Objective:

- To make a strong grip on link-based structure.

Note: In almost all the problems the link list is assumed to be linear single link list unless specified explicitly and you are not allowed to change the node links or nodes data unless specified. Do keep in mind the performance factor while developing logic.

STL classes to surf Link List

http://www.cplusplus.com/reference/forward_list/forward_list/
http://www.cplusplus.com/reference/list/list/

Task – 1: ***Union (Math: Set Operation) of two Link Lists.***

If list1 contain the elements 2, 3, 4, 5, 7, 9, 11 and list2 contain elements 3, 4, 10, 1, 5 then after calling this function the resultant list should contain elements 2, 3, 4, 5, 7, 10, 1, 9, 11

Note: if you are implementing it as a member function of class 'LSLL<T>' then function prototype will be

Prototype

```
LSLL<T> LSLL<T>::doUnion(LSLL<T>& list2);
List1 is represented by *this and list2 will be received in function.
Same pattern can be followed for rest of the problems
```

Task – 2: ***Equality of Lists.***

This function determines that two given lists are equal or not (Math Set Operation: Equality).

Prototype

```
bool LSLL<T>::isEqual(LSLL<T> &list2); //list1 is *this and list2 is being received.
```

Task – 3: ***Create List Clone.***

This function returns the deep copy (clone) of the list (*this).

Prototype

```
LSLL<T> LSLL<T>::createClone();
```

Task – 4: ***Delete Alternate nodes.***

This function will delete every second element from the start of the list e.g. if a list contains the elements 2, 3, 4, 5, 7, 9, 11 then after calling this function the list should contain elements 2, 4, 7, 11.

Prototype

```
Void LSLL<T>::delAlternate(); //delete alternate nodes of the calling object.
```

Task – 5: ***Split List.***

This function will count the number of nodes in a linked list, and split the linked list form the middle on the basis of number of nodes. list1 and list2 be initially empty. If the original linked list contains 8 elements, then after the function call, list1 will contain the first four elements and list2 will contain the last four elements. If the original list contains 7 elements, then list1 will contain 4 elements and list2 will contain 3 elements after the function call. Also, after the function has been called the original list will become empty. You are required to print all three lists after the function has been called.

Prototype

```
void LSLL<T>::splitList (LSLL<T> & list1, LSLL<T> & list2); //the original list means the
                                                            calling object
```

Task – 6: ***Remove all the duplicate nodes in a link list***

When this function will be called on some link list then after the execution of this function the list will not contain any node with its info repeating more then once. E.g. if list contains 23, 5, 4, 23, 6, 78, 4, 5 then after the list will contain only 23, 5, 4, 6, 78

Prototype

```
void LSLL<T>::removeDuplicates();
```

Task – 7: ***Merge two sorted link list into a 3rd link list so that the resultant must also be sorted***

E.g. is list1 contains 1, 2, 3, 40, 50 and list2 contains 10, 40 60 the result list will contain 1, 2, 3, 10, 40, 50, 60.

Prototype

```
LSLL<T> LSLL<T>::mergeSortedLists(LSLL<T> & list2); // list1 means *this
```

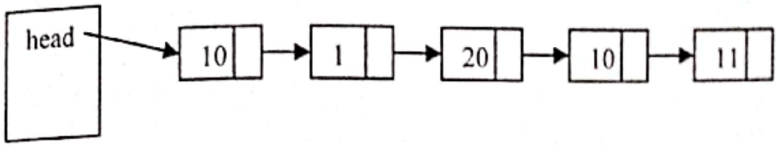
Task – 8: ***Reverse print the link list iteratively and recursively.***

Prototype

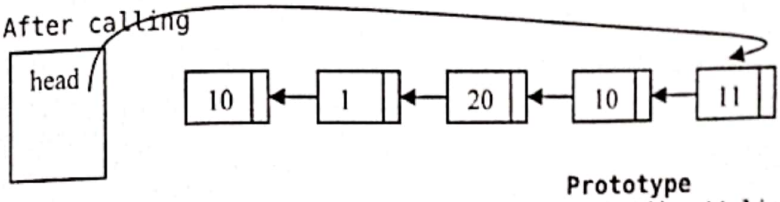
```
void LSLL<T>::reversePrint(); // list means *this
```

Task – 9: ***Reverse the link list structure iteratively and recursively***
Note: No data swapping is allowed

List Before calling



List After calling

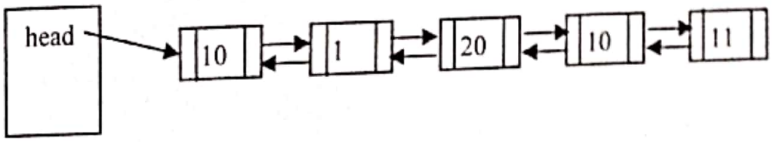


Prototype

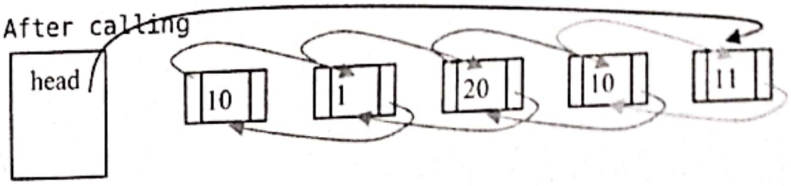
```
void LSLL<T>::reverseList(); // list means *this
```

Task – 10: ***Reverse the link list structure iteratively and recursively***
Note: No data swapping is being done only address swapping is being involved

List Before calling



List After calling



Prototype

```
void LDLL<T>::reverseList(); // list means *this
```

Task – 11: ***Swap M^{th} and N^{th} number nodes***
Note: instead of doing data swapping, the node addresses/links must be swapped.

Task – 12: ***Sort the link list on the bases of information stored in it.***
Note: Make two versions of it: one that adapts address swapping and the other one adapt data swapping.

Task – 13: ***Can we apply the binary search on link list and still getting log(N) searching cost.***

Task – 14: ***Null or Cycle***
You are given a linked list that is either NULL-terminated (acyclic), as shown in Figure 3.5, or ends in a cycle (cyclic), as shown in Figure 3.6.

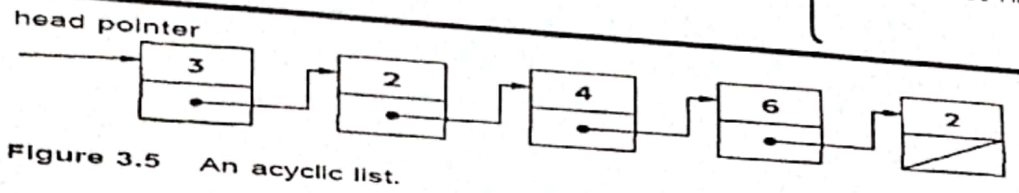


Figure 3.5 An acyclic list.

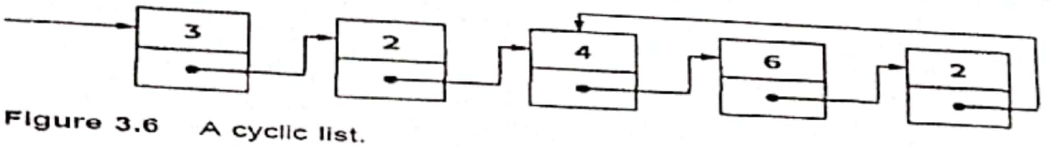


Figure 3.6 A cyclic list.

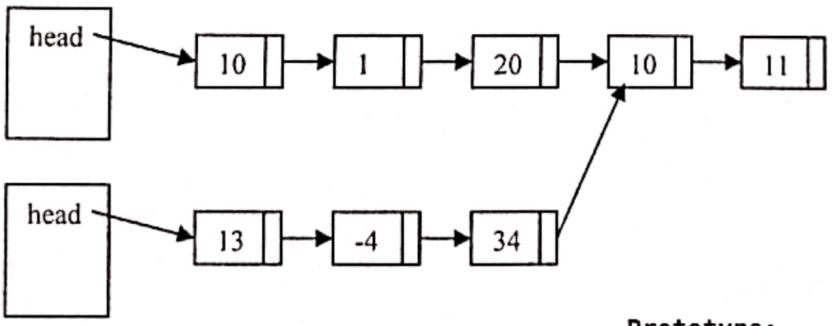
Write a O(N) time bound function that takes a pointer to the head of a list and determines if the list is cyclic or acyclic. Your function should return 0 if the list is acyclic and 1 if it is cyclic. You may not modify the list in any way.

Task – 15: ***Count black and white nodes frequency***

A research student is given a singly-linked list. Each node of the list has a color, which is either "Black" or "White". He must find if there are more black nodes than white nodes, or vice versa. His advisor gives him 5,000Rs to buy a computer to do the work. He goes to the computer store and finds a slightly defective computer which costs a mere 3,000Rs. This computer has the small problem of **not being able to do arithmetic**. This means that he cannot use a counter to count the nodes in the list to determine the majority color. The computer is otherwise fully functional. He has the evil idea that he could buy the defective computer and somehow use it to do his work, so that he can use the rest of the money on enjoyment. Show how he can accomplish this amazing task. Write code for an algorithm called 'findMajorityColor' which takes a singly-linked list, L, with n nodes and returns the majority color among nodes of L. This algorithm should have the same asymptotic running time as counting the nodes (O(n)). **Note:** No arithmetic is allowed.

Task – 16: ***Joining Point***

Write a function which return true if the two lists join at some point otherwise return false.



Prototype:

```
bool LSLL<T>::isJoining(LSLL<T> & );
returns true for the above diagram.
```

Task – 17: ***Find Joining Point***

Modify Problem-16 such that it returns value stored in the joining point.

Task – 18:

Given a singly linked list $L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$. Rearrange the nodes in the list so that the new formed list is: $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \dots$

You are required do this in-place without altering the node's values.

Examples:
Input: 1 → 2 → 3 → 4 → 5
Output: 1 → 5 → 2 → 4 → 3

Task – 19: ***Middle Point***

Write a function which returns the value of the middle node in a given link list. Do it in single pass.

```
Examples:
Input:  1 → 2 → 3 → 4 → 5
Output: 3
```

Prototype:

```
T LSLL<T>::findMiddlePoint();
```

Task – 20:

Given a sorted doubly linked list of positive distinct elements, the task is to find pairs in doubly linked list whose sum is equal to given value x, without using any extra space?

```
Example
Input: head : 1↔ 2 ↔ 4 ↔ 5 ↔ 6 ↔ 8 ↔ 9  x = 7
Output: (6, 1), (5,2)
```

Task – 21:

Given a sorted singly linked list and a value x, your task is to find pair(s) whose sum is equal to x. We are not allowed to use any extra space and expected time complexity is O(n).

```
Examples:
Input: head : 1→ 2 → 4 → 5 → 6 → 8 → 9      x = 7
Output: (6, 1), (5,2)
```

Task – 22:

Given a linked list, rearrange it such that converted list should be of the form a < b > c < d > e < f .. where a, b, c.. are consecutive data node of linked list.

```
Examples :
Input:   1 → 2 → 3 → 4
Output: 1 → 3 → 2 → 4
Input:   11 → 15 → 20 → 5 → 10
Output: 11 → 20 → 5 →15 → 10
```

Task – 23:

Given a singly linked list, remove all the nodes, which have a greater value on right side.

Examples:

a) The list 12→15→10→11→5→6→2→3→NULL should be changed to 15→11→6→3→NULL.
b) The list 10→20→30→40→50→60→NULL should be changed to 60→NULL. Note that 10, 20, 30, 40 and 50 have been deleted because they all have a greater value on the right side.
c) The list 60→50→40→30→20→10→NULL should not be changed.