CSE221

Lecture 7:
List and Iterators

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Outline

- Array List (Vector)
- List
- Iterator

The Array List (Vector) ADT

- The Vector or Array List ADT extends the notion of array by storing a sequence of objects
- An element can be accessed, inserted or removed by specifying its index (number of elements preceding it)
- An exception is thrown if an incorrect index is given (e.g., a negative index)

- Main methods:
  - at(integer i): returns the element at index i without removing it
  - set(integer i, object o): replace the element at index i with o
  - insert(integer i, object o): insert a new element o to have index i
  - erase(integer i): removes element at index i
- Additional methods:
  - size()
  - empty()

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Applications of Array Lists

- Direct applications
  - Sorted collection of objects (elementary database)
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Array-based Implementation

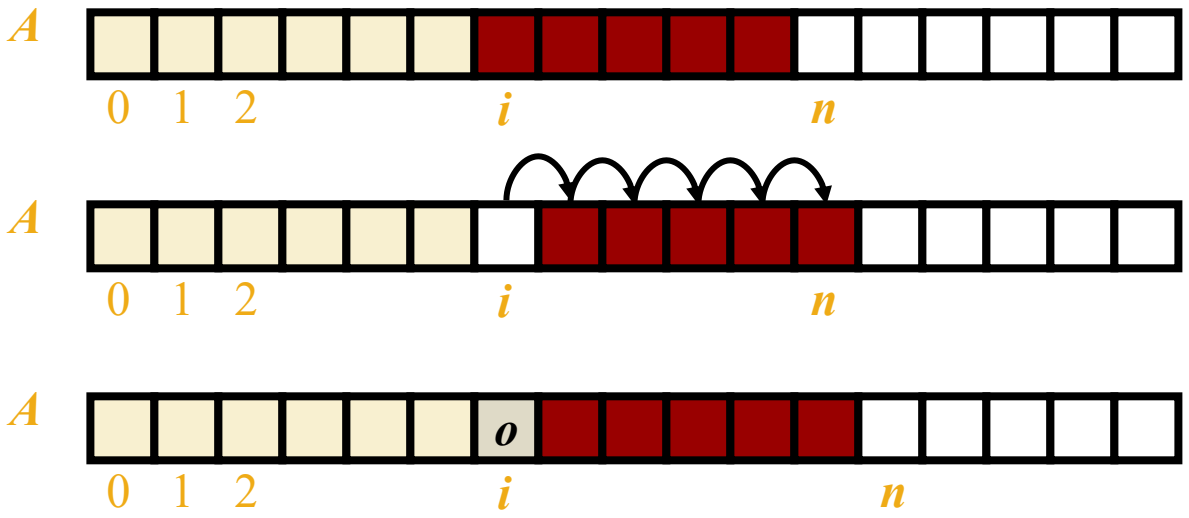
- Use an array A of size N
- A variable n keeps track of the size of the array list (number of elements stored)
- Operation $at(i)$ is implemented in $O(1)$ time by returning $A[i]$
- Operation $set(i,o)$ is implemented in $O(1)$ time by performing $A[i] = o$

© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Insertion

- In operation *insert*(i, o), we need to make room for the new element by shifting forward the $n - i$ elements $A[i], \ldots, A[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time

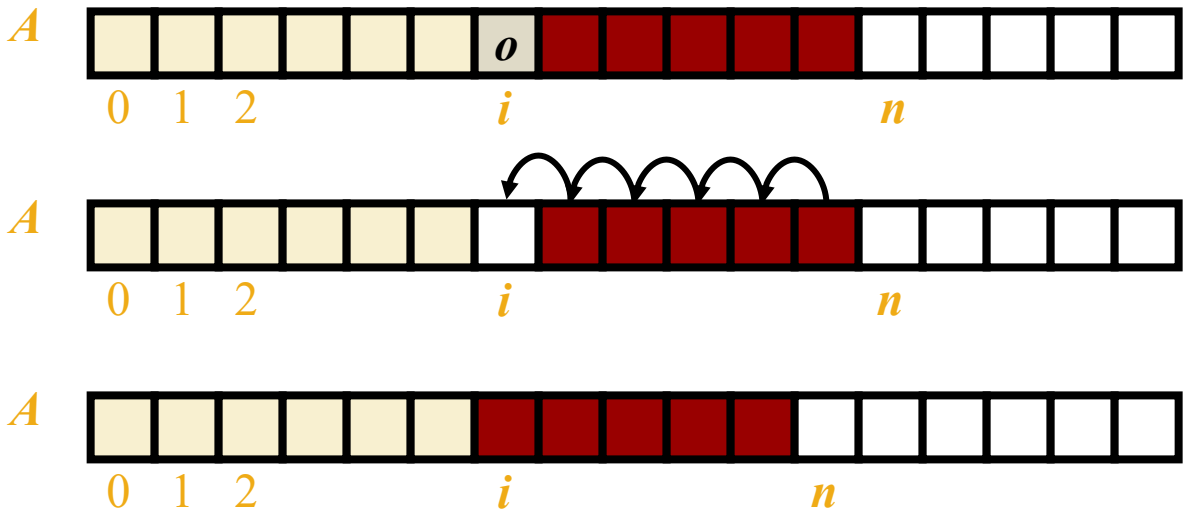


© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Element Removal

- In operation *erase*(i), we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[i + 1], \ldots, A[n - 1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Performance

- In the array based implementation of an array list:
  - The space used by the data structure is $O(n)$
  - *size*, *empty*, *at* and *set* run in $O(1)$ time
  - *insert* and *erase* run in $O(n)$ time in worst case
- If we use the array in a circular fashion, operations $insert(0, x)$ and $erase(0, x)$ run in $O(1)$ time
- In an *insert* operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one

© 2010 Goodrich, Tamassia

Growable Array-based Array List

- In an insert(o) operation (without an index), we always insert at the end
- When the array is full, we replace the array with a larger one
- How large should the new array be?
  - Incremental strategy: increase the size by a constant c
  - Doubling strategy: double the size

Algorithm ***insert(o)***

```
if t = S.length - 1 then
    A ← new array of
        size …
    for i = 0 to n–1 do
        A[i] ← S[i]
    S ← A
n ← n + 1
S[n–1] ← o
```

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Comparison of the Strategies

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of n insert(o) operations
- We assume that we start with an empty vector represented by an array of size 1
- We call amortized time of an insert operation the average time taken by an insert over the series of operations, i.e., $T(n)/n$

© 2010 Goodrich, Tamassia

Incremental Strategy Analysis

- We replace the array $k = n/c$ times
- The total time $T(n)$ of a series of n insert operations is proportional to

$$n + 1 + c+1 + 2c+1 + 3c+1 \ \ldots + (k-1)c+1 =$$

$$n + k + c(1 + 2 + 3 + \ldots + k-1) =$$

$$n + k + ck(k - 1)/2$$

- Since c is a constant, $T(n)$ is $O(n + k^2)$, i.e., $O(n^2)$
- The amortized time of an insert operation is $O(n)$

© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

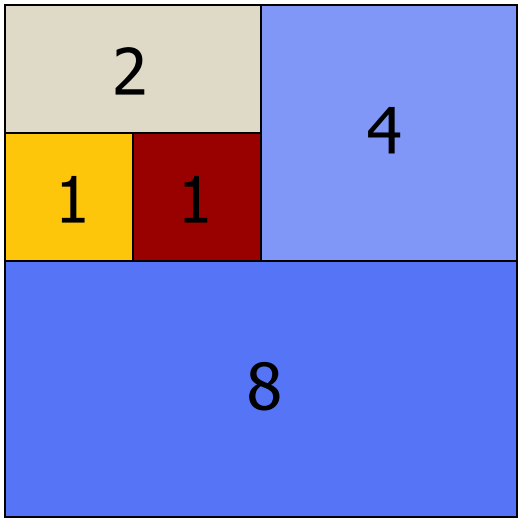
Doubling Strategy Analysis

- We replace the array $k = \log_2 n$ times
- The total time $T(n)$ of a series of n insert operations is proportional to

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$
$$n + 2^{k+1} - 1 =$$
$$3n - 1$$

- $T(n)$ is $O(n)$
- The amortized time of an insert operation is $O(1)$

geometric series



© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Outline

- Array List (Vector)
- List
- Iterator

Position ADT

- The Position ADT models the notion of place within a data structure where a single object is stored
- It gives a unified view of diverse ways of storing data, such as
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object p.element(): returns the element at position
  - In C++ it is convenient to implement this as *p

© 2010 Goodrich, Tamassia

Node List ADT

- The Node List ADT models a sequence of positions storing arbitrary objects
- It establishes a before/after relation between positions
- Generic methods:
  - size(), empty()

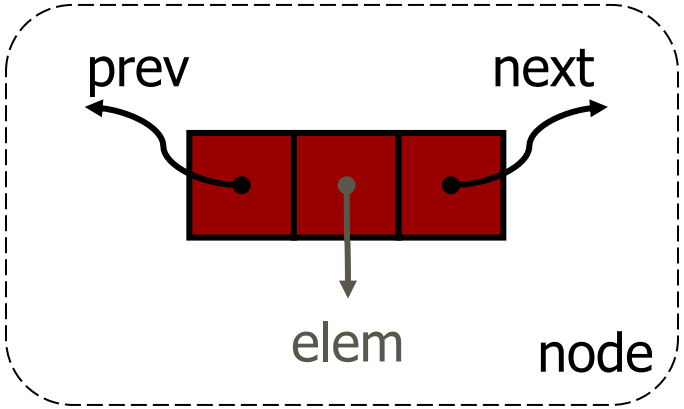
- Iterators:
  - begin(), end()
- Update methods:
  - insertFront(e), insertBack(e)
  - removeFront(), removeBack()
- Iterator-based update:
  - insert(p, e)
  - remove(p)

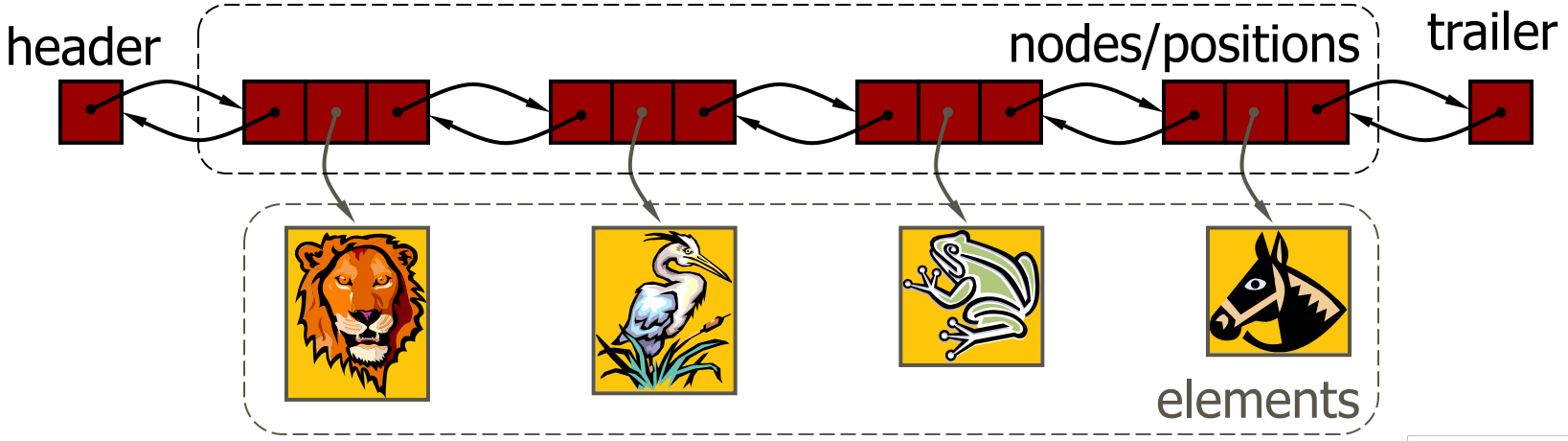
© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Doubly Linked List

- A doubly linked list provides a natural implementation of the Node List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes



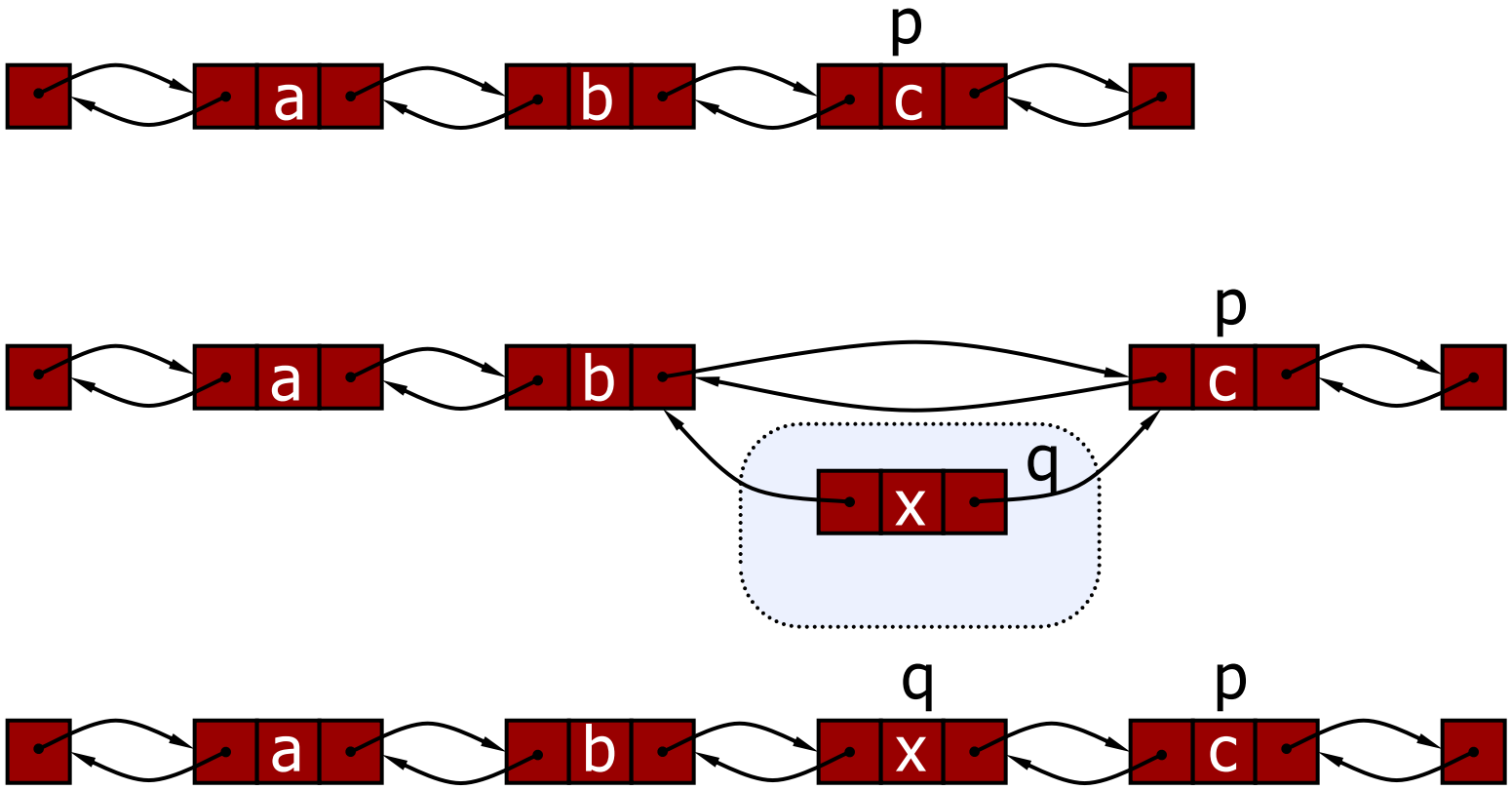


© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Insertion

- We visualize operation insert(p, x), which inserts x before p



© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Insertion Algorithm

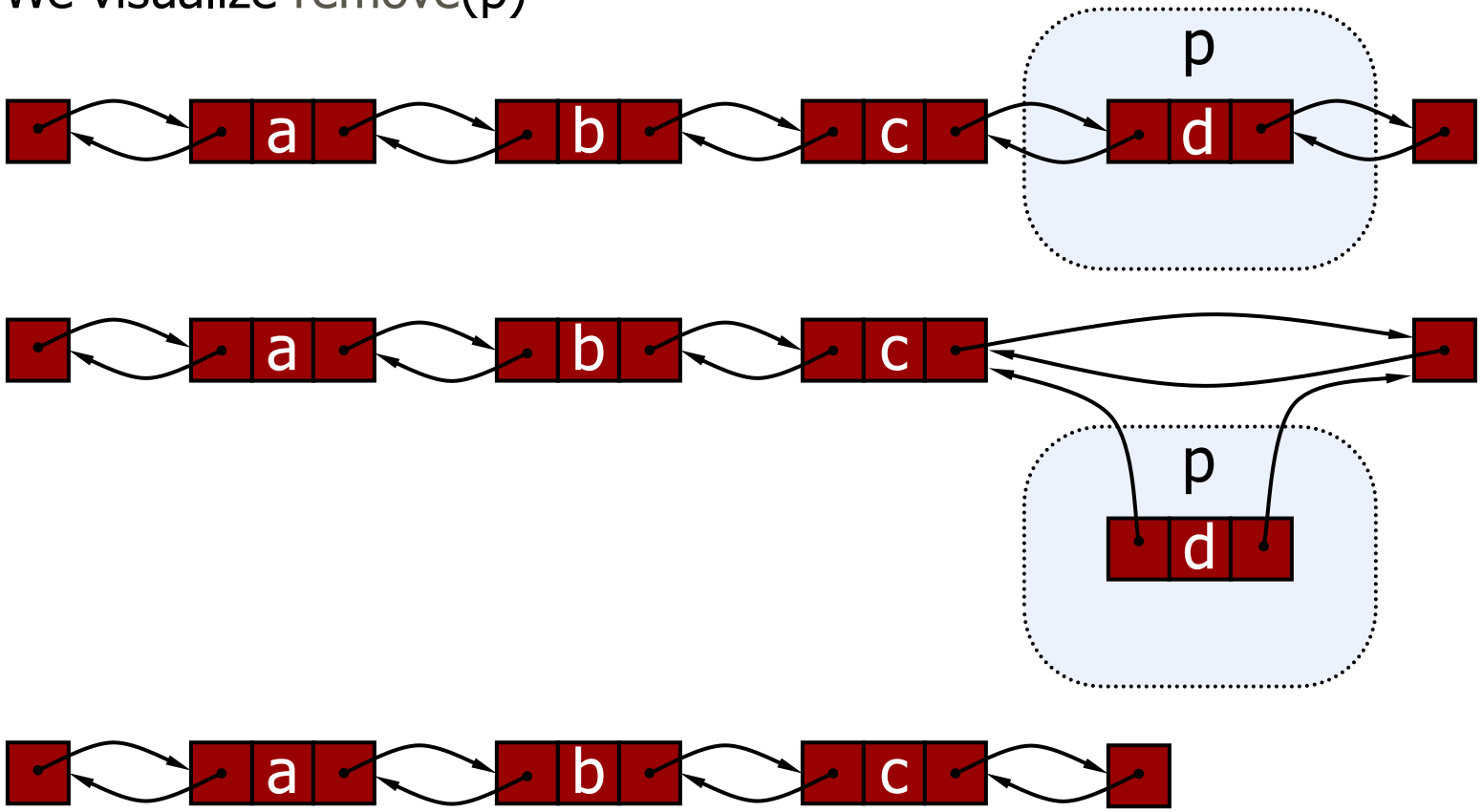
Algorithm insert(p, e): {insert e before p}

```
Create a new node v
v->element = e
u = p->prev
v->next = p;  p->prev = v  {link in v before p}
v->prev = u;  u->next = v  {link in v after u}
```

© 2010 Goodrich, Tamassia

Deletion

- We visualize remove(p)



© 2010 Goodrich, Tamassia

Deletion Algorithm

Algorithm remove(p):

```
u = p->prev
w = p->next
u->next = w {linking out p}
w->prev = u
```

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Performance

- In the implementation of the List ADT by means of a doubly linked list
  - The space used by a list with n elements is $O(n)$
  - The space used by each position of the list is $O(1)$
  - All the operations of the List ADT run in $O(1)$ time
  - Operation element() of the Position ADT runs in $O(1)$ time

© 2010 Goodrich, Tamassia

Outline

- Array List (Vector)
- List
- Iterator

Containers and Iterators

- An iterator abstracts the process of scanning through a collection of elements
- A container is an abstract data structure that supports element access through iterators
  - begin(): returns an iterator to the first element
  - end(): return an iterator to an imaginary position just after the last element
- An iterator behaves like a pointer to an element
  - *p: returns the element referenced by this iterator
  - ++p: advances to the next element
- Extends the concept of position by adding a traversal capability

© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Containers

- Data structures that support iterators are called containers (in C++)
- Examples include Stack, Queue, Vector, List
- Various notions of iterator:
  - (standard) iterator: allows read-write access to elements
  - const iterator: provides read-only access to elements
  - bidirectional iterator: supports both ++p and --p
  - random-access iterator: supports both p+i and p-i

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Iterating through a Container

- Let C be a container and p be an iterator for C

```
for (p = C.begin(); p != C.end(); ++p)
    loop_body
```

- Example: (with an STL vector)

```
typedef vector<int>::iterator Iterator;
int sum = 0;
for (Iterator p = V.begin(); p != V.end(); ++p)
sum += *p;
return sum;
```

© 2010 Goodrich, Tamassia

Implementing Iterators

- Array-based
  - array A of the n elements
  - index i that keeps track of the cursor
  - begin() = 0
  - end() = n (index following the last element)
- Linked list-based
  - doubly-linked list L storing the elements, with sentinels for header and trailer
  - pointer to node containing the current element
  - begin() = front node
  - end() = trailer node (just after last node)

© 2010 Goodrich, Tamassia

STL Iterators in C++

- Each STL container type C supports iterators:
  - C::iterator – read/write iterator type
  - C::const_iterator – read-only iterator type
  - C.begin(), C.end() – return start/end iterators
- This iterator-based operators and methods:
  - *p: access current element
  - ++p, --p: advance to next/previous element
  - C.assign(p, q): replace C with contents referenced by the iterator range [p, q) (from p up to, but not including, q)
  - insert(p, e): insert e prior to position p
  - erase(p): remove element at position p
  - erase(p, q): remove elements in the iterator range [p, q)

© 2010 Goodrich, Tamassia

Sequence ADT

- The Sequence ADT is the union of the Array List and Node List ADTs
- Elements accessed by
  - Index, or
  - Position
- Generic methods:
  - size(), empty()
- ArrayList-based methods:
  - at(i), set(i, o), insert(i, o), erase(i)
- List-based methods:
  - begin(), end()
  - insertFront(o), insertBack(o)
  - eraseFront(), eraseBack()
  - insert (p, o), erase(p)
- Bridge methods:
  - atIndex(i), indexOf(p)

© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Applications of Sequences

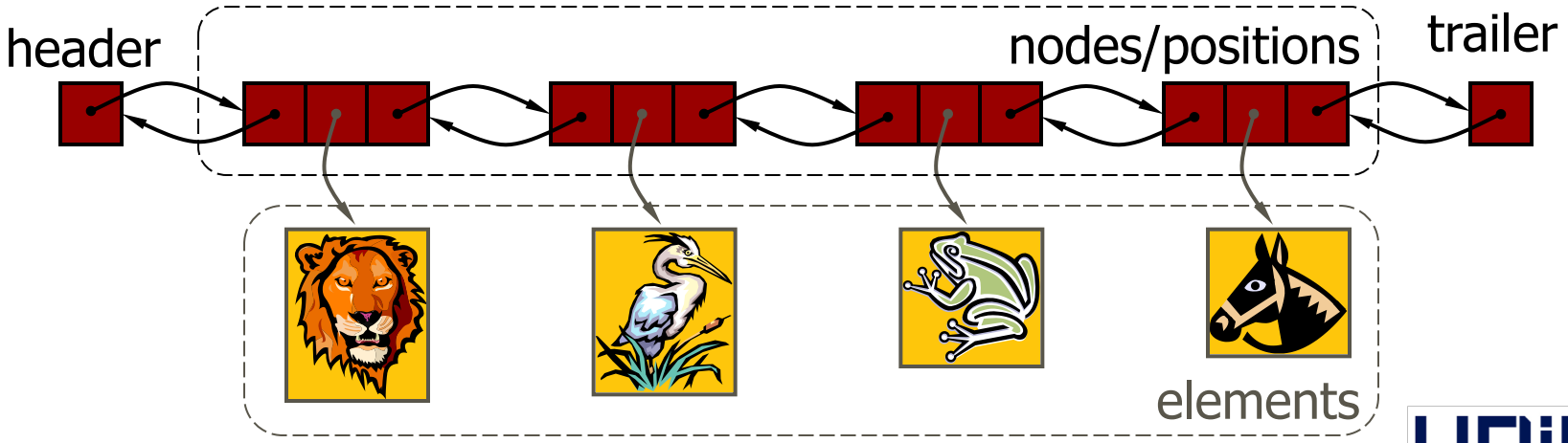
- The Sequence ADT is a basic, general-purpose, data structure for storing an ordered collection of elements
- Direct applications:
  - Generic replacement for stack, queue, vector, or list
  - small database (e.g., address book)
- Indirect applications:
  - Building block of more complex data structures

© 2010 Goodrich, Tamassia

Linked List Implementation

- A doubly linked list provides a reasonable implementation of the Sequence ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes

- Position-based methods run in constant time
- Index-based methods require searching from header or trailer while keeping track of indices; hence, run in linear time

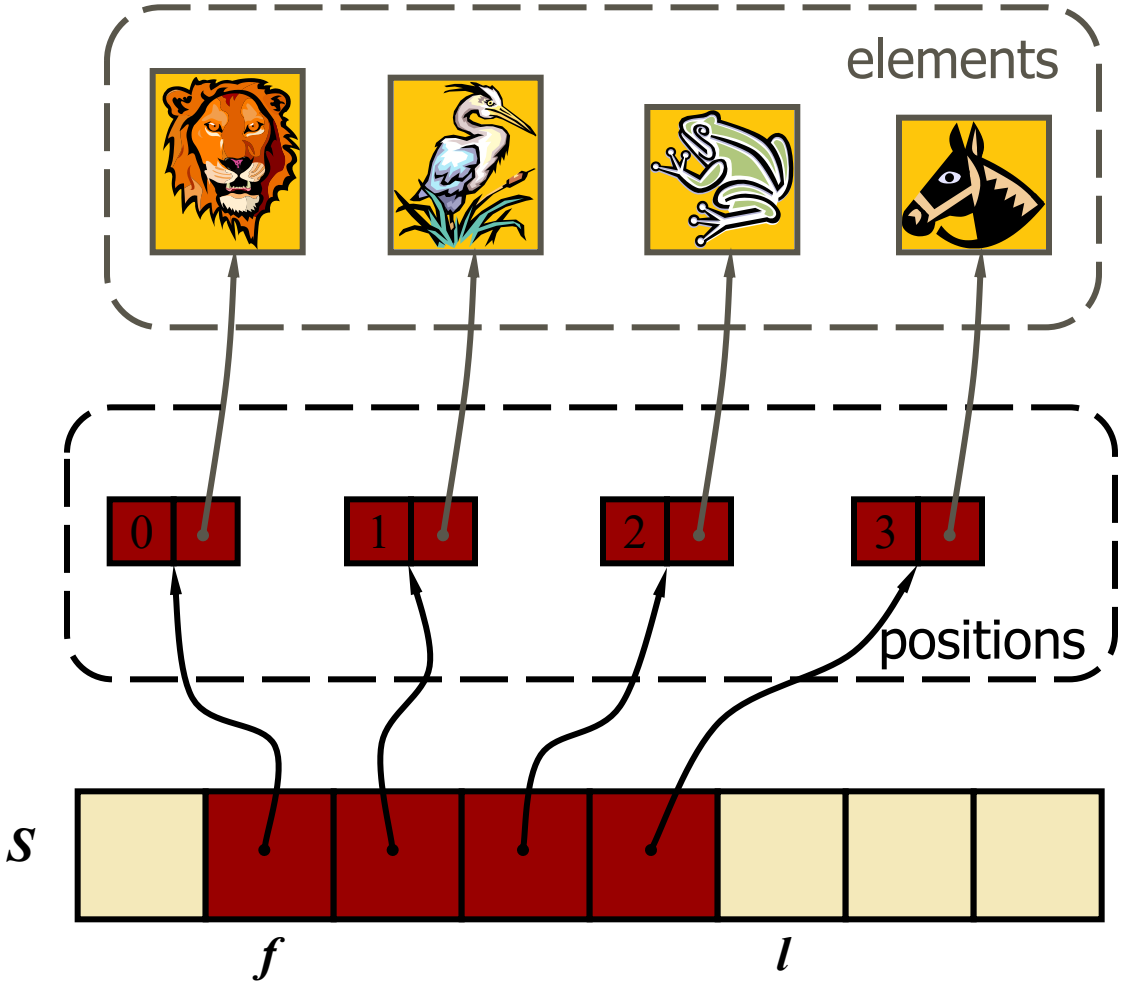


© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Array-based Implementation

- We use a circular array storing positions
- A position object stores:
  - Element
  - Index
- Indices f and l keep track of first and last positions



© 2010 Goodrich, Tamassia

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Comparing Sequence Implementations

Operation	Array	List
size, empty	1	1
atIndex, indexOf, at	1	n
begin, end	1	1
set(p,e)	1	1
set(i,e)	1	n
insert(i,e), erase(i)	n	n
insertBack, eraseBack	1	1
insertFront, eraseFront	n	1
insert(p,e), erase(p)	n	1

© 2010 Goodrich, Tamassia

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Bubble Sort

```
void bubbleSort1(Sequence& S) {                    // bubble-sort by indices
  int n = S.size();
  for (int i = 0; i < n; i++) {                    // i-th pass
    for (int j = 1; j < n−i; j++) {
      Sequence::Iterator prec = S.atIndex(j−1);    // predecessor
      Sequence::Iterator succ = S.atIndex(j);      // successor
      if (*prec > *succ) {                         // swap if out of order
        int tmp = *prec; *prec = *succ; *succ = tmp;
      }
    }
  }
}
```

Bubble Sort

```
void bubbleSort2(Sequence& S) {                    // bubble-sort by positions
  int n = S.size();
  for (int i = 0; i < n; i++) {                    // i-th pass
    Sequence::Iterator prec = S.begin();           // predecessor
    for (int j = 1; j < n−i; j++) {
      Sequence::Iterator succ = prec;
      ++succ;                                      // successor
      if (*prec > *succ) {                         // swap if out of order
        int tmp = *prec; *prec = *succ; *succ = tmp;
      }
      ++prec;                                      // advance predecessor
    }
  }
}
```

Questions?

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY