1. Lexical\_analysier

Source code :

#include <stdio.h>

#include <string.h>

int main()

{

    char a[100];

    printf("Enter the expression\n");

    gets(a);

    int n = strlen(a);

    for (int i = 0; i < n; i++)

    {

        if (a[i] >= 'a' && a[i] <= 'z' || a[i] >= 'A' && a[i] <= 'Z')

        {

            printf("%c is an Identifier\n", a[i]);

        }

        else if (a[i] >= '0' && a[i] <= '9')

        {

            printf("% c is an Number\n", a[i]);

        }

        else if (a[i] == '=')

        {

            printf("%c is an Assignment Operator\n", a[i]);

        }

        else if (a[i] == '+' || a[i] == '-' || a[i] == '\*' || a[i] == '%' || a[i] == '/')

        {

            printf("%c is an Arithmetic operator\n", a[i]);

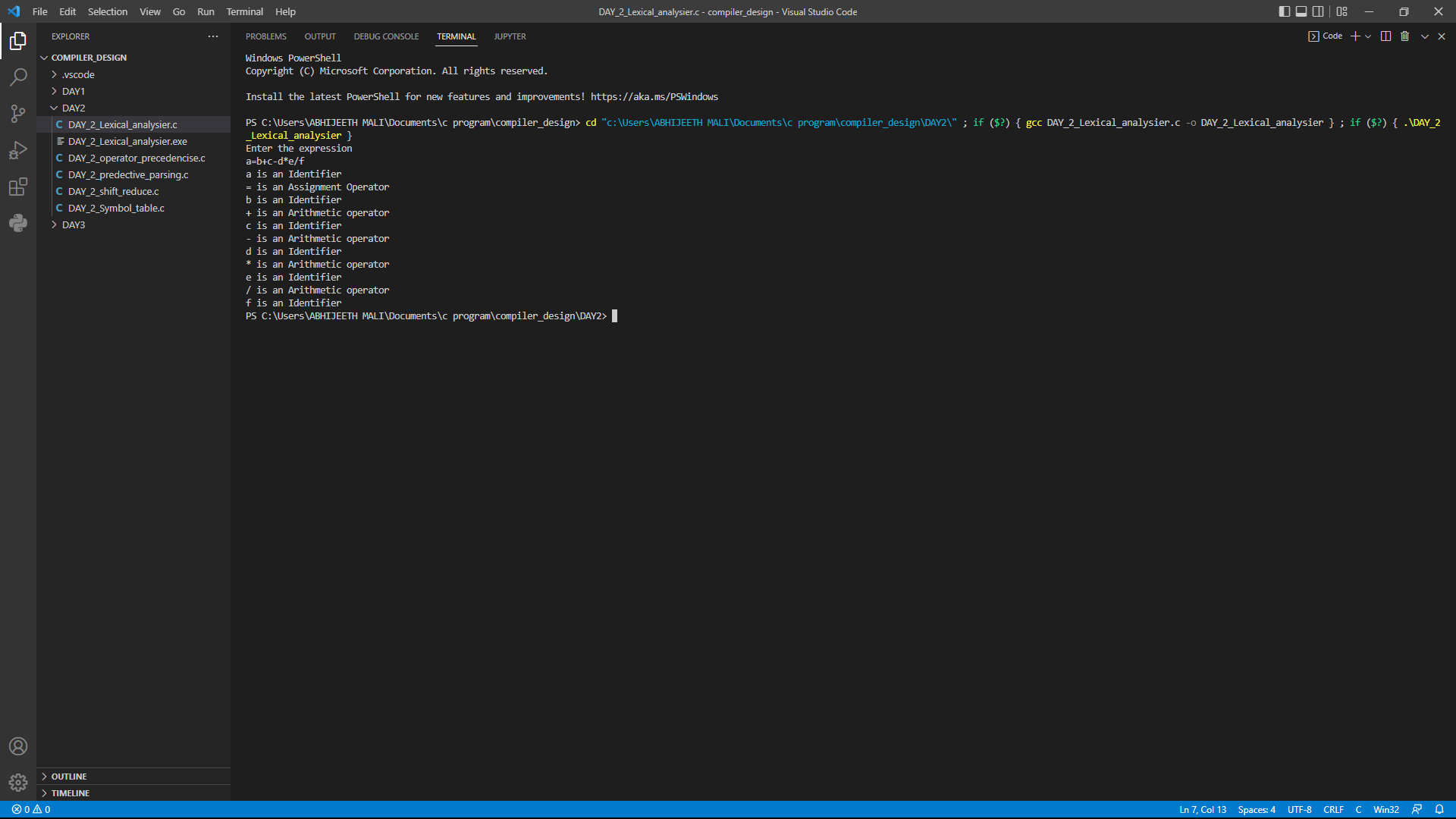
        }

    }

    return 0;

}

Output :



2. operator\_precedencise

Source code :

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

// function f to exit from the loop

// if given condition is not true

void f()

{

    printf("Not operator grammar");

    exit(0);

}

void main()

{

    char grm[20][20], c;

    // Here using flag variable,

    // considering grammar is not operator grammar

    int i, n, j = 2, flag = 0;

    // taking number of productions from user

    printf("\n\nEnter the number of production : ");

    scanf("%d", &n);

    for (i = 0; i < n; i++)

        scanf("%s", grm[i]);

    for (i = 0; i < n; i++)

    {

        c = grm[i][2];

        while (c != '\0')

        {

            if (grm[i][3] == '+' || grm[i][3] == '-' || grm[i][3] == '\*' || grm[i][3] == '/')

                flag = 1;

            else

            {

                flag = 0;

                f();

            }

            if (c == '$')

            {

                flag = 0;

                f();

            }

            c = grm[i][++j];

        }

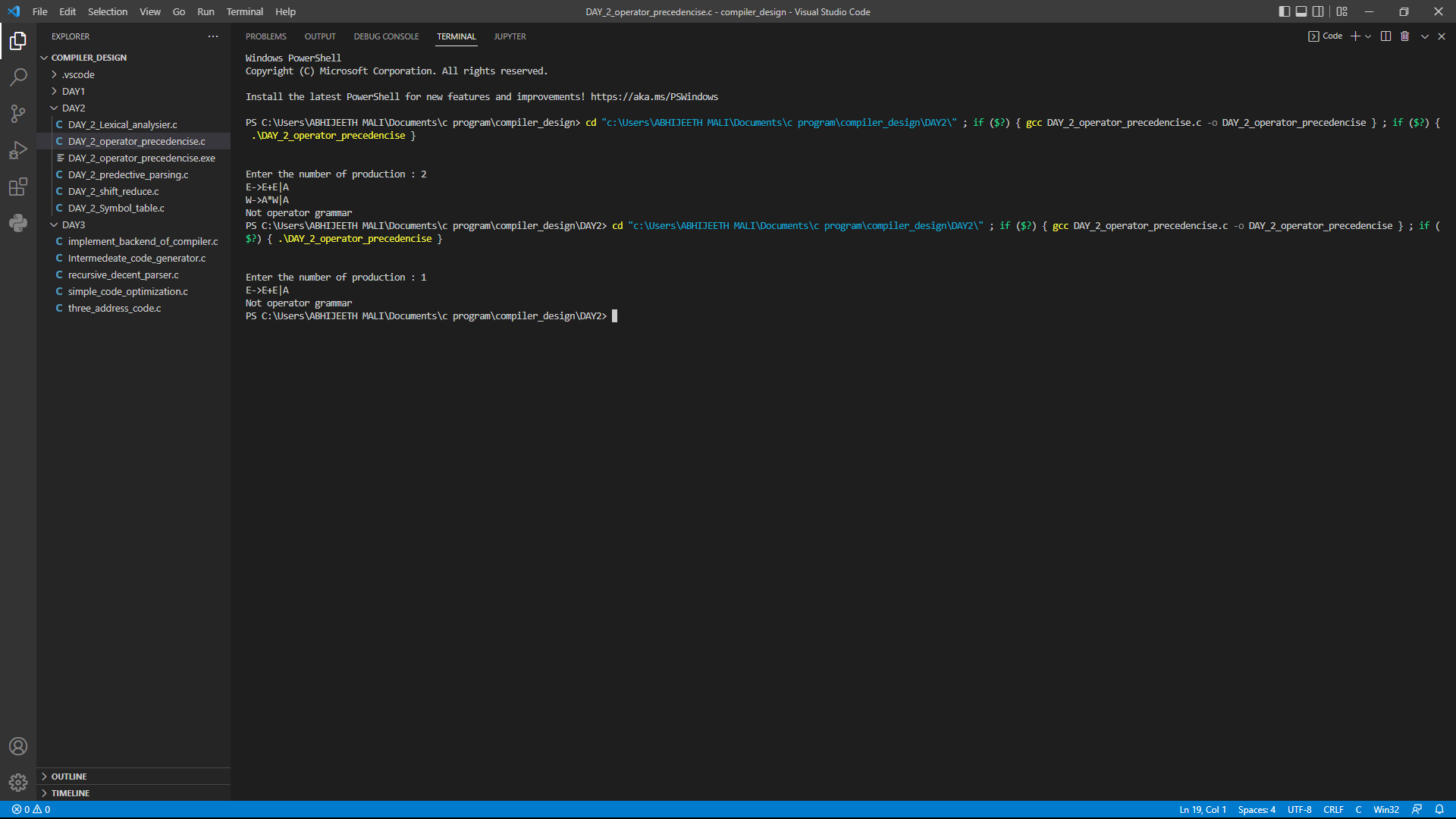
    }

    if (flag == 1)

        printf("Operator grammar");

}

Output :



3. predective\_parsing

Source code :

// Predictive Parser for the Grammar

// E -> E+E | E\*E | (E) | a

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

int isterm(char x)

{

    if ((x == 'a') || (x == '+') || (x == '\*') || (x == '(') || (x == ')') || (x == '$'))

        return 1;

    else

        return 0;

}

void main()

{

    char table[5][6][4] = {{"TA", "", "", "TA", "", ""},

                           {"", "+TA", "", "", "e", "e"},

                           {"FB","","","FB","",},

                           {"", "e", "\*FB", "", "e", "e"},

                           {"a", "", "", "(E)", "", ""}};

    char stack[20], input[20];

    int len, l, i, j, k, nt, t, flag;

    // clrscr();

    printf("Enter the input string to be parsed : ");

    scanf("%s", &input);

    l = strlen(input);

    input[l] = '$';

    input[l + 1] = '\0';

    stack[0] = '$';

    stack[1] = 'E';

    stack[2] = '\0';

    i = 1;

    j = 0;

    printf("\n\n");

    printf("------------------------------------------\n");

    printf("STACK\tINPUT\tOUTPUT\n");

    printf("------------------------------------------\n");

    printf("%s\t%s\n", stack, input);

    while (!(stack[i] == '$' && input[0] == '$'))

    {

        if ((stack[i] == '$') && (input[0] != '$'))

        {

            printf("Not a sentence\n");

            goto a;

        }

        else if (isterm(stack[i]) && (stack[i] != input[0]))

        {

            printf("Not a sentence\n");

            goto a;

        }

        else if (isterm(stack[i]) && (stack[i] == input[0]))

        {

            printf("%s\t%s\n", stack, input);

            stack[i] = '\0';

            i--;

            l = strlen(input);

            for (j = 0; j < l - 1; j++)

            {

                input[j] = input[j + 1];

            }

            input[l - 1] = '\0';

            flag = 0;

        }

        else

        {

            switch (stack[i])

            {

            case 'E':

                nt = 0;

                break;

            case 'A':

                nt = 1;

                break;

            case 'T':

                nt = 2;

                break;

            case 'B':

                nt = 3;

                break;

            case 'F':

                nt = 4;

                break;

            }

            switch (input[0])

            {

            case 'a':

                t = 0;

                break;

            case '+':

                t = 1;

                break;

            case '\*':

                t = 2;

                break;

            case '(':

                t = 3;

                break;

            case ')':

                t = 4;

                break;

            case '$':

                t = 5;

                break;

            }

            stack[i] = '\0';

            i--;

            l = strlen(table[nt][t]);

            if (l == 0)

            {

                printf("Not a sentence\n");

                goto a;

            }

            else if (strcmp(table[nt][t], "e") == 0)

            {

            }

            else

            {

                flag = 1;

                for (k = l - 1; k >= 0; k--)

                {

                    stack[++i] = table[nt][t][k];

                }

            }

        }

        if (flag == 0)

            printf("%s\t%s\n", stack, input);

        else

            printf("%s\t%s\t%s\n", stack, input, table[nt][t]);

    }

    printf("Sentence\n");

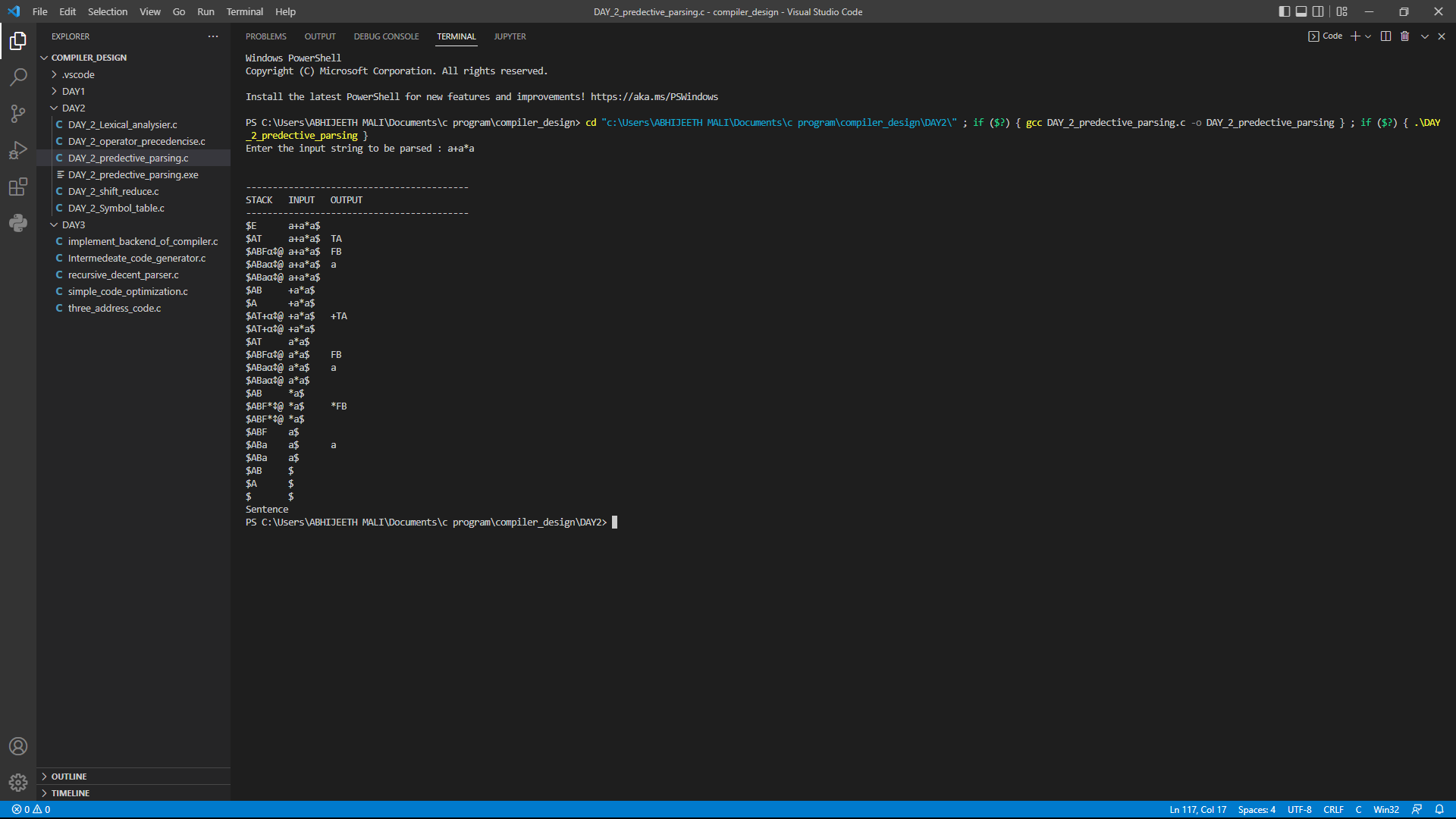
a:

    // getch();

    exit(0);

}

Output :



4. shift\_reduce

Source code :

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

// Rules can be E->2E2 , E->3E3 , E->4

void check()

{

    strcpy(ac,"REDUCE TO E -> ");

    for(z = 0; z < c; z++)

    {

        if(stk[z] == '4')

        {

            printf("%s4", ac);

            stk[z] = 'E';

            stk[z + 1] = '\0';

            //printing action

            printf("\n$%s\t%s$\t", stk, a);

        }

    }

    for(z = 0; z < c - 2; z++)

    {

        //checking for another production

        if(stk[z] == '2' && stk[z + 1] == 'E' &&

                                stk[z + 2] == '2')

        {

            printf("%s2E2", ac);

            stk[z] = 'E';

            stk[z + 1] = '\0';

            stk[z + 2] = '\0';

            printf("\n$%s\t%s$\t", stk, a);

            i = i - 2;

        }

    }

    for(z=0; z<c-2; z++)

    {

        //checking for E->3E3

        if(stk[z] == '3' && stk[z + 1] == 'E' &&

                                stk[z + 2] == '3')

        {

            printf("%s3E3", ac);

            stk[z]='E';

            stk[z + 1]='\0';

            stk[z + 1]='\0';

            printf("\n$%s\t%s$\t", stk, a);

            i = i - 2;

        }

    }

    return ; //return to main

}

//Driver Function

int main()

{

    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

    // a is input string

    strcpy(a,"32423");

    // strlen(a) will return the length of a to c

    c=strlen(a);

    // "SHIFT" is copied to act to be printed

    strcpy(act,"SHIFT");

    // This will print Labels (column name)

    printf("\nstack \t input \t action");

    printf("\n$\t%s$\t", a);

    // This will Run upto length of input string

    for(i = 0; j < c; i++, j++)

    {

        // Printing action

        printf("%s", act);

        // Pushing into stack

        stk[i] = a[j];

        stk[i + 1] = '\0';

        // Moving the pointer

        a[j]=' ';

        // Printing action

        printf("\n$%s\t%s$\t", stk, a);

        check();

    }

    check();

    if(stk[0] == 'E' && stk[1] == '\0')

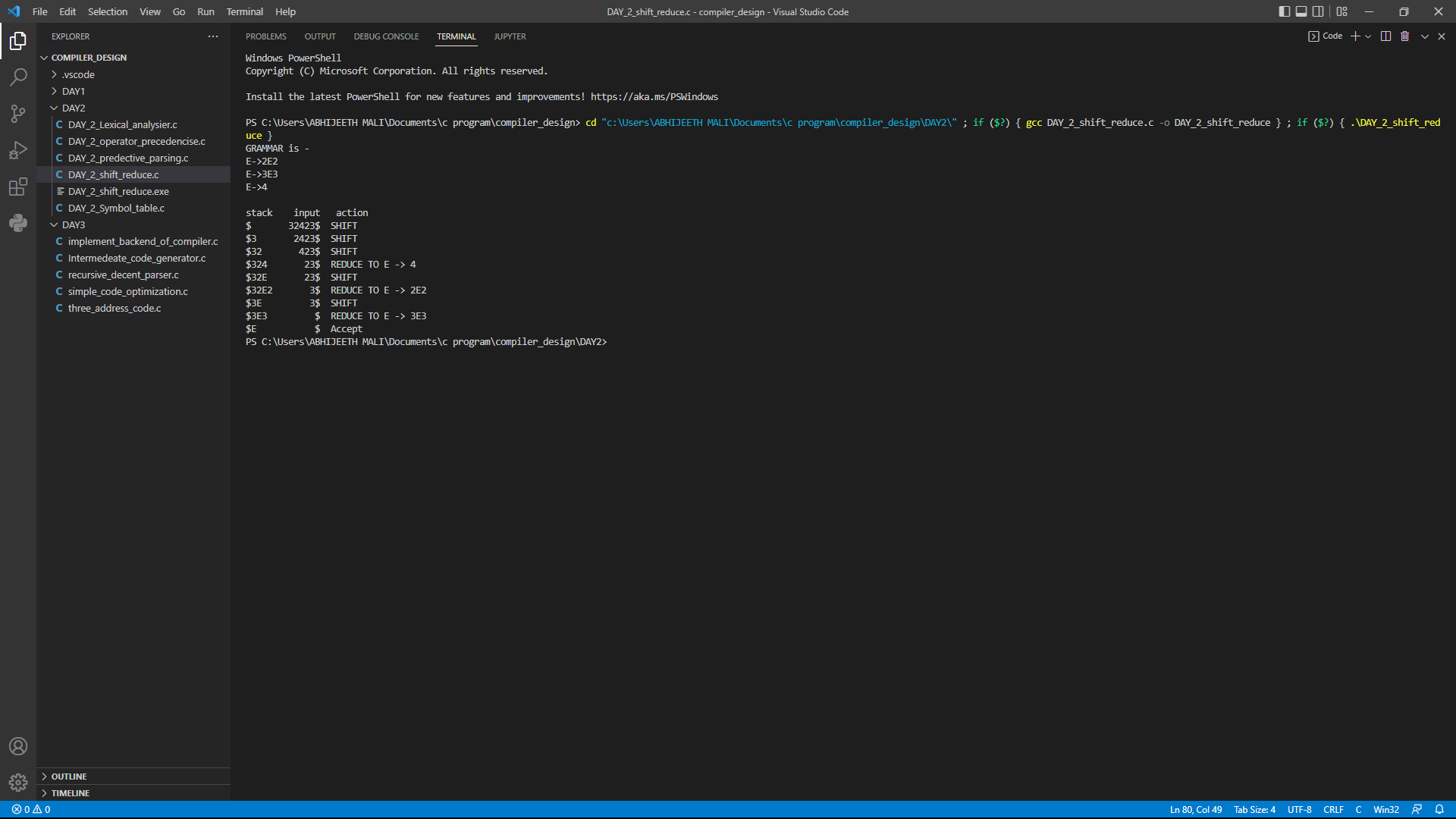
        printf("Accept\n");

    else //else reject

        printf("Reject\n");

}

Output :



5. Symbol\_table

Source code :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

int count = 0;

struct symboltable

{

    char label[20];

    int addr;

} symbol[50];

void insert();

int search(char \*);

void display();

void modify();

void main()

{

    int ch, val;

    char lab[10];

    do

    {

        printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");

        printf("\nEnter any choice from above : ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            insert();

            break;

        case 2:

            display();

            break;

        case 3:

            printf("\nenter the label : ");

            scanf("%s", lab);

            val = search(lab);

            if (val == 1)

                printf("\nlabel is found...\n");

            else

                printf("\nlabel is not found\n");

            break;

        case 4:

            modify();

            break;

        case 5:

            exit(0);

            break;

        }

    } while (ch < 5);

}

void insert()

{

    int val;

    char lab[10];

    printf("enter the label : ");

    scanf("%s", lab);

    val = search(lab);

    if (val == 1)

        printf("duplicate symbol\n\n");

    else

    {

        strcpy(symbol[count].label, lab);

        printf("enter the address : ");

        scanf("%d", &symbol[count].addr);

        count++;

    }

}

int search(char \*s)

{

    int count2 = 0, i;

    for (i = 0; i < count; i++)

    {

        if (strcmp(symbol[i].label, s) == 0)

            count2 = 1;

    }

    return count2;

}

void modify()

{

    int val, ad, i;

    char lab[10];

    printf("enter the label : ");

    scanf("%s", lab);

    val = search(lab);

    if (val == 0)

        printf("no such symbol...");

    else

    {

        printf("label is found...\n\n");

        printf("enter the address : ");

        scanf("%d", &ad);

        for (i = 0; i < count; i++)

        {

            if (strcmp(symbol[i].label, lab) == 0)

                symbol[i].addr = ad;

        }

    }

}

void display()

{

    int i;

    printf("-------------------------\n");

    printf("| LABEL |    ADDRESS    |\n");

    printf("|-------|---------------|\n");

    for (i = 0; i < count; i++)

        printf("| %s\t|\t%d\t|\n", symbol[i].label, symbol[i].addr);

    printf("-------------------------\n");

}

Output :

