LINKED LIST - II

Counting number of nodes in a list
Printing a list
Searching for a specific node in a list
Creating a list
Reversing a list
Deleting a specific node
Inserting a node in a sorted list

The following site gives a good coverage of linked lists.
http://cslibrary.stanford.edu/103/LinkedListBasics.pdf

Counting the nodes in a List

- *Recursive version:*

```
int count (struct node * pHead)
{
     if (pHead==NULL)
          return 0;
     else
     return(1 + count(pHead->next));
}
```

- ***Iterative version:***

```
int count(struct node *pHead)
{
     struct node * p;
     int c = 0;

     p = pHead;
     while (p != NULL){
          c = c + 1;
          p = p->next;
     }
     return c;
}
```

Printing the contents of a list

To print the contents, traverse the list from the head node to the last node.

```
int PrintList( struct node *list)
{
 struct node current =   list ;
```

The dummy name *current*“ is assigned to the first node of the list . We shall be moving down the list by following the *next* link of *current* , without disturbing any item in the list including its name.

```
while (current != NULL)
    {
        printf(“%d”, current -> data);
        current    =   current -> next;
     }
 return  1;
}
```

Searching for a node containing a specific item

Given a target value, the search attempts to locate the requested node in the linked list. If a node in the list matches the target value, the search returns 1; otherwise it returns 0.

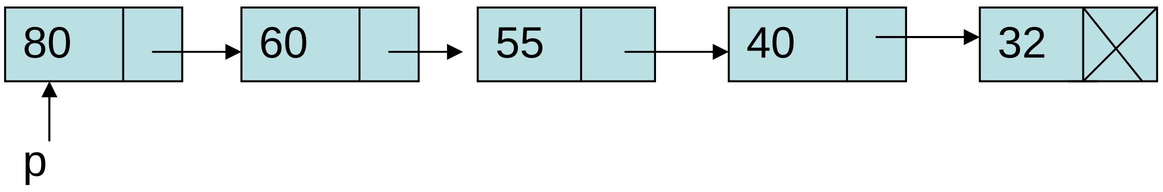
```
// Start with a dummy pointer to headnode
pCur = pHead;
// Search until target is found or we reach
// the end of list
while (pCur != NULL ){
if( pCur->data == target)
return 1;
pCur = pCur->next;
}

// target not found

return 0;
```

Creation of Linked Lists by adding nodes at front of the list

Here we assume that the main program is sending a *list* and an integer to the function. The integer is to be included as the first element of the *list*, and the new *list* is to be sent back to the main. A typical statement in the main program could be :

p =Add_Start(p, 60).

Thus given the values 32,40,55,60 and 80 the following linked list is desired.

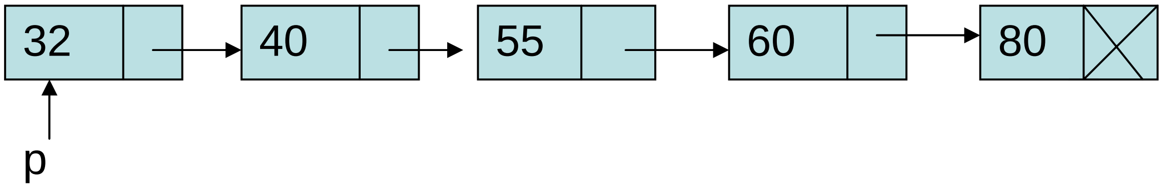


We invoke malloc to allocate space for a new node. The given integer is stored in the data part. To add the node at the front of the *list*, all you have to do is to store the pointer to the *list* in the address part of this node.

```
struct node * Add_Start(struct node *list,int d )
{
     struct node * pNew=(struct node *)
     (malloc(sizeof(struct node)));
     pNew -> data =  d ;
     pNew ->next = list ;

     return pNew;
}
```

Creation of Linked Lists by adding nodes at the end of the list

We want to create a list by adding nodes at the end of the list, as and when the user inputs the elements. The main function sends the list p and the value of the element to the function. Thus given the same values as in previous case, the desired Linked list is:

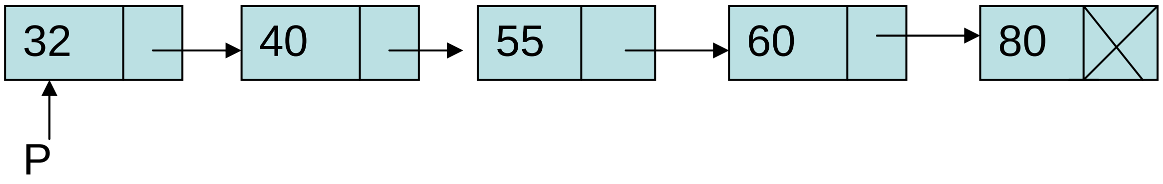


Here is a function, which adds a node to end of the linked list named **list**. Malloc is used to get a new node pNew. The element value is stored in the *data* part.
If there are no elements in the list (empty list), the first pNew is going to be the only node of list., so it returns pNew.
If there are nodes in the list, the while loop finds the last node and the pointer to pNew is stored in its address part.

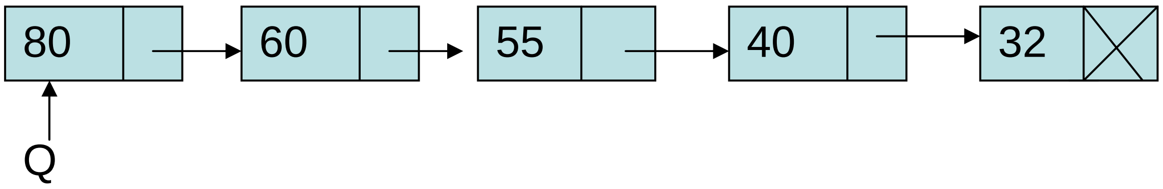
```
struct node* Addrear(struct node *list, int d) {

  struct node *pNew = NULL;
  struct node *current = NULL;

  pNew = (struct node*)malloc(sizeof(struct node));
  pNew ->data = d;
  pNew ->next = NULL;

  // Store front of the linked list
  current = list;

  //  if list is empty then this becomes the first node.
  if (list == NULL)
    return pNew;


  while (current ->next != NULL)
    current = current ->next;

  current ->next = pNew;

  // Return a pointer to the created  list.
  return list;
}
```

Reversing a linked list by storing elements backwards in another list

Given a linked list P



it is desired to reverse the elements to result in another list Q



// Reverse the list by inserting every element in front of the previous one

```
pCur= P;
second = NULL;
while(pCur !=NULL)
{
 struct node* pNew = (struct node*) (malloc(sizeof(struct
node)));
        pNew ->data = pCur ->data;
        pNew->next = second;
        second = pNew;
        pCur = pCur->next;
}
Q = second;
```

Creation of Linked Lists by adding nodes at front of the list *(using double pointers)*

Here we present a different mechanism to create a linked list. Instead of sending the pointer to the list, we send *the address of the pointer to the list* as a parameter in the function call and have the list in the memory updated by this function.

This is nothing but *passing-by-reference* the address of the pointer to head node . This can be done through a double pointer. While **list* is the address of list, the pointer to the address itself can be passed through the double pointer ***list*. See the following code where the function is getting the proper pointer, and the called function can see the changes made to the list. <u>Note this function does not return any list</u>.

.

```
int Add_Front(struct node **list,int d )
{
     struct node * pNew=(struct node *)
     (malloc(sizeof(struct node)));

     pNew-> data =  d ;
     pNew->next = NULL ;

     if( *list== NULL)
          *list  =  pNew;
     else
     {
          pNew->next  =  *list;
          *list  =  pNew ;
     }
     return 1;
}
```

A typical call from the main function would have the following form:

```
Add_Front( &pList, number );
```

A complete code is provided at the end of this lecture notes.

You may also see pages 12-14 of
<u>http://cslibrary.stanford.edu/103/LinkedListBasics.pdf</u>
for more information on this.

Deleting a Node from a Linked List

Deleting a node requires that we logically remove the node from the list by changing various link pointers and then physically deleting the node from the heap.

We can delete
- the first node
- any node in the middle
- the end node

To logically delete a node:

1. first locate the node itself , name the current node as `pCur` and its predecessor node as `pPre`.

2. change the predecessor's link field to point to successor of the current node.

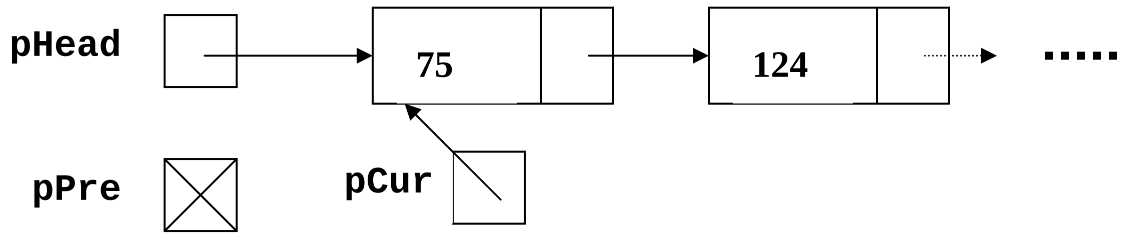
3. recycle the node (send it back to memory) using *free*.

Note: We may be deleting the only node in a list. So take care of it separately.

This will result in an empty list in which case the head pointer is set to NULL.

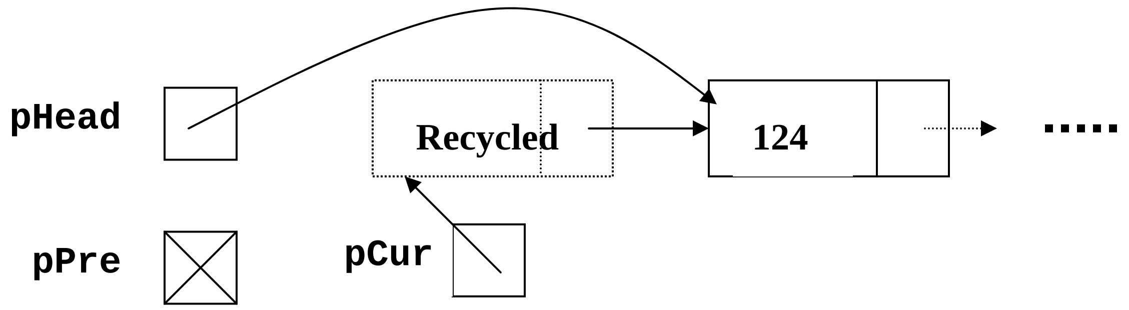
Delete First Node

BEFORE



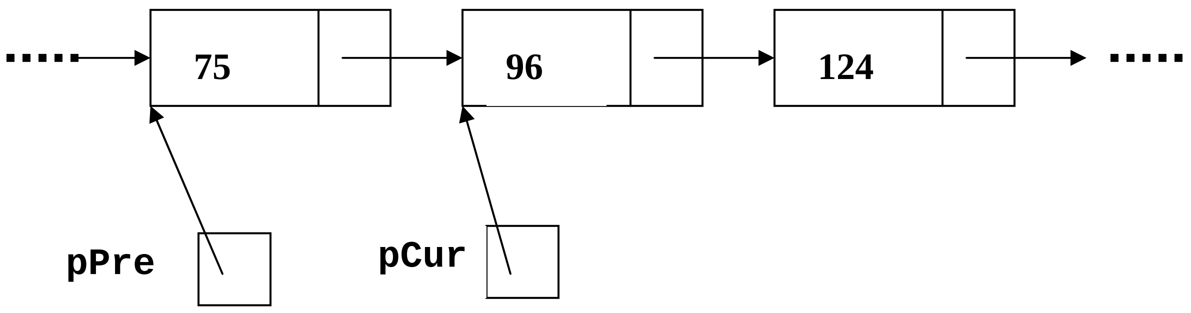
```
pCur = pHead;
pPre = NULL;
pHead = pCur -> next;
free (pCur);
```

AFTERWARDS



General Delete Case

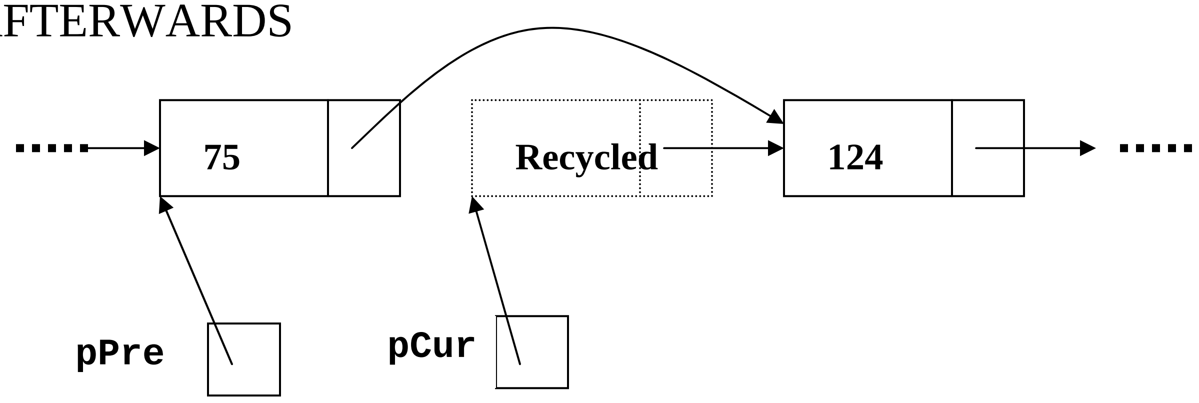
BEFORE



Here , you have to keep track of not only pCur, the node to be deleted but also its predecessor pPre.

```
pPre->next = pCur->next;
free(pCur);
```

AFTERWARDS



Algorithm for Deleting a node containing integer d from the list pHead.

It is assumed that the node containing the integer d is present in the list.
Let PCur point to the first node.
Let pPre be the pointer to the predecessor of the current node pCur. The first step would be to move through the list till pCur is the node containing the integer d. The pointer pPre is also moved so that it is always one step behind pCur.

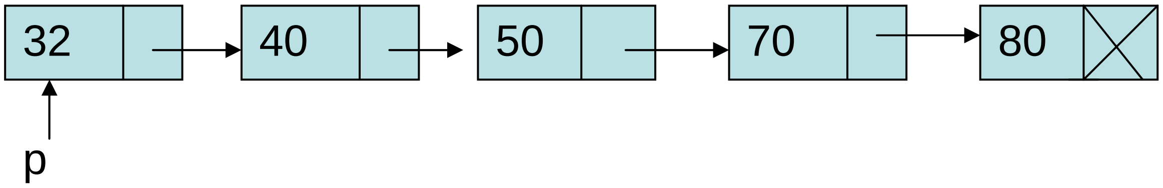
```
       I
pPre=NULL;
       pCur= pHead;
        while(pCur !=NULL  &&  pCur->data  != value)
        {
                pPre = pCur;
                pCur = pCur->next;
        }
//now pCur contains the integer d and pPre is the node previous to it
        if (pPre== NULL){
                // this means that the first node is to be deleted
                  //so make the second node the head node
                pCur = pHead;
                pHead = pCur->next;
        }
        else
                pPre->next = pCur->next;

          free(pCur);
```

Inserting an element in a sorted linked list.

You have already seen how lists can be created. Let us consider a situation where sorted data has been stored in a singly linked linear list (which is being pointed by address *head*). Now let us see how we can enter a new data value in its proper place in the list. Let the new data to be entered be "dd". The call to the function could be

```
add_sorted ( &pList, dd);
```

Let us say we want to insert 60 in the following list



The function uses a double pointer. It expects the calling function to supply the address of the head of the list.

```
void add_sorted( struct node **head,int dd)
{
struct node *pNew, *pCur, *pnext;
pNew = (struct node*) (malloc(sizeof(struct node)));
pNew -> data = dd;
pNew ->next = NULL;
```

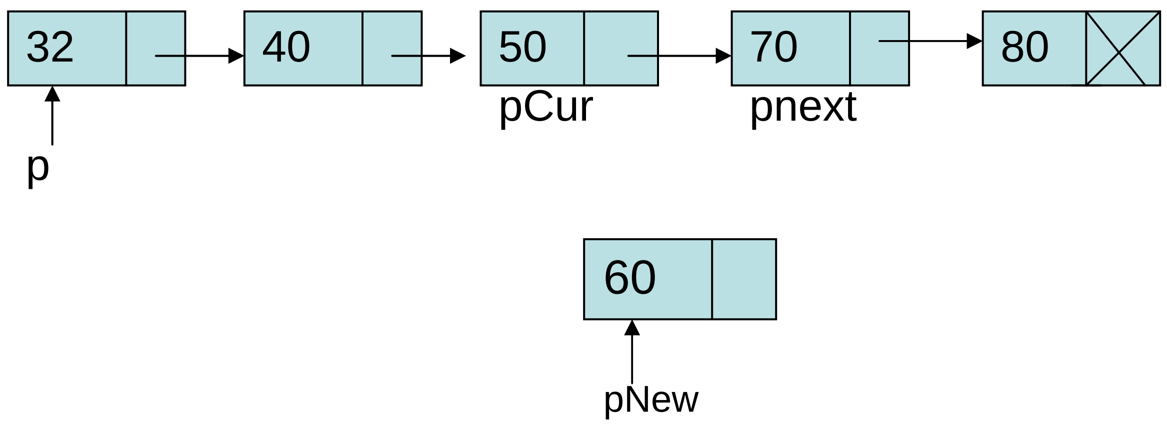
/* check if the list is empty , OR if the data is smaller than the data in the head node*/

```
if (*head==NULL || (*head)->data >= pNew -> data   ){
      pNew ->next = *head ;
      *head = pNew;
   }
else {
```

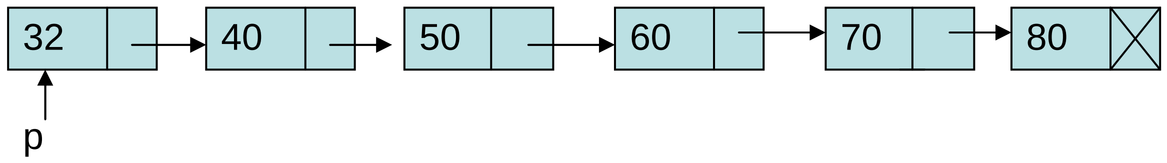
/* now examine the remaining list */

```
        pCur =*head;
        pnext  =  pCur->next;
        while(pnext!=NULL &&  pnext->data <  pNew->data )
        {
                pCur = pCur -> next ;
                pnext = pnext -> next ;
        }
```

```
At this point the list looks like this:
```



```
         pNew -> next = pCur -> next;
         pCur -> next = pNew ;
         }
}
```



Code for Creating and Printing a Linked List

/* To add elements at the start of a list and to print the contents */

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
     int data;
     struct node  *next;
  };



int PrintList( struct node *list);
int Add_Front(struct node* *list,int d );



main( ) {
  int number = 0;
  struct node *pList=NULL;

   while(number!= -1)
   {
       printf("enter data for next node \n ");
       scanf("%d", &number);

       if (number !=-1)
          {
              AddStart ( &pList, number );
             /* pass address of pList to the add function */
           }
   }
   printf("items in linked list \n");
   PrintList (  pList  );
   return 1;
}
```

```
int Add_Front(struct node* *list,int d )
{

   struct node * pNew=(struct node *) (malloc(sizeof(struct
                                                              node)));

   pNew-> data =  d ;
   pNew->next = NULL) ;
   if(*list== NULL)

       *list  =  pNew;

   else
   {
       pNew->next  =  *list;
      *list  =  pNew ;
   }
   return 1;
}


int PrintList( struct node *list)
{
 struct node current =   list ;

while (current != NULL)
    {
        printf(“%d”, current -> data);
        current    =   current -> next;
     }
 return  1;
}
```