Report on

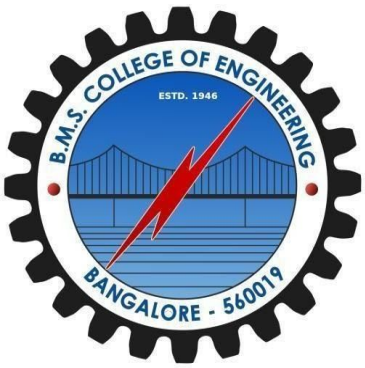
"Data Structures"

Submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering in the course of ***Data Structures (Subject Code)***

Submitted by

Medha Madhusudhan
(1BM19EC074)

Under the Guidance of
Dr. Kayarvizhy N.
Associate Professor
Department of CSE



Department of Computer Science and Engineering
BMS College of Engineering
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019
2020-2021

B M S COLLEGE OF ENGINEERING

P.O. Box No: 1908 Bull Temple Road Bangalore-560019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASSESSMENT

Report on **Data Structures** (**Subject Code),** "Advanced Algorithm assignment" has been successfully completed by **Medha Madhusudhan** at B.M.S College of Engineering in partial fulfillment of the requirements for the 3rd Semester, degree in Bachelor of Engineering in Computer Science and Engineering under Visvesvaraya Technological University, Belgaum during academic year 2020-2021.

Dr. Kayarvizhy N.
Associate Professor
Department of Computer science

Final Marks Awarded

Obtained	Total

CONTENT

LabProgram 1:

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 messages for stack overflow, stack underflow

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

#define MAX 5

int top = -1;
int stack[MAX];

void push(int stack[])
{
    int element;
    if(top == MAX-1)
        printf("Stack is full!");
    else
        {
            top++;
            printf("Enter element: ");
            scanf("%d",&element);
            stack[top] = element;
        }
}

void pop(int stack[])
{
    int del;
    if(top == -1)
        printf("Stack is empty!!!\n");
    else
        {
           del = stack[top];
           top--;
           printf("Element deleted is: %d\n",del);
        }
}

void display(int stack[])
{
```

```
    if(top == -1)
        printf("stack is empty!\n");
    else
        for(int i=top;i>=0;i--)
            printf("%d\t",stack[i]);
        printf("\n");
    }

int main(int argc,char** argv)
{
    int choice;
    while(choice != 4)
        {
            printf("Enter
choice:\n1.Insert\n2.Delete\n3.Display\n4.exit\n");
            scanf("%d",&choice);
            switch(choice)
            {
                case 1:push(stack);
                       break;
                case 2:pop(stack);
                       break;
                case 3:display(stack);
                       break;
                case 4:exit(0);
                default:exit(0);
            }
        }
    return 0;
    }
```

```
void push(int stack[])
{
    int element;
    if (top == MAX-1)
        printf ("stack is full");
    else
      {
        top++;
        printf ("Enter element");
        scanf ("%d",&element);
        stack[top] = element;
      }
}

void pop(int stack[])
{
    int del;
    if (top == -1)
        printf ("stack is empty!!! \n");
    else
      {
        del = stack[top];
```

```
    top--;
    Printf ("Element deleted is: %d \n", del);
  }
}
void display (int stack[])
{
    if (top == -1)
        printf ("Stack is empty! \n");
    else
        for (int i = top; i >= 0; i--)
            printf ("%d \t", stack[i]);
        printf ("\n");
}
```

```
2
Element deleted is: 40
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
Element deleted is: 30
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
3
20      10
Enter choice:
```

LabProgram 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix
expression. The expression consists of single character operands and the binary operators
+ (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 50

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

void push(char c)
{
     if(top == MAX-1)
          printf("Stack overflow!!!");
     else
     {
          top++;
          stack[top] = c;
     }
}

char pop()
{
     char c;
     if(top == -1)
          return -1;
     else
     {
          c = stack[top];
          top--;
          return c;
     }
}

int isOperator(char c)
{
     if(c == '^' || c == '*' || c == '/' || c == '+' || c =='-')
          return 1;
     else
```

```
          return 0;
}

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

void convert(char infix[],char postfix[])
{
     int i=0,j=0;
     char item,x;

     push('(');
     strcat(infix,")");

     item = infix[i];

     while(item != '\0')
     {
          if(item == '(')
               push(item);
          else if(isalpha(item) || isdigit(item))
          {
               postfix[j] = item;
               j++;
          }
          else if(isOperator(item) == 1)
          {
               x = pop();
               while(isOperator(x) == 1 && precedence(x) >=
precedence(item))
               {
                    postfix[j] = x;
                    j++;
                    x = pop();
               }
               push(x);
               push(item);
          }
```

```
          else if(item == ')')
          {
               x = pop();
               while(x != '(')
               {
                    postfix[j] = x;
                    j++;
                    x = pop();
               }
          }
          else
          {
               printf("Invalid expression!\n");
               exit(0);
          }
          i++;
          item = infix[i];
     }

     if(top > 0)
     {
          printf("Invalid expression!!!\n");
          exit(0);
     }
     postfix[j] = '\0';
}

int main(int argc,char **argv)
{
     char infix[MAX],postfix[MAX];

     printf("Enter the infix expression: ");
     gets(infix);
     convert(infix,postfix);
     printf("\nYour postfix expression is:");
     puts(postfix);
     return 0;
}
```

INFIX TO POSTFIX

```
void push (char c)
{
    if (top == MAX-1)
        printf ("stack overflow");
    else
    {
      top++;
      stack[top] = c;
    }
}
char pop()
{
    char c;
    if (top == -1)
        return -1;
    else
    {
        c = stack[top];
        top--;
        return c;
    }
}
int isoperator (char c)
{
    if (c == '^' || c == '*' || c == '/' || c == '+' || c == '-')
      return 1;
    else
      return 0;
}
```

```
int precedence (char c)
{
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else return 0;
}

void convert ( char infix[], char postfix[])
{
    int i=0, j=0;
    char item, x;
    push ('(');
    strcat ( infix, ")");
    item = infix[j];
    while (item != '\0')
    {
        if( item == '(')
            push (item);
        else if ( isdigit(item) || isalpha (item))
        {
            postfix [j] = item;
            j++;
        }
        else if ( isOperator (item) == 1)
        {
            x = pop();
            while ( isoperator (x) == 1 && precedence(x) >= precedence(item))
            {
                postfix[j] = x;
                j++;
                x = pop();
            }
            push (x);
            push (item);
        }
        else if (item == ')')
        { x= pop();
          while (item != '(')
          {
              postfix[j] = x;
              j++;
              x = pop();
          }
        }
        else
```

```
{
    printf("Invalid Expression!\n");
    exit(0);
}
i++;
item = infix[i];
}
if (sop > 0)
{
    printf("Invalid expression!!!\n");
    exit(0);
}
postfix[j] = '\0';
}
```

```
Enter the infix expression: a+b*c-f

Your postfix expression is:abc*+f-
Press any key to continue . . .
```

```
Enter the infix expression: a+b-c^d

Your postfix expression is:ab+cd^-
Press any key to continue . . .
```

Lab Program 3:

WAP to simulate the working of a queue of integers using an array. Provide the following
operations
a) Insert b) Delete c) Display
The program should print appropriate messages for queue empty and queue overflow Conditions

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

#define MAX 3

int front = -1;
int rear = -1;
int Q[MAX];

void enQ()
{
     int element;
    if(rear == MAX-1)
          printf("Q is full!\n");

     else
     {
          if(front == -1 && rear == -1)
               front = 0;
          rear++;
          printf("Enter element: ");
          scanf("%d",&element);
          Q[rear] = element;
     }
}

void deQ()
{
     if(rear == -1)
          printf("Q is empty!\n");
     else if(front == rear)
     {
          printf("Element removed is: %d\n",Q[front]);
          front = rear = -1;
     }
     else
```

```
     {
          printf("Element removed is: %d\n",Q[front]);
          front++;
     }
}

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

int main(int argc,char** argv)
{
    int choice;
    while(choice != 4)
        {
            printf("-----Q-----\nEnter
choice:\n1.Insert\n2.Delete\n3.Display\n4.exit\n");
            scanf("%d",&choice);
            switch(choice)
            {
                case 1:enQ();
                       break;
                case 2:deQ();
                       break;
                case 3:display();
                       break;
                case 4:exit(0);
                default:exit(0);
            }
        }
    return 0;
}
```

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

#define SIZE 3

int front = -1;
int rear = -1;

int Q[SIZE];
void EnQ(int);
int DeQ();
void display();

int main(int argc, char *argv)
{
   int choice, item;
   do
   {
      printf("\nEnter choice 1.EnQ 2.DeQ 3.display 4.Exit:");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1: printf("Enter the element to be added to the Q:");
                 scanf("%d", &item);
                 EnQ(item);
                 break;
         case 2: item = DeQ();
                 if(item == -1)
                    printf("Q is empty");
                 else
                    printf("Item removed from Q: %d", item);
                 break;
         case 3: display();
                 break;
         case 4: exit(0);
```

```
    }
  } while (choice != 4);
  return 0;
}
void EnQ (int el)
{
  if (rear == (SIZE-1))
      printf ("Q is full");
  else
  {
    if (rear == -1)
        front == 0;
    rear += 1;
    Q[rear] = el;
  }
}
int DeQ ()
{
  int item;
  if (front == -1)
      return -1;
  else
  {
    item = Q[front];
    front += 1;
    if (front > rear)
    {
        front = -1;
        rear = -1;
    }
```

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

```
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
1
Enter element: 10
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
1
Enter element: 20
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
1
Enter element: 30
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
1
Q is full!
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
3
10      20      30
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
```

```
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
Element removed is: 10
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
Element removed is: 20
-----Q-----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
```

LabProgram 4:

WAP to simulate the working of a circular queue of integers using an array. Provide the
following operations.
a) Insert b) Delete c) Display
The program should print appropriate messages for queue empty and queue overflow Conditions

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

#define MAX 3

int front = -1;
int rear = -1;
int Q[MAX];

void enQ()
{
     int element;
    if((front ==0 && rear == MAX-1) || (front == (rear+1)%MAX))
          printf("Q is full!\n");

     else
     {
          if(front == -1 && rear == -1)
               front = 0;
          rear = (rear+1)%MAX;
          printf("Enter element: ");
          scanf("%d",&element);
          Q[rear] = element;
     }
}

void deQ()
{
     if(rear == -1)
          printf("Q is empty!\n");
     else if(front == rear)
     {
          printf("Element removed is: %d\n",Q[front]);
          front = rear = -1;
     }
     else
```

```
     {
          printf("Element removed is: %d\n",Q[front]);
          front = (front+1)%MAX;
     }
}

void display()
{
    if(rear == -1)
          printf("Q is empty!\n");
     else if(front <= rear)
     {
          for(int i=front;i<=rear;i++)
               printf("%d\t",Q[i]);
          printf("\n");
     }
     else
     {
          for(int i=front;i< MAX;i++)
               printf("%d\t",Q[i]);
          for(int i=0;i<= rear;i++)
               printf("%d\t",Q[i]);
          printf("\n");
     }
}

int main(int argc,char** argv)
{
    int choice;
    while(choice != 4)
        {
            printf("----Circular Q----\nEnter
choice:\n1.Insert\n2.Delete\n3.Display\n4.exit\n");
            scanf("%d",&choice);
            switch(choice)
            {
                case 1:enQ();
                       break;
                case 2:deQ();
                       break;
                case 3:display();
                       break;
                case 4:exit(0);
                default:exit(0);
            }
        }
    return 0;
```

```
void enQ()
{
    int element;
    if ((front == 0 && rear == MAX-1) || (front == (rear+1)%MAX))
        printf ("Q is full!\n");
    else
    {
      if (front == -1 && rear == -1)
          front = 0;
      rear = (rear+1) % MAX;
      printf ("Enter element: ");
      scanf ("%d", &element);
      Q[rear] = element;
    }
}
void deQ()
{
    if (rear == -1)
        printf ("Q is empty!");
    else if (front == rear)
```

```
    }
    printf("Element removed is %d\n", Q[front]);
    front = rear = -1;
  }
  else
  {
    printf("Element removed is %d\n", Q[front]);
    front = (front+1)%MAX;
  }
}
void display()
{
  if(rear == -1)
    printf("Q is empty");
  else if(front <= rear)
  {
      for(int i=front; i<=rear; i++)
        printf("%d\t", Q[i]);
      printf("\n");
  }
  else
  {
    for(int i=front; i<MAX; i++)
      printf("%d\t", Q[i]);
    for(int i=0; i<=rear; i++)
      printf("%d\t", Q[i]);
    printf("\n");
  }
}
```

```
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
Element removed is: 30
----Circular Q----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
2
Q is empty!
----Circular Q----
Enter choice:
1.Insert
2.Delete
3.Display
4.exit
```

LabProgram 5:

WAP to Implement Singly Linked List with following operations
a) a) Create a linked list. b) Insertion of a node at first position, at any position and at end of
list. c) Display the contents of the linked list.

```
//Linked List 5
#include <stdio.h>
#include <stdlib.h>

struct node{
     int data;
     struct node *next;
};
void insertfirst(struct node **headptr){
     struct node *newnode;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
    newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
     if(*headptr == NULL)
          (*headptr) = newnode;
     else{
          newnode->next = (*headptr);
          (*headptr) = newnode;
     }
}
void insertpos(struct node **headptr){
     struct node *newnode,*temp;
     int count=0,currpos=1,value,pos;
     printf("Enter value: ");
     scanf("%d",&value);
     newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
     if((*headptr) == NULL){
        (*headptr) = newnode;
    }
    else{
          temp = (*headptr);
          while(temp != NULL){
               count++;
               temp = temp->next;
          }
```

```
          printf("There are %d elements in the list.Enter the
position where you want to insert the value: ",count);
          scanf("%d",&pos);
          if(pos > (count+1)){
               printf("no such position\n");
               return;
          }
          if(pos == count+1){
               temp = (*headptr);
               while(temp->next != NULL)
                    temp = temp->next;
               temp->next = newnode;
          }
          else{
               temp = (*headptr);
               while(temp->next != NULL){
                    if(currpos == (pos-1)){
                         newnode->next = temp->next;
                         temp->next = newnode;
                         break;
                    }
                    currpos++;
                    temp = temp->next;
               }
          }
     }
}
void insertlast(struct node **headptr){
     struct node *newnode,*temp;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
    newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
     temp = (*headptr);
     if(*headptr == NULL)
          (*headptr) = newnode;
     else{
          while(temp->next != NULL)
               temp = temp->next;
          temp->next = newnode;
     }
}
void deletelast(struct node **headptr){
     struct node *temp;
     temp = (*headptr);
```

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

int main(int argc,char **argv){
     int choice;
     struct node *head = NULL;
     while(choice != 6){
          printf("Enter choice 1)insertfirst 2)insertpos
3)insertlast 4)deletelast 5)display 6)exit : ");
          scanf("%d",&choice);
          switch(choice){
               case 1:insertfirst(&head);break;
               case 2:insertpos(&head);break;
               case 3:insertlast(&head);break;
               case 4:deletelast(&head);break;
               case 5:display(head);break;
               case 6:
               default:exit(0);
          }
     }
     return 0;
}
```

Hello

papergrid

Date: / /

Lab Program 5

```
struct node {
    int data;
    struct node *next;
};
void insertint (struct node **headptr) {
    struct node *newnode;
    int value;
    printf ("Enter value: ");
    scanf ("%d", &value);
    newnode = (struct node*) malloc (sizeof(struct node));
    if (value);
    printf ("E");
    newnode -> data = value;
    newnode -> next = NULL;
    if (*headptr == NULL)
        (*headptr) = newnode;
    else {
        newnode -> next = (*headptr);
        (*headptr) = newnode;
    }
}
void insertpos (struct node **headptr) {
    struct node *newnode, *temp;
    int count = 0, curpos = 1, value, pos;
    printf ("Enter value: ");
    scanf ("%d", &value);
    newnode = (struct node*) malloc (sizeof (struct node));
    newnode -> data = value;
    newnode -> next = NULL;
    if (*headptr == NULL) {
        (*headptr) = newnode;
    }
```

```
else {
    temp = (*headptr);
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    printf("There are %d elements in the list. Enter position:", count);
    scanf("%d", &pos);
    if (pos > (count+1)) {
        printf("No such position\n");
        return;
    }
    if (pos == count+1) {
        temp = (*headptr);
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newnode;
    }
    else {
        temp = (*headptr);
        while (temp->next != NULL) {
            if (curpos == pos-1) {
                newnode->next = temp->next;
                temp->next = newnode;
                break;
            }
            curpos++;
            temp = temp->next;
        }
    }
}
}
```

```
void insertlast (struct node **headptr) {
  struct node *newnode, *temp;
  int value;
  printf ("\n Enter value");
  scanf ("%d", &value);
  newnode = (struct node *) malloc (sizeof (struct node));
  newnode -> data = value;
  newnode -> next = NULL;
  temp = (*headptr);
  if (*headptr == NULL)
      (*headptr) = newnode;
  else {
      while (temp -> next != NULL)
          temp = temp -> next;
      temp -> next = newnode;
  }
}
void deletelast (struct node **headptr) {
      struct node *temp;
      temp = (*headptr);
      if (*headptr == NULL)
        printf ("The list is empty\n");
      else if ((*headptr) -> next == NULL)
          (*headptr) = NULL;
      else {
          temp = *headptr;
          while ((temp -> next) -> next != NULL)
              temp = temp -> next;
          temp -> next = NULL;
      }
}
```

```
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 1
Enter value: 10
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 2
Enter value: 24
There are 1 elements in the list.Enter the position where you want to insert the value: 2
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 3
Enter value: 45
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 5
10      24      45
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 4
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 5
10      24
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit :
```

```
Enter value: 2
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 1
Enter value: 3
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 3
Enter value: 5
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 5
3       2       5
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 2
Enter value: 32
There are 3 elements in the list.Enter the position where you want to insert the value: 2
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit : 5
3       32      2       5
Enter choice 1)insertfirst 2)insertpos 3)insertlast 4)deletelast 5)display 6)exit :
```

LabProgram 6:

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.

```
//Linked List 6
#include <stdio.h>
#include <stdlib.h>

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

void insertlast(struct node **headptr){
     struct node *newnode,*temp;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
    newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
     temp = (*headptr);
     if(*headptr == NULL)
          (*headptr) = newnode;
     else{
          while(temp->next != NULL)
               temp = temp->next;
          temp->next = newnode;
     }
}
void deletefirst(struct node **headptr){
     if((*headptr) == NULL)
          printf("The list is empty\n");
     else if((*headptr)->next == NULL)
          (*headptr) = NULL;
     else{
          (*headptr) = (*headptr)->next;
     }
}
void deletepos(struct node **headptr){
     struct node *temp;
     int count=0,currpos=1,pos;
     if((*headptr) == NULL){
```

```
          printf("The list is empty\n");
          return;
     }
     else{
          if((*headptr)->next == NULL){
               (*headptr) = NULL;
               printf("The only element in the list was
deleted\n");
               return;
          }
        temp = (*headptr);
        while(temp != NULL){
               count++;
               temp = temp->next;
          }
          printf("There are %d elements in the list.Enter pos of
element to be deleted: ",count);
          scanf("%d",&pos);
        if(pos > (count+1)){
               printf("No such position is present\n");
               return;
          }
          if(pos == (count+1)){
               temp = (*headptr);
               while((temp->next)->next != NULL)
                    temp = temp->next;
               temp->next = NULL;
          }
          else{
               temp = (*headptr);
               while(temp->next != NULL){
                if(pos == 1){
                    (*headptr) = (*headptr)->next;
                    return;
                }
                    if(currpos == pos-1){
                         temp->next = (temp->next)->next;
                         return;
                    }
                    currpos++;
                    temp = temp->next;
               }
               printf("No such element was found!!!\n");
          }
     }
}
void deletelast(struct node **headptr){
```

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

int main(int argc,char **argv){
     int choice;
     struct node *head = NULL;
     while(choice != 6){
          printf("Enter choice 1)insertlast 2)deletefirst
3)deletepos 4)deletelast 5)display 6)exit : ");
          scanf("%d",&choice);
          switch(choice){
               case 1:insertlast(&head);break;
               case 2:deletefirst(&head);break;
               case 3:deletepos(&head);break;
               case 4:deletelast(&head);break;
               case 5:display(head);break;
               case 6:
               default:exit(0);
          }
     }
     return 0;
}
```

Lab program 6

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
void insertlast (struct node **headptr) {
    struct node *newnode, *temp;
    int value;
    printf ("Enter value:");
    scanf ("%d", &value");
    newnode = (struct node *) malloc (sizeof (struct node));
    newnode -> data = value;
    newnode -> next = NULL;
    temp = (*headptr);
    if (*headptr == NULL)
        (*headptr) = newnode;
    else {
        while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = newnode;
    }
}
void deletefirst (struct node **headptr) {
    if (*headptr == NULL)
        printf ("The list is empty \n");
    else if ((*headptr) -> next == NULL)
        (*headptr) = NULL;
    else {
        (*headptr) = (*headptr) -> next;
    }
}
```

```
void deletepos (struct node **headptr){
  struct node *temp;
  int count = 0,curr_pos = 1,pos;
  if ((*headptr) == NULL){
        printf("List is empty\n");
        return;
  }
  else {
      if (*headptr)->next == NULL){
          (*headptr) = NULL;
          return;
      }
      temp = (*headptr);
      while (temp != NULL){
            count++;
            temp = temp->next;
      }
      printf("There are %d elements. Enter position:", count);
      scanf("%d", &pos);
      if (pos > count+1){
          printf("No such position");
          return;
      }
      if (pos == count+1){
          temp = (*headptr);
          while ((temp->next)->next != NULL)
              temp = temp->next;
          temp->next = NULL;
      }
      else {
          temp = (*headptr);
          while (temp->next != NULL){
              if (pos == 1){
                  (*headptr) = (*headptr)->next;
              return;
```

papergrid
Date: / /

```
        if (curpos == pos-1) {
            temp->next = (temp->next)->next;
            return;
        }
        curpos++;
        temp = temp->next;
    }
    printf("No such element found\n");
  }
 }
}
void deletelast(struct node **headptr) {
    struct node *temp;
    temp = (*headptr);
    if ((*headptr) == NULL)
        printf("List is empty\n");
    else if ((*headptr)->next == NULL) {
        (*headptr) = NULL;
    else {
        temp = (*headptr);
        while (temp->next)->next != NULL)
            temp = temp->next;
        temp->next = NULL;
    }
  }
}
```

```
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 10
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 20
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 30
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 40
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 5
10      20      30      40
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 2
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 5
20      30      40
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 4
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 5
20      30
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 3
There are 2 elements in the list.Enter pos of element to be deleted: 1
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 5
30
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit :
```

```
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 1
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 2
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 3
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 1
Enter value: 4
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 5
1       2       3       4
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 2
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit : 3
There are 3 elements in the list.Enter pos of element to be deleted: 6
No such position is present
Enter choice 1)insertlast 2)deletefirst 3)deletepos 4)deletelast 5)display 6)exit :
```

LabProgram 7:

WAP Implement Single Link List with following operations
a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

```
//Lab Program 7
//sort,reverse,concatenate linked lists(s)

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

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

void insertend(struct node **headptr){
     struct node *newnode,*temp;
     newnode = (struct node*)malloc(sizeof(struct node));
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
     newnode->data = value;
     newnode->next = NULL;
     if((*headptr) == NULL)
          (*headptr) = newnode;
     else{
          temp = (*headptr);
          while(temp->next != NULL)
               temp = temp->next;
          temp->next = newnode;
     }
}
void deleteend(struct node **headptr){
     struct node *temp;
     if((*headptr) == NULL)
          printf("The list is empty\n");
     else{
          temp = (*headptr);
          while((temp->next)->next != NULL)
               temp = temp->next;
          temp->next = NULL;
     }
}
void display(struct node *temp){
     if(temp == NULL)
```

```
          printf("The list is empty\n");
     else{
          while(temp != NULL){
               printf("%d\t",temp->data);
               temp = temp->next;
          }
          printf("\n");
     }
}
void sort(struct node *temp){
     struct node *p,*q;
     int a;
     for(p = temp;p != NULL;p = p->next){
          for(q=p->next;q != NULL;q = q->next){
               if(p->data > q->data){
                    a = p->data;
                    p->data = q->data;
                    q->data = a;
               }
          }
     }
     printf("The sorted list is as follows:\n");
     while(temp != NULL){
          printf("%d\t",temp->data);
          temp = temp->next;
     }
     printf("\n");
}
void reverse(struct node *temp){
     struct node *first=NULL,*second,*third;
     second = temp;
     while(second != NULL){
          third = second->next;
          second->next = first;
          first = second;
          second = third;
     }
     temp = first;
     printf("The list after reversal is as follows:\n");
     while(temp != NULL){
          printf("%d\t",temp->data);
          temp = temp->next;
     }
     printf("\n");
}
void concatenate(struct node *temp1,struct node *temp2){
     if(temp1 == NULL && temp2 == NULL){
```

```
        printf("Lists are empty\n");
    }
    else if(temp1 != NULL && temp2 == NULL){
        while(temp1 != NULL){
            printf("%d\t",temp1->data);
            temp1 = temp1->next;
        }
        printf("\n");
    }
    else if(temp1 == NULL && temp2 != NULL){
        while(temp2 != NULL){
            printf("%d\t",temp2->data);
            temp2 = temp2->next;
        }
        printf("\n");
    }
    else{
        struct node *temp;
        temp = temp1;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = temp2;
        temp = temp1;
        printf("After concatenation:\n");
        while(temp != NULL){
            printf("%d\t",temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main(int argc,char **argv){
     int choice;
     struct node *head1 = NULL,*head2 = NULL;
     while(choice != 12){
          printf("Enter choice : 1)insert1 2)insert2 3)delete1
4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1
10)reverse2 11)concatenate 12)exit: ");
          scanf("%d",&choice);
          switch(choice){
               case 1:insertend(&head1);break;
               case 2:insertend(&head2);break;
               case 3:deleteend(&head1);break;
               case 4:deleteend(&head2);break;
               case 5:display(head1);break;
               case 6:display(head2);break;
```

```
                case 7:sort(head1);break;
                case 8:sort(head2);break;
                case 9:reverse(head1);break;
                case 10:reverse(head2);break;
                case 11:concatenate(head1,head2);break;
                case 12:
                default:exit(0);
            }
        }
}
```

Header

papergrid

Date: / /

Lab Program 7:

```
void display (struct node *temp) {
    if (temp == NULL)
        printf ("List is empty\n");
    else {
        while (temp != NULL) {
            printf ("%d\t", temp->data);
            temp = temp->next;
        }
        printf ("\n");
    }
}
void sort (struct node *temp) {
    struct node *p, *q;
    int a;
    for (p = temp; p != NULL; p = p->next) {
        for (q = p->next; q != NULL; q = q->next) {
            if (p->data > q->data) {
                a = p->data;
                p->data = q->data;
                q->data = a;
            }
        }
    }
    printf ("The sorted list is as follows:");
    while (temp != NULL) {
        printf ("%d\t", temp->data);
        temp = temp->next;
    }
    printf ("\n");
}
```

papergrid
Date: / /

```
void reverse ( struct node *temp) {
    struct node *first = NULL, *second, * third;
    second = temp;
    while(second != NULL) {
        third = second -> next;
        second -> next = first;
        first = second;
        second = third;
    }
    temp = first;
    printf(" After Reversal: \n");
    while (temp != NULL) {
        printf (" %d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void concatenate ( struct node *temp1, struct node *temp2){
    struct node *temp;
    temp = temp1;
    while (temp->next != NULL)
        temp = temp -> next;
    temp -> next = temp2;
    temp = temp1;
    printf (" After concatenation:");
    while ( temp != NULL){
        printf ( "%d ", temp -> data );
        temp = temp -> next;
    }
    printf ("\n");
}
}
```

```
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 1
Enter value: 10
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 1
Enter value: 20
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 1
Enter value: 12
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 1
Enter value: 45
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 1
Enter value: 52
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 2
Enter value: 1
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 2
Enter value: 3
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 2
Enter value: 2
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 2
Enter value: 4
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 5
10      20      12      45      52
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 6
1       3       2       4
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 7
The sorted list is as follows:
10      12      20      45      52
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 8
The sorted list is as follows:
1       2       3       4
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 9
The list after reversal is as follows:
52      45      20      12      10
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 10
The list after reversal is as follows:
4       3       2       1
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)concatenate 12)exit: 11
```

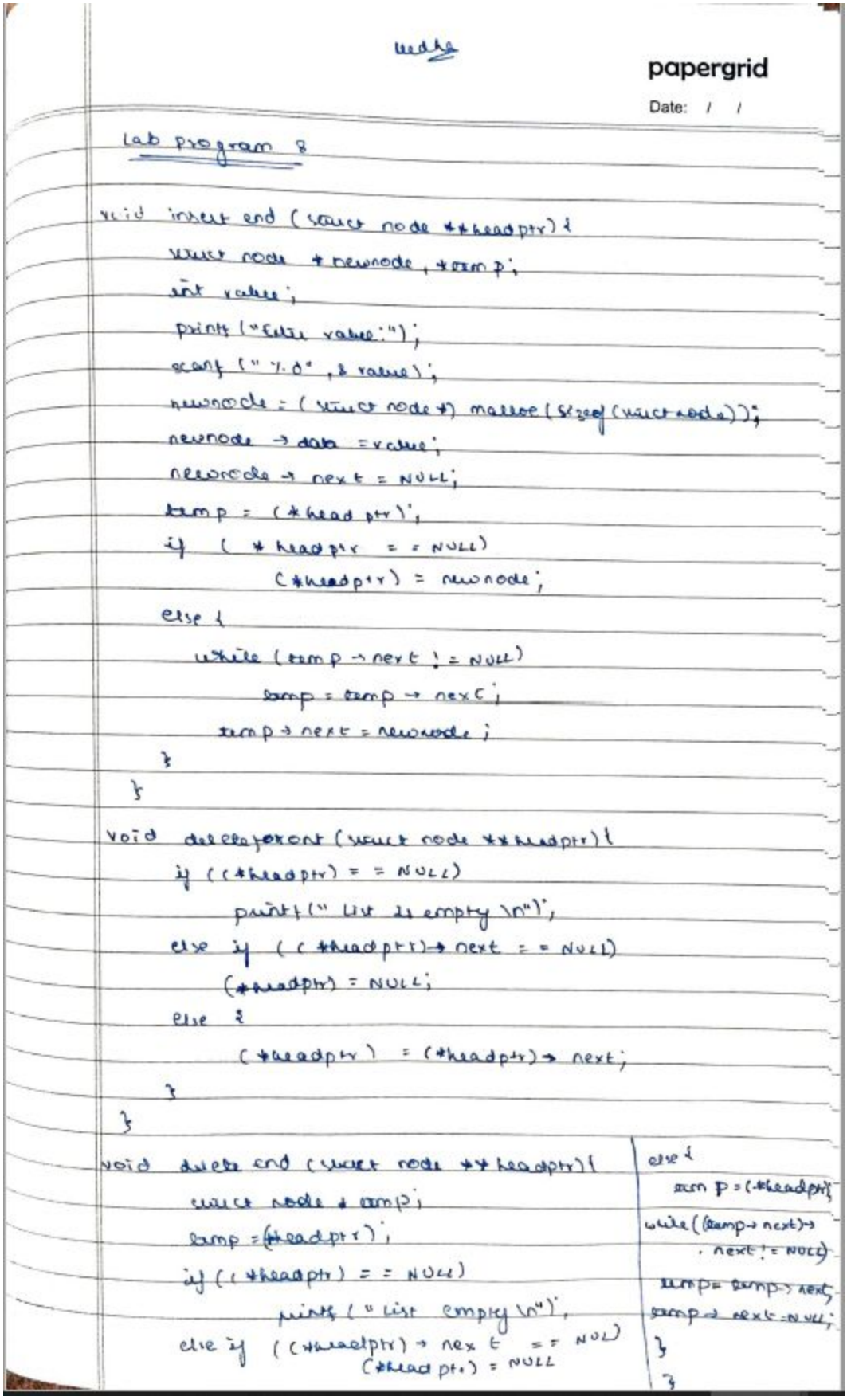
```
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 1
Enter value: 1
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 1
Enter value: 2
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 1
Enter value: 3
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 2
Enter value: 1
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 2
Enter value: 2
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit: 11
After concatenation:
1       2       3       1       2
Enter choice : 1)insert1 2)insert2 3)delete1 4)delete2 5)display1 6)display2 7)sort1 8)sort2 9)reverse1 10)reverse2 11)c
oncatenate 12)exit:
```

LabProgram 8:

WAP to implement Stack & Queues using Linked Representation

```
//Lab Program 8
//stack and Queue implementation
#include <stdio.h>
#include <stdlib.h>
struct node{
     int data;
     struct node *next;
};

void insertend(struct node **headptr){
     struct node *newnode,*temp;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
    newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
     temp = (*headptr);
     if(*headptr == NULL)
          (*headptr) = newnode;
     else{
          while(temp->next != NULL)
               temp = temp->next;
          temp->next = newnode;
     }
}
void deletefront(struct node **headptr){
     if((*headptr) == NULL)
          printf("The list is empty\n");
     else if((*headptr)->next == NULL)
          (*headptr) = NULL;
     else{
          (*headptr) = (*headptr)->next;
     }
}
void deleteend(struct node **headptr){
     struct node *temp;
     temp = (*headptr);
     if((*headptr) == NULL)
          printf("The list is empty\n");
     else if((*headptr)->next == NULL)
          (*headptr) = NULL;
     else{
          temp = *headptr;
```

```
          while((temp->next)->next != NULL)
               temp = temp->next;
          temp->next = NULL;
     }
}
void display(struct node *temp){
     if(temp == NULL){
          printf("The list is empty\n");
        return;
     }
     else{
          while(temp != NULL){
               printf("%d\t",temp->data);
               temp = temp->next;
          }
          printf("\n");
     }
}
int main(int argc,char **argv){
     struct node *head1=NULL,*head2=NULL;
     int choice;
     while(choice != 7){
          printf("Enter choice: 1)pushstack 2)popstack
3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : ");
          scanf("%d",&choice);
          switch(choice){
               case 1:insertend(&head1);break;
               case 2:deleteend(&head1);break;
               case 3:display(head1);break;
               case 4:insertend(&head2);break;
               case 5:deletefront(&head2);break;
               case 6:display(head2);break;
               case 7:
               default:exit(0);
          }
     }
}
```



Linked list

papergrid

Date: / /

Lab program 8

```
void insert end (struct node **headptr) {
    struct node *newnode, *temp;
    int value;
    printf ("Enter value:");
    scanf ("%d", &value);
    newnode = (struct node*) malloc (sizeof (struct node));
    newnode -> data = value;
    newnode -> next = NULL;
    temp = (*head ptr);
    if ( *headptr == NULL)
            (*headptr) = newnode;
    else {
      while (temp -> next != NULL)
            temp = temp -> next;
        temp -> next = newnode;
    }
}

void deleteparent (struct node **headptr) {
    if ((*headptr) == NULL)
        printf (" List is empty \n");
    else if ((*headptr) -> next == NULL)
        (*headptr) = NULL;
    else {
        (*headptr) = (*headptr) -> next;
    }
}

void delete end (struct node **headptr) {
    struct node * temp;
    temp = (*headptr);
    if ((*headptr) == NULL)
        printf ("List empty \n");
    else if ((*headptr) -> next == NULL)
            (*headptr) = NULL
```

```
else {
  temp p = (*headptr);
while ((temp -> next) ->
    . next != NULL)
  temp = temp -> next;
temp -> next = NULL;
}
}
```

```
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 1
Enter value: 1
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 1
Enter value: 2
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 1
Enter value: 3
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 1
Enter value: 4
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 3
1       2       3       4
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 2
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 3
1       2       3
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit :
```

```
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 4
Enter value: 10
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 4
Enter value: 20
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 4
Enter value: 30
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 4
Enter value: 40
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 6
10      20      30      40
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 5
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit : 6
20      30      40
Enter choice: 1)pushstack 2)popstack 3)displaystack 4)EnQ 5)DeQ 6)displayQ 7)exit :
```

LabProgram 9:

WAP Implement doubly link list with primitive operations

a) Create a doubly linked list. b) Insert a new node to the left of the node.

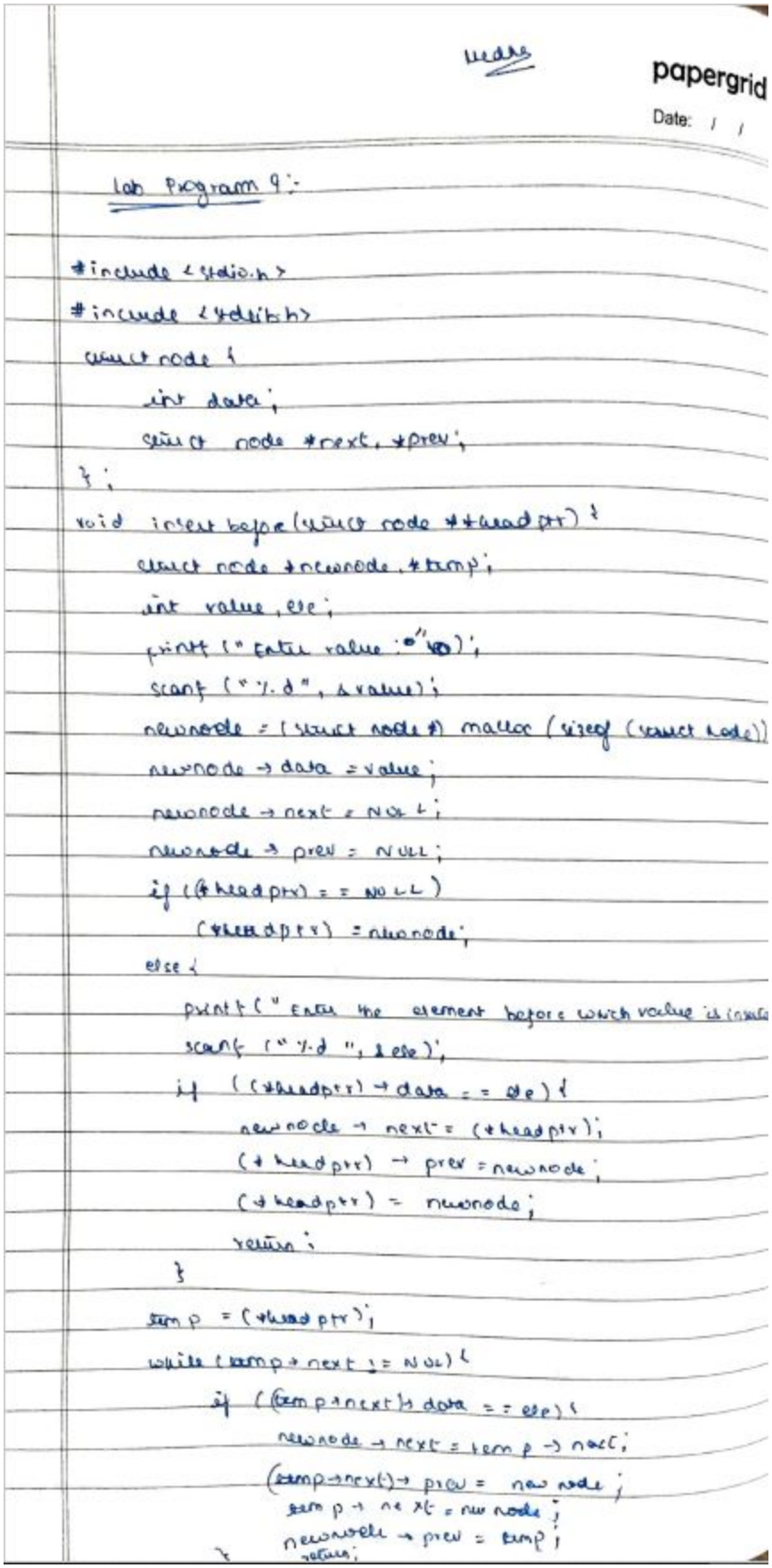
c) Delete the node based on a specific value. c) Display the contents of the list

```
//Lab Program 9
//insert,delete,display
#include <stdio.h>
#include <stdlib.h>

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

void insertend(struct node **headptr){
     struct node *newnode,*temp;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
     newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->prev = NULL;
     newnode->next = NULL;
     if((*headptr) == NULL){
          (*headptr) = newnode;
     }
     else{
          temp = (*headptr);
          while(temp->next != NULL)
               temp = temp->next;
          temp->next = newnode;
          newnode->prev = temp;
     }
}
void insertbefore(struct node **headptr){
     struct node *newnode,*temp;
     int value,ele;
     printf("Enter value: ");
     scanf("%d",&value);
     newnode = (struct node*)malloc(sizeof(struct node));
     newnode->data = value;
     newnode->next = NULL;
```

```
     newnode->prev = NULL;
     if((*headptr) == NULL)
          (*headptr) = newnode;
     else{
          printf("Enter the element before which value is to be
inserted: ");
          scanf("%d",&ele);
        if((*headptr)->data == ele){
            newnode->next = (*headptr);
            (*headptr)->prev = newnode;
            (*headptr) = newnode;
            return;
        }
          temp = (*headptr);
          while(temp->next != NULL){
               if((temp->next)->data == ele){
                    newnode->next = temp->next;
                    (temp->next)->prev = newnode;
                    temp->next = newnode;
                    newnode->prev = temp;
                    return;
               }
               temp = temp->next;
          }
          printf("No such element found!\n");
     }
}
void deleteend(struct node **headptr){
     struct node *temp;
     if((*headptr) == NULL)
          printf("List is empty\n");
     else{
          temp = (*headptr);
          while((temp->next)->next != NULL)
               temp = temp->next;
          temp->next = NULL;
     }
}
void deleteval(struct node **headptr){
     struct node *temp;
     int value;
     if((*headptr) == NULL)
          printf("List is empty\n");
     else{
          printf("Enter the value to be deleted: ");
          scanf("%d",&value);
        if((*headptr)->data == value){
```

```
            (*headptr) = (*headptr)->next;
            (*headptr)->prev = NULL;
            return;
        }
        temp = (*headptr);
        while((temp->next)->next != NULL)
            temp = temp->next;
        if((temp->next)->data == value){
            temp->next = NULL;
            return;
        }
          temp = (*headptr);
          while(temp->next != NULL){
               if((temp->next)->data == value){
                    temp->next = (temp->next)->next;
                    (temp->next)->prev = temp;
                    return;
               }
               temp = temp->next;
          }
          printf("No such element found!\n");
     }
}
void display(struct node *temp){
     if(temp == NULL)
          printf("List is empty\n");
     else{
          while(temp != NULL){
               printf("%d\t",temp->data);
               temp = temp->next;
          }
          printf("\n");
     }
}

int main(int charc,char **argv){
     int choice;
     struct node *head=NULL;
     while(choice != 6){
          printf("Enter choice: 1)insertend 2)insertbefore
3)deleteend 4)deleteval 5)display 6)exit : ");
          scanf("%d",&choice);
          switch(choice){
               case 1:insertend(&head);break;
               case 2:insertbefore(&head);break;
               case 3:deleteend(&head);break;
               case 4:deleteval(&head);break;
```

```
                case 5:display(head);break;
                case 6:
                default:exit(0);
            }
        }
        return 0;
}
```



Nodes

papergrid

Date: / /

Lab Program 9:-

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next, *prev;
};
void insert_before(struct node **headptr) {
    struct node *newnode, *temp;
    int value, else;
    printf("Enter value:");
    scanf("%d", &value);
    newnode = (struct node*) malloc (sizeof (struct node));
    newnode -> data = value;
    newnode -> next = NULL;
    newnode -> prev = NULL;
    if (*headptr == NULL)
        (*headptr) = newnode;
    else {
        printf("Enter the element before which value is inserted");
        scanf("%d", &else);
        if ((*headptr) -> data == else) {
            newnode -> next = (*headptr);
            (*headptr) -> prev = newnode;
            (*headptr) = newnode;
            return;
        }
    temp = (*headptr);
    while (temp -> next != NULL) {
        if ((temp -> next) -> data == else) {
            newnode -> next = temp -> next;
            (temp -> next) -> prev = newnode;
            temp -> next = newnode;
            newnode -> prev = temp;
            return;
```

papergrid

Date: / /

```
    temp = temp->next;
  }
  printf("No such element found\n");
 }
}
void deleteval (struct node **headptr){
    struct node *temp;
    int value;
    if ((*headptr) == NULL)
       printf("List empty\n");
    else {
       printf("Enter value to be deleted :");
       scanf("%d", &value);
       if ((*headptr)->data == value){
          (*headptr) = (*headptr)->next;
          (*headptr)->prev = NULL;
          return;
       }
       temp = (*headptr);
       while ((temp->next)->next != NULL)
          temp = temp->next;
       if (temp->next)->data == value){
          temp->next = NULL;
       return;
       }
       temp = (*headptr);
       while (temp->next != NULL){
          if ((temp->next)->data == value){
             (temp->next) = temp->next->next;
             (temp->next)->prev = temp;
          return;
          }
          temp = temp->next;
       }
       printf("No such element found\n");
    }
```

```
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 10
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 20
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 30
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
10      20      30
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 2
Enter value: 15
Enter the element before which value is to be inserted: 20
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
10      15      20      30
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 3
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
10      15      20
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 4
Enter the value to be deleted: 10
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
15      20
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit :
```

```
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 10
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 3
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
List is empty
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 10
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 20
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 30
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 1
Enter value: 45
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 2
Enter value: 42
Enter the element before which value is to be inserted: 45
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit : 5
10      20      30      42      45
Enter choice: 1)insertend 2)insertbefore 3)deleteend 4)deleteval 5)display 6)exit :
```

LabProgram 10:

Write a program

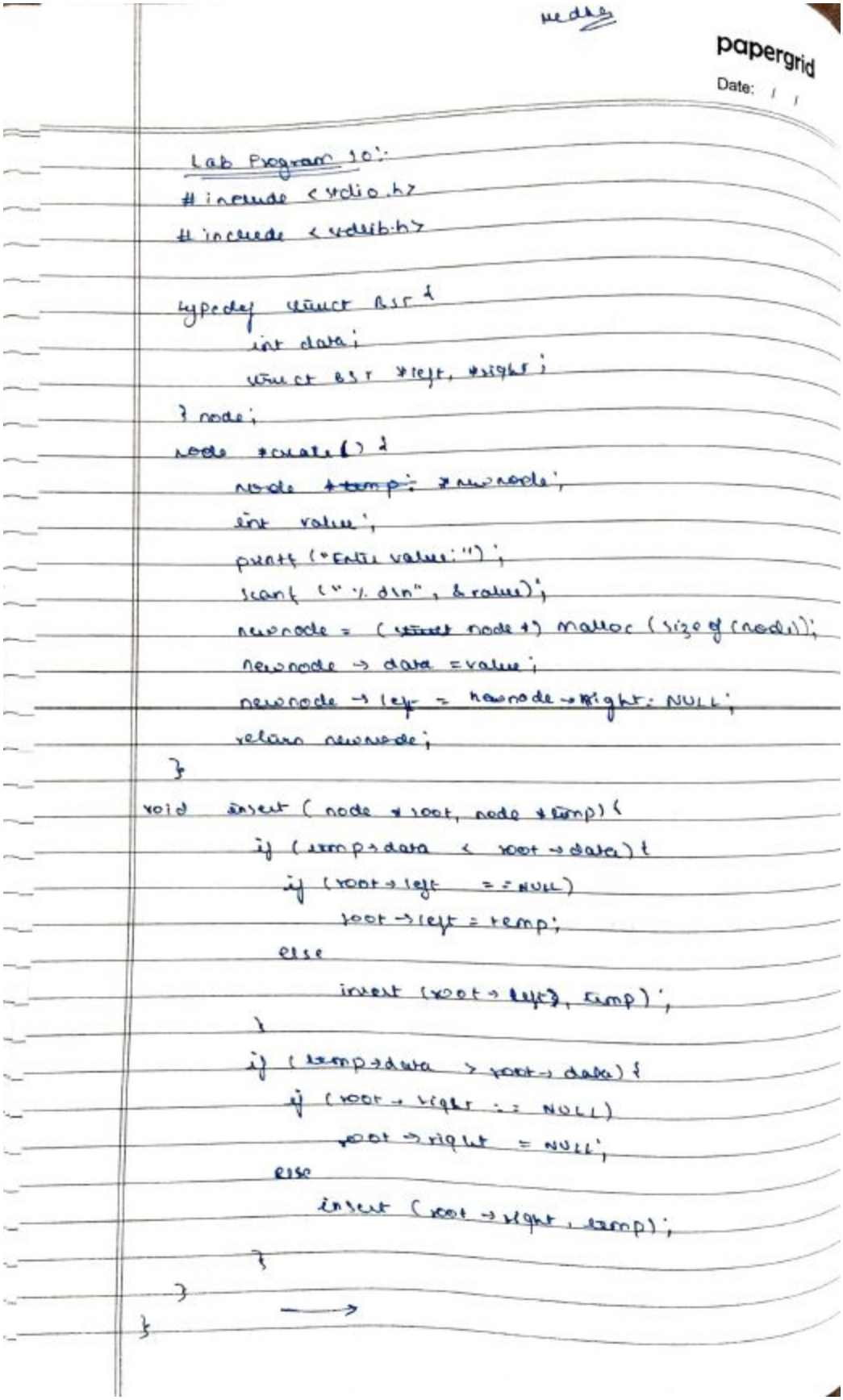
a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

```
//Lab Program 10
//Binary search tree

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

typedef struct BST{
     int data;
     struct BST *left,*right;
}node;

node *create(){
     node *newnode;
     int value;
     printf("Enter value: ");
     scanf("%d",&value);
     newnode = (node*)malloc(sizeof(node));
     newnode->data = value;
     newnode->left = NULL;
     newnode->right = NULL;
     return newnode;
}
void insert(node *root,node *temp){
     if(temp->data < root->data){
          if(root->left == NULL)
               root->left = temp;
          else{
               insert(root->left,temp);
          }
     }
     if(temp->data > root->data){
          if(root->right == NULL)
               root->right = temp;
          else{
               insert(root->right,temp);
          }
     }
}
```

```
void inorder(node *root){
     if(root != NULL){
          inorder(root->left);
          printf("%d\t",root->data);
          inorder(root->right);
     }
}
void preorder(node *root){
     if(root != NULL){
          printf("%d\t",root->data);
          preorder(root->left);
          preorder(root->right);
     }
}
void postorder(node *root){
     if(root != NULL){
          postorder(root->left);
          postorder(root->right);
          printf("%d\t",root->data);
     }
}
//node *minvaluenode(){}
//node *deletenode(){}
int main(int argc,char **argv){
     int choice;
     node *root=NULL,*temp;
     while(choice != 5){
          printf("\nEnter choice 1)insert 2)inorder 3)preorder
4)postorder 5)exit : ");
          scanf("%d",&choice);
          switch(choice){
               case 1:temp = create();
                    if(root == NULL)
                        root = temp;
                    else
                        insert(root,temp);
                    break;
               case 2:inorder(root);break;
               case 3:preorder(root);break;
               case 4:postorder(root);break;
               case 5:
               default:exit(0);
          }
     }
     return 0;
}
```



```
Lab Program 10:
#include <stdio.h>
#include <stdlib.h>

typedef struct BST {
    int data;
    struct BST *left, *right;
} node;
node *create() {
    node *temp, *newnode;
    int value;
    printf("Enter value:");
    scanf("%d", &value);
    newnode = (node *) malloc(sizeof(node));
    newnode -> data = value;
    newnode -> left = newnode -> right = NULL;
    return newnode;
}
void insert(node *root, node *temp) {
    if (temp->data < root->data) {
        if (root->left == NULL)
            root->left = temp;
        else
            insert(root->left, temp);
    }
    if (temp->data > root->data) {
        if (root->right == NULL)
            root->right = NULL;
        else
            insert(root->right, temp);
    }
}
```

Date: / /

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

```
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 10

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 2

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 3

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 7

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 4

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 2
2       3       4       7       10
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 3
10      2       3       7       4
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 4
4       7       3       2       10
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit :
```

```
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 7

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 3

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 5

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 45

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 1
Enter value: 28

Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 2
3       5       7       28      45
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 3
7       3       5       45      28
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit : 4
5       3       28      45      7
Enter choice 1)insert 2)inorder 3)preorder 4)postorder 5)exit :
```