IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁙ Impact Factor 7.918 ⁙ Vol. 11, Issue 10, October 2022

DOI: 10.17148/IJARCCE.2022.111007

Singly Linked List

Prof. Shobhana Gaikwad[1], Prof. Suwarna Nimkarde[2]

Lecturer, Computer Technology, Bharati Vidyapeeth Inst.of Tech, Navi Mumbai, India[1]

Lecturer, Computer Technology, Bharati Vidyapeeth Inst.of Tech, Navi Mumbai, India [2]

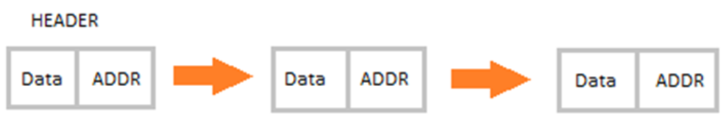
Abstract: This document gives formatting instructions for authors preparing papers for publication in the Proceedings of an International Journal. The authors must follow the instructions given in the document for the papers to be published. You can use this document as both an instruction set and as a template into which you can type your own text.

Keywords: nodes, address, data, link, head.

I. INTRODUCTION

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.

A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory. The last node of the list contains pointer to the null.

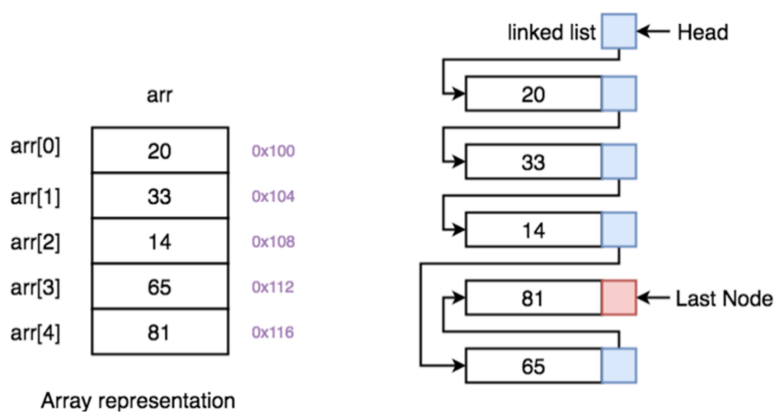


II. USES OF LINKED LIST

Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves optimized utilization of space.
- List size is limited to the memory size and doesn't need to be declared in advance.
- Empty node cannot be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Below we have a pictorial representation showing how consecutive memory locations are allocated for array, while in case of linked list random memory locations are assigned to nodes, but each node is connected to its next node using pointer.



.

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified ⁜ Impact Factor 7.918 ⁜ Vol. 11, Issue 10, October 2022
DOI: 10.17148/IJARCCE.2022.111007

Why use linked list over array?

Array contains following limitations:

- The size of array must be known in advance before using it in the program.
- Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.
- Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because,

1. It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
2. Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

III. TYPES OF LINKED LIST

Types of Linked list are as follows-

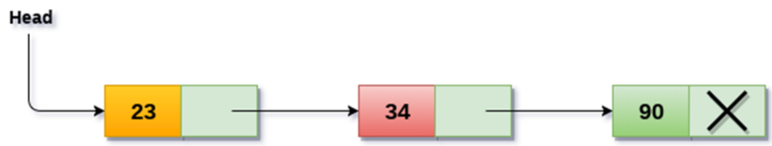
1. Singly linked list
2. Doubly Linked List
3. Circular Linked List

Singly linked list

- Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

- One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only next pointer, therefore we cannot traverse the list in the reverse direction.

- Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.



Operations on Singly Linked List

1. **Node Creation**

```
struct node
{
   int data;
   struct node *next;
};
struct node *head, *ptr;
ptr = (struct node *)malloc(sizeof(struct node *));
```

2. **Insertion**

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified ⁜ Impact Factor 7.918 ⁜ Vol. 11, Issue 10, October 2022
DOI: 10.17148/IJARCCE.2022.111007

- A node can be added in three ways:
- Insertion at the beginning of the list
- Insertion at the end of the list
- Insertion in between the nodes

Algorithm to insert an element at the beginning of linked list:

1. Start
2. Create the node pointer *temp

Struct node * temp

3. Allocate address to temp using malloc

temp = malloc(sizeof(struct node));

4. Check whether temp is null, if null then

Display "Overflow"

Else

temp-> info=data

temp-> next=start

5. Start=temp

6. stop

Algorithm to insert an element at the end of linked list:

1. Start

2. Create two node pointers *temp, *q

struct node * temp, *q;

3. q= start

4. Allocate address to temp using malloc

temp = malloc(sizeof(struct node));

5. Check whether temp is null, if null then

Display "Overflow"

else

temp-> info=data

temp-> next=null

6. While(q->next!=null)

q= q-> next

IJARCCE

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁜ Impact Factor 7.918 ⁜ Vol. 11, Issue 10, October 2022

DOI: 10.17148/IJARCCE.2022.111007

7. q->next= temp

8. stop

3. Deletion

i. Delete first node
ii. Delete last node
iii. Delete the in between node

(i) Deletion from the Beginning:

- Suppose we have four nodes in the list and we want to delete the first node.

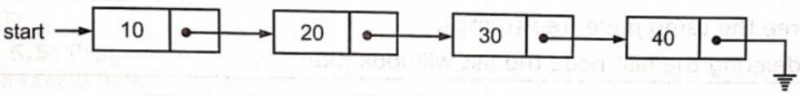


Fig. 5.38

- Now set, `q = start`

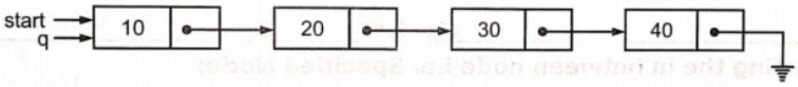


Fig. 5.39

- Now make start to points to the node where starts ptr field is pointing with the following statement.

```
start = start → ptr
```

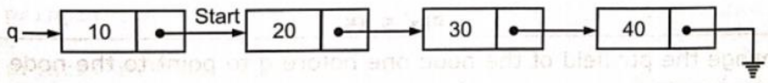


Fig. 5.40

Now make q's ptr field to point to null and then free it with the following statement.

```
q → ptr = null
free (q)
```

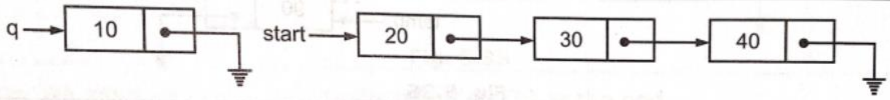


Fig. 5.41

After deleting the node the list will look like.

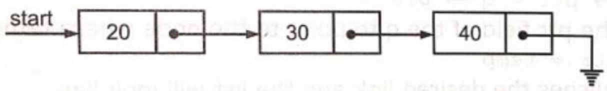


Fig. 5.42

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁜ Impact Factor 7.918 ⁜ Vol. 11, Issue 10, October 2022

DOI: 10.17148/IJARCCE.2022.111007

(ii) Deletion from the End:

- Traverse upto the second last node in the list set it as q.
- Change the ptr field of node q to the null.

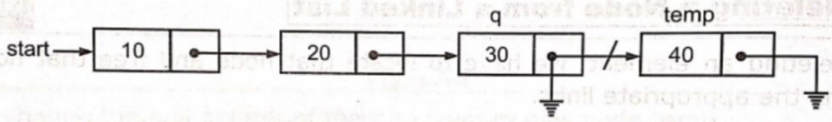


Fig. 5.43

With statement:

```
q → ptr = NULL
```

- Now free the temp node free (temp).
- After deleting the last node the list will look like:

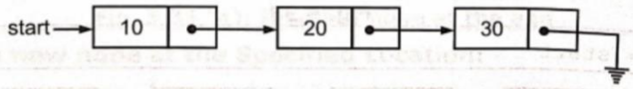


Fig. 5.44

(iii) Deleting the in between node i.e. Specified Node:

- Traverse the list till the node to be deleted is reached say q.

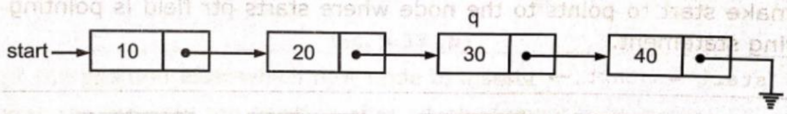
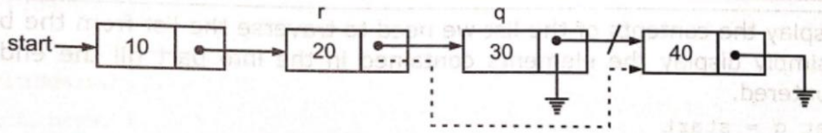


Fig. 5.45

- Now change the ptr field of the node one before q to point to the node where q's ptr field is pointing.
- After that make q's ptr field to point to null with the following statements.




```
r → ptr = q → ptr
q → ptr = null
```

Fig. 5.46

- Now free the node q free (q). After which the list will look like.

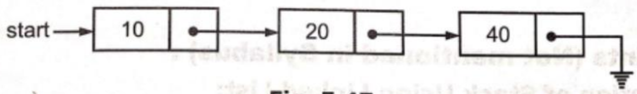


Fig. 5.47

IV. Conclusion

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

DOI: 10.17148/IJARCCE.2022.111007

Disadvantages of Linked Lists

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

REFERENCES

[1]. Data Structures Using ‘C’, Balgurusamy E,McGraw Hill Education ,New Delhi.
[2]. Data Structures Using ‘C’,ISRD Group, McGraw Hill Education ,New Delhi.
[3]. Data Structures Using ‘C’,Lipschutz, McGraw Hill Education ,New Delhi
[4]. Data Structures,Dr. Rajendra Kawale, Devraj Publications