ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

# Iteradores en árboles

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

# Recordatorio

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

# ¿Cómo implementar un iterador?

- Un iterador debe simular este recorrido, pero «por partes».

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

```cpp
auto it = tree.begin();
```

```cpp
visit(*it);
```

```cpp
++it;
```

```cpp
for (auto it = tree.begin(); it ≠ tree.end(); ++it) {
  visit(*it);
}
```

# Interfaz de iteradores

```cpp
template<class T>
class BinTree {
public:

  iterator begin();
  iterator end();

  class iterator {
  public:
    T & operator*() const;
    iterator & operator++();
    bool operator==(const iterator &other);
    bool operator≠(const iterator &other);

    ...
  };

};
```

# Implementación privada de iteradores

```cpp
class iterator {
public:
  T & operator*() const;
  iterator & operator++();
  bool operator==(const iterator &other);
  bool operator≠(const iterator &other);

private:
  iterator();
  iterator(const NodePointer &root);

  std::stack<NodePointer> st;
};
```

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

# Implementación privada de iteradores

```cpp
class iterator {
public:
  T & operator*() const;
  iterator & operator++();
  bool operator==(const iterator &other);
  bool operator≠(const iterator &other);

private:
  iterator() { }
  iterator(const NodePointer &root) {
    BinTree::descend_and_push(root, st);
  }

  std::stack<NodePointer> st;
};
```

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

# Implementación privada de iteradores

```cpp
class iterator {
public:
  T & operator*() const {
    assert(!st.empty());
    return st.top()→elem;
  }
  iterator & operator++();
  bool operator==(const iterator &other);
  bool operator≠(const iterator &other);

private:
  iterator();
  iterator(const NodePointer &root);

  std::stack<NodePointer> st;
};
```

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

7

# Implementación privada de iteradores

```cpp
class iterator {
public:
  T & operator*() const;

  iterator & operator++() {
    assert(!st.empty());
    NodePointer top = st.top();
    st.pop();
    BinTree::descend_and_push(top→right, st);
    return *this;
  }

  bool operator==(const iterator &other);
  bool operator≠(const iterator &other);

private:
  iterator();
  iterator(const NodePointer &root);

  std::stack<NodePointer> st;
};
```

```cpp
void inorder(NodePointer &node) {

  std::stack<NodePointer> st;
  descend_and_push(node, st);

  while (!st.empty()) {
    NodePointer x = st.top();
    visit(x→elem);
    st.pop();
    descend_and_push(x→right, st);
  }

}
```

# Creación de iteradores

```cpp
template<class T>
class BinTree {
public:

  iterator begin() {
    return iterator(root_node);
  }

  iterator end() {
    return iterator();
  }

  ...

};
```

# Ejemplo

```cpp
int main() {
  BinTree<int> tree {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};

  for (auto it = tree.begin(); it ≠ tree.end(); ++it) {
    cout << *it << " ";
  }

  return 0;
}
```

```
9 4 5 7 10 4 6
```

# Ejemplo

```cpp
int main() {
  BinTree<int> tree {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};

  for (int x: tree) {
    cout << x << " ";
  }

  return 0;
}
```

9 4 5 7 10 4 6

# Posibles extensiones

- Iteradores constantes: `cbegin()`, `cend()`, etc.

- Diferencia entre postincremento (`it++`) y preincremento (`++it`).

- Aplicación a `SetTree` y `MapTree`.