**DSA OPEN ENDED PROJECT**

|  |  |  |
| --- | --- | --- |
| **Name** | **USN** | **Roll.No** |
| **Ruchith T** | **01fe20bec129** | **313** |
| **Prajwal J** | **01fe20bec138** | **322** |

**SUB FUNCTIONS INSIDE CODE:**

**1)Creating the stack**

1. **struct** stack {                              //Creating a Structure named stack
2. NODE items[STACKSIZE];
3. **int** top;
4. };
5. /\* Function to create a newnode \*/
6. **struct** node \*GetNode()
7. {
8. **struct** node \*p;
9. p = (NODE) **malloc**(**sizeof**(**struct** node));
10. **if**(p== NULL)
11. {
12. **printf**("Insufficient Memory");
13. **exit**(0);
14. }
15. p->lc = NULL;
16. p->rc=NULL;
17. **return** p;
18. }

**2)Push and Pop Functions**

1. **void** push( **struct** stack \*s , NODE p)           //Push Function
2. {
3. **if**(s->top==STACKSIZE)
4. {
5. **printf**("STACK FULL!!!!!!!!");
6. **exit**(0);
7. }
8. s->top++;
9. s->items[s->top]=p;
10. }
12. NODE pop( **struct** stack \*s )                     // Pop Function
13. {
14. NODE p;
15. **if**(s->top==-1)
16. {
17. **printf**("STACK EMPTY!!!!!!!!");
18. **exit**(0);
19. }
20. p=s->items[s->top];
21. s->top--;
22. **return** p;
23. }

**3)Incoming Priority**

1. **int** icp(**char** x)
2. {
3. **switch**(x)
4. {

7. **case** '+':
8. **case** '-': **return** 1;
9. **case** '/':
10. **case** '\*': **return** 3;
11. **case** '^': **return** 6;
12. }
13. }

**4)In Stack Priority**

1. **int** isp(**char** x)
2. {
3. **switch**(x)
4. {
6. **case** '+':
7. **case** '-': **return** 2;
8. **case** '/':
9. **case** '\*': **return** 4;
10. **case** '^': **return** 5;
11. }
12. }

**5)To Create Tree**

1. NODE  CreateExpTree( **char** a[])
2. {
3. **struct** stack  opers,opnds;
4. opers.top=-1;
5. opnds.top=-1;
7. **int** i=0;
8. **char**  symb;
9. NODE p,q,r,r1,r2;
11. **while**(a[i] != '\0')
12. {
13. symb =a[i++];
14. // printf("%c",symb);
15. p = GetNode();
16. p->data =symb;
17. **if**((symb >='a'  &&   symb <='z' ) || (symb >='A'  &&   symb <='Z' ))
18. push(&opnds, p);
19. **else** **if**( symb =='+' ||  symb =='-'  || symb =='\*' ||  symb =='/' || symb == '^')
20. {
22. **while**( opers.top != -1 )
23. {
24. q= opers.items[opers.top];
25. **if**(isp(q->data) >= icp(symb))
26. {
27. q= pop(&opers);
28. r2=pop(&opnds);
29. r1=pop(&opnds);
30. q->lc=r1;
31. q->rc=r2;
32. push(&opnds,q);
33. }
34. **else**
35. **break**;
36. }
37. push(&opers, p);
39. }
40. }
42. **while**( opers.top != -1 )
43. {
45. q= pop(&opers);
46. r2=pop(&opnds);
47. r1=pop(&opnds);
48. q->lc=r1;
49. q->rc=r2;
50. push(&opnds,q);
51. }
53. **return** (pop(&opnds));
54. }

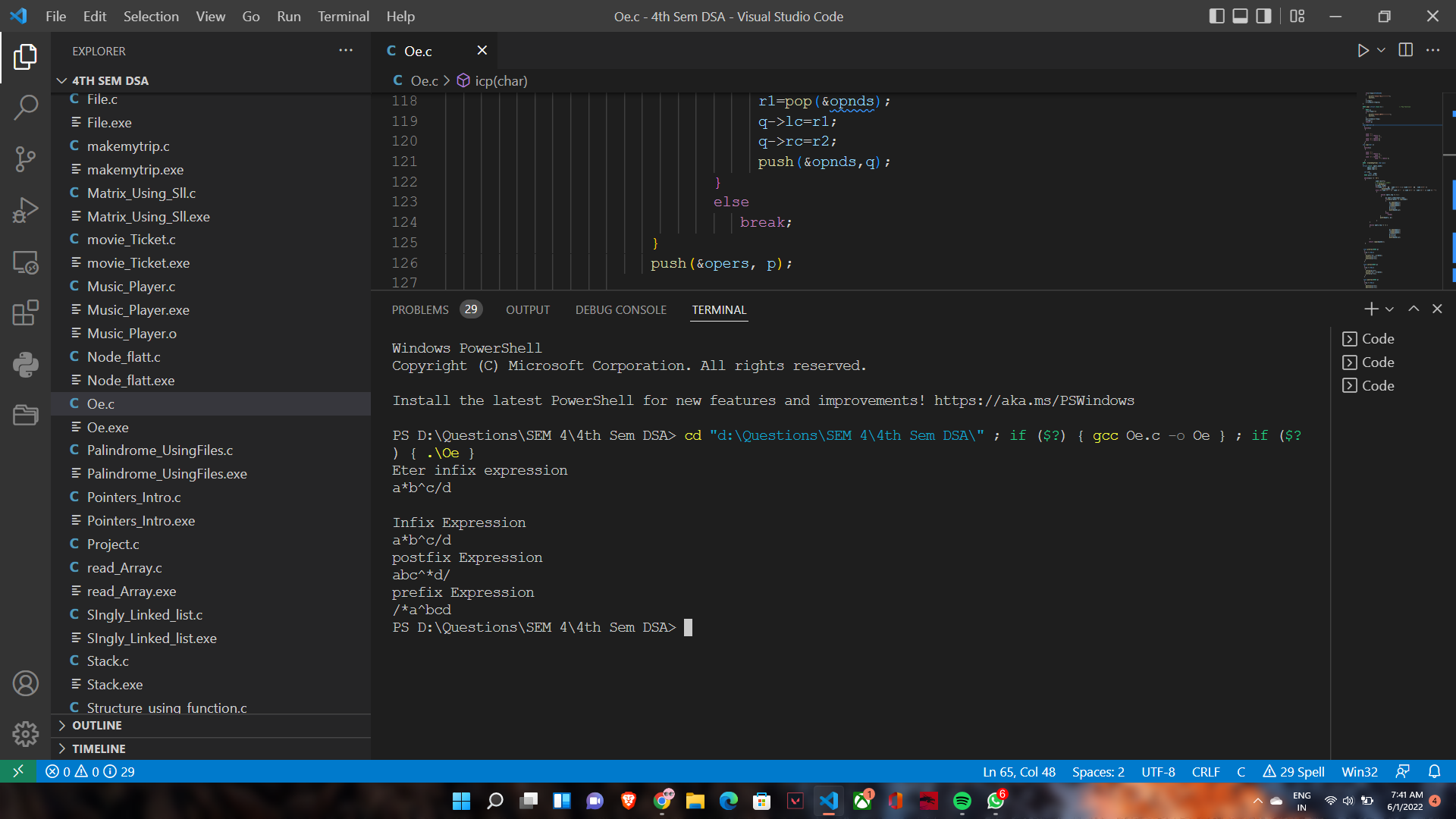
**6)Postfix,Prefix and Infix Conversion**

1. **void** pretrav(NODE p)
2. {
3. **if**(p != NULL)
4. {
5. **printf**("%c", p->data);
6. pretrav(p->lc);
7. pretrav(p->rc);
8. }
9. }
11. **void** intrav(NODE p)
12. {
13. **if**(p != NULL)
14. {
15. intrav(p->lc);
16. **printf**("%c", p->data);
17. intrav(p->rc);
18. }
19. }
21. **void** postrav(NODE p)
22. {
23. **if**(p != NULL)
24. {
25. postrav(p->lc);
26. postrav(p->rc);
27. **printf**("%c", p->data);
28. }
29. }

**TEST CASES AND THEIR OUTPUTS:**

**A screenshot of a computer

Description automatically generated with medium confidence**

****

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**