
MODULE - IV

Linked List

Linked List

Module Description

Until now implementation of data structures were done only in arrays, which consists of contiguous memory allocation. This module covers a new way of representing and implementing data structures linked list. The chapter 4.1 covers advantages, disadvantages of array implementation and introduces the linked lists. The specifications of linked lists which include its definition, its components and representation also form part of the chapter. Moreover the chapter also covers different types of linked lists. The chapter 4.2 majorly focuses on different operations which can be performed on linked lists and their implementation.

Chapter 4.1

Introduction to Linked List

Chapter 4.2

Operations on Linked List

Chapter Table of Contents

Chapter 4.1

Introduction to Linked List

Aim.....	209
Instructional Objectives.....	209
Learning Outcomes.....	209
Introduction to Linked Lists	210
4.1.1 Linked List Specifications.....	211
(i) Definition	212
(ii) Components.....	212
(iii) Representation.....	213
(iv) Advantages and Disadvantages	214
Self-assessment Questions.....	216
4.1.2 Types of Linked Lists	217
(i) Singly Linked Lists	217
(ii) Doubly Linked Lists.....	220
(iii) Circular Linked Lists	222
Self-assessment Questions.....	224
Summary	225
Terminal Questions.....	225
Answer Keys.....	226
Activity.....	227
Case Study:	227
Bibliography.....	229
e-References	229
External Resources	229
Video Links	229



Aim

To provide the students the knowledge of Linked Lists and enable the students to write programs using linked list in C



Instructional Objectives

After completing this chapter, you should be able to:

- Explain components and representation of linked list
- Outline the advantages and disadvantages of linked list
- Differentiate singly, doubly and circular linked list
- Discuss the applications of linked list



Learning Outcomes

At the end of this chapter, you are expected to:

- Construct a table of advantages and disadvantages of linked list
- Discuss various types of linked lists
- Identify the components of all the different types of linked lists

Introduction to Linked Lists

In this chapter, we will introduce the concept of linked list data structures. This chapter focuses on how linked list can be used to overcome the limitations of array data structures. Use of linked list helps to achieve high flexibility in programming. In this chapter, we will come across the basic structure of a linked list and its various representations, types of linked lists, advantages and disadvantages associated with linked lists and its applications.

A linked list is a linear representation of interconnected nodes which consists of two components namely, data and link. The data consists of the value stored at that node and link consists of the address of the next interconnected node.

As already covered in previous chapters, the linear data structures such as stacks and queues can be implemented using arrays which allocates memory sequentially. The contiguous memory allocation of arrays provides several advantages for implementing stacks and queues as given below:

- **Faster data access:** Arrays operate on computed addresses. Therefore, direct access of data is possible which reduces the data access time.
- **Simple to understand and use:** Arrays are very simple to understand. Declaring, accessing, displaying, etc. of data from arrays is very simple. This makes implementation of stacks and queues very simple.
- **Adjacency of data:** In arrays, data are both physically and logically adjacent. Therefore loss of data elements does not affect the other part of the list.

There are also drawbacks associated with sequential allocation of data. The disadvantages of array implementation are given below:

- **Static memory allocation:** While using arrays, the compiler allocates fixed amount of memory for the program before execution begins and this allocated memory cannot be changed during its execution. It is difficult and sometimes not possible to predict the amount of memory that may be required an applications in advance. If more than the required memory is allocated to a program, and the application does not utilize it and hence results in wastage of memory space. In the same way, if less memory is allocated to a program and if application demands more memory then additional amount of memory cannot be allocated during execution.

-
- **Requirement of contiguous memory space:** For implementation of linear data structures using arrays, sufficient amount of contiguous memory is required. Sometimes even though there is memory space available in the memory it is not contiguous, which makes it of no use.
 - Insertion and deletion operations on arrays are time taking and sometimes tedious tasks.

In order to overcome the drawbacks of contiguous memory allocation, the linear data structures like linked list and arrays can be implemented using linked allocation technique. The structure in the form of linked lists can be used to implement linear data structures like stacks and queues efficiently. This mechanism can be used for many different purposes and data storage applications.

4.1.1 Linked List Specifications

Linked lists are used to implement a list of data items in some order. The linked lists structures use memory that grows and shrinks as per the requirement. Figure 4.1.1 below helps better understand linked list structure.

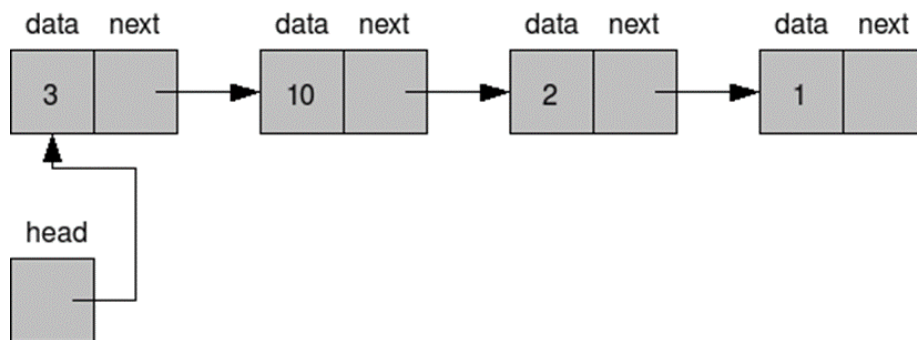


Figure 4.1.1: Linked List

Every component comprising “data” and “next” in the above figure 4.1.1 is called a node and the arrow represents the link to the next node. Head represents the starting of the linked list. The size of the data item can vary as per requirement. Linked lists can grow as much as there is memory space available.

(i) Definition

A linked representation of a data structure known as a linked list is a collection of nodes. An individual node is divided into two fields named data and link. The data field contains the information or the data to be stored by the node. The link field contains the addresses of the next node. Figure 4.1.2(a) below, demonstrates the general structure of a node in a linked list. Here, the field that holds data is termed as Info and the field that holds the address is termed as a Link. Figure 4.1.2(b) demonstrates an instance of a node. Here the info field contains an integer number 90 and the address field contains the address 2468 of the next node in a list. Figure 4.1.3 shows an *example* of linked list for integer numbers.

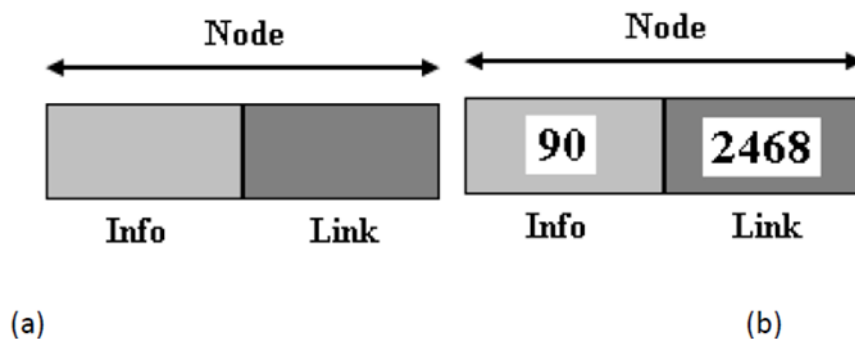


Figure 4.1.2: (a) Structure of Node (b) Instance of Node



Did you know?

Linked lists were developed in 1955–1956 by Allen Newell, Cliff Shaw and Herbert A. Simon at RAND Corporation as the primary data structure for their Information Processing Language. IPL was used by the authors to develop several early artificial intelligence programs, including the Logic Theory Machine, the General Problem Solver, and a computer chess program.

(ii) Components

As already discussed in the previous chapter, linked list basically has a collection of nodes. There, nodes individually have two components; data and link. The data component holds the actual data that the node should hold and link contains the address of the node that is next to it. In this way the linked list grows like a chain. Along with this, linked list also has a head (alternatively called start) pointer which points to the beginning or start of the linked list. The

link part of the last node of the linked list contains null which represents the end of the list. Figure 4.1.3 shows a lined list containing integer data elements.

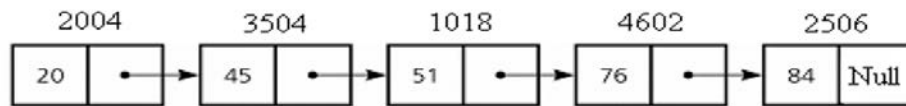


Figure 4.1.3: Linked List Containing Integer Values

(iii) Representation

One way to represent a linked list is by using two linear arrays for memory representation. Assume there are two arrays; INFO and LINK. In these two arrays, INFO[K] contains data part and LINK[k] pointer field to node k. Assume START as a variable storing the starting address of the list and NULL is a pointer indicating the end of the list.

The pictorial representation of linked list as discussed above is shown in figure 4.1.5 and figure 4.1.6.

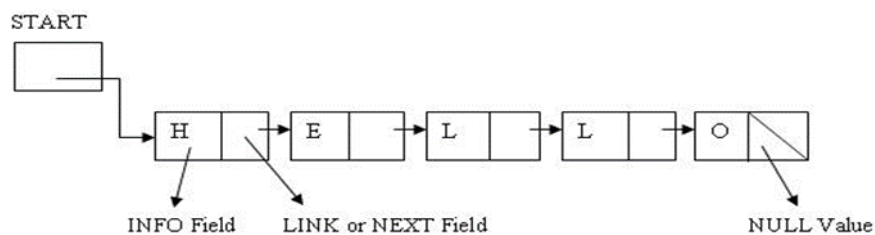


Figure 4.1.4: Representation of Linked List

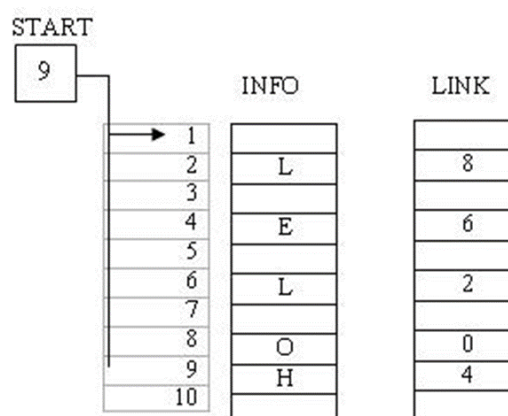


Figure 4.1.5: INFO and LINK Arrays

Here,

START = 9 => INFO[9]= H is the first character

LINK[9]= 4 => INFO[4]= E is the first character

LINK[4]= 6 => INFO[6]= L is the first character

LINK[6]= 2 => INFO[2]= L is the first character

LINK[2]= 8 => INFO[8]= 0 is the first character

LINK[8]= 0 => NULL value so the list ends

Generally a linked list node is implemented in C++ programming using a structure and allocating dynamic memory to the structure. The syntax shown below demonstrates the implementation of the node using a structure.

```
struct test_struct
{
    int val;
    struct test_struct *next;
};
```

(iv) Advantages and Disadvantages

The advantages and disadvantages of linked lists are given below:



Advantages of Linked list specifications

- **Number of elements:** First and foremost advantage of linked lists over arrays is that we need not know in advance about how many elements will form a part of list. Therefore we need not allocate memory for the linked list in advance.
- **Insertion and deletion operations:** While using linked lists, insertion and deletion operations can be performed without fixing the size of the memory in advance.
- **Memory allocation:** One of the most important advantages of linked lists over arrays is that it utilizes only the exact amount of memory required for it to store data and it can be expanded to acquire all the memory locations if needed.
- **Non-contiguous memory:** Unlike arrays, for linked list, we do not require contiguous memory allocation. We do not require elements to be stored in

consecutive memory locations. That means even if the contiguous memory block is unavailable, we can still store data.

Disadvantages of Linked list specifications

- **Pointer memory:** Use of linked list requires extra memory space as pointers are also stored with information or data. This makes implementation expensive considering memory requirement.
- **No random access:** Since we have to access nodes or elements sequentially, we do not have access to elements in linked list directly at particular node. Also sequential access makes data access time consuming depending on location of the elements.
- **Traversal:** In linked lists traversal from end of the list to beginning of the list is not possible.
- **Sorting:** Sorting elements in linked list is not as easy as the sorting operations in arrays.

The differences between static and linked allocation are mentioned below

Table 4.1.1: Differences between Static and Linked Allocation

Static Allocation Technique	Linked Allocation Technique
Memory is allocated during compile time	Memory is allocated during execution time.
The size of the memory allocated is fixed	The size of the memory allocated may vary.
Suitable for applications where data size is fixed and known in advance	Suitable for applications where data size is unpredictable.
Execution is faster	Execution is slow.
Insertion and deletion operations are strictly not defined. It can be done conceptually but inefficient way.	Insertion and deletion operations are defined and can be done more efficiently.



Self-assessment Questions

- 1) Individual node in the linked list consists of _____ number of fields.
 - a) One
 - b) Two
 - c) Three
 - d) Four

- 2) The info or data field of linked list contains _____ and link field contains _____.
 - a) Data to be stored; link to the previous node
 - b) Data to be deleted; link to the next node
 - c) Data to be stored; link to the next node
 - d) Data to be stored; link to the previous node.

- 3) The linked list allocated memory for its data elements in _____ order.
 - a) Even
 - b) Contiguous
 - c) Consecutive
 - d) Non-contiguous

- 4) Linked lists do not require additional memory space for holding pointer.
 - a) True
 - b) False

4.1.2 Types of Linked Lists

Based on the access to the list or traversal linked lists, the different types of linked lists are:

1. Singly linked lists
2. Doubly linked lists
3. Circular linked lists

(i) Singly Linked Lists

It is the simple representation of the linked list. The structure of linked lists discussed till now may be called as singly linked lists. The individual data elements in the list are called a 'node'. The elements in the lists may or may not be present in consecutive memory location. Therefore pointers are used to maintain the order of the list. Every individual node is divided into two parts called INFO and LINK. INFO is used to store data and LINK is used to store the address of the next node. Pointer START is used to indicate the starting of the linked list and NULL is used to represent the end of the list.

The following figure 4.1.6 shows the representation of a singly linked list.

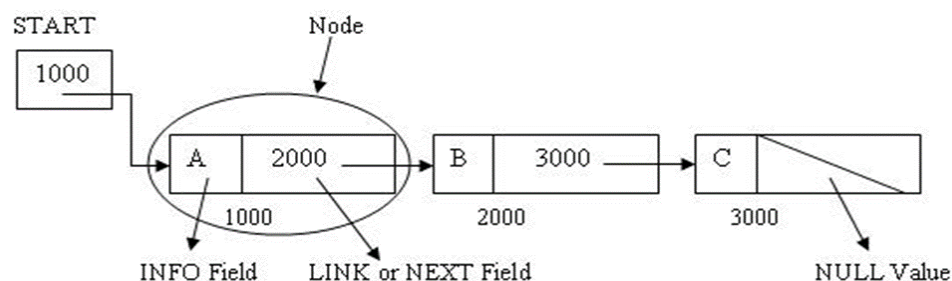


Figure 4.1.6: Singly linked lists

Representation using C programming

```
struct test_struct
{
    int val;
    struct test_struct *next;
};
```

Program to Create Singly Linked List.

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

void main()
{
    struct node
    {
        int n;
        struct node *ptr;
    };
    typedef struct node NODE;

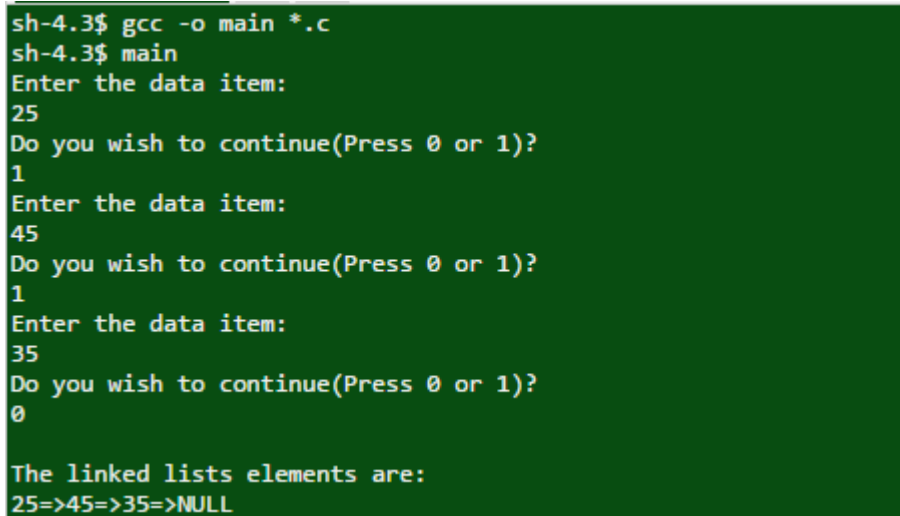
    NODE *head, *first, *temp = 0;
    int cnt = 0;
    int ch = 1;
    first = 0;

    while (choice)
    {
        head = (NODE *)malloc(sizeof(NODE));
        printf("Enter the data item:\n");
        scanf("%d", &head->n);
        if (first != 0)
        {
            temp->ptr = head;
            temp = head;
        }
        else
        {
            first = temp = head;
        }
        fflush(stdin);
        printf("Do you wish to continue(Press 0 or 1)?\n");
        scanf("%d", &ch);

    }
    temp->ptr = 0;
```

```
/* reset temp to the beginning */
temp = first;
printf("\nThe linked lists elements are:\n");
while (temp != 0)
{
    printf("%d=>", temp->n);
    cnt++;
    temp = temp -> ptr;
}
printf("NULL\n");
}
```

Output:



```
sh-4.3$ gcc -o main *.c
sh-4.3$ main
Enter the data item:
25
Do you wish to continue(Press 0 or 1)?
1
Enter the data item:
45
Do you wish to continue(Press 0 or 1)?
1
Enter the data item:
35
Do you wish to continue(Press 0 or 1)?
0

The linked lists elements are:
25=>45=>35=>NULL
```



Advantages of Singly Linked Lists

- The most obvious advantage of singly linked lists is its easy representation and simple implementation.
- Secondly, we only need to keep track of one pointer *i.e.*, forward pointer without having to bother about the previous node information.
- It is persistent data structure. A simple list of objects formed by each carrying a reference to the next in the list. This is persistent because we can take a tail of the list, meaning the last k items for some k , and add new nodes on to the front of it. The tail will not be duplicated, instead becoming shared between both the old list and the new list. So long as the contents of the tail are immutable, this sharing will be invisible to the program.

Disadvantages of Singly Linked Lists

- The very own advantage of singly linked lists of forward sequential access becomes its own drawback when we need to traverse back to previous nodes. Since singly linked list structure does not keep any information of the previous node, traversing backward is impossible.
- If we want to delete an element in the linked list and if the element to be deleted is present at the end of the list, then this becomes worst case for singly linked list as we have to check for all the nodes from beginning till the end. The worst case for this operation is $O(n)$.



Did you know?

Several operating systems developed by Technical Systems Consultants (originally of West Lafayette Indiana, and later of Chapel Hill, North Carolina) used singly linked lists as file structures. A directory entry pointed to the first sector of a file, and succeeding portions of the file were located by traversing pointers. Systems using this technique included Flex (for the Motorola 6800 CPU), mini-Flex (same CPU), and Flex9 (for the Motorola 6809 CPU). A variant developed by TSC for and marketed by Smoke Signal Broadcasting in California, used doubly linked lists in the same manner.

(ii) Doubly Linked Lists

Doubly linked lists also referred as D-lists are data structures where individual node consist of three fields. One is the usual INFO field which hold the data element. The other two are called NEXT and PREV which has addresses or links to next and previous nodes respectively.

Use of D-lists provides us an access to both sides of the list. We can traverse forward using NEXT link and backward using PREV. The figure 4.1.7 below shows the pictorial representation of doubly linked lists.

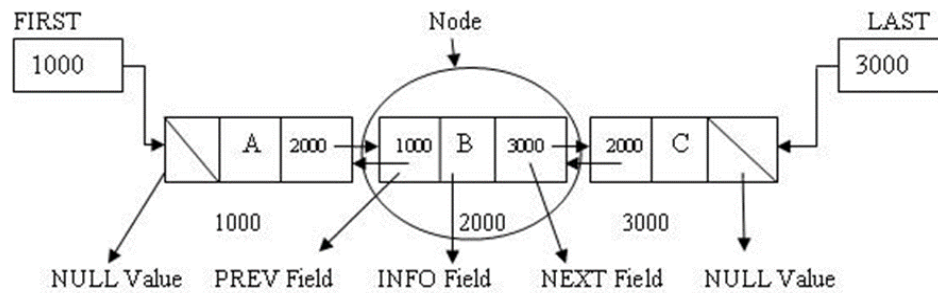


Figure 4.1.7: Doubly Linked List

Representation using a c program

```
struct test_struct
{
    int val;
    struct test_struct *next;
    Struct test_struct *prev
};
```



Advantages of Doubly linked lists

- This structure overcomes the disadvantage of singly linked list of traversing only forward by having two pointers NEXT and PREV. Using NEXT we can move forward and using PREV we can traverse backward.
- This structure also provides us a flexibility to move to any of the node from any node by moving to and fro throughout the list which is not possible in singly linked list.
- A node on a doubly linked list may be deleted with little trouble, since we have pointers to the previous and next nodes. A node on a singly linked list cannot be removed unless we have the pointer to its predecessor.



Disadvantages of Doubly linked lists

- Doubly linked list implementation require higher amount of energy as compared to singly linked list. This is because of its very own structure of having two links for moving forward and backward. The use of extra pointer for each node increases need of extra memory.

- The deletion and insertion operations on doubly linked lists are more time consuming because of the use of two pointers. It consumes a lot of time to update these pointers after each operation and require extra coding which increases time complexity.

(iii) Circular Linked Lists

The third type of linked list is circular linked list. Here, we have START, however the link of the last node is not NULL. The list part of the last node is again START. This makes it connect to START so that we can traverse back to the beginning making a round robin structure. This overcomes the drawbacks of singly linked list where one cannot move back to the first node. Figure 4.1.8 shows the structure of circular linked list.

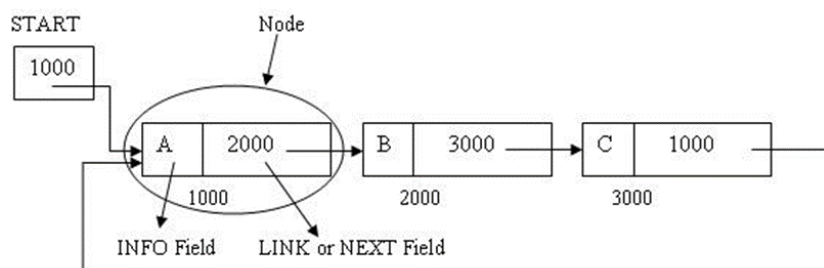


Figure 4.1.8: Circular Linked List



Advantages of Circular Linked Lists

- Circular linked list enables traversal back to the starting of the list by connecting the last node in the list back to the start. This removes the drawback of a singly linked list where one cannot access starting elements once moved forward.
- A circular linked list also eliminates the use of two pointers like that of doubly linked list, and helps us serve our purpose of traversing across the list. This result in saving a lot of memory required (saving extra pointer) for each node.
- Handling pointers in circular linked list becomes as easy as singly linked list as we have to track only one pointer that moves forward.



Disadvantages of Circular Linked Lists

- Though it helps us traverse across the linked list, the traversal to previous node is very time consuming. This is because there is no pointer that keep track of previous

node, which makes us traverse the entire linked list again and reach the node that we desire to access.

- If proper exception handling mechanism is absent, then implementation of circular lists can be dangerous as it might lead to infinite loop.
- Reversal on the list is difficult.

Applications of circular linked lists

- Implementation of waiting and context switch queues in operating system. When there are multiple processes running on operating system and there is mechanism to provide limited time slots for each process, waiting process can form a circular linked list. Task at the head of list is given CPU time, once time allocated finishes, task is taken out and added to list again at the end and it continues.
- Circular linked list is useful in implementation of queues using lists. In this, we need to have two pointers, one to point to head and other to point to end of list, because in queue, addition happens at end and removal at head. With circular list, that can be done using only one pointer.
- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.
- Multiplayer games. All the Players are kept in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.
- Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, whereas in Circular Linked List, only one pointer is required.



Self-assessment Questions

- 5) In _____ type of linked lists we can traverse in both the directions.
- a) Singly linked list
 - b) Circular linked list
 - c) One dimensional linked list
 - d) Doubly linked list
- 6) In circular linked list, the link part of the last element in the list hold the address of _____.
- a) Random node
 - b) NULL
 - c) START
 - d) Previous Node
- 7) It is possible to traverse across the list using circular lists.
- a) True
 - b) False
- 8) In singly linked list it is possible to traverse back to the START.
- a) True
 - b) False



Summary

- A linked representation of a data structure known as a linked list is a collection of nodes.
- Individual node is divided into two fields named data and link.
- The data field contains the information or the data to be stored by the node. The link field contains the addresses of the next node.
- Based on the access to the list or traversal there are three types of linked lists; Singly Linked Lists, Doubly linked lists and Circular linked lists.
- The use of linked list depends on application as there are some applications that demands sequential allocation where linked lists cannot be used.



Terminal Questions

1. Explain the advantages and disadvantages of static memory allocation.
2. Explain the advantages and disadvantages of linked lists.
3. Compare singly linked list and doubly linked lists
4. Compare singly linked list and circular linked lists



Answer Keys

Self-assessment Questions	
Question No.	Answer
1	b
2	c
3	d
4	b
5	d
6	c
7	a
8	b



Activity

Activity Type: Offline

Duration: 30 Minutes

Description:

Students should demonstrate the operations on linked list. Here few students can act as nodes and another student can act as a pointer. Students should perform operations like Insertion, Deletion on these nodes.

Students should act as nodes of singly linked list, doubly linked list, circular linked list. Operations should be performed on different types of linked list.

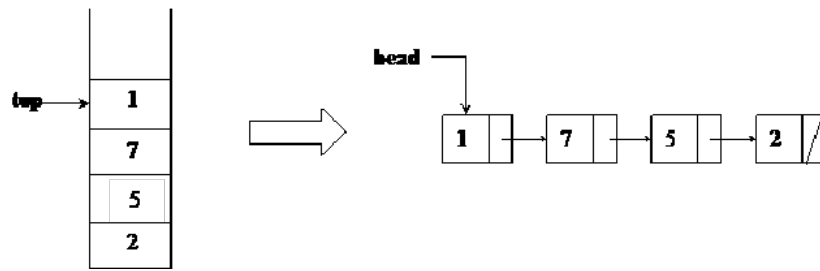
Case Study:

Stack Implementation through Linked List

We can avoid the size limitation of a stack implemented with an array, with the help of a linked list to hold the stack elements. As needed in case of array, we have to decide where to insert elements in the list and where to delete them so that push and pop will run at the fastest. Primarily, there are two operations of a stack; push() and pop(). A stack carries lifo behavior i.e. last in, first out. You know that while implementing stack with an array and to achieve lifo behavior, we used push and pop elements at the end of the array. Instead of pushing and popping elements at the beginning of the array that contains overhead of shifting elements towards right to push an element at the start and shifting elements towards left to pop an element from the start. To avoid this overhead of shifting left and right, we decided to push and pop elements at the end of the array.

Q.1) Now, if we use linked list to implement the stack, where will we push the element inside the list and from where will we pop the element?

Hint: There are few facts to consider, before we make any decision: Insertion and removal in stack takes constant time. Singly linked list can serve the purpose.



There are two parts of above figure. On the left hand, there is the stack implemented using an array. The elements present inside this stack are 1, 7, 5 and 2. The most recent element of the stack is 1. It may be removed if the `pop()` is called at this point of time. On the right side, there is the stack implemented using a linked list. This stack has four nodes inside it which are linked in such a fashion that the very first node pointed by the head pointer contains the value 1. This first node with value 1 is pointing to the node with value 7. The node with value 7 is pointing to the node with value 5 while the node with value 5 is pointing to the last node with value 2. To make a stack data structure using a linked list, we have inserted new nodes at the start of the linked list.

Q.2) Write a pseudo-code to carry out insertion and deletion operations of stack with the help of linked list.

Q.3) Will the stack implementation using linked list be cost effective?

Bibliography



e-Reference

- cs.cmu.edu, (2016). *Linked Lists*. Retrieved on 19 April 2016, from <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html>



External Resources

- Kruse, R. (2006). *Data Structures and program designing using 'C'* (2nd ed.). Pearson Education.
- Srivastava, S. K., & Srivastava, D. (2004). *Data Structures Through C in Depth* (2nd ed.). BPB Publications.
- Weiss, M. A. (2001). *Data Structures and Algorithm Analysis in C* (2nd ed.). Pearson Education.



Video Links

Topic	Link
Doubly Linked List	https://www.youtube.com/watch?v=k0pjD12bzP0
Linked Lists	https://www.youtube.com/watch?v=LOHBGyK3Hbs
Circular Linked List	https://www.youtube.com/watch?v=I4tVBFBoNSA



Notes:

