ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

Iteradores y listas enlazadas

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

Operaciones a implementar

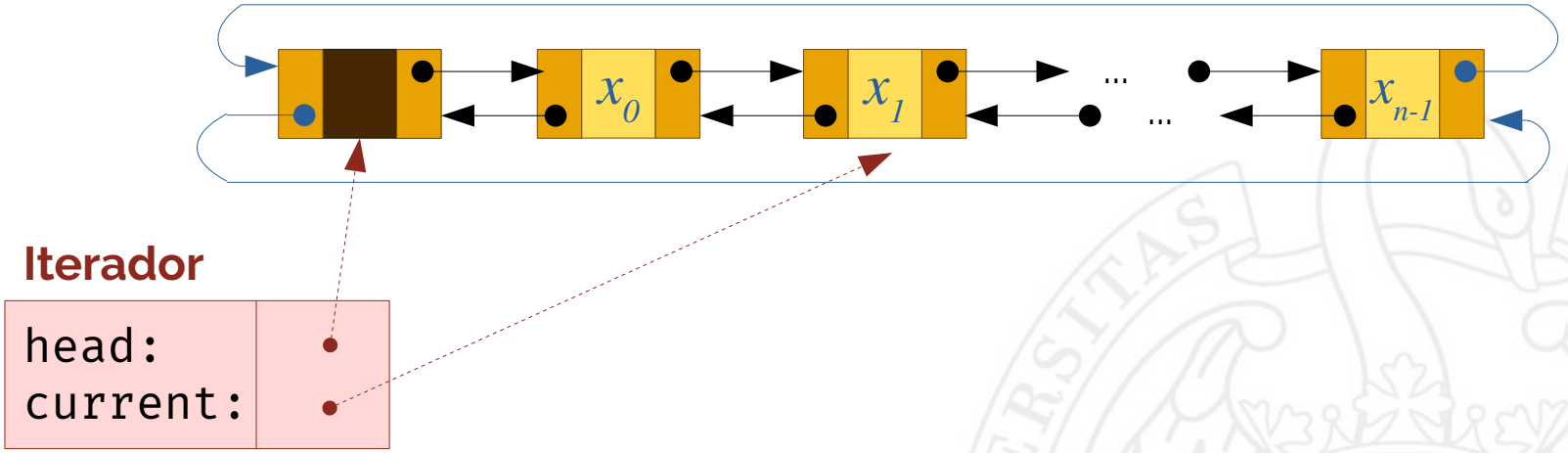
Para iteradores

- Obtener el elemento apuntado por el iterador (***elem***)
- Avanzar el iterador a la siguiente posición de la lista (***advance***)
- Igualdad entre dos iteradores (**==**)
  - Dos iteradores son iguales si recorren la misma lista y apuntan a la misma posición dentro de esta.

Para listas

- Obtener un iterador al principio de la lista (***begin***)
- Obtener un iterador al final de la lista (***end***)

Iteradores en listas doblemente enlazadas



- Un iterador contiene dos atributos:
  - El nodo del elemento apuntado por el iterador (`current`).
  - El nodo fantasma de la lista a la que el iterador pertenece (`head`).

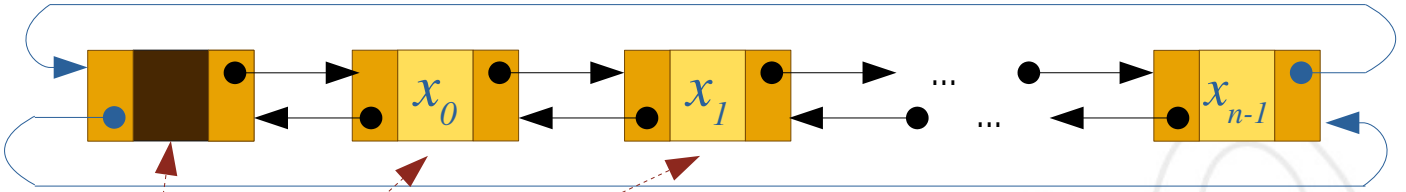
La clase ListLinkedDouble<T> :: iterator

```
template <typename T>
class ListLinkedDouble {
public:
  …
  class iterator {
  public:
    iterator(Node *head, Node *current): head(head), current(current) { }
    …

  private:
    Node *head;
    Node *current;
  };
  …
};
```

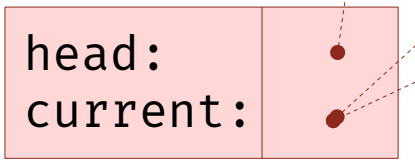
La clase ListLinkedDouble<T> :: iterator

```
template <typename T>
class ListLinkedDouble {
public:
  …
  class iterator {
  public:
    …
    void advance();
    T & elem();
    bool operator==(const iterator &other) const;
    bool operator!=(const iterator &other) const;
    …

  };
  …
};
```

Implementación de advance( )

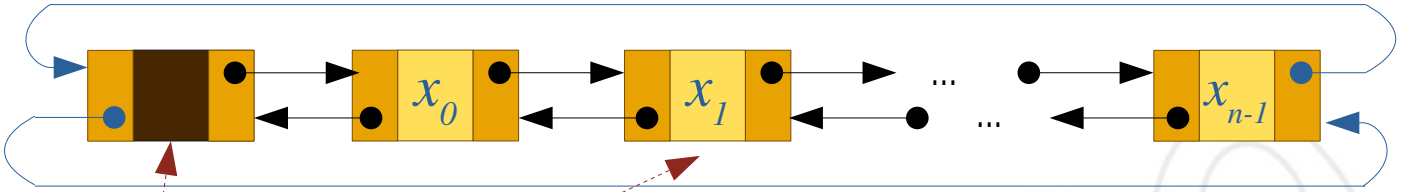


Iterador

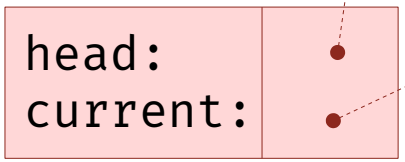



```
class iterator {
  public:
    void advance() {
      assert (current ≠ head);
      current = current→next;
    }
    …
};
```

Implementación de elem( )



Iterador




```
class iterator {
  public:
    T & elem() {
      assert (current ≠ head);
      return current→value;
    }
    …
};
```

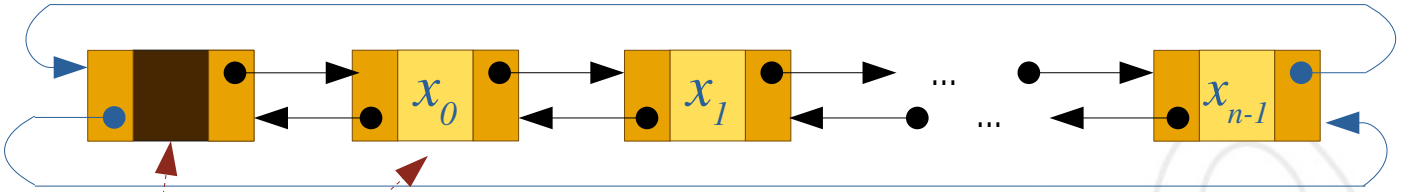
Implementación de los operadores == y !=

```
class iterator {
  public:
    bool operator==(const iterator &other) const {
      return (head == other.head) &&
             (current == other.current);
    }

    bool operator!=(const iterator &other) const {
      return !(*this == other);
    }

};
```

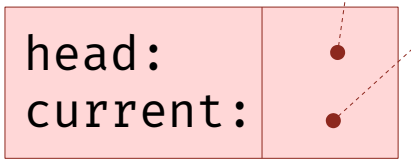
Creación de iteradores

```
template <typename T>
class ListLinkedDouble {
public:
  class iterator { … }
   ...
  iterator begin();
  iterator end();
   ...
};
```

Implementación de begin( )

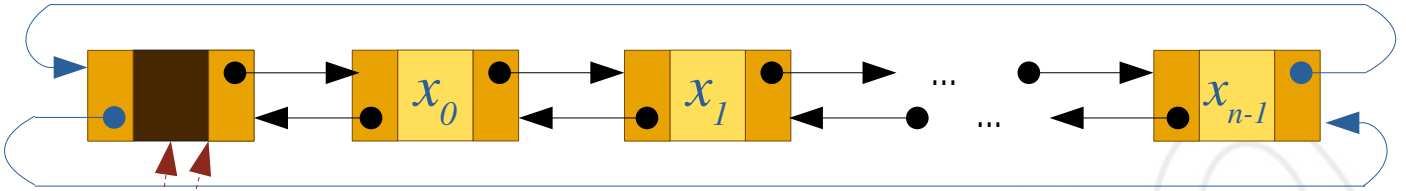


Iterador

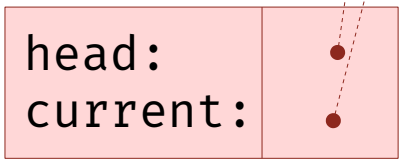



```
template <typename T>
class ListLinkedDouble {
   ...
  iterator begin() {
    return iterator(head, head→next);
  }
};
```

Implementación de end( )



Iterador




```
template <typename T>
class ListLinkedDouble {
   ...
  iterator end() {
    return iterator(head, head);
  }
};
```

Ejemplo

- Dada una lista de nombres de personas, imprimir aquellas que empiecen por un carácter pasado como parámetro:

```
void imprime_empieza_por(ListLinkedDouble<std::string> &l, char c) {
  for (ListLinkedDouble<std::string>::iterator it = l.begin();
       it =/= l.end();
       it.advance()) {

    if (it.elem()[0] == c) {
      std::cout << it.elem() << std::endl;
    }

  }
}
```

Ejemplo

- Modificar todos los elementos de una lista de enteros, multiplicándolos por uno pasado como parámetro.

```
void multiplicar_por(ListLinkedDouble<int> &l, int num) {
  for (ListLinkedDouble<int>::iterator it = l.begin();
       it ≠ l.end();
       it.advance()) {
    it.elem() = it.elem() * num;
  }
}
```

```
class iterator {
public:
  …
  void advance();
  T & elem();
  bool operator==(const iterator &other) const;
  bool operator!=(const iterator &other) const;
  …
};
```