Ejercitación – Desarrollo de iteradores

Introducción

El objetivo de esta ejercitación es implementar un *iterador* para la clase `Taxonomia<T>`. Una taxonomía consta de una serie de categorías organizadas de manera jerárquica. Por ejemplo, la siguiente es una taxonomía (incompleta) de organismos eucariotas:

```
eucariotas {
  animales {
    vertebrados {
      mamíferos { ballenas jirafas murciélagos zorros }
      peces { esturiones truchas }
      reptiles { serpientes }
    }
    invertebrados {
      insectos { abejas hormigas moscas }
      moluscos { caracoles }
    }
  }
  plantas {
    angiospermas { manzanos }
    gimnospermas { araucarias cedros }
  }
}
```

En esta taxonomía hay varias **categorías**. Cada categoría se representa con un nombre de tipo `T`. Por ejemplo, `vertebrados`, `moluscos` y `araucarias` son categorías. La categoría que encabeza la taxonomía, en este caso la categoría de las `eucariotas`, se llama la **raíz** de la taxonomía. Cada categoría puede tener **subcategorías**. Por ejemplo, la categoría de los `vertebrados` tiene tres subcategorías (`mamíferos`, `peces` y `reptiles`). Todas las categorías, exceptuando a la raíz, tienen una **supercategoría**. Por ejemplo, la supercategoría de `caracoles` es `moluscos` y la supercategoría de `moluscos` es `invertebrados`.

En los archivos `Taxonomia.h` y `Taxonomia.hpp` se encuentra ya definida la clase `Taxonomia` incluyendo la representación interna (utilizando punteros), el constructor, el destructor y una operación para mostrar una taxonomía. Por ejemplo el siguiente programa:

```cpp
#include <iostream>
#include "Taxonomia.h"

int main() {
    Taxonomia<int> taxonomia("0 { 1 2 { 21 22 } 3 }");
    std::cout << taxonomia;
}
```

Muestra la taxonomía por pantalla:

```
0 {
  1
  2 {
    21
    22
  }
  3
}
```

Consigna

Se pide implementar la clase miembro `Taxonomia<T>::iterator` que representa un iterador para la clase `Taxonomia<T>`. Se debe completar la representación elegida para el iterador (en el archivo `Taxonomia.h`) y la definición de las operaciones (en el archivo `Taxonomia.hpp`). Implementar las siguientes operaciones.

Operaciones de `Taxonomia<T>`

1. `typename Taxonomia<T>::iterator Taxonomia<T>::begin()` — Devuelve un iterador válido al principio de la taxonomía.
2. `typename Taxonomia<T>::iterator Taxonomia<T>::end()` — Devuelve un iterador válido al final de la taxonomía.

Operaciones de `Taxonomia<T>::iterator`

3. `Taxonomia<T>::iterator::iterator()` — Constructor por defecto del iterador. *Nota:* puede construir un iterador inválido.
4. `T& Taxonomia<T>::iterator::operator*() const` — Referencia mutable al nombre de la categoría actual. *Precondición:* el iterador está posicionado sobre una categoría.
5. `int Taxonomia<T>::iterator::cantSubcategorias() const` — Cantidad de subcategorías de la categoría actual. *Precondición:* el iterador está posicionado sobre una categoría.
6. `void Taxonomia<T>::iterator::subcategoria(int i)` — Ubica el iterador sobre la i-ésima subcategoría. *Precondición:* el iterador está posicionado sobre una categoría y además `0 <= i < cantSubcategorias()`.
7. `bool Taxonomia<T>::iterator::esRaiz() const` — Devuelve true sii la categoría actual es la raíz. *Precondición:* el iterador está posicionado sobre una categoría.
8. `void Taxonomia<T>::iterator::supercategoria()` — Ubica el iterador sobre la supercategoría de la categoría actual. *Precondición:* el iterador está posicionado sobre una categoría y además `!esRaiz()`.
9. `bool Taxonomia<T>::iterator::operator==(const Taxonomia<T>::iterator& otro) const` — Compara dos iteradores por igualdad. *Precondición:* deben ser dos iteradores de la misma taxonomía.
10. `void Taxonomia<T>::iterator::operator++()` — Ubica el iterador sobre la categoría siguiente a la actual en el recorrido **preorder** de la taxonomía. Si se trata de la última categoría de la taxonomía, el iterador resultante debe ser igual al iterador `end()` de la taxonomía.
11. `void Taxonomia<T>::iterator::operator--()` – Ubica el iterador sobre la categoría anterior a la actual en el recorrido **preorder** de la taxonomía. Si se trata de la raíz de la taxonomía el iterador resultante debe ser igual al iterador `end()` de la taxonomía. *Precondición:* el iterador está posicionado sobre una categoría.
12. `void Taxonomia<T>::iterator::insertarSubcategoria(int i, const T& nombre)` — Inserta una subcategoría con el nombre indicado en el lugar i-ésimo. Observación: esta operación modifica la taxonomía y puede invalidar otros iteradores. *Precondición:* el iterador está posicionado sobre una categoría, y además `0 <= i <= cantSubcategorias()`.
13. `void Taxonomia<T>::iterator::eliminarCategoria()` — Elimina la categoría actual de la taxonomía (y todas sus subcategorías). Observación: esta operación modifica la taxonomía y puede invalidar otros iteradores. Debe encargarse de liberar la memoria. *Precondición:* el iterador está posicionado sobre una categoría, y además `!esRaiz()`.

La implementación que realicen **no debe perder memoria**. Recomendamos utilizar **valgrind** para testear si su implementación tiene *leaks* de memoria.