

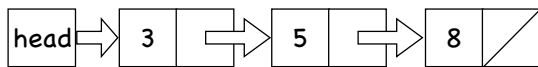
Linked Lists, Queues, and Stacks

CSE 130: Introduction to C Programming
Fall 2004

Linked Lists

- ◀ Dynamic data structure
 - ◀ Size is not fixed at compile time
- ◀ Each element of a linked list:
 - ◀ holds a value
 - ◀ points to the next list element

Linked List Illustration



List Operations

- 1 Create list
- 2 Destroy list
- 3 Retrieve value at specified position
- 4 Insert value at specified position
- 5 Delete value at specified position
- 6 Get number of elements in list

Sample Code

```
struct listNode
{
    int value;
    listNode *next;
};

static listNode *head; /* points to first list item */
static int numItems; /* number of list elements */
```

List Creation

```
void initializeList()
{
    head = NULL;
    numItems = 0;
}
```

List Deletion

```
void deleteList()
{
    listNode *temp = NULL;
    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
    numItems = 0;
}
```

List Retrieval

```
int retrieve (int pos)
{ /* Assumption: pos < list length */
    listNode *ptr = head;
    int i;

    for (i = 0; i < pos; i++)
        ptr = ptr->next;

    return ptr->value;
}
```

Inserting Values

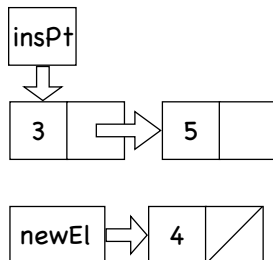
- Two steps:
 - Locate insertion point
 - Insert new list element
- Step two requires some care!

Locating The Insertion Point

- If the list is sorted, this requires a loop
- We want a pointer to the node BEFORE the insertion point

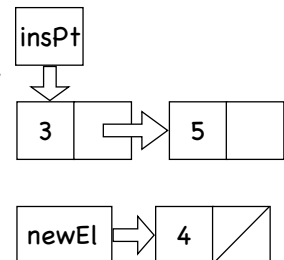
Inserting The New Element

- insPt points to the node BEFORE the insertion point
- newEl points to the new list element



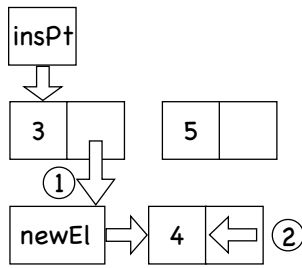
Order Matters!

- insPt->next is our only reference to the rest of the list
- If we change that first, we lose the rest of the list!



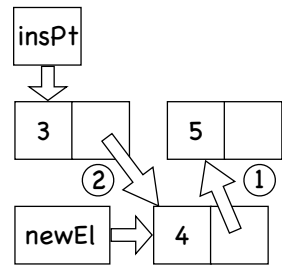
Algorithm: First Try

```
insPt->next = newEl;
newEl->next = insPt->next;
```



Algorithm: Second Try

```
newEl->next = insPt->next;
insPt->next = newEl;
```



Algorithm Summary

- 1 Locate insertion point
- 2 Set next pointer of new element
- 3 Insert new element

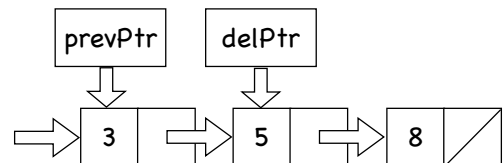
Removing An Element

- Similar to inserting an element
- Three steps:
 - Locate the element to be removed
 - Redirect pointers
 - Delete the selected element

Finding The Removal Point

- Depending on the type of list, this may require a loop
- We need two pointers:
 - delPtr – points to node to remove
 - prevPtr – points to node BEFORE delPtr

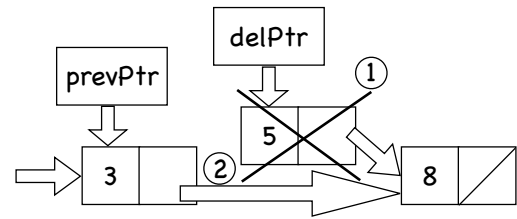
Pointers for Removal



Removal Steps

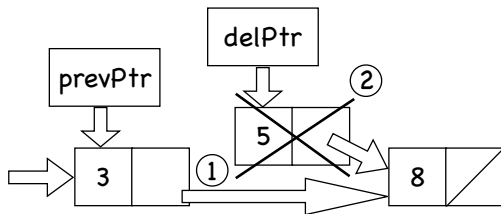
- To remove a list element, we need to:
 - Set the preceding element to point to the following element of the list
 - Delete the former list element
- As with list insertion, order matters!

Algorithm: First Try



```
delPtr->next = null;  
delete delPtr;  
prevPtr->next = delPtr->next;
```

Algorithm: Second Try



```
prevPtr->next = delPtr->next;  
delPtr->next = null;  
delete delPtr;
```

Algorithm Summary

- 1 Locate element to be removed
- 2 Re-route pointers around element
- 3 Perform cleanup
- 4 Delete element

Insertion Summary

- Find insertion point
- Set the new element to point to the rest of the list
- Insert the new element

Removal Summary

- Find removal point
 - Use two pointers
- Re-route pointers around element to be removed
- Clean up and delete the unwanted element

List Variations

- Doubly-linked list
 - Each node also contains a pointer to the preceding list node
- Circular linked lists
 - Last node points to first node, not NULL
 - One pointer (to the tail) stores entire list

Queues

- first-in, first-out (FIFO) data structure
- Real-world examples:
 - grocery checkout, on/off-ramp
- Programming examples
 - print jobs, processes running on a CPU

Queue Operations

- 1 Create queue
- 2 Destroy queue
- 3 Enqueue (insert element at tail)
- 4 Dequeue (remove element at head)
- 5 Get number of elements

Straight Queues

- (as opposed to circular queues)
- Maintain two pointers
 - one to head of queue (for removal)
 - one to tail of queue (for insertion)

Queue Insertion

- Special case: empty queue
 - set head and tail to point to new element
- Otherwise:
 - 1 `newNode->next = tail->next; /*or NULL*/`
 - 2 `tail->next = newNode;`

Queue Removal

- Special case: only one element in queue
 - set tail to point to NULL
- Otherwise:
 - 1 `newPtr = head;`
 - 2 `head = head->next;`

Circular Queues

- ◁ Last node points to first node
- ◁ Can represent a circular queue with one pointer (to tail)
- ◁ Head of queue is just tail->next

Stacks

- ◁ last-in, first-out (LIFO) data structure
- ◁ Real-world example:
 - ◁ stack of books
- ◁ Programming examples:
 - ◁ activation stack (for function calls)

Stack Operations

- 1 Create stack
- 2 Destroy stack
- 3 Push (insert object)
- 4 Pop (remove object)
- 5 Peek (view object on top of stack)
- 6 Get number of elements

Stack Insertion (Push)

- ◁ Very easy – just insert at the head:

```
newNode->next = head;  
head = newNode;
```

Stack Deletion (Pop)

- ◁ Just as easy as insertion

```
oldNode = head;  
head = head->next;
```

Array-Based Implementations

- ◁ Stacks and queues are also easy to implement using arrays
- ◁ Track indices of stack top, or queue head and tail
- ◁ Problem is that array is still of fixed size