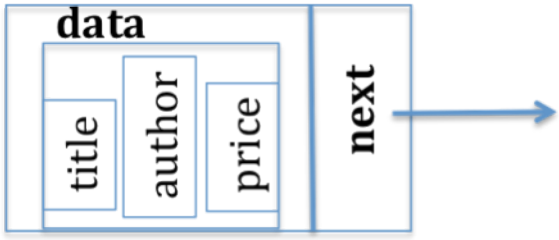
CSCI 2170 Linked List (both unsorted linked list class and sorted list class)

1. **Advantages** of using linked list, instead of array, to store data:
   a. Memory efficiency → exact amount of memory is allocated for the data
   b. Time efficiency → insertion into and deletion from a list are more efficient

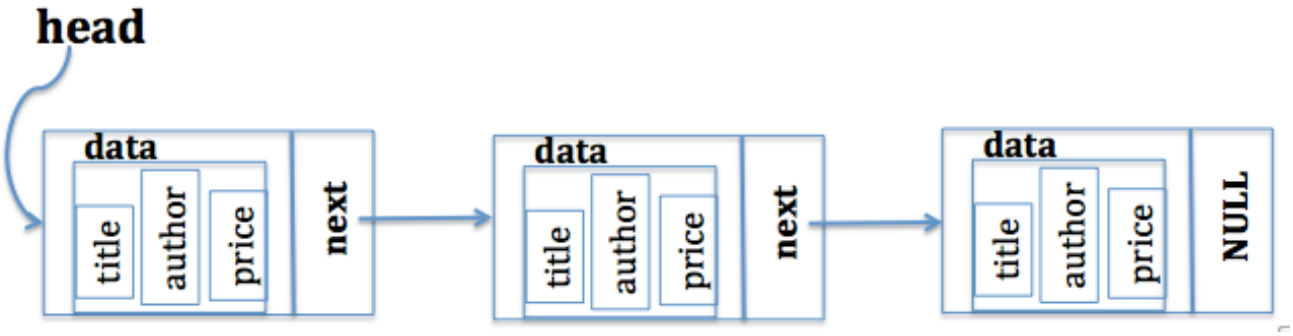
2. **Define the basic structure to build a linked list:**

```
struct   BookStruct
{
    string    title;
    string    author;
    float     price;
};
typedef BookStruct ListItemType;

struct   Node
{
        ListItemType   data;
        Node*          next;
 };
 typedef Node* NodePtr;
```



3. **Examine a linked list of 3 nodes:**



- The 1st element in the list is special. Its name is "head". It is of type NodePtr, not Node
  - NodePtr head;
  - It is the only name by which the list nodes may be accessed
  - When the list is empty, i.e., when the list is first created and no node has been inserted into the list, designate head to be NULL
    - head=NULL;
  - (head==NULL) is a condition we can use to test whether the list is empty
  - (head != NULL) is a condition we can use to test that the end of the list has not been reached

- The ***next*** field of a node contains the memory address of the next node in the list
  - Important!! – it is how the nodes are linked together
  - The next field of the last node in the list has value NULL
    - It provides a way of detecting the end of the list

4. How to create a linked list of data items?

For simplicity, the data will simply be an integer number in the following discussion:

```
typedef int ItemType;
struct Node;            // forward declaration
typedef Node* NodePtr;

struct   Node
{
        ItemType  data;
        NodePtr     next;
};
```

a. create a linked list with 3 nodes to store contact information of three person

```
NodePtr    cur = new  Node;      // create the first node
if (cur != NULL)
{
        cur→data=5;
        cur→next = NULL;
}
head = cur;   // linked list with a single node. Head pointer is pointing to the node

// create the second node for insertion
NodePtr    cur = new Node;
if ( cur != NULL)
{
        cur→data = 9;
        cur→next = NULL;
}

cur→next = head;  // linked the two nodes together by putting the new node
head = cur;               // at the beginning of the list, head is updated to point
cur = NULL;               // to the new "head" of the list
```

practice: create the 3rd node (with a value 100) and put it at the beginning of the list (how about at the end of the list? or in the middle of the list?

5. Traversing the list (starting from the head of the list, visit the nodes in the list one by one)

a. print out the information in the list

```
NodePtr   curr=head;
while (curr!=NULL)   {        // stops when the next field of the last
                              // node in the list is reached.

        cout << curr→data << endl;
        curr= curr→next;   // important! This is how to get from one
}                             // node to the next node
```

b. Given a list of N nodes, print out the information of the node at position "*position*"

```
NodePtr curr=head;
int  i=0;
while (curr !=NULL && i<position)  // detecting end of list should
{                                  // always be the first condition ( "short circuit evaluation" )
        curr = curr→next;
        i++;
};
if (curr!=NULL)
     cout << curr→data;
```

c. Given a list of N nodes, search for a specific number (linear search)

```
bool Search(const NodePtr & head,    ListItemType    toFind)  {

    bool found=false;
    NodePtr  curr=head;

    while (curr !=NULL)    {
            if (curr→data == toFind)    {
                    found=true;
                    break;
            }
            curr=curr→next;
    }
    return found;
}
```

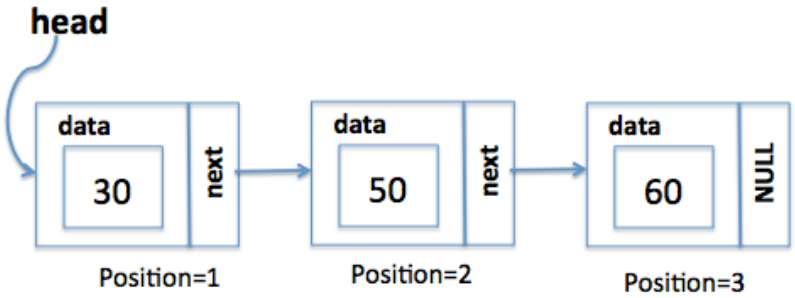
practice :
(1) how to print the position of the item in the list if the item is found?
(2) how to print out the content of the last node in the list?

Study example code on:

- **building an unsorted list**
- **printing values in an unsorted list**
- **search for value in an unsorted list, and**
- **delete an item from an unsorted list**

in the client program

6. C++ class implementing Unsorted Linked list




```
#ifndef LIST_H
#define LIST_H

typedef int ItemType;

struct NodeType;          // forward declaration
typedef NodeType * NodePtr;

struct NodeType  {
  ItemType data;
  NodePtr  next;
};

class List {
public:
  List();                                 // Post: List is the empty list.
  List(const List& otherList);            // Copy-constructor for ItemList.

  void Insert(ItemType  item);            // Pre:  item is not already in the list.
                                          // Post: item is in the list.

  void Delete(ItemType  item);
                                          // Pre:  item is in the list.
                                          // Post: item is no longer in the list.

  void Reset();                           // Post: The current position is reset to the first item in the list

  ItemType GetNextItem();                 // Assumptions: No transformers are called during the iteration.
                                          // There is an item to be returned; that is HasNext is true when this method is involed.
                                          // Pre: ResetList has been called if this is not the first iteration
                                          // Post: Returns item at the current position

  int GetLength() const;                  // Post: Length is equal to the number of items in the list.

  bool IsEmpty() const;                   // Post: Returns true if listis empty: false otherwise
  bool IsFull() const;                    // Post:  Returns true if list is full; false otherwise
  bool IsThere(ItemType item) const; // Post: Returns true if item is in the list and false otherwise
  bool HasNext() const;                   // Post: Returns true if there is another item to be returned false otherwise

  ~List();                                // destructor Post: List has been destroyed.

private:
  NodePtr   head;     // pointer to the first node in the list
  int       length;
  NodePtr   lastPtr;  // pointer to the last node in the list
  NodePtr   currPos;  // pointer to the current position in a traversal
};
```

a. **insert an item into an "unsorted list"**
(This function should be to handle insertion at ALL proper locations)

Need to consider two different cases:
Case 1: insert to an empty list
Case 2: insert into a non-empty list

b. **delete an item from the list**
two cases: (1) delete from the beginning → change the value of "head"
(2) delete from the middle or from the end of list → list traversal

Study example code on:

- **Unsorted linked list class header file : list.h**
- **Unsorted linked list implementation file: list.cpp**

7. Sorted Linked list

a. **What if the list is sorted? Assuming the list is sorted in ascending order, how to insert a node with *value 40* into the list at the appropriate spot in the list?**
(This time, we assume that we don't know ahead of time what is the correct position for this value, it is to be determined by the code itself)

Step 2: decide if the list is empty

```
if (head == NULL)
        head = newNode
else if (40 < head→data) // add the newNode as the new head
{
        … // change link
}
```

Step 3:

```
prev=head;
curr=head;
while (curr!=NULL && 40<curr→data)
{
        prev=curr;
        curr=curr→next;
}
// change link to insert
```

Does this code handle the situation where we want to insert a value 15?
Or insert a value 75?

Study example code on:

- **Sorted linked list class header file : list.h**
- **Sorted linked list implementation file: list.cpp**