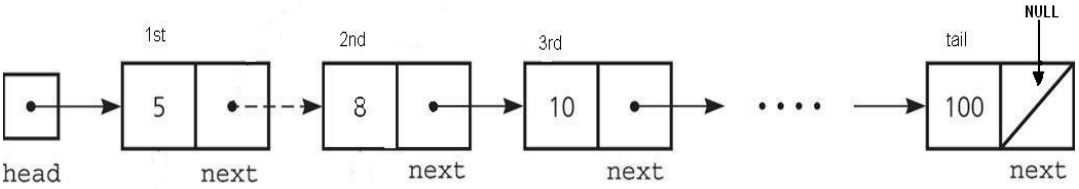
Linked List Review

1. Graphical representation:

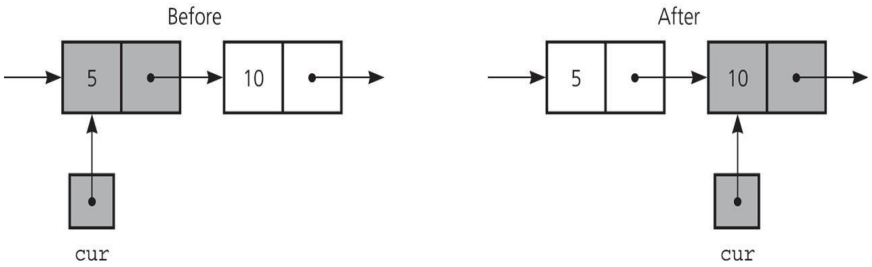


2. Store data one item per node. Node is defined as:

```
struct Node {
        int item
        Node *next;
}; //end struct
```

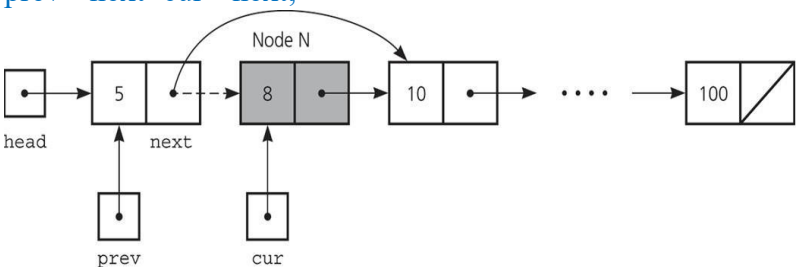
3. head of the list
4. Reference a node member with the -> operator: p->item
5. A traverse operation visits each node in the linked list
   a. A pointer variable cur keeps track of the current node

```
for (Node *cur = head; cur != NULL; cur = cur->next)
        cout << cur->item << endl;
```



6. Deleting a Specific Node
   a. Deleting an interior node

```
prev->next=cur->next;
```

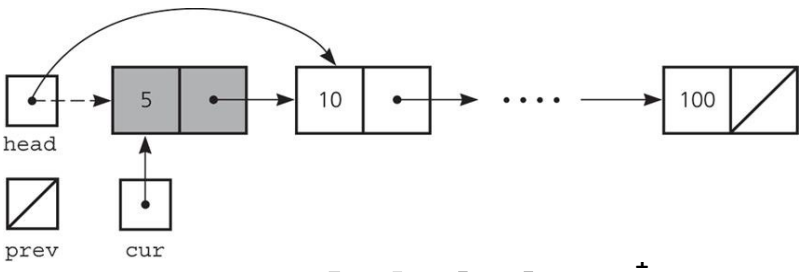


   b. Deleting the first node

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

   c. Return deleted node to system

```
cur->next = NULL; delete cur;
cur = NULL;
```

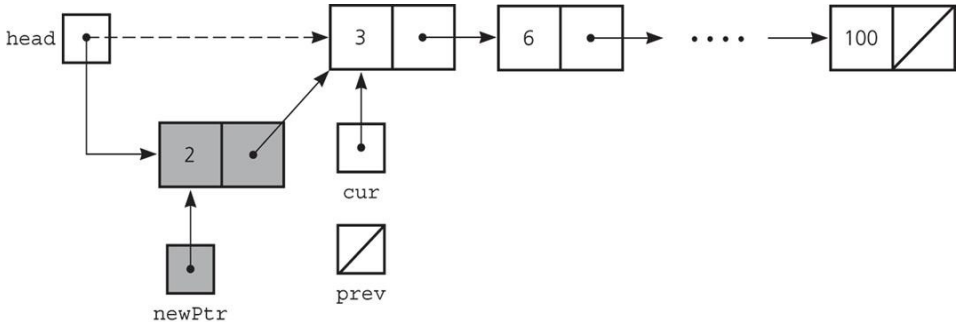


7. Inserting a node into a specific position of a linked list
    a. First step is to create a node with the new operator, put the data in the new node

```
newPtr = new Node;
newPtr->item = newData;
newPtr->next  = NULL;
```

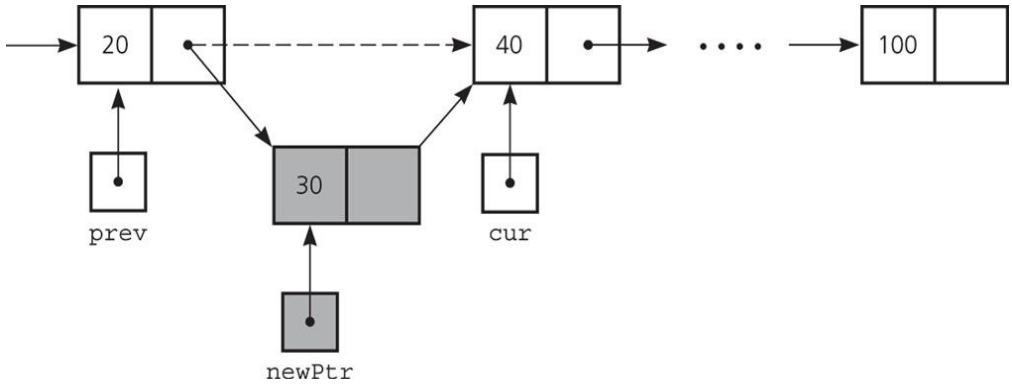
    b. To insert a node at the beginning of a linked list

```
newPtr->next = head;
head = newPtr;
```



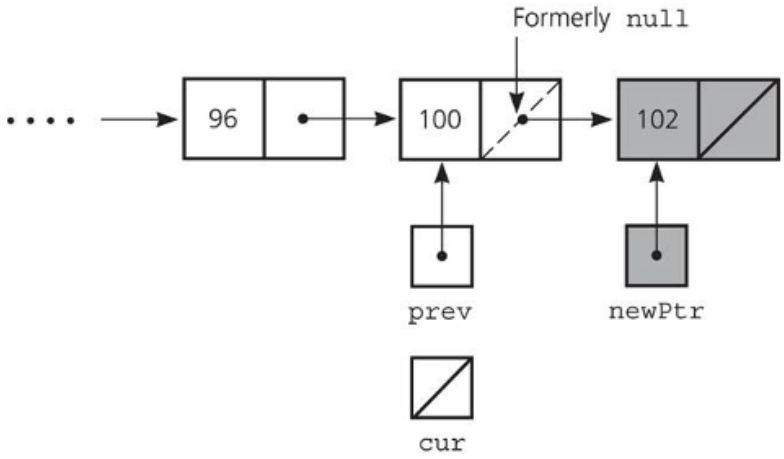
    c. To insert a node between two nodes

```
newPtr->next = cur;
prev->next = newPtr;
```



Inserting at the end of a linked list is not a special case if cur is NULL

```
newPtr->next = cur;
prev->next = newPtr;
```



8. Determining the point of insertion or deletion for a sorted linked list of objects

```
prev = NULL;
cur= head;
while ((cur != NULL) && (newValue > cur->item))   {
    prev = cur,
    cur = cur->next;
  }
```

9. Use typedef to create an alias for pointer to a Node

```
typedef Node * NodePtr;
NodePtr   head;
```

10. Implementing a pointer based (sorted) Linked List Class

List.h

a. Public methods:

Insert
Delete
IsThere
GetLength
IsEmpty

Reset
GetNextItem

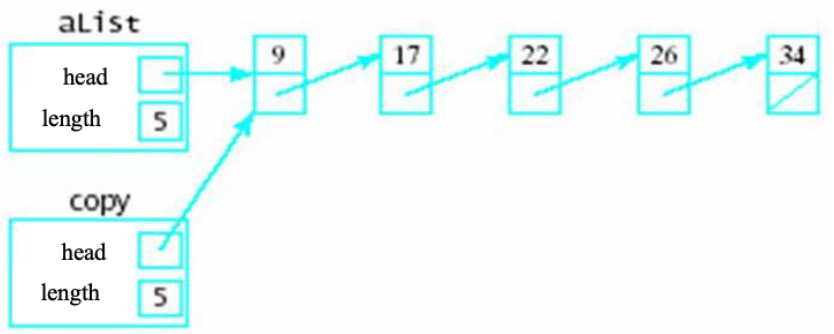
b. Private data:

```
NodePtr    head;     // pointer to the first node in the list
int            length;
NodePtr    currPos; // pointer to the current position in a traversal
```

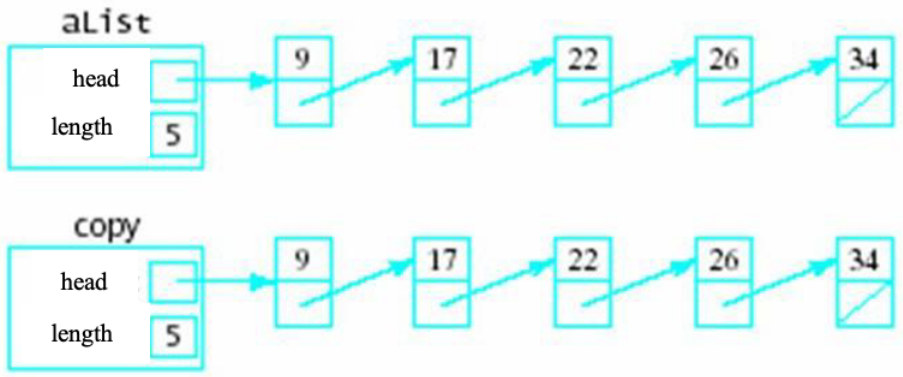
c. Default constructor initializes size and head

d. A destructor is required for dynamically allocated memory

e. Copy constructor allows a deep copy
   - Shallow copy: copy all of the member field values



   - Deep copy: copy all fields and make copies of dynamically allocated memory pointed to by the fields



f. The assignment operator should be overloaded to make a copy of the dynamically allocated memory

g. Data in a linked list node can be an instance of a class

```
typedef ClassName ItemType;
struct Node {
        ItemType item;
        Node   *next;
}; //end struct Node *head;
```