CSA1465 – COMPILER DESIGN DAY 5 PROGRAMS

QUE 1

**Write a LEX program to count the frequency of the given word in a given sentence.**

**CODE:**

%{

#include <stdio.h>

#include <string.h>

int count = 0;

char target\_word[50];

%}

%%

[ \t\n]+ ; // skip whitespace

[a-zA-Z]+ { if(strcmp(yytext, target\_word) == 0) count++; }

.|\n ; // skip other characters

%%

int yywrap() {

return 1; // indicate end of input

}

int main() {

printf("Enter a sentence: ");

char sentence[1000];

fgets(sentence, sizeof(sentence), stdin);

printf("Enter the word to count: ");

scanf("%s", target\_word);

yy\_scan\_string(sentence);

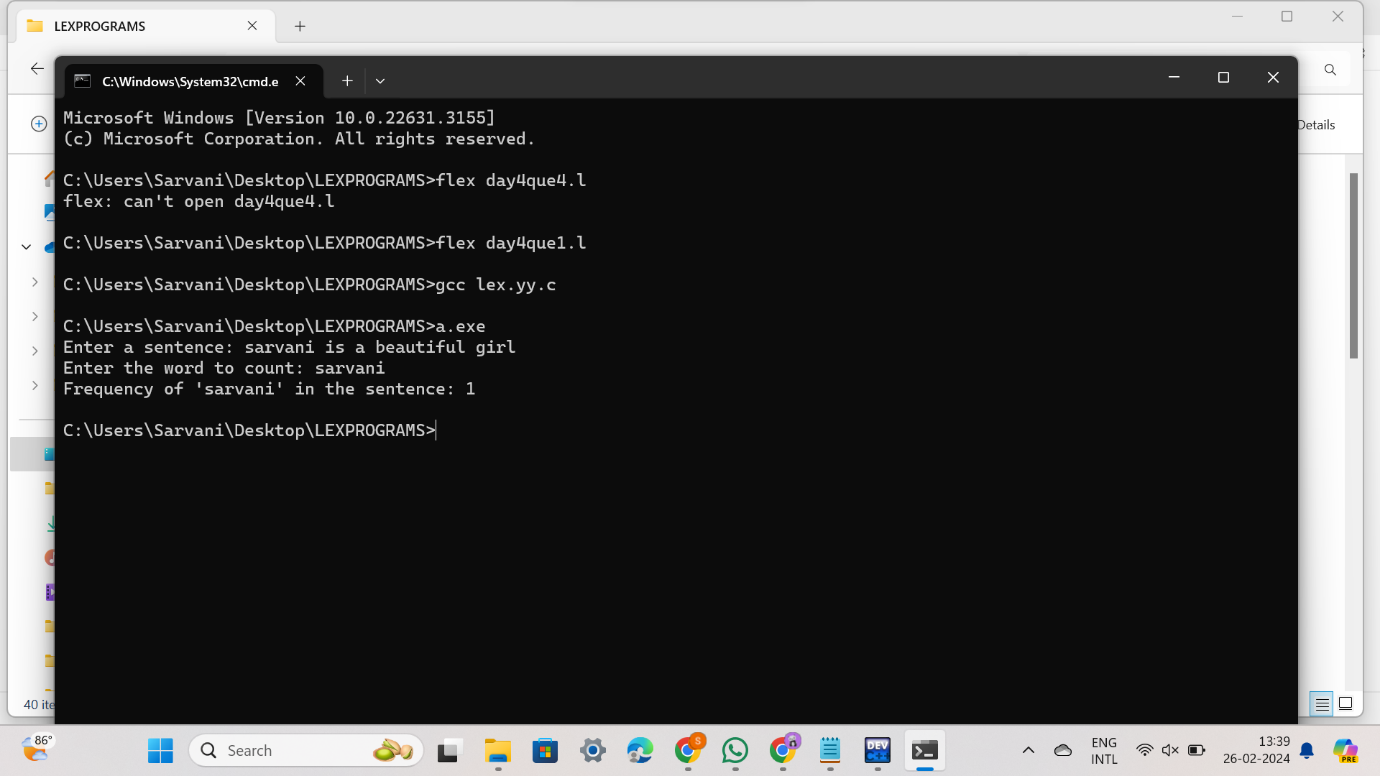
yylex();

printf("Frequency of '%s' in the sentence: %d\n", target\_word, count);

return 0;

}

**OUTPUT:**



**QUE 2**

**The lexical analyser should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, operators using C program.**

**CODE:**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int main()

{

int i,ic=0,m,cc=0,oc=0,j;

char b[30],operators[30],identifiers[30],constants[30];

printf("enter the string : ");

scanf("%[^\n]s",&b);

for(i=0;i<strlen(b);i++)

{

if(isspace(b[i]))

{

continue;

}

else if(isalpha(b[i]))

{

identifiers[ic] =b[i];

ic++;

}

else if(isdigit(b[i]))

{

m=(b[i]-'0');

i=i+1;

while(isdigit(b[i]))

{

m=m\*10 + (b[i]-'0');

i++;

}

i=i-1;

constants[cc]=m;

cc++;

}

else

{

if(b[i]=='\*')

{

operators[oc]='\*';

oc++;

}

else if(b[i]=='-')

{

operators[oc]='-';

oc++;

}

else if(b[i]=='+')

{

operators[oc]='+';

oc++;

}

else if(b[i]=='=')

{

operators[oc]='=';

oc++;

}

}

}

printf(" identifiers : ");

for(j=0;j<ic;j++)

{

printf("%c ",identifiers[j]);

}

printf("\n constants : ");

for(j=0;j<cc;j++)

{

printf("%d ",constants[j]);

}

printf("\n operators : ");

for(j=0;j<oc;j++)

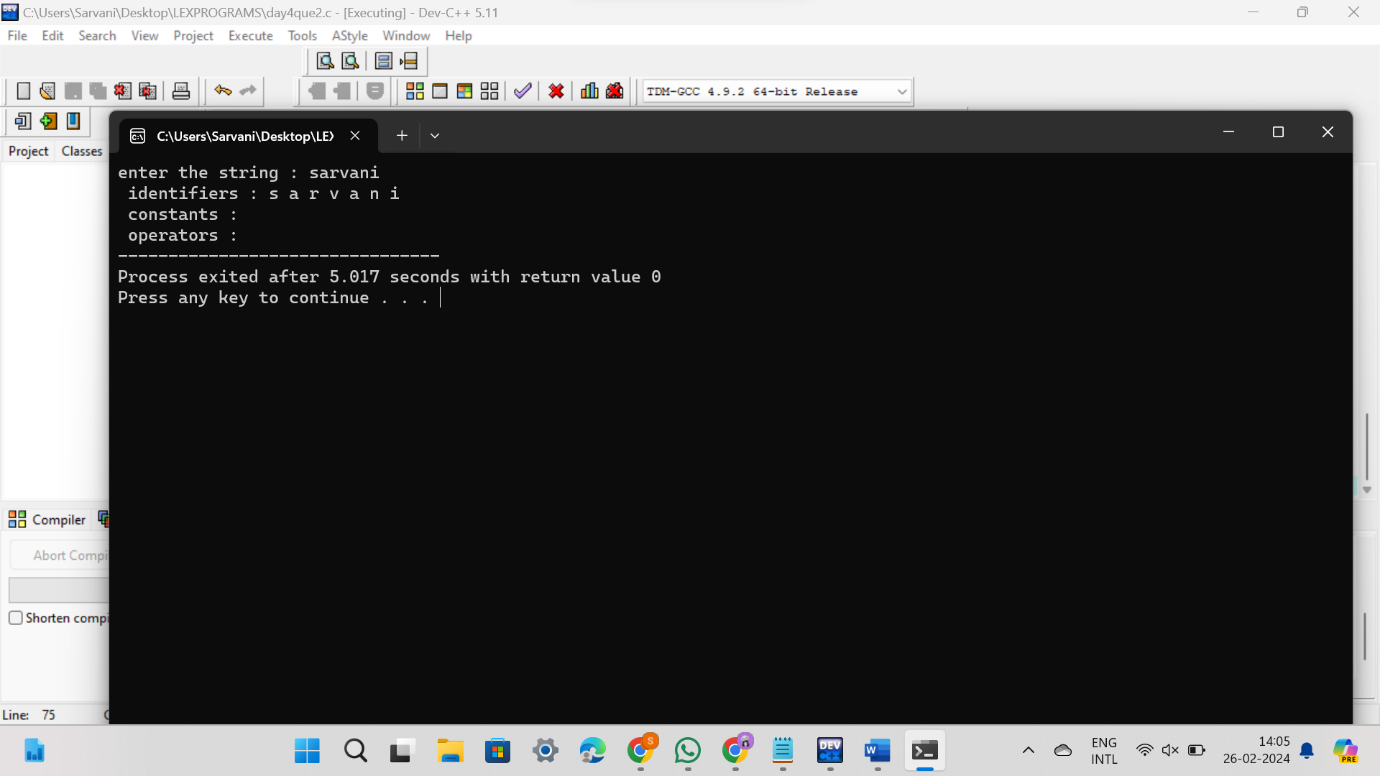
{

printf("%c ",operators[j]);

}

}

**OUTPUT:**



**QUE 3**

**Extend the lexical Analyzer to Check comments, dened as follows in C: a) A comment begins with // and includes all characters until the end of that line. b) A comment begins with /\* and includes all characters through the next occurrence of the character sequence \*/Develop a lexical Analyzer to identify whether a given line is a comment or not.**

**CODE:**

#include<stdio.h>

#include<conio.h>

int main()

{

char com[30];

int i=2,a=0;

printf("\n Enter comment:");

gets(com);

if(com[0]=='/')

{

if(com[1]=='/')

printf("\n It is a comment");

else if(com[1]=='\*')

{

for(i=2;i<=30;i++)

{

if(com[i]=='\*'&&com[i+1]=='/')

{

printf("\n It is a comment");

a=1;

break;

}

else

continue;

}

if(a==0)

printf("\n It is not a comment");

}

else

printf("\n It is not a comment");

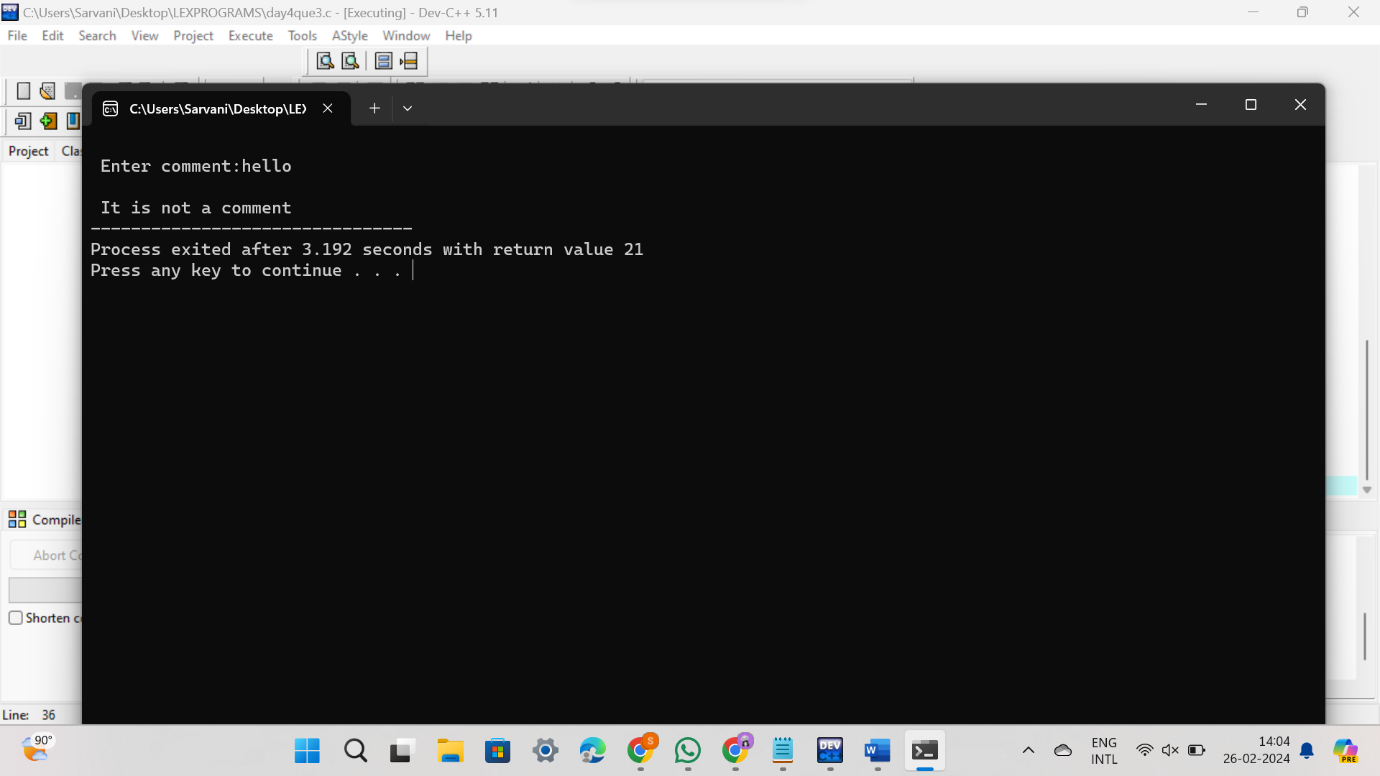
}

else

printf("\n It is not a comment");

}

**OUTPUT:**



**QUE 4**

**Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular Arithmetic operators .**

**CODE:**

#include<stdio.h>

#include<conio.h>

int main()

{

char s[5];

printf("\n Enter any operator:");

gets(s);

switch(s[0])

{

case'>':

if(s[1]=='=')

printf("\n Greater than or equal");

else

printf("\n Greater than");

break;

case'<':

if(s[1]=='=')

printf("\n Less than or equal");

else

printf("\nLess than");

break;

case'=':

if(s[1]=='=')

printf("\nEqual to");

else

printf("\nAssignment");

break;

case'!':

if(s[1]=='=')

printf("\nNot Equal");

else

printf("\n Bit Not");

break;

case'&':

if(s[1]=='&')

printf("\nLogical AND");

else

printf("\n Bitwise AND");

break;

case'|':

if(s[1]=='|')

printf("\nLogical OR");

else

printf("\nBitwise OR");

break;

case'+':

printf("\n Addition");

break;

case'-':

printf("\nSubstraction");

break;

case'\*':

printf("\nMultiplication");

break;

case'/':

printf("\nDivision");

break;

case'%':

printf("Modulus");

break;

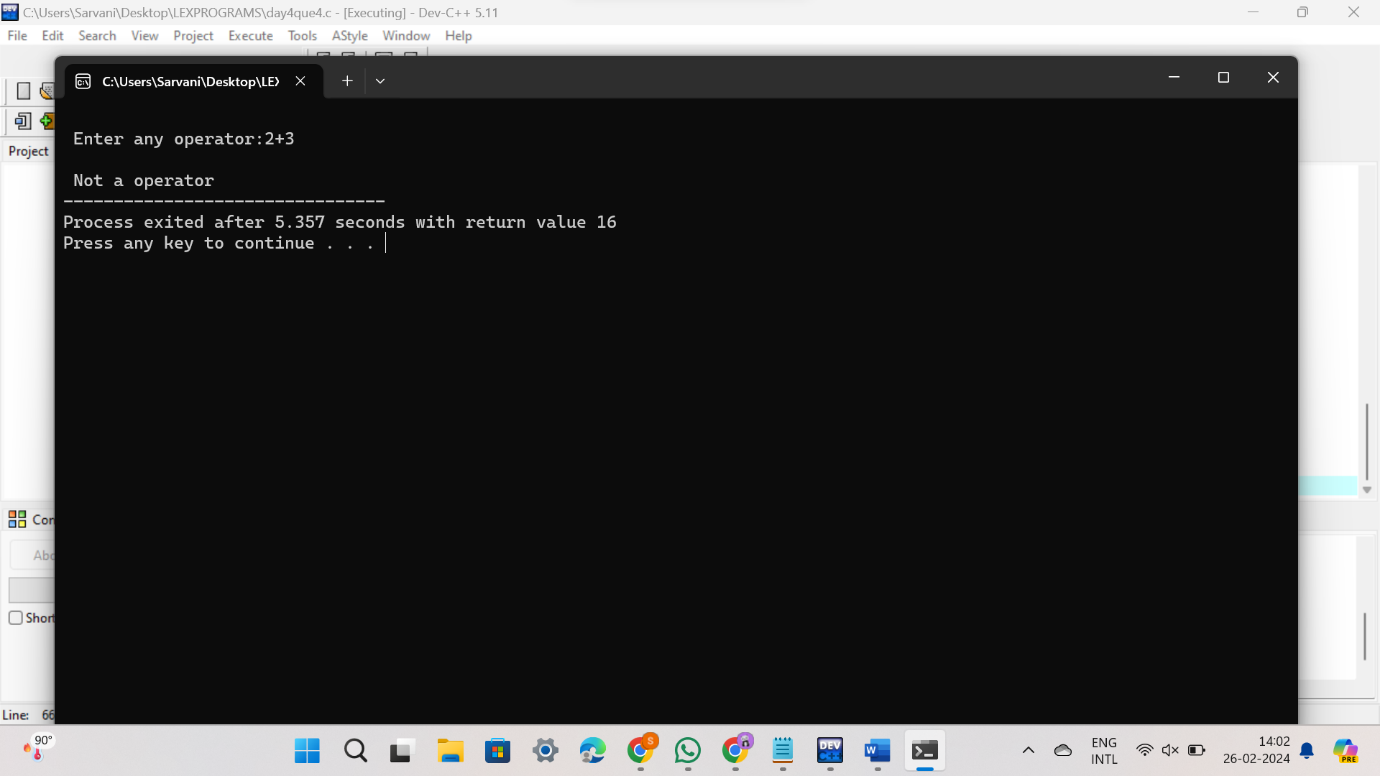
default:

printf("\n Not a operator");

}

}

**OUTPUT:**



**QUE 5**

**Design a lexical Analyzer to find the number of whitespaces and newline characters.**

**CODE:**

%{

#include <stdio.h>

int space\_count = 0;

int newline\_count = 0;

int yywrap(void) {

return 1; // indicate end of input

}

%}

%%

[ \t]+ { space\_count += yyleng; }

\n { newline\_count++; }

. ; // skip other characters

%%

int main() {

printf("Enter a text (Ctrl+D to end input on Unix/Linux, Ctrl+Z on Windows):\n");

yywrap(); // invoke yywrap to start processing

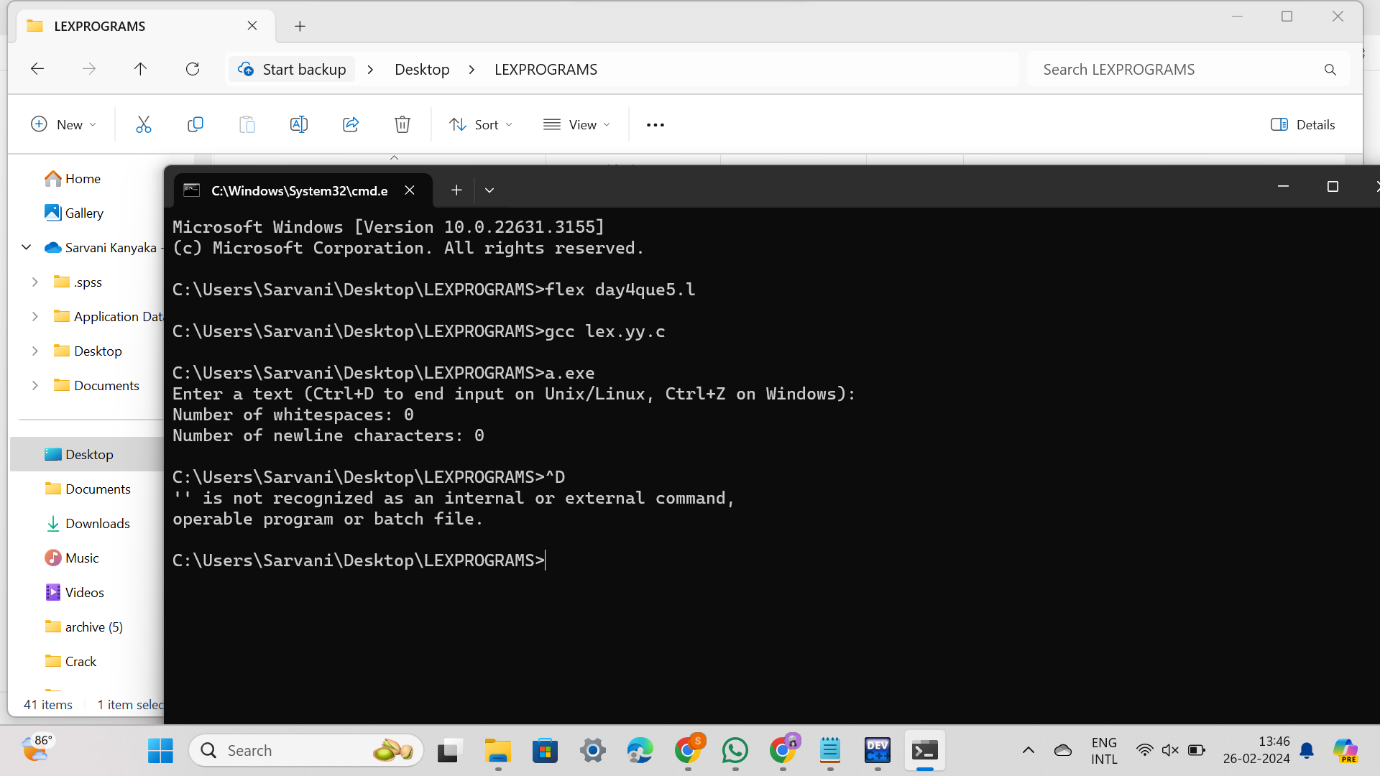
printf("Number of whitespaces: %d\n", space\_count);

printf("Number of newline characters: %d\n", newline\_count);

return 0;

}

**OUTPUT:**



**QUE 6**

**Develop a lexical Analyzer to test whether a given identifier is valid or not.**

**CODE:**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

int main()

{

char a[10];

int flag, i=1;

printf("\n Enter an identifier:");

gets(a);

if(isalpha(a[0]))

flag=1;

else

printf("\n Not a valid identifier");

while(a[i]!='\0')

{

if(!isdigit(a[i])&&!isalpha(a[i]))

{

flag=0;

break;

} i++;

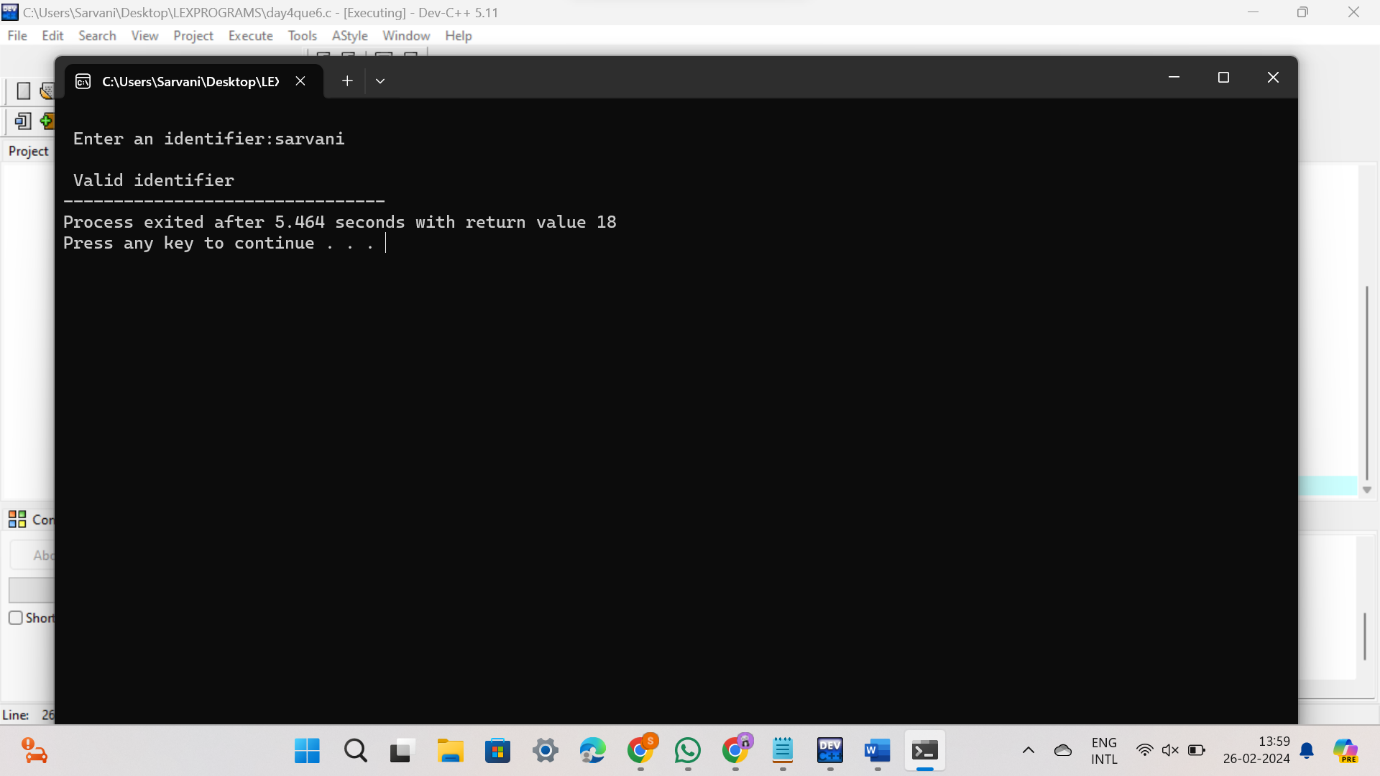
}

if(flag==1)

printf("\n Valid identifier");

}

**OUTPUT:**



**QUE 7**

**Implement a C program to eliminate left recursion.**

**CODE:**

#include<stdio.h>

#include<string.h>

int main() {

char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];

int i=0,j=0,flag=0,consumed=0;

printf("Enter the productions: ");

scanf("%1s->%s",l,r);

printf("%s",r);

while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r)) {

if(temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++],"%s->%s%s'\0",l,temp+1,l);

}

else

sprintf(productions[i++],"%s'->%s%s'\0",l,temp,l);

consumed += strlen(temp)+1;

}

if(flag == 1) {

sprintf(productions[i++],"%s->e\0",l);

printf("The productions after eliminating Left Recursion are:\n");

for(j=0;j<i;j++)

printf("%s\n",productions[j]);

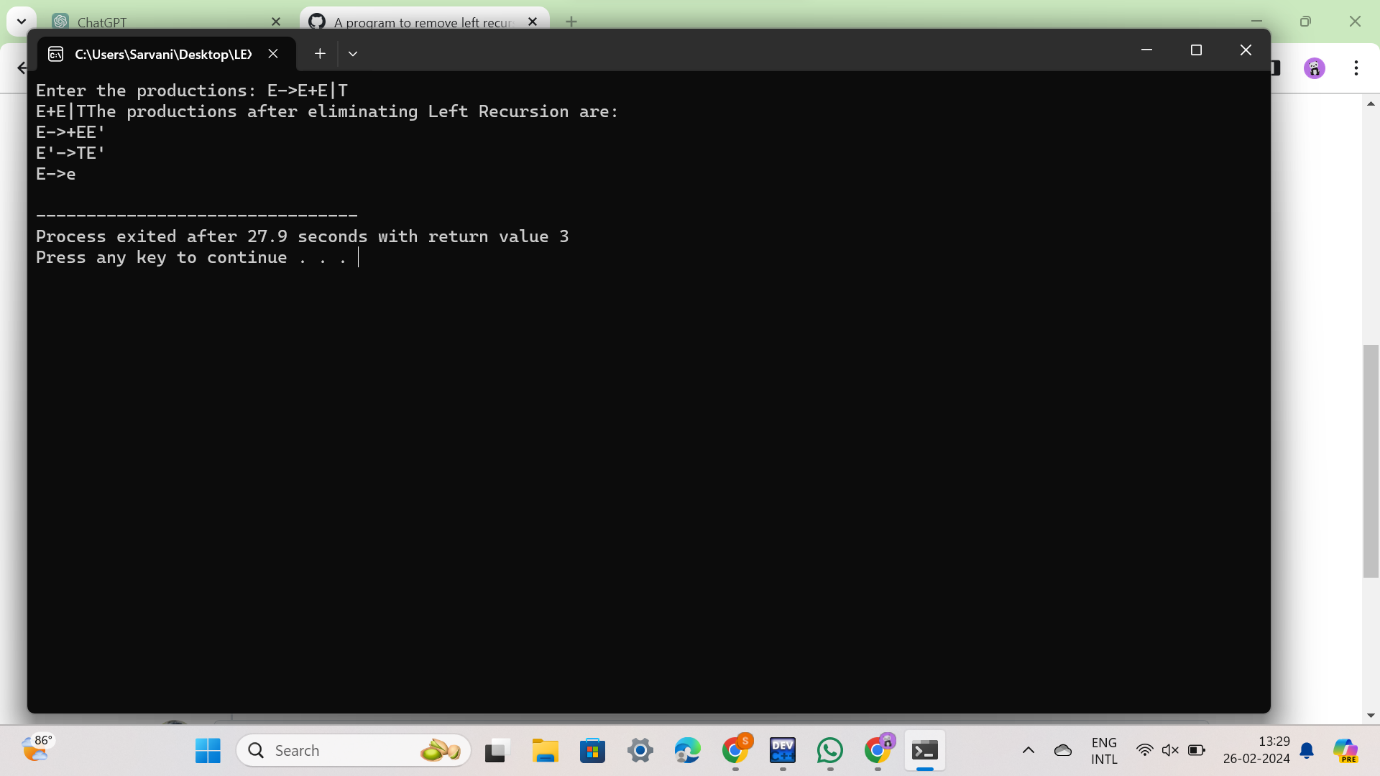
}

else

printf("The Given Grammar has no Left Recursion");

}

**OUTPUT:**



**QUE 8**

**Implement a C program to eliminate left factoring.**

**CODE:**

#include<stdio.h>

#include<string.h>

int main()

{

char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];

int i,j=0,k=0,l=0,pos;

printf("Enter Production : A->");

gets(gram);

for(i=0;gram[i]!='|';i++,j++)

part1[j]=gram[i];

part1[j]='\0';

for(j=++i,i=0;gram[j]!='\0';j++,i++)

part2[i]=gram[j];

part2[i]='\0';

for(i=0;i<strlen(part1)||i<strlen(part2);i++){

if(part1[i]==part2[i]){

modifiedGram[k]=part1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;part1[i]!='\0';i++,j++){

newGram[j]=part1[i];

}

newGram[j++]='|';

for(i=pos;part2[i]!='\0';i++,j++){

newGram[j]=part2[i];

}

modifiedGram[k]='X';

modifiedGram[++k]='\0';

newGram[j]='\0';

printf("\nGrammar Without Left Factoring : : \n");

printf(" A->%s",modifiedGram);

printf("\n X->%s\n",newGram);

}

**OUTPUT:**

