DAYANANDA SAGAR COLLEGE OF ENGINEERING

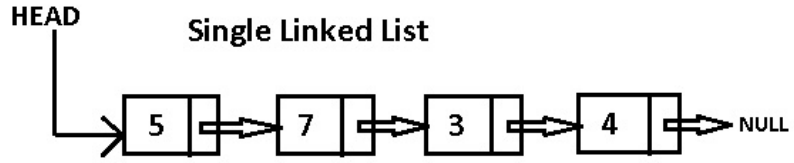
(An Autonomous Institute Affiliated to VTU, Belagavi)

Linked List

Syllabus: Definition of Linked lists and Chains, Representing Chains in C, Types of Linked List: Singly Linked List, Circular Singly Linked List, Doubly Linked Lists & Circular doubly linked list, Application of Linked List.

Self-study components: Operation on Linked list using Stacks, Queues, Polynomials

A **linked list** is a linear collection of data elements, called nodes, each pointing to the next node by mean**s** of a pointer. It is a data structure consisting of a group of nodes which together represent a sequence.



Representation of structure

```
struct node
{
 int info;
 struct node *ptr;
};
```

While declaring the structure for a linked list

Declare a structure with two members is there i.e data member and next pointer member. The data member can be character or integer or depends upon the type of the information that the linked list is having. The link member contains the address of next node.

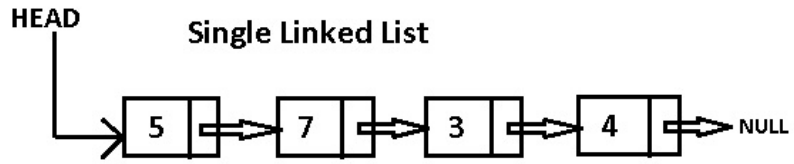
Types of linked list

1. Singly linked list (SLL)
2. Circular singly linked list (CSLL)
3. Doubly linked list (DLL)
4. Circular doubly linked list (CDLL)

Operations on linked list

1. Insert to node at the end of linked list
2. Insert to node at the front of linked list
3. Insert to node at the specified position in the linked list
4. Delete the node from the end of linked list
5. Delete the node from the beginning of linked list
6. Delete the node at the specified position from the linked list
7. Search the node in linked list
8. Implement a linked list based on stack principle
9. Implement a linked list based on queue principle
10. Count the number of nodes in the linked list.

Singly linked list (SLL)



It is called singly linked list only one link field to point to the next node. This is also called linear list because the last elements points to nothing it is linear in nature. The last node contains NULL which means that there is no node further or the last node in the list.

Lab 7. Design, develop and execute a program in C to perform the following operation on Singly linked list

a. Insert the node at the beginning
b. Insert the node at the end of the list
c. Display the list

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

struct node
{
 int info;
 struct node *link;
};
typedef struct node * NODE;
NODE first = NULL; //pointer to the first node in the list

NODE getnode()
{
        NODE temp = (NODE) malloc(sizeof(struct node));
        temp->link=NULL;
        return temp;
}
```

```
//function to add node at the beginning of the list
void insert_front()
{
	NODE temp = getnode();
	printf("\nEnter the data\n");
	scanf("%d",&temp->info);
	if(first==NULL)  //Check if list is empty
	{
		first=temp;
		return;
	}
	temp->link = first;    //add the node
	first=temp;   //update first
}

//function to add node at the end of the list
void insert_rear()
{
	NODE cur;
	NODE temp = getnode();
	printf("\nEnter the data\n");
	scanf("%d",&temp->info);
	if(first==NULL)  //Check if list is empty
	{
		first=temp;
		return;
	}
	cur=first; //cur is used to traverse through the list
	while(cur->link!=NULL)  //traverse to the last node of list
		cur=cur->link;
	cur->link=temp;  //add the node
}

//function to display list
void display()
{
	NODE cur;
	if(first==NULL) //Check if list is empty
	{
		printf("\nList is empty\n");
		return;
	}
	printf("\nList elements are\n");
	cur=first; //cur is used to traverse through the list
	while(cur!=NULL)
	{
		printf("%d\t",cur->info);
		cur=cur->link; //traverse to next node
	}
```

```
}



void main()
{
        int ch;
        clrscr();
        for(;;) //Infinite loop for menu. loop exits on choosing option 4
        {
                printf("\n1.Insert begining\n2.Insert end\n3.Display\n4.Exit\n");
                printf("Enter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:insert_front();      break;
                        case 2:insert_rear();       break;
                        case 3:display();    break;
                        default: exit(0);
                 } //end switch
        } //end for
} //end main
```

Lab 8. Write an algorithm to implement Singly Linked List using stack principles.

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

struct node
{
         int info;
        struct node *link;
};
typedef struct node * NODE;

NODE first = NULL; //pointer to the first node in the list

//function for dynamic memory allocation
NODE getnode()
{
        NODE temp = (NODE) malloc (sizeof(struct node));
        temp->link=NULL;
        return temp;
}
```

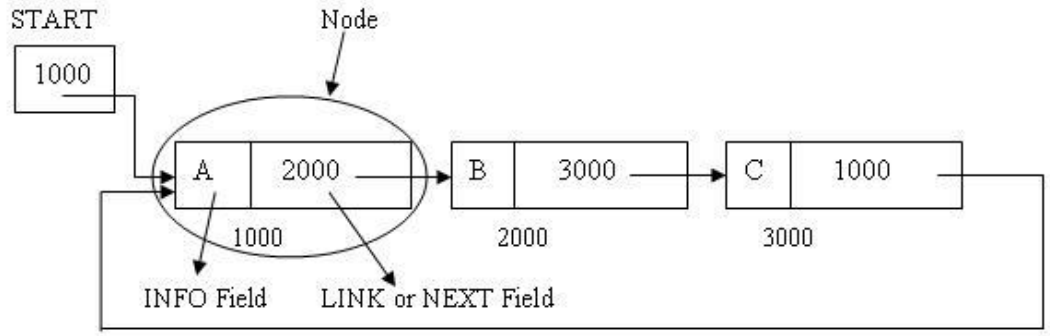
```
//function to add node at the end of the list //PUSH
void insertend()
{
	NODE cur;
	NODE temp = getnode();
	printf("\nEnter the data\n");
	scanf("%d",&temp->info);
	if(first==NULL)  //List empty OR stack empty
	{
		first=temp;
		return;
	}
	cur=first; //cur is used to traverse through the list
	while(cur->link!=NULL)  //traverse to the last node of list
		cur=cur->link;
		cur->link=temp; //add the node
}

//function to delete node at the end of the list //POP
void deleteend()
{
	NODE prev,cur;
	 if(first==NULL)  //Check if list is empty
	{
		printf("\nStack underflow\n");
		return;
	}
	else if(first->link==NULL)
	{
		printf("\nElement popped is %d\n",first->info);
		free(first);
		first=NULL;
		return;
	}
	else
	{
		prev=NULL;
		cur=first;
		while(cur->link != NULL)    //get the reference of the last two nodes
		{
			prev=cur;
			cur=cur->link;
		}
		printf("\nElement popped is %d\n",cur->info);
		free(cur);  //delete last node
		prev->link=NULL;
		return;
	}
```

```
}

//function to display list
void display()
{
        NODE cur;
        if(first==NULL) //Check if list is empty
        {
                printf("\nList is empty\n");
                return;
        }
        printf("\nList elements are\n");
        cur=first; //cur is used to traverse through the list
        while(cur!=NULL)
        {
                printf("%d\t",cur->info);
                cur=cur->link; //traverse to next node
        }
}

void main()
{
        int ch;
        clrscr();
        for(;;) //Infinite loop for menu. loop exits on choosing option 4
        {
                printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
                printf("Enter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:insertend();  break;
                        case 2:deleteend();  break;
                        case 3:display();    break;
                        default: exit(0);
                } //end switch
         } //end for
} //end main
```

Circular singly linked list (CSLL):

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

It is shown below:



Advantages of Circular Linked Lists:

1. Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2. Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

3. Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

4. Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci heap. We will soon be discussing implementation of insert delete operations for circular linked lists.

Basic Operations on a Circular Linked List
Insert – Inserts a new element at the end of the list.
Delete – Deletes any node from the list.
Find – Finds any node in the list.
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 = start. 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.

C program to perform the following task on circular singly linked list: Insert – Inserts a new element at the end of the list. Deletes any node from the list, Finds any node in the list. Prints the list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
	int data;
	struct node *ptr;
};

Typedef struct node *Node;
void insert(Node temp, int data)
{
	Node start = temp;
	while(temp->ptr!=start)
	{
		temp = temp -> ptr;
	}
	temp  = (Node) malloc (sizeof(Node));
	temp = temp->ptr;
	temp->data = data;
	temp->ptr = start;
}

int find(node *pointer, int key)
{
	node *start = pointer;
	pointer =  pointer -> next; //First node is dummy node.
	/* Iterate through the entire linked list and search for the key. */
	while(pointer!=start)
	{
		if(pointer->data == key) //key is found.
		{
			return 1;
		}
		pointer = pointer -> next;//Search in the next node.
	}
	/*Key is not found */
	return 0;
}
void delete(node *pointer, int data)
{
	node *start = pointer;
	/* Go to the node for which the node next to it has to be deleted */
	while(pointer->next!=start && (pointer->next)->data != data)
	{
		pointer = pointer -> next;
	}
	if(pointer->next==start)
```

```
	{
		printf("Element %d is not present in the list\n", data);
		return;
	}
	/* Now pointer points to a node and the node next to it has to be removed
*/
	node *temp;
	temp = pointer -> next;
	/*temp points to the node which has to be removed*/
	pointer->next = temp->next;
	/*We removed the node which is next to the pointer (which is also temp)
*/
	free(temp);
	/* Because we deleted the node, we no longer require the memory used
for it .
	  free() will deallocate the memory. */
	return;
}
void print(node *start, node *pointer)
{
	if(pointer==start)
	{
		return;
	}
	printf("%d ",pointer->data);
	print(start, pointer->next);
}

int main()
{
	/* start always points to the first node of the linked list, temp is used to
point to the last node of the linked list.*/
	node *start,*temp;
	start = (node *) malloc (sizeof(node));
	temp = start;
	temp -> next = start;
	/* Here in this code, 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.  */
	printf("1. Insert 2. Delete 3. Print 4. Find\n");
	while(1)
	{
		int ch;
		scanf("%d",&ch);
		if(query==1)
		{
			int data;
			scanf("%d", &data);
			insert(start, data);
		}
```

```
            else if(ch==2)
            {
                  int data;
                  scanf("%d", &data);
                  delete(start, data);
            }
            else if(ch==3)
            {
                  printf("The list is ");
                  print(start, start->next);
                  printf("\n");
            }
            else if(ch==4)
            {
                  int data;
                  scanf("%d", &data);
                  int status = find(start, data);
                  if(status)
                        printf("Element Found\n");
                  else
                  printf("Element Not Found\n");

             }
      }


}
```

9. Design, develop and execute a program in C to implement a doubly linked list where each node consists of integers. The program should support the following operations:

a. Create a doubly linked list by adding each node at the front
b. Insert a new node to the left of the node whose key value is read as an input.
c. Delete the node of a given data If it is found, otherwise display appropriate message
d. Display the contents of the list.

(Note: Only either (a,b and d) or (a,c and d) may be asked in the examination)
*/

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>

struct node
{
```

```
        int info;
        struct node *llink;
        struct node *rlink;
};
typedef struct node NODE;

NODE *first ,*temp,*cur,*prev,*next;

void main()
{
        int ch;
        clrscr();
        while(1)
        {
           clrscr();
             printf("..... double link list..... ");
             printf("\n1.insert front \t 2.insert before \t 3.delete all occurance “);
             printf(“\t4.display list \t 5.exit\n");
             printf("enter your choice\n");
             scanf("%d",&ch);
             switch(ch)
           {
                case 1: add_front(); break;
                case 2: insert_before(); break;
                case 3: del(); break;
                case 4: display(); break;
                case 5: exit(0);
           }
           getch();
        }
}


add_front()
{
        temp=(NODE*)malloc(sizeof(NODE));
        printf("\n enter the item");
        scanf("%d",&temp->info);
        temp->llink=NULL;
        temp->rlink=NULL;
        if(first==NULL)
        {
                first=temp;
                return;
        }
        temp->rlink=first;
        first->llink=temp;
        first=temp;
}
```

```
display()
{
	if(first==NULL)
	{
		printf("\n list is empty");
		return;
	}
	temp=first;
	printf("\n.... list status...\n");
	while(temp!=NULL)
	{
		printf("%d  ",temp->info);
		temp=temp->rlink;
	}
}

insert_before()
{
	int key;
	if(first==NULL)
	{
		printf("\n list is empty\n");
		return;
	}
	printf("\n enter the key \n");
	scanf("%d",&key);
	if(first->info==key)
	{
		add_front();
		return;
	}
	cur=first;
	while(cur!=NULL)
	{
		if(cur->info==key)
		{
			temp=(NODE*)malloc(sizeof(NODE));
			printf("\n enter item to be inserted");
			scanf("%d",&temp->info);
			prev=cur->llink;
			prev->rlink=temp;
			temp->llink=prev;
			temp->rlink=cur;
			cur->llink=temp;
			return;
		}
		cur=cur->rlink;
	}
	printf("key is not found");
}
```

```
del()
{
        int x,flag=0;
        if(first==NULL)
        {
                printf("\n list is empty\n");
                return;
        }
        printf("\n enter item to be deleted:");
        scanf("%d",&x);
        cur=first;
        while(cur!=NULL)
        {
                if(cur->info==x)
                {
                        flag=1;
                        if(cur==first)
                        first=first->rlink;
                        prev=cur->llink;
                        next=cur->rlink;
                        prev->rlink=next;
                        next->llink=prev;
                }
                cur=cur->rlink;
        }
        if(flag==1)
        printf("\n item %d deleted...\n",x);
        else
        printf("\n item %d not found\n",x);
}
```