Program 1:

Write a program to insert and delete an element at the n^{th} and k^{th} position in a linked list where n and k are taken from user.

Explanation:

A C program to insert and delete an element at user specified positions in a linked list.

Algorithm:

→ Take input of the elements from the user.
→ Take input of the positions at which the number is to be inserted and deleted.
→ For insertion, create a new node whose next should point to the address of the next node.
→ For deletion, take a temp variable and shift the next of the desired node to the next node.
→ Display the resultant linked list.

Code:

```
# include <stdio.h>
# include <stdlib.h>
struct node
{     int data;
      struct node *next;
} *new, *head, *tail, *temp;
void create_list()
{     int value, int ch;
      do
      {     new = (struct node *) malloc (sizeof(struct node));
            printf ("Enter a number to form a list: ");
            scanf ("%d", &value);
```

```
        new -> data = value;
        new -> next = NULL;
        if ( head == NULL)
        {
              head = new;
              tail = new;
        }
        else
        {     tail -> next = new;
              tail = new;
        }
        printf("Enter 1 to continue : ");
        scanf ("%d", &ch);
    }
    while (ch == 1);
}

// Function to insert an element at a specified position
void insert ( int in, int p)
{     int i;
      new = (struct node *)malloc(sizeof(struct node));
      temp = head;
      for(i=0; i<p-1; i++)
          temp = temp -> next;
      new -> data = in;
      new -> next = temp -> next;
      temp -> next = new;
      printf ("\n *** INSERTION SUCCESS! ***\n");
}

// Function to delete an element at a specified position
void delete (int p)
{       int i;
        new = (struct node*) malloc ( sizeof (struct node));
        temp = head;
```

```
    for(i=0; i<p-1; i++)
        temp = temp->next;
    temp-> next = temp-> next -> next;
    printf("\n***DELETION SUCCESS!***\n");
}

// Function to print the list
void display()
{
    temp = head;
    while (temp! = NULL)
    {
        printf("%d \n", temp-> data);
        temp = temp->next;
    }
}

void main()
{
    int ins, pos, pos1;
    create_list();
    printf("Enter the number to be inserted :");
    scanf("%d", &ins);
    printf("Enter the position:");
    scanf("%d", &pos);
    insert(ins, pos);
    display();
    printf("Enter the position at which number is to be deleted:");
    scanf("%d", &pos1);
    delete(pos1);
    display();
}
```

Output:

```
Enter a number to form a list : 1
Enter 1 to continue: 1
Enter a number to form a list : 2
Enter 1 to continue: 1
Enter a number to form a list : 3
Enter 1 to continue: 2
```

```
Enter the number to be inserted: 234
Enter the position: 2
*** INSERTION SUCCESS ***
1
2
234
3
Enter the position at which number is to be deleted: 3
*** DELETION SUCCESS ***
1
2
234
```

Program 2:

Construct a new linked list by merging alternate nodes of two lists.

Explanation:

Create a new linked list by joining alternate nodes of two lists.

Algorithm:

→ Take a temp node as the start of the result list.
→ The tempnode gives tail something to point to initially when the result list is empty.
→ The pointer tail always points to the last node in the result list, therefore, appending new nodes.
→ The loop proceeds, removing one node from either 'a' or 'b', and adding it ~~to~~ to tail.
→ Finally, the result is in temp.next.

Code:

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

};
```

```
// A function to print linked list
void printList(struct node * head)
{
        struct node * ptr = head;
        while (ptr)
        {   printf ("%d ", ptr-> data);
            ptr = ptr-> next;
        }
        printf ("NULL \n");

// Function to insert a new node to the linked list
void insert (struct node ** head, int data)
{
        struct node * newNode = (struct node*) malloc (sizeof (struct node));
        newNode -> data = data;
        newNode -> next = * head;
        * hedd = newNode;
}

// A function to merge alternate nodes of list using temp
void merge (struct node ** a, struct node **b)
{    struct node temp;
     struct node * tail = & temp;
     temp. next = NULL;
     while (1)
  {      if (*a == NULL)
         {   tail-> next = NULL;
             break;
         }
         else if (*b == NULL)
         {   tail -> next = *a;
             break;
         }
         else
         {   tail -> next = *a;
```

```
        tail = *a;
        *a = (*a)->next;
        tail->next = *b;
        tail = *b;
        *b = (*b)->next;
    }
  }
  *a = temp.next;
}
int main()
{
    struct node *a = NULL, *b = NULL;
    // construct first list
    for (int i=3; i>=1; i--)
        insert (&a, i);
    // construct second list
    for(int i=6; i>=4; i--)
        insert (&b,i);
    // print both linked list
    printf("First list : ");
    printList(a);
    printf("Second list: ");
    printList(b);
    merge(&a,&b);
    printf("\n After merge: \n\n");
    printf("First list : ");
    printList(a);
    printf("Second list : ");
    printList(b);
    return 0;
}
```

Output:

```
First list : 1  2  3  NULL
Second list : 4  5  6  NULL
After Merge:
First list: 1 4 2 5 3 6 NULL
Second list: NULL
```

Program 3:

Find all the elements in the stack whose sum is equal to k.

Explanation:

Find all the possible set of elements in a stack whose sum is equal to k.

Algorithm:

→ Take input of number of elements and each element value from the user.

→ Using nested loops, take the first element and store it in a variable 'i', pop 'i' from the stack and store the remaining elements in stack 2.

→ Store the first element of stack 2 in variable 'j'. Now, add i and j to check if it is equal to k.

→ If the sum is less than the expected value, we store it in a stack "store" and continue the iteration until expected value and the sum are equal.

→ Once the condition is satisfied, the elements are displayed.

Code:

```
#include<stdio.h>
#include<limits.h>
#define max 1000
typedef struct stack
{   int ar[max];
    int top;
}stack;
void push(stack *s, int data)
{    if(s->top>=max-1)
        return;
     s->top++;
     s->ar[s->top]=data;
}
```

```
int pop(stack *s)
{   if (s->top<0)
        return INT_MIN;
    s->top--=1;
    return s->ar[s->top+1];
}
void display(stack *s)
{   int i;
    for(i=s->top;i>-1;i--)
        printf("%d", s->ar[i]);
    printf("stack end\n");
}
int main(int argc, char const *argv[])
{   stack s1;
    s1.top=-1;
    int expected, i,j,k, sum, n, num;
    printf("Enter the number of elements you want in the stack\n");
    scanf("%d",&n);
    while(n--)
    {   printf("number\n");
        scanf("%d",&num);
        push(&s1, num);

    }
    printf("Enter expected value\n");
    scanf("%d",& expected);
    while((i=pop(&s1))!=INT_MIN)
    {       stack s2=s1;
            while((j=pop(&s2))!=INT_MIN)
            {       sum=i+j;
                    stack s3=s2;
                    stack store;
                    store.top=-1;
```

```
        push(&store,i);
        push(&store,j);
        while((s3.top!=-1)||(expected>=sum))
        {   if(sum==expected)
            {   display(&store);
                break;
            }
            int temp;
            if((temp=pop(&s3))==INT_MIN)
              break;
            sum+=temp;
            push(&store,temp);
        }
    }
  }
  return 0;
}
```

Output:

```
Enter the number of elements you want in the stack : 4
number : 1
number : 1
number : 2
number : 3
Enter expected value : 5
2 3  stack end
1 1 3 stack end
```

Program 4.i. :
Write a program to print the elements in a queue in reverse order.

Algorithm:

→ Take input of all the elements of queue using enqueue function.
→ In display function, do iteration from i=rear when i>=front.
→ Display the queue.

Code:

```
# include <stdio.h>
# include <stdlib.h>
# define SIZE 10
int queue[SIZE];
int front, rear = -1;
void enqueue(int add)
{    if(rear== SIZE-1)
        printf("\n*** QUEUE OVERFLOW ***\n");
     else
     {  if(front==-1)
           front=0;
        rear++;
        queue[rear]=add;
     }
}
void display()
{    int i;
     if(rear==-1 || front>rear)
        printf("\n***QUEUE IS EMPTY***\n");
     else
       for(i=rear; i>=front; i--)
          printf("%d\t", queue[i]);
}
```

```
void main ()
{   int add, ch;
    do
    {   printf ("\n Enter a number : ");
        scanf ("%d", &add);
        enqueue (add);
        printf ("\n Enter 1 to continue :");
        scanf ("%d ", &ch);
    }
    while (ch == 1);
    printf ("\n The queue displayed in reverse order :\n");
    display ();
}
```

Output:

```
Enter a number : 1
Enter 1 to continue : 1
Enter a number : 2
Enter 1 to continue : 1
Enter a number : 3
Enter 1 to continue : 2
The queue displayed in reverse order :
 3  2  1
```

Program 4.ii :

Write a program to print the elements in a queue in alternate order.

Algorithm:

→ Take input of all the elements of queue using enqueue function.

→ In display function, increment 'i' by 2 to print alternate elements.

→ Display the resultant queue.

Code:

```
# include <stdio.h>
# include<stdlib.h>
# define SIZE 10
int queue[SIZE];
int front, rear = -1;
void enqueue (int add)
{   if(rear == SIZE -1)
    printf("\n *** QUEUE OVERFLOW ***\n");
    else
    {  if (front == -1)
          front =0;
       rear ++;
       queue[rear] = add;
    }
}
void display ()
{   int i;
    if (rear == -1 || front > rear)
       printf ("\n QUEUE IS EMPTY\n");
    else
      for (i=front; i<=rear; i+=2)
          printf("%d\t", queue[i]);
}
void main ()
{   int add, ch;
    do
    {  printf("\n Enter a number: ");
       scanf ("%d", &add);
       enqueue (add);
       printf (" \n Enter 1 to continue : ");
       scanf ("%d", &ch);
    }
    while (ch==1);
    printf ("\n The queue displayed in alternate order: \n");
    display(); }
```

Output:

```
Enter a number: 1
Enter 1 to continue :1
Enter a number: 2
Enter 1 to continue :1
Enter a number: 3
Enter 1 to continue : 1
Enter a number: 4
Enter 1 to continue : 2
The queue displayed in alternate order:
  1  3
```

Program 5.i :

How array is different from the linked list?

Ans)

ARRAY	LINKED LIST
• A data structure consisting of a collection of elements each identified by array index.	A linear collection of data elements whose order is not given by their location in memory.
• supports random access, so the programmer can directly access an element in the array using index.	Supports sequential access, so the programmer has to sequentially go through each node until reaching the required element.
• Elements are stored in contiguous memory locations.	Elements can be stored anywhere in the memory.
• Programmer has to specify the size of the array at the time of declaring the array.	There is no need for specifying the size of a linked list.
• Memory allocation happens at compile time; it is a static memory allocation.	Memory allocation happens at run time; it is a dynamic memory allocation.
• Elements are independent of each other.	A node points to the next node or both next node and previous node.

Program 5.ii :

Write a program to add the first element of one list to another list.

Algorithm:

→ Take two lists, for example, {1,2,3} and {4,5,6}

→ To add the first node of one list to another, take the head pointer as return value and append it to the second list.

→ To delete the used node from previous list, shift the head pointer to the next node.

→ Display the resultant list.

Code:

```
# include<stdio.h>
# include<stdlib.h>
struct node
{    int data;
     struct node * next;
};
struct node * new (int data)
{    struct node * new_node = (struct node*) malloc(sizeof(struct node));
     new_node → data = data;
     new_node → next = NULL;
     return new_node;
}
void push(struct node ** head, int new_data)
{    struct node * new_node=(struct node *)malloc(sizeof(struct node));
     new_node → next = (* head);
     (*head) = new_node;
}
```

```
// A function to add the first node to the other list
Void insert (struct node **first, struct node**second)
{
   struct node *second_d=(struct node*)malloc(sizeof (struct node));
   second_d = * second;
   struct node * new=(struct node*) malloc (sizeof (struct node));
   new → data = second_d → data;
   new → next = * first ;
   * first =new;
   * second = second_d → next;
}

void print(struct node *Node)
{
      while( Node ! =NULL)
      {
            printf ("%d ", Node → data);
            Node = Node → next;
      }
      printf("\n");
}
int main()
{   struct node * first =NULL;
    struct node * second =NULL;
    // create first list 1→2→3
      push (&first, 3);
      push (&first, 2);
      push (&first, 1);
      printf ("The first list is ");
      print (first);

   // create second list 4→5→6
      push(& second, 6);
      push(&second, 5);
      push(&second, 4);
      printf ("Second list is ");
```

```
    print(second);
    insert(&first, &second);
    printf("After adding an element from one list to another:
            \n The first list is: ");
    print(first);
    printf("The second list is: ");
    print(second);
    return 0;
}
```

Output:

```
First list is  1 2 3
Second list is  4 5 6
After adding an element from one list to another:
The first list is: 4 1 2 3
The second list is: 5 6
```