GS1

Generado por Doxygen 1.7.5

Miércoles, 17 de Diciembre de 2014 11:18:07

Índice general

Capítulo 1

GS1 Código Electrónico de Producto

Versión

v0

Autor

Estructuras de Datos

1.1. Introducción

El objetivo de esta práctica es el de profundizar en el uso de conceptos asociados a la abstracción de datos en general, y los Tipos de Datos Abstractos. En esta practica se pretende avanzar en el uso de las estructuras de datos gerárquicas complejas como es el tipo tree (definido en tree.h)

1.2. Descripcion

Un objeto de la clase gs1Set representara un contenedor que permite almacenar un codigo electrónico de producto, en concreto consideraremos la normativa gs1-128

El gs1-128 es un sistema estándar de identificación mediante código de barras utilizado internacionalmente para la identificación de mercancías en entornos logísticos y no detallistas. Este sistema se utiliza principalmente para la identificación de unidades de expedición.

El código gs1-128 se representará mediante un string (que además suele venir representado como un código de barras) como indica la figura,

Figura 1.1: Ejemplo de codificación GS1

ejemplos de códigos válidos son:

- (01)09501101020917, representando el código de la agrupación.
- (01)09501101020917(17)131016(00)095011011234567889, representando el código del envio.

Una multinacional de la distribución debe ser capaz de almacenar información sobre todos los productos con los que está trabajando, por lo que necesita una estructura de datos rápida para poder incluir nuevos productos, identificar si un producto está dentro de su cadena, así como borrar los productos cuando estos han llegado a su destino. El número de productos puede ser muy elevado, pues para cada uno de ellos, tanto de forma aislada como cuando se empaquetan, debe tener asignado un código distinto.

1.3. Nuestros Códigos

En esta practica consideraremos códigos correctos, aquellos obtenidos según la siguiente definicion: Una código correcto sólo puede contener dígitos (del 0 al 9) junto con paréntesis de apertura y cierre. Por tanto, no debe tener espacios ni delimitadores del tipo comillas, llaves, comas, puntos, etc.

En el código podemos distinguir dos partes, encerrados entre paréntesis los identificadores de aplicación, IA, que son unos prefijos numéricos creados para dar significado inequívoco a los elementos de datos estandarizados que se encuentran situados a continuación (son subcódigos dentro el código gs1). Cada prefijo identifica el significado y el formato de los subcódigos que le siguen.

En la actualidad, existen más de 100 identificadores de aplicación estandarizados internacionalmente. Por ejemplo,

- 00 Código Seriado de la Unidad de Envío (SSCC)
- 01 Código de agrupación
- 02 Código del artículo / agrupación contenido
- 37 Cantidades (acompañando al IA 02)
- 10 Número de lote
- 11 Fecha de fabricación
- 13 Fecha de envasado
- 15 Fecha de consumo preferente
- 17 Fecha de caducidad etc.

En la práctica NO nos preocuparemos de generar códigos correctos, sino que consideraremos como correcto cualquier codigo que tenga el formato (yy)xxxxz(yyy)xxxxxxxz(yy)xxxxxz, donde yyy e xxxxxz son digitos del 0 al 9 representando el IA y el código asociado. (El alumno interesado puede consultar cómo se construyen los códigos en el enlace http://www.aecoc.es/BAJAR/.-php?id_doc=1178&id=GS1%20128.pdf&folder=documento_socio, aunque no es necesario para realizar la práctica pues utilizaremos una versión simplificada, y no necesariamente correcta de los mismos.) Ejemplos de códigos, obtenidos con un generador que se entrega junto a la práctica, son:

- (02)8423(10)0980(11)141215
- (02)8423(10)0981(11)141215
- (02)8324(10)0982(13)141010
- (03)842442(10)211(73)0120(11)12334
- (01)0283428792861(00)61009429(10)13061567017(37)949293936

- (01)2280008351471(10)55261553(01)24270549214(00)633449065
- (00)43138107(00)61008194(02)25302556869(10)3249883(02)27649784857924
- (00)48922275014(01)2066504(01)5361653843303(00)3253312(00)0598954
- (00)5225693162600(10)129503060529(37)934291064(10)62825123(02)50617840299718
- (00)1571291688(10)8604376635(00)2973865108

1.4. Especificación del tipo

En esta práctica nos proponemos ayudar a la multinacional en la gestión de los códigos diseñando un nuevo contenedor, que lo llamaremos gs1Set, siguiendo la especificación que nos ha proporcionado, en gs1Set.h. El objetivo es el de permitir el acceso lo más rápido posible a un determinado código, asumiendo que cada uno de los códigos que han sido obtenidos (por ejemplo, leidos de un código de barras del producto) son correctos.

1.5. Uso del gs1Set

1.5.1. Declaración del gs1Set

un sg1Set debe declararse como sigue

```
#include <string>
#include "sg1Set.h"

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

   sg1Set epc;

    return 0;
}
```

1.5.2. Inserción y borrado de elementos

```
#include "sg1Set.h"
#include <string>
#include <iostream>
 using namespace std;
int main(int argc, char ** argv) {

    sg1Set epc;


    // insercion
    if (epc.insert("(02)8423(10)0980(11)141215") {
        cout << "Insertado codigo"<< endl;
      }
       epc.insert("(02)8423(10)0980(11)141215");
       epc.insert("(02)8423(10)0981(11)141215");
```

```
        epc.insert("(03)842442(10)211(73)0120(11)12334");


    // borrado
    if (epc.erase("(03)842442(10)211(73)0120(11)12334")) {
       cout << "borrado" << endl;
    if (epc.erase("(02)8423") {
         cout << "borrado de todos los códigos que comienzan con (02)8423" <<
     endl;
    return 0;
}
```

1.5.3. buscar un valor

```
#include "sg1Set.h"
#include <string>
#include <iostream>
using namespace std;
int main(int argc, char ** argv) {

    sg1Set epc;
    sg1Set epc::iterator it;

    // adding key value pair to the Trie
    if (epc.insert("(02)8423(10)0980(11)141215") {
        cout << "Insertado codigo"<< endl;
      }
       epc.insert("(02)8423(10)0980(11)141215");
       epc.insert("(02)8423(10)0981(11)141215");
       epc.insert("(03)842442(10)211(73)0120(11)12334");
    if (epc.find("(02)8423(10)0980")!=epc.end() {
       cout << "código encontrado" <<  endl;
    }
    it = epc.find("(02)8423(10)0980");
     cout << "Tiene " << (*it).second << "subcodigos en el siguiente nivel" <<
      std::endl;
    }
    while (it!= epc.end()){
          cout << (*it).first << (*it).second << endl;
          ++it;
      }
    return 0;
}
```

1.6. Representación del tipo gs1Set

Para representar el conjuto de códigos utilizaremos un árbol general, donde por ejemplo dado el código con el formato

```
(yy)xxxxz(yyy)xxxxxxxz(yy)xxxxxz
```

en cada nodo almacenamos o bien un IA, representado por los caracteres yy en el código y que debe ir entre paréntesis, o bien cada uno de los caracteres aislados del subcódigo. Así, dado el código (02)8423(10)0985(11)141217, podemos identificar tres subcódigos, el primero para el IA=(02) con valor 8423, el segundo asociado al IA (10) con valor 0985 y el tercero asociado al IA (11) con valor 141217.

La siguiente figura nos muestra cómo se representa el código,

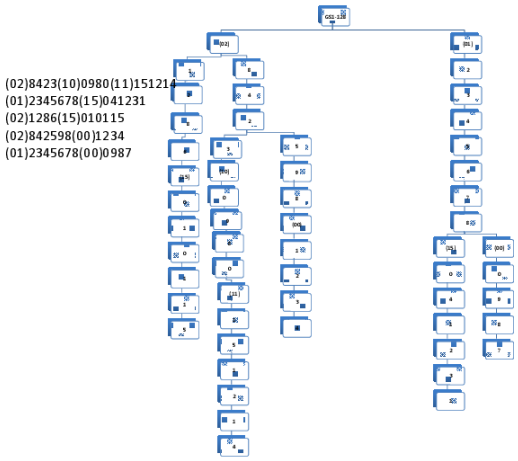

Figura 1.2: Árbol asociado a distintos códigos gs1

Así, siguiendo el camino desde la raíz del árbol hasta cada hoja obtenemos la secuencia de símbolos que representa al código. Si nos ubicamos en un nodo, el camino hacia la raíz nos representa un prefijo del código que es común a cada uno de los hijos donde cada uno representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre.

La búsqueda de un código, implica avanzar desde la raíz descendiendo de forma similar a como se hacen una búsqueda en un diccionario. Donde dado un nodo, que llamaremos el nodo actual y que representa a un caracter del código, se busca dentro de la lista de sus hijos el nodo que contienen el caracter siguiente. Una vez encontrado, dicho nodo pasa a ser el nodo actual y el proceso se repite hasta alcanzar un nodo hoja, (final de código) o bien hasta que no encontremos el caracter dentro de los hijos, por lo que decidimos que el código no está en el árbol.

Una de las ventajas de esta estructura es que las búsquedas de claves se hace más eficiente, pues tendrá en el peor caso un ordern O(l), siendo l la longitud del código, siendo además independiente del número de elementos almacenados en el árbol. - Situación que no se da cuando consideramos otras posibles implementaciones como el set o map, donde la búsqueda depende más del número de elementos en el conjunto (O(log(n)).

Además, otra ventaja de esta representacíon es que nos permite hacer búsquedas eficientes de nodos que comparten prefijos. En nuestro caso, el prefijo puede representar el país, la empresa, el tipo de paquete y/o producto, etc.

Por tanto, y para facilitar estas búsquedas, dentro del árbol, distinguimos un tipo de nodo especial que se corresponde con los caractereres fin de subcódigo, representados como z en el ejemplo anterior, esto es, aquel caracter que precede al paréntesis de apertura '(' y el último caracter del código.

Con el objetivo de poder identificar tanto IA como caracteres individuales, la etiqueta asociada a cada node contendrá un string, pero además debe contener un valor entero cuya semántica se muestra a continuación. Por tanto, el gs1 se codificará como

```
tree< pair<string,int> > arbol;
```

donde dado un nodo n, de tipo tree< pair<string,int> >::node, tenemos que (∗n) es una referencia al par que almacena, cuyo primer elemento , esto es (∗n).first, es un string que representa un indentificador de aplicación IA, o un caracter aislado del código, en cuyo caso (∗n).first.size()==0. En este último caso, necesariamente debe tomar valores en {0123456789}. Por otro lado, el segundo elemento del par, (∗n).second, es un entero que se utiliza para:

- identificar si el nodo es un IA, en cuyo caso debe verificarse que ((∗n).second < 0)
- identificar si el nodo contiene el caracter final de subcódigo (representado por z en el ejemplo anterior), donde (∗n).second > 0. En este caso, el valor será el número total de codigos completos que cuelgan del mismo. Destacar que este valor NO corresponde con el número de hijos directos, sino con el número total de hojas que cuelgan del subárbol.
- Los hijos de un nodo deben estar NECESARIAMENTE ORDENADOS de menor a mayor, esto es

  ```
  (*n).first < (*n.right_sibling()).first
  ```

1.7. Ficheros que se entregan

Descripción de los ficheros

- gs1Set.h

Fichero de especificacion de la clase gs1Set

- gs1Set.cpp

Fichero de implementacion de la clase gs1Set. En este caso, se entregan algunos codigos implementados. OJO estos códigos NO son los correctos, sólo se entregan a modo de ejemplo para simplificar (sirviendo como ayuda) el comienzo de la implementaciones que se deben realizar con gs1Set

- tree.h

Especificación de la clase tree

- tree.hxx

- nodetree.hxx

Estos dos ficheros contiene la implementacion de la clase tree, no se deben tocar.

- generadorCodigos.cpp Generador de códigos, este fichero permite generar tantos códigos de ejemplo como sea necesario. Para compilarlo podemos usar

```
g++ -o generador generadorCodigos.cpp -std=c++0x
```

y para obtener, por ejemplo, el fichero "datos1M.txt" con un millón de códigos (1000000) podemos ejecutar

```
./generador 1000000 > datos1M.txt
```

- prueba_gs1.cpp

Fichero de prueba, donde se incluyen el código para cargar un fichero de códigos en cualquier contenedor de string,

```
 @brief Carga el fichero en memoria
 @param contenedor contenedor de salida
 @param s nombre del fichero
 @pre T debe tener el método insert(const string &)
template <typename T>
void load(T  &  contenedor, const string & s) {
 ifstream fe;
 string cadena;
 cout << "Abrimos "<< s << endl;
 fe.open(s.c_str(), ifstream::in);
 if (fe.fail())
 {
   cerr << "Error al abrir el fichero " << s << endl;
 } else {

    while ( !fe.eof() )
      { getline(fe,cadena,'\n');
        if (!fe.eof()) {
           cout << "leo:: "<< cadena << endl;

           contenedor.insert(cadena);
         }
     }

    } // else
  fe.close();
 }
```

Para compilar el fichero de prueba es necesario realizar

```
> g++ -o prueba prueba_gs1.cpp gs1Set.cpp -std=c++0x
```

y lo ejecutamos como

```
./prueba datos100.txt
```

1.8. Tareas a Realizar.

Se pide implementar los metodos de la clase gs1Set y un codigo de prueba donde se inserten, busquen y borren al menos un millón de códigos obtenidos con el generador. Hacer una comparativa en tiempo de ejecución cuando utilizamos directamente un contenedor asociativo set y unordered set:

- unordered_set<string>
- set<string>

Capítulo 2

Lista de tareas pendientes

Clase gs1Set

Tareas a realizar: El alumno debera implementar la clase gs1Set , junto con el codigo de prueba de los distintos metodos.

Miembro gs1Set::insert (const string &s)

implementar este metodo correctamente OJO ESTE METODO SOLO SIRVE PARA DAR UNA IDEA DEL PROCESO DE INSERCION, HAY QUE IMPLEMENTARLO CORRECTAMENTE

Miembro gs1Set::reading_gs1Set ()

implementar este metodo correctamente OJO ESTE METODO OS SIRVE PARA PODER CONSTRUIR UN ARBOL NO TIENE EN CUENTA EL INVARIANTE DE LA REPRESENTACION AL NO CONSIDERAR EL CAMPO INT DEL NODO!!!!! DEBEIS MODIFICARLO PARA QUE LO HAGA DE FORMA CORRECTA

Capítulo 3

Índice de clases

3.1. Lista de clases

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

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 5

Documentación de las clases

5.1. Referencia de la Clase tree::const_inorderiterator

Métodos públicos

- **const_inorderiterator** (node n)
- bool **operator!=** (const const_inorderiterator &i) const
- const T & **operator*** () const
- const_inorderiterator & **operator++** ()
- bool **operator==** (const const_inorderiterator &i) const

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.2. Referencia de la Clase gs1Set::const_iterator

Métodos públicos

- const_iterator ()

  Constructor primitivo.
- const_iterator (const const_iterator &it)

  Constructor de copia.
- bool **operator!=** (const const_iterator &it) const
- string operator* ()

  devuelve el codigo completo al que apunta el iterador. Nota: Se encuentra en el camino que hay desde el nodo hacia la raiz.
- const_iterator & operator++ ()

avanza hacia el siguiente final de (sub)codigo en preorden en el gs1Set . Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.

- const_iterator operator++ (int)

  avanza hacia el siguiente final de (sub)codigo en preorden en el gs1Set. Nota: Avanza por el arbol hasta el siguiente nodo que es final de (sub)codigo.
- const_iterator & **operator=** (const const_iterator &it)
- bool **operator==** (const const_iterator &it) const
- const_iterator & upper_IA ()

  sube hacia el identificador de aplicacion (IA) anterior en el codigo.

Amigas

- class **gs1Set**

5.2.1. Descripción detallada

const_iterator: const_iterator, operator∗, operator++

Descripcion

Un objeto de la clase gs1Set::const_iterator representara un iterador sobre el conjunto de codigos en el gs1Set .

Nota: Solo itera sobre (subcodigos) y no sobre los elementos individuales

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

- gs1Set.h

5.3. Referencia de la Clase tree::const_leveliterator

Métodos públicos

- **const_leveliterator** (node n)
- bool **operator!=** (const const_leveliterator &i) const
- const T & **operator∗** () const
- const_leveliterator & **operator++** ()
- bool **operator==** (const const_leveliterator &i) const

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.4. Referencia de la Clase tree::const_node

Métodos públicos

- const_node ()

  Constructor primitivo.
- const_node (const const_node &n)

  Constructor copia.
- const_node (const node &n)

  Constructor copia, transforma un node en un const_node.
- const_node left () const

  Devuelve el hizo izquierdo del nodo receptor.
- const_node next_sibling () const

  Devuelve el hermano derecho del nodo receptor.
- bool null () const

  Devuelve si el nodo es nulo.
- bool operator!= (const const_node &n) const

  Operador de comparación de desigualdad.
- const T & operator* () const

  Devuelve la etiqueta del nodo.
- const_node & operator= (const const_node &n)

  Operador de asignación.
- bool operator== (const const_node &n) const

  Operador de comparación de igualdad.
- const_node parent () const

  Devuelve el padre del nodo receptor.

Amigas

- class **tree**< **T** >

5.4.1. Documentación de las funciones miembro

5.4.1.1. tree< T >::const_node tree::const_node::left ( ) const [inline]

Devuelve el hizo izquierdo del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 247 del archivo nodetree.hxx.

5.4.1.2. tree< T >::const_node tree::const_node::next_sibling ( ) const [inline]

Devuelve el hermano derecho del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 254 del archivo nodetree.hxx.

5.4.1.3. bool tree::const_node::operator!= (const const_node & *n*) const [inline]

Operador de comparación de desigualdad.

Parámetros

n	el nodo con el que se compara

Definición en la línea 276 del archivo nodetree.hxx.

5.4.1.4. const T & tree::const_node::operator* ( ) const [inline]

Devuelve la etiqueta del nodo.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 261 del archivo nodetree.hxx.

5.4.1.5. const_node& tree::const_node::operator= (const const_node & *n*) [inline]

Operador de asignación.

Parámetros

n	el nodo a asignar

5.4.1.6. bool tree::const_node::operator== (const const_node & *n*) const [inline]

Operador de comparación de igualdad.

Parámetros

n	el nodo con el que se compara

Definición en la línea 269 del archivo nodetree.hxx.

5.4.1.7. tree< T >::const_node tree::const_node::parent ( ) const [inline]

Devuelve el padre del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 240 del archivo nodetree.hxx.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- nodetree.hxx

5.5. Referencia de la Clase tree::const_postorderiterator

Métodos públicos

- **const_postorderiterator** (node n)
- bool **operator!=** (const const_postorderiterator &i) const
- const T & **operator*** () const
- const_postorderiterator & **operator++** ()
- bool **operator==** (const const_postorderiterator &i) const

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.6. Referencia de la Clase tree::const_preorderiterator

Métodos públicos

- **const_preorderiterator** (node n)
- bool **operator!=** (const const_preorderiterator &i) const
- const T & **operator*** () const
- const_preorderiterator & **operator++** ()
- bool **operator==** (const const_preorderiterator &i) const

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.7. Referencia de la Clase gs1Set

`#include <gs1Set.h>`

Clases

- class const_iterator

Tipos públicos

- typedef unsigned int size_type

Métodos públicos

- const_iterator begin () const

  iterador a la primera palabra del conjunto.
- list< string > codesWithPrefix (const string &pr)

  obtiene todos los codigos que tienen la misma secuencia prefijo
- bool empty () const

  Chequea si el gs1Set esta vacio (size()==0)
- const_iterator end () const

  iterador al fin del conjunto
- bool erase (const string &s)

  elimina el codigo de un gs1Set
- const_iterator find (const string &s)

  busca una codigo
- gs1Set ()

  Constructor primitivo crea un gs1Set con el caracter '-' en el nodo raiz.
- gs1Set (const gs1Set &x)

  Constructor de copia.
- int insert (const string &s)

  Inserta una nuevo codigo dentro del gs1Set.
- gs1Set & operator= (const gs1Set &org)

  operador de asignacion
- void print () const

  imprime todos los codigos almacenados
- void reading_gs1Set ()

  Lectura de un gs1Set por teclado. Se genera el gs1Set utilizando un recorrido por nivel.
- size_type size () const

  tamanioo

5.7.1. Descripción detallada

gs1Set :gs1Set , operator=, size, empty, insert, erase, find,

Descripcion

(las tildes han sido omitidas deliberadamente, debido a fallos en la generacion de documentacion con doxygen)

Un objeto de la clase gs1Set representara un contenedor que permite almacenar un codigo electronico de producto, en concreto consideraremos la normativa gs1-128

El gs1-128 es un sistema estandar de identificacion mediante codigo de barras utilizado internacionalmente para la identificacion de mercancias en entornos logisticos y no detallistas. Este sistema se utiliza principalmente para la identificacion de unidades de expedicion.

En esta practica consideraremos codigos correctos, aquellos obtenidos segun la siguiente definicion: Una codigo correcto solo puede contener digitos (del 0 al 9) junto con parentesis de apertura y cierre. Por tanto, no debe tener espacios ni delimitadores del tipo comillas, llaves, comas, puntos, etc.

El codigo gs1-128 vendra se representara mediante un string, ejemplos de codigos validos son: (01)18456789012342 (02)18456789012359(37)1234(00)384567890123456782

En la practica NO nos preocuparemos de generar codigos correctos, sino que consideraremos como correcto cualquier codigo que tenga

En el codigo podemos distinguir dos partes, encerrados entre parentesis los identificadores de aplicacion, IA, que son unos prefijos numericos creados para dar significado inequivoco a los elementos de datos estandarizados que se encuentran situados a continuacion (son subcodigos dentro el codigo gs1). Cada prefijo identifica el significado y el formato de los subcadigos que le siguen.

En la actualidad, existen mas de 100 identificadores de aplicacion estandarizados internacionalmente. Por ejemplo,

- 00 Codigo Seriado de la Unidad de Envio (SSCC)
- 01 Codigo de agrupacion
- 02 Codigo del articulo / agrupacion contenido
- 37 Cantidades (va junto al IA 02)
- 10 Numero de lote
- 11 Fecha de fabricacion
- 13 Fecha de envasado
- 15 Fecha de consumo preferente
- 17 Fecha de caducidad etc.

En la practica NO nos preocuparemos de generar codigos correctos, sino que consideraremos como correcto cualquier codigo que tenga el formato

(yy)xxxxz(yyy)xxxxxxxz(yy)xxxxxz, donde yyy e xxxxxz son digitos del 0 al 9 representando el IA y el codigo asociado. (El alumno interesado puede consultar como se construyen los codigos enel enlace http://www.aecoc.es/BAJAR/.php?id-_doc=1178&id=GS1%20128.pdf&folder=documento_socio

Veamos un ejemplo simple, podemos considerar los siguientes codigos

(02)8423(10)0980(11)141215 (02)8423(10)0981(11)141215 (02)8324(10)0982(13)141010 (03)842442(10)211(73)0120(11)12334

El objetivo de la practica es construir un contenedor de codigos gs1, donde el objetivo principal es permitir el acceso lo mas rapido posible a un determinado codigo, asumiendo que cada uno de los codigos que han sido obtenidos (por ejemplo, leidos de un codigo de barras del productos) es correcto.

Tareas pendientes Tareas a realizar: El alumno debera implementar la clase gs1Set , junto con el codigo de prueba de los distintos metodos.

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

5.7.2.1. typedef gs1Set::size_type

Hace referencia al tipo asociado al numero de elementos en el codigo.

Definición en la línea 88 del archivo gs1Set.h.

5.7.3. Documentación del constructor y destructor

5.7.3.1. gs1Set::gs1Set (const gs1Set & *x*)

Constructor de copia.

Parámetros

in	*x*	gs1Set que se copia

Definición en la línea 12 del archivo gs1Set.cpp.

5.7.4. Documentación de las funciones miembro

5.7.4.1. const_iterator gs1Set::begin ( ) const

iterador a la primera palabra del conjunto.

Este iterador debe apuntar al nodo en el que se encuentra el ultimo caracter de la primera palabra en el conjunto.

5.7.4.2. list<string> gs1Set::codesWithPrefix (const string & *pr*)

obtiene todos los codigos que tienen la misma secuencia prefijo

Parámetros

in	*pr*	prefijo a buscar

Devuelve

una lista con todos los codigos epc que contienen el mismo prefijo

5.7.4.3. bool gs1Set::erase (const string & *s*)

elimina el codigo de un gs1Set

Parámetros

in	*s*	elemento a borrar. Este elemento puede identificar a un prefijo, por ejemplo el codigo asociado a un producto, por lo que todos los codigos que contienen dicho prefijo seran eliminados.

Devuelve

el numero de codigos que se han borrado, cero si el borrado no se ha podido realizar con exito

Postcondición

el size() sera decrementado.

5.7.4.4. const_iterator gs1Set::find (const string & *s*)

busca una codigo

Parámetros

in	*s*	nombre del codigo (o prefijo) a buscar

Devuelve

un iterador que apunta al codigo o end() si el codigo (prefijo) no existe.

5.7.4.5. bool gs1Set::insert (const string & *s*)

Inserta una nuevo codigo dentro del gs1Set.

insercion del string, cada caracter en una posicion independiente

Parámetros

in	*s*	elemento a insertar

Devuelve

bool true si la insercion se ha podido realizar con exito, esto es, el codigo no pertenecia al gs1Set

Postcondición

el size() sera incrementado

Tareas pendientes implementar este metodo correctamente OJO ESTE METODO SOLO SIRVE PARA DAR UNA IDEA DEL PROCESO DE INSERCION, HAY QUE IMPLEMENTARLO CORRECTAMENTE

Definición en la línea 22 del archivo gs1Set.cpp.

5.7.4.6. gs1Set& gs1Set::operator= (const gs1Set & *org*)

operador de asignacion

Parámetros

in	*org*	gs1Set a copiar. Crea un gs1Set duplicado exacto a org.

5.7.4.7. void gs1Set::reading_gs1Set ( )

Lectura de un gs1Set por teclado. Se genera el gs1Set utilizando un recorrido por nivel.

Tareas pendientes implementar este metodo correctamente OJO ESTE METODO OS SIRVE PARA PODER CONSTRUIR UN ARBOL NO TIENE EN CUENTA EL INVARIANTE DE LA REPRESENTACION AL NO CONSIDERAR EL CAMPO INT DEL NODO!!!!! DEBEIS MODIFICARLO PARA QUE LO HAGA DE FORMA CORRECTA

Definición en la línea 70 del archivo gs1Set.cpp.

5.7.4.8. size_type gs1Set::size ( ) const

tamanioo

Devuelve

delvuelve el numero de palabras del gs1Set

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- gs1Set.h
- gs1Set.cpp

5.8. Referencia de la Clase tree::inorderiterator

`#include <tree.h>`

Métodos públicos

- **inorderiterator** (node n)
- bool **operator!=** (const inorderiterator &i) const
- T & **operator*** ()
- inorderiterator & **operator++** ()
- bool **operator==** (const inorderiterator &i) const

5.8.1. Descripción detallada

Clase iterator para recorrer el árbol en Inorder

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.9. Referencia de la Clase tree::leveliterator

`#include <tree.h>`

Métodos públicos

- **leveliterator** (node n)
- bool **operator!=** (const leveliterator &i) const
- T & **operator*** ()
- leveliterator & **operator++** ()
- bool **operator==** (const leveliterator &i) const

5.9.1. Descripción detallada

Clase iterator para recorrer el árbol por niveles

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.10. Referencia de la Clase tree::node

Métodos públicos

- node left () const

  Devuelve el hizo izquierdo del nodo receptor.
- node next_sibling () const

  Devuelve el hermano derecho del nodo receptor.
- node ()

  Constructor primitivo.
- node (const node &n)

  Constructor copia.
- bool null () const

  Devuelve si el nodo es nulo.
- bool operator!= (const node &n) const

  Operador de comparación de desigualdad.
- T & operator∗ ()

  Devuelve la etiqueta del nodo.
- const T & operator∗ () const

  Devuelve la etiqueta del nodo.
- node & operator= (const node &n)

  Operador de asignación.
- bool operator== (const node &n) const

  Operador de comparación de igualdad.
- node parent () const

  Devuelve el padre del nodo receptor.
- void setlabel (const T &e)

  Modifica la etiqueta.

Amigas

- class **tree**< **T** >

5.10.1. Descripción detallada

Descripción

Representa a los nodos del árbol

Descripción

Representa a los nodos del árbol, se usa con arboles constantes

5.10.2. Documentación de las funciones miembro

5.10.2.1. tree< T >::node tree::node::left ( ) const [inline]

Devuelve el hizo izquierdo del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 121 del archivo nodetree.hxx.

5.10.2.2. tree< T >::node tree::node::next_sibling ( ) const [inline]

Devuelve el hermano derecho del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 128 del archivo nodetree.hxx.

5.10.2.3. bool tree::node::operator!= (const node & *n*) const [inline]

Operador de comparación de desigualdad.

Parámetros

n	el nodo con el que se compara

Definición en la línea 163 del archivo nodetree.hxx.

5.10.2.4. T & tree::node::operator* ( ) [inline]

Devuelve la etiqueta del nodo.

Precondición

Si se usa como consultor, !n.Nulo()

Definición en la línea 135 del archivo nodetree.hxx.

5.10.2.5. const T & tree::node::operator∗ ( ) const [inline]

Devuelve la etiqueta del nodo.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 142 del archivo nodetree.hxx.

5.10.2.6. node& tree::node::operator= (const node & *n*) [inline]

Operador de asignación.

Parámetros

n	el nodo a asignar

5.10.2.7. bool tree::node::operator== (const node & *n*) const [inline]

Operador de comparación de igualdad.

Parámetros

n	el nodo con el que se compara

Definición en la línea 156 del archivo nodetree.hxx.

5.10.2.8. tree< T >::node tree::node::parent ( ) const [inline]

Devuelve el padre del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

Definición en la línea 114 del archivo nodetree.hxx.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- nodetree.hxx

5.11. Referencia de la Clase tree::nodewrapper

Métodos públicos

- **nodewrapper** (const T &)

Atributos públicos

- T **etiqueta**
- node **hermanodcha**
- node **izda**
- node **pad**

5.11.1. Descripción detallada

TDA nodo. Modela los nodos del árbol.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- nodetree.hxx

5.12. Referencia de la Clase tree::postorderiterator

```
#include <tree.h>
```

Métodos públicos

- bool **operator!=** (const postorderiterator &i) const
- T & **operator**∗ ()
- postorderiterator & **operator++** ()
- bool **operator==** (const postorderiterator &i) const
- **postorderiterator** (node n)

5.12.1. Descripción detallada

Clase iterator para recorrer el árbol en PostOrden

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.13. Referencia de la Clase tree::preorderiterator

`#include <tree.h>`

Métodos públicos

- bool **operator!=** (const preorderiterator &i) const
- T & **operator*** ()
- preorderiterator & **operator++** ()
- preorderiterator **operator++** (int)
- bool **operator==** (const preorderiterator &i) const
- **preorderiterator** (node n)

5.13.1. Descripción detallada

Clase iterator para recorrer el árbol en PreOrden

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

5.14. Referencia de la Clase tree

`#include <tree.h>`

Clases

- class const_inorderiterator
- class const_leveliterator
- class const_node
- class const_postorderiterator
- class const_preorderiterator
- class inorderiterator
- class leveliterator
- class node
- class nodewrapper
- class postorderiterator
- class preorderiterator

Tipos públicos

- typedef unsigned int size_type

Métodos públicos

- void assign_subtree (const tree< T > &a, node n)

  Reemplaza el receptor por una copia de subárbol.
- inorderiterator **beginInorder** ()
- const_inorderiterator **beginInorder** () const
- leveliterator **beginlevel** ()
- const_leveliterator **beginlevel** () const
- postorderiterator **beginPostorder** ()
- const_postorderiterator **beginPostorder** () const
- preorderiterator **beginPreorder** ()
- const_preorderiterator **beginPreorder** () const
- void clear ()

  Hace nulo un árbol.
- bool empty () const

  Comprueba si un árbol esta vacío.
- inorderiterator **endInorder** ()
- const_inorderiterator **endInorder** () const
- leveliterator **endlevel** ()
- const_leveliterator **endlevel** () const
- postorderiterator **endPostorder** ()
- const_postorderiterator **endPostorder** () const
- preorderiterator **endPreorder** ()
- const_preorderiterator **endPreorder** () const
- node insert_left (node n, const T &e)

  Insertar un nodo como hijo a la izquierda de un nodo.
- node insert_left (node n, tree< T > &rama)

  Insertar un árbol como subárbol hijo a la izquierda de un nodo.
- node insert_right_sibling (node n, const T &e)

  Insertar un nodo como hermano a la derecha de un nodo.
- node insert_right_sibling (node n, tree< T > &rama)

  Insertar un árbol como subárbol hermano a la derecha de un nodo.
- bool is_external (node v) const

  Comprueba si un nodo es exterior.
- bool is_internal (node v) const

  Comprueba si un nodo es interior.
- bool is_root (node n) const

  Comprueba si un nodo es la raíz.
- bool null () const

  Comprueba si un árbol es nulo.
- bool operator!= (const tree< T > &a) const

  Operador de comparación de desigualdad.
- tree< T > & operator= (const tree< T > &a)

  Operador de asignación.
- bool operator== (const tree< T > &a) const

Operador de comparación de igualdad.

- void prune_left (node n, tree< T > &dest)

  Podar el subárbol hijo a la izquierda de un nodo.

- void prune_right_sibling (node n, tree< T > &dest)

  Podar el subárbol hermano a la derecha de un nodo.

- node root () const

  Obtener el nodo raíz.

- node setroot (const T &v)

  Asigna la raiz al arbol vacío.

- size_type size () const

  Obtiene el nmero de nodos.

- tree ()

  Constructor primitivo por defecto.

- tree (const T &e)

  Constructor primitivo.

- tree (const tree< T > &a)

  Constructor de copia.

- ~tree ()

  Destructor.

5.14.1. Descripción detallada

tree::tree, assign_subtree, setroot, root, ~tree, =, prune_left, prune_right_sibling,insert-_left, insert_right_sibling, clear, size, empty, ==, !=, is_root, internal, external

Representa un árbol general con nodos etiquetados con datos del tipo T.

T debe tener definidas las operaciones:

- T & operator=(const T & e);
- bool operator!=(const T & e);
- bool operator==(const T & e);

Son mutables. Residen en memoria dinámica.

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

5.14.2.1. tree::size_type

Hace referencia al tipo asociado la tamaño del tree

Definición en la línea 46 del archivo tree.h.

5.14.3. Documentación del constructor y destructor

5.14.3.1. tree::tree ( ) [inline]

Constructor primitivo por defecto.

Crea un árbol nulo.

Definición en la línea 109 del archivo tree.hxx.

5.14.3.2. tree::tree (const T & *e*)

Constructor primitivo.

Parámetros

e,:	Etiqueta para la raíz.

Crea un árbol con un único nodo etiquetado con e.

Definición en la línea 115 del archivo tree.hxx.

5.14.3.3. tree::tree (const tree< T > & *a*)

Constructor de copia.

Parámetros

	a árbol que se copia.

Crea un árbol duplicado exacto de a.

Definición en la línea 131 del archivo tree.hxx.

5.14.3.4. tree::~tree ( ) [inline]

Destructor.

Destruye el receptor liberando los recursos que ocupaba.

Definición en la línea 168 del archivo tree.hxx.

5.14.4. Documentación de las funciones miembro

5.14.4.1. void tree::assign_subtree (const tree< T > & *a,* node *n*)

Reemplaza el receptor por una copia de subárbol.

Parámetros

a,:	árbol desde el que se copia.
n,:	nodo raíz del subárbol que se copia.

El receptor se hace nulo y después se le asigna una copia del subárbol de a cuya raíz es n.

5.14.4.2. void tree::clear ( )

Hace nulo un árbol.

Destruye todos los nodos del árbol receptor y lo hace un árbol nulo.

Definición en la línea 296 del archivo tree.hxx.

5.14.4.3. bool tree::empty ( ) const [inline]

Comprueba si un árbol esta vacío.

Devuelve

true, si el receptor es un árbol vacío. false, en otro caso.

Definición en la línea 313 del archivo tree.hxx.

5.14.4.4. node tree::insert_left (node *n,* const T & *e*)

Insertar un nodo como hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
e,:	etiqueta del nuevo nodo.

Inserta un nuevo nodo con etiqueta e como hijo a la izquierda, el anterior hijo más a la izquierda queda como hermano a la derecha del recién insertado

5.14.4.5. node tree::insert_left (node *n,* tree< T > & *rama*)

Insertar un árbol como subárbol hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
rama,:	subárbol que se inserta. Es MODIFICADO.

Si rama no es un árbol vacío: Inserta rama como hijo a la izquierda de n, el anterior hijo más a la izquierda queda como hermana a la derecha del recién insertado. y rama se hace árbol nulo. En caso contrario no se hace nada

5.14.4.6. node tree::insert_right_sibling (node *n,* const T & *e*)

Insertar un nodo como hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. !n.Nulo().
e,:	etiqueta del nuevo nodo.

Inserta un nuevo nodo con etiqueta e como hermano a la derecha, el anterior hermano a la derecha de n queda como hermano a la derecha del nodo insertado

5.14.4.7. node tree::insert_right_sibling (node *n,* tree< T > & *rama*)

Insertar un árbol como subárbol hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. !n.Nulo().
rama,:	subárbol que se inserta. Es MODIFICADO.

Si rama no es un árbol vacío: Asigna el valor de rama como nuevo subárbol hermano a la derecha, el anteriot hermano a la derecha de n queda como hermano a la derecha del nodo insertado y rama se hace árbol nulo. En caso contrario no se hace nada

5.14.4.8. bool tree::is_external (node *v*) const [inline]

Comprueba si un nodo es exterior.

Parámetros

v,:	nodo que se evala.

Devuelve

true, si n es exterior. false, en otro caso.

Definición en la línea 247 del archivo tree.h.

5.14.4.9. bool tree::is_internal (node *v*) const [inline]

Comprueba si un nodo es interior.

Parámetros

v,:	nodo que se evala.

Devuelve

true, si n es interior. false, en otro caso.

Definición en la línea 237 del archivo tree.h.

5.14.4.10. bool tree::is_root (node *n*) const [inline]

Comprueba si un nodo es la raíz.

Parámetros

n,:	nodo que se evala.

Devuelve

true, si n es la raíz del receptor. false, en otro caso.

Definición en la línea 227 del archivo tree.h.

5.14.4.11. bool tree::null ( ) const

Comprueba si un árbol es nulo.

Devuelve

true, si el receptor es un árbol nulo. false, en otro caso.

5.14.4.12. bool tree::operator!= (const tree< T > & *a*) const [inline]

Operador de comparación de desigualdad.

Parámetros

a,:	árbol con que se compara el receptor.

Devuelve

true, si el receptor no es igual, en estructura o etiquetas a a. false, en otro caso.

Definición en la línea 327 del archivo tree.hxx.

5.14.4.13. tree< T > & tree::operator= (const tree< T > & *a*)

Operador de asignación.

Parámetros

a,:	árbol que se asigna.

Destruye el contenido previo del receptor y le asigna un duplicado de a.

Definición en la línea 176 del archivo tree.hxx.

5.14.4.14. bool tree::operator== (const tree< T > & *a*) const [inline]

Operador de comparación de igualdad.

Parámetros

a,:	árbol con que se compara el receptor.

Devuelve

true, si el receptor es igual, en estructura y etiquetas a a. false, en otro caso.

Definición en la línea 320 del archivo tree.hxx.

5.14.4.15. void tree::prune_left (node *n,* tree< T > & *dest*)

Podar el subárbol hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
dest,:	subárbol hijo a la izquierda de n. Es MODIFICADO.

Desconecta el subárbol hijo a la izquierda de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

5.14.4.16. void tree::prune_right_sibling (node *n,* tree< T > & *dest*)

Podar el subárbol hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
dest,:	subárbol hermano a la derecha de n. Es MODIFICADO.

Desconecta el subárbol hermano a la derecha de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

5.14.4.17. tree< T >::node tree::root ( ) const [inline]

Obtener el nodo raíz.

Devuelve

nodo raíz del receptor.

Definición en la línea 189 del archivo tree.hxx.

5.14.4.18. tree< T >::node tree::setroot (const T & *v*)

Asigna la raiz al arbol vacío.

Parámetros

v,:	el valor a almacenar en la raiz.

Precondición

el receptor es el árbol nulo.

Definición en la línea 121 del archivo tree.hxx.

5.14.4.19. tree< T >::size_type tree::size ( ) const [inline]

Obtiene el nmero de nodos.

Devuelve

nmero de nodos del receptor.

Definición en la línea 306 del archivo tree.hxx.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- tree.h
- tree.hxx

Capítulo 6

Documentación de archivos

6.1. Referencia del Archivo tree.h

TDA tree.

```
#include <queue>  #include <iostream>  #include <stack> ×
#include <list>  #include "tree.hxx"  #include "nodetree.-
hxx"
```

Clases

- class tree::const_inorderiterator
- class tree::const_leveliterator
- class tree::const_node
- class tree::const_postorderiterator
- class tree::const_preorderiterator
- class tree::inorderiterator
- class tree::leveliterator
- class tree::node
- class tree::nodewrapper
- class tree::postorderiterator
- class tree::preorderiterator
- class tree

6.1.1. Descripción detallada

TDA tree.

Definición en el archivo tree.h.