introduccion.html

contenido.css

```
Tema 3 'PRO2' {

    [Conjuntos Diccionarios]

    < parejas de datos, sets y maps>

}
```

C++

introduccion.html

contenido.css

Tabla de ‘Contenidos’ {

01 Pairs

< Metodos, Aplicaciones >

02 Sets

< Metodos, Aplicaciones >

03 Maps

< Metodos, Aplicaciones>

}

C++

introducción.html

contenido.css

01 {

[Parejas <pair>]

< Ejemplo, Métodos y Aplicaciones >

}

C++

Como usamos las ‘pairs’{

```
int main() {
    pair<type1, type2> name;
}
```

}

Metodos clase '<pair>' {

first

x.first

< Se apunta al primer elemento de la pareja >

second

x.second

< Se apunta al segundo elemento de la pareja >

makepair()

makepair(val1,val2)

< Crea una pareja con los valores 1 y 2 >

}

Como usamos las ‘pairs’{

```
int main() {

    pair<type1, type2> name;
    name.first = x;
    name.second = z;

    # name = <x, z>
}
```

}

02 {

[Conjuntos <sets>]

< Ejemplo, Métodos y Aplicaciones >

}

introducción.html

contenido.css

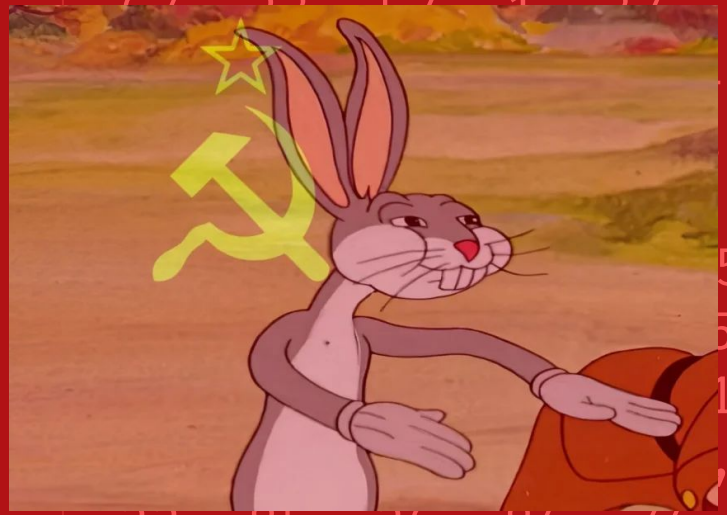
```
Situación ‘previa’ {
    67, 23, 57, 1, 37, 21, 4, 75, 95, 17, 45, 34,
    81, 13, 96, 84, 24, 61, 56, 84, 16, 57, 65,
    71, 27, 32, 84, 33, 41, 5, 77, 3, 27, 97, 38,
    82, 99, 95, 8, 20, 84, 92, 3, 89, 58, 77, 94,
    1, 49, 13, 56, 5, 53, 54, 87, 43, 63, 13, 7,
    65, 77, 61, 82, 95, 14, 3, 97, 28, 59, 54,
    73, 48, 96, 56, 51, 57, 74, 51, 15, 4, 63,
    16, 62, 53, 63, 5, 77, 50, 27, 40, 45, 53,
    22, 85, 26, 84, 74, 76, 56, 42

}
```

C++

introducción.html

contenido.css

```
Situación ‘previa’ {
    67, 23, 57, 1, 37, 21, 4, 75, 95, 17, 45, 34,
    22, 85, 26, 84, 74, 76, 56, 42
}
```



[MERGE SORT]

[REC. LINEAL de 250 lineas]

C++

introducción.html

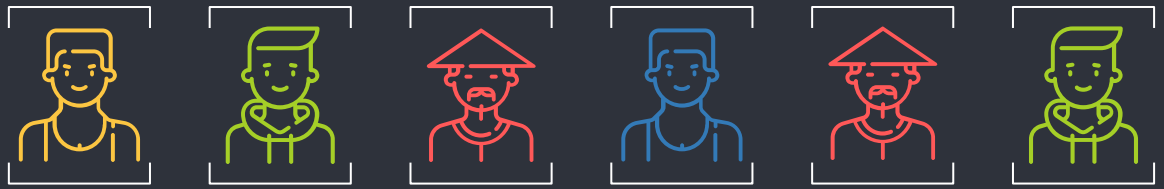
contenido.css

Que son los ‘sets’{

< Simulan un **CONJUNTO** de objetos **NO REPETIDOS** >

< Se ordenan, de manera ascendente, automaticamente >

< Los elementos **NO SE MODIFICAN**: o se añaden o se eliminan >



}

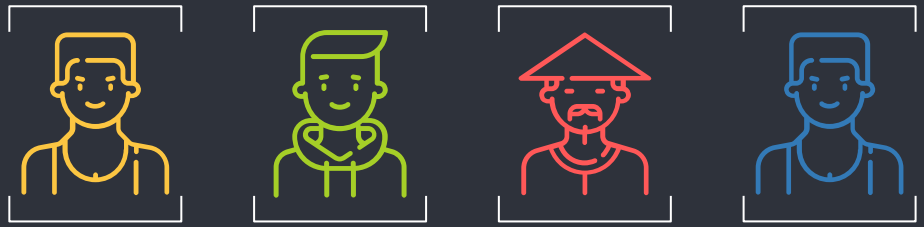
C++

Que son los ‘sets’{

< Simulan un **CONJUNTO** de objetos **NO REPETIDOS** >

< Se ordenan, de manera ascendente, automaticamente >

< Los elementos **NO SE MODIFICAN**: o se añaden o se eliminan >



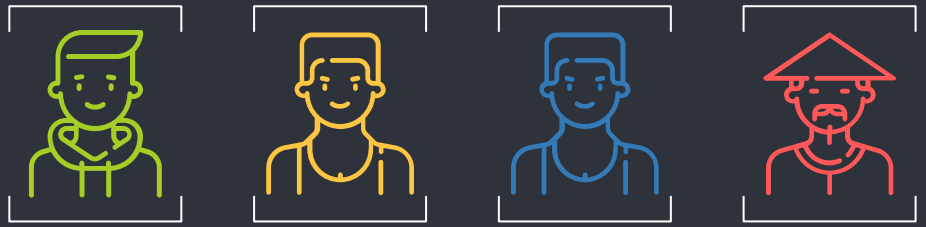
}

Que son los ‘sets’{

< Simulan un **CONJUNTO** de objetos **NO REPETIDOS** >

< Se ordenan, de manera ascendente, automaticamente >

< Los elementos **NO SE MODIFICAN**: o se añaden o se eliminan >



}

Como usamos los ‘sets’{

< Usamos la clase set de la librería estándar <set> >

```cpp
#include <set>

int main() {

    set<type> name;
    set<type>::iterator it;

}
```

}

Metodos clase ‘<set>’ {

.begin()

set.begin()

< Iterador que marca el inicio del set >

.end()

set.end()

< Iterador del elemento ficticio final >

.empty()

set.empty()

< Indica si el set está vacío >

.insert()

set.insert(val)

< Inserta el valor en el set >

.erase()

set.erase(it/val)

< Elimina el valor referenciado del set >

.find()

set.find(val)

< Indica si se encuentra el valor >

}

03 {

[Diccionarios <maps>]

< Ejemplo, Iteradores,
Métodos y Aplicaciones >

}

introducción.html

contenido.css

Situación ‘previa’ {

Buscar en diccionario/pokedex pokemon:

ALTARIA
Peso

RESHIRAM
Altura

VENUSAUR
Regiones

}

C++

Que son los ‘maps’{

< Simulan un **DICCIONARIO** de objetos **NO REPETIDOS** >

< Se ordenan, de manera ascendente, automaticamente >

< Los elementos **NO SE MODIFICAN**: o se añaden o se eliminan >

Key		Value
A	→	Antonio, Andres, Ander, Armando…
B	→	Bernardo, Bafunko, Bolantro…
C	→	Carlos, Camilo, Camello, Capo…

}

Como usamos los ‘map’{

< Usamos la clase map de la librería estándar <map> >

```cpp
#include <map>

int main() {

    map<type> name;
    map<type>::iterator it;

}
```

}

Metodos clase ‘<set>’ {

.begin()

map.begin()

< Iterador que marca el inicio del map >

.end()

map.end()

< Iterador del elemento ficticio final >

.empty()

map.empty()

< Indica si el map está vacío >

.insert()

map.insert(val)

< Inserta el valor en el map >

.erase()

map.erase(it/key)

< Elimina el valor referenciado del map >

.find()

map.find(val)

< Indica si se encuentra el valor >

}

Metodos clase ‘<set>’ {

ALTERNATIVAS

CONSTRUCTORES

```
map[key];
```

< Constructor para inicializar un valor con clave:key y valor 0 >

```
Map[key] = val;
```

< Constructor para inicializar un valor con clave:key y valor val >

}

introducción.html

contenido.css

```cpp
#include <map>

struct Pokemon {
      string name;
      string type;
}

int main() {

       map<int,Pokemon> Pokedex;

      // Inicialización Pokédex
      Pokedex[1] = {"Bulbasaur", "Hierba"};
      Pokedex[2] = {"Squirtle", "Agua"};
      Pokedex[3] = {"Charmander", "Fuego"};

      string name = "Pikachu";
      string ty = "Electrico";
      Pokedex.insert(make_pair(4, Pokemon{name, ty}));

}
```

C++

introducción.html

contenido.css

```cpp
#include <map>

struct Pokemon {
      string name;
      string type;
}

int main() {

       map<int,Pokemon> Pokedex;

      // Inicialización Pokédex
      Pokedex[1] = {"Bulbasaur", "Hierba"};
      Pokedex[2] = {"Squirtle", "Agua"};
      Pokedex[3] = {"Charmander", "Fuego"};

      string name = "Pikachu";
      string ty = "Electrico";
      Pokedex.insert(make_pair(4, Pokemon{name, ty}));


      map<int,Pokemon>::iterator it;
      for (it = Pokedex.begin(); it ≠ Pokedex.end(); ++it) {
           cout << it→first << " " << it→second.Name << " " << it→second.ty << endl;
      }

}
```

C++