Linked list

Singly linked list :-

It is a linear collection of nodes where each node contains data value and a pointer to hold the address of its coming after node. The first node is tracked by keeping its address in a pointer called start or home. To traverse the complete list of to goto a particular node we must have to access the first node. That is the list can be traversed only from begining to end this is why it is called as one way or singly linked list. The following operation are done on such a list :-

(1) **Creation of the list :-** As each node contains at least two parts to create the linked list at first we need to define a structure that describes the type of datapart and pointer.

After that the 'Home' pointer is declared i.e assignment with null to create the empty list:

Algorithm :-

Step 1:- Define structure node with members data as any primitive data type and a pointer link as node type.

Step 2:- Declare a pointer variables start of node type.

Step 3:- Let start = null

Step 4:- End

Ex- typedef struct nd
{
int data;
struct nd * link;
}
node; (new datatype based on structured)
node * start = Null;

(12) Traversal of singly linked list —
Since, singly linked list is an one way list, to traverse it we must have to locate the first node and from that we can go ahead through its link part untill the list is finished. This traversal is done by assigning a pointer with the start or home then accessing the location pointed by this pointer and assigning the pointer with the link part of the node. The whole process procedure is repeated untill the pointer becomes null.

Algorithm

1. Start Traverse to show()
2. Declare pointer Ptr as node type
3. Let Ptr = start
4. Repeat step 5 & 6 while Ptr != Null
5. Print data of Ptr
6. Let Ptr = link of Ptr
7. End of step 4 while
8. End of function.

Algorithm for inserting 1st node

Step 1 :- start, insfirst (val)

Step 2 :- Declare pointer ptr As node type

Step 3 :- Allocate memory for node to ptr

Step 4 :- Let data of ptr = val

Step 5 :- Let link of ptr = start

Step 6 :- Let start = ptr

Step 7 :- End of function.

* **Inserting at the end of list :-**
To insert a node the end of an existing list we need to allocate memory location for a node and to assign the next member or linked with the address of deallocated memory but to traverse to the last node starting from the first node. This operation is done by using the following algorithm.

Step 1:- Start Ins last (v)

Step 2:- Declare pointers ptr and s as node type.

Step 3:- Allocate memory to pointer ptr.

Step 4:- Let data of ptr = v

Step 5:- Let Link of ptr = Null

Step 6:- If start = Null then

6A Let start = ptr

6B Return

Step 7:- Let S = start

Step 8:- while Link of S != Null

Step 9:- Let S = Link of S

Step 10:- End of while

Step 11:- Let S = ptr

Step 12:- Return

* Inserting a node at the given position:-
After allocating memory for the new node and assigning its data-part we need to traverse to the node coming before the given position by changing the link part of the pointer starting from start. After the we assign the link part of the new node with connect it to the node currently present at the given position and then we assign the link part of the ~~tran~~ traverse node with the address of newly allocated node. It will follow the given algorithm.

Step 1:- Start Insmid (v, p)

Step 2:- Declare pointers ptr and s of node type
Declare counter c = 0

Step 3:- If start = Null then

3.A Print "List empty"

3.B Return

Step 4: If P = 1 Then

4.A call Instlist(v)

4.B Return

Step 5:- Let S = start

Step 6:- while S != Null And C < P-1

Step 7:- Let S = Link of S

Step 8:- Let C = C+1

Step 9:- End of step 6 Loop

Step 10:- If S = Null And C < P-1 Then,

10.A Print "Position not found"

10.B Return

Step 11:- end of if

Step 12:- Allocate memory to ptr

Step 13:- Let Data of ptr = v

Step 14:- Let Link of ptr = Link of S

Step 15:- Let Link of S = ptr

step 16:- Return

* Deletion from the linked list :-

Since linked list is a dynamic structure deletion of node is performed physically. We can delete a node from any where within a link list. There are different algorithm from the beginning from the end and from the middle. But in any algorithm the ~~middle~~ deletion is performed by holding a node address in pointer, adjusting the links and then releasing the pointers.

* Deletion of first node — To delete the first node first of all we need to check that is there any node or ~~node~~ not, if it is then we assign pointer with the value of start and adjust the start with link the pointer before releasing it.

The algorithm is given below

Step 1:- Start Delfirst ()
Step 2:- Declare pointer ptr of node type
step 3:- If start = Null then

3.A Print "List empty"
3.B Return
Step 4:- Let ptr = start
Step 5:- Let start = link of ptr
Step 6:- Release ptr
Step 7:- Return

* Deletion of last node :- To delete the last node we need to traverse to the second last node because it holds the address of last node as its link. Then we keep the link value in a pointer and assign the link with null to indicate the end. After then we release the pointer.

Algorithm for deletion of last node

Step 1:- Start DelLast()
Step 2:- Declare pointer ptr and s of node type.
Step 3:- If start = Null Then
3.A Print "List empty"
3.B Return
Step 4:- Let s = start
Step 5:- While Link of s & Link != Null
Step 6:- Let s = link of s

Step 7 :- end of while
Step 8 :- Let ptr = link of
Step 9 :- Let Link of S = Null
Step 10 :- Release ptr
Step 11 :- Return

Deletion of node at given position

Step 1 :- start Del_Pos (Pos)
Step 2 :- Declare pointer ptr, S, a pointer C = 1, and x
Step 3 :- If Start = Null Then
3.A Print "List is empty"
3.B return with 0 (zero)
Step 4 :- end of If
Step 5 :- If Pos = 1 Then
5.A call Delfirst()
5.B Return
Step 6 :- end of If
Step 7 :- Let S = start
Step 8 :- Do step 9 while C < Pos-1 And Link of S != Null
Step 9 :- Let S = link of S
Step 10 :- end of while
Step 11 :- If C < Pos Then
11.A Print "given position not found"
11.B Return with 0
Step 12 :- end of if
Step 13 :- Let ptr = link of S

Step 14:- Let link of S = link of ptr
Step 15:- Let V = Data of ptr
Step 16:- Declare ptr
Step 17:- Return with V

Doubly linked list :-

A doubly linked list is also a collection of nodes scattered within the memory and connected together logically by holding the address of consequence nodes. The differente between SLL and DLL is that it can be traversed in two ways from the beginning this two way traversal provides a lot of advantages over the SLL It reduces a remarkable processing time particularly when we have to search a value and have to append a node.

For the two way traversal, the nodes of DLL holds before and coming after. The nodes of DLL have at least three parts - the datapart, the previous link pointer and the next link pointer. The previous