# Problem Solving With C

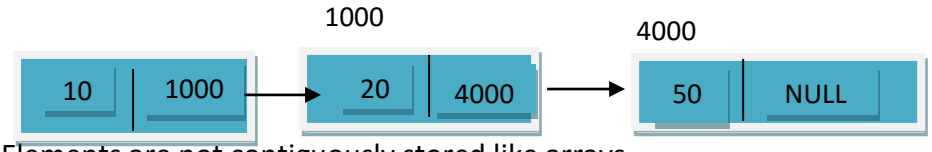
UE15CS151

# Linked List

**This Notes can be used only for reference and kindly do not solely depend on it. Only those topics which need more explanation are included here. Please Note "The prescribed Text book has to be referred for the examination"**

**LINKED LIST**

Linked list consists of group or list of nodes in which each node has data part and link part which holds the address of the next node.




- Elements are not contiguously stored like arrays.
- List can grow or shrink at runtime

**Self-referential structure**

Self-referential structure is a structure which has a pointer to itself.

```
struct node
{
        int data;
        struct node *link;
};
```

- Link(which is a pointer ) stores address of Structure node.
- This is useful in dynamic memory allocation and creating linked list

## LINKED LIST IMPLEMENTATION

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

struct node
{
        int data;
        struct node *link;
};

struct node *first;
void insert_at_front()
{
        int elem;
        struct node *p;
        printf("Enter the element\n");
        scanf("%d", &elem);
        p = malloc(sizeof(struct node));
        p -> data = elem;
```

```
	p -> link = NULL;
	if(first == NULL)
	{
		first = p;
	}
	else
	{
		p -> link = first;
		first = p;
	}
}


void insert_at_rear()
{
	int elem;
	struct node *p, *temp;
	printf("Enter the element\n");
	scanf("%d", &elem);
	p = malloc(sizeof(struct node));
	p -> data = elem;
	p -> link = NULL;
	if(first == NULL)
	{
		first = p;
	}
	else
	{
		temp = first;
		while(temp -> link != NULL)
		{
			temp = temp -> link;
		}
		temp -> link = p;
	}
}

void delete_front()
{
	struct node *temp;
	if(first == NULL)
	{
		printf("Deletion not possible\n");
	}
	else if(first -> link == NULL)
	{
		printf("%d\n", first -> data);
		free(first);
		first = NULL;
	}
```

```
	else
	{
		temp = first;
		printf("%d\n", first -> data);
		first = first -> link;
		free(temp);
		temp = NULL;
	}


}

void delete_rear()
{
	struct node *temp, *temp1;
	if(first == NULL)
	{
		printf("NP\n");
	}

	else if(first -> link == NULL)
	{
		printf("%d\n", first -> data);
		free(first);
		first = NULL;

	}
	else
	{
		temp = first;
		while(temp -> link -> link != NULL)
		{
			temp = temp -> link;
		}
	  temp1=temp->link;
		printf("Element deleted is %d\n", temp -> link -> data);
	  free(temp1);
		temp -> link = NULL;

	}
}


void delete_key()
{
	struct node *temp,*prev;
	int key;
	printf("Enter the key to be deleted\n");
	scanf("%d", &key);
	if(first == NULL)
```

```
	{
		printf("list empty\n");
		return;
	}
	if(first -> data == key)
	{
		if(first -> link == NULL)
		{
			printf("Element deleted is %d\n", first -> data);
			free(first);
		first = NULL;
		}

		else
		{
			printf("Element deleted is %d\n", first -> data);
			temp = first;
			first = first -> link;
			free(temp);

		}

		return;

	}
	temp = first;
	prev = NULL;
	while(temp -> data != key && temp != NULL)
	{
		prev = temp;
		temp = temp -> link;
		if (temp == NULL)
		{
			break;
		}
	}
	if(temp != NULL)
	{
		prev -> link = temp -> link;
		printf("Element deleted is %d\n", temp -> data);
		free(temp);

	}
	else
	{
		printf("key not found\n");
	}
	return;
}
```

```
void display()
{
	struct node *temp;
	if(first == NULL)
	{
		printf("list empty\n");
	}
	else
	{
		temp = first;
		while(temp != NULL)
		{
			printf("Element is %d\n", temp -> data);
			temp = temp -> link;
		}
	}
}
int main()
{
	int choice;
	while(1)
	{
		printf("Hit 1 to insert at front end\n");
		printf("Hit 2 to insert at rear end\n");
		printf("Hit 3 to delete at front end\n");
		printf("Hit 4 to delete at rear end\n");
		printf("Hit 5 to delete based on key\n");
		printf("Hit 6 to display\n");
		printf("Enter your choice\n");
		scanf("%d", &choice);
		switch(choice)
		{
			case 1: insert_at_front();
				break;

			case 2: insert_at_rear();
				break;

			case 3: delete_front();
				break;

			case 4: delete_rear();
				break;

			case 5: delete_key();
				break;

			case 6: display();
				break;
```

```
                }
        }
}
```

## QUEUE IMPLEMENTATION Using arrays

```
#include<stdio.h>
#include<stdlib.h>
# define size 4
int q[size], r = -1, f = -1,choice;
void insert();
void delete();
void display();
int main()
{
        system("clear");
        while(1)
        {
                printf("Hit 1 to insert\n");
                printf("Hit 2 to delete\n");
                printf("Hit 3 to display\n");
                printf("Enter your choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: insert();
                                break;
                        case 2: delete();
                                break;
                        case 3: display();
                                break;
                }
        }
}
void insert()
{
        int data;
        if (r == size - 1)
        {
                printf("Insertion not possible\n");
        }
        else
        {
                printf("Enter the data to be inserted\n");
                scanf("%d",&data);
                r = r + 1;
                q[r] = data;
                if( f== -1)
                {
                        f = f+1;
```

```
                }
        }
}

void delete()
{
        if (f > r || f == -1)
        {
                printf("Deletion not possible\n");
        }
        else
        {
                printf("Element deleted is %d\n", q[f]);
                f = f + 1;
        }
}

void display()
{
        if(f > r || f == -1)
        {
                printf("Display not possible\n");
        }
        else
        {
                for(int i = f; i<= r; i++)
                {
                        printf("%d\n", q[i]);
                }
        }

}
```

## STACK IMPLEMENTATION Using Arrays

```
#include<stdio.h>
#include<stdlib.h>
#define size 5

typedef struct stack
{
        int data[size];
        int top;
}S;


void push(S *s, int elem)
{
        if(s->top == size - 1)
```

```
	{
		printf("PUSH NOT POSSIBLE\n");
	}
	else
	{
		s->top++;
		s->data[s->top] = elem;
	}
}

int pop(S *s)
{
	int elem;
	if(s->top == -1)
	{
		printf("POP NOT POSSIBLE\n");
		return -1;
	}
	else
	{
		elem = s->data[s->top];
		s->top--;
		return elem;
	}
}

void display(S *s)
{
	if(s->top == -1)
	{
		printf("DISPLAY NOT POSSIBLE\n");
	}
	else
	{
		for(int i = s->top; i>=0; i--)
		{
			printf("%d\n", s->data[i]);
		}
	}
}

int main()
{
	int choice,elem;
	S s;
	s.top = -1;
	while(1)
	{
		printf("Hit 1 to push\n");
		printf("Hit 2 to pop\n");
```

```
                printf("Hit 3 to display\n");
                printf("Enter your choice\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("Enter the element to be pushed\n");
                                scanf("%d",&elem);
                                push(&s,elem);
                                break;
                        case 2: elem = pop(&s);
                                if(elem != -1)
                                {
                                        printf("Element popped is %d\n", elem);
                                }
                                break;
                        case 3: display(&s);
                                break;
                        default: exit(0);
                }
        }
}
```

## Queue implementation Using Linked list

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

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Enque");
   printf("\n 2 - Deque");
```

```
   printf("\n 3 - Front element");
   printf("\n 4 - Empty");
   printf("\n 5 - Exit");
   printf("\n 6 - Display");
   printf("\n 7 - Queue size");
   create();
   while (1)
   {
      printf("\n Enter choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         printf("Enter data : ");
         scanf("%d", &no);
         enq(no);
         break;
      case 2:
         deq();
         break;
      case 3:
         e = frontelement();
         if (e != 0)
            printf("Front element : %d", e);
         else
            printf("\n No front element in Queue as queue is empty");
         break;
      case 4:
         empty();
         break;
      case 5:
         exit(0);
      case 6:
         display();
         break;
      case 7:
         queuesize();
         break;
      default:
         printf("Wrong choice, Please enter correct choice  ");
         break;
      }
   }
}

/* Create an empty queue */
void create()
{
   front = rear = NULL;
```

```
}

/* Returns queue size */
void queuesize()
{
   printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
   if (rear == NULL)
   {
      rear = (struct node *)malloc(1*sizeof(struct node));
      rear->ptr = NULL;
      rear->info = data;
      front = rear;
   }
   else
   {
      temp=(struct node *)malloc(1*sizeof(struct node));
      rear->ptr = temp;
      temp->info = data;
      temp->ptr = NULL;

      rear = temp;
   }
   count++;
}

/* Displaying the queue elements */
void display()
{
   front1 = front;

   if ((front1 == NULL) && (rear == NULL))
   {
      printf("Queue is empty");
      return;
   }
   while (front1 != rear)
   {
      printf("%d ", front1->info);
      front1 = front1->ptr;
   }
   if (front1 == rear)
      printf("%d", front1->info);
}
```

```
/* Dequeing the queue */
void deq()
{
   front1 = front;

   if (front1 == NULL)
   {
      printf("\n Error: Trying to display elements from empty queue");
      return;
   }
   else
      if (front1->ptr != NULL)
      {
         front1 = front1->ptr;
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = front1;
      }
      else
      {
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = NULL;
         rear = NULL;
      }
      count--;
}

/* Returns the front element of queue */
int frontelement()
{
   if ((front != NULL) && (rear != NULL))
      return(front->info);
   else
      return 0;
}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
      printf("\n Queue empty");
   else
     printf("Queue not empty");
}
```

## Stack Implementation using Linked list

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

```
#include <stdlib.h>

struct node
{
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Push");
   printf("\n 2 - Pop");
   printf("\n 3 - Top");
   printf("\n 4 - Empty");
   printf("\n 5 - Exit");
   printf("\n 6 - Dipslay");
   printf("\n 7 - Stack Count");
   printf("\n 8 - Destroy stack");

   create();

   while (1)
   {
      printf("\n Enter choice : ");
      scanf("%d", &ch);

      switch (ch)
      {
      case 1:
         printf("Enter data : ");
         scanf("%d", &no);
         push(no);
         break;
      case 2:
         pop();
         break;
```

```
      case 3:
         if (top == NULL)
            printf("No elements in stack");
         else
         {
            e = topelement();
            printf("\n Top element : %d", e);
         }
         break;
      case 4:
         empty();
         break;
      case 5:
         exit(0);
      case 6:
         display();
         break;
      case 7:
         stack_count();
         break;
      case 8:
         destroy();
         break;
      default :
         printf(" Wrong choice, Please enter correct choice  ");
         break;
      }
   }
}

/* Create empty stack */
void create()
{
   top = NULL;
}

/* Count stack elements */
void stack_count()
{
   printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
   if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct node));
      top->ptr = NULL;
```

```
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct node));
      temp->ptr = top;
      temp->info = data;
      top = temp;
   }
   count++;
}

/* Display stack elements */
void display()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("Stack is empty");
      return;
   }

   while (top1 != NULL)
   {
      printf("%d ", top1->info);
      top1 = top1->ptr;
   }
 }

/* Pop Operation on stack */
void pop()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("\n Error : Trying to pop from empty stack");
      return;
   }
   else
      top1 = top1->ptr;
   printf("\n Popped value : %d", top->info);
   free(top);
   top = top1;
   count--;
}

/* Return top element */
```

```
int topelement()
{
   return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
   if (top == NULL)
      printf("\n Stack is empty");
   else
      printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
   top1 = top;

   while (top1 != NULL)
   {
      top1 = top->ptr;
      free(top);
      top = top1;
      top1 = top1->ptr;
   }
   free(top1);
   top = NULL;

   printf("\n All stack elements destroyed");
   count = 0;
}
```