INDEX

1BM22CS199

Name PRAJWAL.K.KEMPALINGANNAVAR Std ____ Sec ____

Roll No. 199 Subject ____ School/College ____

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
01	7/12/23	LAB-0		
02	21/12/23	LAB-1		21/12
03	28/12/23	LAB-02		28/12
04	11/01/24	LAB-03		11/1/24
05	18/01/24	LAB-04		18/1/24
06	25/01/24	LAB-05		
07	25/01/24	OS Programs.		

1BM22CS199

INDEX

Name PRAJWAL.K.K. Std 3rd sem Sec "D"

Roll No 199 Subject OS [LAB] School/College

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
08	01/02/24	LAB-07	10	
09	15/02/24	LAB-08	10	8th 22/2.
10	21/02/24	LAB-09	10	
11	29/02/24	LAB-10	10	8th 29/2

10

8th 29/2/24

Date 21/12/23
Page

LAB-01

01. Swapping of two number using Pointers.

Soln

```
#include<stdio.h>
int main()
{
  int a, b, temp;
  int *ptr1, ptr2;
  printf("\n Before swapping a=%d and
        b=%d, a, b);
  scanf("%d %d", &a, &b);
  printf("\n Before swapping a=%d and
        b=%d, a, b);
  *ptr1 = &a;
   ptr2 = &b;
   temp = *ptr1;
  *ptr1 = *ptr2;
  *ptr2 = temp;
  printf("\n After swapping a=%d and
        b=%d, a, b);
  return(0);
}
```

∴ **Output :-**

```
Enter the two number a and b
77
19

Before swapping a=77 and b=19
After swapping a=19 and a=77.
```

Date __/__/__
Page _____

Q2 Dynamic memory allocation ?
Using {malloc, free, calloc, realloc}

Sol?

```
#include <stdio.h>
#include <stdio.h>
int main()
{
int *alloc;
int n, i;
printf("Enter the number of elements:");
scanf("%d", &n);
printf("Entered number of elements:", n);
alloc = (int*)malloc(n * sizeof(int));
if(alloc == NULL)
{
  printf("Memory not Allocated:");
}
else
{
printf("Memory Allocation successfull:");
for(i=0; i<n; i++)
{
  alloc[i] = i+1;
}
printf("The elements of Array are:");
for(i=0; i<n; i++)
{
  printf("%d", alloc[i]);
}
}
return(0);
}
```

Date / /
Page

Output :-

Enter number of element: 3
Enter number of element: 3
Memory successfully allocated using malloc
The element of the array are: 1, 2, 3.

☆ Output :-

Enter the number of element 3

Entered number of elements Memory Allocation successful the elements of array are 1, 2, 3.

Date __/__/__
Page _____

03. Stack Implementation?

Q. Write a program to simulate the working of stack using an array with the following:

(a) Push
(b) POP
(c) Display.

The program should print appropriate message for stack overflow, stack underflow.

Sol?

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int top =-1, input [SIZE];
void push();
void pop();
void display();
int main()
{
  int c;
  while (1)
  {
   printf("Performance operation on the stack");
   printf("\n1. PUSH elements \n2. POP
        element \n3. display. element");
   printf("\n\n Enter the choice:");
   scanf("%d", &c);
   switch(c);
   {
```

```
Case 1:
    push();
    break;
  Case 2:
      pop();
      break();
     case 3:
       display();
       break;
        Case 4:
           exit(0);
       default:
       printf("\n Invalid choice");
    }
  }
}
Void push()
 {
   int x;
   if (top = SIZE-1)
   {
     printf(" \n Overflow !");
   }
   else
   {
     printf("\n Enter elements to be
          added onto the stack:");
     scanf("%d", &x);
     top = top+1;
      inpu(top) = x;
   }
}
```

```
void pop()
{
  if (top == -1)
  {
    printf("\n Underflow !!");
  }
  else
  {
    printf("\n Poped element: %d", inpu[top]);
    top = top - 1;
  }
}
void display()
{
  if (top == -1)
  {
    printf("\n Underflow !!");
  }
  else
  {
    printf("\n Element present in the stack:");
    for (int i = top ; i >= 0; i--)
    {
      printf("%d\n", inpu[i]);
    }
  }
}
```

Date / /
Page

:- Output -

Performance operation on stack
N1. Push element
N2. POP element
N3. Display element

Enter the choice: 1

Enter element to be added: 4

Operation on Stack

N1. Push element
N2. POP element
N3. Display element

Enter the choice: 3

Elements present in stack = 4.

Date 28/12/23
Page

LAB-02

(1) Q Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators {+, -, *, /}.

Sol?

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

#define MAX_SIZE 100

int isoperator(char ch)
{
  return (ch=='+' || ch=='-' || ch=='*'
    || ch=='/' || ch=='%');
}

int precedence(char operator)
{
  if (operator=='+' || operator=='-')
    return (1);
  if (operator=='*' || operator=='/' ||
    operator=='%')
    return (2);
  return (0);
}

void infixToPostfix(char infix[], char postfix[])
{
  char stack[MAX_SIZE];
  int top=-1;
  int i,j;
```

```
for (i=0, j=0; infix[i] != '\0'; i++)
{
  if (infix[i] >= '0' && infix[i] <= '9')
  {
    postfix[j++] = infix[i];
  }
  else if (isOperator(infix[i]))
  {
    while (top >= 0 && precendence
    (stack[top] >= precendence(infix[i]))
    {
      postfix[j++] = stack[top--];
    }
    stack[++top] = infix[i];
  }
  else if (infix[i] == '(')
    {
      stack[++top] = infix[i];
    }
    else if (infix[i] == ')')
    {
      while (top >= 0 && stack[top] != '(')
      {
        postfix[j++] = stack[top--];
      }
      if (top >= 0 && stack[top] == '(')
      {
        top--;
      }
    }
}
```

```
while (top >= 0)
    {
  postfix[j++] = stack[top--];
    }
  postfix[j] = '\0';
}
int main()
{
 char infix[MAX_SIZE], postfix[MAX_SIZE];

 printf("Enter infix expression : ");
 scanf("%s", infix);

 infixToPostfix(infix, postfix);

 printf("Postfix expression : %s\n", postfix);
 return(0);
}
```

:- Output :-

>> Enter infix expression : 12*34**+5-
Postfix expression : 1234**5+-

(9) Q. Evaluation of Postfix ?

Sol?

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

#define MAX 100
int stack[MAX];
int top = -1;
void push (int item)
{
    if (top >= MAX-1)
    {
        printf ("Stack overflow\n");
        return;
    }
    stack[++top] = item;
}
int pop ()
{
    if (top < 0)
    {
        printf ("Stack Underflow\n");
        return (-1);
    }
    return stack[top--];
}
int evaluate_postfix (char postfix[])
{
    int i, operand1, operand2, result;
    char ch;
```

```
for(i=0; postfix[i] != '\0'; i++)
{
  ch = postfix[i];
  if (isdigit(ch))
  {
  push(ch - '0');
  }
  else
  {
  operand2 = Pop();
  operand1 = pop();
  switch (ch)
  {
   case '+':
     result = operand1 + operand2;
     break;
   case '-':
     result = operand1 - operand2;
     break;
   case '*':
     result = operand1 * operand2;
     break;
   case '/':
     result = operand1 / operand2;
     break;
   default;
     printf("Invalid Expression\n");
     exit(1);
  }
  push(result);
  }
}
```

```
    return pop();
}

int main()
{
    char postfix[MAX];
    printf("Enter postfix expression :");
    gets(postfix);
    int result = evaluate_postfix(postfix);
    printf("Result :%d\n", result);
    return(0);
}
```

:- Output :-

>> Enter postfix expression 12*34*+5-
Result : 9

>> Enter postfix expression : 34*25*+
Result : 22

Reference

Input	Stack	Postfix evaluation
34*25*+	—	—
4*25*+	3	—
25+	4 3	Pop 4 & Pop 3 from stack do 3*4, push 12
25*+	12	Push 2 into stack
5*+	2 12	" 5 " "
*+	5 2 12	Pop 5, Pop 2 from stack do 2*5, Push 10 into stack
+	10 12	Pop 10 & Pop 12 from stack do 12+10, Push 22 into stack
	22	

Date __/__/__
Page _____

Program3:- Linear Queue Implementation

Sol?

```
#include <stdio.h>
#include <string.h>
#define MAX 100
int rear = -1,
    front = -1,
int queue[MAX];
void enqueue();
void display();
int dequeue();
int main();
{
 int option, val;
 do
 {
printf("\n Enter an option to perform
following operation \n
    1. Insert \n
    2. Delete \n
    3. Display \n
    4. Exit \n");
scanf("%d", &option);
switch (option)
  {
 case 1: enqueue();
 break;
 case 2: val = dequeue();
 printf(" Element deleted from queue is
  %d", val);
break;
```

```
case 3: display();
break;
}
}
while(option != 4);
return 0;

void enqueue()
{
int x;
printf("Enter the number to be inserted
in the queue \n");
scanf("%d", &x);

if(rear == MAX-1)
printf("Overflow");
else if(front == -1 && rear == -1)
front = rear = 0;
else
rear = rear+1;
queue[rear] = x;
}

int dequeue()
{
int y;

if(front == -1 || front > rear)
printf("Underflow");
else
{
y = queue[front];
front = front+1;
```

```
}
return g;
}
void display ()
{
int i;
printf("Elements in the queue :\n");
if(front == -1 || front > rear)
printf("Underflow");
for(i=front; i<=rear; i++)
printf("\t%d", queue[i]);
}
```

∴ Output :-

```
Enter an option to perform following
operations
1. Insert
2. Delete
3. Display
4. Exit
1
Enter the no. to be inserted in the queue
17

Enter an option to perform following
operations
1. Insert
2. Delete
3. Display
4. Exit
2.
```

Date __/__/__
Page _____

Element deleted from queue is: 18

Enter the option to perform following operations

1. Insert
2. Delete
3. Display
4. Exit

3

Elements in the queue:
Underflow.

"LAB-04"

3b

WAP to simulate the working of a circular queue of integers using an array.

Sol?

```
#include <stdio.h>

#define SIZE 5

int CQ[SIZE];
int front=-1;
int rear=-1;

void enqueue (int element)
{
    if (front==-1 && rear==-1)
    {
        front=0;
        rear=0;
        CQ[rear]=element;
    }
    else if ((rear+1)%SIZE==front)
    {
        printf("Queue is overflow...");
    }
    else
    {
        rear=(rear+1)%SIZE;
        CQ[rear]=element;
    }
}
```

```
int dequeue()
{
    if ((front==-1)&&(rear==-1))
    {
        printf("\n Queue is underflow...");
    }
    else if (front==rear)
    {
        printf("\n The dequeued element is
                %d", q[front]);
        front=-1;
        rear=-1;
    }
    else
    {
        printf("\n The dequeued element
                is %d", q[front]);
        front=(front+1)%SIZE;
    }
}

void display()
{
    int i=front;
    if (front==-1&& rear==-1)
    {
        printf("\n queue is empty...");
    }
    else
    {
```

```
printf("\n Elements in a queue are: ");
while (i<=rear)
{
    printf("%d", cq[i]);
    i = (i+1) % SIZE;
}
}
}
int main()
{
    int choice=1,2;

while (choice<4 && choice!=0)
{
printf("\n Press 1: Insert");
printf("\n Press 2: Delete");
printf("\n Press 3: Display");
printf("\n Press 4: Exit");
printf("Enter your choice");
scanf("%d", &choice);

switch (choice)
{
case 1:
printf("Enter the element which is
    to be inserted");
scanf("%d", &x);
enqueue(x);
break;
```

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

case 3:
display ();

}

}
return (0);

}
```

→ Output :-

```
Press 1: Insert
Press 2: Delete
Press 3: Display
Enter your choice 1

Enter the element which is to be inserted
17

Press 1: Insert
Press 2: Delete
Press 3: Display
Enter your choice 1

Enter the element which is to be inserted
18
```

Press 1: Insert
Press 2: Delete
Press 3: Display
Enter your choice 1

Enter the element which is to be inserted 12

Press 1: Insert
Press 2: Delete
Press 3: Display
Enter your choice 2

The dequeued element is 17

Press 1: Insert
Press 2: Delete
Press 3: Display
Enter your choice 3

Elements in a queue are : 18, 12 //

Date / /
Page

☆ WAP to Implement Singly linked list with following operations:

Sol?

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

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

Node * head = NULL;

void push();
void append();
void insert();
void display();

int main()
{
    int choice;
    while (1)
    {
    printf("1. Insert at beginning\n");
    printf("2. Insert at end\n");
    printf("3. Insert at position\n");
    printf("4. Display\n");
    printf("5. Exit\n");
    printf("Enter choice:");
```

```
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");
    return(0);
  }
}

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 the 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 the 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");
}
```

Date / /
Page

Output

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

Enter choice = 1
Enter data in the new node: 11

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

Enter choice: 2
Enter data in the new node: 27

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

Enter choice: 3
Enter data in the new node: 17
Enter position of the new node: 1

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

Enter choice: 4

11 -> 27 -> 17

18/1/24

Date __/__/__
Page _____

LAB-05

☆ WAP to Implement Singly Linked List with following operations
(a) Create a linked list.
(b) Deletion of first element, Specified element and last element in the list
(c) Display the contents of the linked list

Sol:

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

struct node
{
int info;
struct node * link;
};
struct node *start = NULL;

Void CreateList()
{
 if (start == NULL)
 {
   int n;
   printf("\n Enter the no. of Nodes \n");
   scanf("%d", &n);
```

```
if (n>0)
{
int data;
struct node * newnode;
struct node * temp;

newnode = malloc (sizeof (struct node));
start = newnode;
* temp = start;

printf("\n Enter number to be inserted:");
scanf ("%d", &data);
start -> info = data;

for (int i = 2; i <= n; i++)
  {
    newnode = malloc (sizeof (struct node));
    temp -> link = newnode;

printf("\n Enter the number to be
        inserted: ");
scanf ("%d", &data);
newnode -> info = data;
 temp = temp -> link;
  }
 temp -> link = NULL;
}
printf ("\n The list is created in");
}
else
 {
printf("\n The list is already
        created \n");
```

```
    }
}

void display()
{
struct node *temp;

if(start == NULL)
printf("\n list is empty\n");

else
{
temp = start;
while(temp != NULL)
{
printf("Data = %d\n", temp->info);
temp = temp->link;
}
}
}

void deleteFirst()
{
struct node *temp;
if(start == NULL)
printf("\n list is empty\n");
else
{
temp = start;
start = start -> link;
free(temp);
}
}
```

```
Void deleteEnd()
{
struct node *temp, *prevnode;
if(start == NULL)
 printf("\n List is Empty \n");
else
 {
 temp = start;
 while (temp -> link != NULL)
 {
  prevnode = temp;
  temp = temp -> link;
 }
 free(temp);
 prevnode -> link = NULL;
 }
}

Void deletePosition()
 {
 struct node *temp, *position, *prevnode;
 int i=1, pos;

 if (start == NULL)
   printf("\n List is empty: \n");
 else
   {
    printf("\n Enter index: ");
    scanf("%d", &pos);
```

```
if (pos <= 0)
{
    printf("\n Invalid position \n");
    return;
}

temp = start;
position = NULL;

if (pos == 1)
{
    start = start -> link;
    free(temp);
    return;
}

while (i < pos && temp != NULL)
{
    prevnode = temp;
    temp = temp -> link;
    i++;
}

if (temp == NULL)
{
    printf("\n Invalid position \n");
    return;
}
```

```
        position = temp;
        prevnode -> link = temp -> link;
        free(position);
    }
}

int main()
{
createlist();
int choice;
while (1)
{
printf("\n It 1. To see list \n");
printf("\n It 2. For deletion of first
        element \n");
printf("\n It 3. For deletion of last
        element \n");
printf("\n It 4. For deletion of element
at any position \n");
printf("It 5. To exit \n");
printf("Enter choice :\n");
scanf("%d", &choice);

switch (choice)
{
case 1:
display();
break;

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

```
case 3:
deleteEnd();
break;

case 4:
deletePosition();
break;

case 5:
exit(1);
break;
default:
printf("Incorrect Choice\n");
}
}
return(0);
}
```

→ Output :-

```
Enter the number of nodes: 4
Enter number to be inserted: 11
Enter number to be inserted: 23
Enter number to be inserted: 17
Enter number to be inserted: 18
```

The list is created

01. To see list
02. For deletion of first element
03. " " " last "
04. " " " element at any position
05 To exist

Enter choice:
2

01. To see list
02 For deletion of first element
03 " " " last "
04. " " " element at any position
05 To exit

Enter choice:
1

Data = 23
Data = 17
Data = 18

01 To See list
02 For deletion of first element
03 " " " last element
04 " " " element at any position
05 To exit

Enter choice:
4
Enter index: 2

02. To see list
03 For deletion of first element
03 For deletion of last element
04 For deletion of element at any pos
05. Exit

Enter choice:
1
Data = 23
Data = 18

02 To see list
03 For deletion of first element
03 For deletion of last element
04 For deletion of element at any pos
05 Exit

Enter choice:
5

//

18/4/24

Leet Code - 1 :-

```
#include <stdlib.h>

typedef struct {
    int * stack;
    int * minstack;
    int top;
}
  minstack;

minstack * minstackCreate()
{
  minstack * stack = (minstack *) malloc
     (sizeof(minstack));

  stack-> stack = (int *) malloc(sizeof(int)*30);
  stack->minstack = (int *) malloc(sizeof(int)*30);
  return stack;
  }

void minstackPush(minstack* obj, int val)
  {
  obj -> top++;
  obj -> stack [obj -> top] = val;
if (obj -> top == 0 || val <= obj ->
      minstack [obj -> top-1])
obj -> minstack [obj -> top] = val;
  }
else
  {
  obj -> minstack [obj -> top] = obj -> minstack
        [obj -> top-1];
```

```
}
}
Void minstackPop (minStack *obj {
obj->top--;
}
int minstack top (min stack* obj)
{
return obj->stack[obj->top];
}
int minstack getmin (minstack *obj) {
return obj->min stack[obj->top];
}
Void minstackFree (minStack *obj) {
free(obj->stack),
free(obj->minstack),
free(obj);
}
//
```

÷ Output :-

[null, null, null, null, -3, null, 0, -2]

Expected :

[null, null, null, null, -3, null, 0, -2]

Leetcode-02:-

```
Struct List Node *ReverseBetween
    (Struct ListNode * start, int a, int b)
{
    a -= 1;
    b -= 1;
    Struct ListNode* node1 = NULL;
                *node2 = NULL, *nodeb = NULL,
                *node3 = NULL, *ptr = &start;

int c = 0;
while (ptr != NULL)
{
    if (c == a-1)
        node b = ptr;
    else if (c = a)
        node 1 = ptr;
    else if (c == b)
        node 2 = ptr;
    else if (c == b+1)
        node a = ptr;
        break;
    }
    c += 1;
    ptr & = ptr -> next;
    }
    Struct ListNode * ptr = node, * temp;
        ptr = Struct;
        c = 0;

while (ptr != NULL)
    {
```

```
if(c>=a && c<b)
{
    temp = ptr->next;
    ptr->next = ptr;
    ptr->ptr;
    ptr = temp;
}
else if(c==h)
{
    ptr->next = ptr;
    if(a==0)
        start = ptr;
    else
        node b->next = ptr;
        break;
    }
    else
        ptr = ptr->next;
        c+1;
    }
return start;
}
```

∴ Output:-

head

[1, 2, 3, 4, 5]

left = 2

height = 3

=> [1, 3, 4, 2, 5] //

25/1/24
AB-06

Date __/__/__
Page

A WAP to Implement Single linked list with following operations:
Sort the linked list, Reverse the linked list, Concatenation of two linked list.

Sol?

```
#include <stdio.h>
#include <stdlib.h>
Struct node
 {
  int data;
  Struct node* next;
  };
 Struct node *S1=NULL;
 Struct node *S2=NULL;
 struct node *Start=NULL;
 Struct node *Create(struct node*);
 Void sort();
  Struct node *concatonate(struct node*,
        Struct node*);
  Void reverse();
  Void display(struct node*);

int main()
{
  int option;
  struct node *a=NULL;
  do{
  printf("\n MAIN MENU \n\n
      1. Create a linked list \n
        2. Create two linked list
          for concatonation \n
        3. Sort \n
```

4. Concatenate \n
5. Reverse \n
6. Display linked list \n
7. Display Concatenated linked list \n
8. Exit \n");

printf("\n Enter an option to perform the following operations:");
scanf("%d", &option);

switch (option)
{
case 1: start = create (start);
printf("\n linked list created successfully \n");
break;
case 2: printf("\n linked list 1: \n");
s1 = create (s1);
printf("\n linked list 2: \n");
s2 = create (s2);
printf("\n linked lists created successfully \n");
break;
case 3: sort();
printf("\n linked list sorted \n");
break;
case 4: a = concatenate (s1, s2);
printf("\n linked lists concatenated successfully \n");
break;

```
case 5: reverse();
        printf("\n Linked list reversed \n");
        break;
case 6: printf("\n Elements in the linked
        list \n");
        display(start);
        break;
case 7: printf("\n Elements in the linked
        list after concatenation : \n");
        display(a);
        break;
    }
  }
  while(option != 8);
  return(0);
}

struct node *create(struct node *start)
{
  struct node *ptr, *new_node;
  int num;
  printf("Enter -1 to exit \n");
  printf("\n Enter the data : ");
  scanf("%d", &num);

  while(num != -1)
  {
    new_node = (struct node*) malloc
      (sizeof(struct node));

    new_node->data = num;
```

```
if (start == NULL)
{
    start = new_node;
    new_node -> new = NULL;
}
else
{
    ptr = start;
    while (ptr -> next != NULL)
    ptr = ptr -> next;
    ptr -> new = new_node;
    new_node -> next = NULL;
}
printf ("Enter the data : ");
scanf ("%d", &num);
}
return start;
}

void sort ()
{
    Struct node *i; *j;
    int temp;
    for (i = start; i -> next != NULL;
        i = i -> next)
    {
    for (j = i -> next; j != NULL; j = j -> new)
    {
    if (i -> data > j -> data)
    {
    temp = i -> data;
    i -> data = j -> data;
```

```
        j->data = temp;
      }
    }
  }
}

struct node *concatenate(struct node
 *t1, struct node *t2)
 {
 struct node *ptr;
 ptr = t1;
 while (ptr->next != NULL)
  {
   ptr = ptr->next;
  }

void reverse()
 {
 Struct node *prev = NULL;
 Struct node *next = NULL;
 Struct node *Cur = Start;
 while (Cur != NULL)
  {
   next = Cur->next;
   Cur->next = prev;
   prev = Cur;
   Cur = next;
  }
  Start = prev;
 }
```

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

÷ Output :-

MAIN MENU

1. Create a linked list
2. Create two linked for concatenation
3. Sort
4. Concatenate
5. Reverse
6. Display linked list
7. Display concatenated linked list
8. Exit

Enter an option to perform the following operation : 1

Date __/__/__
Page _____

Enter the data : 1
" " " : 23
" " " : 17
" " " : 8
" " " : 55

Linked list created Successfully.

MAIN MENU

1. Create a linked list
2.
3.
4.
5.
6.
7.
8.

Enter an option to perform the following operation: 3

Linked list sorted

MAIN MENU

1.
2.
3.
4.
5.
6.
7.
8.

Date / /
Page

Enter an option to perform the following operation: 6

Elements in the linked list
1 8 17 23 55

//.

☆ WAP to implement single linked list to simulate Stack & Queue Operation

Sol?

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

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

struct node *start = NULL
void push();
void pop();
void display();
int main()
{
  int val, option;
  do
  {
    printf("\nEnter the number of
Perform following operations\n
```

Date __/__/__
Page _____

1. Push ()
2. Pop ()
3. Display ()
4. Exit ();

```
scanf("%d", &option);
switch(option)
{
  case 1: push();
  break;
  case 2: pop();
  break;
  case 3: display();
  break;
  }
}
while (option != 4);
  return(0);
}
void push()
{
  struct node *new_node;
  int num;
  printf("Enter the data in");
  scanf("%d", &num);
  new_node = (struct node *) malloc
  (sizeof(struct node));
  new_node -> data = num;
  new_node -> next = start;
  start = new_node;
}
```

```
void pop()
{
 struct node *ptr;
 ptr = start;
 if (start = NULL)
 {
   printf("Stack is empty\n");
   exit(0);
 }
 else
 {
   ptr = start;
   start = ptr->next;
   printf("\n Element popped from
   the stack is: %d\n", ptr->data);
   free(ptr);
 }
}

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

Date __/__/__
Page _____

÷ Output :-

Enter the number to perform following operation

1. PUSH
2. POP
3. DISPLAY
4. Exit

1.

Enter the data
17

1. PUSH
2. POP
3. DISPLAY
4. Exit

1.

Enter the data
11

1. PUSH
2. POP
3. DISPLAY
4. Exit

3.

17 11

1. PUSH
2. POP
3. DISPLAY
4. Exit

2.

Ele poped :17

Date __/__/__
Page _____

1. PUSH
2. POP
3. DISPLAY
4. Exit

3.

11

11

Queue

```
#include <stdio.h>
#include <stdlib.h>
 struct node
 {
  int data;
  struct node * next;
 };
struct node *start = NULL;
 void enqueue ();
 void dequeue ();
 void display ();
  int main()
  {
   int val, option;
    do
    {
    printf("\n Enter the no. of
perform following operations \n
```

1. Enqueue \n
2. Dequeue \n
3. Display \n
4. Exit \n");

```
scanf("%d", &option);
switch (option)
{
case 1: enqueue();
break;
case 2: dequeue();
break;
case 3: display();
break;
}
}
while (option != 4);
return(0);
}
void enqueue()
{
struct node *new_node;
int main num;
printf("Enter the data\n");
scanf("%d", &num);
new_node = (struct node *) malloc
(sizeof(struct node));
new_node -> data = num;
new_node -> next = start;
start = new_node;
}
void dequeue()
{
struct node *ptr, *preptr;
ptr = start;
if (start == NULL)
{
```

```
printf("Stack is empty\n");
exit(0);
}
else if (start -> next == NULL)
{
start = start -> next;
printf("\n Element popped: %d",
ptr -> data);
free(ptr);
}
else
{
while (ptr -> next != NULL)
{
preptr = ptr;
ptr = ptr -> next;
}
preptr -> next = NULL;
printf("\n Element poped: %d",
ptr -> data);
free(ptr);
}
}
void display()
{

struct node *ptr;
ptr = start;
while (ptr != NULL)
{
```

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

÷ Output :-

Enter the number to perform following operations

1. En queue
2. Dequeue
3. Display
4. Exit

÷ Output :-

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

1.

Enter the data
4

Enter the no. perform following opera
1. Enqueue
2. Dequeue
3. Display
4. Exit

1.

Date __/__/__
Page _____

Enter the data
17

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

3.

11 17

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

2.

Element popped from the stack
22:11

1. Enqueue
2. [illegible]
3. Display
4. Exit

3.

17

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

LAB-07:-

Q WAP to implement doubly link list with primitive operations

Sol?

```
#include <stdio.h>
#include <stdlib.h>
Struct Node
{
  int data;
  struct Node* prev;
  struct Node* next;
};

Struct Node* CreateNode (int data)
  {
   struct Node* newNode = (struct Node*)
  malloc (sizeof (struct Node));
  newNode -> data = data;
  new Node -> prev = NULL;
  new Node -> next = NULL;
  return newNode;
  }

Void insertLeft (struct Node** head,
int value, int target)
{
  Struct Node* newNode = CreateNode (value);
  Struct Node* current = *head;

while (current != NULL && current -> data
!= target)
```

```
{
current = current->next;
}

if (current != NULL)
{
newNode->prev = current->prev;
newNode->next = current;
if (current->prev != NULL)
{
current->prev->next = newNode;
}
else
{
*head = newNode;
}
current->prev = newNode;
}
else
{
printf("Node with value %d
not found.\n", target);
}
}

Void deleteNode (struct Node** head, int
value)
{
struct Node* current = *head;

while (current != NULL && current->data
!= value)
```

```
{
    current = current->next;
}
if (current != NULL)
{
    if (current->prev != NULL)
    {
        current->prev->next = current->next;
    }
    else
    {
        *head = current->next;
    }
    if (current->next != NULL)
    {
        current->next->prev = current->prev;
    }
    free(current);
}
else
{
    printf("Node with value %d not found in ", value);
}
}

void displaylist(struct Node* head)
{
    struct Node* current = head;
```

```
while (current != NULL)
  {
  printf("%d -> ", current -> data);
  current = current -> next;
  }
printf("NULL\n");
}

int main()
  {
  struct Node * head = NULL;
  int choice, value, traget;

do {

  printf("\n Main Menu\n");
  printf(" 1.) Create a doubly linked
      list \n");
  printf(" 2) Insert a new node to the
      left of the node \n");
  printf(" 3) Delete the node based
      on a specific value \n");
  printf(" 4) Exit \n");

  printf("Enter your choice : ");

  scanf("%d", &choice);
```

```
switch (choice)
 {
case 1:
    if (head != NULL)
    {
      printf ("Doubly linked list already
          exists. \n");
    }
    else
    {
    head = CreateNode (1);
    head -> next = CreateNode (2);
    head -> next -> prev = head;
    head -> next -> next = CreateNode (3);
  head -> next -> next -> prev = head -> next;
  printf ("Doubly linked list created. \n");
    }
  break;
case 2:

  printf ("Enter the value to be inserted:");
  scanf ("%d", &value);
  printf ("Enter the target value :");
  scanf ("%d", &target);
  insertLeft (&head, value, target);
  printf ("After insertion :");
  displayList (head);
  break;
```

```
Case 3:

printf("Enter the value to be deleted:");
scanf("%d", &value);
deleteNode(&head, value);
printf("After deletion:");
displayList(head);
break;

Case 4:

printf("Exiting the program.\n");
break;
default:
printf("Invalid choice. Please enter a
    valid option.\n");
    }
  }
while (choice != '4');

return(0);
}
```

Output :-

Menu :

02 Create
02 Insert a new node to the left of the node
03 Delete the node based on a specific value
04 Exit.

Enter your choice: 1
Doubly linked list Created.

Menu:
01
02
03
04

Enter your choice: 2
Enter the value to be inserted: 33
Enter the target value: 2

After insertion: 1->33->2->3->NULL

Menu:
01
02
03
04

Enter your choice: 2
Enter the value :44
Enter the target value: 3

After insertion: 1->33->2->44->3->NULL

Menu:
01
02
03
04

"Leet Code-03"

Split linked list in Parts.

```
struct ListNode ** splitListToParts
(struct ListNode * head, int k, int *
returnSize)
{
 Struct listnode* ptr=head;
   *returnSize=k;
    int count=0;

 While (ptr!=NULL)
 {
  count++;
  ptr=ptr->next;
 }
int nums = count/k, a=count%k;

Struct ListNode **L=(struct ListNode**)
Calloc (k, sizeof (struct ListNode *));

ptr=head;

for (int i=0; i<k; i++)
 {
  L[i]=ptr;
int Segment size =nums+(a-->0?1:0);
for (int j=1; j< Segment size ; j++)
 {
  ptr=ptr->next;
 }
```

```
if (ptr != NULL)
{
struct ListNode * next = ptr->next;
ptr->next = NULL;
ptr = next;
}
}
return L;
}
```

☆ Output:-

(i) Input : lead = [1,2,3], K=5
Output : [[1],[2],[3],[],[]]

(ii) Input : lead = [1,2,3,4,5,6,7,8,9,10], K=3
Output : [(1,2,3,4),[5,6,7),[8,9,10]]

LAB-08:-

Q Write a program :-
(a) To construct a binary search tree.
(b) To traverse the tree using all the methods i.e. inorder, preorder and postorder.

Sol?

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

struct node
 {
 int data;
 struct node *left;
 struct node *right;
 };

struct node *newNode (int data)
 {
 struct node *node =(struct node*)
malloc (sizeof (struct node));
 node -> data = data;
 node -> left = NULL;
 node -> right = NULL;

 return node;
 }

struct node *insert (struct node *root,
        int data)
{
```

```
if (root == NULL)
{
return newNode(data);
}
else
{
if (data < root->data)
{
root->left = insert(root->left, data);
}
else if (data > root->data)
{
root->right = insert(root->right, data);
}
}
return root;
}

void inorder(struct node *root)
{
if(root != NULL)
{
inorder(root->left);
printf("%d", root->data);
inorder(root->right);
}
}

Void preOrder(struct node * root)
{
```

```
if (root != NULL)
{
printf("%d", root->data);
preOrder(root->left);
preOrder(root->right);
}
}

void postOrder(struct node *root)
{
if (root != NULL)
{
postorder(root->left);
postorder(root->right);
printf("%d", root->data);
}
}

void display(struct node *root)
{
printf("In-order traversal:");
inOrder(root);
printf("\n");
printf("Pre-order traversal:");
preOrder(root);
printf("\n");
printf("Post-order traversal:");
postOrder(root);
printf("\n");
}
```

```
int main()
{
    struct node * root = NULL;
    int data;
    printf("Enter the elements of the
            tree (-1 to stop):");
    while (scanf("%d", &data) && data
            != -1)
    {
        root = insert(root, data);
    }
    display(root);
    free(root);
    return 0;
}
```

Output :-

```
Enter the elements of the tree :
(-1 to stop)
7
9
8
2
3
4
1
-1
```

In order traversal : 1 2 3 4 7 8 9
Pre-order traversal : 7 2 1 3 4 9 8
Post-order traversal : 1 4 3 2 8 9 7

Leet code - 4 :- Rotate list

```
struct ListNode* rotateRight(struct
    ListNode *head, int k)
{
 struct ListNode *ptr, *ptr1;
 int count = 0, num;
 if (head == NULL)
  if (head -> next == NULL)
   {
    return head;
   }
  ptr = head;
  while (ptr -> next != NULL)
   {
    count++;
    ptr = ptr -> next;
   }
  num = k % (count + 1);
  while (num--)
   {
    ptr = head;
    while (ptr -> next != NULL)
     {
      ptr1 = ptr;
      ptr = ptr -> next;
     }
    ptr -> next = head;
    ptr1 -> next = NULL;
    head = ptr;
   }
  return head;
}
```

A Output:-

head = [1,2,3,4,5]
K=2
output 1 = [4,5,1,2,3]

head = [0,1,2], K=4

output
22[2,0,1] //

Still 2/2

Lab: 9

BFS:-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct queue
{
 int item [MAX_SIZE];
 int front;
 int rear;
 };
struct queue *create queue()
{
 struct queue *queue = (struct queue*)
    malloc queue -> front = -1;
    queue -> rear = -1;
    return queue;
  }

int isEmpty (struct queue * queue)
 {
  if (queue -> rear == -1)
     return 1;
   else return 0;
  }
 void enqueue (struct queue *queue, int value)
  {
  if (queue -> rear == MAX_SIZE-1)
  printf("\n queue is FULL!!");

else {
if (queue -> front == -1)
```

```
queue->front = 0;
queue->rear++;
queue->item[queue->rear] = value;
}
}

int dequeue(struct queue* queue)
{
int item;
if(isEmpty(queue))
{
printf("queue is Empty");
item = -1;
}
else {
item = queue->item[queue->front];
queue->front++;
if(queue->front > queue->rear)
{
queue->front = queue->rear = -1;
}
}
return item;
}

struct Graph {
int vertices;
int* adjMatrix;
};

struct Graph* createGraph(int vertices)
{
struct Graph* graph = (struct Graph*)
malloc(sizeof(struct Graph));
```

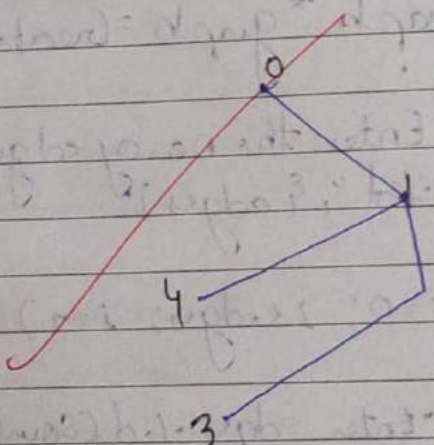
```
graph->vertices = vertices;
graph->adjMatrix = (int**) malloc
(vertices * sizeof(int*));

for(int i=0; i<vertices; i++)
{
graph->adjMatrix[i][j]=0;
}
return graph;
}

void addEdge(struct Graph* graph, int
src int dig)
{
graph->adjMatrix[src][dst]=1;
graph->adjMatrix[dest][drc]=1;
}

void BFS(struct Graph* graph, int
start vertex)
{
int visited[MAX_SIZE]={0};
struct Queue* queue = createQueue();
visited[start vertex]=1;

enqueue(queue, start vertex);
printf("Breadth Search Traversal:");
while(!isEmpty(queue))
{
int current vertex = dequeue(queue);
printf("%d", current vertex);
```

```
for (int i=0 ;i<graph->vertix;i++)
printf("%d", current vertex);
for(int i=0; i<graph->vertices;i++)
{
if (graph->adjmatrix[currentVertex][]==1
visited[i]==0)
{
visited[i]-1;
enqueue(queue,i);
}
}
}
printf("\n")
}
int main()
{
int vertices, edges, src, dest;
printf("Enter the no. of vertices:");
scanf("%d", &vertices);
Struct Graph *graph=Create Graph
vertices;
printf("Enter the no. of edges:");
scanf("%d", &edges);
for(int i=0; i<edges; i++)
{
printf("Enter edges %d (Source dest): i+1);
scanf("%d %d", &src, &dest);
addEdge(graph, src, dest);
}
```

```
int start Vertex;
printf("Enter the starting vertex for
    BFS:");
scanf("%d", &start Vertex);
BFS(graph, start Vertex)
return 0;
}
```

÷ Output :-

```
Enter the number of vertices : 5
Enter the no. of edges : 4
Enter edge 1 (src, dest) : 01
Enter edge 2 (src, dest) : 03
Enter edge 3 (src, dest) : 14
Enter edge 4 (src, dest) : 12

Enter the starting vertex for BFS : 0

Breadth Search Traversal : 0 1 2 4 3
```



DFS :-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

Struct Graph
{
  int Vertices;
  int ** adjMatrix;
};

Struct Graph * Create Graph (int Vertices)
{
  Struct Graph *graph = (Struct Graph *)
malloc (Size of (Struct Graph));

  graph -> Vertices = Vertices )
  Graph -> adjMatrix = (int **) malloc
  (Vertices * Size of (int));

for (int i=0; i<Vertices; i++)
{
  graph -> adjMatrix [i] = (int *) malloc
(Vertices * Size of (int));

for (int j=0; j<Vertices; j++)
  graph -> adjMatrix [i][j] = 0;
return graph;
}
```

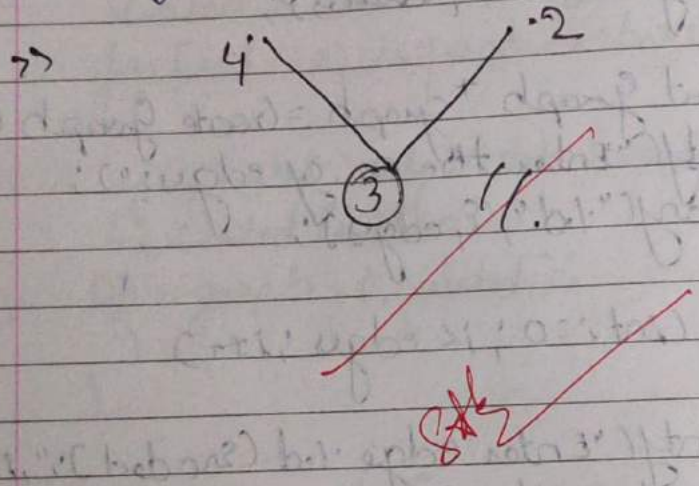
```
void addEdge(struct graph* graph,
int src, int dst)
{
graph->adjMatrix[src][dst] = 1;
graph->adjMatrix[dst][src] = 1;
}

void DFS(struct graph *graph,
int src, int dst)
{
visited[start vertex] = 1;
for(int i=0; i<graph->vertices; i++)
{
if(graph->adjMatrix[start vertex][i]==1
&& visited[i]==0)
DFS(graph, i, visited);
}
}

int isConnected(struct graph *graph)
{
int * visited = (int*) malloc(graph->vertices
* sizeof(int));

for(int i=0; i<graph->vertices; i++)
visited[i]=0;

DFS(graph, 0, visited);

for(int i=0; i<graph->vertices; i++)
```

```
{
if (visited [i] == 0)
    return 0;
}
return 1;
}
int main()
{
int vertices, edges, src, dest;
printf("Enter the no. of vertices:");
scanf("%d", &vertices);

Struct Graph *graph = Create Graph (vertices);
printf("Enter the no. of edges:");
scanf("%d", &edges);

for (int i=0; i<edges; i++)
{
printf("Enter edge %d (src dest):", i++);
scanf("%d %d", &src, &dest);
add Edge (graph, src, dest);
}

if (isConnected (graph))
printf("the graph is connected\n");
else
printf("The graph is not connected\n");
return 0;
}
```

Output :-

Enter the no. of Vertices: 5
Enter the no. of edges: 4
Enter edge 1 (Src dest) : 01
Enter edge 2 (Src dest) : 03
Enter edge 3 (Src dest) : 33
Enter edge 4 (Src dest) : 34

The graph is not connected



HackerRank : [Tree]

SoP

```
#include <stdlib.h>
#include <assert.h>
#include <string.h>

typedef struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
} Node;

Node* CreateNode (int data)
{
    Node* newNode = (Node*) malloc
        (sizeof (Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void inorderTraversal (Node* root,
int* result, int* index)
{
    if (root == NULL) return;
    inorderTraversal (root->left, result,
    index);
    result [(*index)++] = root->data;
    inorderTraversal (root->right, result,
    index);
}
```

```
Void swapAtLevel (Node * root, int k,
  int level)
  {
   if (root == NULL) return;
   if (level % k == 0)
     {
       Node * temp = root -> left;
       root -> left = root -> right;
       root -> right = temp;
     }
  swapAtLevel ( root -> left , k, level+1);
  swapAtLevel ( root -> right , k, level+1);
  }

swapAtLevel (root -> left, k, level+1);
swapAtLevel (root -> right , k, level+1);
  }

int ** swapNodes ( int indexes_rows,
 int indexes_columns,
 int ** indexes,
 int * queries,
 int * result_rows,
 int * result_columns)
   {


Node ** nodes = (Node **) malloc
((indexes_rows +1) * sizeof (Node*));
for (int i = 1; i <= indexes_rows ; i++)
   {
     nodes[i] = CreateNode (i);
   }
```

```
for (int i=0; i< indexes_rows ; i++)
{
    int leftIndex = indexes[i][0];
    int rightIndex = indexes[i][1];
    if (leftIndex != -1) nodes[i+1]
    -> left = nodes[leftIndex];
    if (rightIndex != -1) nodes[i+1]
    -> right = nodes[rightIndex];
}

int** result = (int**) malloc
(queries_count * sizeof(int*));
*result_rows = queries_count;
*result_columns = indexes_rows;
for (int i=0; i< queries_count; i++)
{
    scoopAtLevel(nodes[i], queries[i], 1);
int * traversalResult = (int*) malloc
(indexes_rows * sizeof(int));
int index=0;
inOrderTraversal(nodes[1], traversalResult,
&index);
result[i] = traversalResult;
}

free(nodes);
return result;
}
```

```
int main()
{
int n;
scanf("%d", &n);
int** indexes = malloc(n * sizeof(int*));
for (int i=0; i<n; i++)
{
    indexes[i] = malloc(2 * sizeof(int));
    scanf("%d %d", &indexes[i][0],
    &indexes[i][1]);
}

int queries_count;
scanf("%d", &queries_count);

int *queries = malloc(queries_count *
sizeof(int));
for (int i=0; i<queries_count; i++)
{
    scanf("%d", &queries[i]);
}

int result_rows;
int result_columns;
int** result = swapNodes(n, 2,
indexes, queries_count, queries,
result_rows, &result_columns);
```

```
for(int i=0; i<result.rows; i++)
{
  for(int j=0; j<result.columns; j++)
  {
    printf("%d", result[i][j]);
  }
  printf("\n");
  free(result[i]);
}
free(result);

for(int i=0; i<n; i++)
{
  free(indices[i]);
}
free(indices);
free(queries);
return 0;
}
```

÷ Output :-

```
3
2 3
-1 -1
-1 -1
2
1
1
```

" Your o/p

```
3 1 2
2 1 3
```

" Expected o/p

```
3 1 2
2 1 3
```

LAB-10 Hashing :-

```
#include <stdio.h>
#define TABLE_SIZE 10
int HashFunction(int key)
{
  return key % TABLE_SIZE;
}

void insertValue(int hashtable[],
     int key)
{
  int i=0;
  int hKey = hashFunction(key);
  int index;

  do {
    index = (hKey+i) % TABLE_SIZE;
    if (hashTable[index] == -1)
    {
      hashTable[index] = key
      printf("Inserted key %d at index %d\n",
        key, index);
      return;
    }
    i++
  }
  while (i<TABLE_SIZE);
  printf("unable to insert key %d.
    Hash Table
    is full \n", key);
}
```

```
int searchValue(int hashTable[], int key)
{
int i=0;
int key = hashFunction(key);
int index;

do{
index = (hkey + i) % TABLE_SIZE;
if(hashTable[index] == key)
{
printf("key %d found at index %d\n",
key, index);
return index;
}
i++;
}
while (i<TABLE_SIZE);
printf("key %d not found in hash
Table\n");
return -1;
}

int main()
{
int hashTable[TABLE_SIZE];
for(int i=0; i<TABLE_SIZE; i++)
{
hashTable[i] = -1;
}
```

```
insert value (hash Table, 18);
insert value (hash Table, 46);
insert value (hash Table, 7);
insert value (hash Table, 17);
insert value (hash Table, 1);

return 0;
}
```

∴ Output ∴

```
inserted Key 18 at index 8
inserted Key 46 at index 6
inserted Key 7 at index 7
inserted Key 17 at index 9
inserted Key 1 at index 2
```

:-HackerRank:-

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

Struct node
{
 int id;
 int depth;
 Struct node * left, * right;
};

Void
inorder (Struct node * tree)
 {
   if (tree == NULL)
      return;
 inorder (tree -> left);
 printf ("%d", tree -> id);
 inorder ((tree -> right);
 }

int main (void)
{
  int no_of_node, i=0;
  int l, r, max_depth, k;
  Struct node * temp = NULL;
  scanf ("%d", &no_of_nodes);
  Struct node * tree = (Struct node *) calloc
     (no_of_nodes, sizeof (Struct node));
     tree[0] depth = 1;
   while (i < no_of_nodes)
    {
```

```
tree[i].id = i+1;
scanf("%d %d %d", &l, &r);
if (l == -1)
    tree[i].left = NULL;
else
{
    tree[i].left = &tree[l-1];
    tree[i].left->depth = tree[i].depth+1;
    max_depth = tree[i].left->depth;
}

if (r == -1)
    tree[i].right = NULL;
else{
    tree[i].right = &tree[r-1];
    tree[i].right->depth = tree[i].depth+1;
    max_depth = tree[i].right->depth;
}

i++;
{
scanf("%d", &i);
while (i--)
{
scanf("%d", &l);
    k = l;
while (l != max_depth)
{
    for (k=0; k<(no_of_nodes); k++)
if (tree[k].depth == l)
{
    tree = tree[k].left;
```

```
tree[k].left = tree[k].right;
tree[k].right = temp;
}
}
l = l+1;
}
inorder(tree);
printf("\n");
return(0);
}
```

÷ Output :-

Hash Table

~~Tree is empty~~

Enter Key : 5
Enter Data : 50

Enter Key : 15
Enter Data : 150

Enter Key : 25
Enter Data : 250

Enter Key : 35
Enter Data : 350

Hash Table:

Index 0 : Empty
Index 1 : Empty
Index 2 : Empty
Index 3 : Empty
Index 4 : Empty
Index 5 : Empty
Index 6 : Key 5, Data 50
Index 7 : Key 15, Data 150
Index 8 : Key 25, Data 250
Index 9 : Key 35, Data 350
Index 10 : Empty