Class Node

```
template <class T>
class Node {
  // declare class List a friend so that it can access
  //Node's private vars.
   friend class List<T>;
 public:
    // constructors
    Node()  { nextPtr = 0; prevPtr=0;  }
    Node(const T & d)  { data = d; nextPtr = 0; prevPtr=0;}
    Node(const T & d, Node<T> *n, Node<T> *p)
          {data = d; nextPtr = n; prevPtr=p; }

  private:
    T data;                    // data
    Node<T> *nextPtr;    // next node in the list
    Node<T> *prevPtr;    // Prev node in the list

};
```

Class DList

```
// The DList will contain nodes linked together by pointers
template <class T>
class DList {
public:
   DList() { HeadPtr = 0;TailPtr=0; CurPtr=0; size = 0; }
// constructor
   DList(const DList<T> &); // copy constructor
   ~DList();     // destructor
   void deleteDList();
   void insertAtFront( const T & );
   void removeAtFront();
   void printForward() const;
   void printReverse() const;
   int getSize() const  { return size;  }
   DList& operator = (const DList<T> &);
private:
   Node<T> *HeadPtr;  // pointer to first node
   Node<T> *TailPtr;  // pointer to first node
   Node<T> *CurPtr;  // pointer to first node
   int size;
};
```

Destructor

```
// Destructor
template<class T>
DList<T>::~DList()
{
   deleteList();
   size = 0;
}
```

Delete all Node from the List

```
template<class T>
void DList<T>::deleteDList()
{
   if (HeadPtr != 0) {    // List is not empty
      cout << "Destroying nodes ...\n";
      Node<T> *currentPtr = HeadPtr;
      Node<T> *tempPtr;

      while ( currentPtr != 0 ) {  // not end of list
         tempPtr = currentPtr;
         cout << "Deleting " << tempPtr->data << endl;
         currentPtr = currentPtr->nextPtr; // move ptr
         delete tempPtr; // delete last node
      }
   }
   cout << "All nodes destroyed\n\n";
}
```

Insert At the Front of the List

```
// Insert a node at the front of the list
template<class T>
void DList<T>::insertAtFront( const T &value )
{
    // create a new node with the value in it.
   Node<T> *newPtr = new Node<T>(value);
   assert(newPtr != 0);

   if ( HeadPtr == 0 )  // if List is empty
       TailPtr = newPtr; // point to new node
   else            // if List is not empty
       HeadPtr->prevPtr = newPtr; // point to list
```

```
   newPtr->nextPtr = HeadPtr;
   HeadPtr=newPtr;  // move up firstPtr

   size++;
}
```

Delete From the Front of the List

```
// Delete a node from the front of the list
template<class T>
void DList<T>::removeAtFront()
{
   if ( HeadPtr == 0 )              // List is empty
       cout << "Error: list is empty!\n";
   else {
      Node<T> *tempPtr = HeadPtr;
      cout <<"Deleting " << tempPtr->data << endl;
      HeadPtr->nextPtr->prevPtr=0;
      HeadPtr = HeadPtr->nextPtr; // move to next node
      delete tempPtr;
      size--;
   }

}
```

Insert At the End of the List

```
// Insert a node at the front of the list
template<class T>
void DList<T>::insertAtEnd( const T &value )
{
    // create a new node with the value in it.
   Node<T> *newPtr = new Node<T>(value);
   assert(newPtr != 0);

   if ( HeadPtr == 0 )  // if List is empty
       HeadPtr = newPtr; // point to new node
   else            // if List is not empty
       {TailPtr->nextPtr = newPtr; // point to list
        newPtr->prevPtr = TailPtr;
       }
```

```
    TailPtr=newPtr;  // move up firstPtr

    size++;
}
```

Delete From the End of the List

```
// Delete a node from the front of the list
template<class T>
void DList<T>::removeAtEnd()
{
     Node<T> *tempPtr = TailPtr;
   if ( HeadPtr-> nextPtr == 0 )         // if only one item
       HeadPtr=0;
   else
      TailPtr->PrevPtr->nextPtr=0;
   TailPtr = TailPtr->prevPtr; // move to next node
      delete tempPtr;
      size--;
   }

}
```

Print the contents Of the List in Forward Direction

```
// Display the contents of the List
template<class T>
void List<T>::printForward() const
{
   if ( HeadPtr == 0 ) {
      cout <<"The list is empty\n\n";
   }
   else   {
     Node<T> *currentPtr = HeadPtr;
     cout <<"The list's Contents are : ";
     while ( currentPtr != 0 ) { // not end of list
         cout <<currentPtr->data << " -> ";
         currentPtr = currentPtr->nextPtr;
     }
     cout <<"\n";
   }
}
```

Print the contents Of the List in Reverse Direction

```
// Display the contents of the List
template<class T>
void List<T>::printReverse() const
{
   if ( HeadPtr == 0 ) {
      cout <<"The list is empty\n\n";
   }
   else   {
     Node<T> *currentPtr = TailPtr;
     cout <<"The list's Contents are : ";
     while ( currentPtr != 0 ) { // not end of list
         cout <<currentPtr->data << " -> ";
         currentPtr = currentPtr->prevPtr;
     }
     cout <<"\n";
   }
}
```

Driver Program

```
#include <iostream.h>
#include "DList.h"

int main()
{
  DList<int> L1;
  cout << "The size of the list is: " << L1.getSize() <<
endl;
  L1.insertAtFront(123);
  L1.insertAtEnd(456);
  L1.insertAtFront(789);
  cout << "The size of the list now is: " << L1.getSize()
<< endl;
  L1.printForward();
  L1.removeAtFront();
  L1.printReverse();
  cout << "The size of the list now is: " << L1.getSize()
<< endl;
  return 0;
}
```