INDEX

Name Rachana. M. D Std Sec D

Roll No. Subject School/College

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.		Lab practice programs		
2.		Lab program - 1		
3.		Lab program - 2		
4.		Lab program - 3		
5.		Lab program - 4 Leetcode		
6.		Lab program - 5		
7.		Lab program - 6		
8.		Lab program - 7 Leetcode		
9.		Lab program - 8 Leetcode		
10.		Lab progam - 9 Hackerrank		
11.		Lab program - 10.		

Stack Implementation

```
# include <stdio.h>
int stack[100];
int i, j, choice = 0, n, top = -1;
void push();
void pop();
void show();
void main()
{
  printf("Enter the number of elements
                          in stack\n");
  scanf("%d", &n);

  while (choice != 4)
  {
   printf("Choose one from below options\n");
   printf("\n1. push\n2. pop\n3. show
   printf("Enter your choice\n\n4. Exit");
   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("overflow");
  else
    printf("Enter value");
    scanf("%d", &val);
    top = top+1;
```

```
    stack[top] = val;
  }
}

void pop()
{
  if(top == -1)
    print("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");
  }

}.
```

output

Enter no of elements : 2

1. push
2. pop
3. show
4. Exit

Enter the ~~data~~ choice : 1

Enter value : 2

1. push
2. pop
3 : show
4 : exit

~~Enter~~ 2

Enter value : 4

24/12/23

Week - 2

28/12/23

Infix to postfix.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 20

char s[MAX];
int top=-1;

int prec(char c)
{
  if(c=='^')
    return 5;
  else if(c=='/')
    return 4;
  else if(c=='*')
    return 3;
  else if(c=='+')
    return 2;
  else if(c=='-')
    return 1;
  else
    return -1;
}

int isEmpty()
{
  return top==-1;
}

int isFull()
{
  return top==MAX-1;
}
```

```
char peek()
{
  return s[top];
}

char pop()
{
  if (isEmpty())
  {
    printf("stack is empty\n");
    return -1;
  }
  else
  {
    char ch = s[top];
    top--;
    return ch;
  }
}
void push(char opr)
{
  if (isfull())
  {
    printf("stack is full\n");
  }
  else
  {
    top = top+1;
  }
}
```

```
void infixtopostfix (char infix[], char postfix[])
{
    int i, j;
    char ch;
    for (i=0, j=0; i<strlen(infix); i++)
    {
        ch = infix[i];
        if ((ch>='0' && ch<='9') || (ch>='A' && ch<='Z')
            || (ch>='a' && ch<='z'))
        {
            postfix[j++] = ch;
        }
        else if (ch==')')
        {
            while (!isempty() && peek()!='(')
            {
                postfix[j++] = pop();
            }
            if (!isempty && peek()!='(')
            {
                printf("Invalid expression.\n");
                return;
            }
            else
            {
                pop();
            }
        }
    }
```

```
else
{
  while( !isEmpty() && prec(ch) <=
                        prec(peek()))
  {
    postfix[j++] = pop();
  }
  push(ch);
}
}
while(!isEmpty())
{
  postfix[j++] = pop();
}
postfix[j] = '\0';
}

int main()
{
  char infix[20], postfix[20];
  printf("Enter the infix expression\n");
  scanf("%s", infix);
  infixtopostfix(infix, postfix);
  printf("postfix express : %s", postfix);
  return 0;
}
```

output

Enter infix expression

a * b + c * d

ab * cd * +

28/12/23

Date __/__/__
Page

evaluation of postfix.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int n, top = -1, t;
int val, res, ans;
int stack[100];
char expn[100];
void push(int[], int);
int pop(int[]);
int evaluate(char exp[]);

void main()
{
    printf("Enter postfix expression \n");
    scanf("%s", exp);
    ans = evaluate(expn);
    printf("Answer of above expression \n");
    printf("%d", ans);
}

int evaluate(char exp[])
{
    int op1, op2, i = 0;
    n = strlen(expn);
    while(expn[i] != '\0')
    {
        if(isdigit(expn[i]))
            push(stack, (int)(expn[i] - '0'));
```

```
else
{
    op1 = pop(stack);
    op2 = pop(stack);
    switch (expn[i])
    {
        case '+':
            res = op1 + op2;
            break;

        case '-':
            res = op1 - op2;
            break;
        case '*':
            res = op1 * op2;
            break;
        case '/':
            res = op1 / op2;
            break;
        case '%':
            res = op2 % op2;
            break;
        default:
            continue;
    }
    if (res<0)
        res = res * (-1);
        push(stack, res);
    }
    i++;
} return stack[top];
}
```

```
void push(int stack[], int val)
{
    if (top == n-1)
        printf("overflow");
    else
    {   top += 1;
        stack[top] = val;
    }
}
int pop(int stack[])
{
    if (top == -1)
        printf("underflow");
    else
    {
        val = stack[top];
        top -= 1;
        return val;
    }
}
```

output:

Sl.1 Enter a postfix expression:
12*34*45-
28/12/23

value: 9.

week - 3

11/1/24

Date __/__/__
Page

Circular Queue

```
#include <stdio.h>
#define SIZE 30

int rear = -1;
int front = -1;
int Q[SIZE];
int isfull()
{
    if (front == rear+1) || (front == 0 &&
                              rear ==
                                SIZE-1)
        return 0;
    else
        return 0-1;
}
int isEmpty()
{   if(front == -1 && rear == -1)
        return 0;
    else
        return 0-1;
}
```

```
void Enqueue (int x)
{
    int item;
    if (isfull() == 0)
    {
        printf ("Queue overflow \n");
        return;
    }
    else
    {
        if (isEmpty() == 0)
        {
            front = 0;
            rear = 0;
        }
        else
        {
            rear = (rear + 1) % SIZE;
        }
        Q[rear] = x;
    }
}
int Dequeue()
{
    int x;
    if (IsEmpty() == 0)
    {
        printf ("Queue underflow \n");
    }
    else
    {
        if (front == rear)
        {
```

```
        x = Q[front];
        front += -1;
        rear = -1;
    }
    else
    {
        x = Q[front];
        front = (front + 1) %. SIZE;
    }
    return x;
  }
}
void Display ()
{
    int i;
    if (IsEmpty() == 0)
    {
        printf("Queue is empty \n");
    }
    else
    {
        printf("Queue elements : \n");
        for (i = front; i != rear; i = (i+1) % SIZE)
        {
            printf("%d \n", Q[i]);
        }
        printf("%d \n", Q[i]);
    }
}
```

```
void main()
{
    int choice, x, b;
    while(1)
    {
        printf("Enter the number \n
                1. Enqueue\n
                2. Dequeue\n
                3. Display\n
                4. Exit\n");
        printf("Enter the choice\n");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1: Printf("Enter x\n");
                scanf("%d", &x);
                Enqueue(x);
                break;
        case 2:
                b = Dequeue()
                break;
        case 3:
                Display()
                break;
        case 4:
                exit(1);
        default:
                Printf("Invalid");
                break;
        }
    }
```

O/P

1. Enque
2. Dequeue
3. Display
4. Exit

Enter your choice
1
Enter no. the no to be inserted.
10

Enter your choice
2
10 was removed from queue

Enter you choice
3

queue elements
20

Enter your choice
4

11/1/24

Date __/__/__
Page _____

Queue Implementation

```
#include <stdio.h>
int q[50], rear=-1, front=-1, size;
void enqueue();
void dequeue();
void display()
void main()
{
    int ch;
    printf("Enter the size of the queue:");
    scanf("%d", &size);
    printf("Enter choice\n");
    printf("press 1. insert, 2. delete,
                3. Display and 4. Exit\n");

    while(ch)=4)
    {
        printf("Enter choice:");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                enqueue();
                break;

            case 2:
                dequeue()
                break;

            case 3: display()
                break;
```

4

```
void enqueue()
{
  int item;
  if (rear == size - 1)
    printf("Queue is full\n");
  else
  {
    if (front == -1)
      front = 0;
    printf("Insert an element:");
    scanf("%d", &item);
    rear += 1;
    q[rear] = item;
  }
}

void dequeue()
{
  if (front == -1 || front > rear)
    printf("Queue is empty\n");
  else
  {
    printf("Deleted element is: %d\n",
           q[front]);
    front += 1;
  }
}
```

```
void display()
{
    int i;
    if (front == -1)
        printf("Queue is empty");
    else
    {
        printf("Queue is : \n");
        for(int i=front; i<=rear; i++)
            printf("%d\t", q[i]);
        printf("\n");
    }
}
```

output :-

```
Enter the size of queue :
    50
Enter choice
    1
Queue is full.
```

11/1/24.

Singly linked list

```
#include <stdio.h>

typedef struct Node
{
    int data;
    struct node * next;
} Node;

Node * head = NULL;
void push();
void append();
void insert();
void display();
void main()
{
    int choice;
    while (1)
    {
        printf("1. Insert at beginning \t
                2. Insert at end \t
                3. Insert at position \t
                4. Display \t
                5. Exit \n");
        printf("Enter choice: \n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1: push()
                    break;
```

```
        case 2 :
                append();
                break;
        case 3 :
                insert();
                break;
        case 4 :
                display()
                break;
        default:
                printf("Exiting the program");
        }
}

void push()
{
    Node * temp = (Node *)malloc(sizeof(Node));
    int new_data;
    printf("enter data in the new node:");
    scanf("%d", &new_data);
    temp -> data = new_data;
    temp -> next = head;
    head = temp;
}

void append()
{
    Node * temp = (Node *)malloc(sizeof(Node));
    int new_data;
    printf("enter data in new node:");
    scanf("%d", &new_data);
    temp -> data = new_data;
```

```
        temp -> next = NULL;
        if (head == NULL)
        {
            head = temp;
            return;
        }
        Node * temp1 = head;
        while (temp1 -> next != NULL)
        {
            temp1 = temp1 -> next;
        }
        temp1 -> next = temp;
    }
void insert()
{
    Node * temp = (Node *) malloc (sizeof (Node));
    int new_data, pos;
    printf ("Enter data in the new node: ");
    scanf ("%d", &new_data);
    printf ("Enter position of new node: ");
    scanf ("%d", &pos);
    temp -> data = new_data;
    temp -> next = NULL;
    if (pos == 0)
    {
        temp -> next = head;
        head = temp;
        return;
    }
    Node * temp1 = head;
```

```
while(pos-- -)
{
    temp1 = temp1 → next;
}
Node * temp2 = temp1 → next;
temp → next = temp2;
temp1 → next = temp;
}

void display()
{
    Node * temp1 = head;
    while (temp1 != NULL)
    {
        printf("%d → ", temp1 → data);
        temp1 = temp1 → next;
    }
    printf("NULL\n");
}
```

O/P

1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit

Enter choice
1
Enter data in the new node = 10

Enter choice

2

Enter

Enter data in new node: 3 2

Sef
11/1/24

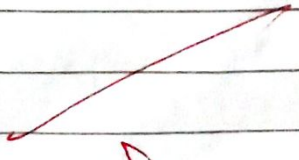
18/1/24

Week - 4

Date __/__/__
Page _____

Deletion

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

struct node
{
  int data;
  struct node *next;
};
struct node * head = NULL;

void delbegin()
{
   struct node *ptr;
   if (head == NULL)
   {
      printf("List is empty\n");
   }
   else
   {
      ptr = head;
      head = head -> next;
      free(ptr);
      printf("First element is deleted\n");
   }
}
```



NP
18/1/24

```
void dend()
{
    struct node *ptr;
    struct node *ptr1;
    if(head == NULL)
    {
        printf("List is empty\n");
    }
    else if(head->Next == NULL)
    {
        free(head);
    }
    else
    {
        Ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr
            Ptr = ptr->next;
        }
        free(ptr);
        ptr1->next = NULL.
        printf("Element at the end is
                deleted\n");
    }
}
```

```
void dpos()
{
  struct node *ptr;
  struct node *ptr1;
  int pos, i;
  printf("Enter the position from which
          data to be deleted \n");
  scanf("%d", &pos);
  ptr = head;
  if (head == NULL)
  {
    printf("List is Empty \n");
  }
  else if (head->next == NULL)
  {
    free(head);
  }
  else
  {
    for (i=0; i<pos; i++)
    {
      ptr1 = ptr;
      ptr = ptr->next;
    }
    ptr1->next = ptr->next;
    free(ptr);
    printf("Elements at the position
            %d is deleted \n", pos);
  }
}
```

```
void display()
{
  struct node * node = head;
  if (head == NULL)
  {
    printf("list is empty\n");
  }
  else
  {
    while (node != NULL)
    {
      printf("%d->", node->data);
      node = node->next;
    }
    printf("\n");
  }
}

void main()
{
  int n, i, data;
  printf("Enter the number of elements
          in linked list\n");
  scanf("%d", &n);
  printf("Enter data to be inserted\n");
  for(i=0; i<n; i++)
  {
```

```
struct node *last = head;
struct node *new_node;
new_node = (struct node*) malloc
                (sizeof(struct node));

scanf("%d", &data);
new_node->data = data;
new_node->next = NULL;
if (head == NULL)
{
    head = new_node;
}
else
{
    while (last->next != NULL)
    {
        last = last->next;
    }
    last->next = new_node;
}
}
int ch;
printf("Enter\n 1. Delete from beginning
    \n 2: Delete at the end \n
    3: Delete at pos \n 4. Display
    elements \n 5. Exit \n");
```

```
while (ch!=5)
{
  printf("Enter your choice\n");
  scanf("%d", &ch);
  switch(ch)
  {
    case 1: dstart();
            break;
    case 2: dend();
            break;
    case 3: dpos();
            break;
    case 4: display();
            break;
  }
}
```

Leetcode Minstack

Date __/__/__
Page _____

```
typedef struct
{
  int top;
  int max;
  int * minstack;
  int * data;
} minstack;

minstack * minstackCreate()
{
  minstack * st (minstack *) malloc (sizeof (
                                  minstack));
  if (st == NULL)
  {
    printf ("memory allocation failed ");
    exit(0);
  }
  st → size = 5000000;
  st → top = -1;
  st → s = (int *) malloc (st → size *
                          sizeof (int));
  st → minstack = (int *) malloc (st → size *
                          sizeof (int));
  if (st → s == NULL)
  {
    printf ("memory allocation failed");
    free (st → s);
    free (st → minstack);
  }
  return st;
}
```

```
void minstackpush (Minstack *obj, int val)
{
  if (obj->top == obj->size - 1)
      printf ("stack overflow");

  else
  {
    obj->top++;
    obj->s[obj->top] = val;
    if (obj->top == 0 || val < obj->
                          minstack[obj->top]

    { obj->minstack[obj->top] = val;
    }

    else
    {
      obj->minstack[obj->top]
                = obj->minstack[obj->
                                top-1];
    }
}}

void minstack pop (minstack *obj)
{ int value;
  if (obj->top == -1)
      printf ("underflow");

  else
  { value = obj->s[obj->top];
    obj->top--;
    printf ("%d is popped \n", value);
}}
```

```
int minstackTop (minstack *obj)
{
    int value = -1
    if (obj->top == -1)
        printf ("underflow");

    else
    {
        value = obj->s[obj->top];
        return value;
    }
}

int minstackFree (minstack * obj)
{
    free (obj->s);
    free (obj->minstack);
    free (obj);
}
```

LAB - 6 (25/01/24)

Date
Page

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

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

struct node * head = NULL;

void main()
{
    int ch;
    printf("enter the choice \n");
    scanf("%d", &ch);
    printf("Enter 1: pop \n 2: push");
    while(ch!=4)
    {
      switch(ch);
       {
          case 1:  pop();
                   break;
          case 2 : push();
                   break;
          case 3 : display();
                   break;
          case 4 : exit();

       }
    }
}
```

```
void push()
{
 int i,data;
 printf("Enter the data\n");
 struct node *last = head;
 struct node * new_nod;
 new_nod = (struct node*) malloc( sizeof
                                  (struct node));
 scanf("%d",&data);
 new_node -> data = data;
 new_node -> next = NULL;
 if(head == NULL)
 {
    head = new_node;
 }
 else
 {
    while(last -> next != NULL)
    {
        last = last -> next;
    }
    last -> next = new_node;
 }
}
void pop()
{
 struct node * ptr;
 struct node * ptr1;
 if(head == NULL)
 {
    printf("List is empty\n");
 }
```

```
else if (head -> next = NULL)
{
    free (head);
}
else
{
  ptr = head;
  while (ptr -> next != NULL)
  {
      ptr1 = ptr;
      ptr = ptr -> next;
  }
  free (ptr)
  ptr1 -> next = NULL;
  printf ("element is poped \n");
  }
}
void display()
{ struct node * node = head;
  if (head == NULL)
  {
    printf ("list is empty \n");
  }
  else
  {
    while (node != NULL)
    { printf ("%d -> ", node -> data);
      node = node -> next;
    }
    printf (\n);
  }
}
```

Date: / /
Page

Queue Implementation.

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

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

struct node *head = NULL;
void main ()
{
  printf (" Enter 1: enque \n 2: dequeue \n
                3: display ");
  printf (" enter choice \n");
  scanf ("%d", &ch);
  while (ch != 4)
  {
    switch (ch)
    {
      case 1: enqueue()
              break;

      case 2: dequeue()
              break;

      case 3: display ()
              break

      case 4 : exit();
    }
  }
```

15/2/24

```
void enqueue()
{
  int data;
  printf("enter the data to be inserted \n");
  struct node *last = head;
  struct node * new_node;
  new_node = (struct node*) malloc
             (sizeof(struct node));
  scanf("%d", &data);
  new_node->data = data;
  new_node->next = NULL;
  if (head == NULL)
  {
    head = new_node;
  }
  else
  {
    while(last->next != NULL)
    {
      last = last->next;
    }
    last->next = new_node;
  }
}
void dequeue()
{
  struct node *ptr;
  if(head == NULL)
  {
    printf("list is empty\n");
  }
```

```
else if (head -> next == NULL)
{
  free(head);
}
else
{
  ptr = head;
  head = head -> next;
  free(ptr);
  printf("an element is removed \n");
}

void display()
{
  struct node * node = head;
  if (head == NULL)
  {
    printf("list is empty \n");
  }
  else
  {
    while (node -> next != NULL)
    {
      printf("%d", node -> data);
      node = node -> next;
    }
    print("\n");
  }
}
```

output:

Enter
1: enqueue
2: dequeue
3: display
4: exit
Enter the choice
1
enter the data
1
Enter the choice
1
Enter data
2
enter the choice
3
1 → 2
Enter the choice
2
first element is deleted
enter choice
3
2
Enter choice
4

output stack:-

Enter
1. pop
2. push
3: display
4: exit

Enter the choice
2
enter the data
1
enter choice
2
enter the data
2

Date __/__/__
Page _____

Enter the choice
1
last element is deleted
enter choice
3
1

25/1/24 concatenation, sorting, reversing.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node* next;
};
struct node* head = NULL;
void main()
{
  int n, i, data;
  printf("Enter no of elements in first
          linked list\n");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
    struct node* last1 = head1;
    struct node* new_node1;
    new_node1 = (struct node*) malloc
                (sizeof(struct node));
    scanf("%d", &data);
    new_node1 -> data = data;
    new_node -> next = NULL;
    if(head1 == NULL)
    {  head1 = new_node1);
```

```
else
{
  while (last1 -> next != NULL)
  { last1 = last1 -> next;
  }
  last1 -> next = new_node;
}
printf ("elements after sorting \n");
void sort();
                          reversing
printf("elements after ~~sorting~~ \n");
reverse();
struct node *head1 = NULL;
struct node *last2 = head;
printf("enter the no of elements in
        second linked list to concatenate \n");
int n;
scanf("%d", &n);
printf("enter data \n");
for(i=0; i<n; i++)
{
  struct node * last2 = head;
  struct node * new_node2;
  new_node2 = (struct node*) malloc
              (sizeof(struct node));
  scanf("%d", &data);
  new_node2 -> data = data;
  new_node2 -> next = NULL;
  if(head == NULL)
  {
    head = new_node2;
  }
```

```
else
{
 while (last2 -> next != NULL)
 {
    last = last2 -> next;
 }
 last2 -> next = new_node2;
}
}
printf ("Elements after concatenation \n");
void concat();
}

void sort()
{
  struct node *sor = head;
  struct node *ptr = NULL;
  int temp;
  while (sor != NULL)
  {
      ptr = sor -> next;
      while (ptr != NULL)
      {
        if (sor -> data > ptr -> data)
        {
            temp = sor -> data;
            sor -> data = ptr -> data;
            ptr -> data = temp;
        }
        ptr = ptr -> next;
      }
      sor = sor -> next;
  }
```

```
struct node* node = head1;
if (head1 == NULL)
{
    printf("List is Empty \n");
}
else
{
    while (node != NULL)
    {
        printf ("%d", node->data);
        node = node->next;
    }
    printf("\n");
}}

void reverse()
{
    struct node* prev = NULL;
    struct node* ptr = NULL;
    while (head1 != NULL)
    {
        ptr = head1->next;
        head1->next = prev;
        prev = head1;
        head1 = ptr;
    }
    head1 = prev;
}
struct node* node = head1;
if (head == NULL)
{
    printf("List is empty \n");
}
```

```
else
{
while (node != NULL)
{ printf ("%d", node->data);
  node = node->next;
}
printf ("\n");
}
}
void concat()
{
struct node *temp;
if (head1 == NULL)
{
struct node * node = head2;
if (head2 == NULL)
{
printf ("List is empty\n");
}
else
{
while (node != NULL)
{
printf ("%d", node->data);
node = node->next;
}
}
else
{
while (temp->next != NULL)
{
    temp = temp->next;
}
```

```
temp->next = head2;

struct node * node = head1;
while(node != NULL)
{
  printf("%d", node->data);
  node = node->next;
}
}
```

Enter the no of elements in linked list
3

Enter the elements to be inserted for fs
3 2 1

Implementation of sort-
1 2 3

reverse
3 → 2 → 1

Enter elements in 2nd list
1 → 2 → 3

after concatenation

3 → 2 → 1 → 1 → 2 → 3 →

15/2/24

Date / /
Page

Reversed Linked List

```
struct ListNode * reverseBetween (struct
ListNode * head, int left, int right)
{
    if (head == NULL || left == right)
        return head;

    struct ListNode * dummy = (struct ListNode*)
    malloc (sizeof (struct ListNode));
    dummy → next = head;

    struct ListNode * preleft = dummy;
    for (int i=1; i<left; i++)
    {   preleft = preleft → next;
    }
    struct ListNode * curr = preleft → next;
    struct ListNode * prev = NULL;
     struct ListNode * next = NULL;
    for (i=0 ; i<=right-left; i++)
    {
        next = curr → next;
        curr → next = prev;
        Prev = current;
        current = next;
    }
    preleft → next → next = current;
    preleft → next = Prev;
    if (left == 1)
    {   head = dummy → next
    }
    free (dummy);
    return head;
}
```

Input
1, 2, 3, 4, 5
left = 2
right = 4
output
1 4 3 2 5

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

struct node
{
  int data;
  struct node * next;
  struct node * prev;
};
struct node * head = NULL;
void insert()
{
  struct node *newnode;
  int val
   int pos;
   printf("Enter position before which
          a left to be inserted \n");
   scanf("%d", &pos);
   newnode = (struct node *) malloc (sizeof (struct node));
   printf("Enter value\n");
   scanf("%d", &val);
   if (head == NULL)
   {
     new -> prv = NULL;
     new -> next = NULL;
     head = new;
     printf("Node inserted\n");
   }
```

```
else
{
    for(i=0; i<pos-1; i++)
    {
        ptr = ptr -> next;
        new -> prev = ptr -> prev;
        ptr -> prev -> next = new;
        ptr -> prev = new;
        printf("Node inserted\n");
    }
}
void delete()
{
    int val;
    printf("Enter the value:");
    scanf("%d", &val);
    struct node *ptr = head
    if(head -> data == val)
    {
        head = ptr -> next;
        free(ptr);
        printf("Node deleted\n");
        return;
    }
    while(ptr -> data != val)
    {   ptr = ptr -> next;
        if(ptr -> next == NULL)
        {
            ptr -> prev -> next = NULL;
            free(ptr);
            printf("Node deleted\n");
            return;
        }
    }
```

```
    ptr->prev->next = ptr->next;
    ptr->next->prev = ptr->next;
    free(ptr);
    printf("Node deleted\n");
}

void display()
{
    struct node *p = head;
    while(p!=NULL)
    {
        printf("%d->", p->data);
        p = p->next;
    }
    printf("NULL\n");
}

void main()
{
    int ch;
    printf("Enter 1.Insert 2.Delete
                3.Display 4.Exit");
    while(ch!=4)
    {
        printf("Enter choice\n");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: insert_left();
                    break;
```

```
case 2: delete ();
        break;

case 3 : display();
        break;

case 4 : exit()

}

}

}
```

output:

```
enter 1 to insert left
enter 2 to delete
enter 3 to display
enter 4 to exit.

enter choice
  1
enter data
  1
enter choice
  1
enter data
  2
enter choice
  1
enter data
  3

enter choice
  2
enter value to be deleted
  2
Node deleted

enter choice
  3
3 → 1
```

leet code

```
struct listnode
{
    int val;
    struct listNode * next;
};

struct listNode *rotateRight (struct listNode* head,
                              int k)
{

struct listnode * ptr = head;
int count = 1;
if (head == NULL || head → next == NULL || k == 0)
{
    return head;
}
else
{
    while (ptr → next != NULL)
    {
      ptr = ptr → next;
      count ++;
    }
  if (k % count == 0)
  {  return head;
  }
  Ptr → next = head;
```

```
for ( int i = ky.count ; i<count ; i++)
{
    ptr = ptr → next ;
}

head = ptr → next ;
ptr → next = NULL ;
return head ;

}

}
```

Sol 1

week-1

Binary search Tree

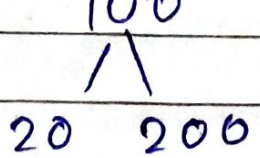
```
#include <stdio.h>
#include <stdlib.h>
struct node
{ int data;
   struct node * left;
   struct node * right;
};
struct node *root = NULL;
void creat ( int val)
{ int val;
  struct node * ptr, * ptr1, * new_node;

  new_node = (struct node *) malloc (
                              size of (struct node));

     new_node → data = val;
     new_node → left = NULL;
     new_node → right = NULL;

     if (root == NULL)
     {
        root = new_nod;
     }
```

```
void preorder ( struct node * )
{
        printf ("%d", root → data);
        preorder ( root → left);
        preorder ( root → right);
}

void postorder (struct node * )
{
      Postorder (root → left);
      postorder (root → right);
      printf ("%d", root → data);
}
void inorder ( struct node * )
{
        inorder (root → left);
        printf ("%d", root → data);
        inorder (root → right);
}

void main ()
{
      int ch;
      while ( ch != 5)
      {
            printf (" 1 : creat
                    2 : postorder
                    3 : pre
                    4 : inorder ");
```

```
Else
{
    while (root != NULL)
    {
        Ptr1 = NULL;
        Ptr = root;
        while ( Ptr != NULL)
        {
            Ptr1 = Ptr;
            if (val < Ptr -> data)
                ptr = ptr -> left;
            else
                Ptr = Ptr -> right
        }
    }
    if ( val < Ptr1 -> data)
        ptr1 -> left = new node;
    else
        Ptr1 -> right = new node;
}
return root;
}
}
```

```
scanf ("%d",&ch);
switch (ch)
{
    case 1 : create ()
             break;
    case 2 : postorder (root);
             break;
    case 3 : preorder (root);
             break;
    case 4 : inorder (root);
             break;
}
}
```

Enter your choice
1
Enter the value
100
Enter value
20
Enter value
200



Enter your choice
2

choice 20 → 200 → 100
20 → 100 → 200

Enter choice
3

100 → 20 → 200

22/2/24

BFS

```
#include <stdio.h>
int queue[100];
int front = 0, back = 0;
void enqueue (int var)
{
    queue[back] = var;
    back++;
}

void dequeue()
{
    queue[front]=0;
    front++;
}
int main()
{
  int n;
  printf("enter the number of vertices\n");
  scanf("%d", &n);
  int graph[10][10];
  int visited[10] = {0};
  printf("enter adjacency matrix");
  for(int i=0; i<n; i++)
  {
    for(int j=0; j<n; j++)
    {
        scanf("%d", &graph[i][j]);
    }
}
```

```
enqueue(1);
visited[0] = 1;
while(front != back)
{
    int current = queue[front];
    printf("%d", current);
    dequeue();
    for(int i=0; i<n; i++)
    {
        if((graph[current-1][i] == 1)
                && (visited[i] == 0))
        {
            visited[i] = 1;
            enqueue(i+1);
        }
    }
}
return 0;
}
```

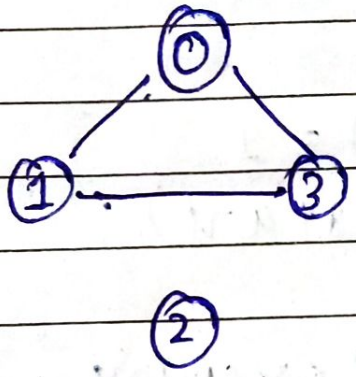
0 1 0 1

enter number of vertices.

4

Enter adjacency matrix

	0	1	2	3
0	0	1	0	1
1	1	0	0	1
2	0	0	0	0
3	1	1	0	0



output

1 0 3

Sp.T

dfs

```
#include <stdio.h>
#define MAX 100

int visited[MAX];

void dfs(int adjMatrix[][MAX], int vcount,
                                int start)
{
    visited[start] = 1
    printf("%d", start);

    for(int i=0; i<vcount; i++)
    {
        if(adjMatrix[start][i] && !visited[i])
        {
            dfs(adjMatrix, vcount, i);
        }
    }
}

int main()
{
    int adjMatrix[MAX][MAX]
    int vcount;
    printf("Enter no of vertices");
    scanf("%d", &vcount);
```

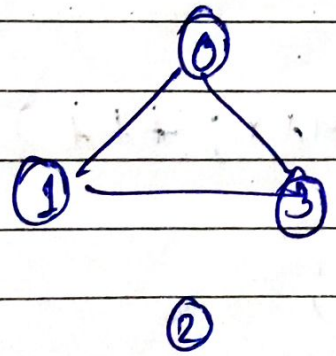
```
printf("enter adjacency matrix\n");
for(int i=0; i<MAX; i++)
{
    for(int j=0; j<MAX; j++)
    {
        scanf("%d", &adjMatrix[i][j]);
    }
}

int start;

for(int i=0; i<MAX; ++i)
{
    visited[i]=0;
}
printf("DFS Traversal: ");
dfs(adjMatrix, vcount, start);

return 0;
}
```

enter no of vertices

4

enter adjacency Matrix

	0	1	2	3
0	0	1	0	1
1	1	0	0	1
2	0	0	0	0
3	1	1	0	0



output

1 0 3

week - 10

linear probing

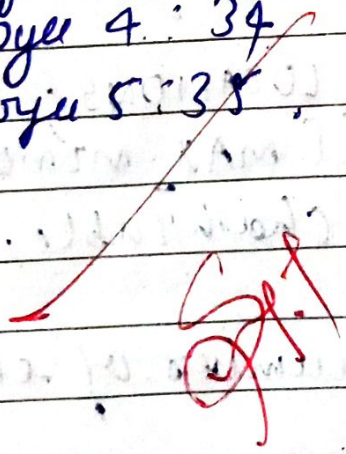
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_EMPLOYEES 100
#define MAX_MEMORY_LOCATIONS 10

typedef struct
{
    int key;
} Employee;

typedef struct
{
    Employee data;
    int status;
} HashNode;

void initialiseHashTable (HashNode hashTable[], int size
{
    for(int i=0; i<size; i++)
    {
        hashTable[i].status = 0;
    }
}

int hashFunction(int key, int size)
{
    return key % size;
}
```

```
int linearprobe(int hashvalue, int size)
{
    return (hashvalue + attempt) % size;
}

void insertEmployee(HashNode hashTable[],
    int size, Employee emp)
{
    int hashvalue = hashFunction(emp.key,
                                 size);
    int index = hashvalue;
    int attempt = 1;

    while(hashTable[index].status == 1)
    {
        index = linearprobe(hashValue,
                            attempt, size);
        attempt++;
    }

    hashTable[index].data = emp;
    hashTable[index].status = 1;

void displayHashTable(HashNode hashTable[],
                      int size)
{
    printf("\n Hash table\n");
    printf("Index\t key\n");
```

```
for(int i=0; i<size; i++)
{
printf("%d\t", i);
if(hashTable[i].status == 1)
{
    printf("%d\n", hashTable[i].data.key
}
else
{
printf("Empty\n");
}
}

int main()
{
int m = MAX_MEMORY_LOCATIONS;
HashNode hashTable[MAX_MEMORY_LOCATIONS
initialiseHashTable(hashTable, m);
int n;
printf("Enter the number of employee
                                records:");
scanf("%d", &n);
Employee employees[MAX_EMPLOYEES];
printf("Enter the employee records:\n");
for(int i=0; i<n; i++)
{
    printf("employee %d: ", i+1);
    scanf("%d", &employees[i].key);
}
```

```
for(int i = 0; i < n; i++)
{
    insertEmployee(hashTable, m,
                   employees[i]);
}
displayHashTable(hashtable, m);
return 0;
}
```

```
Enter the number of employee records: 5
Enter the employee keys:
Employee 1: 1
Employee 2: 3
Employee 3: 5
Employee 4: 34
Employee 5: 38
```



HackerRank

```
typedef struct TreeNode
{
  int data;
  struct TreeNode * left;
  struct TreeNode * right;
} TreeNode;

void inorderTraversal (TreeNode* root, int *
                       result, int * index)
{
   if (root == NULL)
      return;

   inorderTraversal (root->left, result, index
   result[(*index)++] = root->data;
   inOrderTraversal (root->right, result, inde

}
void swapsubtrees (TreeNode *root, int k, int
                                        depth)
{  if (root == NULL)
      return;

   if (depth % K == 0)
   {
     TreeNode * temp = root->left;
     root->left = root->right;
     root->right = temp;
   }
```

```
swapsubtrees(root->left, k, depth+1);
swapsubtrees(root->right, k, depth+1);

TreeNode* buildTree(int indexesRows,
                    int indexesColumns,
                    int** indexes)
{
    TreeNode* root = (TreeNode*)
        malloc(sizeof
            (TreeNode))

    root->data = 1;
    root->left = NULL;
    root->right = NULL;
    TreeNode* nodes[indexesRows+1];
    nodes[1] = root;

    for(int i=0; i<indexesRows; i++)
    {
        if(indexes[i][1] != -1)
        {
            curr->right = (TreeNode*)
                malloc(sizeof
                    (TreeNode))

            curr->right->data = indexes[i][0];
            curr->left->left = NULL;
            curr->left->right = NULL;
            nodes[indexes[i][0]] = curr->left;
        }
    }
    return root;
}
```

```
int ** swapnodes(int** indexes_rows, int
                 indexes_columns)
{
    int ** result = (int**)malloc(queries_
                    count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;

    TreeNode* root = buildTree(indexes_rows,
                     indexes_columns,
                     indexes);

    for(int i=0; i<queries_count; i++)
    {
        int k = queries[i];
        swapSubtrees(root, k, 1);
        int* traversal = (int*) malloc(index_rows
                         * sizeof(int));
        int index = 0;
        inOrderTraversal(root, traversal, index);
        result[i] = traversal;
    }
    return result;
}
```