1. **What is recursion? Write C function:**
   i)Tower of Hanoi
   ii)GCD of two numbers
   iii)Ackermann's Function. Also solve A(1,5)

Recursion is the process of repeating items in a self-similar way. If a program allows you to call a function inside the same function, then it is called a recursive call of the function.

i)

```c
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
  int num;
  printf("Enter the number of disks : ");
  scanf("%d", &num);
  printf("The sequence of moves involved in the
  Tower of Hanoi are :\n");
  towers(num, 'A', 'C', 'B');
  return 0;
}
void towers(int num, char frompeg, char topeg, char
auxpeg)
{
  if (num == 1)
  {
    printf("\nMove disk 1 from peg %c to peg %c",
    frompeg, topeg);
    return;
  }
  towers(num - 1, frompeg, auxpeg, topeg);
  printf("\nMove disk %d from peg %c to peg %c", num,
   frompeg, topeg);
  towers(num - 1, auxpeg, topeg, frompeg);
}
```

ii)

```c
#include <stdio.h>
int gcd(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);
  printf("G.C.D of %d and %d is %d.", n1, n2, gcd(n1,
   n2));
  return 0;
}
int gcd(int n1, int n2)
{
  if (n2 != 0)
    return gcd(n2, n1 % n2);
  else
    return n1;
}
```

iii)

```c
#include<stdio.h>
int A(int m, int n);
int main()
{
   int m,n;
   printf("Enter two numbers :: \n");
   scanf("%d%d",&m,&n);
   printf("\nOUTPUT :: %d\n",A(m,n));
}
int A(int m, int n)
{
   if(m==0)
      return n+1;
   else if(n==0)
      return A(m-1,1);
   else
      return A(m-1,A(m,n-1));
}
```

2. What is a Queue/Linear Queue? List different types of Queue. Write C implementation for insert() and delete operations.

Queue is a linear data structure. It follows the principle of FIFO(First In First Out). Insertion operation is performed from the REAR end. Deletion operation is performed from the FRONT end. The different types of queues are:- Simple Queue, Circular Queue, Double Ended Queue and Priority Queue.

```c
void insertQ()
{
  int num;
  printf("\n Enter the number to be inserted in the
  queue : ");
  scanf("%d", & num);
  if (rear == MAX - 1)
    printf("\n OVERFLOW");
  else if (front == -1 && rear == -1)
    front = rear = 0;
  else
    rear++;
  queue[rear] = num;
}
```

```c
int deleteQ()
{
  int val;
  if (front == -1 || front > rear)
  {
    printf("\n UNDERFLOW");
    return -1;
  }
  else
  {
    val = queue[front];
    front++;
    if (front > rear)
      front = rear = -1;
    return val;
  }
}
```

3. Write a C program to implement a circular queue using dynamic arrays.

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int cqueue_arr[MAX];
int front=-1;
int rear=-1;
void display( );
void insert(int item);
int del();
int peek();
int isEmpty();
int isFull();
int main(void)
{
  int choice,item;
  while(1)
  {
    printf("\n1.Insert\n");
    printf("2.Delete\n");
    printf("3.Peek\n");
    printf("4.Display\n");
    printf("5.Quit\n");
    printf("\nEnter your choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
```

```c
      case 1: printf("\nInput the element for insertion:");
              scanf("%d",&item);
              insert(item);
              break;
      case 2: printf("\nElement deleted is : %d\n",del());
              break;
      case 3: printf("\nElement at the front is  : %d\n",peek());
              break;
      case 4: display();
              break;
      case 5: exit(1);
      default: printf("\nWrong choice\n");
    }
  }
  return 0;
}
void insert(int item)
{
  if( isFull() )
  {
    printf("\nQueue Overflow\n");
    return;
  }
  if(front == -1 )
    front=0;
  if(rear==MAX-1)
```

```c
    rear=0;
  else
    rear=rear+1;
  cqueue_arr[rear]=item ;
}
int del()
{
  int item;
  if( isEmpty() )
  {
    printf("\nQueue Underflow\n");
    exit(1);
  }
  item=cqueue_arr[front];
  if(front==rear)
  {
    front=-1;
    rear=-1;
  }
  else if(front==MAX-1)
    front=0;
  else
    front=front+1;
return item;
}
int isEmpty()
```

```c
{
  if(front==-1)
    return 1;
  else
    return 0;
}
int isFull()
{
  if((front==0 && rear==MAX-1) || (front==rear+1))
    return 1;
  else
    return 0;
}
int peek()
{
  if( isEmpty() )
  {
    printf("\nQueue Underflow\n");
    exit(1);
  }
  return cqueue_arr[front];
}
void display()
{
  int i;
  if(isEmpty())
```

```
  {
    printf("\nQueue is empty\n");
    return;
  }
  printf("\nQueue elements :\n");
  i=front;
  if( front<=rear )
  {
    while(i<=rear)
    printf("%d ",cqueue_arr[i++]);
  }
  else
  {
    while(i<=MAX-1)
      printf("%d ",cqueue_arr[i++]);
    i=0;
    while(i<=rear)
      printf("%d ",cqueue_arr[i++]);
  }
  printf("\n");
}
```

4. Write a note on application of stacks, queues and recursion.

Stacks

- Evaluation of arithmetic expressions.
- Backtracking.
- Delimiter checking.
- Reverse a data.
- Processing function calls.

Queues

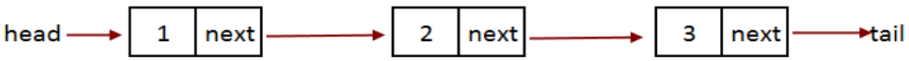
- CPU scheduling and Disk scheduling.
- Buffers, pipes i.e., when data is transferred asynchronously between two processors.
- In OS(spooling in printers, buffer for devices like keyboard).
- In networks(queues in routers/switches, mail queues).
- Variation(dequeue, priority queue).

Recursion

- TOH(Tower of Hanoi).
- GCD(Greatest Common Divisor) of two numbers.
- Ackermann function.
- Fibonacci sequence.
- Factorial of a number.

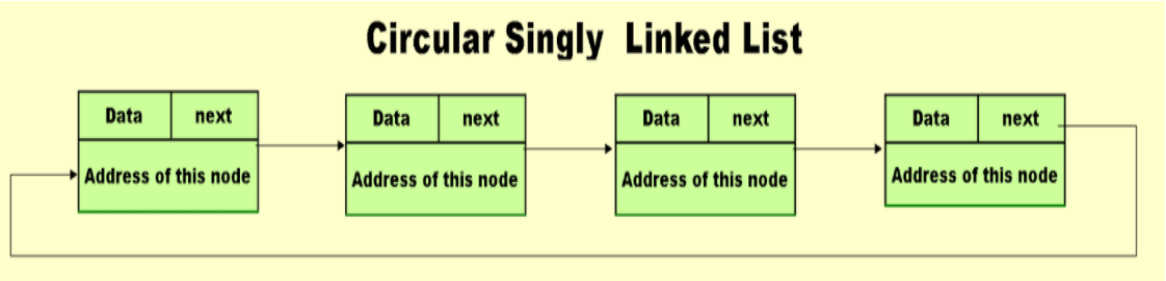
5. What is Linked List? Explain the different types of linked lists with neat diagram.

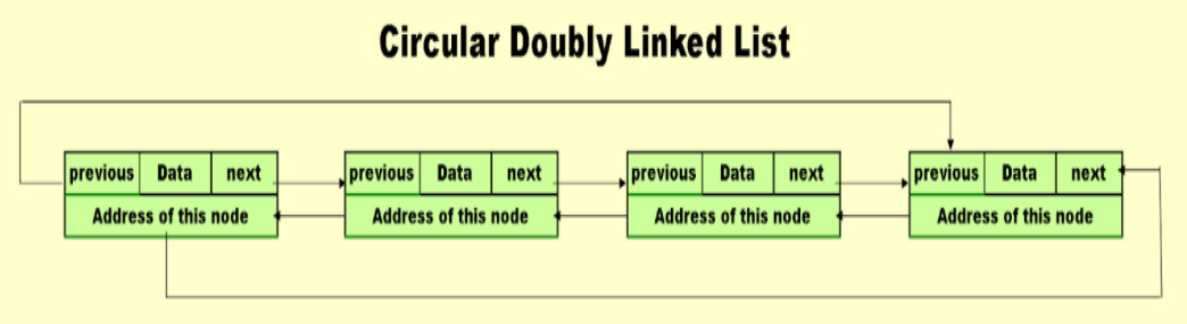
A linked list, or one-way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers. That is, each node is divided into two parts: the first part contains the information of the element and the second part, called the link field or next pointer field, contains the address of the next node in the list.



Singly Linked List

Doubly Linked List





6. **Write a C function:**
   - **a) Delete the node with given ITEM in SLL.**
   - **b) Concatenating the DLL.**
   - **c) Insert rear in DLL.**
   - **d) Insert front in SLL.**
   - **e) Delete rear in CSLL.**

a)

```
struct Node *prev = head;
while(prev->next != NULL && prev->next != n)
  prev = prev->next;
if(prev->next == NULL)
{
  printf("\n Given node is not present in Linked
  List");
  return;
}
prev->next = prev->next->next;
free(n);
return;
}
```

b)

```
struct List *merge_list(struct List *list1, struct List *list2)
{
    struct Node *hand1 = list1->head;
    struct Node *hand2 = list2->head;
    struct Node *tmp1, *tmp2 = NULL;
    struct List *list3 = malloc(sizeof(struct List));

    while(list1 && list2 != NULL)
    {
            if(ptr1->id > ptr2->id)
            {
                    ptr1 = list3->head;
                    ptr1 = ptr1->next;
            }
            else
            {
                    ptr2 = list3->head;
                    ptr2 = ptr2->next;
            }
    }
    return list3;
}
```

c)

```
void insertend()
{
        if(h==NULL)
        {
                create();
                h = temp;
                temp1 = h;
        }
        else
        {
                create();
                temp1->next = temp;
                temp->prev = temp1;
                temp1 = temp;
        }

}
```

d) void insert_atfirst()

```
{
    if (first == NULL)
    {
        create();
        first = temp;
        last = first;
    }
    else
    {
        create();
        temp->next = first;
```

```
                first = temp;
            }
        }
```

e)

```
void DeleteLast(struct Node** head)
{
    struct Node *current = *head, *temp = *head,
    *previous;
    if (*head == NULL)
    {
        printf("\nList is empty\n");
        return;
    }
    if (current->next == current)
    {
        *head = NULL;
        return;
    }
    while (current->next != *head)
    {
        previous = current;
        current = current->next;
    }
    previous->next = current->next;
    *head = previous->next;
    free(current);
    return;
}
```