]

Constructor por defecto.

4.5.1.2. template<class Tbase> ArbolGeneral< Tbase >::postorden_iterador::postorden_iterador (const Nodo & *n*) [inline], [private]

Constructor con nodo.

Parámetros

n	nodo fuente.

4.5.1.3. template<class Tbase> ArbolGeneral< Tbase >::postorden_iterador::postorden_iterador (const postorden_iterador & *n*) [inline], [private]

Constructor de copia.

Parámetros

n	nodo fuente.

4.5.2. Documentación de las funciones miembro

4.5.2.1. template<class Tbase> postorden_iterador ArbolGeneral< Tbase >::postorden_iterador::derecha ( ) [inline], [private]

Nodo hermano.

Devuelve

Devuelve un nodo apuntando al hermano derecha.

4.5.2.2. template<class Tbase> postorden_iterador ArbolGeneral< Tbase >::postorden_iterador::izquierda ( ) [inline], [private]

Nodo hijo.

Devuelve

devuelve un nodo apuntando al hijo más a la izquierda.

4.5.2.3. template<class Tbase> bool ArbolGeneral< Tbase >::postorden_iterador::nulo ( ) [inline], [private]

Dice si un nodo es nulo.

Devuelve

true si es nulo, false en caso contrario.

4.5.2.4. template<class Tbase> bool ArbolGeneral< Tbase >::postorden_iterador::operator!= (const postorden_iterador & *n*) [inline], [private]

Operación de desigualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son desiguales false en caso contrario.

4.5.2.5. template<class Tbase> const Tbase& ArbolGeneral< Tbase >::postorden_iterador::operator*( ) const [inline], [private]

Acceso a la información del nodo.

4.5.2.6. template<class Tbase> Tbase& ArbolGeneral< Tbase >::postorden_iterador::operator*( ) [inline], [private]

4.5.2.7. template<class Tbase > ArbolGeneral< Tbase >::postorden_iterador & ArbolGeneral< Tbase >::postorden_iterador::operator++( ) [private]

Siguiente elemento en postorden.

Devuelve

Devuelve un iterador al siguiente elemento en postorden.

4.5.2.8. template<class Tbase> bool ArbolGeneral< Tbase >::postorden_iterador::operator==(const postorden_iterador & *n*) [inline], [private]

Operación de igualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son iguales false en caso opuesto.

4.5.2.9. template<class Tbase> postorden_iterador ArbolGeneral< Tbase >::postorden_iterador::padre( ) [inline], [private]

Nodo del padre.

Devuelve

devuelve un nodo apuntando al padre.

4.5.3. Documentación de las funciones relacionadas y clases amigas

4.5.3.1. template<class Tbase> friend class ArbolGeneral [friend]

4.5.4. Documentación de los datos miembro

4.5.4.1. template<class Tbase> Nodo ArbolGeneral< Tbase >::postorden_iterador::p [private]

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.6. Referencia de la Clase ArbolGeneral< Tbase >::preorden_iterador

`#include <ArbolGeneral.h>`

Métodos públicos

- preorden_iterador ()
  Constructor por defecto.
- preorden_iterador (const Nodo &n)
  Constructor con nodo.
- preorden_iterador (const preorden_iterador &i)
  Constructor de copia.
- const Tbase & operator* () const
  Acceso a la información del nodo.
- Tbase & operator* ()
- bool operator== (const preorden_iterador &n)
  Operación de igualdad entre dos posiciones.
- bool operator!= (const preorden_iterador &n)
  Operación de desigualdad entre dos posiciones.
- preorden_iterador padre ()
  Nodo del padre.
- preorden_iterador izquierda ()
  Nodo del hijo.
- preorden_iterador hermano ()
  Nodo del hermano.
- bool nulo ()
  Informa de si es nulo.
- preorden_iterador & operator++ ()
  Siguiente elemento.
- preorden_iterador & operator-- ()
  Anterior elemento.

Atributos privados

- Nodo p

Amigas

- class ArbolGeneral
- class reverse_preorden_iterador

4.6.1. Documentación del constructor y destructor

4.6.1.1. template<class Tbase> ArbolGeneral< Tbase >::preorden_iterador::preorden_iterador ( ) [inline]

Constructor por defecto.

4.6.1.2. template<class Tbase> ArbolGeneral< Tbase >::preorden_iterador::preorden_iterador (const Nodo & *n*) [inline]

Constructor con nodo.

Parámetros

n	nodo que se va a copiar.

4.6.1.3. template<class Tbase> ArbolGeneral< Tbase >::preorden_iterador::preorden_iterador (const preorden_iterador & *i*) [inline]

Constructor de copia.

Parámetros

i	iterador que se va a copiar.

4.6.2. Documentación de las funciones miembro

4.6.2.1. template<class Tbase> preorden_iterador ArbolGeneral< Tbase >::preorden_iterador::hermano ( ) [inline]

Nodo del hermano.

Devuelve

devuelve un nodo apuntando al hermano de la derecha.

4.6.2.2. template<class Tbase> preorden_iterador ArbolGeneral< Tbase >::preorden_iterador::izquierda ( ) [inline]

Nodo del hijo.

Devuelve

devuelve un nodo apuntando al hijo más a la izquierda.

4.6.2.3. template<class Tbase> bool ArbolGeneral< Tbase >::preorden_iterador::nulo ( ) [inline]

Informa de si es nulo.

Devuelve

true si es nulo, false en otro caso.

4.6.2.4. template<class Tbase> bool ArbolGeneral< Tbase >::preorden_iterador::operator!= (const preorden_iterador & *n*) [inline]

Operación de desigualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son distintos false en caso contrario.

4.6.2.5. template<class Tbase> const Tbase& ArbolGeneral< Tbase >::preorden_iterador::operator*( ) const [inline]

Acceso a la información del nodo.

4.6.2.6. template<class Tbase> Tbase& ArbolGeneral< Tbase >::preorden_iterador::operator*( ) [inline]

4.6.2.7. template<class Tbase > ArbolGeneral< Tbase >::preorden_iterador & ArbolGeneral< Tbase >::preorden_iterador::operator++( )

Siguiente elemento.

Devuelve

Devuelve un iterador al siguiente elemento en preorden.

4.6.2.8. template<class Tbase > ArbolGeneral< Tbase >::preorden_iterador & ArbolGeneral< Tbase >::preorden_iterador::operator--( )

Anterior elemento.

Devuelve

Devuelv un iterador al elemento anterior en preorden.

4.6.2.9. template<class Tbase> bool ArbolGeneral< Tbase >::preorden_iterador::operator==(const preorden_iterador & *n*) [inline]

Operación de igualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son iguales false en caso contrario.

4.6.2.10. template<class Tbase> preorden_iterador ArbolGeneral< Tbase >::preorden_iterador::padre( ) [inline]

Nodo del padre.

Devuelve

Devuelve un nodo apuntando al padre.

4.6.3. Documentación de las funciones relacionadas y clases amigas

4.6.3.1. template<class Tbase> friend class ArbolGeneral [friend]

4.6.3.2. template<class Tbase> friend class reverse_preorden_iterador [friend]

4.6.4. Documentación de los datos miembro

4.6.4.1. template<class Tbase> Nodo ArbolGeneral< Tbase >::preorden_iterador::p [private]

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.7. Referencia de la Clase ArbolGeneral< Tbase >::reverse_preorden_iterador

`#include <ArbolGeneral.h>`

Métodos públicos

- reverse_preorden_iterador ()
  Constructor por defecto.
- reverse_preorden_iterador (const Nodo &n)
  Constructor con nodo.
- reverse_preorden_iterador (const reverse_preorden_iterador &i)
  Constructor de copia.
- const Tbase & operator* () const
  Acceso a la información del nodo.
- Tbase & operator* ()
- bool operator== (const reverse_preorden_iterador &n)
  Operación de igualdad entre dos posiciones.
- bool operator!= (const reverse_preorden_iterador &n)
  Operación de desigualdad entre dos posiciones.
- reverse_preorden_iterador padre ()
  Nodo del padre.
- reverse_preorden_iterador izquierda ()
  Nodo del hijo.
- reverse_preorden_iterador hermano ()
  Nodo del hermano.
- bool nulo ()
  Informa de si es nulo.
- reverse_preorden_iterador & operator++ ()
  Siguiente elemento.
- reverse_preorden_iterador & operator-- ()
  Anterior elemento.

Atributos privados

- Nodo p

Amigas

- class ArbolGeneral
- class preorden_iterador

4.7.1. Documentación del constructor y destructor

4.7.1.1. template<class Tbase> ArbolGeneral< Tbase >::reverse_preorden_iterador::reverse_preorden_iterador ( ) [inline]

Constructor por defecto.

4.7.1.2. template<class Tbase> ArbolGeneral< Tbase >::reverse_preorden_iterador::reverse_preorden_iterador (const Nodo & *n*) [inline]

Constructor con nodo.

Parámetros

n	nodo que se va a copiar.

4.7.1.3. template<class Tbase> ArbolGeneral< Tbase >::reverse_preorden_iterador::reverse_preorden_iterador (const reverse_preorden_iterador & *i*) [inline]

Constructor de copia.

Parámetros

i	iterador que se va a copiar.

4.7.2. Documentación de las funciones miembro

4.7.2.1. template<class Tbase> reverse_preorden_iterador ArbolGeneral< Tbase >::reverse_preorden_iterador::hermano ( ) [inline]

Nodo del hermano.

Devuelve

devuelve un nodo apuntando al hermano de la derecha.

4.7.2.2. template<class Tbase> reverse_preorden_iterador ArbolGeneral< Tbase >::reverse_preorden_iterador::izquierda ( ) [inline]

Nodo del hijo.

Devuelve

devuelve un nodo apuntando al hijo más a la izquierda.

4.7.2.3. template<class Tbase> bool ArbolGeneral< Tbase >::reverse_preorden_iterador::nulo ( ) [inline]

Informa de si es nulo.

Devuelve

true si es nulo, false en otro caso.

4.7.2.4. template<class Tbase> bool ArbolGeneral< Tbase >::reverse_preorden_iterador::operator!= (const reverse_preorden_iterador & *n*) [inline]

Operación de desigualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son distintos false en caso contrario.

4.7.2.5. template<class Tbase> const Tbase& ArbolGeneral< Tbase >::reverse_preorden_iterador::operator* ( ) const [inline]

Acceso a la información del nodo.

4.7.2.6. template<class Tbase> Tbase& ArbolGeneral< Tbase >::reverse_preorden_iterador::operator* ( ) [inline]

4.7.2.7. template<class Tbase > ArbolGeneral< Tbase >::reverse_preorden_iterador & ArbolGeneral< Tbase >::reverse_preorden_iterador::operator++ ( )

Siguiente elemento.

Devuelve

Devuelve un iterador al siguiente elemento en preorden inverso.

4.7.2.8. template<class Tbase > ArbolGeneral< Tbase >::reverse_preorden_iterador & ArbolGeneral< Tbase >::reverse_preorden_iterador::operator-- ( )

Anterior elemento.

Devuelve

Devuelv un iterador al elemento anterior en preorden inverso.

4.7.2.9. template<class Tbase> bool ArbolGeneral< Tbase >::reverse_preorden_iterador::operator== (const reverse_preorden_iterador & *n*) [inline]

Operación de igualdad entre dos posiciones.

Parámetros

n	nodo con el que se compara.

Devuelve

true si son iguales false en caso contrario.

4.7.2.10. template<class Tbase> reverse_preorden_iterador ArbolGeneral< Tbase >::reverse_preorden_iterador::padre ( ) [inline]

Nodo del padre.

Devuelve

Devuelve un nodo apuntando al padre.

4.7.3. Documentación de las funciones relacionadas y clases amigas

4.7.3.1. template<class Tbase> friend class ArbolGeneral [friend]

4.7.3.2. template<class Tbase> friend class preorden_iterador [friend]

4.7.4. Documentación de los datos miembro

4.7.4.1. template<class Tbase> Nodo ArbolGeneral< Tbase >::reverse_preorden_iterador::p [private]

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/ArbolGeneral.h

4.8. Referencia de la Clase Tablero

T.D.A. Tablero.

```
#include <tablero.h>
```

Métodos públicos

- Tablero ()

  Constructor por defecto. Crea un tablero de tamaño predefinido.
- Tablero (const int filas, const int columnas)

  Constructor. Crea un tablero introduciendo el tamaño del mismo. El estado inicial del tablero es todo 0, es decir, todo el tablero está libre. El turno inicial es el del jugador 1.
- Tablero (const Tablero &t)

  Constructor de copia. Crea un tablero a partir de otro dado.
- ∼Tablero ()
- int hayHueco (int pos)

  Comprueba si hay hueco en una columna 'pos'.
- bool colocarFicha (int pos)

  Coloca una ficha en la columna especificada del jugador correspondiente. Si le toca al jugador 1, inserta la ficha en esa posición. (Si hay hueco)
- int cambiarTurno ()

  Cambia el turno del jugador que toca.
- const int GetColumnas () const

  Devuelve la anchura del tablero.
- const int GetFilas () const

  Devuelve la altura del tablero.
- vector< vector< int > > GetTablero () const

  Función que devuelve el atributo tablero.
- void SetTablero (vector< vector< int > > tablero)

  Asigna un tablero introducido como parámetro.
- int GetTurno ()

  Turno del estado actual.
- Tablero & operator= (const Tablero &derecha)

  Operador de igualdad. Asigna los valores del tablero de la derecha al de la izquierda.
- int quienGana ()

  Función que calcula en un estado concreto del tablero, quién gana. En cualquier estado del tablero, se puede decidir por el estado del juego: gana el jugador 1, empate o gana el jugador 2.

Atributos públicos estáticos

- static const int N_FICHAS_GANAR = 4

Métodos privados

- void reserve ()

  Crea el tablero de tamaño filas/columnas.

Atributos privados

- vector< vector< int > > tablero

  Matriz que representa un estado del juego.
- const int filas

  Número de filas que tiene el tablero.
- const int columnas

  Número de columnas que tiene el tablero.
- int turno

  Indica a qué jugador le toca poner ficha. 1 para el jugador 1, 2 para el jugador 2.

Amigas

- ostream & operator<< (ostream &os, const Tablero &t)

  Operador flujo de salida. Imprime el tablero por el flujo de salida.

4.8.1. Descripción detallada

T.D.A. Tablero.

4.8.2. Documentación del constructor y destructor

4.8.2.1. Tablero::Tablero ( )

Constructor por defecto. Crea un tablero de tamaño predefinido.

4.8.2.2. Tablero::Tablero (const int *filas*, const int *columnas*)

Constructor. Crea un tablero introduciendo el tamaño del mismo. El estado inicial del tablero es todo 0, es decir, todo el tablero está libre. El turno inicial es el del jugador 1.

Parámetros

filas	: Número de filas que tendrá el tablero.
columnas	: Nümero de columnas del tablero.

4.8.2.3. Tablero::Tablero (const Tablero & *t*)

Constructor de copia. Crea un tablero a partir de otro dado.

Parámetros

t	: Tablero origen que se va a copiar.

4.8.2.4. Tablero::~Tablero ( )

4.8.3. Documentación de las funciones miembro

4.8.3.1. int Tablero::cambiarTurno ( )

Cambia el turno del jugador que toca.

Devuelve

Devuelve el turno que toca.

4.8.3.2. bool Tablero::colocarFicha (int *pos*)

Coloca una ficha en la columna especificada del jugador correspondiente. Si le toca al jugador 1, inserta la ficha en esa posición. (Si hay hueco)

Parámetros

pos	: Columna en la que se va a intentar colocar la ficha.

Devuelve

Devuelve true si se ha introducido la ficha en la posición. False en otro caso.

4.8.3.3. const int Tablero::GetColumnas ( ) const [inline]

Devuelve la anchura del tablero.

Devuelve

Número de columnas del tablero (anchura).

4.8.3.4. const int Tablero::GetFilas ( ) const [inline]

Devuelve la altura del tablero.

Devuelve

Número de filas del tablero (altura).

4.8.3.5. vector<vector<int> > Tablero::GetTablero ( ) const [inline]

Función que devuelve el atributo tablero.

Devuelve

Devuelve un vector de vectores de enteros (una matriz) de enteros representando un tablero.

4.8.3.6. int Tablero::GetTurno ( ) [inline]

Turno del estado actual.

Devuelve

Devuelve el turno del jugador. {1, 2}

4.8.3.7. int Tablero::hayHueco (int *pos*)

Comprueba si hay hueco en una columna 'pos'.

Parámetros

pos	: Columna sobre la que se va a comprobar si hay hueco para introducir una ficha.

Devuelve

Devuelve la fila en la que hay hueco (en esa columna). Si no hay hueco devuelve -1.

4.8.3.8. Tablero& Tablero::operator= (const Tablero & *derecha*)

Operador de igualdad. Asigna los valores del tablero de la derecha al de la izquierda.

Parámetros

derecha	: Tablero origen que se va a copiar.

Devuelve

Devuelve la referencia al tablero destino que se ha copiado.

4.8.3.9. int Tablero::quienGana ( )

Función que calcula en un estado concreto del tablero, quién gana. En cualquier estado del tablero, se puede decidir por el estado del juego: gana el jugador 1, empate o gana el jugador 2.

Devuelve

Devuelve {0, 1, 2} 0 si no ha ganado nadie. 1 si ha ganado el jugador 1 y 2 si ha ganado el jugador 2.

4.8.3.10. void Tablero::reserve ( ) [private]

Crea el tablero de tamaño filas/columnas.

4.8.3.11. void Tablero::SetTablero (vector< vector< int > > *tablero*)

Asigna un tablero introducido como parámetro.

Parámetros

tablero	: Matriz (vector de vectores de enteros) representante de un estado del juego.

4.8.4. Documentación de las funciones relacionadas y clases amigas

4.8.4.1. ostream& operator<< (ostream & *os*, const Tablero & *t*) [friend]

Operador flujo de salida. Imprime el tablero por el flujo de salida.

Parámetros

os	: Flujo de salida.
t	: Tablero que se va a imprimir.

4.8.5. Documentación de los datos miembro

4.8.5.1. const int Tablero::columnas [private]

Número de columnas que tiene el tablero.

4.8.5.2. const int Tablero::filas [private]

Número de filas que tiene el tablero.

4.8.5.3. const int Tablero::N_FICHAS_GANAR = 4 [static]

4.8.5.4. vector<vector<int> > Tablero::tablero [private]

Matriz que representa un estado del juego.

4.8.5.5. int Tablero::turno [private]

Indica a qué jugador le toca poner ficha. 1 para el jugador 1, 2 para el jugador 2.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/tablero.h

5. Documentación de archivos

5.1. Referencia del Archivo doc/doxys/guion.dox

5.2. Referencia del Archivo include/ArbolGeneral.h

```
#include <cassert>
#include <algorithm>
#include <queue>
#include <string>
#include <string.h>
```

Clases

- class ArbolGeneral< Tbase >
  T.D.A. ArbolGeneral.
- struct ArbolGeneral< Tbase >::nodo
  repConjunto Rep del TDA ArbolGeneral
- class ArbolGeneral< Tbase >::preorden_iterador
- class ArbolGeneral< Tbase >::reverse_preorden_iterador
- class ArbolGeneral< Tbase >::inorden_iterador
- class ArbolGeneral< Tbase >::postorden_iterador

Funciones

- template<class T >
  istream & operator>> (istream &in, ArbolGeneral< T > &v)
- template<class Tbase >
  ostream & operator<< (ostream &out, const ArbolGeneral< Tbase > &v)

5.2.1. Documentación de las funciones

5.2.1.1. template<class Tbase > ostream& operator<< (ostream & *out,* const ArbolGeneral< Tbase > & *v*)

5.2.1.2. template<class T > istream& operator>> (istream & *in,* ArbolGeneral< T > & *v*)

5.3. Referencia del Archivo include/mando.h

```
#include "tablero.h"
```

Clases

- class Mando
  TDA Mando.

5.4. Referencia del Archivo include/tablero.h

```
#include <vector>
#include <iostream>
```

Clases

- class Tablero
  T.D.A. Tablero.

Funciones

- template<class T >
  ostream & operator<< (ostream &s, const vector< T > &c)
  Imprime un conjunto de tipo T sobre el flujo de salida.
- template<class T >
  ostream & operator<< (ostream &s, const vector< vector< T > > &c)
  Imprime una matriz de tipo T sobre el flujo de salida.

5.4.1. Documentación de las funciones

5.4.1.1. template<class T > ostream& operator<< (ostream & *s,* const vector< T > & *c*)

Imprime un conjunto de tipo T sobre el flujo de salida.

Parámetros

s	Flujo de salida.
c	Conjunto con los elementos a imprimir.

Devuelve

Devuelve el flujo de salida.

5.4.1.2. template<class T > ostream& operator<< (ostream & *s*, const vector< vector< T > > & *c*)

Imprime una matriz de tipo T sobre el flujo de salida.

Parámetros

s	Flujo de salida.
c	Vector de vectores (matriz) de tipo T a imprimir.

Devuelve

Devuelve el flujo de salida.

Índice alfabético