PROGRAM-1

```
#include<stdio.h>
#include<string.h>
#define MAX 20
int top = -1;
char stack[MAX];
char push(char item)
{
if(top == (MAX-1))
printf("Stack Overflow\n");
else
stack[++top] =item;
}
char pop()
{
if(top == -1)
printf("Stack Underflow\n");
else
return stack[top--];
}
main()
{
char str[20];
int i;
printf("Enter the string : " );
gets(str);
for(i=0;i<strlen(str);i++)
push(str[i]);
for(i=0;i<strlen(str);i++)
str[i]=pop();
printf("Reversed string is : ");
puts(str);
}
```

PROGRAM-2

```
#include<stdio.h>
char stack[20];
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;
}
main()
{
 char exp[20];
 char *e, x;
 printf("Enter the expression :: ");
scanf("%s",exp);
 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)
 {
 printf("%c",pop());
 }
}
```

PROGRAM-3

```
}
int main()
{
 struct queue *q = (struct queue*)malloc(sizeof(struct queue));
 int f = 0, a;
 char ch = 'y';
 q->stack1 = NULL;
 q->stack2 = NULL;
 while (ch == 'y'||ch == 'Y') {
 printf("enter ur choice\n1.add to queue\n2.remove
 from queue\n3.display\n4.exit\n");
 scanf("%d", &f);
 switch(f) {
 case 1 : printf("enter the element to be added to queue\n");
 scanf("%d", &a);
 enqueue(q, a);
 break;
 case 2 : dequeue(q);
 break;
 case 3 : display(q->stack1, q->stack2);
 break;
 case 4 : exit(1);
 break;
 default : printf("invalid\n");
 break;
 }
 }
}
```

PROGRAM-4

```
#include<stdlib.h>
#include<stdio.h>
struct bin_tree {
int data;
struct bin_tree * right, * left;
};
typedef struct bin_tree node;
void insert(node ** tree, int val)
{
 node *temp = NULL;
 if(!(*tree))
 {
 temp = (node *)malloc(sizeof(node));
 temp->left = temp->right = NULL;
 temp->data = val;
 *tree = temp;
 return;
 }
 if(val < (*tree)->data)
 {
 insert(&(*tree)->left, val);
 }
 else if(val > (*tree)->data)
 {
 insert(&(*tree)->right, val);
 }
}
void deltree(node * tree)
{
 if (tree)
 {
 deltree(tree->left);
 deltree(tree->right);
 free(tree);
 }
}
node* search(node ** tree, int val)
{
 if(!(*tree))
 {
```

```
 return NULL;
 }
 if(val < (*tree)->data)
 {
 search(&((*tree)->left), val);
 }
 else if(val > (*tree)->data)
 {
 search(&((*tree)->right), val);
 }
 else if(val == (*tree)->data)
 {
 return *tree;

}
}
void main()
{
 node *root;
 node *tmp;
 int i;
 root = NULL;
 insert(& root, 2);
 insert(& root, 41);
 insert(& root, 9);
 insert(& root, 18);
 insert(& root, 6);
 insert(& root, 7);
 insert(& root, 14);

 tmp = search(& root, 4);
 if (tmp)
 {
 printf("Searched node=%d\n", tmp->data);
 }
 else
 {
 printf("Data Not found in tree.\n");
 }
 deltree(root);
}
```