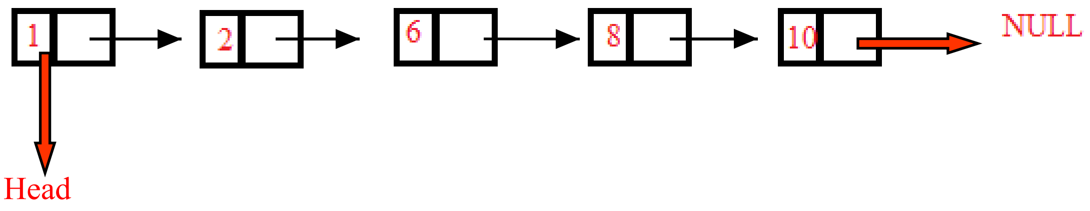
Singly Linked List

Singly linked list is the most basic linked data structure. In this the elements can be placed anywhere in the heap memory unlike array which uses contiguous locations. Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node contains the NULL reference. It has a dynamic size, which can be determined only at run time.



Basic operations of a singly-linked list are:

1. Insert – Inserts a new element at the end of the list.
2. Delete – Deletes any node from the list.
3. Find – Finds any node in the list.
4. Print – Prints the list.

Algorithm:

The node of a linked list is a structure with fields data (which stored the value of the node) and *next (which is a pointer of type node that stores the address of the next node).

Two nodes *start (which always points to the first node of the linked list) and *temp (which is used to point to the last node of the linked list) are initialized. Initially temp = start and temp->next = NULL. Here, we take the first node as a dummy node. The first node does not contain data, but it used because to avoid handling special cases in insert and delete functions.

Functions –

1. Insert – This function takes the start node and data to be inserted as arguments. New node is inserted at the end so, iterate through the list till we encounter the last node. Then, allocate memory for the new node and put data in it. Lastly, store the address in the next field of the new node as NULL.
2. Delete - This function takes the start node (as pointer) and data to be deleted as arguments. Firstly, go to the node for which the node next to it has to be deleted,

    If that node points to NULL (i.e. pointer->next=NULL) then the element to be deleted is not present in the list.

    Else, now pointer points to a node and the node next to it has to be removed, declare a temporary node (temp) which points to the node which has to be removed. Store the address of the node next to the temporary node in the next field of the node pointer (pointer->next = temp->next). Thus, by breaking the link we removed the node which is next to the pointer (which is also temp). Because we deleted the node, we no longer require the memory used for it, free() will deallocate the memory.

3. Find - This function takes the start node (as pointer) and data value of the node (key) to be found as arguments. First node is dummy node so, start with the second node. Iterate through the entire linked list and search for the key.

    Until next field of the pointer is equal to NULL, check if pointer->data = key. If it is then the key is found else, move to the next node and search (pointer = pointer -> next). If key is not found return 0, else return 1.

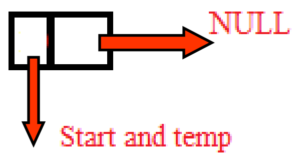
4. Print - function takes the start node (as pointer) as an argument. If pointer = NULL, then there is no element in the list. Else, print the data value of the node (pointer->data) and move to the next node by recursively calling the print function with pointer->next sent as an argument.

Performance:

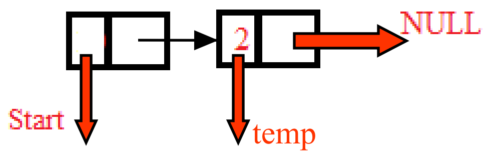
1. The advantage of a singly linked list is its ability to expand to accept virtually unlimited number of nodes in a fragmented memory environment.
2. The disadvantage is its speed. Operations in a singly-linked list are slow as it uses sequential search to locate a node.

Example:

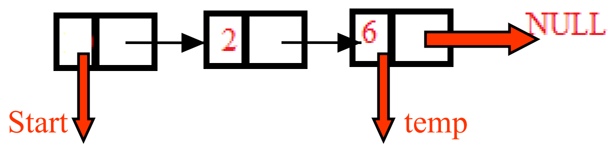
Initially



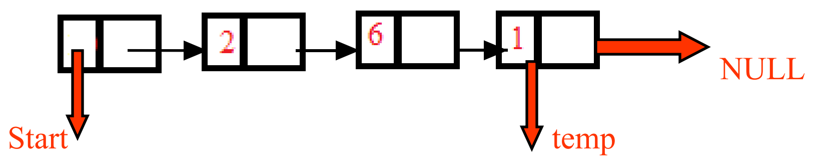
Insert(Start,2) - A new node with data 2 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



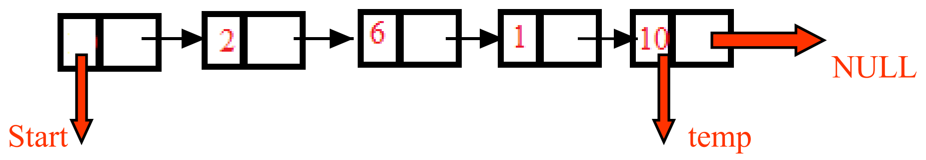
Insert(Start,6) - A new node with data 6 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



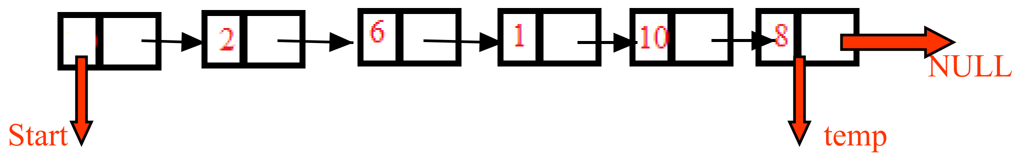
Insert(Start,1) - A new node with data 1 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.



Insert(Start,10) - A new node with data 10 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.

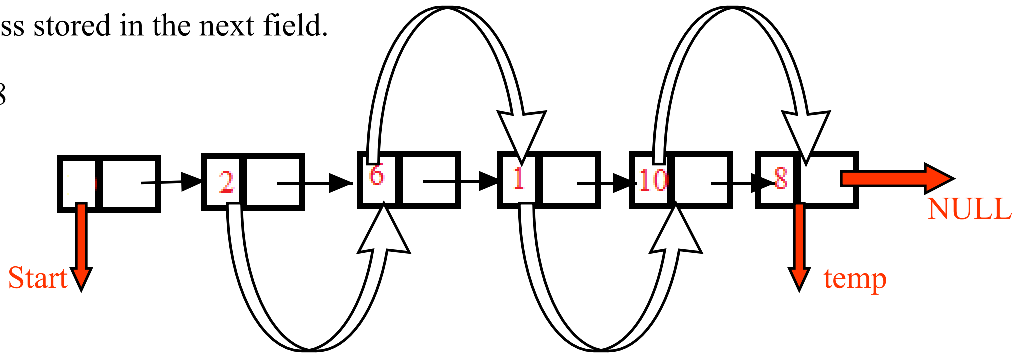


Insert(Start,8) - A new node with data 8 is inserted and the next field is updated to NULL. The next field of previous node is updated to store the address of new node.

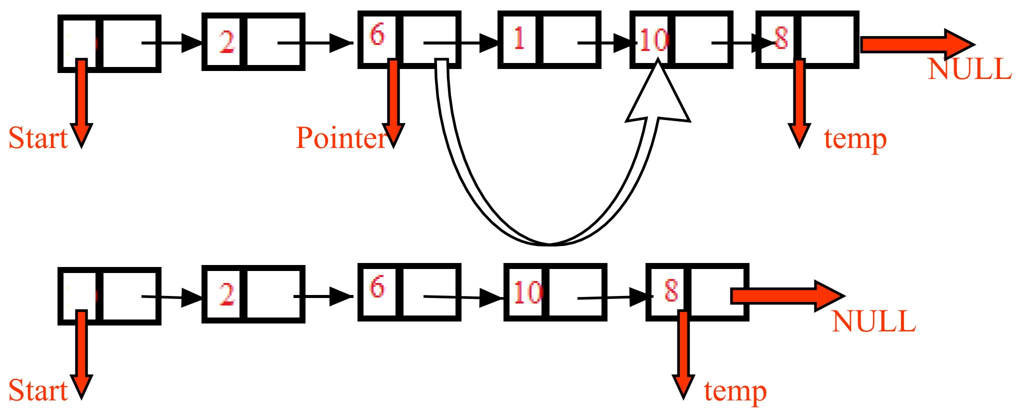


Print(start->next) - To print start from the first node of the list and move to the next with the help of the address stored in the next field.

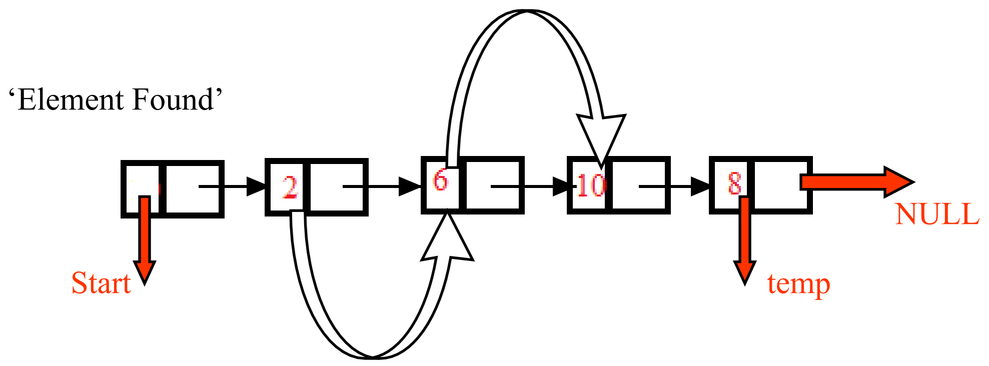
2 ,6 ,1 ,10 ,8



Delete(start,1) - A node with data 1 is found and the next field is updated to store the NULL value. The next field of previous node is updated to store the address of node next to the deleted node.



Find(start,10) - To find an element start from the first node of the list and move to the next with the help of the address stored in the next field.



'Element Found'
2
6
10
8
NULL
Start
temp