EXP NO: 01

DATE:23.08.2022

IMPLEMENTATION OF SINGLY LINKED LIST AND ITS OPERATION

AIM:

To write c programs to implement singly linked list and its operations.

PROGRAM 1.1: INSERT A NODE AT THE HEAD OF A LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the node ; store it in the data field

STEP 4: Aline the value in the given value.

STEP 5: If head is equal to NULL asign the value to the .

STEP 6: Else follow step 6.1.

6.1:newnode->next=head.and head=newnode.

STEP 7: Display fuction.

STEP 8: Creat the Structure.

STEP 9: Assign the p to head.

STEP 10: Using while loop

10.1:p->next!=NULL.

10.2:print the values.

10.3:p=p->next.

10.4: Print the last value.

STEP 11: Stop.

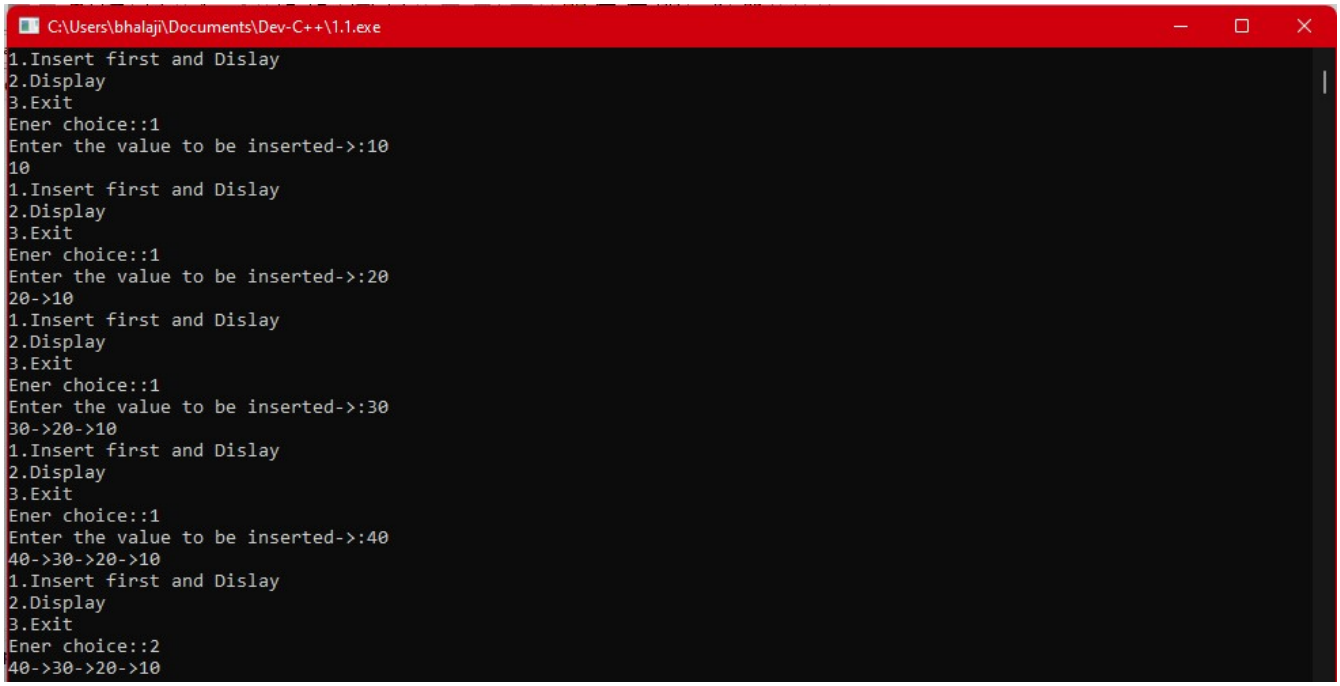
PROGRAM 1.1-CODING:

```
//*********INSERT FIRST AND DISPLAY*********
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
```

```
{
        int value;
        struct node *next;
}*head=NULL;
void insertfirst();
void display();
int main()
{
        int ch;
        while(1)
        {
                printf("\nEnter the choice\n1.Insert first and Dislay\n2.Display\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                insertfirst();
                                display();
                                break;
                        case (2):
                                display();
                                break;
                        case (3):
                                exit(0);
                }
        }
        return 0;
}
void insertfirst()
{
        int data;
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter the value to be inserted->:");
```

```
        scanf("%d",&data);
        newnode->value=data;
        newnode->next=NULL;
        if(head==NULL)
        {
                head=newnode;
        }
        else
        {
                newnode->next=head;
                head=newnode;
        }
}
void display()
{
        struct node *p;
        p=head;
        while(p->next!=NULL)

                printf("%d->",p->value);
                p=p->next;
        }
    printf("%d",p->value);
}
```

PROGRAM 1.1-OUTPUT:



PROGRAM 1.2: REMOVE DUPLICATES FROM SORTED LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the node ; store it in the data field

STEP 4: Assign p as head.

STEP5: For deleting the duplicate first we have to sort the list.

STEP6: Using for loop p=head and p!=NULL inctement the value by p=p->next.

STEP7: Inside that for loop using another for loop s==p->next until s!=NULL increment the value by s=s->next.

STEP8:if the p->value > s->value

8.1.temp=p->value and p->value=s->value and s->value=temp.

STEP9: Now removinng the duplicates create the structure *p and *temp.

STEP10: Assign the p to head and if the head==NULL there id=s no list.

STEP11: Else using while loop p->next !=NULL

11.1:if the p->value==p->next->value then follow the next step

11.2:temp=p->next->value then free(p->next) then p->next=temp

11.3: Else p=p->next.

STEP12: Then display the elements after removing the duplicate element.

STEP 7: Stop.

PROGRAM 1.2-CODING:

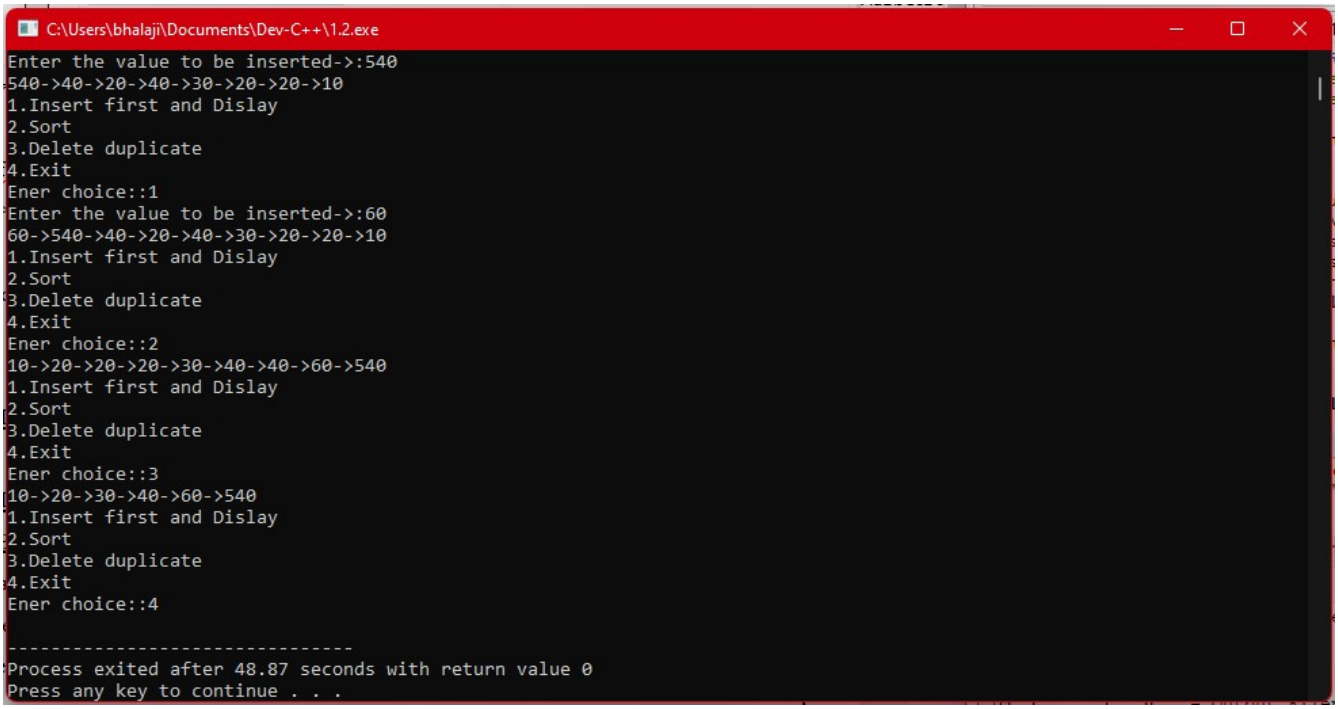
```
/******REMOVR THE DUPLICATE ELEMENTS FROM THE SORTED LINKED LIST******/
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int value;
    struct node *next;
}*head=NULL;

void insertfirst();
void display();
void sort();

void deleteduplicate();
int main()
{
      int ch;
      while(1)
      {
          printf("\nEnter the choice\n1.Insert first and Dislay\n2.Sort\n3.Delete duplicate\n4.Exit\nEner choice::");
          scanf("%d",&ch);
          switch(ch)
          {
                case (1):
                      insertfirst();
                      display();
                      break;
                case (2):
                      sort();
                      display();
                      break;
                case (3):
```

```
                    deleteduplicate();
                    display();
                    break;
                default:
                    exit(0);
            }
        }
        return 0;
}
void insertfirst()
{
   int data;
   struct node *newnode;
   newnode=(struct node*)malloc(sizeof(struct node));
   printf("Enter the value to be inserted->:");
   scanf("%d",&data);
   newnode->value=data;
   newnode->next=NULL;
   if(head==NULL)
   {
      head=newnode;
   }
   else
   {
      newnode->next=head;
      head=newnode;
   }
}
void sort()
{
   struct node *p,*s;
   int temp;
   for(p=head;p!=NULL;p=p->next)
   {
```

```
        for(s=p->next;s!=NULL;s=s->next)
        {
             if(p->value>s->value)
             {
                    temp=p->value;
                    p->value=s->value;
                    s->value=temp;
             }
        }
    }
}
void deleteduplicate()
{
    struct node *p,*temp;
    p=head;
    if(head==NULL)
    {
        printf("There is no list");
    }
    else
    {
        while(p->next!=NULL)
        {
           if(p->value==p->next->value)
           {
              temp=p->next->next;
              free(p->next);
              p->next=temp;
           }
           else
          {
              p=p->next;
          }
      }
```

```
   }
}
void display()
{
     struct node *p;
     p=head;
     while(p->next!=NULL)
     {
          printf("%d->",p->value);
          p=p->next;
       }
    printf("%d",p->value);
}
```

PROGRAM 1.2-OUTPUT:



PROGRAM 1.3: REVERSE A LINKED LIST IN K GROUP

ALGORITHM:

STEP 1: Start

STEP 2: Create the list by following steps 3-7

STEP 3: Create the object and allocates memory for object using malloc function

STEP 4: Get the value to be stored in the node store it in the data field and assign newnode next pointerto NULL.

STEP 5: Check the existence of list by checking the head pointer is equivalent to null or not

STEP 6: If head==NULL then there is no list already existing creating newnode is the first node of thelist by making the head pointer to point to the newnode.

STEP 7: If head!=NULL then there exist a list already

7.1: To find the last node assign the head value to pointer p and traverse the list untilp->next becomes NULL and assign the newnode to p->next

STEP 8: To reverse the list in kth group follow step 8.1-10

8.1: First find the number of element in the list using pointer p

8.2: Intialize i=1 and increment i until p->next becomes null\

STEP 9: After finding the total number of elements(i) get k from user

9.1: create pointer p points to head and moves n/k times

9.2: create another pointer p1 which points to pointer p move the pointer p1 for k times

9.3: swap the value of p and p1 using temporary variables

STEP 10: Display the list

STEP 11: Stop

PROGRAM 1.3-CODING:

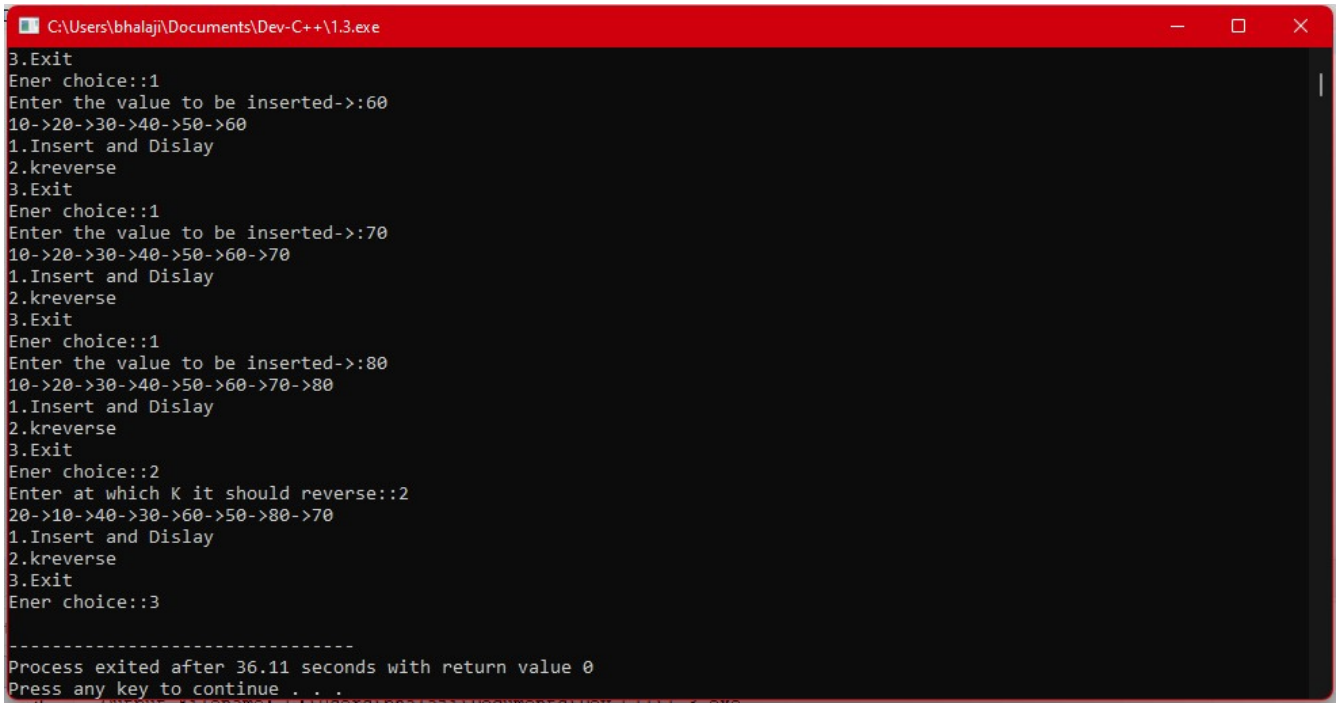
```
/******REVERSE THE ELEMENTS IN K-GROUPS-SINGLY LINKED LIST******/
#include<stdio.h>
#include<stdlib.h>
struct Node
{
        int data;
        struct Node* next;
}*head=NULL;
struct Node *newnode;
struct Node *reverse (struct Node *head, int k)
{
```

```
        if (!head)
                return NULL;
        struct Node* current = head;
        struct Node* next = NULL;
        struct Node* prev = NULL;
        int count = 0;
        while (current != NULL && count < k)
        {
                next = current->next;
                current->next = prev;
                prev = current;
                current = next;
                count++;
        }
        if (next != NULL)
           head->next = reverse(next, k);
        return prev;
}
void insertlast(struct Node *Newnode)
{
        int data;
        struct Node *p;
        Newnode=(struct Node*)malloc(sizeof(struct Node));
        printf("Enter the value to be inserted->:");
        scanf("%d",&data);
        Newnode->data=data;
        Newnode->next=NULL;
        if(head==NULL)
        {
/               head=Newnode;
        }
        else
        {
                p=head;
```

```
                while(p->next!=NULL)
                    p=p->next;
                p->next=Newnode;
        }
}
void display(struct Node *node)
{
        while(node->next!=NULL)
        {
                printf("%d->",node->data);
                node=node->next;
        }
   printf("%d",node->data);
}
int main(void)
{
   int ch,n;
        while(1)
        {
                printf("\nEnter the choice\n1.Insert and Dislay\n2.kreverse\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                insertlast(newnode);
                                display(head);
                                break;
                        case (2):
                                printf("Enter at which K it should reverse::");
                                scanf("%d",&n);
                                head = reverse (head, n);
                                display(head);
                                break;
                        case (3):
```

```
                    exit(0);
                }
        }
        return(0);
}
```

PROGRAM 1.3-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on singly linked list are executed and outputs are verified.

EXP NO: 02

DATE:30.09.2022

IMPLEMENTATION OF DOUBLY LINKED LIST AND ITS OPERATION

AIM:

To write c programs to implement doubly linked list and its operations.

PROGRAM 2.1: REMOVE ELEMENT AT THE K TH POSITION LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: For Remove the node follow the steps 4-7.

STEP 4: Create the structure in *p.

STEP 5: If the head == NULL then print there is no element to remove

5.1else get the position to be removed.

STEP 6: Assign the the p=head and pos =1

6.1: Using while condition kpos!=pos

6.2:p=p->next then pos++

STEP7:p->prev->next=p->next then p->next->prev=p->prev then free(p);

STEP8: Print the elements present in the after removal of the given position element.

STEP 7: Stop.

PROGRAM 2.1-CODING:

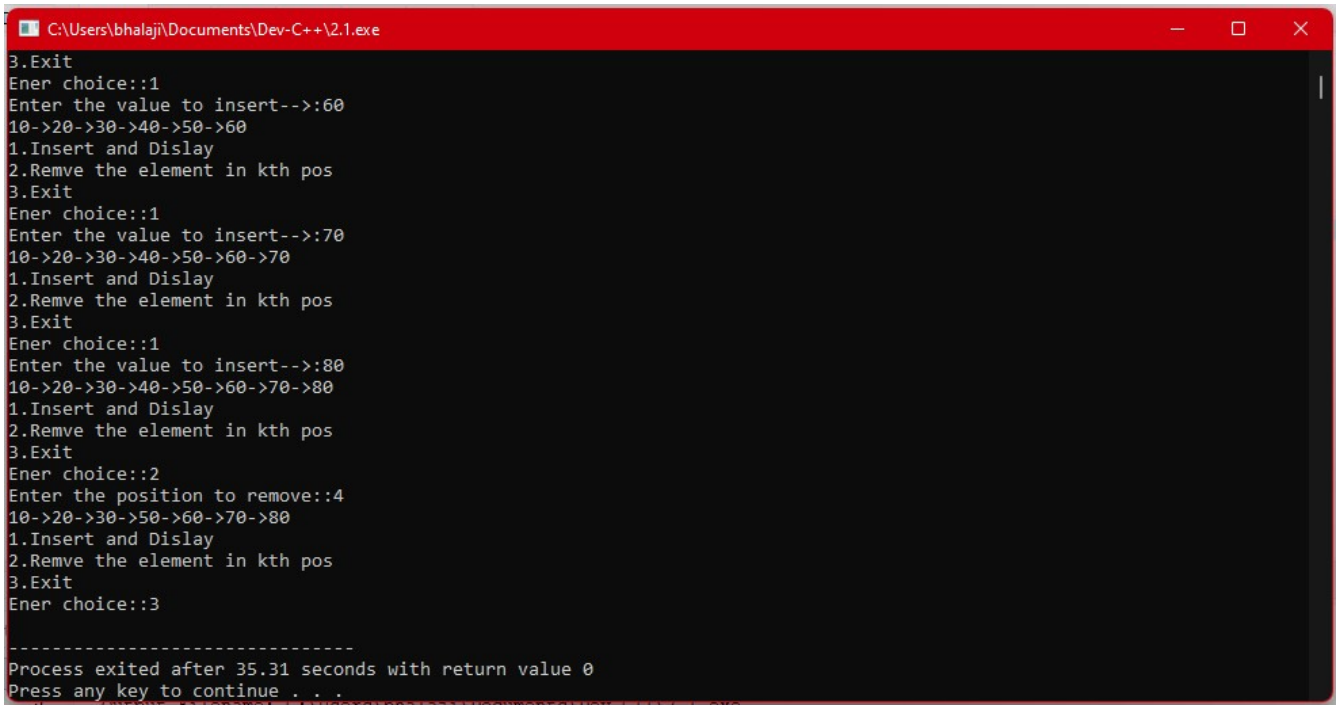
```
/******REMOVE ELEMENTS AT Kth POSITION IN DOUBLY LINKED LIST******/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        struct node *prev;
        int value;
        struct node *next;
```

```
}*head=NULL;
void insert();
void removekthpos();
void insert()
{
        int data;
        struct node *newnode,*p;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter the value to insert-->:");
        scanf("%d",&data);
        newnode->value=data;
        newnode->next=NULL;
        newnode->prev=NULL;
        if(head==NULL)
           head=newnode;
        else
        {
                p=head;
           while(p->next!=NULL)
                p=p->next;
                p->next=newnode;
                newnode->prev=p;
        }
}
void removekthpos()
{
        int pos,kpos;
        struct node *p;
        if(head==NULL)
           printf("**NO ELEMENT TO REMOVE**");
        else
        {
                printf("Enter the position to remove::");
                scanf("%d",&kpos);
```

```
                p=head;
                pos=1;
                while(kpos!=pos)
                {
                        p=p->next;
                        pos++;
                }
                p->prev->next=p->next;
                p->next->prev=p->prev;
                free(p);
        }
}
void display()
{
        struct node *p;
        p=head;
        while(p->next!=NULL)
        {
                printf("%d->",p->value);
                p=p->next;
        }
    printf("%d",p->value);
}
int main()
{
        int ch;
        while(1)
        {
                printf("\n1.Insert and Dislay\n2.Remve the element in kth pos\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                insert();
```

```
                display();
                break;
            case (2):
                removekthpos();
                display();
                break;
            case (3):
                exit(0);
        }
    }
    return 0;
}
```

PROGRAM 2.1-OUTPUT:



PROGRAM 2.2: ADD ONE TO DOUBLY LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: For inserting the element to the doubly linked list follow step 2-5.1

STEP3: Create object and allocate memory using malloc function.

STEP 4: Get the value to be stored in the node ; store it in the data field

STEP 5: Set the next and the previous value to NULL.

STEP 6:if the head ==NULL the head=newnode

6.1: Else newnode->next=head then head->prev=newnode then head=newnode.

STEP 7: For displaying the element follow the step 7

7.1: Using the while loop p->next!=NULL

7.2: Print the values

STEP 7: Stop.

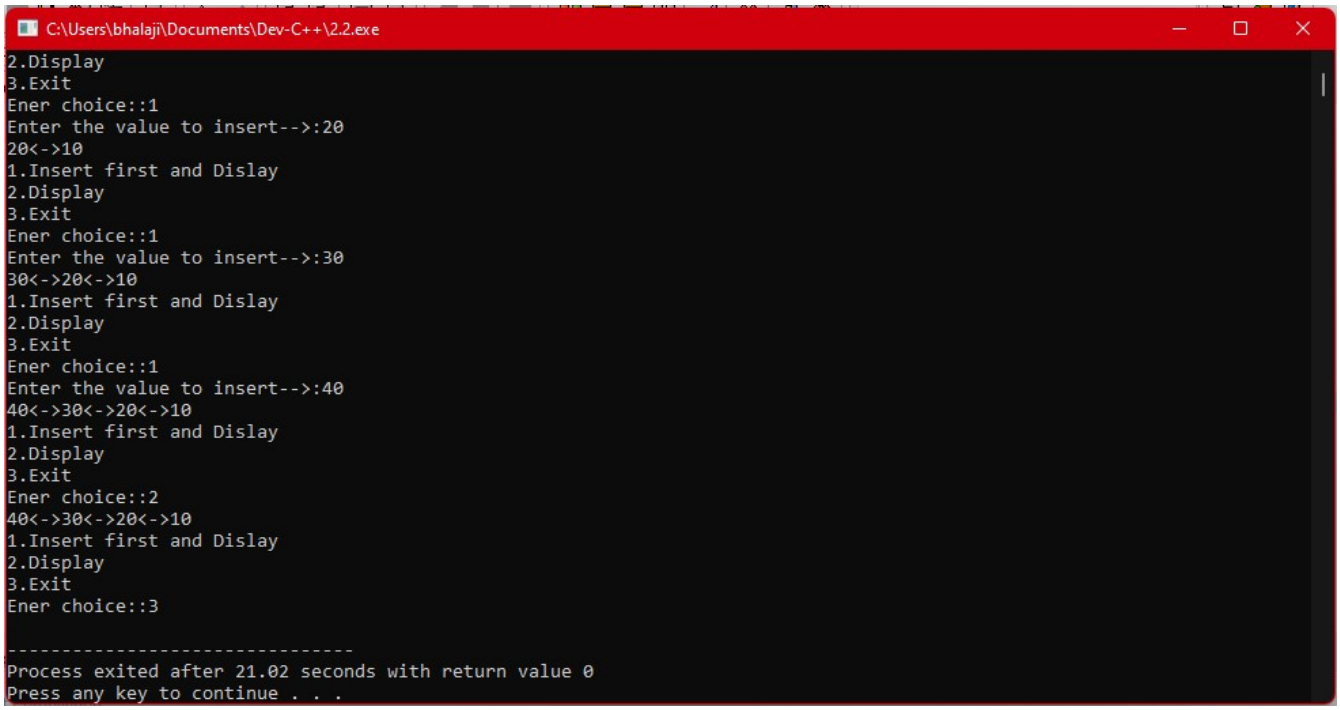
PROGRAM 2.2-CODING:

```
/******ADD ONE TO DOUBLY LINKED LIST******/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        struct node *prev;
        int value;
        struct node *next;
}*head=NULL;
void insertfirst();
void display();
void insertfirst()
{
        int data;
        struct node *newnode,*p;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter the value to insert-->:");
        scanf("%d",&data);
```

```
        newnode->value=data;
        newnode->next=NULL;
        newnode->prev=NULL;
        if(head==NULL)
           head=newnode;
        else
        {
                newnode->next=head;
                head->prev=newnode;
                head=newnode;
        }
}
void display()
{
        struct node *p;
        p=head;
        while(p->next!=NULL)
        {
                printf("%d<->",p->value);
                p=p->next;
        }
    printf("%d",p->value);
}
int main()
{
        int ch;
        while(1)
        {
                printf("\n1.Insert first and Dislay\n2.Display\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                insertfirst();
```

```
                        display();
                        break;
                case (2):
                        display();
                        break;
                case (3):
                        exit(0);
            }
        }
        return 0;
}
```

PROGRAM 2.2-OUTPUT:



PROGRAM 2.3: REVERSE A DOUBLY LINKED LIST IN K-GROUPS

ALGORITHM:

STEP 1: start

STEP 2: Create the list by following steps 3-7

STEP 3: Create the object and allocates memory for object using malloc function

STEP 4: Get the value to be stored in the node store it in the data field and assign new node next pointer to head.

STEP 5: Check the existence of list by checking the head pointer is equivalent to null or not

STEP 6: If head==NULL then there is no list already existing creating new node is the first node of the list by making the head pointer to point to the new node.

STEP 7: If head!=NULL then there exist a list already

7.1: To find the last node assign the head value to pointer p and traverse the list until p->next becomes NULL and assign the newnode to p->next

STEP 8: To reverse the list in kth group follow step 8.1-10

8.1: First find the number of element in the list using pointer p

8.2: Intialize i=1 and increment i until p->next becomes null\

STEP 9: After finding the total number of elements(i) get k from user

9.1: create pointer p points to head and moves n/k times

9.2: create another pointer p1 which points to pointer p move the pointer p1 for k times

9.3: swap the value of p and p1 using temporary variables

STEP 10: Display the list

STEP 11: Stop

PROGRAM 2.3-CODING:

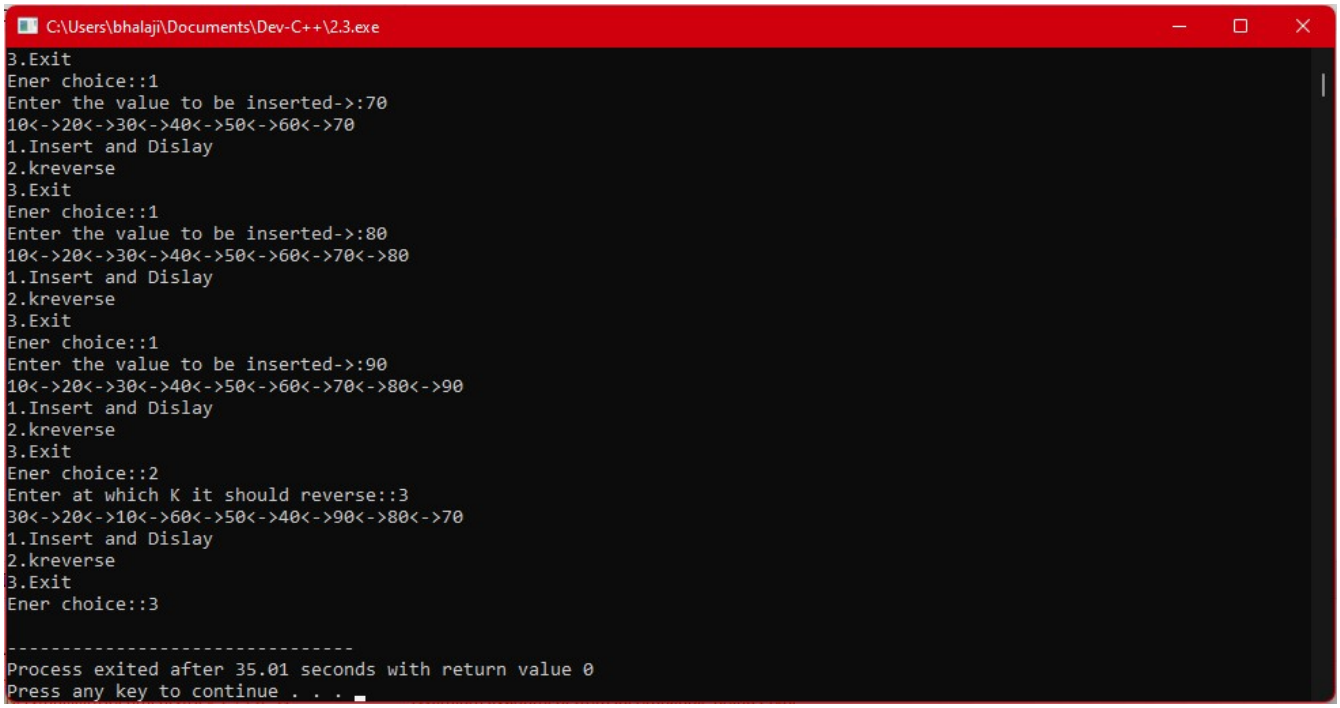
```
/******REVERSE THE ELEMENTS IN K-GROUPS-DOUBLY LINKED LIST******/
#include<stdio.h>
#include<stdlib.h>

struct Node
{
        int data;
        struct Node *next;
        struct Node *prev;
}*head=NULL;
```

```
struct Node *newnode;
struct Node *reverse (struct Node *head, int k)
{
	if (!head)
		return NULL;

	head->prev=NULL;
	struct Node *curr = head;
	struct Node *temp;
	struct Node *newhead;
	int count = 0;

	while (curr != NULL && count < k)
	{
		newhead=curr;
		temp=curr->prev;
		curr->prev=curr->next;
		curr->next=temp;
		curr=curr->prev;
		count++;
	}
	if(count>=k)
	{
		struct node *rest=reverse (curr,k);
		head->next=rest;
	}
	return newhead;
}
void insertlast(struct Node *Newnode)
{
	int data;
	struct Node *p;
	Newnode=(struct Node*)malloc(sizeof(struct Node));
	printf("Enter the value to be inserted->:");
```

```
        scanf("%d",&data);
        Newnode->data=data;
        Newnode->next=NULL;
        if(head==NULL)
        {
                head=Newnode;
        }
        else
        {
                p=head;
                while(p->next!=NULL)
                    p=p->next;
                p->next=Newnode;
                Newnode->prev=p;
        }
}
void display(struct Node *node)
{
        while(node->next!=NULL)
        {
                printf("%d<->",node->data);
                node=node->next;
        }
   printf("%d",node->data);
}
int main(void)
{
   int ch,n;
        while(1)
        {
                printf("\n1.Insert and Dislay\n2.kreverse\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
```

```
                        case (1):
                                insertlast(newnode);
                                display(head);
                                break;
                        case (2):
                                printf("Enter at which K it should reverse::");
                                scanf("%d",&n);
                                head = reverse (head, n);
                                display(head);
                                break;
                        case (3):
                                exit(0);
                }
        }
        return(0);
}
```

PROGRAM 2.3-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on doubly linked list and operations are executed and outputs are verified.

EXP NO: 03

DATE:07.09.2022

APPLICATIONS OF LINKED LIST

AIM:

To write c programs to implement applications of linked list and its operations.

PROGRAM 3.1: POLYNOMIAL SUBTRACTION USING LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create a structure using the struct Node i.e.,int data,stuct Node*next.

STEP 3: Get the value to be stored in the node ; store it in the data field.

using the my_create_poly(my_poly **) function.Push the values in the linked list

STEP 4: while(poly1 && poly2) then check if (poly1->pow > poly2->pow) then do

tmp_node->pow = poly1->pow,tmp_node->coeff = poly1->coeff and poly1 = poly1->next;

4.1:else if (poly1->pow < poly2->pow) then do tmp_node->pow = poly2->pow,

tmp_node->coeff = poly2->coeff and poly2 = poly2->next.

4.2: else do tmp_node->pow = poly1->pow,tmp_node->coeff = poly1->coeff - poly2->coeff;

poly1 = poly1->next,poly2 = poly2->next;

4.3:if(poly1 && poly2) then do tmp_node->next = (my_poly *) malloc(sizeof(my_poly));

tmp_node = tmp_node->next,tmp_node->next = NULL.

4.4:while(poly1 || poly2) then do tmp_node->next = (my_poly *) malloc(sizeof(my_poly))

tmp_node = tmp_node->nex,tmp_node->next = NULL;

4.5:if(poly1) then do tmp_node->pow = poly1->pow,tmp_node->coeff = poly1->coeff;

poly1 = poly1->next and if(poly2) then do tmp_node->pow = poly2->pow,

tmp_node->coeff = poly2->coeff,poly2 = poly2->next.

STEP 5:using display function display the result.

STEP 6: Stop.

PROGRAM 3.1-CODING:

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
   int coeff;
   int pow;
   struct Node* next;
};
// Function to create new node
struct node* create_node(int x, int y, struct Node **temp)
{
   struct Node *r, *z;
   z = *temp;
   if (z == NULL)
         {
      r = (struct Node*)malloc(sizeof(struct Node));
      r->coeff = x;
      r->pow = y;
      *temp = r;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
   else
         {
      r->coeff = x;
      r->pow = y;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
}
// Function Adding two polynomial numbers
void polysub(struct Node* poly1, struct Node* poly2,struct Node* poly)
```

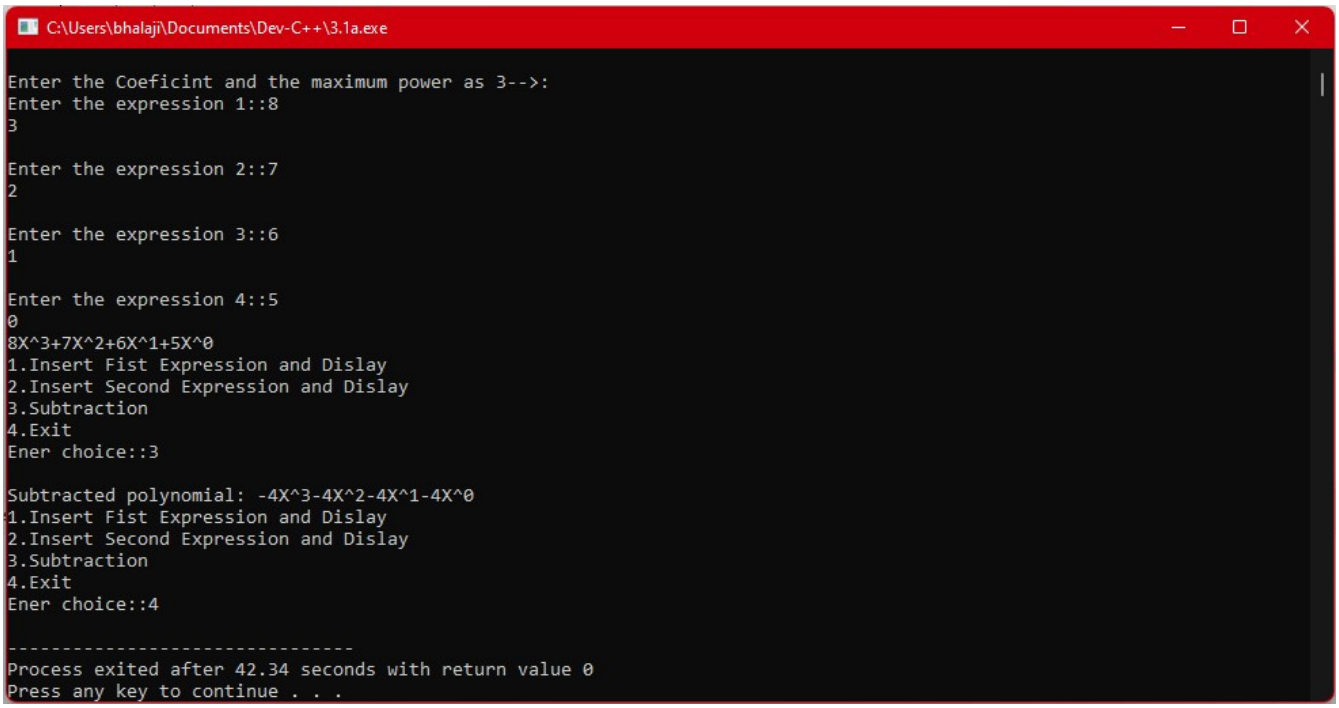
```
{
   while (poly1->next && poly2->next) {
      if (poly1->pow > poly2->pow) {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff;
         poly1 = poly1->next;
      }

      else if (poly1->pow < poly2->pow) {
         poly->pow = poly2->pow;
         poly->coeff = poly2->coeff;
         poly2 = poly2->next;
      }
      else {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff - poly2->coeff;
         poly1 = poly1->next;
         poly2 = poly2->next;
      }

      // Dynamically create new node
      poly->next= (struct Node*)malloc(sizeof(struct Node));
      poly = poly->next;
      poly->next = NULL;
   }
}
// Display Linked list
void show(struct Node* node)
{
   while (node->next != NULL)
         {
      printf("%dX^%d", node->coeff, node->pow);
      node = node->next;
      if (node->coeff >= 0)
```

```
                {
        if (node->next != NULL)
            printf("+");
      }
   }
}

// Driver code
int main()
{
   struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;
   int ch,a,b,c,d,e,f,g,h;
        while(1)
        {
                printf("\nEnter the choice\n1.Insert Fist Expression and Dislay\n2.Insert Second Expression and Dislay\n3.Addition\n4.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                printf("\nEnter the Coeficint and the maximum power as 3-->:");
                                printf("\nEnter the expression 1::");
                                scanf("%d",&a);
                                scanf("%d",&b);
                                printf("\nEnter the expression 2::");
                                scanf("%d",&c);
                                scanf("%d",&d);
                                printf("\nEnter the expression 3::");
                                scanf("%d",&e);
                                scanf("%d",&f);
                                printf("\nEnter the expression 4::");
                                scanf("%d",&g);
                                scanf("%d",&h);
                                create_node(a, b, &poly1);
```

```
                create_node(c, d, &poly1);
                create_node(e, f, &poly1);
                create_node(g, h, &poly1);
                show(poly1);
                break;
        case (2):
                printf("\nEnter the Coeficint and the maximum power as 3-->:");
                printf("\nEnter the expression 1::");
                 scanf("%d",&a);
                scanf("%d",&b);
                printf("\nEnter the expression 2::");
                scanf("%d",&c);
                scanf("%d",&d);
                printf("\nEnter the expression 3::");
                scanf("%d",&e);
                scanf("%d",&f);
                printf("\nEnter the expression 4::");
                scanf("%d",&g);
                scanf("%d",&h);
                create_node(a, b, &poly2);
                create_node(c, d, &poly2);
                create_node(e, f, &poly2);
                create_node(g, h, &poly2);
                show(poly2);
                break;
        case (3):
                poly = (struct Node*)malloc(sizeof(struct Node));
                polysub(poly1, poly2, poly);
                printf("\nSubtracted polynomial: ");
                show(poly);
                break;
        case (4):
                exit(0);
    }
```

```
        }
   return 0;
}
```

PROGRAM 3.1-OUTPUT:



PROGRAM 3.2: CREATE A STUDENT LIST USING LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create the structure node with roll_no,name and the address of the next node.

STEP 3: Get the information to be added to the list.

STEP 4: For inserting the element follow the step 5- 5.2 .

STEP 5:if head==NULL then head=newnode

5.1: Else p=head then using the while loop p->next!=NULL

5.2:trversing using p=p->next then p=p->next

STEP 6: For displaying the function follow step6

6.1: Using the while loop p->next!=NULL

6.2: Print the roll_no and name

STEP 7: Stop.

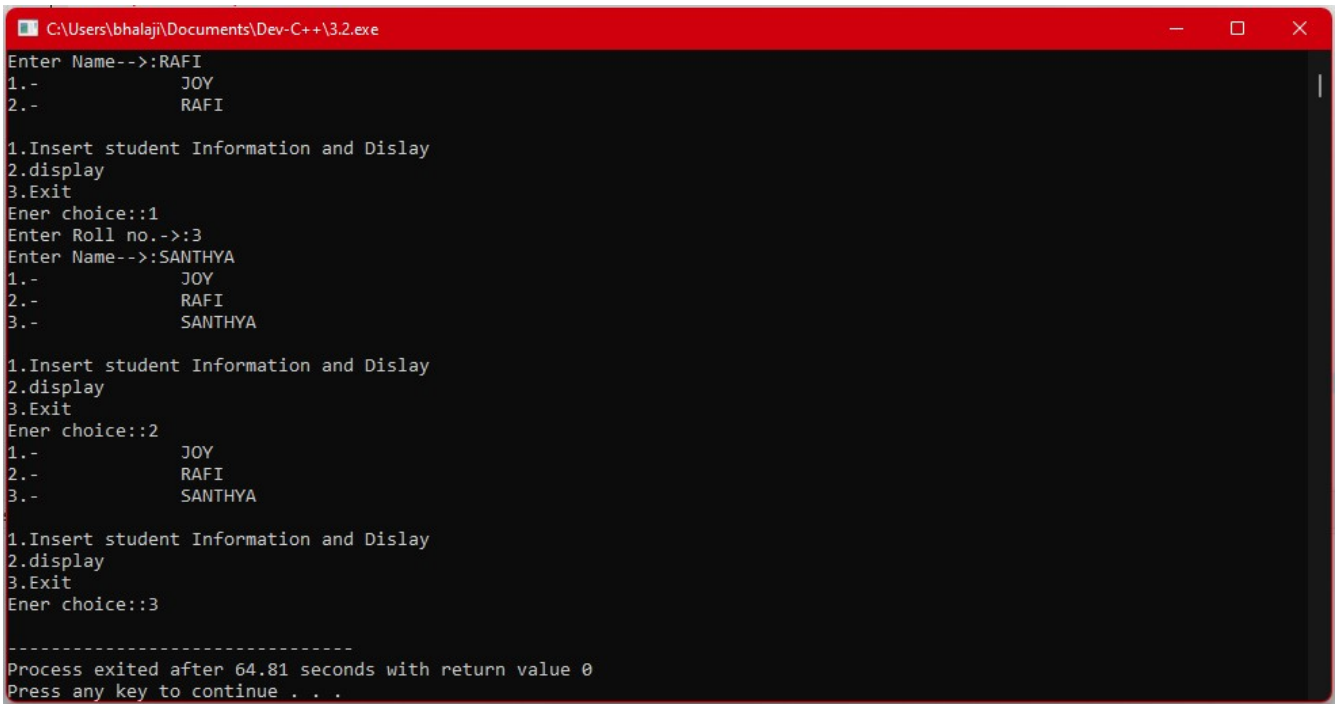
PROGRAM 3.2-CODING:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
struct node
{
        int roll_no;
        char name[20];
        struct node *next;
}*head=NULL;
void insertinformation();
void display();
int main()
{
        int ch;
        while(1)
        {
                printf("\n1.Insert student Information and Dislay\n2.display\n3.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                insertinformation();
                                display();
                                break;
                        case (2):
                                display();
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
```

```
}
void insertinformation()
{
        int data;
        char nname[20];
        struct node *newnode,*p;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter Roll no.->:");
        scanf("%d",&data);
        printf("Enter Name-->:");
        scanf("%s",&nname);
        newnode->roll_no=data;
        strcpy(newnode->name,nname);
        newnode->next=NULL;
        if(head==NULL)
        {
                head=newnode;
        }
        else
        {
                p=head;
                while(p->next!=NULL)
                  p=p->next;
                p->next=newnode;
        }
}
void display()
{
        struct node *p;
        p=head;
        while(p->next!=NULL)
        {
                printf("%d.-\t\t%s\n",p->roll_no,p->name);
                p=p->next;
```

```
        }
    printf("%d.-\t\t%s\n",p->roll_no,p->name);
}
```

PROGRAM 3.2-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on applications of linked list are executed and outputs are verified.

EXP NO: 04

DATE:15.09.2022

IMPLEMENTATION OF STACK ADT

AIM:

To write c programs to implement stack adt and its operations.

PROGRAM 4.1.1: IMPLIMENTATION OF STACK USING ARRAY

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Insert and display.
- •Delete.
- •Display first element.
- •Exit.

STEP 5: To insert first and display by using the push() operation.

5.1: Get the data to be inserted.

5.2: Then assign the value of data to stack[top++].

STEP 6: To delete by using pop() operation.

6.1: Top is decremented[top--].

STEP 7: To display first element by peek() opertion.

7.1: Print the value of first element(stack[top]).

STEP 8: To display the choices by using display() operation.

8.1: Get a variable i.

8.2: By using for loop:

```
for(i=top;i>=0;i--)
        {
          printf("%d\t",stack[i]);
          }
```

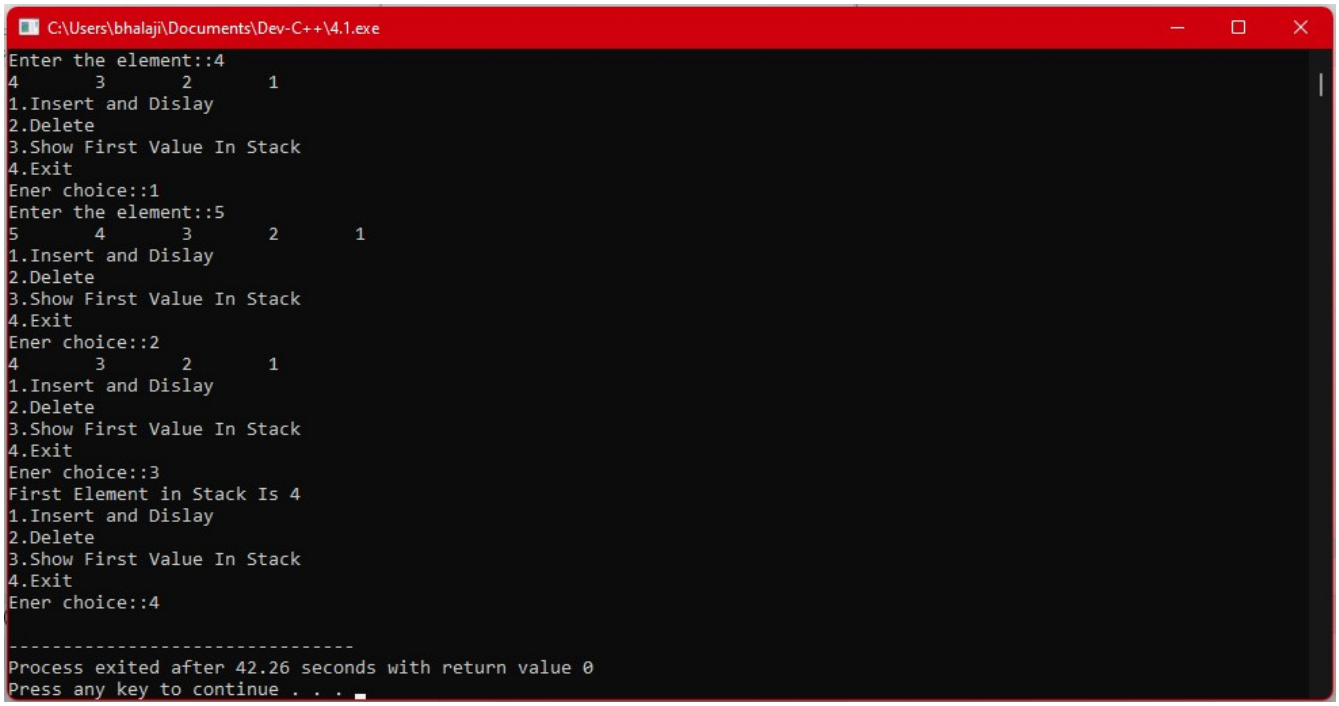
STEP 9: Stop.

PROGRAM 4.1.1-CODING:

```
/*******Implimentation Of Stack Using Array*******/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int stack[100];
int top=-1;
int main()
{
	int ch;
	while(1)
	{
		printf("\n1.Insert and Dislay\n2.Delete\n3.Show First Value In Stack\n4.Exit\nEner choice::");
		scanf("%d",&ch);
		switch(ch)
		{
			case (1):
				push();
				display();
				break;
			case (2):
				pop();
				display();
				break;
			case (3):
				peek();
				break;
			default:
				exit(0);
		}
	}
	return 0;
}
```

```
int push()
{
	int data;
	printf("Enter the element::");
	scanf("%d",&data);
	top++;
	stack[top]=data;
}
void pop()
{
	top--;
}
void display()
{
	int i;
	for(i=top;i>=0;i--)
	{
		printf("%d\t",stack[i]);
	}
}
void peek()
{
	printf("First Element in Stack Is %d",stack[top]);
}
```

PROGRAM 4.1.1-OUTPUT:



PROGRAM 4.1.2: IMPLIMENTATION OF STACK USING LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Insert and display.
- •Delete.
- •Display first element.
- •Exit.

STEP 5: To insert first and display by using the push() operation.

5.1: Get the data to be inserted.

5.2: Then assign the value of data to stack[top++].

STEP 6: To delete by using pop() operation.

6.1: Top is decremented[top--].

STEP 7: To display first element by peek() opertion.

7.1: Print the value of first element(stack[top]).

STEP 8: To display the choices by using display() operation.

8.1: Get a variable i.

8.2: By using for loop:

```
for(i=top;i>=0;i--)
        {
         printf("%d\t",stack[i]);
        }
```

STEP 9: Stop.

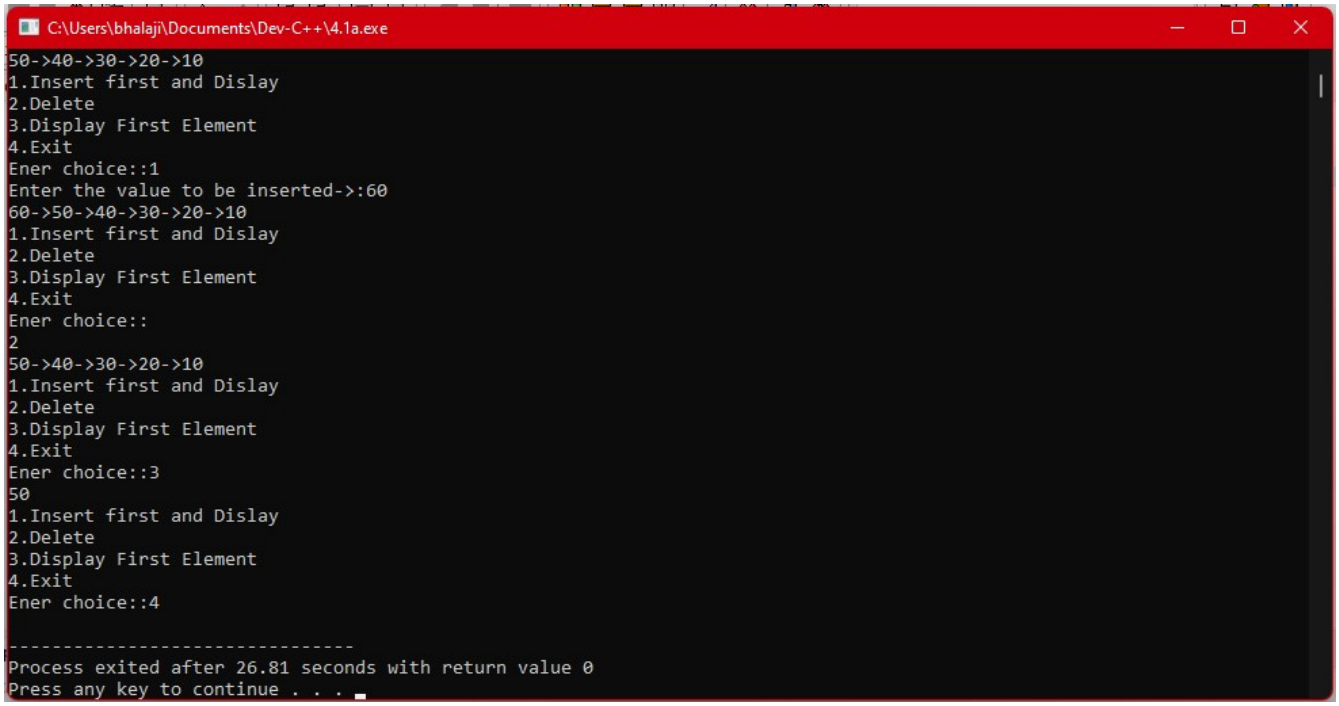
PROGRAM 4.1.2-CODING:

```
/*******Implimentation Of Stack Using Linked List*******/
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int value;
        struct node *next;
}*head=NULL;
void push();
void pop();
void display();
int main()
{
        int ch;
        while(1)
        {
                printf("\n1.Insert first and Dislay\n2.Delete\n3.Display First Element\n4.Exit\nEner
choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                push();
                                display();
                                break;
                        case (2):
```

```
                        pop();
                        display();
                        break;
                case (3):
                        peek();
                        break;
                default:
                        exit(0);
                }
        }
        return 0;
}
void push()
{
        int data;
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter the value to be inserted->:");
        scanf("%d",&data);
        newnode->value=data;
        newnode->next=NULL;
        if(head==NULL)
        {
                head=newnode;
        }
        else
        {
                newnode->next=head;
                head=newnode;
        }
}
void pop()
{
        struct node *temp;
```

```
        if(head==NULL)
          printf("There isno node to display");
        else
        {
                temp=head;
                head=temp->next;
                free(temp);
        }
}
void peek()
{
        struct node *temp;
        temp=head;
        printf("%d",temp->value);
}
void display()
{
        struct node *p;
        p=head;
        while(p->next!=NULL)
        {
                printf("%d->",p->value);
                p=p->next;
        }
    printf("%d",p->value);
}
```

PROGRAM 4.1.2-OUTPUT:



PROGRAM 4.2: Tom is a string freek. He has got sequences of n words to manipulate. If in a sequence, two same words come together then hell destroy each other. He wants to know the number of words left in the sequence after this pairwise destruction.

Input:

5

V[ ]={“ab”,”aa”,”aa”,”bcd”,”ab” }

Output:

3

ALGORITHM:

STEP 1: Start

STEP 2: Create array and stack as global variable.

STEP 3: The fuctions used hear are

Creating the array

Displaying the array

Creating and push the element to the stack.

STEP 4: Creating the element

STEP 5: Get the number of element in the array

5.1: Print the arry element

STEP 6:using the for loop(

6.1: for (i = 0; i < size; i++)then if (array[i][0]== '$') then break

6.2:for (j = i+1; j < size; j++) then len1=strlen(array[i]) then len2=strlen(array[j]) then for (k = 0; k < len1; k++) then found=0;

6.3:for (l = 0; l < len2; l++) then if (array[i][k]==array[j][l]) assign found=1 then break

if (found!) then break if(found==1) then array[j][0]= '$';

6.4:for (i = 0; i < size; i++) then if(array[i][0] != '$') then continue fin the len1=strlen(array[i]);

for (j = 0; j < len1; j++) then stack[i][j]=array[i][j] then top+=1 then stack[i][j]='\0'.

STEP8: Print the no of non repeated elements in the stack.

STEP 7: Stop.

PROGRAM 4.2-CODING:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
char array[10][20],stack[10][20];
int size,top=-1;
void arrayz()
{
   int i;
   char data[20];
   printf("Enter the Size of the Array:");
   scanf("%d",&size);
   printf("Enter Elements\n");
   for ( i = 0; i < size; i++)
   {
      scanf("%s",array[i]);
   }
}
void arrdisplay()
{
   int i,j;
   printf(" Elements Are\n");
   for ( i = 0; i < size; i++)
```

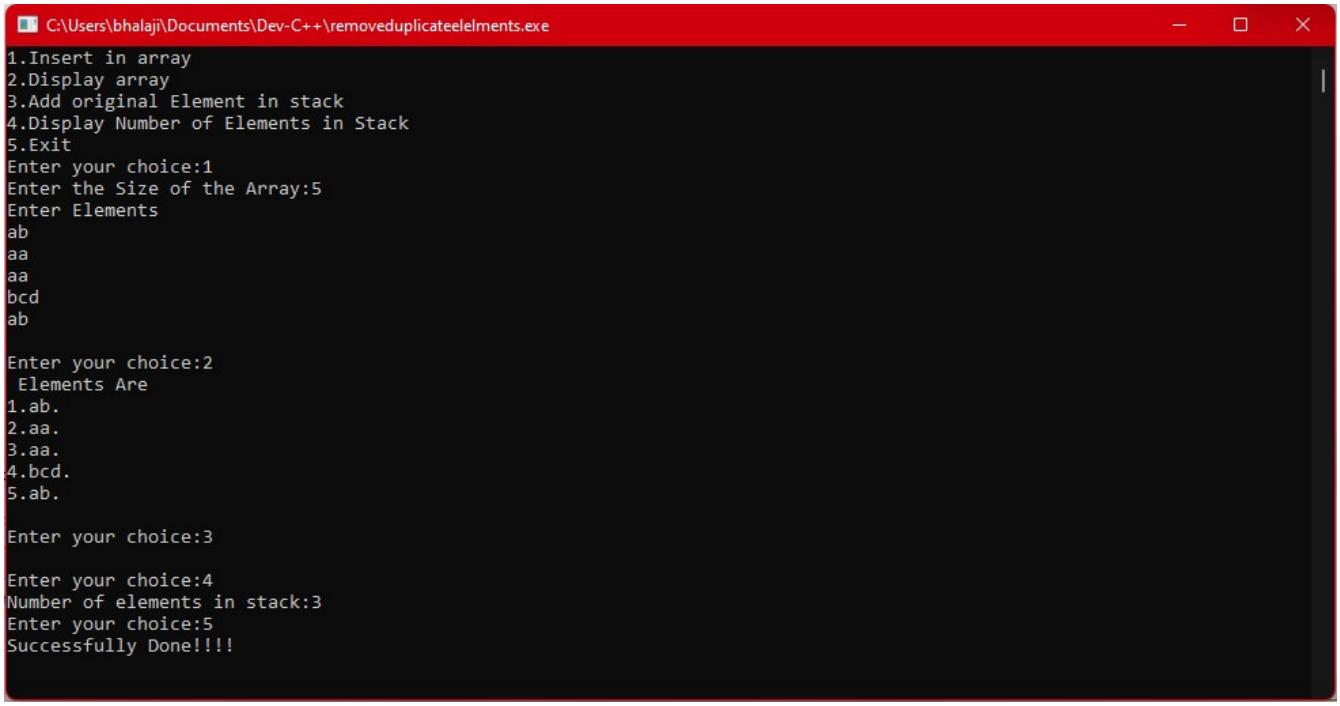
```
    {
        printf("%d.",i+1);
        for ( j = 0; array[i][j]!='\0'; j++)
        {
            printf("%c",array[i][j]);
        }
        printf(".\n");
    }
}
void stacky()
{
    int i,j,k,l,len2,len1,found=0;
    for ( i = 0; i < size; i++)
    {
        if (array[i][0]== '$')
        {
            break;
        }
        for ( j = i+1; j < size; j++)
        {
            len1=strlen(array[i]);
            len2=strlen(array[j]);
            for ( k = 0; k < len1; k++)
            {
                found=0;
                for (l = 0; l < len2; l++)
                {
                    if (array[i][k]==array[j][l])
                    {
                        found=1;
                        break;
                    }

                }
```

```
            if (found!=1)
            {
               break;
            }
         }
         if(found==1)
         {
            array[j][0]= '$';
         }
      }
   }
   for ( i = 0; i < size; i++)
   {
      if(array[i][0] != '$')
      {
         continue;
      }
      len1=strlen(array[i]);
      for ( j = 0; j < len1; j++)
      {
         stack[i][j]=array[i][j];
         top+=1;
      }
      stack[i][j]='\0';
   }
}
void stdisplay()
{
   int count=0,i;

   for (i=0; i<top; i++)
   {
      count+=1;
   }
```

```
   printf("Number of elements in stack:%d",count);
}
void main()
{
   int c;
   printf("1.Insert in array \n2.Display array \n3.Add original Element in stack \n4.Display Number of Elements in Stack\n5.Exit");
   do
   {
      printf("\nEnter your choice:");
      scanf("%d",&c);
      switch (c)
      {
         case 1:
         {
            arrayz();
            break;
         }
         case 2:
         {
            arrdisplay();
            break;
         }
         case 3:
         {
            stacky();
            break;
         }
         case 4:
         {
            stdisplay();
            break;
         }
         default:
```

```
            break;
        }
    }while(c<=4);
    printf("Successfully Done!!!!");
    getch();
}
```

PROGRAM 4.2-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on implementation of stack ADT are executed and outputs are verified.

EXP NO: 05

DATE:22.09.2022

IMPLEMENTATION OF QUEUE ADT

AIM:

To write c programs to implement queue ADT and its operations.

PROGRAM 5.1.1: IMPLIMENTATION OF QUEUE USING ARRAY

ALGORITHM:

STEP 1: Start

STEP 2: Initialize the global variable Queue,front=0 and rear=-1.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Insert and display.
- •Delete.
- •Display first element.
- •Exit.

STEP 5: In main fuction get the number of elements in the Queue .

5.1 To insert first and display by using the enqueue() operation.

5.2: Get the data to be inserted.

5.3: Then assign the value of data to Queue[rear++].

STEP 6: To delete by using dequeue() operation.

6.1: Top is decremented[front--].

STEP 7: To display first element by peek() operation.

7.1: Print the value of first element(Queue[front]).

STEP 8: To display the choices by using display() operation.

8.1: Get a variable i.

8.2: By using for loop:

for(i=top;i>=0;i--) the printf("%d\t",Queue[i]);

STEP 9: Stop.

PROGRAM 5.1.1-CODING:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void dequeue();
void peek();
void display();
int Q[100];
int front=0;
int rear=-1;
int main()
{
        int ch,n;
        printf("\nEnter the number of elements in Queue-->:");
   scanf("%d",&n);
        while(1)
        {
                printf("\n1.Insert and Dislay\n2.Delete\n3.Show First Value In Stack\n4.Exit\nEner
choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                enqueue(n);
                                display();
                                break;
                        case (2):
                                dequeue();
                                display();
                                break;
                        case (3):
                                peek();
                                //display();
                                break;
```

```
                        default:
                                exit(0);
                }
        }
        return 0;
}
int enqueue(int n)
{
        int data;
        printf("Enter Element-->:");
        scanf("%d",&data);
        if(rear==n-1)
        {
            printf("\nQueue is FULL");
    }
   else
   {
        rear++;
        Q[rear]=data;
   }
}
void dequeue()
{
    printf("%d",Q[front]);
         front=front+1;
}
void display()
{
        int i;
        printf("\nThe elements are-->:\n");
        for(i=front;i<=rear;i++)
        {
                printf("%d\t",Q[i]);
        }
```

```
}
void peek()
{
        printf("\nFirst Element in Stack Is %d",Q[front]);
}
```

PROGRAM 5.1.1-OUTPUT:

```
C:\Users\bhalaji\Documents\Dev-C++\5.1.1.exe

Enter the number of elements in Queue-->:5

1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::1
Enter Element-->:1

The elements are-->:
1
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::1
Enter Element-->:2

The elements are-->:
1       2
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::1
Enter Element-->:3

The elements are-->:
1       2       3
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::1
Enter Element-->:4

The elements are-->:
1       2       3       4
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::2
1
The elements are-->:
2       3       4
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::3

First Element in Stack Is 2
1.Insert and Dislay
2.Delete
3.Show First Value In Stack
4.Exit
Ener choice::4

--------------------------------
Process exited after 25.31 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 5.1.2: IMPLIMENTATION OF QUEUE USING LINKED LIST

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Insert and display.
- •Delete.
- •Display first element.
- •Exit.

STEP 5: To insert first and display by using the enqueue() operation.

5.1: Get the data to be inserted.

5.2: Then assign the value of data to stack[front++].

5.2:if(front==NULL&&rear==NULL)then front=newnode and rear=newnode else follow next step.

5.3:rear->next=newnode and rear=newnode;

STEP 6: To delete by using dequeue() operation.

6.1:if(front==NULL&&rear==NULL) then printf("There isno node to delete") else follow the next step.

6.2:temp=front then front=temp->next and free(temp).

STEP 7: To display first element by peek() opertion.

7.1: Print the value of first element assign temp=front then print ("%d",temp->value).

STEP 8: To display the choices by using display() operation.

8.1:if(front==NULL&&rear==NULL) then printf("There isno node to display") else follow next step.

8.2: Assign p=front using while(p->next!=NULL) then printf("%d--",p->value) and increment p=p->next then to print last value printf("%d",p->value);

STEP 9: Stop.

PROGRAM 5.1.2-CODING:

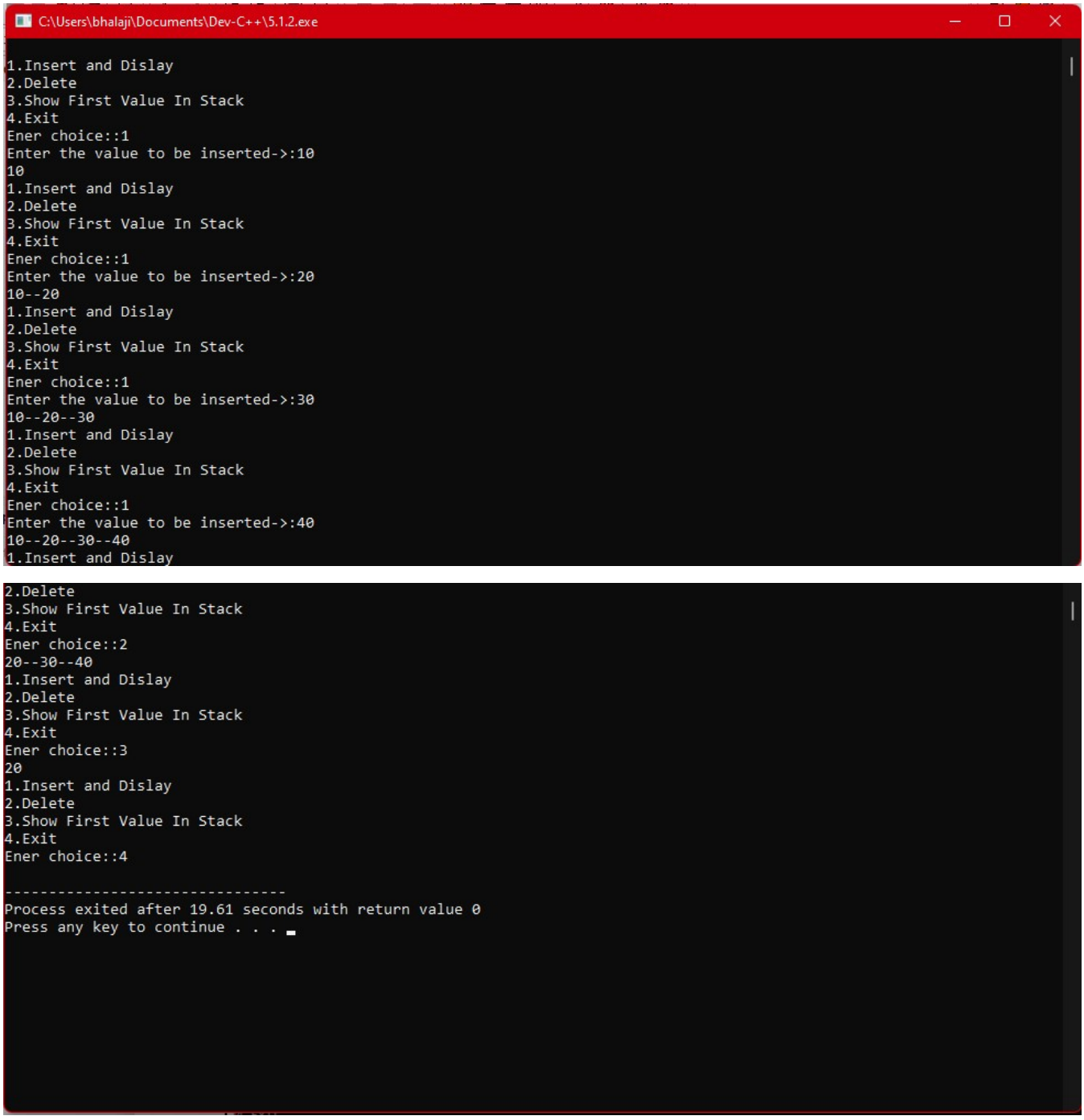
```
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int value;
```

```
        struct node *next;
};
struct node *front=NULL,*rear=NULL;
void dequeue();
void enqueue();
void peek();
void display();
int main()
{
        int ch;
        while(1)
        {
                printf("\n1.Insert and Dislay\n2.Delete\n3.Show First Value In Stack\n4.Exit\nEner
choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                enqueue();
                                display();
                                break;
                        case (2):
                                dequeue();
                                display();
                                break;
                        case (3):
                                peek();
                                //display();
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
```

```
}
void enqueue()
{
        int data;
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        printf("Enter the value to be inserted->:");
        scanf("%d",&data);
        newnode->value=data;
        newnode->next=NULL;
        if(front==NULL&&rear==NULL)
        {
                front=newnode;
                rear=newnode;
        }
        else
        {
                rear->next=newnode;
                rear=newnode;
        }
}
void dequeue()
{
        struct node *temp;
        if(front==NULL&&rear==NULL)
          printf("There isno node to delete");
        else
        {
                temp=front;
                front=temp->next;
                free(temp);
        }

}
```

```
void peek()
{
        struct node *temp;
        if(front==NULL&&rear==NULL)
          printf("There isno node to display");
        temp=front;
        printf("%d",temp->value);
}
void display()
{
        struct node *p;
        //p=front;
        if(front==NULL&&rear==NULL)
          printf("There isno node to display");
       else
       {
           p=front;
           while(p->next!=NULL)
           {
                   printf("%d--",p->value);
                    p=p->next;
            }
       printf("%d",p->value);
   }
}
```

PROGRAM 5.1.2-OUTPUT:



PROGRAM 5.2: IMPLIMENTATION OF STACK USING QUEUE

ALGORITHM:

STEP 1: Start

STEP 2: Initialize the global variable Queue,front=0 and rear=-1.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Insert and display.
- •Delete.
- •Display first element.
- •Exit.

STEP 5: In main fuction get the number of elements in the Queue .

5.1 To insert first and display by using the push() operation.

5.2: Get the data to be inserted in the push fuction and pass the data using the argument.

5.3: Do the enqueue Operation.

STEP 6: To delete using pop() fuction call the dequeue operation

6.1: To delete by using dequeue() operation.

6.2: Top is decremented[front--].

STEP 7: To display first element by peek() operation.

7.1: Print the value of first element(Queue[front]).

STEP 8: To display the choices by using display() operation.

8.1: Get a variable i.

8.2: By using for loop:

```
for(i=top;i>=0;i--)
        {
         printf("%d\t",Queue[i]);
         }
```

STEP 9: Stop.

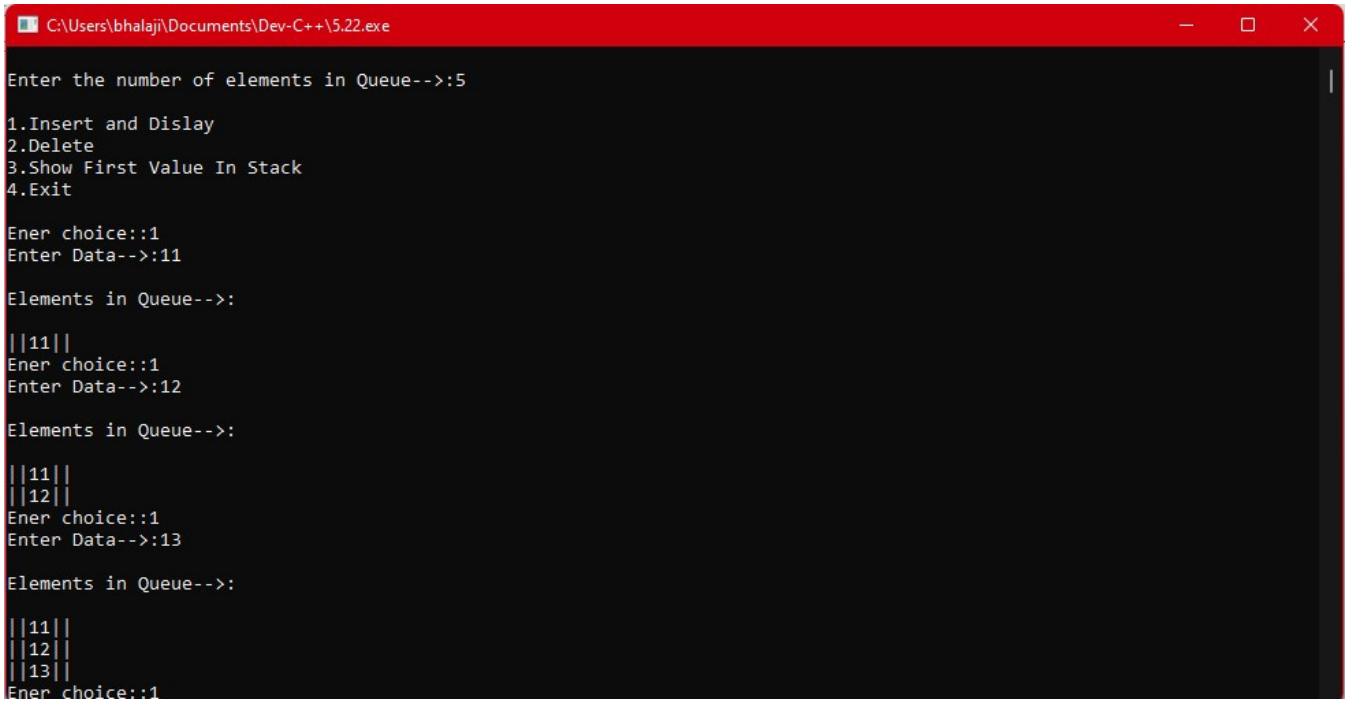
PROGRAM 5.2-CODING:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int Q[100];
int front=0;
int rear=-1;
int main()
{
        int ch,n;
        printf("\nEnter the number of elements in Queue-->:");
   scanf("%d",&n);
   printf("\n1.Insert and Dislay\n2.Delete\n3.Show First Value In Stack\n4.Exit\n");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                push(n);
                                display();
                                break;
                        case (2):
                                pop();
                                display();
                                break;
                        case (3):
                                peek();
                                break;
                        default:
                                exit(0);
                }
        }
```

```
	return 0;
}
void push(int n)
{
	int data,s;
	s=n;
	printf("Enter Data-->:");
	scanf("%d",&data);
	enqueue(data,s);
}
void enqueue(int data,int n)
{
	if(rear==n-1)
	{
		printf("Queue Is Full");
	}
	else
	{
		rear++;
		Q[rear]=data;
	}
}
void pop()
{
	dequeue();
}
void dequeue()
{
	printf("%d",Q[front]);
	front=front+1;
}
void peek()
{
	printf("First Element-->:||%d||",Q[front]);
```

```
}
void display()
{
        int i;
        printf("\nElements in Queue-->:\n");
        for(i=front;i<=rear;i++)
        {
                printf("\n||%d||",Q[i]);
        }
}
```

PROGRAM 5.2-OUTPUT:



```
Enter Data-->:14

Elements in Queue-->:

||11||
||12||
||13||
||14||
Ener choice::1
Enter Data-->:15

Elements in Queue-->:

||11||
||12||
||13||
||14||
||15||
Ener choice::2
11
Elements in Queue-->:

||12||
||13||
||14||
||15||
Ener choice::3
First Element-->:||12||
Ener choice::4
```

```
--------------------------------
Process exited after 19.55 seconds with return value 0
Press any key to continue . . .
```

DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on queue ADT are executed and outputs are verified.

EXP NO: 06

DATE:30.09.2022

APPLICATIONS OF STACK AND QUEUE

AIM:

To write c programs to applications of stack and queue.

PROGRAM 6.1: DECIMAL TO BINARY CONVERTOR USING STACK

ALGORITHM:

STEP 1: Start

STEP 2: Create object and allocate memory using malloc function.

STEP 3: Get the value to be stored in the stack; store it in the data field.

STEP 4: Get the choices

- •Binary to decimal convertor.
- •Exit.

STEP5: To convert to binary intialize int n,s and get the number in n variable to conver to change to binary

Using while condition while true follow next step

5**.1:**Store the n%2 in variable s and push the s in to the stack.

5**.2:**Using if loop if n==1 then push n and break the loop.

STEP 6: To do push(int data) operation.

6.1: Using the if condition if(head==NULL) then head=newnode else follow the next step.

6.2:newnode->next=head then head=newnode;

STEP 7: To do the next decimal we have to delete every node in the stack by using pop() operation.

7**.1:**Using the while condition while true follow the next steps.

7.2: Using if condition if(head==NULL) then break else follow the next step

7.3:temp=head then do head=temp->next and free(temp);

STEP 8: To display the choices by using display() operation.

8.1: Assign p=head then print The converted elements--> by using the while loop

while(p->next!=NULL) then printf("%d\t",p->value) and p=p->next after the while loop printf("%d",p->value);

STEP 9: Stop.

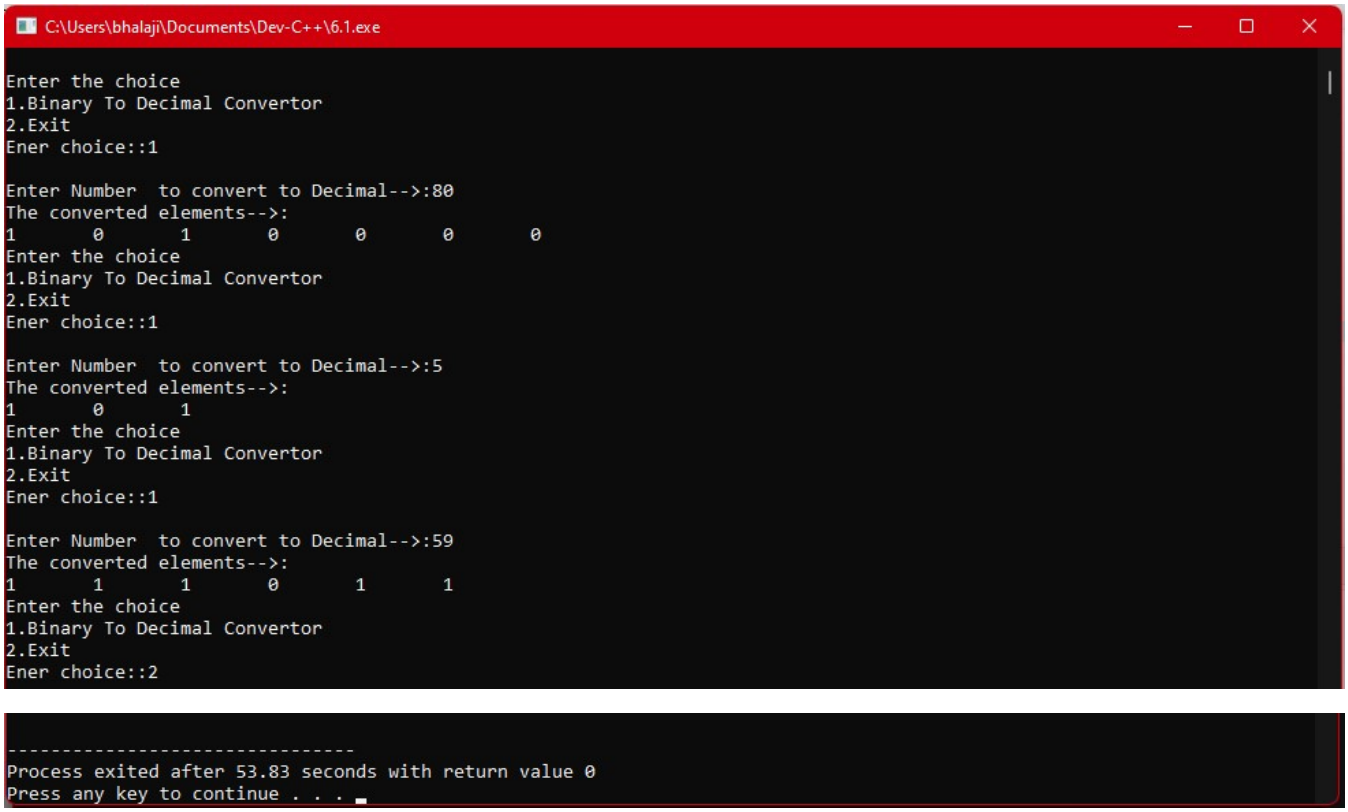
PROGRAM 6.1-CODING:

```
/******Decimal to binary convertor******/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int value;
        struct node *next;
}*head=NULL;
void bintodec();
void push(int data);
void display();
void pop();
int main()
{
        int ch;
        while(1)
        {
                printf("\nEnter the choice\n1.Binary To Decimal Convertor\n2.Exit\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                bintodec();
                                display();
                                pop();
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
}
```

```
void bintodec()
{
	int n,s,t;
	printf("\nEnter Number  to convert to Decimal-->:");
	scanf("%d",&n);
	while(1)
	{
		s=n%2;
		push(s);
		n=n/2;
		if(n==1)
		{
			push(n);
			break;
		}
	}
}
void push(int data)
{
	struct node *newnode;
	newnode=(struct node*)malloc(sizeof(struct node));
	newnode->value=data;
	newnode->next=NULL;
	if(head==NULL)
	{
		head=newnode;
	}
	else
	{
		newnode->next=head;
		head=newnode;
	}
}
```

```
void pop()
{
        struct node *temp;
        while(1)
        {
                if(head==NULL)
                {
                        break;
                 }
                else
                {
                        temp=head;
                        head=temp->next;
                        free(temp);
                }
        }
}
void display()
{
        struct node *p;
        p=head;
        printf("The converted elements-->:\n");
        while(p->next!=NULL)
        {
                printf("%d\t",p->value);
                p=p->next;
        }
    printf("%d",p->value);
}
```

PROGRAM 6.1-OUTPUT:



PROGRAM 6.2: VEHICLES IN TOLL USING QUEUE

ALGORITHM:

STEP 1: Start

STEP 2: Create the structure with veh_name[100] and next pointer assign the front and rear to NULL.

STEP 3: Create the object and allocate memory using malloc function.

STEP 4: Get the choices

- •Vehicle IN.
- •First Vehicle OUT.
- •Display first vehicle.
- •Display all Vehicle.
- •Exit.

STEP 5: To insert the vehicle into the Toll and display by using the Vehicle IN() operation.

5.1: Get the vehicle name to be inserted in the TOLL.

5.2:if(front==NULL&&rear==NULL)then front=newnode and rear=newnode else follow next step.

5.3:rear->next=newnode and rear=newnode;

STEP 6: To move the first vehicle out by using Vchicle OUT() operation.

6.1:if(front==NULL&&rear==NULL) then printf("There isno vechicle in TOLL") else follow the next step.

6.2:temp=front then front=temp->next and free(temp).

STEP 7: To display first vehicle by First_Veh() opertion.

7.1: Print the value of first element assign temp=front then print ("%s",temp->value).

STEP 8: To display the all the elements in the TOLL by using display() operation.

8.1:if(front==NULL&&rear==NULL) then printf("There is no vechicle in the TOLL") else follow next step.

8.2: Assign p=front using while(p->next!=NULL) then printf("%d--",p->value) and increment p=p->next then to print last value printf("%d",p->value);

STEP 9: Stop.

PROGRAM 6.2-CODING:

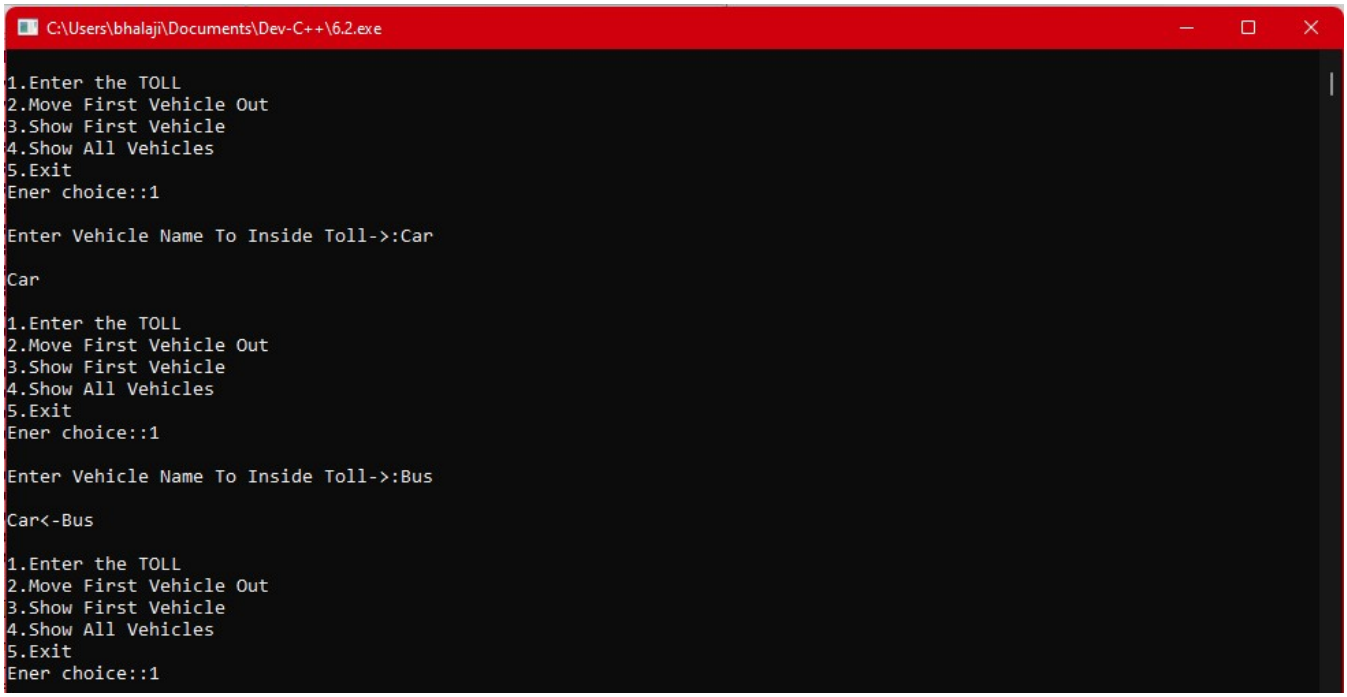
```
/******Vehicles in toll using Queue******/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>


struct node

{

        char veh[100];

        struct node *next;

};


struct node *front=NULL,*rear=NULL;


void Veh_IN();

void Veh_OUT();

void First_Veh();

void display();


int main()

{

        int ch;

        while(1)

        {
```

```
		printf("\n1.Enter the TOLL\n2.Move First Vehicle Out\n3.Show First Vehicle\n4.Show All Vehicles\n5.Exit\nEner choice::");
		scanf("%d",&ch);
		switch(ch)
		{
			case (1):
				Veh_IN();
				display();
				break;
			case (2):
				Veh_OUT();
				display();
				break;
			case (3):
				First_Veh();
				break;
			case (4):
				display_Veh();
				break;
			default:
				exit(0);
		}
	}
	return 0;
}

void Veh_IN()
{
	char data[100];
	struct node *newnode;
	newnode=(struct node*)malloc(sizeof(struct node));
	printf("\nEnter Vehicle Name To Inside Toll->:");
	scanf("%s",&data);
	strcpy(newnode->veh,data);
```

```
        newnode->next=NULL;
        if(front==NULL&&rear==NULL)
        {
                front=newnode;
                rear=newnode;
        }
        else
        {
                rear->next=newnode;
                rear=newnode;
        }
}

void Veh_OUT()
{
        struct node *temp;
        if(front==NULL&&rear==NULL)
          printf("There Is No Vehical In The TOLL");
        else
        {
                temp=front;
                front=temp->next;
                free(temp);
        }
}

void First_Veh()
{
        struct node *temp;
        if(front==NULL&&rear==NULL)
          printf("There Is No Vehical In The TOLL");
        temp=front;
        printf("%s",front->veh);
}
```

```
void display_Veh()
{
          struct node *p;
          if(front==NULL&&rear==NULL)
             printf("There Is No Vehical In The TOLL");
         else
        {
             p=front;
             printf("\n");
             while(p->next!=NULL)
             {
                       printf("%s<-",p->veh);
                       p=p->next;
              }
          printf("%s",p->veh);
          printf("\n");
    }
}
```

PROGRAM 6.2-OUTPUT:



```
Enter Vehicle Name To Inside Toll->:Lorry

Car<-Bus<-Lorry

1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
5.Exit
Ener choice::1

Enter Vehicle Name To Inside Toll->:Tractor

Car<-Bus<-Lorry<-Tractor

1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
5.Exit
Ener choice::1

Enter Vehicle Name To Inside Toll->:JCB

Car<-Bus<-Lorry<-Tractor<-JCB

1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
```

```
5.Exit
Ener choice::2

Bus<-Lorry<-Tractor<-JCB

1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
5.Exit
Ener choice::3
Bus
1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
5.Exit
Ener choice::4

Bus<-Lorry<-Tractor<-JCB

1.Enter the TOLL
2.Move First Vehicle Out
3.Show First Vehicle
4.Show All Vehicles
5.Exit
Ener choice::5

--------------------------------
Process exited after 153.7 seconds with return value 0
```

DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the two given programs based on applications of stack and queue are executed and outputs are verified.

EXP NO: 07

DATE:12.10.2022

BINARY SEARCH TREE

AIM:

To write c programs to implement binary search tree and its operations.

PROGRAM 7.1: BINARY TREE INSER4TION AND DELETION

ALGORITHM:

STEP 1: Start

STEP 2:for Insertion

```
IF TREE = NULL
    Allocate memory for TREE
    SET TREE DATA = VAL
    SET TREE LEFT = TREE RIGHT = NULL
ELSE
    IF VAL < TREE DATA
        Insert(TREE LEFT, VAL)
    ELSE
        Insert(TREE RIGHT, VAL)
        [END OF IF]
[END OF IF]
```

STEP 3: for deletion

```
IF TREE = NULL
    Write "VAL not found in the tree"
ELSE IF VAL < TREE DATA
    Delete(TREE->LEFT, VAL)
ELSE IF VAL > TREE DATA
    Delete(TREE RIGHT, VAL)
ELSE IF TREE LEFT AND TREE RIGHT
    SET TEMP = findLargestNode(TREE LEFT)
    SET TREE DATA = TEMP DATA
    Delete(TREE LEFT, TEMP DATA)
```

```
        ELSE
            SET TEMP = TREE
            IF TREE LEFT = NULL AND TREE RIGHT = NULL
                SET TREE = NULL
            ELSE IF TREE LEFT != NULL
                 SET TREE = TREE LEFT
            ELSE
                 SET TREE = TREE RIGHT
            [END OF IF]
            FREE TEMP
        [END OF IF]
```

STEP 4: Stop.

PROGRAM 7.1-CODING:

```
/******Binary InserT and Delete******/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
        int data;
        struct node *right;
        struct node *left;
}*right=NULL,*left=NULL;

//struct node *Tr=NULL;
struct node *tree;
void create_tree(struct node *);
struct node *insertElement(struct node *, int);
struct node *deleteElement(struct node *, int);
void inorderTraversal(struct node *);

int main()
{
```

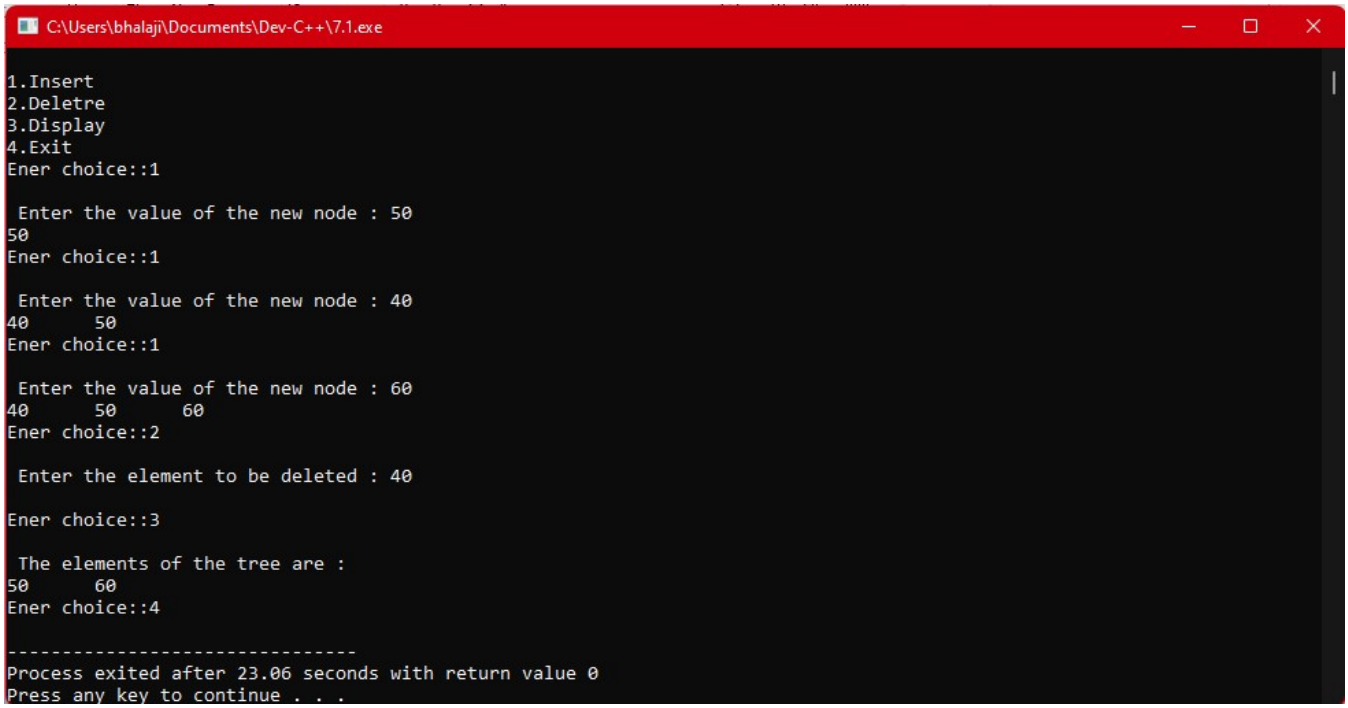
```
        //struct node *T;
        int ch,val;
        printf("\n1.Insert\n2.Deletre\n3.Display\n4.Exit");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                printf("\n Enter the value of the new node : ");
                                scanf("%d", &val);
                                tree = insertElement(tree, val);
                                inorderTraversal(tree);
                                break;
                        case (2):
                                printf("\n Enter the element to be deleted : ");
                                scanf("%d", &val);
                                tree = deleteElement(tree, val);
                                break;
                        case (3):
                                printf("\n The elements of the tree are : \n");
                                inorderTraversal(tree);
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
}
void create_tree(struct node *tree)
{
        tree = NULL;
}
```

```
struct node *insertElement(struct node *tree, int val)
{
        struct node *ptr, *nodeptr, *parentptr;
        ptr = (struct node*)malloc(sizeof(struct node));
        ptr->data= val;
        ptr->left= NULL;
        ptr->right = NULL;
        if(tree==NULL)
        {
                tree=ptr;
                tree->left=NULL;
                tree->right=NULL;
        }
        else
        {
                parentptr=NULL;
                nodeptr=tree;
                while(nodeptr!=NULL)
                {
                        parentptr=nodeptr;
                        if(val<nodeptr->data)
                           nodeptr=nodeptr->left;
                        else
                           nodeptr = nodeptr->right;
                }
                if(val<parentptr->data)
                   parentptr->left = ptr;
                else
                   parentptr->right = ptr;
        }
        return tree;
}
```

```
struct node *deleteElement(struct node *tree, int val)
{
        struct node *cur, *parent, *suc, *psuc, *ptr;
        if(tree->left==NULL)
        {
                printf("\n The tree is empty ");
                return(tree);
        }
        parent = tree;
        cur = tree->left;
        while(cur!=NULL && val!= cur->data)
        {
                parent = cur;
                cur = (val<cur->data)? cur->left:cur->right;
        }
        if(cur == NULL)
        {
                printf("\n The value to be deleted is not present in the tree");
                return(tree);
        }
        if(cur->left == NULL)
           ptr = cur->right;
        else if(cur->right == NULL)
           ptr = cur->left;
        else
        {
                // Find the in–order successor and its parent
                psuc = cur;
                cur = cur->left;
                while(suc->left!=NULL)
                {
                        psuc = suc;
                        suc = suc->left;
                }
```

```
                if(cur==psuc)
                {
                        // Situation 1
                        suc->left = cur->right;
                }
                else
                {
                        // Situation 2
                        suc->left = cur->left;
                        psuc->left = suc->right;
                        suc->right = cur->right;
                }
                ptr = suc;
        }
        // Attach ptr to the parent node
        if(parent->left == cur)
           parent->left=ptr;
        else
           parent->right=ptr;
        free(cur);
        return tree;
}
void inorderTraversal(struct node *tree)
{
        if(tree != NULL)
        {
                inorderTraversal(tree->left);
                printf("%d\t", tree->data);
                inorderTraversal(tree->right);
        }
}
```

PROGRAM 7.1-OUTPUT:



PROGRAM 7.2: TRAVERSING BINARY SEARCH TREE

ALGORITHM:

STEP 1: Start

STEP 2: Repeat Steps 2 to 4 while TREE != NULL

STEP 3: Write TREE DATA

STEP 4: PREORDER(TREE LEFT)

STEP 5: PREORDER(TREE RIGHT)

[END OF LOOP]

STEP 6: Repeat Steps 2 to 4 while TREE != NULL

STEP 7: INORDER(TREE LEFT)

STEP 8: Write TREE DATA

STEP 9: INORDER(TREE RIGHT)

[END OF LOOP]

STEP 10: Repeat Steps 2 to 4 while TREE != NULL

STEP 11: POSTORDER(TREE LEFT)

STEP 12: POSTORDER(TREE RIGHT)

STEP 13: Write TREE DATA

[END OF LOOP]

STEP 14: Stop.

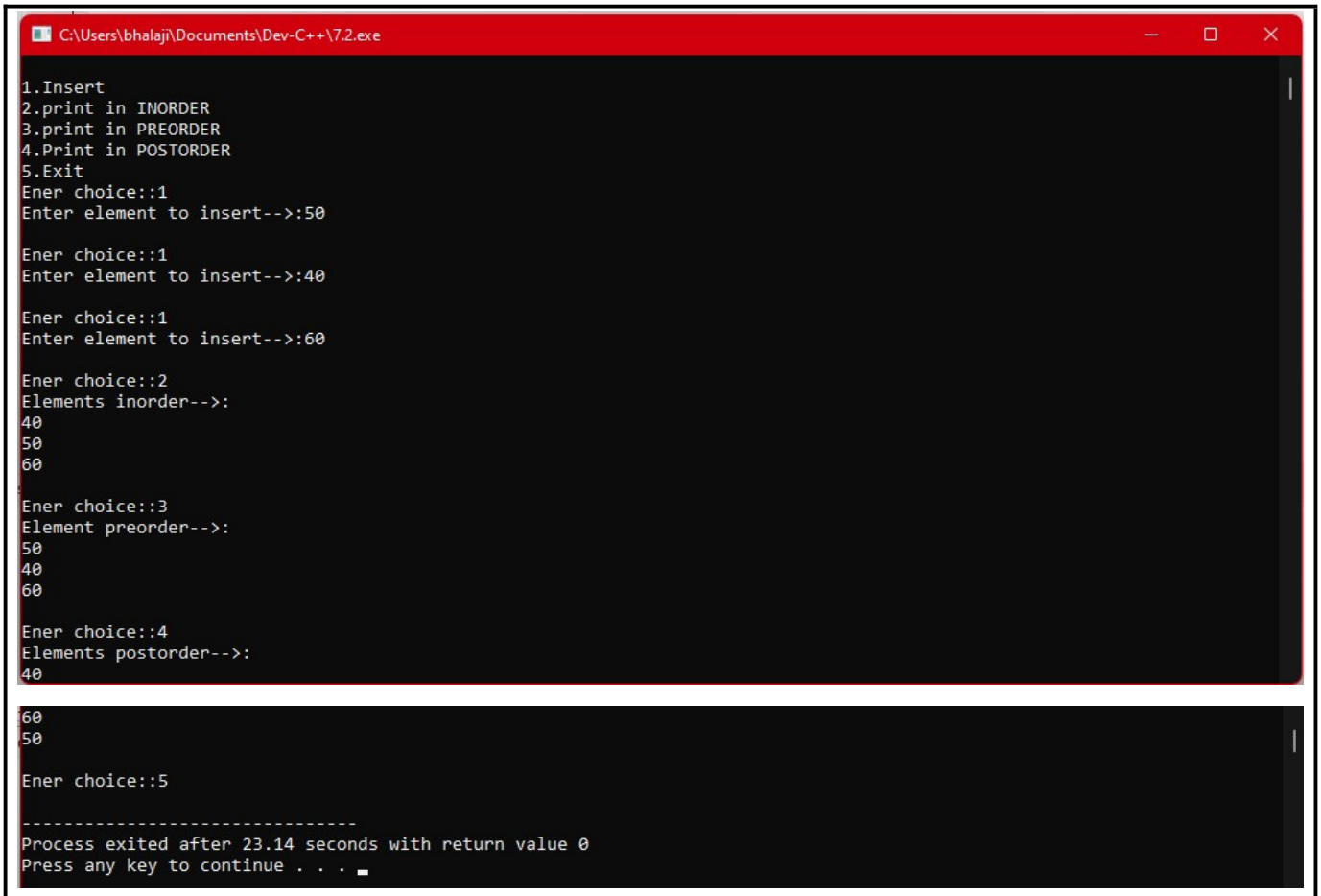
PROGRAM 7.2-CODING:

```
/*******Binary terr Traversal Inorder Preorder Postorder*******/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
        int data;
        struct node *right;
        struct node *left;
}*right=NULL,*left=NULL;
struct node *Tr=NULL;
void insert();
void print_inorder();
void print_preorder();
void print_postorder();
int main()
{
        int ch,val;
                printf("\n1.Insert\n2.print in INORDER\n3.print in PREORDER\n4.Print in
POSTORDER\n5.Exit");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                printf("Enter element to insert-->:");
                                scanf("%d",&val);
                                insert(&Tr,val);
                                break;
                        case (2):
                                printf("Elements inorder-->:\n");
```

```
                                print_inorder(Tr);
                                break;
                        case (3):
                                printf("Element preorder-->:\n");
                                print_preorder(Tr);
                                break;
                        case (4):
                                printf("Elements postorder-->:\n");
                                print_postorder(Tr);
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
}
void insert(struct node ** tree, int val)
{
   struct node *temp = NULL;
   if(!(*tree))
   {
      temp = (struct node *)malloc(sizeof(struct node));
      temp->left = temp->right = NULL;
      temp->data = val;
      *tree = temp;
      return;
   }
   else if(val < (*tree)->data)
   {
      insert(&(*tree)->left, val);
   }
   else if(val > (*tree)->data)
   {
      insert(&(*tree)->right, val);
```

```
    }
}
void print_preorder(struct node * tree)
{
    if (tree)
    {
        printf("%d\n",tree->data);
        print_preorder(tree->left);
        print_preorder(tree->right);
    }
}
void print_inorder(struct node * tree)
{
    if (tree)
    {
        print_inorder(tree->left);
        printf("%d\n",tree->data);
        print_inorder(tree->right);
    }
}
void print_postorder(struct node * tree)
{
    if (tree)
    {
        print_postorder(tree->left);
        print_postorder(tree->right);
        printf("%d\n",tree->data);
    }
}
```

PROGRAM 7.2-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on binary search tree are executed and outputs are verified.

EXP NO: 08

DATE:20.10.2022

TREE TRAVERSAL

AIM:

To write c programs to tree traversal and its operations.

PROGRAM 8.1: HEIGHT OF THE BINARY SEARCH TREE

ALGORITHM:

STEP 1: Start

STEP 2:for Insertion

```
IF TREE = NULL
    Allocate memory for TREE
    SET TREE DATA = VAL
    SET TREE LEFT = TREE RIGHT = NULL
ELSE
    IF VAL < TREE DATA
        Insert(TREE LEFT, VAL)
    ELSE
        Insert(TREE RIGHT, VAL)
        [END OF IF]
[END OF IF]
```

STEP 3: for finding the height

```
IF TREE = NULL
Return
ELSE
SET LeftHeight = Height(TREE LEFT)
SET RightHeight = Height(TREE RIGHT)
IF LeftHeight > RightHeight
Return LeftHeight+1
ELSE
Return RightHeight+1
[END OF IF]
```

[END OF IF]

STEP 4:INORDER(TREE LEFT)

STEP 5: Write TREE DATA

STEP 6: INORDER(TREE RIGHT)

[END OF LOOP]

STEP 7: Repeat Steps 4 to 6 while TREE != NULL

STEP 8: Stop.

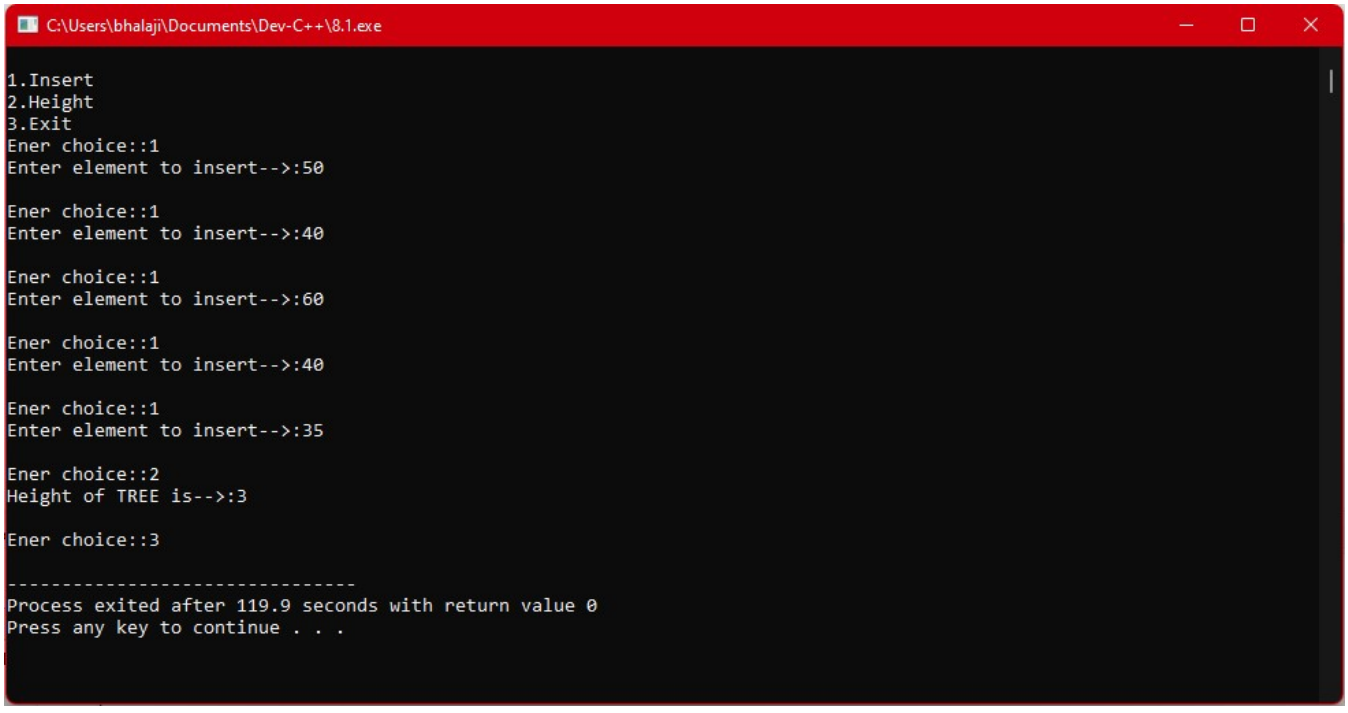
PROGRAM 8.1-CODING:

```
/*******Height of the tree*******/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
struct node
{
        int data;
        struct node *right;
        struct node *left;
}*right=NULL,*left=NULL;
struct node *Tr=NULL;
void insert(struct node ** tree, int val);
int heightoftree(struct node *t);
int main()
{
        int ch,val,hei;
        printf("\n1.Insert\n2.Height\n3.Exit");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                printf("Enter element to insert-->:");
```

```
                                scanf("%d",&val);
                                insert(&Tr,val);
                                break;
                        case (2):
                                hei=heightoftree(Tr);
                                printf("Height of TREE is-->:%d\n",hei);
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
}
void insert(struct node ** tree, int val)
{
   struct node *temp = NULL;
   if(!(*tree))
   {
      temp = (struct node *)malloc(sizeof(struct node));
      temp->left = temp->right = NULL;
      temp->data = val;
      *tree = temp;
      return;
   }
   else if(val < (*tree)->data)
   {
      insert(&(*tree)->left, val);
   }
   else if(val > (*tree)->data)
   {
      insert(&(*tree)->right, val);
   }
}
int heightoftree(struct node *t)
```

```
{
        int L_height,R_height;
        if(t==NULL)
        {
                return 0;
        }
        else
        {
                L_height= heightoftree(t->left);
                R_height= heightoftree(t->right);
                if(L_height > R_height)
          {
        return L_height + 1;
                }
                else
                {
                        return R_height + 1;
                }
        }
}
```

PROGRAM 8.1-OUTPUT:



PROGRAM 8.2: CREATE A BINART TREE FOR THE GIVEN ARRAY

ALGORITHM:

STEP 1: Start

STEP 2: creating binary tree

IF I<N

CREATRE DINAMIC MEMORY ALOCATION

ASSING node->data = arr[i];

node->left = node->right = NULL;

ASSIGN root=node;

// insert left child

root->left = insertLevelOrder(arr,2 * i + 1, n,Tr);

// insert right child

root->right = insertLevelOrder(arr,2 * i + 2, n,Tr);

STEP 3:create a array using the for loop.

STEP 4: Insert the element in the tree using the for loop.

STEP 5: Print inorder using the above given algorithem.

STEP 6: Stop.

PROGRAM 8.2-CODING:

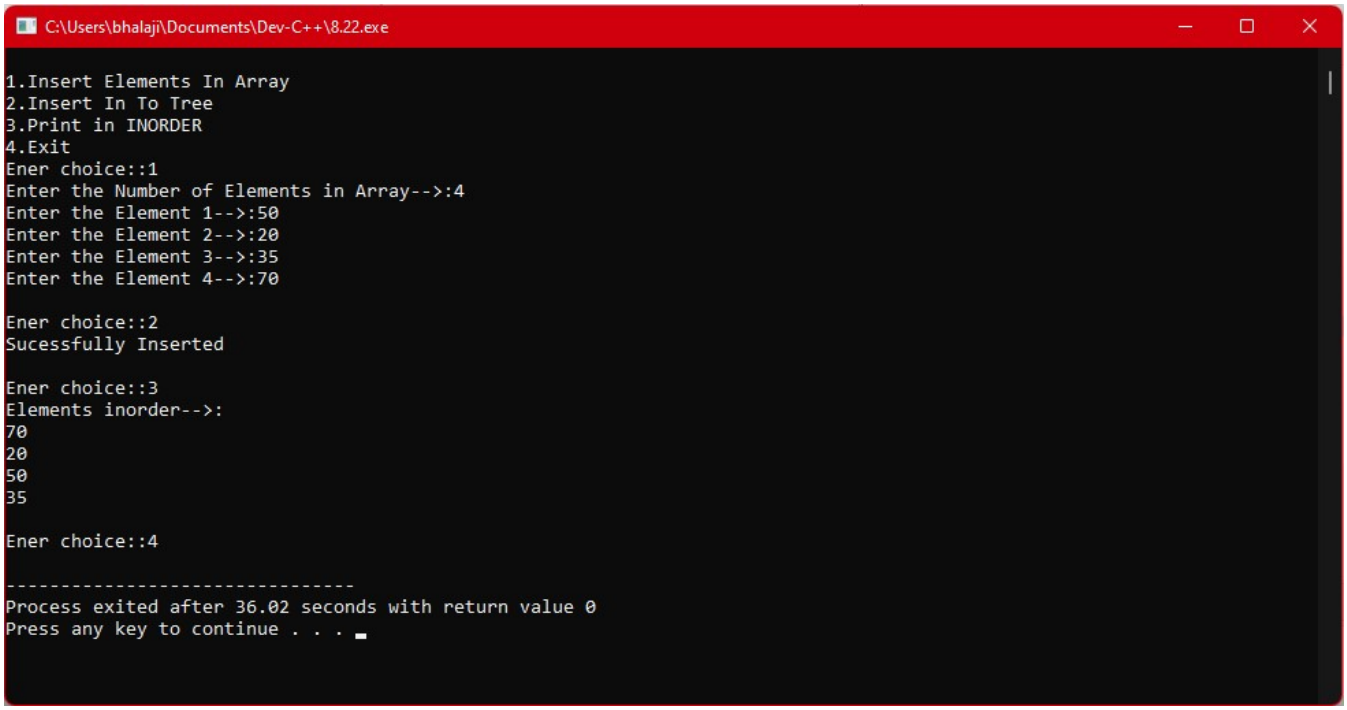
```
/*******Creat Binary Tree For A Given Array*******/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
struct node
{
        int data;
        struct node *right;
        struct node *left;
}*right=NULL,*left=NULL;
struct node *Tr=NULL,*root;
int i,n,a[100];
struct node* insertLevelOrder(int arr[],int i, int n,struct node *root);
void arraycreation();
void print_inorder(struct node * tree);
```

```
void insert_in_tree();
int main()
{
        int ch,val,hei;
        printf("\n1.Insert Elements In Array\n2.Insert In To Tree\n3.Print in INORDER\n4.Exit");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                arraycreation();
                                break;
                        case (2):
                                insert_in_tree();
                                break;
                        case (3):
                                printf("Elements inorder-->:\n");
                                print_inorder(root);
                                break;
                        default:
                                exit(0);
                }
        }
        return 0;
}
struct node* insertLevelOrder(int arr[],int i, int n,struct node *root)
{
   root = NULL;
   if (i < n)
   {
        struct node* node = (struct node*)malloc(sizeof(struct node));
      node->data = arr[i];
```

```
        node->left = node->right = NULL;
        root=node;
        // insert left child
        root->left = insertLevelOrder(arr,2 * i + 1, n,Tr);
        // insert right child
        root->right = insertLevelOrder(arr,2 * i + 2, n,Tr);
    }
    return root;
}
void print_inorder(struct node * tree)
{
    if (tree)
    {
        print_inorder(tree->left);
        printf("%d\n",tree->data);
        print_inorder(tree->right);
    }
}
void insert_in_tree()
{
            for(i=0;i<n;i++)
            {
                        root=insertLevelOrder(a,0,n,Tr);
            }
            printf("Sucessfully Inserted\n");
}
void arraycreation()
{
            printf("Enter the Number of Elements in Array-->:");
            scanf("%d",&n);
            for(i=0;i<n;i++)
            {
                        printf("Enter the Element %d-->:",i+1);
                        scanf("%d",&a[i]);
```

```
        }
}
```

PROGRAM 8.2-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on tree traversal are executed and outputs are verified.

EXP NO: 09

DATE:31.10.2022

IMPLEMENTATION OF AVL TREE

AIM:

To write c programs to implement queue ADT and its operations.

PROGRAM 9: IMPLIMENTATION OF QUEUE USING LINKED LIST

ALGORITHM:

Step 1: Start
Step 2: Create a node using Create Node function with pointer and value
Step 2.1: if pointer points the null
Step 2.1.1: Assign value to pointer data
Step 2.2: Else if value is less than pointer data
Step 2.2.1: Goto step 2 with move the pointer to the left
Step 2.2.2: Check the balance factor is greater than 2
Step 2.2.2.1: If value is less than pointer left data
Step 2.2.2.1.1: Call the function LL(root)
Step 2.2.2.2: Else
Step 2.2.2.2.1: Call the function LR(root)
Step 2.3: Else if value is greater than pointer data
Step 2.3.1: Goto step 2 with move the pointer to the right
Step 2.3.2: Check the balance factor is greater than 2
Step 2.3.2.1: If value is less than pointer left data
Step 2.3.2.1.1: Call the function RR(root)
Step 2.3.2.2: Else
Step 2.3.2.2.1: Call the function RL(root)
Step 2.4: Else
Step 2.4.1: Display Duplicate Node
Step 3: Using inorderTraversal function with pointer to Display the tree
Step 3.1: If root == null
Step 3.1.1: Return
Step 3.2: Else
Step 3.2.1: Goto step3 with pointer->left
Step 3.2.2: Display the element
Step 3.2.3: Goto step3 with pointer->right
Step 4: Using nodeheight with pointer to find height of the tree
Step 4.1: if(root==NULL)
Step 4.1.1: return 0;
Step 4.2: if(root->left==NULL)
Step 4.2.1: lh=0;
Step 4.3: else
Step 4.3.1: lh=1+nodeHeight(root->left);
Step 4.4: if(root->right==NULL)
Step 4.4.1: rh=0;
Step 4.5: else
Step 4.5.1: rh=1+nodeHeight(root->right);

Step 4.6: if(lh > rh)
Step 4.6.1: return lh;
Step 4.7: else
Step 4.7.1: return rh;
Step 5: Using balancefactor with pointer to find balanceFactor of the tree
Step 5.1: if(root==NULL)
Step 5.1.1: return 0;
Step 5.2: if(root->left==NULL)
Step 5.2.1: lh=0;
Step 5.3: else
Step 5.3.1: lh=1+nodeHeight(root->left);
Step 5.4: if(root->right==NULL)
Step 5.4.1: rh=0;
Step 5.5: else
Step 5.5.1: rh=1+nodeHeight(root->right);
Step 5.6: return (lh-rh);
Step 6: Using rotateright with pointer to rotate the tree to right
Step 6.1: node *temp;
Step 6.2: temp=root->right;
Step 6.3: free(root->right);
Step 6.4: temp->left=root;
Step 6.5: root->height=nodeHeight(root);
Step 6.6: temp->height=nodeHeight(temp);
Step 6.7: return temp;
Step 7: Using rotateleft with pointer to rotate the tree to left
Step 7.1: node *temp;
Step 7.2: temp=root->left;
Step 7.3: free(root->left);
Step 7.4: temp->right=root;
Step 7.5: root->height=nodeHeight(root);
Step 7.6: temp->height=nodeHeight(temp);
Step 7.7: return temp;
Step 8:Stop.

PROGRAM 9-CODING:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node
{
    int data;
    struct node *left;
    struct node *right;
    int height;
}node;
node *createNode(node *, int);
```

```
void inorderTraversal(node *);
int balanceFactor(node *);
int nodeHeight(node *);
node *rotateRight(node *);
node *rotateLeft(node *);
node *LL(node *);
node *RR(node *);
node *LR(node *);
node *RL(node *);

int main()
{
	int n,choice;
	node *root=NULL;
	while(1)
	{
		printf("\n1. Create/Insert Node");
		printf("\n2. Convert into AVL Tree");
		printf("\n3. Exit\n");
		printf("\nEnter your Choice: ");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:
				printf("Enter Node to Insert: ");
				scanf("%d",&n);
				root=createNode(root, n);
				break;
			case 2:
				printf("\nBALANCED AVL TREE \n");
				inorderTraversal(root);
				break;
			case 3:
				exit(1);
```

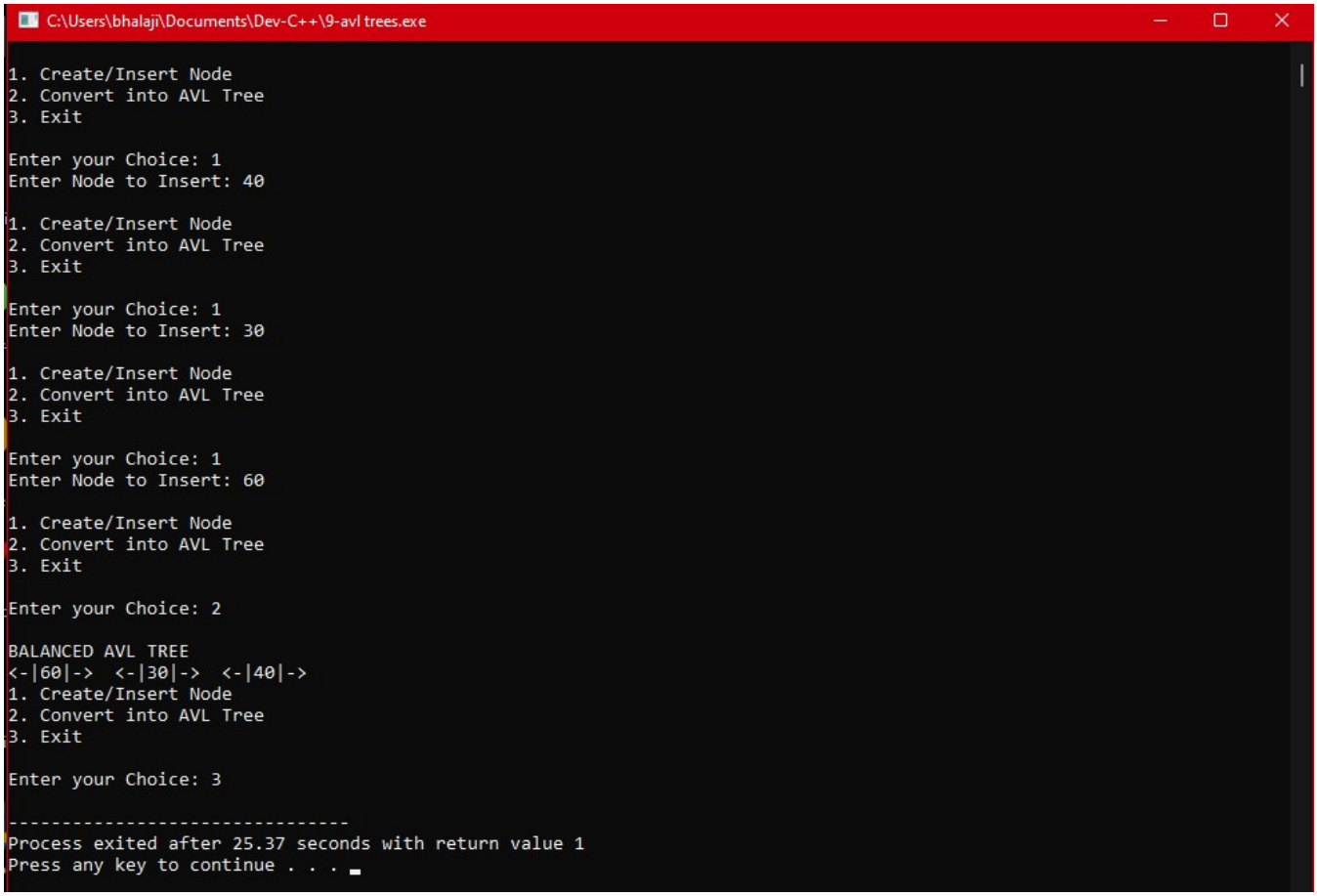
```
                }
        }
}
node *createNode(node *root, int n)
{
        if(root==NULL)
        {
                root=(node *)malloc(sizeof(node));
                root->data=n;
                root->right=NULL;
                root->left=NULL;
        }
        else if(n < root->data)
        {
                root->left=createNode(root->left, n);
                if(balanceFactor(root)>=2)
                if(n < root->left->data)
                        root=LL(root);
                else
                        root=LR(root);
        }
        else if(n > root->data)
        {
                root->left=createNode(root->left, n);
                if(balanceFactor(root)<=-2)
                if(n > root->right->data)
                        root=RR(root);
                else
                        root=RL(root);
        }
        else
                printf("\nDuplicate Node\n");
        return root;
}
```

```
void inorderTraversal(node *root)
{
	if(root==NULL)
		return;
	else
	{
		inorderTraversal(root->left);
		printf("<-|%d|->  ",root->data);
		inorderTraversal(root->right);
	}
}
int nodeHeight(node *root)
{
	int lh, rh;
	if(root==NULL)
		return 0;
	if(root->left==NULL)
		lh=0;
	else
		lh=1+nodeHeight(root->left);
	if(root->right==NULL)
		rh=0;
	else
		rh=1+nodeHeight(root->right);
	if(lh > rh)
		return lh;
	else
		return rh;
}
int balanceFactor(node *root)
{
	int lh,rh;
	if(root==NULL)
		return 0;
```

```
        if(root->left==NULL)
                lh=0;
        else
                lh=1+nodeHeight(root->left);
        if(root->right==NULL)
                rh=0;
        else
                rh=1+nodeHeight(root->right);
        return (lh-rh);
}
node *rotateLeft(node *root)
{
        node *temp;
        temp=root->left;
        free(root->left);
        temp->right=root;
        root->height=nodeHeight(root);
        temp->height=nodeHeight(temp);
        return temp;
}
node *rotateRight(node *root){
        node *temp;
        temp=root->right;
        free(root->right);
        temp->left=root;
        root->height=nodeHeight(root);
        temp->height=nodeHeight(temp);
        return temp;
}
node *LL(node *root)
{
        root=rotateLeft(root);
        return root;
}
```

```
node *RR(node *root)
{
        root=rotateRight(root);
        return root;
}
node *LR(node *root)
{
        root=rotateRight(root->left);
        return root;
}
node *RL(node *root)
{
        root=rotateLeft(root->right);
        return root;
}
```

PROGRAM 9-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on AVL Tree are executed and outputs are verified.

EXP NO: 10

DATE:07.11.2022

IMPLEMENTATION OF HEAP & GRAPH

AIM:

To write c programs to implement of graph and its operations.

PROGRAM 10.1: IMPLIMENTATION OF HEAP

ALGORITHM:

STEP 1: Start

STEP 2: First we have to slect the array in to the all -1.

STEP 3: the for Linear Probing inside the while loop index=(k%h+i)%m if(a[ind]!=-1) increment i++;

else a[ind]=k then break;

STEP 4:the for Quadratic Probing inside the while loop index=(k%h+c1i+c2i^2)%m if(a[ind]!=-1) increment i++ else a[ind]=k then break;

STEP 5:the for Double hashing inside the while loop index=(k%h+c1i+c2i^2)%m if(a[ind]!=-1) increment i++ else a[ind]=k then break;

STEP 6: Stop.

PROGRAM 10.1-CODING:

```
/*****Implimentation of Hashing****/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void linear_prob(int k);
void double_has(int k);
void quadratic_prob(int k);
void display(int d[10]);
void convert(int e[10]);
int a[10];
int b[10];
int c[10];
int m=10;
int main()
{
```

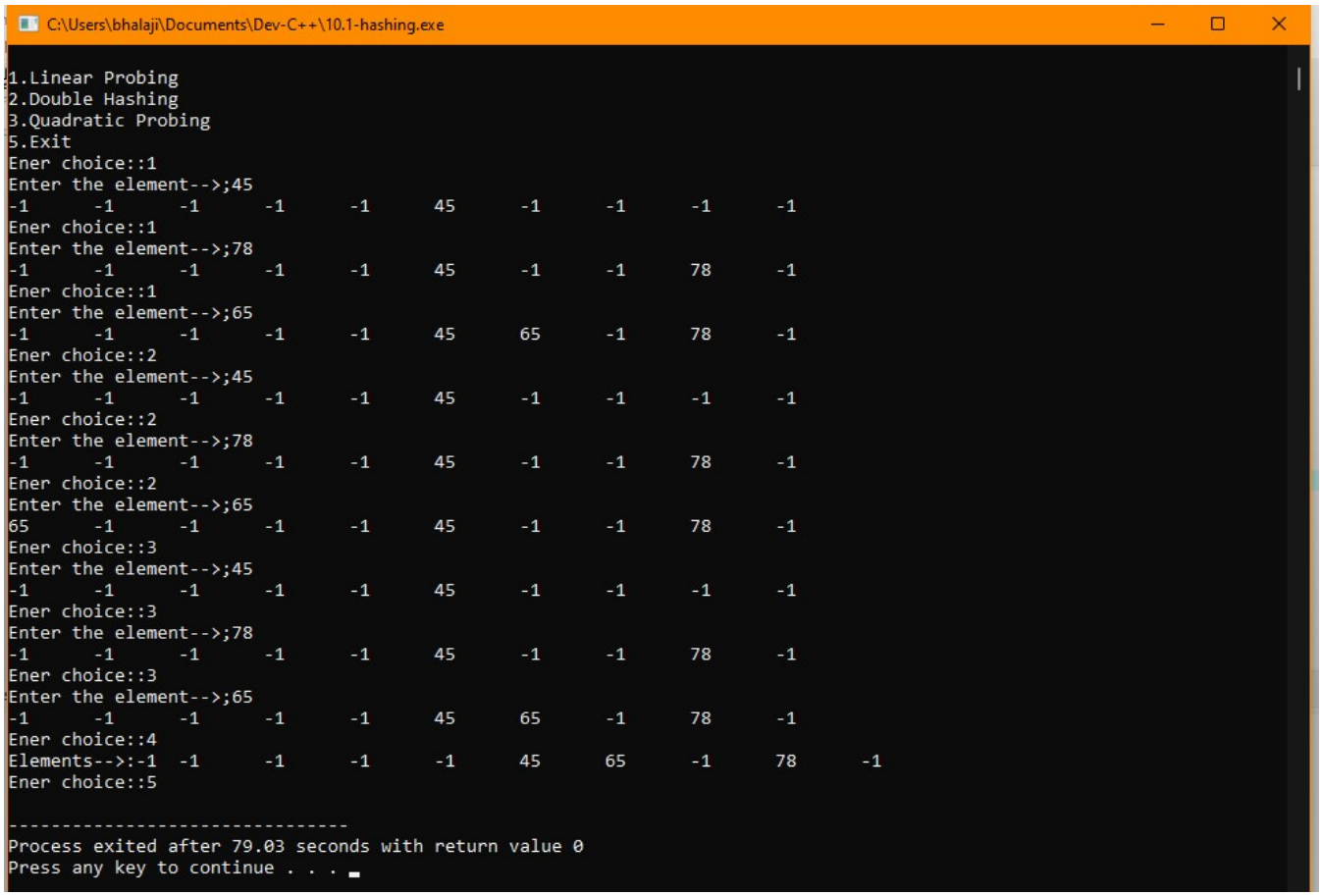
```
//struct node *T;
int ch,k;
printf("\n1.Linear Probing\n2.Double Hashing\n3.Quadratic Probing\n5.Exit");
convert(a);
convert(b);
convert(c);

while(1)
{
        printf("\nEner choice::");
        scanf("%d",&ch);
        switch(ch)
        {
                case (1):
                        printf("Enter the element-->;");
                        scanf("%d",&k);
                        linear_prob(k);
                        display(a);
                        break;
                case (2):
                        printf("Enter the element-->;");
                        scanf("%d",&k);
                        double_has(k);
                        display(b);
                        break;
                case (3):
                        printf("Enter the element-->;");
                        scanf("%d",&k);
                        quadratic_prob(k);
                        display(c);
                        break;
                case (4):
                        printf("Elements-->:");
                        display(a);
```

```
                        break;
                default:
                        exit(0);
                }
        }
        return 0;
}
void linear_prob(int k)
{
        //convert
        int i=0,ind;
        int hd=k%m;
        while(1)
        {
                ind=(hd+i)%m;
                if(a[ind]!=-1)
                {
                        i++;
                }
                else
                {
                        a[ind]=k;
                        break;
                }
        }
}
void double_has(int k)
{
        int i=0,ind,c1=2,c2=3;
        int hd=k%m;
        while(1)
        {
                ind=(hd+c1*i+c2*i*i)%m;
                if(b[ind]!=-1)
```

```
                {
                        i++;
                }
                else
                {
                        b[ind]=k;
                        break;
                }
        }
}
void quadratic_prob(int k)
{
        int i=0,ind;
        int hd1=k%m;
        int hd2=k%8;
        while(1)
        {
                ind=(hd1+i*hd2)%m;
                if(c[ind]!=-1)
                {
                        i++;
                }
                else
                {
                        c[ind]=k;
                        break;
                }
        }
}
void display(int d[10])
{
        int i;
        for(i=0;i<m;i++)
        {
```

```
                printf("%d\t",d[i]);
        }
}
void convert(int e[10])
{
        int i;
        for(i=0;i<m;i++)
        {
                e[i]=-1;
        }
}
```

PROGRAM 10.1-OUTPUT:



PROGRAM 10.2: IMPLIMENTATION OF GRAPH TRAVERSAL

ALGORITHM:

STEP 1: Start

STEP 2: for Breadthe First Traversal

SET STATUS=1 (ready state)

for each node in G

STEP 3: Enqueue the starting node A

and set its STATUS=2

(waiting state)

STEP 4: Repeat Steps 4 and 5 until

QUEUE is empty

STEP 5: Dequeue a node N. Process it

and set its STATUS=3

(processed state).

STEP 6: Enqueue all the neighbours of

N that are in the ready state

(whose STATUS=1) and set

their STATUS=2

(waiting state)

[END OF LOOP]

STEP 7: for Depth First Traversal

SET STATUS=1 (ready state) for each node in G

STEP 8: Push the starting nodeAon the stack and set

its STATUS=2 (waiting state)

STEP 9: Repeat Steps 4 and 5 until STACK is empty

STEP 10: Pop the top node N. Process it and set its

STATUS=3 (processed state)

STEP11: Push on the stack all the neighbours of N that

are in the ready state (whose STATUS=1) and

set their STATUS=2 (waiting state)

[END OF LOOP]

STEP 12: Stop.

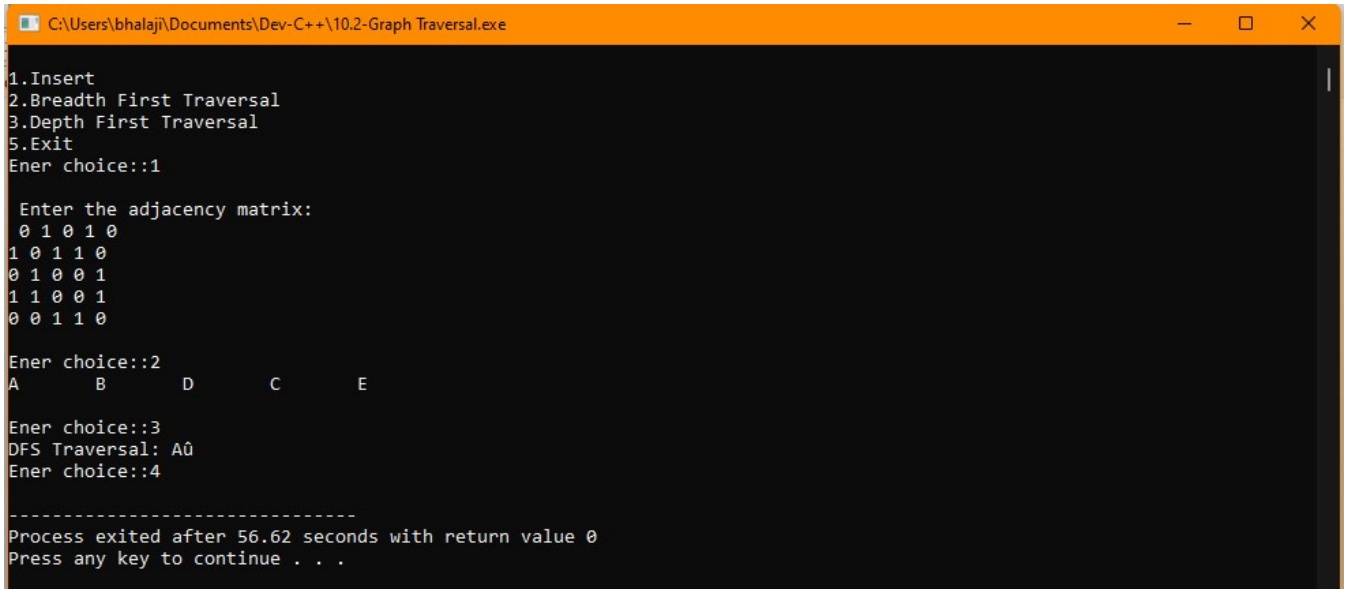
PROGRAM 10.2-CODING:

```
#include <stdio.h>
#define MAX 5
int main()
{
        int ch,k;
        int visited[MAX] = {0};
        int adj[MAX][MAX], i, j;
        printf("\n1.Insert \n2.Breadth First Traversal\n3.Depth First Traversal\n5.Exit");
        while(1)
        {
                printf("\nEner choice::");
                scanf("%d",&ch);
                switch(ch)
                {
                        case (1):
                                printf("\n Enter the adjacency matrix:\n ");
                                for(i = 0; i < MAX; i++)
                                   for(j = 0; j < MAX; j++)
                                           scanf("%d", &adj[i][j]);
                                //breadth_first_search(adj,visited,0);
                                break;
                        case (2):
                                /*printf("\n Enter the adjacency matrix: ");
                                for(i = 0; i < MAX; i++)
                                   for(j = 0; j < MAX; j++)
                                           scanf("%d", &adj[i][j]);
                                printf("DFS Traversal: ");*/
                                breadth_first_search(adj,visited,0);
                                printf("\n");
                                break;
                        case (3):
                                printf("DFS Traversal: ");
                                depth_first_search(adj,visited,0);
```

```
                                break;
                        default:
                                exit(0);
                }
        }
 return 0;
}
void depth_first_search(int adj[][MAX],int visited[],int start)
{
        int stack[MAX];
        int top = -1, i;
        printf("%c–",start + 65);
        visited[start] = 1;
        stack[++top] = start;
        while(top != -1)
        {
                start = stack[top];
                for(i = 0; i < MAX; i++)
                {
                        if(adj[start][i] && visited[i] == 0)
                        {
                                stack[++top] = i;
                                printf("%c–", i + 65);
                                visited[i] = 1;
                                break;
                        }
                }
                if(i == MAX)
                {
                        top--;
                }
        }
}
void breadth_first_search(int adj[][MAX],int visited[],int start)
```

```
{
        int queue[MAX],rear = -1,front =-1, i;
        queue[++rear] = start;
        visited[start] = 1;
        while(rear != front)
        {
                start = queue[++front];
                if(start == MAX)
                   printf("5\t");
                else
                  printf("%c \t",start + 65);
                  for(i = 0; i < MAX; i++)
                  {
                          if(adj[start][i] == 1 && visited[i] == 0)
                                  {
                                          queue[++rear] = i;
                                          visited[i] = 1;
                                  }
                          }
        }
}
```

PROGRAM 10.2-OUTPUT:



DESCRIPTION	MAXIMUM MARK	MARKS SCORED
OBSERVATION	20	
RECORD	05	
TOTAL	25	

RESULT:

Thus the all the three given programs based on queue ADT are executed and outputs are verified.