Lab Programs

1) Develop a C Program that Simulates banking System with Functionalities like account Creation, Withdrawal, deposite, and balance inquiry Write different user defined function for each

Output:

Banking System menu

1) Create account
2. Withdrawal
3. deposite
4. Check balance
5. Exit.

Enter your choice : 1

Enter account number: 1BM22CS228

Enter account holder name: navanalini

Enter initial balance : 25000

Account Created Successfully!

Enter your choice: 2

Enter Withdrawal amount : 2000

Withdrawal Successful.

New balance : 23000.00

Enter your choice : 3

Enter deposite amount : 4000

deposite Successful

New balance : 27000.00

Enter your Choice 4:

2) Account Holder : 1BY22CS228
Account number : 1
Current Balance : 27000.00

Enter your choice : 5

Existing Program, Goodbye!

2) Implement C Program that Sorts Strings lexicographically Considering Uppercase & lowercase Letters, without using Standard library Sorting Functions

Output: ~~Manual~~ Apple
Pooja banana
ball
~~batt~~ cat

apple
banana
cat
ball

3) Implement C Program to check if given Element is Present in 2D array with a user defined function.

Output Enter the number of rows and columns upto (10 Each): 2 3
Enter the Elements of 2D array
Element at Position [0] [0] : 1
Element at position [0][1] : 2
Element at Position [0][2] : 3
Element at Position [1][0] : 4
Element at position [1][1] : 5
Element at position [1][2] : 6
Enter the Element to search : 5
Element 5 is Present in the 2D array

4. Create a C Program to search for a specific Substring within a large String with User defined function.

Output: Enter the larger String : Mrunal
Enter the Substring to search for : nal
Substring found at index 4

5) Write a C Program to find index of last occurence of a number in an array with User defined function

for Output: Enter the size of array: 5
Enter the array elements :
1 2 3 2 4
Enter the number to find : 2
Last occurence of 2 is at index 3

6) Write a C Program ~~to~~ for linear Search

Output: Enter the size of array : 5
Enter 5 Elements :
10 20 30 40 50
Enter the Element to Search : 30
Element 30 found at Index 2.

7) Implement C Program to Perform binary Search function of array.

Output: Enter the size of array : 5
Enter the Elements of array in Sorted Order
1 2 3 4 5
Enter the Element to be Searched : 3
Element 3 found at index 2

8) Create a Program for to Search the minimum and maximum Elements in an array with a user defined function.

Output: Enter the size of array : 5
Enter 5 Elements :
Enter Element 1 : 1 2 3 4 5
Element 2 : Element 3 : Element 4 :
Element 5 :
Minimum Element : 1
Maximum Element : 5

21/dec/23

Lab Program - 1

Swapping Using Pointers:

```
#include <stdio.h>
void Swap (int *a, int * b) {
    int temp = *a;
    int *a = * b;
    *b = temp;
}
int main () {
    int num1, num2;
    Printf("Enter the first number:");
    Scanf ("%d", num1);
    Printf ("Enter the Second number:");
    Scanf ("%d", num2);

Printf ("Before swapping: num1 = %d,
num 2 = %d \n", num1, num 2 );
Swap (&num 1., &num 2) ;
Printf ("After Swapping : num1 = %d,
num 2 = %d \n", num1, num2);
return 0;
}
```

Output: Enter the first number : 10
Enter the Second number : 15

Before Swapping : num 1 = 10, num 2 = 15
After Swapping : num 1 = 15, num 2 = 10

Prg - 2

dynamic memory allocation

```
#include <stdio.h>
void MallocEx(int);
Void CallocEx (int);

Void main ()
{
  int n ;
  Printf (" Enter the value of n \n");
  Scanf ("%d", &n);
  MallocEx (n);
  CallocEx (n);
}
Void MallocEx (int n)
{
  int *Ptr;
  int i;
  int arr [n];
  Ptr = (int *) malloc (n * sizeof(int));
  for (i=0 ; i<n ; i++)
  {
    Ptr [i] = i+1;
  }
```

```
Printf ("malloc dynamic memory allocation\n");
Printf ("the Elements of array are :\n");
for (i=0 ; i<n ; i++)
  {
    Printf ("%d", Ptr [i]);
  }

  Printf ("malloc dynamic memory
          allocation\n");
  Printf (" the Elements of array are :\n");
  for  (i=0 ; i<n ; i++)
    {
      Printf ("%d", Ptr [i]);
    }
  Printf ("\n");
  free (Ptr);
}
void CallocEx (int n)
{
  int *Ptr;
  int i;
  int arr[n];
  Ptr = (int*) calloc (n, sizeof (int));
  for (i=0 ; i<n ; i++)
  {
    Ptr [i] = i + 1;
  }
```

Output

```
Printf("Calloc dynamic memory
        allocation \n");
Printf("the Elements of array are: \n");
for (i=0; i<n; i++)
  {
    Printf("%d", Ptr[i]);
  }
Printf("\n");
Printf("\n");
Printf("Realloc dynamic memory allocation \n");
Printf("the Elements of array are: \n");
n=15;
Ptr = (int *) realloc (Ptr, n * sizeof(int));
for (i=10; i<n; i++)
  {
    Ptr[i] = i+1;
  }
for (i=0; i<n; i++)
  {
    Printf("%d", Ptr[i]);
  }
free(Ptr);
}
```

output: Enter number of elements : 5

memory number Successfully allocated using malloc

The Elements of array are : 1, 2, 3, 4, 5

memory Successfully allocated using Calloc

The Elements of array are : 1, 2, 3, 4, 5

Enter the new size of array : 10

memory Successfully re-allocated using realloc

The Elements of array are : 1, 2, 3, 4, 5, 6, 2, 8, 9, 10.

Program - 3

Stack Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int top = -1;
int inp_array [SIZE];
void Push();
void POP();
void Show();
void main()
{
  int ch;
  while (1)
  {
  Printf ("Operations on Stack:\n");
  Print("1:1. Push the elements to \n2. POP the
         Element \n3. Show \n4. End \n");
   Scanf ("%d", &ch);
  Switch (ch)
   {
    case 1;
       Push();
       break;
    case 2:
       POP()
       break;
```

```
        case 3:
            Show()
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }
}
void Push()
{
int x;
if (top == SIZE-1)
  {
    printf("overflow\n");
  }
  else
  {
    printf("Enter the element to be
      added in stack:\n");
      scanf("%d", &x);
      top = top+1
      inp_array[top] = x;
  }
}
void pop()
    if (top == -1)
```

```
{
  Printf ("Underflow \n");
  }
 Else
 {
  Printf ("Popped Element: %d \n",
        inp_array [top] );
        top = top - 1;
 }
}
void Show ()
{ if (top == -1)
 {
  Printf ("Underflow \n");
 }
 Else
 {
 Printf ("Elements in stack are \n");
 {
 Printf ("%d \n", inp_array[i]);
 }
}
```

Output

Operations on Stack;

1) Push the Element

2) POP the Element

3) Show

4) End

Enter the choice;

3

Underflow

Operations on Stack;

1. Push the Element;

2) POP the Element

3) Show

4) End

Enter the choice

1

Enter the Element to be added in Stack

5

Operations on Stack

1) Push the Element

2) POP the Element

3) Show

4) End

Enter the choice

Enter the Element to be added in Stack:

3

Operations on Stack:

1) Push the Element

2) POP the Element

3) Show

4) End

Enter the choice:

2

Enter the Element to be removed:

3

POPPed Element- 3.

Week - 3
lab Program - 3

Writ a Program to Convert a given Valid Parenthesized infix arithmetic Expression to Postfix Expression. The Expression Consists of Single Character Operands & binary Operators + (Plus), - (minus), * (multiply) / (divide) and ^ (POWER).

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

# dyine MAX 100

Char Stack [MAX];
Char infix [MAX];
Char Postfix [MAX];
Int top = -1;

Void Push (Char);
Char POP ()
int isEmpty ();
void inToPost ();
Void Print ();
int Precendence (char)

int main ()
```

```
{
  Printf ("Enter infix Expression :");
  gets (infix);
  InTo Post ();
  Print ();
 return 0;
}

Void InTo Post ()
{
  int i, j = 0;
  Char Symbol, next;
  for (i=0; i< Strlen (infix); i++)
  {
     Symbol = infix[i];
     Switch (Symbol);
     break;
  Case ')':
     While ((next = POP()) != '(')
          Post fix [j++] = next;
         break;
   Case '+':
   Case '-':
   Case '*':
   Case '/':
 While (!is Empty () && Precendence (Stack >[top])
        >= Precendence (symbol))
```

```
        postfix[j++] = pop();
    push(symbol);
    break;
default:
    postfix[j++] = symbol;
  }
}
while(precedence(char symbol))
{
  switch(symbol)
  {
  case '^':
    return 3;
  case '/':
  case '*':
    return 2;
  case '+':
  case '-':
    return 1;
  default:
    return 0;
  }
}
```

```
void Print()
{
    int i = 0;
    Printf ("The Equivalent Postfix Expression
            is :");
    while (Postfix[i])
    {
        Printf ("%c", Postfix[i++]);
    }
    Printf ("\n");
}
void Push (char c)
{
    if (top == MAX-1)
    {
        Printf (" Stack Overflow ");
        return;
    }
    top++;
    Stack [top] = c;
}
char Pop()
{
    char c;
    if (top == -1)
```

```
{
    Printf ("Stack Underflow");
    exit (1);
}
c = stack [top];
top = top - 1;
return c;
}
int isEmpty ()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
```

18/11/24

Output:

```
a
Enter infix Expression : a* b +c*d-e
```

The equivalent Postfix expression is

a b * cd * + c -

Lab Program - 3 (Week - 3) 28/12/23

> Write a Program to Convert a given Valid Parenthesized infix arithmetic Expression to Postfix Expression. The Expression consists of Single character Operands & binary operators and power Write a Program to demonstrate Postfix Evaluation;

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 100
int stack [MAX_SIZE];
int top = -1;
void push (int element)
{
  if (top >= MAX_SIZE - 1)
  {
    Printf ("Stack Overflow");
      return;
  }
  else {
    stack [++top] = element;
  }
}
int pop ()
{
  if (top < 0) {
    Printf ("Stack Underflow \n");
    return -1;
  }
  else {
    return
  stack [top--];
```

NP 18/1/24

```
};
int Evaluate Postfix (char* exp) {
    int i, result;
    int len = Strlen (exp);
    for (i=0 ; i<len ; i++)
  {
   if
   (isdigit (exp [i])) {
     Push (exp [i] - '0');
     }
      Else {
   int Operator operand2 = POP ();
    int operand 1 = POP ();
  Switch (exp[i])
  {
      Case '+';
Push (operand 1 + operand2);
  break;
    Case '-':
 Push (operand 1 - operand 2);
  break;
 Case '*';
Push [operand 1 * operand 2);
      break;
 Case '/';
```

```
            Push (Operand1 / Operand2);
            break;
        }
    }
}
result = Pop();
return result;
}
int main() {
    char exp[MAX_SIZE];
    printf("Enter the Postfix Expression: ");
    scanf("%s", exp);
    int res = EvaluatePostfix(exp);
    printf("Result = %d\n", res);
    return 0;
}
```

Output:

```
Enter the Postfix Expression:
23* 5+
Result: 11
```

Week-3

Program - 3

> Write a Program to demonstrate Postfix Evaluation:

> WAP to Simulate working of a Queue of Integers using an array

```
#include <stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int Queue_array [MAX];
int rear = -1;
int front = -1;
main()
{
   int choice;
   while (1)
   {
    Printf(" insert Element to Queue \n");
    Printf (" Delete Element from Queue \n");
    Printf (" Display all Elements of Queue \n");
    Printf (" Quit \n");
    Printf ( " Enter your choice");
    Scanf ("%d", &choice);
    Switch (choice)
    {
     Case 1;
     insert();
     break;
```

```
        case 2:
        delete();
        break;
        case 3:
        display();
        break;
        case 4:
        exit();
        default:
        Printf("Wrong choice \n");
        }
    }
}
void insert()
{
    int add_item;
    if (rear == MAX-1)
    Printf ("Queue Overflow \n");
    else
    {
    if (front == -1)
    front = 0;
    Printf ("Insert the element in Queue:");
    Scanf ("%d", &add_item);
    rear = rear + 1;
    Queue_array [rear] = add_item;
    }
}
```

NLP 18/11/2024

```
void delete()
{
 if (front == -1 || front > rear)
 {
  Printf ("Queue Underflow \n");
  return;
 }
else
{
 Printf ("Element deleted from
  Queue is : %d \n", Queue_array[front]);
    front = front + 1;
  }
}
 void display ()
{
  int i;
  if (front == -1)
   Printf ("Queue is Empty \n");
 else
 {
  Printf ("Queue is : \n");
   for (i = front; i <= rear ; i++)
    Printf ("%d", Queue_array[i]);
    Printf ("\n");
 }
}
```

Output

1. insert Element to Queue
2. Delete Element from Queue
3. display all Elements of Queue
4. Quit

Enter your choice : 1

Insert the Element in Queue : 42

Enter your choice : 3

Queue is 42

Enter your choice: 2

Element deleted from Queue is 42

Enter your choice 4

Quit

Week - 3 11/01/24

Implementation of circular Queue

```
#include <stdio.h>
#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;
int isFull() {
  if ((front == rear+1) || (front == 0 &&
      rear == SIZE-1)) {
    return 1;
    return 0;
}
int isEmpty() {
  if (front == -1)
    return 1;
    return 0;
}

Void EnQueue (int element) {
  if (isFull())
  Printf ("\n Queue is full!!\n");
  Else {
  If (front == -1) {
    front = 0;
  rear = (rear+1) % SIZE;
  Item[rear] = element;
```

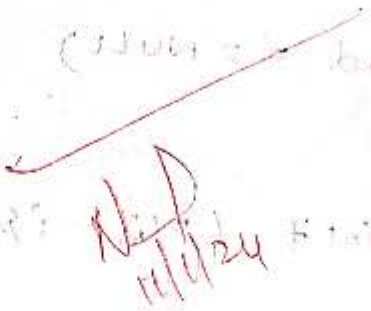
```
Printf ("\n Inserted -> %d", element);
    }
  }
}
int deQueue (){
int element;
if (isEmpty()) {
Printf ("\n Queue is Empty!! \n");
return (-1);
} else {
element = items [front];
  if (front == rear) {
  front = -1;
  rear = -1;
} else {
front = (front + 1) % SIZE;
}
Printf ("\n Delete element -> %d \n",
    element);
return (element);
}
}
```

```
Void display ()
{
int i;
if (isEmpty ())
Printf ("\n Empty Queue \n");
else {
Printf ("\n Front -> %d", front);
Printf ("\n Items -> ");
for (i = front; i != rear; i = (i+1) % SIZE)
Printf ("%d", Items [i]);
}
Printf ("%d", Items [i]);
Printf ("\n Rear -> %d \n", rear);
}
}
int main () {
En Queue (1);
En Queue (2);
En Queue (3);
En Queue (4);
En Queue (5);
display ();
```

```
deQueue (6);
deQueue (7);

display ();

return 0;

}
```

Output

Inserted → 6
Inserted → 7
Front → 2
Items → 3 4 5 6 7.
Rear → 1



Week - 3 11/1/24

Insertion Operation on Linked List

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
    int data;
    Struct node * next;
};
Void display ();
Void insert_bugin ();
Void insert_End ();
void insert_POS ();
Struct node * Ptr;
if (head == NULL)
  {
    Print ("List is Empty \n");
    return;
  }
Else
  {
Ptr = head;
While (Ptr != NULL)
{
```

```
        Printf ("%d \n", Ptr -> data);
        Ptr = Ptr ->  next;
      }
   }
}
void  insert_begin() :
{
   Struct node* temp;
   temp  = (Struct node*)  malloc
         (Sizeof (Struct node));
   Printf ("Enter the value to be
            Entered \n");
   Scanf ("%d", &temp -> data);
   temp -> next = NULL;}
   if (head = temp);
   if (head = NULL)
   head = temp;
   else {
      temp -> next = head;
      head = temp;
      }
   }
void  insert_end()
{
   Struct node * temp, * Ptr;
   temp =  (Struct node*) malloc
         (Sizeof (Struct node));
   Printf ("Enter the value to be
            Inserted \n");
```

```
temp -> next = NULL;
if (head = NULL)
{
  head = temp;
}
else
{
 ptr = head;
 while (ptr -> next != NULL)
 {
   ptr = ptr -> next;
  }
 ptr -> next = temp;
}
};
void insert_pos()
{
 int pos, i;
 struct node * temp, * ptr;
 Printf (" enter the Position ");
 Scanf ("%d", &pos);
 temp = (struct node *) malloc (size of
                  (struct node));
Printf (" enter the value to be
          inserted \n");
```

```
Scanf ("%d", &temp->data);
temp->next = NULL;
if (pos==0)
{
    temp->next=head;
    head=temp;
}
Else {
for (i=0, ptr=head; i<pos-1; i++)
{
    ptr=ptr->next;
}
temp->next=ptr->next;
}
temp->next=ptr->next;
ptr->next=temp;
void main()
{
int choice;
while(1)
{
Printf("\n 1. to Insert at beginning\n
2. to insert at end \n,
3 to insert at position
4. to display \n 5. exit");
```

```
switch (choice)
{
 case 1:
   insert_beginning();
   break;
 Case 2:
   insert_end();
   break;
 case 3;
   insert_pos();
   break;
 Case 4;
   display();
   break;
 case 5:
   exit(0);
   break;
 default;
   Print f("Invalid choice \n");
   break;
}
}
}
```

ALP 11/12/24

Output.

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Print list display
5. Exit

Enter your choice : 1
Enter data to insert at beginning : 35

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. display
5. Exit

Enter your choice : 3
Enter data to insert ~~at Beginning~~ : 57
Enter the position to insert at : 2

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. display ~~exit~~
5. Exit

Enter your choice 2
Enter data to ~~detect~~ insert at End : 15

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. display
5. Exit

Enter your choice : 4
linked list : 35 57 15

Enter your choice : 5

Exit

LeetCode : -1 1114

Week - 3

```
#include <stdlib.h>

typedef Struct{
  int * Stack;
  int * minStack;
  int top
  } MinStack;

MinStack * minStackCreate (){
MinStack * Stack = (MinStack*) malloc
   (sizeof(MinStack));
Stack -> stack = (int*) malloc (size of(int)*
          10000);

Stack -> minStack = (int* ) malloc (size of (int) * 10000;

Stack -> top = -1;
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 minStackTop (MinStack* obj) {
    return obj->stack[obj->top];
}
int minStackGetMin (MinStack* obj) {
    return obj->minStack[obj->top];
}
void minStackFree (MinStack* obj) {
    free (obj->stack);
    free (obj->minStack);
    free (obj);
}
```

Output: ["MinStack", "Push", "Push", "Push", "getMin", "Pop", "top", "getMin"]

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

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

NLP 18/1/24

> Lab Program 5: deletion

Singly displayed linked list delete and display Implementation.

```
#include <stdio.h>
#include <stdlib.h>
 Struct node {
 int data;
 struct node *next;
 };
 Void display ();
 Void insert_begin();
 Void insert_End();
 Void insert_Pos();
 Void begin_delete();
 Struct node *head = NULL;
 Void display()
{
   Printf ("elements are : \n");
   Struct node *Ptr;
   If (head == NULL)
   {
      Print ("list is empty");
      return;
   }
Else {
      Ptr = head;
      While (Ptr != NULL)
```

```
    {
      Printf ("%d\n", Ptr->data);
      Ptr = Ptr -> next;
    }
  }
}
void insert_beginc)
{
  Struct node * temb;
  temb = (Struct node*) malloc (sizeof(Structnode));
  printf ("enter the value to be inserted\n");
  Scanf ("%d", &temb->data);
  temb -> next = NULL;
  if (head == NULL)
      head = temb);
  Else {
      temb -> next = head;
      head = temb;
    }
  }
void insert_end()
{
  Struct node * temb, * Ptr;
  temb = (Struct node*) malloc (sizeof(Structnode));
  printf ("enter the value to be inserted\n");
  temb -> next = NULL;
  If (head == NULL)
  {
    head = temb;
  }
  Ues
    {
```

```
        ptr = head;
        while (ptr -> next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> next = temp;
    }
}
void insert_pos()
{
    int pos, i;
    struct node *temp, *ptr;
    Printf ("Enter the position");
    Scanf ("%d", &pos);
    temp = (struct node*) malloc (sizeof (struct node));
    Printf ("enter the value to be inserted\n");
    Scanf ("%d", &temp -> data);
    temp -> next = NULL;
    if (pos == 0)
    {
        temp -> next = head;
        head = temp;
    }
    else
    {
        for (i = 0, ptr = head; i < pos - 1; i++)
        {
            ptr = ptr -> next;
        }
        temp -> next = ptr -> next;
        ptr -> next = temp;
    }
}
```

```
void begin_delete()
{
  struct node *Ptr;
  if (head == NULL)
  { Printf ("\n list is empty \n");
  }
  else
  {
    Ptr = head;
    head = Ptr -> next;
    free (Ptr);
    Printf ("\n Node deleted from the
            begining ... \n");
  }
}
void last_delete()
{
  struct node *ptr, *ptr1;
  if (head == NULL)
  {
    Printf ("\n list is empty");
  }
  else if (head -> next == NULL)
  {
    head = NULL;
    free (head);
    Printf ("\n Only node of the list
            deleted ... \n");
  }
else
  {
    Ptr = head;
```

```
while (Ptr -> next != NULL)
{
  Ptr1 = Ptr;
  Ptr = Ptr -> next;
}
Ptr1 -> next = NULL;
free (Ptr);
Printf ("\n Deleted 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 ("\n can't delete");
    return;
  }
}
Ptr1 -> next = Ptr -> next;
free (Ptr);
Printf ("\n Deleted node %d", loc+1);
}
```

```
void main ()
{
  int choice;
  while(1)
  {
  Printf ("\n1. to insert at beginning \n"
  "2. to insert at end \n"
  "3. to insrt at Position \n"
  "4. to display \n"
  "5: delete from beginning\n"
  "6. delete from end\n"
  "7. random delete \n"
  "8. Exit \n");
Printf ("Enter your choice: \n");
  Scanf("%d", &choice);
  Switch (choice)
  {
  case 1:
      insert_begin();
      break;
  case2:
      insert_end();
      break;
  case 3:
      insert_Pos();
  case 4:
      display();
      break;
  case 5:
      begin_delete();
      break;
```

```
        case 6;
            last_delete();
            break;
        case 7;
            random_delete();
            break;
        case 8;
            exit(0);
            break;
        default;
            printf("invalid choice \n");
            break;
        }
    }
}
```

output:

1. to insert at beginning
2. to insert at End
3. to ingest at Position
4. to display
5. delete from beginning
6. delete from end
7. delete random delete
8. exit

Enter your choice: 1.
Enter the value to be inserted : 10

Enter your choice: 2
Enter the value to be inserted : 20

Enter your choice : 3
Enter the position : 1
Enter the value to be inserted : 15

Enter your choice : 4
Elements are : 10
15
20

Enter your choice : 5
Node deleted from the beginning...

Enter your choice : 7
Enter the location of node after which you want to Perform deletion : 1
Deleted node 2

Enter your choice : 6
Only node of the list is deleted

Enter your choice : 4
list is empty.

Enter your choice : 8

Exit

NP
18/1/24

Leetcode Program - 2 18/Jan/24

On Singly linked list given in link.

```
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* Prev = dummy;
for (int i=1; i<left; ++i){
      Prev = Prev->next;
   }
Struct ListNode* current = Prev->next;
Struct ListNode* next = NULL;
Struct ListNode* tail = current;

for (int i = left; i<= right; ++i){
      Struct ListNode* temp = current->next;
       current->next = next;
      next = current;
      current = temp;
     }
    Prev->next = next;
    tail->next = current;

    Struct ListNode* result = dummy->next;
    free(dummy);
    return result;
}
```

output:

Input: Case 1

head =

[1, 2, 3, 4, 5]

left = 2

right = 4

output: [1, 4, 3, 2, 5]

Case 2

head = [5]

left = 1

right = 1

output: [5]

NA
25/1/24

Week-D
Lab-6

25/Jan/24

1. WAP to Implement Single linked list with following Operations Sort the linked list, Reverse the linked list, Concatenation of two linked list.

```
#include <stdio.h>
#include <stdlib.h>
Struct Node {
        int data;
        struct Node * next;
      };
Void append (Struct Node** head_ref, int new_data){
    Struct Node* new_node = (Struct Node*) malloc (sizeof
                            (Struct Node));
    Struct Node* last = * head_ref;
    Struct Node* new_node -> data = new_data;
                 new_node -> next = NULL;
    if (* head_ref == NULL) {
        * head_ref == new_node;
         return;
      }
    While (last -> next != NULL) {
         last = last -> next;
     }
     last -> next = new_node;
  }
Void PrintList (Struct Node* node) {
      While (node != NULL) {
         Printf ("%d -> ", node -> data);
         node = node -> next;
      }
    Printf ("NULL\n");
}
Void SortList (Struct Node** head_ref) {
      if (* head_ref == NULL) {
          return;
       }
    int Swapped, temp;
    Struct Node* ptr1;
    Struct Node* lPtr = NULL;
```

```
do {
    swapped = 0;
    ptr1 = *head_ref;
    while (ptr1->next != lptr) {
        if (ptr1->data > ptr1->next->data) {
            temp = ptr1->data;
            ptr1->data = ptr1->next->data;
            ptr1->next->data = temp;
            swapped = 1;
        }
        ptr = ptr1->next;
    }
    lptr = ptr1;
} while (swapped);
}

void reverseList (struct Node** head_ref) {
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

void concatenateLists(struct Node** head1,
        struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}
int main
```

25/11/24

```
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n, data;
    printf("Enter the number of elements for list 1:");
    scanf("%d", &n);
    printf("Enter the elements for list 1:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&list1, data);
    }
    printf("Enter the number of elements for list 2:");
    scanf("%d", &n);
    printf("Enter the elements for list 2:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&list2, data);
    }
    printf("\nOriginal list 1:");
    printList(list1);
    printf("Original list 2:");
    printList(list2);
    sortList(&list1);
    sortList(&list2);
    printf("\nSorted list 1:");
    printList(list1);
    printf("Sorted list 2:");
    printList(list2);
    concatenateLists(&list1, list2);
    printf("\nConcatenated list:");
    printList(list1);
    printf("\nReversed list:");
    printList(list1);
    return 0;
}
```

Output:

Enter the number of Element for list 1 : 3
Enter the Elements for List 1:
10
20
30
Enter the number of Elements for list 2 : 3
Enter the Elements for List 2:
40
50
60
original List 1: 10 → 20 → 30 → NULL
original List 2: 40 → 50 → 60 → NULL

Sorted List 1: 10 → 20 → 30 → NULL
Sorted List 2: 40 → 50 → 60 → NULL

Concatenated List: 10 → 20 → 30 → 40 → 50 → 60 → NULL
Reverse List: 60 → 50 → 40 → 30 → 20 → 10 → NULL

Program - 2

```
#include <stdio.h>
#include <stdlib.h>
Struct Node {
    int data;
    Struct Node* next;
};
Void Push (Struct Node** top_ref, int new_data){
    Struct Node* new_node = (Struct Node*) malloc
        (sizeof (Struct Node));
new_node -> data = new_data;
new_node -> next = *top_ref;
*top_ref = new_node;
}

int Pop (Struct Node** top_ref) {
    if (*top_ref == NULL) {
    Printf ("Stack is Empty. cannot Pop.\n");
    return -1;
}
```

```
Struct Node* temp = *top_ref;
int PoPped_data = temp -> data;
*top_ref = temp -> next;
free (temp);
return Popped_data;
}
void PrintStack (struct Node * top) {
    Printf ("Stack :");
    while (top! = NULL) {
    Printf ("%d -> ", top -> data);
    top = top -> next;
    }
    Printf ("NULL\n");
}
int main() {
    Struct Node *  StackTop = NULL;
    int choice, data;
do {
    Printf ("\n Stack Menu:\n");
    Printf ("1. Push \n");
    Printf ("2. Pop \n");
    Printf ("3. Print Stack\n");
    Printf ("4. Exit \n");
    Printf (" Enter your choice: ");
    Scanf ("%d", &choice);
Switch (choice) {
    case 1:
        Printf ("Enter the element to Push:");
        Scanf ("%d", &data);
        Push (&StackTop, data);
        Printf ("Element %d Pushed onto the Stack.\n
        break;
    Case 2:
        data = Pop (&StackTop);
        if (data ! = -1) {
            Printf ("Element %d Popped from the
                Stack . \n", data);
        }
        break;
```

Week -6
Lab - C
Reg - 2

25/Jan/2024

```
Case 3:
    PrintStack (StackTop);
    break;

Case 4:
    Printf ("Exiting the Program.\n");
    break;

default:
    Printf ("Invalid Choice. Please Enter a valid
            Option \n");
    }
} while (Choice != 4);

While (StackTop != NULL) {
    Struct Node* temp = StackTop;
    StackTop = StackTop -> next;
    free (temp);
}
return 0;
}
```

Output: Stack menu:

1. Push
2. POP
3. Print Stack
3. Print Stack
4. Exit

Enter your choice: 1
Enter the Element to Push: 10
Element 10 pushed onto the stack

Enter Your choice - 1
Element Enter the Element to Push: 20
Element 20 Pushed onto the Stack
Enter your choice - 1
Enter the Element to Push: 30
Element 30 Pushed onto the Stack
Enter your Choice - 2
Enter the Element 30 Popped from Pushed onto the Stack.
Enter your Choice - 3
Stack : 20 → 10 → NULL
Enter your Choice 4.
Exit.

29 Jan 2024

Week-6

Lab-6

Program-3

```
#include<stdio.h>
#include<stdlib.h>
Linked
Struct Node {
    int data;
    Struct Node * next;
};
Element into the Queue
Void Enqueue (Struct Node ** front_rf,
Struct Node ** rear_rf, int new_data)
{
  Struct Node * New_node = (Struct Node*)
  malloc (Size of (Struct Node));
    new_node -> data = new_data;
    new_node -> next = NULL;

  if (* rear_rf == NULL) {
    * front_rf = * rear_rf = new_node;
    } else {
      (* rear_rf) -> next = new_node;
      * rear_rf = new_node;
    }
}
from int dequeue (Struct Node ** front_rf,
    Struct Node ** rear_rf) {
    if (* front_rf == NULL) {
    Printf ("Queue is Empty. Cannot dequeue\n");
    return -1;
    }
```

```
struct Node * temp = *front_ref;
int dequeued_data = temp->data;

*front_ref = temp->next;
if (*front_ref == NULL)
    *rear_ref = NULL;
free(temp);
return dequeued_data;
}
void PrintQueue (struct Node *front) {
    Printf("Queue:\n");
    while (front != NULL) {
        Printf("%d -->", front->data);
        front = front->next;
    }
    Printf("NULL \n");
}
int main() {
    struct Node * QueueFront = NULL;
    struct Node * QueueRear = NULL;
    int choice, data;
```

25/11/24

```
do {
    Printf("\n Queue Menu : \n");
    Printf("1. Enqueue \n");
    Printf("2. Dequeue \n");
    Printf("3. Print Queue \n");
    Printf("4. Exit \n");
    Printf("Enter your choice: ");
    Scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf ("Enter the element to Enqueue: ");
        scanf ("%d", &data);
        Enqueue (&queueFront, &queueRear, data);
        printf ("Element %d Enqueued into the
                Queue.\n", data);
        break;
    case 2:
        data = dequeue (&queueFront, &queueRear);
        if (data != -1) {
        printf ("Element %d dequeued from the Queue
        break;
    case 3:
        PrintQueue (queueFront);
        break;
    case 4:
        printf ("Existing the Programming");
        break;
    case 5:
        printf ("Existing the Programming");
        break;

    default:
        printf ("Invalid choice. Please enter a
                valid option.\n");
    }
} while (choice != 4);
While (queueFront != NULL) {
    struct node* temp = queueFront;
    queueFront = queueFront->next;
    free (temp);
```

```
}
return 0;
}
```

Output: Queue menu:
1) Enqueue
2) Dequeue
3) Print Queue
4) Exit

Enter your choice: 1
Enter the element to Enqueue: 10
Element 10 enqueued into Queue.

Queue Menu:-
Enter your choice: 1
Enter the element to Enqueue: 20
Element 20 enqueued into Queue.

Enter your choice: 1
Enter the element to Enqueue: 30
Element 30 enqueued into Queue.

Enter your Choice: 2
Element 10 dequeued from the Queue.

Enter your choice: 3
Queue: 20 → 30 → NULL

Enter your choice: 4
Exit.

Week-7

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
   Struct node *Prev;
   Struct node *next;
   int data;
};
Struct node * head;
void insertion_beginning()
{
 Struct node *Ptr;
 int item;
 Ptr = (Struct node *) malloc (sizeof (Struct node));
 if (Ptr == NULL)
 {
    Printf ("\n OVERFLOW");
 }
 else
 {
  Printf ("\n Enter 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 ("\n Node inserted \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 ("\n Can't delete \n");
}
Else if (Ptr -> nex -> next == NULL)
{
  Ptr (-> next = NULL;
}
  Else
{
  temp = Ptr -> next;
Ptr -> next = temp -> next;
```

```
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 main()
{
  Int choice = 0;
  While (choice != 9)
  {
  Printf("\n Choose one option from
        the following list ... \n");
  Printf("\n1. Insert in beginning \n
2. Delete the node after the given
   data
3. Show \n
4. Exit \n");
  Printf("\n Enter your choice ? \n");
  Scanf("\n %d", &choice);
  Switch (choice)
{
  Case 1;
```

```
insertion_beginning();
break;
case 2:
deletion_specified();
break;
case 3:
display();
break;
case 4:
Exit(0);
break;
default:
Printf("Please Enter valid choice..");
}
}
}
```

Output: 1. Insert a node
2. Delete a node
3. display the list
4. Exit

Enter your Choice: 1
Enter data for new node: 10

Enter your choice: 1
Enter data for new node: 20

Enter your Choice: 1
Enter data for new node: 30

Enter your choice: 2
Enter the value of node to delete: 10
Node with value 10 deleted successfully.

Enter your choice: 3.
Doubly linked list : 20 → 30 → NULL

Menu:

1. Insert a node
2. Delete a node
3. Display the list
4. Exit

Enter your Choice: 4

15/11/2024

Week - 8

1) Construct a binary Search tree, Traverse the tree using all 3 methods and display the elements in the tree.

```
#include <Stdio.h>
#include <Stdlib.h>
Struct node {
int value;
struct node * left;
Strict node * right;
} * root = NULL, * temp = NULL, *t2, *t1, *tP;

int key;
int s;
int n;
int count;
Void Search (Struct node *t);

Void insert ()
{
 int data;
Printf ("Enter data to be inserted");
Scanf ("%d", &data)
temp = (Struct node *) malloc (sizeof((Struct nod));
 temp -> value = data;
 temp -> left = temp -> right = NULL;
 if (root == NULL)

  root = temp;
  else
  Search (root);
}
```

```
void search (struct node *t)
{
  if ((temp -> value > t -> value) && (t -> right != NU
      search (t -> right);
  else if ((temp -> value < t -> value) && (t -> right != N
      t -> right = temp;
  else if ((temp -> value < t -> value) && (t -> left != NULL)
      search (t -> left);
  else if ((temp -> value < t -> value) && (t -> left == NULL)
      t -> left = temp;
}

void inorder (struct node *t)
{
  if (root == NULL)
  {
    printf ("No elements in the tree \n");
    return;
  }
  if (t -> left != NULL)
    inorder (t -> left);
  printf ("%d", t -> value);
  if (t -> right != NULL)
    inorder (t -> right);
}

void preorder (struct node *t)
{
  if (root == NULL)
  {
    printf ("No elements in tree \n");
    return;
  }
```

```
if (t->left != NULL)
  Postorder (t->left);
if (t->right != NULL)
Postorder (t->right);
Printf ("%d -> ", t->value);
}
Struct node *maxvaluenode (struct node *t)
{
  Struct node* current = t;
While (current && current->right != NULL)
    current = current->right;
return current;
}
int main ()
{
  int choice;
  while (1)
  {
  Printf ("\n*** menu ***\n");
  Printf ("1. insert an Element into tree\n");
  Printf ("2. to get the max value in tree\n");
  Printf ("3. to Print the tree Elements in order traversal\n");
  Printf ("4. to Print the tree Elements in Preorder traversal\n");
  Printf ("5. to Print tree Elements in Postorder traversal\n");
  Printf ("6. to exit\n");
  Printf ("Enter your choice\n");
  Scanf ("%d", &choice);
  }
```

```
Switch (choice)
{
case 1 :
  insert ();
  break;
case 2 :
  tp = maxvalue node (root);
  Printf ("***** Max Value Node *****");
  Printf ("Node with Maximum Value = %d", tp->value);
  break;

case 3 :
  Printf ("**** inorder traversal ***");
  inorder (root);
  break;

case 4 :
  Printf ("*** Order traversal **");
  Preorder (root);
  break;

case 5 :
  Printf ("** postorder * traversal **");
  Postorder (root);
  break;

case 6 :
  exit(0)
default :
  Printf ("Invalid choice");
  break;
}
break return 0;
}
```

Leetcode 3

```
/**
 * Definition for singly-linked list.
 struct ListNode {
     int val;
     struct ListNode *next;
 };
 */

/**
 * Note: The returned array must be malloced,
 assume caller calls free().
 */
struct ListNode** splitListToParts(struct ListNode*
 head, int k, int* returnSize) {
    int length = 0;
    struct ListNode * current = head;
    while (current != NULL) {
        length++;
        current = current->next;
    }
int partSize = length / k;
int extraNodes = length % k;
struct ListNode** result = (struct ListNode*)malloc
        (k * sizeof(struct ListNode));
current = head;
for(int i = 0; i < k; i++) {
int currentPartSize = partSize + (i < extraNodes ?
        1:0);
```

dividing length by k to determine avg size of each part

extra nodes that need to be distributed among parts

```
result[i] = current;
for (int j=0; j < currentPointSize - 1 &&
        current != NULL; j++) {
    current = current -> next;
}
if (current != NULL) {
    struct ListNode* nextNode = current -> next;
    current -> next = NULL;
    current = nextNode;
}
}
*returnSize = k;
return result;
}
```

Output: input = head = [1, 2, 3]

output: [[1], [2], [3], [], []]

[[1], [2], [3], [], []]

output: *** menu ***

1) insert an element into tree
2) to get the max value
3) to Print the tree elements in inorder Traversal
4) to Print the tree elements in Preorder Traversal
5) to Print the tree Elements in Postorder traversal
6) to Exit

Enter your choice:
1. insert an Element into tree

~~Enter your choice~~.
Enter data to be inserted
20

Enter your choice: 1
Enter the data to be inserted
30
Enter your choice: 1
enter the data to be inserted
10

Enter your choice: 2
Node with max value is = 30

Enter your choice: 3
inorder traversal
10 → 20 → 30 →

Enter your choice : 4

Preorder traversal

20 → 10 → 30 →

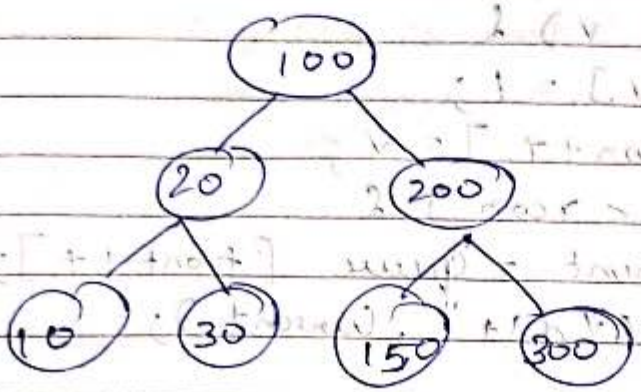
Enter your choice : 5

Postorder traversal

10 → 30 → 20 →

Enter your choice : 6

Exit



Inorder traversal

left, root, right

10 → 20 → 30 → 100 → 150 → 200 → 300 →

Preorder traversal

Root, left, right

100 → 20 → 10 → 30 → 200 → 150 → 300 →

Postorder traversal

left, right, root

10 → 30 → 20 → 150 → 300 → 200 → 100 →

15/2/24

22/02/24

Week : 9

BFS.

```
#include <stdio.h>
#define MAX_VERTICES 10
int n, i, j, visited[MAX_VERTICES],
    queue[MAX_VERTICES], front = 0, rear = 0;
int adj[MAX_VERTICES][MAX_VERTICES];

void bfs(int v) {
    visited[v] = 1;
    queue[rear++] = v;
    while (front < rear) {
        int current = queue[front++];
        printf("%d\t", current);

        for (int i = 0; i < n; i++) {
            if (adj[current][i] && !visited[i])
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}

int main() {
    int v;
    printf("Enter number of vertices:");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        visited[i] = 0;
    }
```

```
Printf ("Enter the graph data in 'matrix form");
        for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                Scanf ("%d", &adj[i][j]);
Printf ("Enter the Starting vertex");
Scanf ("%d", &v);
bfs(v);
for (i=0; i<n; i++)
if (!visited[i])
Printf ("In BFS it's not Possible. Not all nodes
        are reachable. \n");
return 0;
}
return 0;
}
```

Output: Enter the number of vertices: 7

Enter graph data in matrix form:

0 1 0 1 0 0 0

0 1 0 0 1 0 0

0 1 0 0 0 0 1

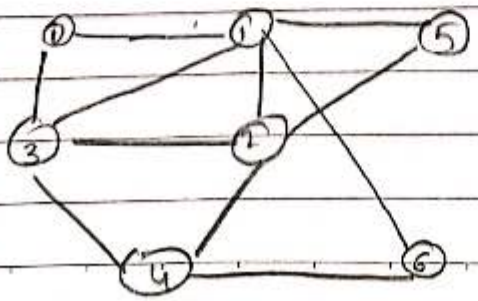
0 0 1 0 0 0 1

0 1 1 0 0 0 0

0 1 0 0 1 0 0

Enter the Starting vertex: 4

4 2 3 6 1 5 0



Visited 4, 2, 3, 6

4 → 2, 3, 6

2 → 1, 3, 4, 5, 6

1 → 0, 2, 3, 5, 6

Week - 9

DFS:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10

void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int num_vertices, bool visited[MAX_VERTICES], int vertex) {
    visited[vertex] = true;
    int i;
    for (i = 0; i < num_vertices; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, num_vertices, visited, i);
        }
    }
}

bool isConnected(int graph[MAX_VERTICES][MAX_VERTICES], int num_vertices) {
    bool visited[MAX_VERTICES] = {false};

    dfs(graph, num_vertices, visited, 0);

    for (int i = 0; i < num_vertices; i++) {
        if (!visited[i]) {
            return false;
        }
    }

    return true;
}
```

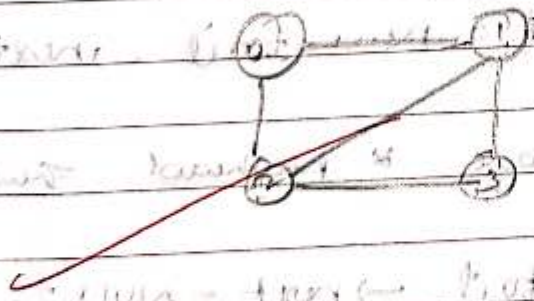
```
int main() {
    int num_vertices;
    printf("Enter number of vertices");
    scanf("%d", &num_vertices);
    int i, j;
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix:\n");
    for (i=0; i<num_vertices; ++i) {
        scanf("%d", &graph[i][j]);
    }
```

Output: Enter the number of vertices : 4

Enter the adjacency matrix:

0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0

The graph is Connected.



Leetcod-61

Rotate list

```
struct ListNode* rotateRight(struct ListNode* head, int k){
    if (head == NULL || head->next == NULL || k == 0)
    return head;
    int length = 1;
    struct ListNode *tail = head;
    while (tail->next){
        length++;
        tail = tail->next;
    }
    k = k % length;
    if (k == 0)
    return head;
    struct ListNode *new_tail = head;
    for (int i = 0; i < length - k - 1; i++)
    {
        new_tail = new_tail->next;
    }
    struct ListNode *newhead = new_tail->next;
    new_tail->next = NULL;
    tail->next = head;
    return newhead;
}
```

output

Case 1

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

k = 2

Case 2 :

head = [0, 1, 2]

k = 4

Q.7

23/2/24

22/4/2024

Hacker rank code:

```
#include <iostream>
#include <algorithm>
#include <vector>

Using namespace std;

Struct Vertex {
    int left, right;
};

typedef Vector <Vertex> Vertices;

Void SwapNodes (vertices &vs, int k,
int root, int depth, bool &Start) {
  if (depth % k == 0) {
    Swap (vs[root].left, vs[root].right);
  }
  if (vs [root].left != -1) {
    SwapNodes (vs, k, vs [root].left,
    depth+1, Start);
  }
  if (&Start) {
    Start = false;
  } else {
    Count << " ";
  }
  Count << root;
  if (vs[root].right != -1) {
    SwapNodes (vs, k, vs [root].right,
      depth+1, Start);
    }
  }
```

```
int main() {
size_t num_vertices;
cin >> num_vertices;
Vertices Vertices (num_vertices + 1);
for (size_t v = 0; v < num_vertices; ++v) {
  cin >> Vertices[v+1].left >> vertices[v+1].right;
}
size_t num_tests;
cin >> num_tests;
for (size_t t = 0; t < num_tests; ++t)
{
int k;
cin >> k;
bool start = true;
swapNodes (vertices, k, 1, 1, start);
    count << "\n";
    }
return 0;
}
```

output:

input: 3

2 3

-1 -1

2

1

1

Page No.:
Date:
YOUVA

Output: 3 1 2

2 1 3

Week - 10

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

Struct Employee {
    int key;
};
int hashFunction(int key);
Void insertEmployee(Struct Employee Employees[],
        int hashTable[], Struct Employee emp);
Void displayHashTable(int hashTable[]);
Int main() {
    Struct Employee Employees[MAX_EMPLOYEES];
    int hashTable[HASH_TABLE_SIZE] = {0};
    int n, m, i;

Printf("Enter the number of Employees: ");
Scanf("%d", &n);

Printf("Enter Number of Employees: ");
Scanf("%d", &n);
Printf("Enter Employee records: ");
Scanf("%d", &n);
Printf("Enter the number of Employees: ");
Scanf("%d", &n);

Printf("Enter Employee records: \n");
for(i=0; i<n; ++i) {
```

```
        Printf ("Employee %d : \n", i+1);
        Printf ("%d", employees[i].key);
        insert Employee (employees, hashTable, employees[i]);
    }
    Printf ("\n Hash Table: \n");
    displayHashTable (hasTable);
    return 0;
}
int hashFunction (int key) {
    return key % HashTableSize;
}
Void insertEmployee (struct Employee employees[],
    int hasTable[], struct Employee emp) {
    int index = (emp+1) % HASH_TABLE_SIZE;
    {
    hashTable[index] = emp.key;
    }
Void displayTable (int hashTable[]) {
    int i;
    for (i=0; i < HashTableSize; ++i) {
        Printf ("%d ", i);
        if (hashTable[i] == 0) {
            Printf ("Empty \n");
        } else
            Printf ("%d \n", hashTable[i]);
        }
    }
}
```

Output:

```
Enter the number of employees: 4
Enter Employee records:
Employee 1:
Enter 4-digit key: 550
Employee 2:
Enter 4-digit key: 300
Employee 3:
Enter 4-digit key: 550
Employee 4:
Enter 4-digit key: 376
```

Hash Table:

```
0 → 550
1 → 300
2 → 550
3 → Empty
4 → Empty
5 → 376
6 → Empty
7 → Empty
8 → Empty
9 → Empty
```

20/9/24