CMSC 341
Lecture 6

Announcements

Doubly-Linked Lists

)ption: add pointer to previous node

ssues

- doubles number of pointers
- allows immediate (O(1)) access to previous node

Circular Linked Lists

- Configurations
  - header in circle
  - header before circle
  - no header

Doubly-Linked Circular Linked List

Λay or may not have a header

Jseful for applications requiring fast access to head and tail

Multilist

Used to hold elements which belong to two different types of lists

Sparse storage of 2D array

Vector Implementation

```
emplate <class Object>
oid List<Object>::insert(const Object &x,
     const ListItr<Object> &p) {
 if (!p.isPastEnd()) {
     for (i=this.last; i > p.current+1; i--)
           this.nodes[i+1] = this.nodes[i];
     this.nodes[p.current+1] = x;
     }
```

Cursor Implementation

inked list look&feel

- data stored as collection of nodes: data and next
- nodes allocated and deallocated

ithout dynamic memory allocation

asic concepts

- have node pool of all nodes, keep list of free ones
- use pool indices in place of pointers; 0 == NULL

Cursor List Class

```
emplate <class Object>
lass List {
  List();
  // same old list public interface
ublic:
  struct CursorNode {
      CursorNode() : next() {}
  private:
      CursorNode(const Object &theElement, int n) :
  element(theElement), next(n) {}
      Object element;
      int next;
      friend class List<Object>;
      friend class ListItr<Object>;
  };
```

Cursor List Class (cont.)

```
rivate:
  int header;
  static vector<CursorNode> cursorSpace;
  static void initializeCursorSpace();
  static int alloc();
  static void free (int p);
  friend class ListItr<Object>;
;
```

Cursor Initialization

```
emplate <class Object>
oid List<Object>::initializeCursorSpace() {
 static int p cursorSpaceIsInitialized = false;
 if (!cursorSpaceIsInitialized) {
     cursorSpace.resize(100);
     for(int i=0; i < cursorSpace.size(); i++)
           cursorSpace[i].next = i+1;
     cursorSpace[cursorSpace.size()-1].next = 0;
     cursorSpaceIsInitialized = true;
     }
```

cursorSpace

slot	element	next
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Cursor Allocation

```
emplate <class Object>
oid List<Object>::alloc() {
 int p = cursorSpace[0].next;
 cursorSpace[0].next = cursorSpace[p].next;
 return p;

emplate <class Object>
oid List<Object>::free(int p) {
 cursorSpace[p].next = cursorSpace[0].next;
  cursorSpace[0].next = p;
```

Cursor Implementation (cont.)

```
emplate <class Object>
istItr<Object> List<Object>::find(const Object
 &x) const {
 int itr = cursorSpace[header].next;
 while (itr!=0 && cursorSpace[itr].element != x)
     itr = cursorSpace[itr].next;
 return ListItr<Object>(itr);
```

Cursor Implementation (cont.)

```
emplate <class Object>
oid List<Object>::insert(const Object &x,
     const ListItr<Object> &p) {
 if (!p.isPastEnd()) {
     int pos = p.current;
     int tmp = alloc();
     cursorSpace[tmp] =
           CursorNode(x, cursorSpace[pos].next);
     cursorSpace[pos].next = tmp;
     }
```

Cursor Implementation (cont.)

```
emplate <class Object>
oid List<Object>::insert(const Object &x) {
 ListItr<Object> p = findPrevious(x);
 int pos= p.current;

 if (cursorSpace[pos].next != 0) {
     int tmp = cursorSpace[pos].next;
     cursorSpace[pos].next =
                       cursorSpace[tmp].next;
     free (tmp);
     }
```

Comparing Performance

	Linear	S Linked	D Linked	Cursor
constructor	O(1)	O(1)	O(1)	O(1)
ind	O(n)	O(n)	O(n)	O(n)
indPrev	O(n)	O(n)	O(n)	O(n)
nsert	O(n)	O(1)	O(1)	O(1)
emove	O(n)	O(n)	O(n)	O(n)
nakeEmpty	O(1)	O(n)	O(n)	O(n)