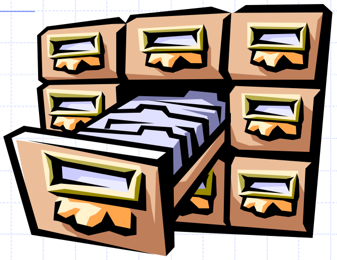
Maps



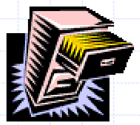
Maps

- A map models a searchable collection of key-value entries
- The main operations of a map are for searching, inserting, and deleting items
- Multiple entries with the same key are not allowed
- Applications:
  - address book
  - student-record database

Entry ADT

- An entry stores a key-value pair (k,v)
- Methods:
  - key(): return the associated key
  - value(): return the associated value
  - setKey(k): set the key to k
  - setValue(v): set the value to v

The Map ADT



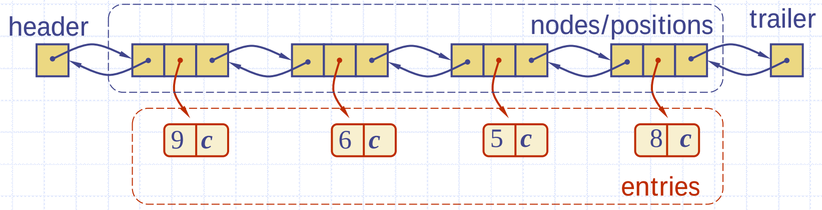
- find(k): if the map M has an entry with key k, return and iterator to it; else, return special iterator end
- put(k, v): if there is no entry with key k, insert entry (k, v), and otherwise set its value to v. Return an iterator to the new/modified entry
- erase(k): if the map M has an entry with key k, remove it from M
- size(), empty()
- begin(), end(): return iterators to beginning and end of M

Example

Operation	*Output*	*Map*
empty()	**true**	Ø
put(5,A)	[(5,A)]	(5,A)
put(7,B)	[(7,B)]	(5,A),(7,B)
put(2,C)	[(2,C)]	(5,A),(7,B),(2,C)
put(8,D)	[(8,D)]	(5,A),(7,B),(2,C),(8,D)
put(2,E)	[(2,E)]	(5,A),(7,B),(2,E),(8,D)
find(7)	[(7,B)]	(5,A),(7,B),(2,E),(8,D)
find(4)	end	(5,A),(7,B),(2,E),(8,D)
find(2)	[(2,E)]	(5,A),(7,B),(2,E),(8,D)
size()	4	(5,A),(7,B),(2,E),(8,D)
erase(5)	—	(7,B),(2,E),(8,D)
erase(2)	—	(7,B),(8,D)
find(2)	end	(7,B),(8,D)
empty()	**false**	(7,B),(8,D)

A Simple List-Based Map

- We can efficiently implement a map using an unsorted list
  - We store the items of the map in a list S (based on a doubly-linked list), in arbitrary order



The find Algorithm

```
Algorithm find(k):
    for each p in [S.begin(), S.end()) do
        if p→key() = k then
            return p
    return S.end() {there is no entry with key equal to k}
```

We use p→key() as a shortcut for (*p).key()

The put Algorithm

```
Algorithm put(k,v):
    for each p in [S.begin(), S.end()) do
        if p→key() = k then
            p→setValue(v)
            return p
    p = S.insertBack((k,v)) {there is no entry with key k}
    n = n + 1 {increment number of entries}
    return p
```

The erase Algorithm

```
Algorithm erase(k):
   for each p in [S.begin(), S.end()) do
      if p.key() = k then
         S.erase(p)
         n = n – 1      {decrement number of entries}
```

Performance of a List-Based Map

- Performance:
  - put takes $O(n)$ time since we need to determine whether it is already in the sequence
  - find and erase take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key
- The unsorted list implementation is effective only for maps of small size or for maps in which puts are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)