# 1. Singly linked list:-

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

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===============================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n
        5.Delete from last\n6.Delete node after specified location\n7.Search for an
element\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            randominsert();
            break;
            case 4:
            begin_delete();
            break;
            case 5:
            last_delete();
            break;
            case 6:
            random_delete();
            break;
            case 7:
            search();
            break;
            case 8:
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
```

```
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            ptr -> next = NULL;
            head = ptr;
            printf("\nNode inserted");
        }
        else
        {
            temp = head;
            while (temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp->next = ptr;
            ptr->next = NULL;
            printf("\nNode inserted");

        }
    }
}
void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
```

```
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }

        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
}
void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}
void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
```

```
            free(ptr);
            printf("\nDeleted Node from the last ...\n");
        }
    }
    void random_delete()
    {
        struct node *ptr,*ptr1;
        int loc,i;
        printf("\n Enter the location of the node after which you want to perform deletion
    \n");
        scanf("%d",&loc);
        ptr=head;
        for(i=0;i<loc;i++)
        {
            ptr1 = ptr;
            ptr = ptr->next;

            if(ptr == NULL)
            {
                printf("\nCan't delete");
                return;
            }
        }
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted node %d ",loc+1);
    }
    void search()
    {
        struct node *ptr;
        int item,i=0,flag;
        ptr = head;
        if(ptr == NULL)
        {
            printf("\nEmpty List\n");
        }
        else
        {
            printf("\nEnter item which you want to search?\n");
            scanf("%d",&item);
            while (ptr!=NULL)
            {
                if(ptr->data == item)
                {
                    printf("item found at location %d ",i+1);
                    flag=0;
                }
                else
                {
                    flag=1;
                }
                i++;
                ptr = ptr -> next;
            }
            if(flag==1)
            {
                printf("Item not found\n");
            }
        }

    }

    void display()
```

```
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```

DS

## 2. doubly linked list :-

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
{
int choice =0;
    while(choice != 9)
    {
        printf("\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ....\n");
        printf("\n===============================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n
        5.Delete from last\n6.Delete the node after the given
data\n7.Search\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            insertion_beginning();
            break;
            case 2:
                    insertion_last();
            break;
            case 3:
            insertion_specified();
            break;
            case 4:
            deletion_beginning();
            break;
            case 5:
            deletion_last();
            break;
            case 6:
            deletion_specified();
            break;
            case 7:
            search();
            break;
            case 8:
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
```

```
    }
}
void insertion_beginning()
{
   struct node *ptr;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
       printf("\nOVERFLOW");
   }
   else
   {
    printf("\nEnter Item value");
    scanf("%d",&item);

   if(head==NULL)
   {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       head=ptr;
   }
   printf("\nNode inserted\n");
}

}
void insertion_last()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node *) malloc(sizeof(struct node));
   if(ptr == NULL)
   {
       printf("\nOVERFLOW");
   }
   else
   {
       printf("\nEnter value");
       scanf("%d",&item);
        ptr->data=item;
       if(head == NULL)
       {
           ptr->next = NULL;
           ptr->prev = NULL;
           head = ptr;
       }
       else
       {
          temp = head;
          while(temp->next!=NULL)
          {
              temp = temp->next;
          }
          temp->next = ptr;
```

```
            ptr ->prev=temp;
            ptr->next = NULL;
            }

        }
      printf("\nnode inserted\n");
     }
void insertion_specified()
{
    struct node *ptr,*temp;
    int item,loc,i;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\n OVERFLOW");
    }
    else
    {
        temp=head;
        printf("Enter the location");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\n There are less than %d elements", loc);
                return;
            }
        }
        printf("Enter value");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = temp->next;
        ptr -> prev = temp;
        temp->next = ptr;
        temp->next->prev=ptr;
        printf("\nnode inserted\n");
    }
}
void deletion_beginning()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }

}
```

```
void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        [redacted] temp -> next -> prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
void search()
```

```
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("\nitem found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("\nItem not found\n");
        }
    }

}
```

circular singly linked list



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

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 7)
    {
        printf("\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n================================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from
Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            begin_delete();
            break;
            case 4:
            last_delete();
```

```
            break;
            case 5:
            search();
            break;
            case 6:
            display();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the node data?");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }
```

```
}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp -> next != head)
            {
                temp = temp -> next;
            }
            temp -> next = ptr;
            ptr -> next = head;
        }

        printf("\nnode inserted\n");
    }

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
```

```
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }

    else
    {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");

    }
}
```

```
void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("item found at location %d",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
      }
      }
      if(flag != 0)
      {
          printf("Item not found\n");
      }
    }

}
```

```
void display()
{
  struct node *ptr;
  ptr=head;
  if(head == NULL)
  {
    printf("\nnothing to print");
  }
  else
  {
    printf("\n printing values ... \n");

    while(ptr -> next != head)
    {

      printf("%d\n", ptr -> data);
      ptr = ptr -> next;
    }
    printf("%d\n", ptr -> data);
}

}
```

4a) stack using Arrays:

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
void main ()
{

   printf("Enter the number of elements in the stack ");
   scanf("%d",&n);
   printf("*********Stack operations using array*********");

printf("\n----------------------------------------\n");
   while(choice != 4)
   {
      printf("Chose one from the below options...\n");
      printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
      printf("\n Enter your choice \n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
            break;
         }
         case 3:
         {
            show();
            break;
         }
         case 4:
         {
            printf("Exiting....");
            break;
         }
         default:
         {
            printf("Please Enter valid choice ");
```

```
            }
        };
    }
}

void push ()
{
    int val;
    if (top == n )
    printf("\n Overflow");
    else
    {
        printf("Enter the value?");
        scanf("%d",&val);
        top = top +1;
        stack[top] = val;
    }
}

void pop ()
{
    if(top == -1)
    printf("Underflow");
    else
    top = top -1;
}
void show()
{
    for (i=top;i>=0;i--)
    {
        printf("%d\n",stack[i]);
    }
    if(top == -1)
    {
        printf("Stack is empty");
    }
}
```

4b)stack using pointer:

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

#define MAX 50
int size;

// Defining the stack structure
struct stack {
   int arr[MAX];
   int top;
};

// Initializing the stack(i.e., top=-1)
void init_stk(struct stack *st) {
   st->top = -1;
}

// Entering the elements into stack
void push(struct stack *st, int num) {
   if (st->top == size - 1) {
      printf("\nStack overflow(i.e., stack full).");
      return;
   }
   st->top++;
   st->arr[st->top] = num;
}

//Deleting an element from the stack.
int pop(struct stack *st) {
   int num;
   if (st->top == -1) {
      printf("\nStack underflow(i.e., stack empty).");
      return NULL;
   }
   num = st->arr[st->top];
   st->top--;
   return num;
}

void display(struct stack *st) {
   int i;
   for (i = st->top; i >= 0; i--)
```

```
        printf("\n%d", st->arr[i]);
}
void
int main() {
   int element, opt, val;
   struct stack ptr;
   init_stk(&ptr);
   printf("\nEnter Stack Size :");
   scanf("%d", &size);
   while (1) {
      printf("\n\ntSTACK PRIMITIVE OPERATIONS");
      printf("\n1.PUSH");
      printf("\n2.POP");
      printf("\n3.DISPLAY");
      printf("\n4.QUIT");
      printf("\n");
      printf("\nEnter your option : ");
      scanf("%d", &opt);
      switch (opt) {
      case 1:
         printf("\nEnter the element into stack:");
         scanf("%d", &val);
         push(&ptr, val);
         break;
      case 2:
         element = pop(&ptr);
         printf("\nThe element popped from stack is : %d", element);
         break;
      case 3:
         printf("\nThe current stack elements are:");
         display(&ptr);
         break;
      case 4:
         exit(0);
      default:
         printf("\nEnter correct option!Try again.");
      }
   }
 return (0);
}
```