Clase #27 de 29
Templates & Árboles

Agosto 13, Jueves

Agenda para esta clase

- Metaprogramación y Programación Genérica
- Arreglos Asociativos
- Introducción a Árboles

Programación Genérica mediante Metaprogramación

Programación Genérica y Templates

- La programación genércia permite implementar tipos y funciones parametrizados
- Al implementar una función genérica, en realidad estamos implementando una familia de funciones
- Lo mismo aplica a tipos

- La metaprogramación es programar un constructo que le indique al compilador que programe en lenguaje fuente por nosotros
- Nuestro constructo resultado no es un programa, si no, un programa para que le compilador programe
- El constructo en C++ es template (molde)
- En C++, la Programación Genérica se logra con metaprogramación

- En Java no hay metaprogramación, pero sí programación genérica, con impacto en el tiempo de ejecución pero con código más breve
- C# y D siguen un modelo más similar a C++
- C lo logra con métodos nos seguros como void*.

Familia de Funciones sobre Tamaño, Tipo y Ambos

https://josemariasola.wordpress.com/aed/papers#ArraysAndTemplates

```
array<int, 5> a = {{1,2,3,4,5}};
```

```
void PrintArrayInt5(const array<int,5>& x){
   for(auto e : x)
      cout << e << '\t';
   cout << '\n';
}

PrintArrayInt5(a);
```

```
template <size_t n>
void PrintArrayInt(const array<int, n>& x){
   for(auto e : x)
      cout << e << '\t';
   cout << '\n';
}

PrintArrayInt(a);
```

```
template <typename T, size_t n>
void PrintArray(const array<T, n>& x){
   for(auto e : x)
      cout << e << '\t';
   cout << '\n';
}

PrintArray(a);
```

Familia de Tipo Matrices por Tipo & Tamaño y Defaults

```
array<array<int,3>,5> aai35;
Matrix<int,5,3>       mi53;
Matrix<>              mi11;
Matrix<double>        md11;
Matrix<char,2>        mc22;
```

```
template <typename T = int, std::size_t rows = 1, std::size_t columns = rows>
using Matrix = std::array<std::array<T,columns>,rows>;
```

```
template <typename T = int, std::size_t rows = 1, std::size_t columns = rows>
using Matrix =
  std::array<    // array
    std::array<  //   of arrays
      T,         //     of T
      columns    //     of columns elements
    >,
    rows         //   of rows elements
  >;
```

Arreglo Asociativo

a.k.a. :

- Lookup Table,
- Tabla de Búsqueda,
- Tabla de Símbolos, ó
- Mapa

Agenda Telefónica

- Problema
  - Almacenar pares de nombres y teléfonos y buscar por nombre
- Tipo de Dato Abstracto: Agenda Telefónica
  - Valores
    - Agenda={(nombre,teléfono)/nombre, teléfono in Sigma*, nombre not empty, telefono not empty}
    - Agenda=Nombre xx Teléfono, Nombre=Teléfono=Sigma*
  - Operaciones
    - Add(a,nombre,teléfono)
    - Get(a,nombre)
    - Set(a,nombre,telefono)
    - Remove(a,nombre)
- Ejemplo

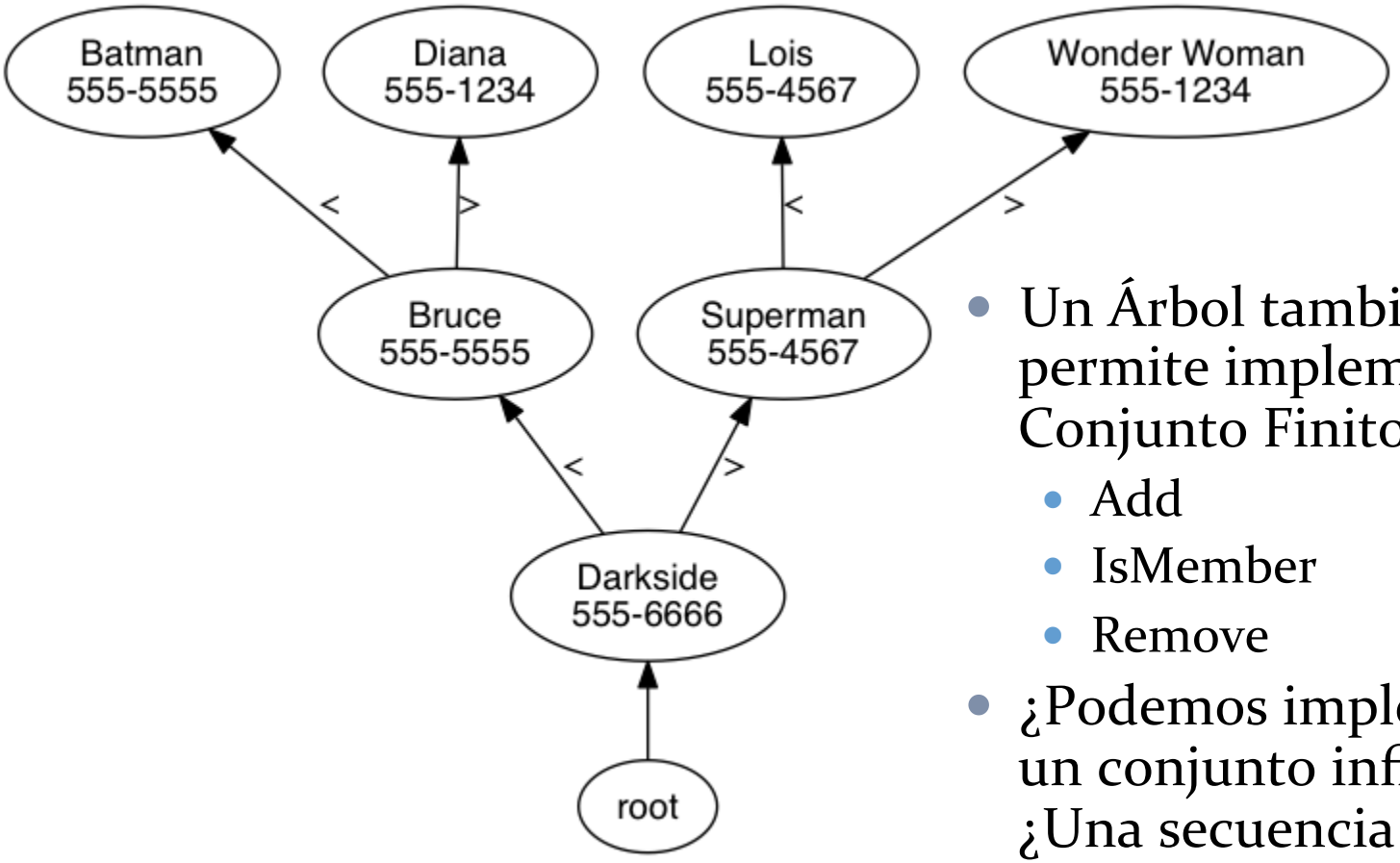
```
Agenda a;
Add(a,    "Superman","555-4567");
Add(a,      "Batman","555-5555");
Add(a,"Wonder Woman","555-1234");
Add(a,        "Lois","555-4567");
Add(a,       "Bruce","555-5555");
Add(a,       "Diana","555-1234");
Add(a,    "Darkside","555-6666");
cout<<Get(a,"Lois");//555-4567
```

- Implementación
  - Representación en:
    - Arreglo
    - Lista enlazada
    - Comparar
      - Espacio
        - O(n)
      - Tiempos de operaciones
        - O(n)
        - O(log n)

Introducción a Árboles

Árbol de Búsqueda Binario: BST

- Mejor Solución Árbol
  - Tiempos logarítmicos



- Un Árbol también permite implementar Conjunto Finito
  - Add
  - IsMember
  - Remove
- ¿Podemos implementar un conjunto infinito? ¿Una secuencia infinita?

¿Consultas?

Fin de la clase