Teacher's Sign / Remarks

Programme - 1

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 100

void push(int);
void pop();
void display();

int stack[SIZE], top = -1;

void push (int value)
{
    if (top == size - 1)
        Printf("\n stack is Full");
    else
    {
        top = top + 1;
        stack[top] = value;
    }
}
void pop()
{
    if (top == -1)
    Printf("\n stack is empty");
    else
    {
        Printf("\n Deleted : %d", stack[Top]);
        top = top - 1;
    }
}
```

Evaluated
N P
1/1/2024

```
void display ()
{
   if (top == -1)
     printf ("\n Stack is empty!");
   else
   {
      int i;
      printf ("\n Stack elements are:\n");
       for(i= top; i>=0; i--)
         printf ("%d \n", stack[i]);
   }
}
Void main ()
{
   int value, choice;
   while (1)
   {
      printf ("\n\n *** Menu *** \n");
      Print f ("1. Push \n 2. Pop \n 3 Display \n 4 Exit");
      Scanf ("%d", &choice);
   Switch ( choice)
   {
      Case 1: Printf (" Enter the value to be inserted");
              Scanf ("%d", &value);
              Push ( value);
              break;
      Case 2: Pop ();
              break;
      Case 3 display();
              break;
   Case 4 exit (0);
```

OUTPUT

```
*** Menu ***
1. PUSH
2. POP
3. DISPLAY
4. EXIT
1
Enter the value to be inserted : 1
*** MENU ***
1. PUSH
2. POP
3. DISPLAY
4. EXIT
2
Enter the value to be inserted : 2
** MENU **
.
.
3
Stack Elements are
2
1
*** MENU ***
.
.
2
Deleted 2
*** MENU ***
.
.
4
```

Programme 2 (Infix to Postfix)

```
#include <stdio.h>
#include <ctype.h>
# define SIZE 30
char stack [SIZE];
int top = -1;

void Push (char ele)
{
    stack [++top] = ele; }
char Pop ()
{
    return (stack [top--]); }

int Px ( char symbol)
{
    if (symbol == '^')
        return (3);
    else if ( symbol == "*" || symbol == '/')
        return (2);
    else if ( symbol == "+" || symbol == '-')
        return (1);
    else
        return (0);
}

void main ()
{
    char infix[50], postfix[50], ch, ele;
    int i=0, k=0;
    Printf ("Enter Infix expression : ");
    scanf ("%s", infix);
    Push ("#");
while ( ch = infix [i++] ! = '\0')
{
    if ( ch == 'c')
    {
        Push (ch)
    }
    else if ( isalnum (ch))
    {
        postfix [k++] = ch;
    }
    else if ( ch == ')')
    {
        while ( stack [top] ! = 'c')
        {
            postfix [k++] = Pop ();
        }
        ele = pop ();
    }
    else
    {
        while ( Px ( stack [top]) > Px (ch) {
            postfix [k++] = Pop (); }
        Push (ch);
    }
}
while ( stack [top] ) = "#")
{
    postfix [k++] = pop ();
}
postfix [k] = '\0';
Printf ("The postfix expression is = %s",
        postfix); }
```

OUTPUT:

Enter Infix expression : a+b*c

The postfix expression is : abc*+

Programme 3

Linear Queue

```
# include <stdio.h>
# include <conio.h>
# include <process.h>
# define size 5
int item, front=0, rear=-1, q[10];
void insert()
{
  if (rear == size-1)
  {
    printf("queque overflow\n");
    return;
  }
  rear = rear+1;
  q[rear] = item;
}
int delete()
{
  if (front > rear)
    return -1;
  return q[front++];
}
void display()
{
  int i;
  if (front > rear)
  {
    printf("Queue is empty");
    return;
  }
  printf("Contents of queue\n");
  for (i=front; i<rear; i++)
  {
    printf("%d\n", q[i]);
  }
}
void main()
{
  int choice;
  while(1)
  {
    printf("\n1: insert \n2: delete \n3: display \n
      4: exit \n");
    printf("enter choice \n");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1: printf("enter the item to be
          inserted\n");
        scanf("%d", &item);
        insert();
        break;
      case 2: item = delete();
        if (item == -1)
        printf("queue is empty\n");
        else
        printf("item deleted = %d\n", item);
        break;
      case 3: display()
        break;
      default: exit(0);
    }
  }
}
```

else
{
printf("deleted item is %d", item);
}

Link programme

1) Single linked list with all scenario of insertion
2) Single linked list with all scenario of deletion

```
struct Node * create_node (int data)
{
    struct Node * newNode = (struct Node *) malloc (
        sizeof (struct Node));
    if (newNode == NULL)
    {
        printf ("Allocation failed \n");
        exit();
    }
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}

Struct Node * insertAtFirst (struct Node * head,
                             int data)
{
    struct Node * newNode = create_node(data);
    newNode -> next = head;
    return newNode;
}

Struct Node *
        insert_At position (struct Node *,
                            int data,
                            int position)
{
    if (position < 1)
```

```
printf ("Invalid position\n");
return head;
}

struct Node * newNode = create_node (data);
if (position == 1)
{
    newNode -> next = head;
    return newNode;
}
struct Node * temp = head;
for (int i = 1; i < position - 1 && temp != NULL; i++)
{
    temp = temp -> next;
}
if (temp == NULL)
{
    printf ("Invalid position\n");
    return head;
}
newNode -> next = temp -> next;
temp -> next = newNode;
return head;
}

Struct Node * insertAtEnd ( struct Node * head,
                            int data)
{
    struct Node * newNode = Create node (data);
    if (head == NULL)
    {
        return newNode;
    }
    struct Node * temp = head;
    while ( temp -> next != NULL)
    {
        temp = temp -> next;
```

```
temp -> next = newNode
 return head;
}

void displayList ( struct Node * head)
 {
    printf ("Linked List :");
    while ( head != NULL)
     {
       printf ("%d ->", head -> data);
       head = head -> next;
     }
    printf ("NULL\n");
 }

 int main()
 {
     struct Node* head = createList();
     head = insertAtFirst (head, 5)
     head = insertAtEnd (head, 4)
     head = insertAtFirst(head, 1)
     head = insertAtEnd (head, 24);
     head = insertAtFirst (head, 10);
      display_list (head);
      head = insertAtposition (head, 10, 2);
       display_list (head);
       return 0;
     }
```

OUTPUT

Linked List 10 -> 1 -> 5 -> 4 -> 24 -> NULL

Linked List 10 -> 1 -> 5 -> 4 -> 24 -> NULL

Linked List 10 -> 1 -> 10 -> 5 -> 4 -> 24 -> NULL

Programme 6

(Linked List deletion)

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

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

Struct Node* createList()
 {
    return NULL;
 }
  struct Node* createNode(int data)
   {
       Struct Node* newNode = (Struct Node*)
                              malloc(sizeof(struct
                                     Node));
     if (newNode == NULL)
       {
          printf("Allocation failed \n");
          exit(1);
       }
 newNode -> data = data;
 newNode -> next = NULL;
 return newNode;
}

Struct Node* insertAtposition (struct Node*
                               head, int data,
                               int position);

Struct Node* deleteAtBeginning (struct Node
                                * head)
```

```
{
  if (head == NULL)
  {
    printf("List is empty\n");
    return NULL;
  }
struct Node* temp = head;
head = head->next
free(temp);
return(head);
}
struct Node* deleteAtEnd(struct Node* head)
{
  if (head == NULL)
  {
    printf("List is empty");
    return NULL;
  }
  if (head->next == NULL)
  {
    free(head);
    return NULL;
  }
struct Node* temp = head;
while (temp->next->next != NULL)
  {
    temp = temp->next;
  }
free(temp->next);
temp->next = NULL;
return head;
```

```
struct Node* deleteAtPosition(struct Node* head,
                              int position)
{
  if (head == NULL || position < 1)
  {
    printf("Invalid position");
    return head;
  }
  if (position == 1)
  {
    struct Node* temp = head;
    head = head->next;
    free(temp);
    return head;
  }
struct Node* temp = head;
  for (int i = 1; i < position - 1 && temp != NULL;
       ++i);
    printf("Invalid position\n");
    return head;
  }
struct Node* nodeToDelete = temp->next;
temp->next = nodeToDelete->next;
free(head);
}
void displayList(struct Node* head)
{
  printf("Linked List:");
  while (head != NULL)
  {
    printf("%d ->", head->data);
    head = head->next;
  }
```

```
printf("NULL\n");
}
int main()
{
    struct node* head = create_list();
    head = insertAtposition(head, 1);
    head = insertAtposition(head, 2);
    head = insertAtposition(head, 3);
    head = insertAtposition(head, 2);
    display_list(head);
    head = deleteAtBeginning(head);
    display_list(head);
    head = deleteAtend(head);
    display_list(head);
    deleteAtposition(head, 2);
    display_list(head);
    return(0);
}
```

Exit of Menu Option

OUTPUT:

Linked list : 10 → 2 → 55 → 50 → 50 → 40 → NULL
Linked list : 2 → 55 → 50 → 50 → 40 → NULL
Linked list : 1 → 55 → 50 → 50 → NULL
Linked list : 1 → 50 → 50 → NULL

Lab Programme

2a) Implementation of stack using SLL

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node{
int data;
struct node* next;
};
struct node *top = NULL;
void PUSH()
{
    struct node *new_node;
    new_node = (struct node*)malloc(sizeof
                (struct node));
    printf("Enter the element\n");
    scanf("%d", &new_node->data);
    new_node->next = NULL;
    if(top == NULL)
    {
        top = new_node;
    }
    else{
        new_node->next = top;
        top = new_node;
    }
}
void pop()
{
    if(top == NULL)
        printf("Stack is empty");
    else
```

```
printf(" Deleted item %d", top->data);
    top = top->next;
  }
}
void display()
{
    struct node *temp;
     if(top == NULL)
      printf(" Stack is empty\n");
     else
     {
        temp = top;
        while(temp!=NULL)
        {
            Printf("%d\n", temp->data);
            temp = temp->next;
        }
     }
}
Void Main()
{
    int choice;
    while(1)
    {
        Printf("1. push\n");
        Printf("2. pop\n");
        Printf("3. display\n");
        printf("4. Exit\n");
        printf(" Enter your choice");
        Scanf("%d", &choice);
        Switch(choice)
        {
            case 1: Push(); break;
            case 2: Pop(); break;
            case 3: display(); break;
            case 4: Exit(0);
        }
    }
}
```

2. Implementation of queue using SLL

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front=NULL, *rear=NULL;
void insert()
{
    struct node *new_node;
    new_node=(struct node*)malloc(sizeof(struct node));
    printf("Enter the element\n");
    scanf("%d",&new_node->data);
    new_node->next=NULL;
    if(rear==NULL)
    {
        rear=new_node;
        front=new_node;
    }
    else
    {
        rear->next=new_node;
        rear=new_node;
    }
}
void del()
{
    if(front==NULL)
    printf("queue is empty");
    else {
        printf("deleted element is %d",front->data);
        front=front->next;
    }
}
void display()
{
    struct node *temp;
    if(front==NULL)
    printf("queue is empty");
    temp=front;
    while(temp!=null)
    {
        printf("%d\n",temp->data);
        temp=temp->next;
    }
}
void main()
{
    int choice;
    while(1){
        printf("1 insert\n");
        printf("2 delete\n");
        printf("3 Display\n");
        printf("4 Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: insert(); break;
            case 2: del(); break;
            case 3: display(); break;
            case 4: exit(0);
        }
    }
```

2. Sorting, concatenate, reverse SLL

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
void insertend(struct node **head, int
                value)
{
    struct node *newNode = (struct node*)malloc
                (sizeof(struct node));
    newNode->data = value;
    newNode->next = NULL;
    if(*head == NULL)
    {
        *head = newNode;
        return;
    }
    struct node* current = *head;
    while(current->next != NULL)
    {
        current = current->next;
    }
    current->next = newNode;
}
void printlist(struct node *head)
{
    while(head != NULL)
    {
        printf("%d ->", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
void insertsort(struct node** head){
if(*head == NULL || (*head)->next == NULL)
    {
        return;
    }
struct node *sorted = NULL;
struct node *current = *head;
while(current != NULL)
{
    struct node *next = current->next;
    if(sorted == NULL || sorted->data >= current->data)
    {
        current->next = sorted;
        sorted = current;
    }
    else{
        struct node* temp = sorted;
        while(temp->next != NULL && temp->
              next->data < current->data)
        {
            temp = temp->next;
        }
        current->next = temp->next;
        temp->next = current;
    }
    current = next;
}
void concatenate(struct node** list1,
                 struct node** list2)
{
    if(*list1 == NULL)
    {
        *list1 = *list2;
    }
    else
    {
        struct node* current = *li
```

```
while (current -> next != null)
{
    current = current -> next;
}
current -> next = *list2;
}
int main()
{
    struct node* list2 = null;
    struct node* list3 = null;
    insertat(&list2, 2);
    insertat(&list2, 4);
    insertend(&list2, 8);
    insertend(&list2, 10);
    insertend(&list3, 7);

    Printlist(list2);
    Reverselist(&list2);
    Printlist(list2);

Concatenatelists(&list3, list2);
    Printlist(list3);
    return 0;
}
```

1) WAP to implement doubly link list with primitive operations
a) Create doubly linked list
b) Insert a new node to left of node
c) Insert the node based on specific place

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    struct node * next;
    int data;
    struct node * prev;
};
struct node* start = NULL;
struct node* create_ll (struct node*);
struct node* insert_before (struct node*);
struct node* del_specific (struct node*);

struct node* create_ll (struct node* start)
{
    struct node * newnode, *ptr;
    int num;
    printf("Enter -1 to end");
    printf("Enter a value ");
    scanf("%d", &num);
    while (num != -1)
    {
        if (start == null)
        {
            new-node = (struct node*)
                malloc (sizeof (struct*node));
            newnode -> prev = NULL;
```

```
newnode -> data = null;
newnode -> next = null;
start = newnode;
}
else
{
ptr = start;
newnode = (struct node*) malloc (sizeof(
                    struct node));
newnode -> data = num;
while (ptr->next != NULL)
{
    ptr = ptr->next;
}
ptr->next = newNode;
newnode -> prev = ptr;
newnode -> next = NULL;
}
printf("\n enter the data:");
scanf("%d", &num);
}
return start;
};

struct node* insert_before (struct node*
                            start)
{
struct node *ptr, *newnode;
int num, val;
printf(" enter data :");
scanf("%d", &num);
printf(" Enter value before which
                data is inserted:");
scanf("%d", &val);
```

```
newnode = (struct node*) malloc(sizeof(
                struct node));
newnode -> data = num;
ptr = start;
while(ptr -> data != val)
{
    ptr = ptr -> next;
}
newnode -> next = ptr;
newnode -> prev = ptr -> prev;
ptr -> prev -> next = newnode;
ptr -> prev = newnode;
return start;
};

struct node* del_specific(struct node* start)
{
int num, value;
printf(" Enter value to be deleted");
scanf("%d", value);
if (start == NULL)
{
    printf(" LL is empty\n");
    return start;}
else
{
    struct node *ptr;
    ptr = start;
    while(ptr -> data != value)
    {
        ptr = ptr -> next;}
    ptr -> prev -> next = ptr -> next;
    ptr -> next -> prev = ptr -> prev;
    free(ptr);
    Print("value deleted");
}return start;
```

```
void main()
{
    int choice;
    printf("1 create 2 insert node on left in
        3 delete node at specific position");
    while(1)
    {
     printf(" enter choice ");
     scanf("%d", &choice);
     switch(choice)
     {
        case 1: start = create_ll(start);
                break;
        case 2: start = insert_left(start);
                break;
        case 3: start = del_specific(start);
                break;
        default: exit(0);
     }
    }
}
```

1) write a program
   a) to conduct BST
   b) traverse the tree using inorder, preorder, postorder
   c) display the elements in the tree

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

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

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

struct Node* insert(struct Node* root, int
                    data)
{
    if (root == NULL)
    {
        return createNode(data);
    }
    if (data < root->data)
    {
        root->left = insert(root->left, data);
    }
    else if (data > 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 * Node)
{
    if (root) = NULL)
    {
        Post Order (root -> left);
        Post Order (root -> right);
        Printf(" %d", root -> data);
    }

void display ( struct Node * root)
{
    Printf (" Inorder Traversal");
    inorder (root);
    Printf(" Preorder Traversal;
    Preorder (root);
    Printf (" Postorder Traversal: ");
    postorder (root);
    Print ("\n"));
}
```

```
int main ()
{
    struct Node * root = NULL;
    int data, c;
    Printf (" 1 Into data into BST\n 2 to stop\n");
    while(1)
    {
        Printf (" Enter choice :");
        Scanf (" %d", &c);
        Switch (c)
        {
            case 1:
                Printf (" Enter data :");
                Scanf (" %d", &data);
                root = insert (root, data);
                break;
            case 2:
                display (root);
                exit (root);
                exit (0);
        }
    }
    return 0;
}
```