2nd International Conference on Science, Technology and Management
University of Delhi (DU), Conference Centre, New Delhi (India)
27 September 2015, www.conferenceworld.in, www.icstmdu.com
978-81-931039-6-8

# LINKED LIST IMPLEMENTATION USING C LANGUAGE: A REVIEW

**Ekta Nehra**

*Assistant Professor (Extn.), C.R.M jat college, Hisar, Haryana, (India)*

## ABSTRACT

*This paper describes about linear data structure i.e. linked list. Linked list is dynamic in nature means there is no need to know size of data in advance.It is linear collection of data elements which may or may not be stored at consecutive memory location. In linked list pointers are used for providing the linear order and each node is divided into parts for storing data and link part. This paper provides only overview that how linked list was created and what are its advantages over another data structure. It also defines how data is stored in linked list. What type of linked list can be? What basic operations can be performed on singly linked list? .And also tried to define applications of data structure. And how disadvantage of sequential search can be degrade using binary search on linked list.*

***Keywords: Data Structure, Doubly Linked List, Singly Linked List, Linear Structure***

## I. INTRODUCTION

Linked list was developed by Cliff Shaw, Allen Newell and Herbert A. Simon at RAND Corporation in year of 1955-1956. It was firstly used in information processing language for artificial intelligence.There are many disadvantages of array as compare to linked list. First is array is a static data structure and size have to define before use for example: int marks[10] here size of array have fixed of 10 elements. Second array stores only similar type of elements in integer value according to upper example it will store only integer value and third one is in array insertion and deletion operation are very difficult for performing that operations huge amount of data have to move upward and downward because in array data is stored at continues memory location. To overcome with disadvantages of array a data structure is used that is called linked list. Linked list is a dynamic data structure means according to requirement of memory it can be grow or shrink so like array there is no need to define size in advance. In it data is not stored at continuous location so insertion and deletion operations can be easily performed.

## II. DEFINITION OF LINKED LIST

In linked list data is stored in form of nodes that are also called data elements. It allows insertion and deletion in very easy way of insertion and deletion but takes time to access any element in the list[2]. It is a linear data structure means data will be accessed sequentially but may not stored at continuous location and in it linear order is given by means of pointers. Each node in liked list consists of two parts:


**Fig. 1. Node of linked list**

**1.Information field: -** it contains the data of the node.

**2.Link field/next pointer field: -** it contains reference to next node.

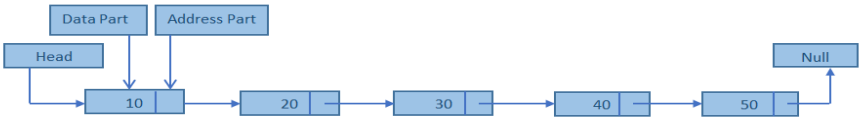


**Fig. 2. How linked list is maintained**

In linked list one pointer is used for pointing the starting node and here it is taken as head. And last node link part of linked list contains null pointer or any invalid address. In given figure 2 data is 10, 20, 40, 50 and each node link part is pointing to next node. In linked list there can be a special case that linked list have no node or start pointer contains null value and it is known as empty list or null list and linked list is stored in memory by using parallel array one for information and second for link part[1].

## III. TYPES OF LINKED LIST

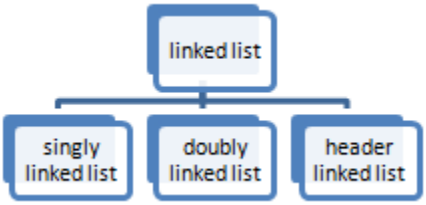


**Fig. 3. Types of Linked List**

### 3.1 Singly Linked List

It is also known as one way linked list or unidirectional list. In this list nodes can be accessed in one direction, in this list only one sided pointer is provided. In such kind of list each successive node are linked in linear order. And every element contains address of next node. Traversing can be done in forward and linear manner, no direct access of elements is allowed. It is sequence of dynamically allocated nodes. We can have multiple data element in single node but link will be always one. The last node link part and previous to first node is considered as null[4].


**Fig. 4. Singly linked list**

Here one data field that is 12,99,37 and one is link pointer and data can be accessed only in one direction.

**3.1.1Operations performed on singly linked list**

**3.1.1.1Creation of linked list**

```
void create()
{
int a,nx;
int i=1;
node *p;
p=head;
do
{
printf("/n enter element information\n");
scanf("%d",&nx);
if(i==1)
{
insert_at_begin(nx);
i=0;
}
else
insert_at_end(nx);
printf("/n enter 0 to exit or any other number to continue");
scanf("%d",&a);
}while(a!=10);
}
```

**3.1.1.2Traversing in singly linked list**

```
void trav()
{
node *p;
p=head;
printf("/n the element of list are\n");
while(p!=NULL)
 {
printf("%d /t",p->info);
p=p->next;
}
}
```

**3.1.1.3.Insertion**

There are three situations in linked list for insertion

1. Insertion at front of linked list
2. Insertion in middle of linked list
3. Insertion at the end of linked list

**3.1.1.3.1.Insertion at the front of list**

```
void insert_at_begin(int nx)
{
  node *p;
  p=(node*)malloc(sizeof(node));
  p->info=nx;
  p->next=head;
  head=p;
 }
```

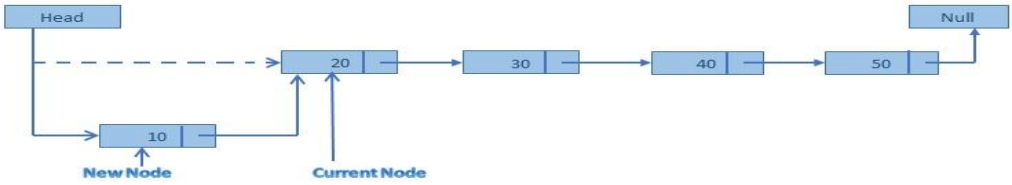


**Fig. 5. Insertion at Front of List**

**3.1.1.3.2. Insertion at the end of the list**

```
void insert_at_end(int nx)
{
 node *p,*q;
 q=head;
 p=(node*) malloc(sizeof(node));
 p->info=nx;
 p->next=NULL;
 while(q->next!=NULL)
 q=q->next;
 q->next=p;
}
```

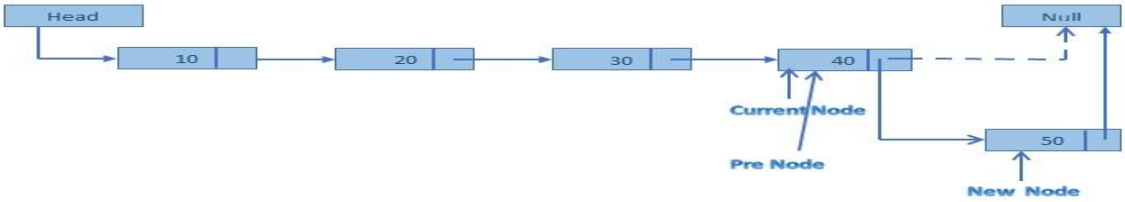


**Fig. 6. Insertion at end of list**

**3.1.1.3.2. Insertion Node in given location Linked List**

```
void insert_in_middle(int n,int nx)
{
 node *p,*q;
 p=head;
 q=(node*) malloc(sizeof(node));
 q->info=nx;
 while(p->info!=n)
 p=p->next;
 q->next=p->next;
 p->next=q;
}
```

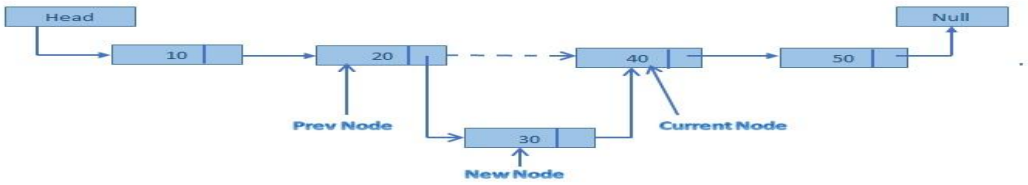


**Fig. 7. Insertion in Middle of List**

**3.1.1.4. Deletion from linked list**

There are three situations for deletion in linked list

1. From beginning
2. From middle
3. From end

```
void del(int nx)
{
node *p, *q;
q=head;
if(head==NULL)
return;
if(q->info==nx)
{
head=head->next;
free(q);
return;
}
while(q->info!=nx)
{
p=q;
```

```
q=p->next;
}
p->next= q->next;
free(q);
}
```

**Deletion Node 30 in Single Linked List**


**Fig. 8. Linked list in which node 30 is to be deleted**

**After Deletion Linked List**


**Fig. 9. Linked list after deletion**

**3.1.1.5 Concatenation of Two Linked Lists**

```
void con()
{
struct node *ptr;
ptr= f1 ;
while( ptr! = NULL)
ptr= ptr---> link;
ptr---> link = f2;
}
```

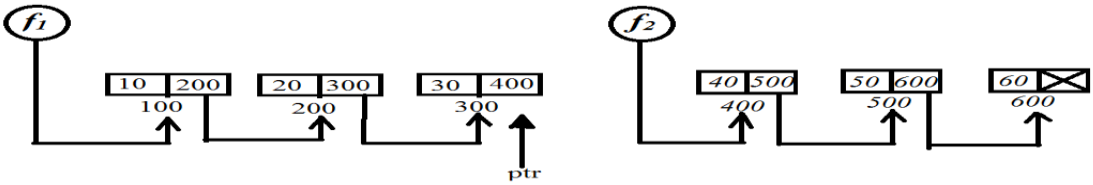


**Fig. 10. Concatenation of two linked list**

## 3.2. Doubly linked list

It is also known as two-way list. In this list each node contains two address fields. One address field is used for storing address of previous node and another is used for storing address of next node. In it accessing of elements is possible from two directions from start and from last. It is also known as bidirectional because traversing can be done in two directions. The advantage of a doubly linked list is that we don't need to keep track of the previous node for traversal or no need of traversing the whole list for finding the previous node. There is a condition that should be fulfilled for doubly linked list: for example there exist two nodes named as A and B and

LA, LB is location of A and B. the way that the pointers FORW[LOCA]=LOCB if and only if BACK[LOCB]=LOCA.It contains three parts:

1. **INFO**
2. **A pointer field FORW.**
3. **A pointer field BACK**

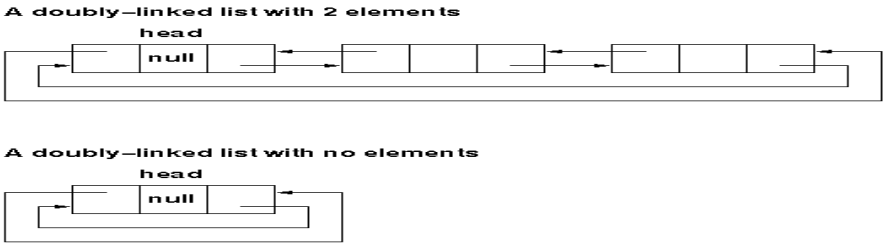


**Fig. 11. Doubly linked list**

**3.2.1. Operations on doubly linked list**

**3.2.1.1 Backward Traversing Of Doubly Linked List**

```
void traverse()
{
struct node *ptr;
ptr=first;
while(ptr--->rpt!= NULL)
ptr = ptr---> rpt
while( ptr!= NLL)
{
printf ("%d", ptr---> info);
ptr = ptr ---> lpt;
        }
```

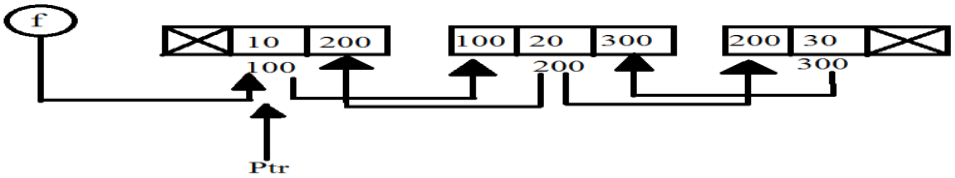


**Fig. 12. Traversing in linked list**

**3.2.1.2. Deletionfrom Doubly Linked List**

```
void delete()
{
struct node *ptr;
ptr=first;
first = ptr ---> rpt
first---> lpt = NULL
free (ptr);
```

}

### 3.3 Header linked list

It is also known as Dummy or sentinel node. Header linked list is a list in which an extra node is added in beginning of list which does not contain item of list. Information part of such node may not contain any information. It is used for handling the special case of insertion and deletion at the beginning of list[3]. Data list may contain header node but free storage list will always be maintained as an ordinary linked list. Such kinds of lists are used for maintaining polynomials in memory.In header linked list first node of the list is followed by the header node. In ordinary linked list address of first node is stored in START pointer but in header linked list it is found in LINK [START].

#### 3.3.1. Types of header linked list

1. **Grounded header linked list**: in such kind of list last node contains null pointer.
2. **Circular header linked list**: where last node point back to the header node.

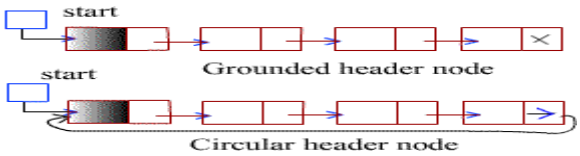


**Fig. 13**

#### 3.3.2Reason of using header node in linked list

- For inserting a node in linked list as the first node or in between two nodes, we can use only one procedure. Same will be applied in case of deletion.
- Null pointer is not used so all pointers contain valid addresses.
- Using header node there is no need to check underflow condition because list can never be empty.
- Header node may contain information about the complete file.

## IV. ADVANTAGES OF LINKED LIST

- It is dynamic data structure means new data can be added easily at run time.
- Due to its dynamic nature there is no need of knowing size in advance.
- Insertion and deletion operations can be performed easily. No need of movement of nodes.
- Stacks and queue can be maintained easily with linked list.
- In linked list space is not wasted like array.
- It is less expensive as compare to array.

## V. DISADVANTAGES OF LINKED LIST

- It requires more space as pointers are also stored along with information.
- Elements cannot be accessed randomly. If have to go to particular node then we have to go through all those nodes that come before that node.
- Elements are not stored in contiguous location so access time is more in linked list.
- Reverse traversing is difficult in linked list. in case of singly linked list it is not possible and in case of doubly linked list memory is wasted in pointers.

## VI. APPLICATIONS OF LINKED LIST

Linked list can be used many places. Some of them are discussed below:

- **Polynomials:** for maintaining polynomials in memory header linked lists are used. Header node is needed to represent the zero polynomials.
- **Sparse Array:** sparse array is an array in which nonzero elements are less than zero elements. The information part of node is divided into three fields representing the row number, column number and value of the nonzero element respectively.
- **Radix sort**
- **Linked stack**
- **Linked queue**

## VII. DIFFERENT CONCEPT IN LINKED LIST

### 7.1. Binary Search on Linked List

As all know only sequential search can be applied on linked list due to pointers used in it. But this limitation can be degrading by using sorted linked list and loop counter iterations[5]. It performs n/2 key comparisons for sequential search and O(logn) key comparisons for binary search and (n-1) loop iterations. Let us assume that X is time for key comparisons and Y is time for loop counter updating. Then formula can be given below:

$$\text{Time (sequential)} = (n/2)X$$

$$\text{Time (binary)} = (\log n)X + (n-1)Y.$$

Here are some conditions where binary search is more efficient then linear searches. So for satisfying this condition:

$$(n/2)X > (\log n)X + (n-1)Y.$$

This can be modified as:

$$(n/2 - \log n)X > (n-1)Y.$$

Let us assume that X= cY and try to find out the value of c to that point where the binary search will be more efficient then sequential search. But here is a condition that the cost of update counter should be twice less than that of key comparison. So find out the value of c as:

$$c > n-1/(n/2-\log n).$$

## VIII. CONCLUSION

Linked list is a linear data structure which store elements at run time. It is dynamic in nature means it can be grow and shrink at run time. Linked list can be future divided into many parts. In singly linked we can traverse only forward direction. In doubly linked we can traverse in both forward and backward direction but there is disadvantage that to storing two link parts with one node there is memory wastage. Header linked list is used for handling disadvantage of singly linked list of insertion and deletion. New research are being done in field of linked list like for handling disadvantage of linked list like how operations can be performed on non-blocking

linked list, how binary search can be performed on linked list and how singly linked list can be worked like doubly linked list.

## REFERENCES

[1.] Parlante, Nick (2001). "Linked list basic" . Stanford University.

[2.] Juan, Angel (2006). "Ch20 –Data Structures; ID06 - PROGRAMMING with JAVA (slide part of the book 'Big Java', by CayS. Horstmann)".

[3.] Linked Lists Plus: Complexity Trade-offs. Juniata College. Spring 2008.

[4.] WLi, S. Mohanty, and K. Kavi. "A page-based hybrid (software hardware) dynamic memory allocator," IEEE Computer Architecture Letters, 2006.

[5] Khosraviyani,F., "Using Binary Search On A Linked List", SIGCSE Bulletin, Vol.22,no 3, sept,1990.