CS 261
Linked Lists
List Stack
by
Tim Budd
Ron Metoyer
Sinisa Todorovic
Dynamic Array -- Problems

• Data kept in a single large block of memory

• Often more memory used than necessary
  – especially when repeatedly growing and shrinking the dynamic array
Linked List

• A good alternative

• The memory use is always proportional to the number of elements in the collection
Characteristics of Linked Lists

- Elements are held in objects called **Links**
- Links are 1-1 with elements, allocated and released as necessary
- Each link points to next link in sequence, sometimes to previous link
Single and Double Linked Lists

Each link points to
only next link
next and previous links
in the sequence
Elements of Linked Lists

- **Sentinel** -- special link for start or end
- Points to
  - first or last link only (single linked list)
  - first and last link (double linked list)
Link Structure

```c
struct Link {
    TYPE value;
    struct Link *next;
};
```
List Stack

• Sentinel points to the first element
• Sentinel points to null if stack empty
• Add or remove elements only from front
• Allow only singly linked list
• Can access only first element
List Stack

stk

sentinel
next:

val: 2
next:

val: 7
next:

val: 4
next: null
Implementation of List Stack

```c
struct Link {
    TYPE  value;
    struct Link * next;
};

struct ListStack {
    struct Link * sentinel;
};
```

How to initialize List Stack?
void InitStack(struct ListStack * stk) {
    struct Link *sentinel = (struct Link *)malloc(sizeof(struct Link));
    assert(sentinel != 0);
    sentinel->next = 0;
    stk->sentinel = sentinel;
}

stk

sentinel

next
Push List Stack: 3 Steps

allocate new link

stk

sentinel =

val: 2
next:

val: 5
next: null

val: 7
next:

val: 4
next: null
Push List Stack: 3 Steps

1. Sentinel
2. Add the element to top
3. Link the element
void pushStack (struct listStack *stk, TYPE val){
    struct Link * new =
        (struct Link *) malloc(sizeof(struct Link));
    assert (new != 0);
    new->value = val;
}

allocate new link
void pushStack (struct listStack *stk, TYPE val) {
    struct Link * new =
        (struct Link *) malloc(sizeof(struct Link));
    assert (new != 0);
    new->value = val;
    new->next = stk->sentinel->next;
    stk->sentinel->next = new;
}

Push List Stack: 3 Steps

1. Create a new link using malloc
2. Assign the value to the new link
3. Set the next pointer of the new link to the current top of the stack
4. Move the current top of the stack to the new link
void PopStack (struct ListStack *stk) {
    struct Link * lnk = stk->sentinel->next;
    assert (!lnk);
    stk->sentinel->next = lnk->next;
    free(lnk);
}
topStack, pushStack, isEmpty...

• Should be done on your own
• Worksheet 17
# List Stack vs. Dyn. Array Stack

<table>
<thead>
<tr>
<th></th>
<th>List</th>
<th>Dyn. Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pushStack</strong></td>
<td>$O(1)$</td>
<td>$O(1)+$</td>
</tr>
<tr>
<td><strong>popStack</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>topStack</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>