

Linked Lists

Vector and Array:

Vector is similar to array in the way of accessing an element by its index. But, array has a fixed size of elements and vector has a dynamical size of elements. That is, vector can grow or shrink at the end of the sequence.

Why do we sometimes use a list in an application instead of using a vector?

- To insert or delete an element from a vector with n elements takes $O(n)$ time.
- To insert or delete an element from a list with n elements takes $O(1)$ time.

Declare a node class to use for implementing a linked list:

```
//linked list node
template <typename T>
class Node
{
public:
    T nodeValue;           //data held by the node
    Node<T> * next;       //next node in the list

    // default constructor with no initial value
    Node ()
    {
        next = NULL;
    }

    // constructor. Initialize nodeValue and next
    Node (const T& item, Node<T> * nextNode = NULL)
    {
        nodeValue = item;
        next = nextNode;
    }
};
```

Declare an object of a Node:

```
Node<string> *p, *q, *head;           //declare a Node pointer
p = new Node<string> ("Chu");         //dynamically allocate a Note
head = p;                             //head points to the beginning.
q = new Node<string> ("Savur");
p->next =q;                             //p points to q
```

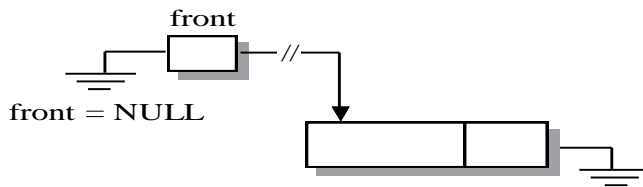
//Print the list

```
p=head;
while(p != NULL)
{
    cout<<p->nodeValue<<" ";
    p = p->next;
}
cout<<endl;
```

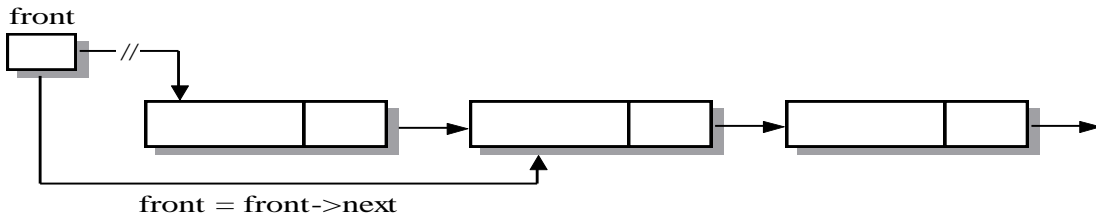
//Deallocate all dynamic memories

```
Node<string> * toBeDelete;
p= head;
while(p != NULL)
{
    toBeDelete = p->next; //Save the address of the next node
    delete p; //Release the heap variable.
    p = toBeDelete;
}
```

Delete a front node:

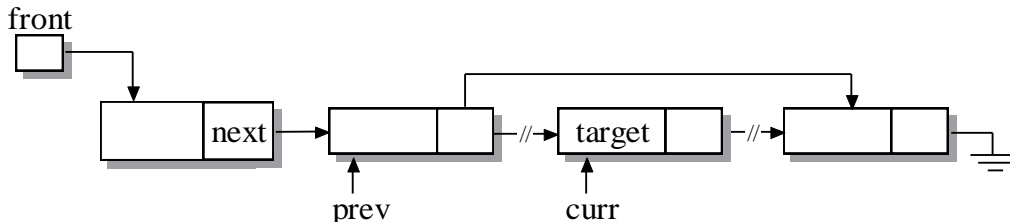


Deleting front of a 1-node list



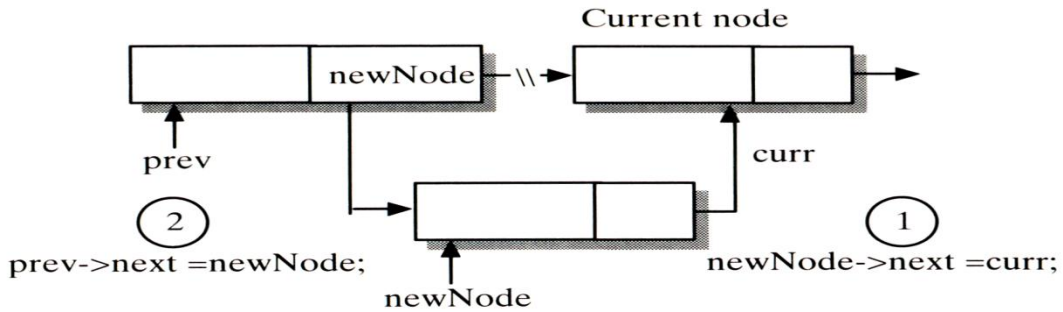
Deleting front of a multi-node list

Removing a Target Node:

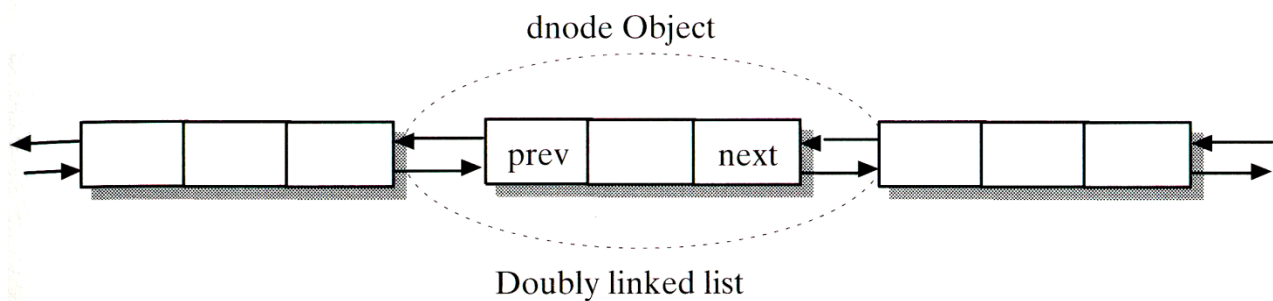


```
prev->next = curr ->next; //reconnect prev to the successor of curr
delete curr; //delete the unlinked node
```

Insert a new Node before the current Node:



Double Linked Lists:



```

template <typename T>
class DNode
{
public:
    T nodeValue;           //data held by the node
    DNode<T> * prev;       //previous node in the list
    DNode<T> * next;       //next node in the list

    // default constructor with no initial value
    DNode ()
    {
        prev = NULL;
        next = NULL;
    }

    // constructor. Initialize nodeValue and next
    DNode (const T& item, DNode<T> * prevNode = NULL, DNode<T> * nextNode = NULL)
    {
        nodeValue = item;
        prev = prevNode;
        next = nextNode;
    }
};

```

Another example of [Double Linked List](#) (click it)