Data Structures – Lab (Record)

Name: *B V Vineeth*
USN: *1BM19CS033*
Department: *Computer Science and Engineering*
Section: *3A*
Academic Year: *2019 - 2023*

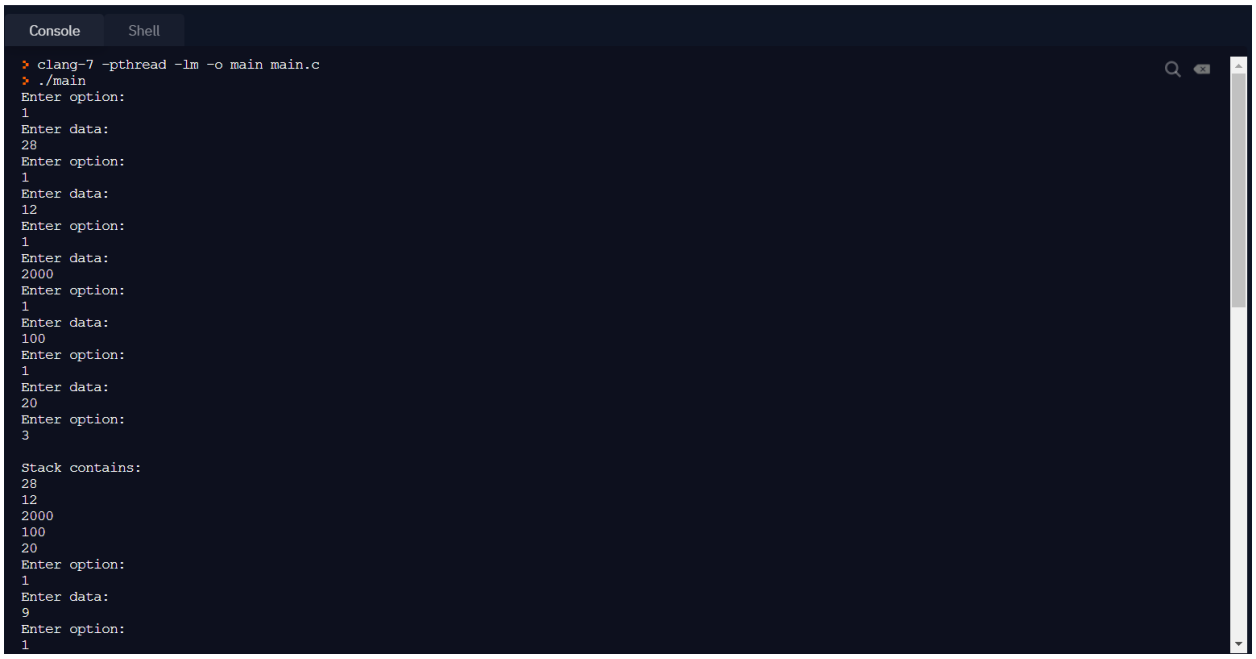
Index:

Stack

```
#include <stdio.h>
#define SIZE 10

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

void push(int value)
{
   if(top<SIZE-1)
   {
      if (top < 0)
      {
         stack[0] = value;
         top = 0;
      }
      else
      {
         stack[top+1] = value;
         top++;
      }
   }
   else
   {
      printf("Stack Overflow\n");
   }
}

int pop()
{
   if(top >= 0)
   {
      int n = stack[top];
      top--;
      return n;
   }

   printf("Stack Underflow.\n");
   return -1;
}

int Top()
{
   return stack[top];
}
```

```
int isempty()
{
   return top<0;
}

void display()
{
   int i;
   for(i=0;i<=top;i++)
   {
      printf("%d\n",stack[i]);
   }
}

int main(void)
{
   int opt, data, poppedData;

   while (opt != 4){
     printf("Enter option:\n");
     scanf("%d", &opt);

     if (opt == 1){
       printf("\nEnter data:\n");
       scanf("%d", &data);
       push(data);
     }
     else if (opt == 2){
       poppedData = pop();
       printf("\nPopped data: %d\n", poppedData);
     }
     else if (opt == 3){
       printf("\nStack contains:\n");
       display();
     }
     else if (opt == 4){
       break;
     }
     else
       printf("Invalid operational value.\n");
   }
}
```

Output


Console
Shell
clang-7 -pthread -lm -o main main.c
./main
Enter option:
1
Enter data:
28
Enter option:
1
Enter data:
12
Enter option:
1
Enter data:
2000
Enter option:
1
Enter data:
100
Enter option:
1
Enter data:
20
Enter option:
3
Stack contains:
28
12
2000
100
20
Enter option:
1
Enter data:
9
Enter option:
1


Console
Shell
3
Stack contains:
28
12
2000
100
20
9
7
2002
18
100
Enter option:
1
Enter data:
12
Stack Overflow
Enter option:
2
Popped data: 100
Enter option:
3
Stack contains:
28
12
2000
100
20
9
7
2002
18
Enter option:

Linear Queue

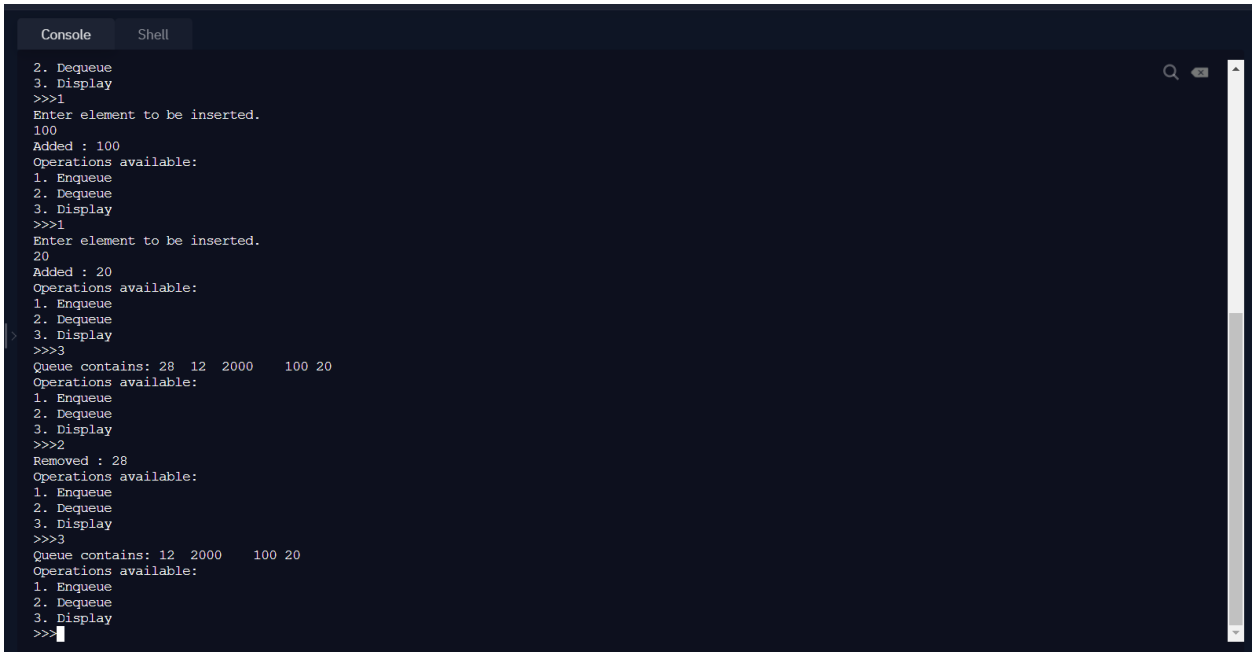
```
#include <stdio.h>

#define SIZE 5


int queue[SIZE], front = -1, rear = -1;


void enqueue();

void dequeue();

void display();


int main(void) {

 int opt = 0;


 while (opt != 4){

  printf("Operations available:\n1. Enqueue\n2. Dequeue\n3. Display\n>>>");

  scanf("%d", &opt);


  switch (opt){

    case 1:

      enqueue();

      break;

    case 2:

      dequeue();

      break;
```

```
      case 3:
        display();
        break;
      case 4:
        break;
      default:
        printf("Invalid arguments.\n");
    }
  }
}

void enqueue() {
  int ele;

  if (rear == front + (SIZE - 1))
    printf("Queue overflow.\n");
  else {

    printf("Enter element to be inserted.\n");
    scanf("%d", &ele);

    if (rear == -1){
      front = 0;
    }
```

```
    rear++;
    queue[rear] = ele;

    printf("Added : %d\n", ele);
  }
}

void dequeue() {
  if ((front > rear) || (rear == -1))
    printf("Queue underflow.\n");
  else {
    printf("Removed : %d\n", queue[front]);
    front++;
  }
}

void display() {
  if (front < 0)
    printf("Empty queue\n");
  else {
    printf("Queue contains: ");

    for (int i = front; i <= rear; i++)
      printf("%d\t", queue[i]);
```

```
    printf("\n");
  }
}
```

Output



Infix to Postfix

```
#include<stdio.h>

#include<ctype.h>


char stack[100];

int top = -1;


void push(char x)

{

   stack[++top] = x;

}


char pop()

{

   if(top == -1)

      return -1;

   else

      return stack[top--];

}


int priority(char x)

{

   if(x == '(')

      return 0;
```

```
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}

int main(void)
{
    char exp[100];
    char *e, x, *postFix;
    int i = 0;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
```

```
            while((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c ",pop());
            push(*e);
        }
        e++;
    }

    while(top != -1)
    {
        postFix[i] = pop();
        i++;
    }

    for (int j = 0; j <= i; j++)
      printf("%c", postFix[j]);
}
```

Output


Console
Shell
clang-7 -pthread -lm -o main main.c
./main
Enter the expression : a+b*(c/d)-e
a b c d / * + e -

Circular Queue

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 3


int front=-1;

int rear=-1;


int queue[MAX];


void Enque(int);

void Deque();

void display();

int main(int argc, char **argv)

{

            int option;

   int item;

   do{

      printf("\nCircular Queue\n");

      printf("\n 1. Insert to Queue (EnQueue)");

      printf("\n 2. delete from the Queue (DeQueue)");

      printf("\n 3. Display the content ");

      printf("\n 4. Exit\n");

      printf("Enter the option :");
```

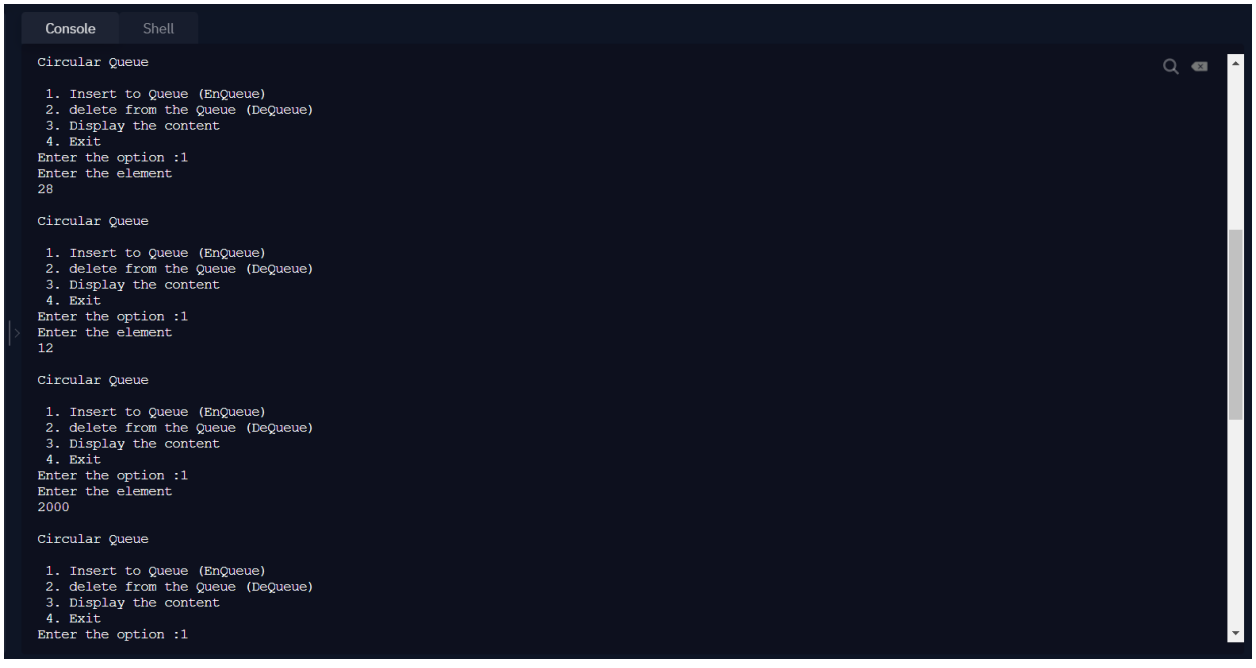
```
        scanf("%d",&option);
        switch(option)
        {
           case 1:  printf("Enter the element\n");
                    scanf("%d",&item);
                    Enque(item);
                    break;
           case 2: Deque();

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

void Enque(int ele)
{
    if(((front == 0 && rear == MAX - 1))|| (front == rear + 1) )
    {
       printf("Queue is full\n");return;

    }
```

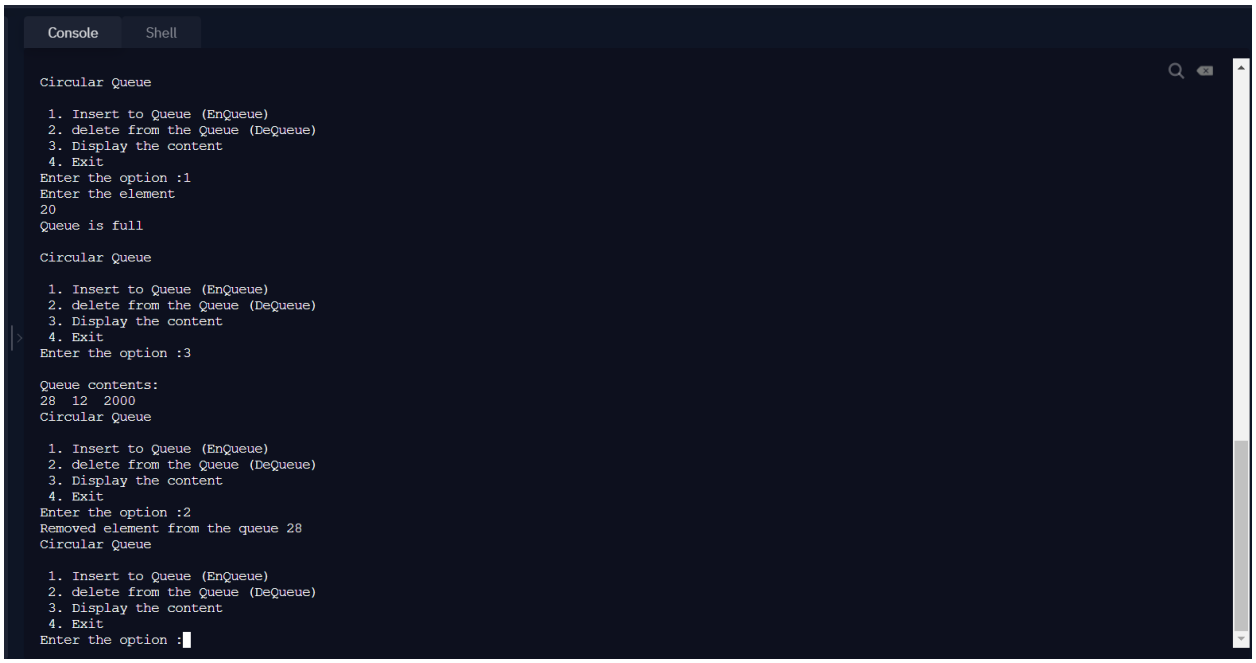
```
    else
    {
      rear=(rear+1)%MAX;
      queue[rear]=ele;
      if(front ==-1)
         front=0;



    }
}
void Deque()
{
    int item;
    if((front == -1)&&(rear == -1))
    {

        printf("Queue is empty");
    }
    else
    {
        item=queue[front];
        printf("Removed element from the queue %d",item);
        if(front==rear)
        {
           front=-1;
```

```
        rear=-1;
      }
      else
      {
        front=(front+1)%MAX;
      }

   }

}

void display()
{
   int i;
   if((front==-1)&& (rear==-1))
   {

      printf("Queue is empty\n");return;

   }
   else
   {
      printf("\nQueue contents:\n");
      i=front;
      do
```

```
    {
      printf("%d\t",queue[i]);
      if(i==rear)
        break;
      i=(i+1)%MAX;
    }while (i!=front);

  }
}
```

Output




Console
Shell
Circular Queue
1. Insert to Queue (EnQueue)
2. delete from the Queue (DeQueue)
3. Display the content
4. Exit
Enter the option :1
Enter the element
20
Queue is full
Circular Queue
1. Insert to Queue (EnQueue)
2. delete from the Queue (DeQueue)
3. Display the content
4. Exit
Enter the option :3
Queue contents:
28 12 2000
Circular Queue
1. Insert to Queue (EnQueue)
2. delete from the Queue (DeQueue)
3. Display the content
4. Exit
Enter the option :2
Removed element from the queue 28
Circular Queue
1. Insert to Queue (EnQueue)
2. delete from the Queue (DeQueue)
3. Display the content
4. Exit
Enter the option :

Singly Linked List (Insert, Delete, Sort, Reverse)

```
#include <stdio.h>

#include <malloc.h>


typedef struct Node {

  int data;

  struct Node *next;

} node;


node* head = NULL; //The first node in the list

node* temp = NULL; //The last node in the list

int count = 0; //Tracks the number of elements in the node


//Push element (add at beginning) of List

void push(int data) {

  node* nPtr = (node*)malloc(sizeof(node));


  nPtr->data = data;


  //Checks for empty list

  if (head == NULL) {

    nPtr->next = NULL;

    head = nPtr;
```

```
  printf("First node created: %d\n", data);
  count++;
  return;
 }

 //Pushes new node
 nPtr->next = head;
 head = nPtr;
 count++;
 printf("New node added at beginning: %d\n", data);
}

//Adding new node at nth position (Offset from first node) in List
void addAtPos(int data, int offset) {
 node* nPtr = (node*)malloc(sizeof(node));
 temp = head;
 int pos = offset;

 nPtr->data = data;

 //Validates the offset
 if (offset > (count + 1) || offset < 0) {
  printf("Out of limits.\n");
  return;
 }
```

```
//Checks for empty list
if (head == NULL) {
  nPtr->next = NULL;
  head = nPtr;
  printf("First node created: %d\n", data);
  count++;
  return;
}

//Checks if offset is at beginning of list
if (offset == 0){
  push(data);
  return;
}

//Traverses the list until offset is 0
while (--offset) {
  temp = temp->next;

  if (offset == 0)
    break;
}

//Adds new node to the list
```

```
  nPtr->next = temp->next;
  temp->next = nPtr;
  printf("New node added at %d: %d\n", pos, data);
  count++;
 }

//Adding new node to end of List
void append(int data) {
  node* nPtr = (node*)malloc(sizeof(node));
  temp = head;

  nPtr->data = data;

  //Checks for empty list
  if (head == NULL) {
    nPtr->next = NULL;
    head = nPtr;
    printf("First node created: %d\n", data);
    count++;
    return;
  }

  //Traverses the list till temp is NULL
  while (temp->next!=NULL) {
    temp = temp->next;
```

```
  }

  //Appends the new node to the list
  temp->next = nPtr;
  printf("New node appended: %d\n", data);
  count++;
}

//Deleting node at beginning of list
void delAtStart() {
  node* temp;

  //Checks for empty list
  if (head == NULL) {
    printf("Empty list\n");
    return;
  }

  //Deletion of first element
  temp = head;
  printf("Removed = %d\n", head->data);
  head = head->next;
  free(temp);
  count--;
}
```

```
void delAtEnd();


//Deleting node at specified location
void delAtPos(int offset) {
 node *temp = head, *del = NULL;


 //Validates the given offset within the list
 if ((offset > count) || (offset < 1)){
  printf("Out of bounds\n");
  return;
 }


 //Checks for empty list
 if (head == NULL) {
  printf("Empty list\n");
  return;
 }


 //Checks if specified offset is the first element in the list
 if (offset == 0) {
  delAtStart();
  return;
 }
```

```
  //Checks if specified offset is the last element in the list
  if (offset == count) {
    delAtEnd();
    return;
  }

  //Traverse through the entire list
  while (--offset > 1) {
    temp = temp->next;
  }

  del = temp->next;
  temp->next = temp->next->next;
  printf("Removed = %d\n", del->data);
  free(del);
  count--;
}

//Deleting node at end of list
void delAtEnd() {
  node* temp = head;

  //Checks for empty list
  if (head == NULL) {
    printf("Empty list\n");
```

```
    return;
  }

  //Checks if only 1 node remains
  if (temp->next == NULL) {
    printf("Removed = %d\n", temp->data);
    free(temp);
    head = NULL;
    count--;
    return;
  }

  //Traverses the list
  while (temp->next->next != NULL) {
    temp = temp->next;
  }

  //Deletion of last element
  printf("Removed = %d\n", temp->next->data);
  free(temp->next);
  temp->next = NULL;
  count--;
}

void swap(node *a, node *b);
```

```
//Sorts the elements in the list
void Sort(struct Node *start)
{
  int swapped, i;
  node *ptr1;
  node *lptr = NULL;

  //Checking for empty list
  if (start == NULL)
    return;

  do
  {
    swapped = 0;
    ptr1 = start;

    while (ptr1->next != lptr) {
      if (ptr1->data > ptr1->next->data) {
        swap(ptr1, ptr1->next);
        swapped = 1;
      }

      ptr1 = ptr1->next;
    }
```

```
    lptr = ptr1;
  }
  while (swapped);
}

//Swap data of two nodes a and b
void swap(struct Node *a, struct Node *b)
{
  int temp = a->data;
  a->data = b->data;
  b->data = temp;
}

//Reverses the order of elements in the list
static void reverse(struct Node** head_ref)
{
  node* prev = NULL;
  node* current = *head_ref;
  node* next = NULL;
  while (current != NULL) {
    // Store next
    next = current->next;

    // Reverse current node's pointer
```

```
    current->next = prev;

    // Move pointers one position ahead.
    prev = current;
    current = next;
  }

  *head_ref = prev;
}

//Print all elements in the list
void printList() {
  node* nPtr = head;

  printf("List conatins: \n");

  //Checks for empty list
  if (head == NULL) {
    printf("\tNothing!!\n");
    return;
  }

  //Traverses the list while printing the elements
  while (nPtr != NULL) {
    printf("\t\t\t%d\n", nPtr->data);
```

```
    nPtr = nPtr->next;
  }

  //Gives node count in the list
  printf("Node count in current list: %d\n", count);
}

int main(void) {
  printList();
  append(28);
  append(12);
  append(10);
  append(20);
  push(2000);
  push(1);
  printf("Initial list with all elements: \n");
  printList();
  printf("Sorted list with all elements: \n");
  Sort(head);
  printList();
  printf("Reversed order of elements in list: \n");
  reverse(&head);
  printList();
  delAtPos(5);
  printList();
```

```
 addAtPos(100, 5);

 printList();

 delAtStart();

 printList();

 delAtPos(5);

 printf("List after deleting certain elements: \n");

 printList();
}
```

Output

```
            10
            1
Node count in current list: 6
Removed = 10
List conatins:
            2000
            28
            20
            12
            1
Node count in current list: 5
New node added at 5: 100
List conatins:
            2000
            28
            20
            12
            1
            100
Node count in current list: 6
Removed = 2000
List conatins:
            28
            20
            12
            1
            100
Node count in current list: 5
Removed = 100
List after deleting certain elements:
List conatins:
            28
            20
            12
            1
Node count in current list: 4
>
```

Singly Linked List (Concatenate, Merge)

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


        struct node
        {
         int data;
         struct node *next;
        };
        struct node *head=NULL;
        struct node *head1=NULL;
        struct node *head2=NULL;
        struct node *head3=NULL;
        void Reverse()
        {
            struct node *newnode,*temp;
          int item;
          int choice;
          do
          {
          newnode =(struct node *) malloc (sizeof(struct node));
          printf("Enter the data : ");
          scanf("%d",&item);
```

```
newnode->data=item;
newnode->next=NULL;
if (head==NULL)
 {

  head=newnode;

 }
 else
{
temp=head;
   while(temp->next!=NULL)
   {
          temp=temp->next;
    }
  temp->next=newnode;
  newnode->next=NULL;

}
 printf("Do u want to add element 1-yes, 2-no\n");
 fflush(stdin);
 scanf("%d",&choice);
}while(choice==1);

struct node *prev=NULL,*current=head, *next=NULL;
```

```
while(current!=NULL)
{
   next=current->next;
   current->next=prev;
   prev=current;
   current=next;
}
head=prev;

printf("DISPLAY:\n");
  struct node *ptr;
ptr=head;

if(ptr==NULL)
{
   printf("Nothing to print\n");
}
else
{
   while(ptr!=NULL)
  {
   printf("%d ",ptr->data);
   ptr=ptr->next;
  }
}
```

```
}
void Concat()
{
     struct node *newnode1,*temp1;
   int item1;
   printf("LIST ONE ELEMENTS\n");
   int choice1;
   do
   {
   newnode1 =(struct node *) malloc (sizeof(struct node));
   printf("Enter the data : ");
   scanf("%d",&item1);
   newnode1->data=item1;
   newnode1->next=NULL;
   if (head1==NULL)
    {

     head1=newnode1;

    }
    else
   {
   temp1=head1;
     while(temp1->next!=NULL)
     {
```

```
        temp1=temp1->next;
    }
  temp1->next=newnode1;
  newnode1->next=NULL;

}
 printf("Do u want to add element 1-yes, 2-no\n");
 fflush(stdin);
 scanf("%d",&choice1);
}while(choice1==1);

struct node *newnode2,*temp2;
int item2;
printf("LIST TWO ELEMENTS\n");
int choice2;
do
{
newnode2 =(struct node *) malloc (sizeof(struct node));
printf("Enter the data : ");
scanf("%d",&item2);
newnode2->data=item2;
newnode2->next=NULL;
if (head2==NULL)
 {
```

```
  head2=newnode2;


 }
 else
{
 temp2=head2;
    while(temp2->next!=NULL)
   {
          temp2=temp2->next;
    }
   temp2->next=newnode2;
   newnode2->next=NULL;


}
  printf("Do u want to add element 1-yes, 2-no\n");
  fflush(stdin);
  scanf("%d",&choice2);
 }while(choice2==1);


 temp1=head1;
 temp2=head2;


 while(temp1->next!=NULL)
     temp1=temp1->next;
```

```
    temp1->next=temp2;


    printf("DISPLAY:\n");
    struct node *ptr;
  ptr=head1;


  if(ptr==NULL)
  {
     printf("Nothing to print\n");
  }
  else
  {
     while(ptr!=NULL)
    {
     printf("%d ",ptr->data);
     ptr=ptr->next;
    }
  }
}
void Sort()
{
    struct node *newnode3,*temp3;
  int item;
  int choice;
  do
```

```
{
newnode3 =(struct node *) malloc (sizeof(struct node));
printf("Enter the data : ");
scanf("%d",&item);
newnode3->data=item;
newnode3->next=NULL;
if (head3==NULL)
 {

  head3=newnode3;


 }
 else
{
temp3=head3;
   while(temp3->next!=NULL)
   {
          temp3=temp3->next;
    }
  temp3->next=newnode3;
  newnode3->next=NULL;

}
  printf("Do u want to add element 1-yes,2-no\n");
  fflush(stdin);
```

```
 scanf("%d",&choice);
}while(choice==1);
struct node *count;
temp3=head3;
struct node *min;

int i;
while(temp3!=NULL)
{
        min=temp3;
        count=temp3;
        while(count!=NULL)
        {
                if(count->data<=min->data)
                min=count;
                count=count->next;
        }
        i=temp3->data;
        temp3->data=min->data;
        min->data=i;

        temp3=temp3->next;
  }

  printf("DISPLAY:\n");
```

```
    struct node *ptr;
   ptr=head3;

   if(ptr==NULL)
   {
      printf("Nothing to print\n");
   }
   else
   {
      while(ptr!=NULL)
     {
      printf("%d ",ptr->data);
      ptr=ptr->next;
     }
   }
}



int main()
{

     int choice;
```

```
do
{
	printf("\n1. Reverse\n2. Sorting\n3. Concatenation\n4. Exit\n");
	printf("enter choice\n");
	scanf("%d",&choice);
	switch(choice)
	{
		case 1:
		Reverse();
		break;

		case 2:
		Sort();
		break;

		case 3:
		Concat();

		case 4:
		break;

		default:
		printf("WRong\n");
		break;
	}
```

```
        }while(choice!=4);

        return 0;

}
```

Output

```
1. Reverse
2. Sorting
3. Concatenation
4. Exit
enter choice
1
Enter the data : 1
Do u want to add element 1-yes, 2-no
1
Enter the data : 2
Do u want to add element 1-yes, 2-no
1
Enter the data : 3
Do u want to add element 1-yes, 2-no
2
DISPLAY:
3 2 1
1. Reverse
2. Sorting
3. Concatenation
4. Exit
enter choice
2
Enter the data : 10
Do u want to add element 1-yes,2-no
1
Enter the data : 3
Do u want to add element 1-yes,2-no
1
Enter the data : 12
Do u want to add element 1-yes,2-no
2
```

```
DISPLAY:
3 10 12
1. Reverse
2. Sorting
3. Concatenation
4. Exit
enter choice
3
LIST ONE ELEMENTS
Enter the data : 1
Do u want to add element 1-yes, 2-no
1
Enter the data : 2
Do u want to add element 1-yes, 2-no
1
Enter the data : 3
Do u want to add element 1-yes, 2-no
1
Enter the data : 4
Do u want to add element 1-yes, 2-no
2
LIST TWO ELEMENTS
Enter the data : 5
Do u want to add element 1-yes, 2-no
1
Enter the data : 6
Do u want to add element 1-yes, 2-no
1
Enter the data : 5
Do u want to add element 1-yes, 2-no
1
Enter the data : 6
Do u want to add element 1-yes, 2-no
1
Enter the data : 7
Do u want to add element 1-yes, 2-no
2
DISPLAY:
1 2 3 4 5 6 7
1. Reverse
2. Sorting
3. Concatenation
4. Exit
enter choice
4


------------------
(program exited with code: 0)
```

Singly Linked List (As stacks and queues)

Case 1: As stacks

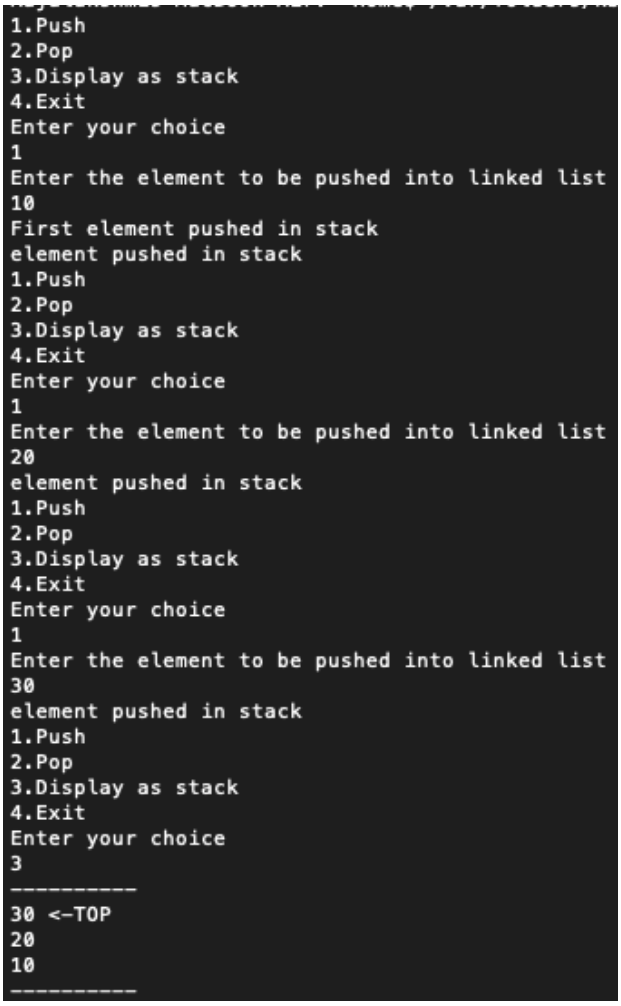
```
#include <stdio.h>

#include <stdlib.h>


struct node

{

        int data;

        struct node *next;

};

struct node *head=NULL;

void push(int it)

{

        struct node *newnode;

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

   newnode->data =it;

   newnode->next=NULL;

   if(head==NULL)

   printf("First element pushed in stack\n");

   newnode->next=head;

   head=newnode;

   printf("element pushed in stack\n");

}
```

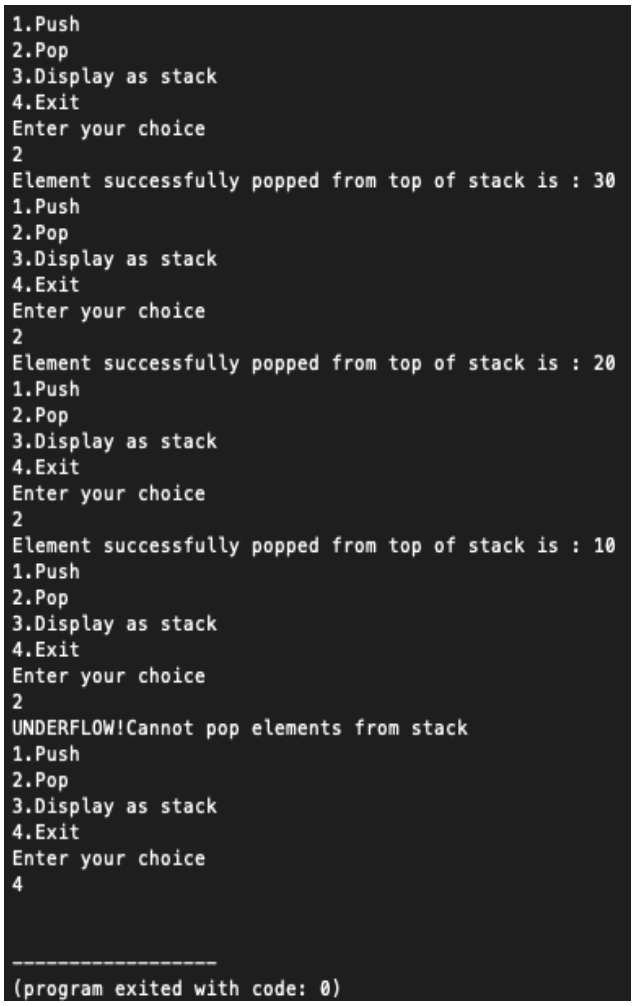
```
void pop()
{
        if(head==NULL)
        printf("UNDERFLOW!Cannot pop elements from stack\n");
        else
        {
                struct node *temp;
                temp=head;
                head=head->next;
                printf("Element successfully popped from top of stack is : %d\n",temp->data);
                free(temp);
        }
}
void display()
{
        struct node *ptr;
   ptr=head;
    if(ptr==NULL)
   {
      printf("Stack is empty!\n");
   }
   else
        {
      printf("%d <-TOP\n",ptr->data);
                ptr=ptr->next;
```

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

  }

}

int main()
{
                int ch,ele;
                do
                {
                        printf("\n1.Push\n2.Pop\n3.Display as stack\n4.Exit\n");
                        printf("Enter your choice\n");
                        scanf("%d",&ch);
                        if(ch==4)
                        break;
                        switch(ch)
                        {
                                case 1:
                                printf("Enter the element to be pushed into linked
list\n");
                                scanf("%d",&ele);
```

```
                push(ele);
                break;

                case 2:
                pop();
                break;

                case 3:
                printf("----------\n");
                display();
                printf("----------\n");
                break;

                case 4:
                break;

                default:
                printf("wrong choice!\n");
                break;
            }
        }while(ch!=4);
        return 0;
}
```

Output





Case 2: As queues

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

        int data;

        struct node *next;
```

```
};
struct node *head=NULL;
void enq(int it)
{
        struct node *newnode;
        struct node *temp;
        newnode=(struct node*)malloc(sizeof(struct node));
   newnode->data =it;

   if (head==NULL)
   {
    newnode->next=NULL;
    head=newnode;
    printf("Very first element of queue is created \n");
   }
    else
    {
                temp=head;
      while(temp->next!=NULL)
      {
                        temp=temp->next;
      }
                temp->next=newnode;
                newnode->next=NULL;
                printf("Element enqueued in list\n");
```

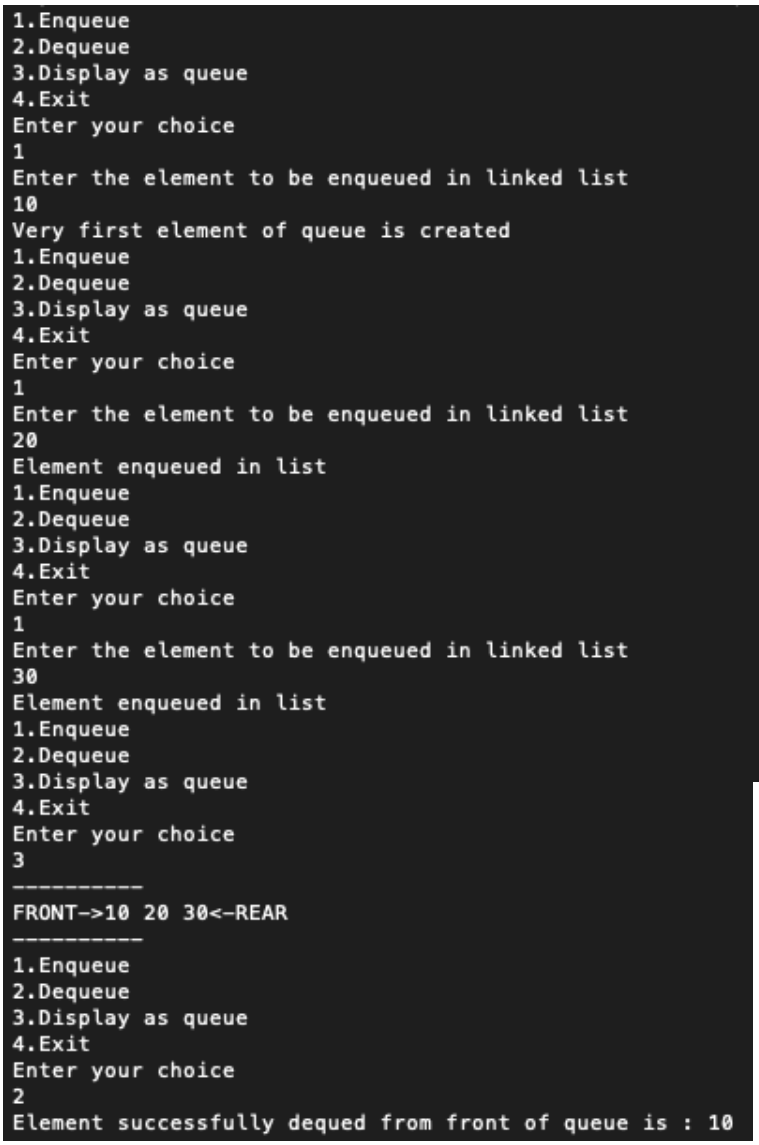
```
            }
}
void dq()
{
            if(head==NULL)
            printf("UNDERFLOW!Cannot delete elements from queue\n");
            else
            {
                        struct node *temp;
                        temp=head;
                        head=head->next;
                        printf("Element successfully dequed from front of queue is : %d\n",temp->data);
                        free(temp);
            }
}
void display()
{
            struct node *ptr;
   ptr=head;
    if(ptr==NULL)
   {
      printf("queue is empty!\n");
   }
   else if(ptr->next==NULL)
   printf("FRONT->%d<-REAR\n",ptr->data);
```

```
    else
        {
        printf("FRONT->%d ",ptr->data);
                ptr=ptr->next;
        while(ptr->next!= NULL)
        {
                printf("%d ",ptr->data);
                ptr=ptr->next;
        }
        printf("%d<-REAR\n",ptr->data);
        ptr=NULL;

    }

}

int main()
{
        int ch,ele;
        do
        {
                printf("\n1.Enqueue\n2.Dequeue\n3.Display as queue\n4.Exit\n");
                printf("Enter your choice\n");
                scanf("%d",&ch);
                if(ch==4)
```

```
break;
switch(ch)
{
        case 1:
        printf("Enter the element to be enqueued in linked list\n");
        scanf("%d",&ele);
        enq(ele);
        break;

        case 2:
        dq();
        break;

        case 3:
        printf("----------\n");
        display();
        printf("----------\n");
        break;

        case 4:
        break;

        default:
        printf("wrong choice!\n");
        break;
```

```
                }
            }while(ch!=4);
        return 0;
    }
```

Output



```
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
1
Enter the element to be enqueued in linked list
10
Very first element of queue is created
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
1
Enter the element to be enqueued in linked list
20
Element enqueued in list
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
1
Enter the element to be enqueued in linked list
30
Element enqueued in list
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
3
----------
FRONT->10 20 30<-REAR
----------
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
2
Element successfully dequed from front of queue is : 10
```

```
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
2
Element successfully dequed from front of queue is : 20
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
2
Element successfully dequed from front of queue is : 30
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
2
UNDERFLOW!Cannot delete elements from queue
1.Enqueue
2.Dequeue
3.Display as queue
4.Exit
Enter your choice
4


------------------
(program exited with code: 0)
```

Doubly Linked List (Insertion, Deletion, Display)

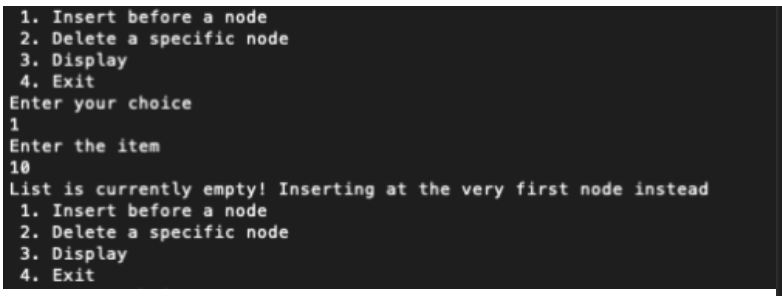
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next;
        struct node *prev;
};
struct node *head=NULL;
void insert_bef()
{
        int listele;
        struct node *new_node,*temp;
         new_node=(struct node*)malloc(sizeof(struct node));
         printf("Enter the item\n");
         scanf("%d",&new_node->data);
         new_node->next=NULL;
         new_node->prev=NULL;

         if(head==NULL)
         {
                 printf("List is currently empty! Inserting at the very first node instead\n");
                 head=new_node;
        }
        else
        {
                printf("Enter the element in the list\n");
                scanf("%d",&listele);
                temp=head;
                if(head->data== listele)
                {
                        new_node->next=head;
                        head->prev=new_node;
                        head=new_node;
                }
                else
                {
                        while(temp->next->data!=listele)
                        {
                                temp=temp->next;
```

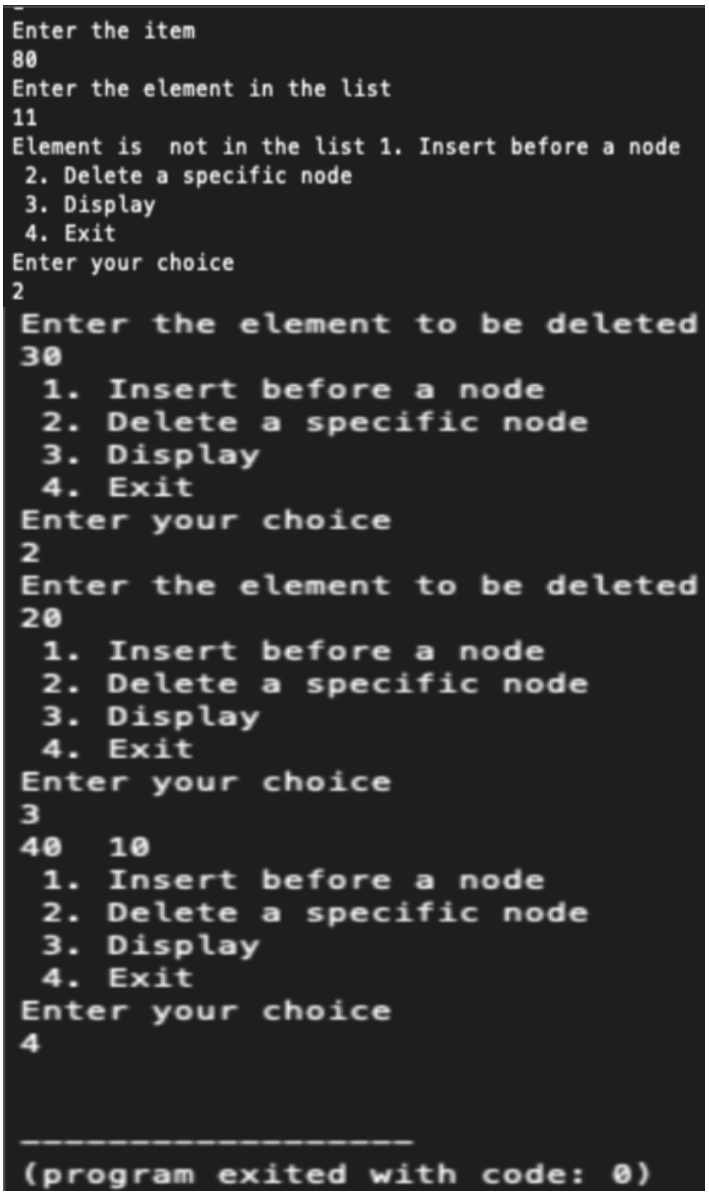
```
                                        if(temp->next==NULL)
                                        {
                                                printf("Element is  not in the list");
                                                return;
                                        }
                                }
                                new_node->next= temp->next;
                                new_node->prev= temp;
                                temp->next->prev=new_node;
                                temp->next=new_node;

                        }
                }
}
void del()
{
        struct node *temp;
        int ele;
  if(head==NULL)
  {
     printf("Empty List \n");
     return;
  }
        printf("Enter the element to be deleted\n");
        scanf("%d",&ele);
        temp=head;
        while(temp->data!=ele)
        {
                temp=temp->next;
                if(temp==NULL)
                {
                 printf("Element is not in the list\n");
                 return;
                }
        }
        if(temp==head)
        {
                 head=head->next;
        }
        else if(temp->next==NULL)
        {
                        temp=temp->prev;
                        temp->next=NULL;
        }

        else
```

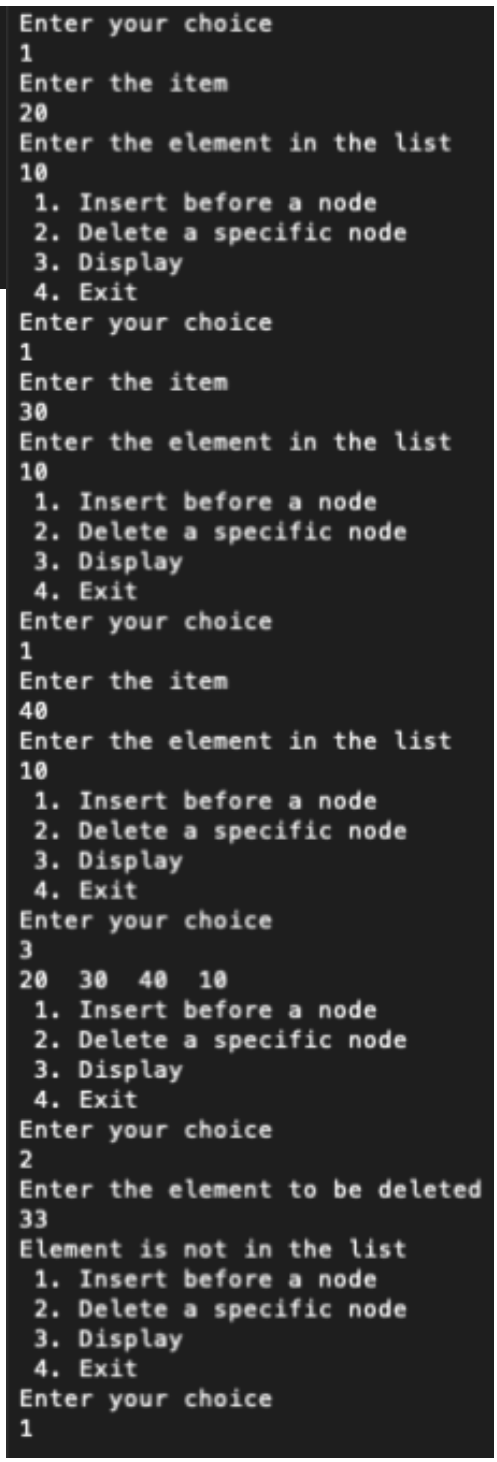
```
        {
                temp->prev->next=temp->next;
                temp->next->prev=temp->prev;
        }
}
void display()
{
        struct node *temp;
        temp=head;
        while(temp!=NULL)
        {
                printf("%d  ",temp->data);
                temp=temp->next;
        }
        printf("\n");
}
int main()
{
        int choice;

        do
        {
                        printf(" 1. Insert before a node \n");
                        printf(" 2. Delete a specific node \n");
                        printf(" 3. Display\n");
                        printf(" 4. Exit\n");
                        printf("Enter your choice\n");
                        scanf("%d",&choice);
                        switch(choice)
                        {
                                case 1: insert_bef();
                                break;

                                case 2: del();
                                break;

                                case 3: display();
                                break;

                                case 4:
                                break;

                                default:
                                printf("Wrong choice!\n");
                                break;
                        }
```

```
}while(choice!=4);
return 0;

}
```

Output







Binary Search Tree (Insertion, Traversal)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node* left;
        struct node *right;
};
struct node *create()
{
        struct node *temp;
        printf("\n Enter data:");
        temp=(struct node*)malloc(sizeof(struct node));
        scanf("%d",&temp->data);
        temp->left=temp->right=NULL;
        return temp;
}
 void insert(struct node *root,struct node *temp)
{
        if(temp->data<root->data)
        {
                if(root->left!=NULL)
                        insert(root->left,temp);
                else
                        root->left=temp;
        }

        if(temp->data>root->data)
        {
                if(root->right!=NULL)
                        insert(root->right,temp);
                else
                        root->right=temp;
        }
}
```

```
void preorder(struct node *root)
{
	if(root!=NULL)
	{
		printf("%d ",root->data);
		preorder(root->left);
		preorder(root->right);
	}
}
void postorder(struct node *root)
{
	if(root!=NULL)
	{
		postorder(root->left);
		postorder(root->right);
		printf("%d ",root->data);
	}
}
void inorder(struct node *root)
{
	if(root!=NULL)
	{
		inorder(root->left);
		printf("%d ",root->data);
		inorder(root->right);
	}
}
int main()
{
	int ch,count=1;
	struct node *tree;
	struct node *rt;
	do
	{
		printf("1.Create and insert node in
BST\n2.preorder\n3.postorder\n4.inorder\n5.exit\n");
		printf("enter choice\n");
		scanf("%d",&ch);
		switch(ch)
		{
```

```
                        case 1:
                        if(count==1)
                        {
                                rt=create();
                                count++;
                        }
                        else
                        {
                                tree=create();
                                insert(rt,tree);
                        }
                        break;

                        case 2:
                        preorder(rt);
                        break;

                        case 3:
                        postorder(rt);
                        break;

                        case 4:
                        inorder(rt);
                        break;

                        case 5:
                        break;

                        default:
                        printf("wrong choice!\n");
                        break;
                }
        }while(ch!=5);
        return 0;
}
```

Output

```
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
1

 Enter data:10
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
1

 Enter data:7
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
1

 Enter data:12
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
1
```

```
 Enter data:5
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
1

 Enter data:9
1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
2
10 7 5 9 12 1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
3
5 9 7 12 10 1.Create and insert node in BST
```

```
5 9 7 12 10 1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
4
5 7 9 10 12 1.Create and insert node in BST
2.preorder
3.postorder
4.inorder
5.exit
enter choice
5
```