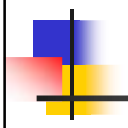

Linked List

By: Mr. Khaleifah Al.jada'
Data Structures
Al- Majma'ah College

Linked List Features

- We can add or remove from any place in the list.
- There is a very important pointer which is called "head" must be at the head of the list.
- there are many types of linked list like doubly linked list, and circular linked list.

Class of linked list

```
class List
{
public:
   List(); // constructor
   ~List(); // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
```

Cont…

```
private:
   ListNode*firstPtr; // pointer to first node
   ListNode*lastPtr; // pointer to last node

   // utility function to allocate new node
   ListNode*getNewNode( const NODETYPE & );
}; // end class List
```

Constructor

```
List::List()
{
  firstPtr = NULL;
  lastPtr = NULL;
}
```

Destructor

```
List::~List()
{
  if ( !isEmpty() ) // List is not empty
  {
    cout << "Destroying nodes ...\n";

    ListNode *currentPtr = firstPtr;
    ListNode *tempPtr;

    while ( currentPtr != 0 ) // delete remaining nodes
    {
      tempPtr = currentPtr;
```

Cont..

```
        cout << tempPtr->data << '\n';
        currentPtr = currentPtr->nextPtr;
        delete tempPtr;
      } // end while
    } // end if

    cout << "All nodes destroyed\n\n";
} // end List destructor
```

Add at front

```
void List::insertAtFront( const NODETYPE &value )
{
  ListNode*newPtr = getNewNode( value ); // new node

  if ( isEmpty() ) // List is empty
    firstPtr = lastPtr = newPtr; // new list has only one node
  else // List is not empty
  {
    newPtr->nextPtr = firstPtr;
   firstPtr = newPtr; // aim firstPtr at new node

  } // end else

} // end function insertAtFront
```

Add at back

```
void List::insertAtBack( const NODETYPE &value )
{
   ListNode *newPtr = getNewNode( value );
   if ( isEmpty() ) // List is empty
      firstPtr = lastPtr = newPtr; // new list has only one node
   else // List is not empty
   {
      lastPtr->nextPtr = newPtr; // update previous last node
      lastPtr = newPtr; // new last node
   } // end else
} // end function insertAtBack
```

Remove from front

```
bool List::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() ) // List is empty
      return false; // delete unsuccessful
   else
   {
      ListNode *tempPtr = firstPtr; // hold tempPtr to delete

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0; // no nodes remain after removal
```

Cont…

```
else
        firstPtr = firstPtr->nextPtr; // point to next 2nd node

        value = tempPtr->data; // return data being removed
        delete tempPtr; // reclaim previous front node
        return true; // delete successful
    } // end else
} // end function removeFromFront
```

Remove from Back

```
bool List::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() ) // List is empty
      return false; // delete unsuccessful
   else
   {
      ListNode *tempPtr = lastPtr; // hold tempPtr to delete

      if ( firstPtr == lastPtr ) // List has one element
         firstPtr = lastPtr = 0; // no nodes remain after removal
      else
      {
         ListNode *currentPtr = firstPtr;

         // locate second-to-last element
```

Cont…

```
while ( currentPtr->nextPtr != lastPtr )
          currentPtr = currentPtr->nextPtr; // move to next node

        lastPtr = currentPtr; // remove last node
        currentPtr->nextPtr = 0; // this is now the last node
      } // end else

      value = tempPtr->data; // return value from old last node
      delete tempPtr; // reclaim former last node
      return true; // delete successful
   } // end else
} // end function removeFromBack
```

Is Empty

```
bool List::isEmpty() const
{
   return firstPtr == 0;
} // end function isEmpty
```

```
ListNode *List::getNewNode(const NODETYPE
   &value )
{
   return new ListNode( value );
} // end function getNewNode
```

Print List

```
void List::print() const
{
   if ( isEmpty() ) // List is empty
   {
      cout << "The list is empty\n\n";
      return;
   } // end if

   ListNode*currentPtr = firstPtr;

   cout << "The list is: ";
```

Cont..

```
while ( currentPtr != 0 ) // get element data
   {
      cout << currentPtr->data << ' ';
      currentPtr = currentPtr->nextPtr;
   } // end while

   cout << "\n\n";
} // end function print
```

"وآخر دعواهم أن الحمدلله رب العالمين"