**CSA1465 – COMPILER DESIGN DAY 5 – LAB PROGRAMS**

**QUE 1**

**Implement a C program to perform symbol table operations.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 100

struct Symbol {

char name[50];

int value;

};

struct Symbol symbolTable[SIZE];

int symbolCount = 0;

void insertSymbol(char name[], int value) {

if (symbolCount < SIZE) {

strcpy(symbolTable[symbolCount].name, name);

symbolTable[symbolCount].value = value;

symbolCount++;

printf("Symbol inserted successfully.\n");

} else {

printf("Symbol table is full. Cannot insert more symbols.\n");

}

}

int searchSymbol(char name[]) {

for (int i = 0; i < symbolCount; i++) {

if (strcmp(symbolTable[i].name, name) == 0) {

return i;

}

}

return -1;

}

void deleteSymbol(char name[]) {

int index = searchSymbol(name);

if (index != -1) {

for (int i = index; i < symbolCount - 1; i++) {

strcpy(symbolTable[i].name, symbolTable[i + 1].name);

symbolTable[i].value = symbolTable[i + 1].value;

}

symbolCount--;

printf("Symbol deleted successfully.\n");

} else {

printf("Symbol not found.\n");

}

}

void displaySymbolTable() {

printf("Symbol Table:\n");

printf("Name\tValue\n");

for (int i = 0; i < symbolCount; i++) {

printf("%s\t%d\n", symbolTable[i].name, symbolTable[i].value);

}

}

int main() {

insertSymbol("x", 10);

insertSymbol("y", 20);

insertSymbol("z", 30);

displaySymbolTable();

printf("\n");

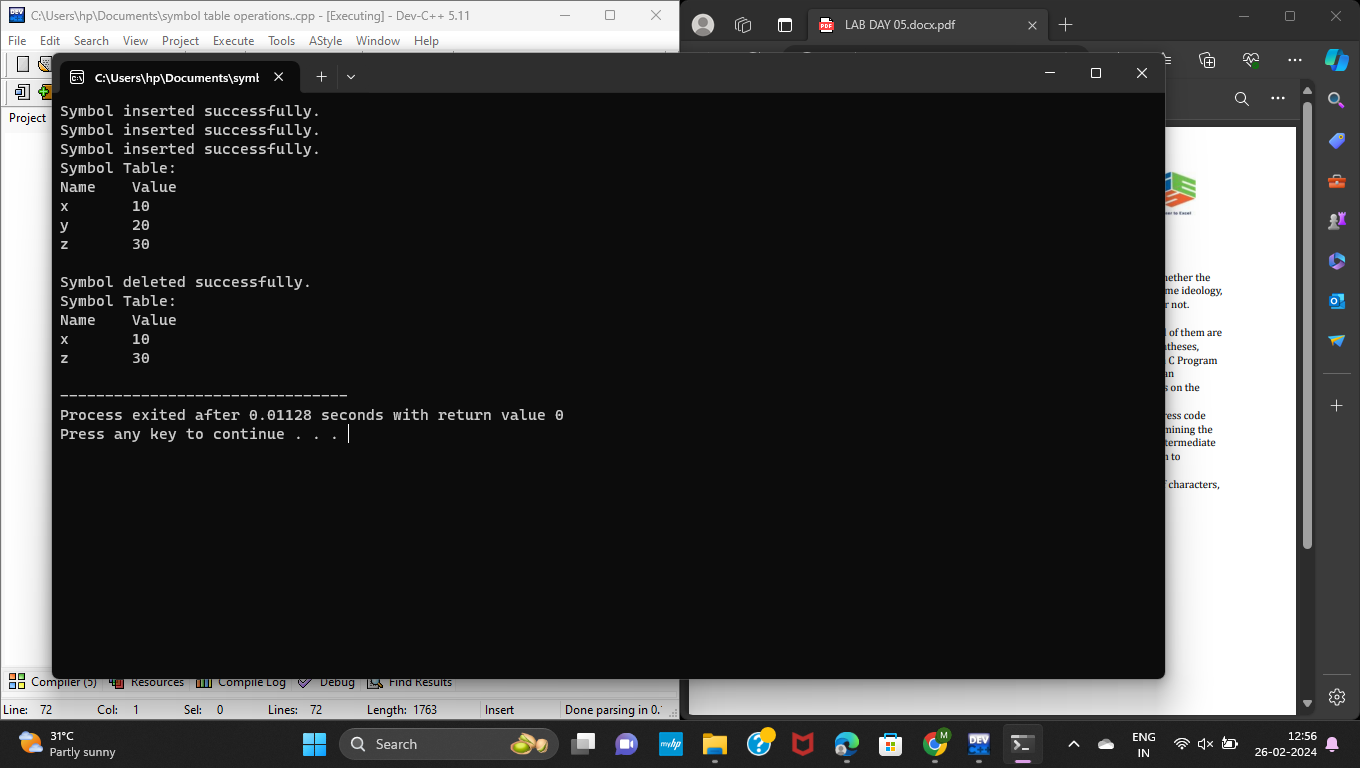
deleteSymbol("y");

displaySymbolTable();

return 0;

}

**OUTPUT:**

****

**QUE 2**

**All languages have Grammar. When people frame a sentence, we usually say whether the sentence is framed as per the rules of the Grammar or Not. Similarly use the same ideology, implement to check whether the given input string is satisfying the grammar or not.**

**CODE:**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdbool.h>**

**bool checkSentence(char str[])**

**{**

**int len = strlen(str);**

**if (str[0] < 'A' || str[0] > 'Z')**

**return false;**

**if (str[len - 1] != '.')**

**return false;**

**int prev\_state = 0, curr\_state = 0;**

**int index = 1;**

**while (str[index])**

**{**

**if (str[index] >= 'A' && str[index] <= 'Z')**

**curr\_state = 0;**

**else if (str[index] == ' ')**

**curr\_state = 1;**

**else if (str[index] >= 'a' && str[index] <= 'z')**

**curr\_state = 2;**

**else if (str[index] == '.')**

**curr\_state = 3;**

**if (prev\_state == curr\_state && curr\_state != 2)**

**return false;**

**if (prev\_state == 2 && curr\_state == 0)**

**return false;**

**if (curr\_state == 3 && prev\_state != 1)**

**return (str[index + 1] == '\0');**

**index++;**

**prev\_state = curr\_state;**

**}**

**return false;**

**}**

**int main()**

**{**

**char \*str[] = { "I love cinema.", "The vertex is S.",**

**"I am single.", "My name is KG.",**

**"I lovE cinema.", "GeeksQuiz. is a quiz site.",**

**"I love Geeksquiz and Geeksforgeeks.",**

**" You are my friend.", "I love cinema" };**

**int str\_size = sizeof(str) / sizeof(str[0]);**

**int i = 0;**

**for (i = 0; i < str\_size; i++)**

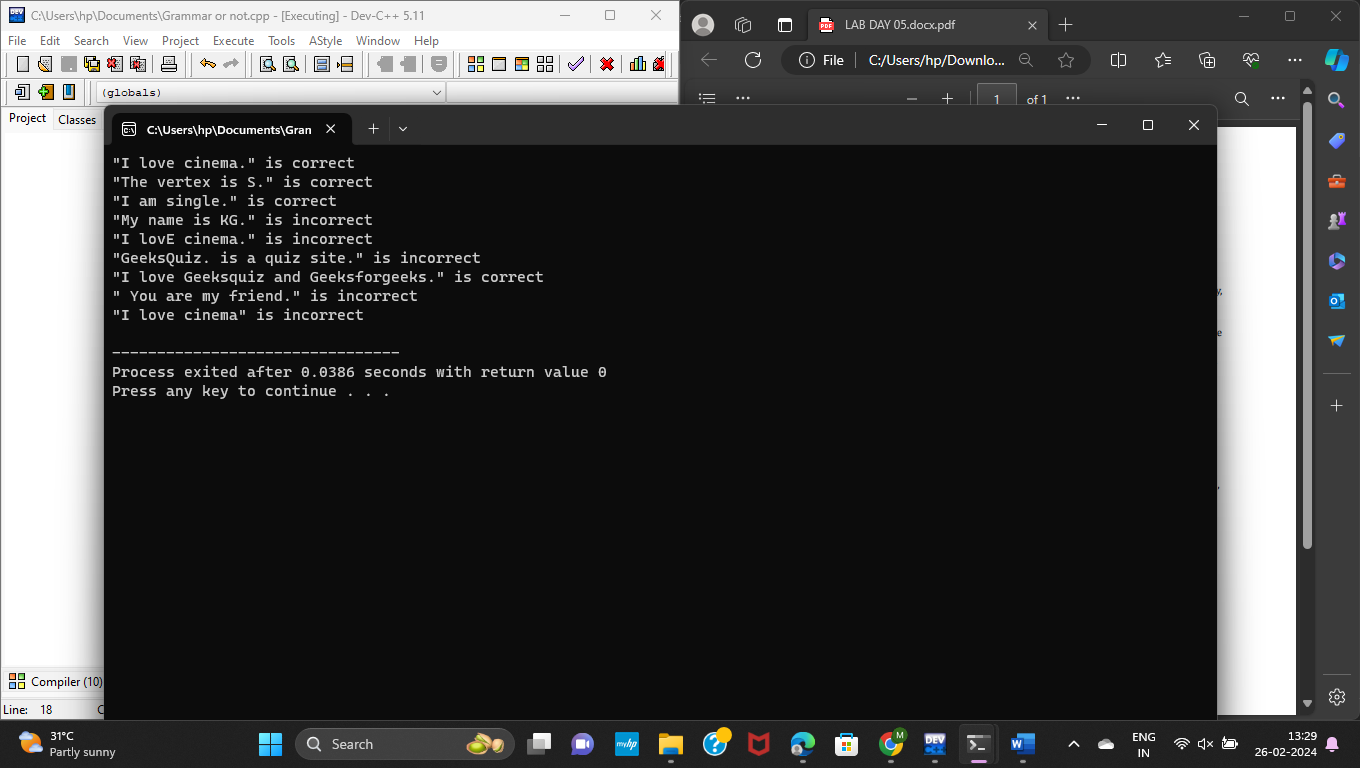
**checkSentence(str[i])? printf("\"%s\" is correct \n", str[i]):**

**printf("\"%s\" is incorrect \n", str[i]);**

**return 0;**

**}**

**Output:**

****

**3. Write a C program to construct recursive descent parsing.**

**CODE:**

**#include <stdio.h>**

**#include <string.h>**

**#define SUCCESS 1**

**#define FAILED 0**

**int E(), Edash(), T(), Tdash(), F();**

**const char \*cursor;**

**char string[64];**

**int main()**

**{**

**puts("Enter the string");**

**sscanf("i+(i+i)\*i", "%s", string);**

**cursor = string;**

**puts("");**

**puts("Input Action");**

**puts("--------------------------------");**

**if (E() && \*cursor == '\0') {**

**puts("--------------------------------");**

**puts("String is successfully parsed");**

**return 0;**

**} else {**

**puts("--------------------------------");**

**puts("Error in parsing String");**

**return 1;**

**}**

**}**

**int E()**

**{**

**printf("%-16s E -> T E'\n", cursor);**

**if (T()) {**

**if (Edash())**

**return SUCCESS;**

**else**

**return FAILED;**

**} else**

**return FAILED;**

**}**

**int Edash()**

**{**

**if (\*cursor == '+') {**

**printf("%-16s E' -> + T E'\n", cursor);**

**cursor++;**

**if (T()) {**

**if (Edash())**

**return SUCCESS;**

**else**

**return FAILED;**

**} else**

**return FAILED;**

**} else {**

**printf("%-16s E' -> $\n", cursor);**

**return SUCCESS;**

**}**

**}**

**int T()**

**{**

**printf("%-16s T -> F T'\n", cursor);**

**if (F()) {**

**if (Tdash())**

**return SUCCESS;**

**else**

**return FAILED;**

**} else**

**return FAILED;**

**}**

**int Tdash()**

**{**

**if (\*cursor == '\*') {**

**printf("%-16s T' -> \* F T'\n", cursor);**

**cursor++;**

**if (F()) {**

**if (Tdash())**

**return SUCCESS;**

**else**

**return FAILED;**

**} else**

**return FAILED;**

**} else {**

**printf("%-16s T' -> $\n", cursor);**

**return SUCCESS;**

**}**

**}**

**int F()**

**{**

**if (\*cursor == '(') {**

**printf("%-16s F -> ( E )\n", cursor);**

**cursor++;**

**if (E()) {**

**if (\*cursor == ')') {**

**cursor++;**

**return SUCCESS;**

**} else**

**return FAILED;**

**} else**

**return FAILED;**

**} else if (\*cursor == 'i') {**

**cursor++;**

**printf("%-16s F ->i\n", cursor);**

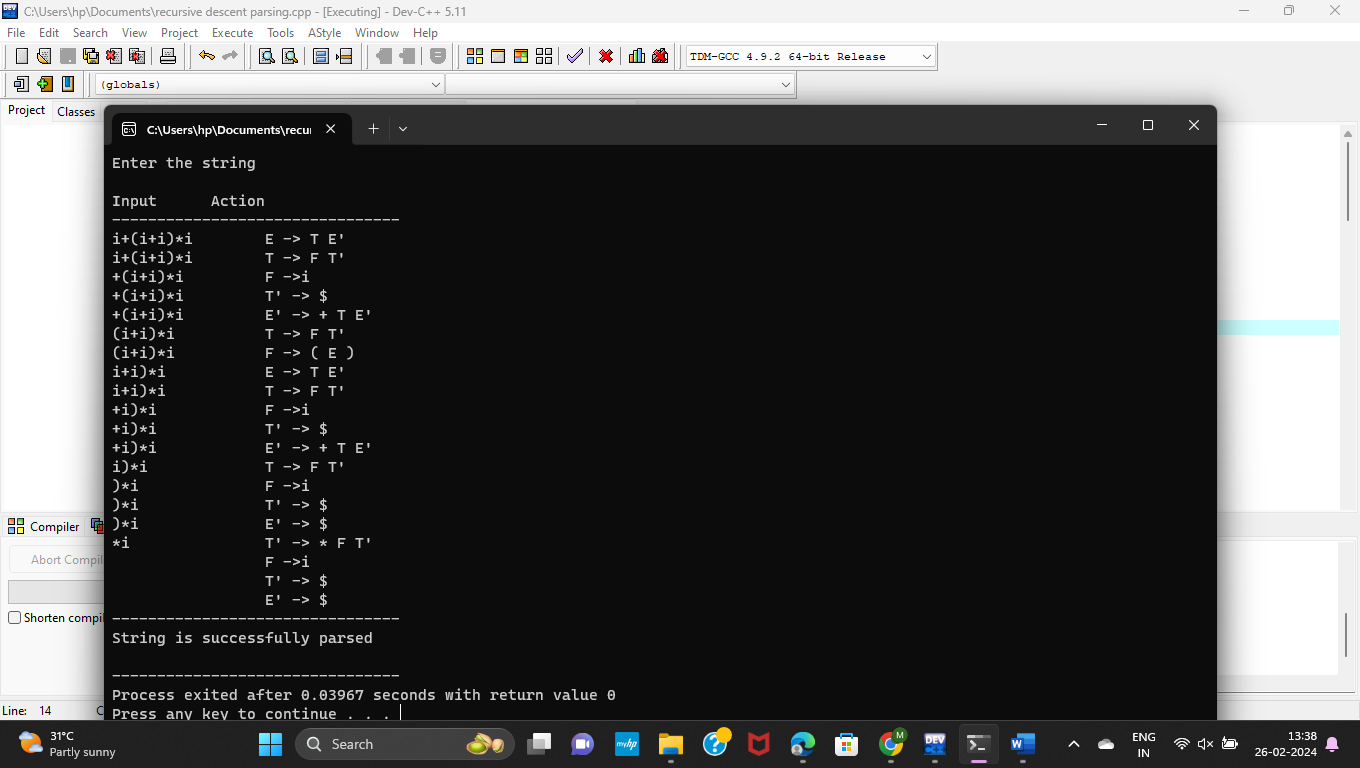
**return SUCCESS;**

**} else**

**return FAILED;**

**}**

**OUTPUT:**

****

**QUE 4**

**In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS? All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations.**

**CODE:**

#include<conio.h>

#include<stdio.h>

char arr[18][3] ={{'E', '+', 'F'}, {'E', '\*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'},

{'E', '$', 'F'}, {'F', '+', 'F'}, {'F', '\*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'},

{'F', '$', 'F'}, {'T', '+', 'F'}, {'T', '\*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},

{'T', '$', 'F'},

};

char prod[6] = "EETTFF";

char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0', '\0'}, {'T', '\*', 'F'}, {'F', '\0', '\0'}, {'(', 'E', ')'}, {'i', '\0', '\0'},};

char stack [5][2];

int top = -1;

void install(char pro, char re) {

int i;

for (i = 0; i < 18; ++i) {

if (arr[i][0] == pro && arr[i][1] == re) {

}

}

++top;

arr[i][2] = 'T';

stack[top][0] = pro;

stack[top][1] = re;

}

int main() {

int i = 0, j;

char pro, re, pri = ' ';

for (i = 0; i < 6; ++i) {

for (j = 2; j >= 0; --j) {

if (res[i][j] == '+' || res[i][j] == '\*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$') {

install(prod[i], res[i][j]);

break;

} else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') {

if (res[i][j - 1] == '+' || res[i][j - 1] == '\*' || res[i][j - 1] == '(' || res[i][j -

1] == ')' || res[i][j - 1] == 'i' || res[i][j - 1] == '$') {

install(prod[i], res[i][j - 1]);

break;

}

}

}

}

while (top >= 0) {

pro = stack[top][0];

re = stack[top][1];

--top;

for (i = 0; i < 6; ++i) {

for (j = 2; j >= 0; --j) {

if (res[i][0] == pro && res[i][0] != prod[i]) {

install(prod[i], re);

break;

} else if (res[i][0] != '\0') break;

}

}

}

for (i = 0; i < 18; ++i) {

printf("\n\t");

for (j = 0; j < 3; ++j)

printf("%c\t", arr[i][j]);

}

printf("\n\n");

for (i = 0; i < 18; ++i) {

if (pri != arr[i][0]) {

pri = arr[i][0];

printf("\n\t%c -> ", pri);

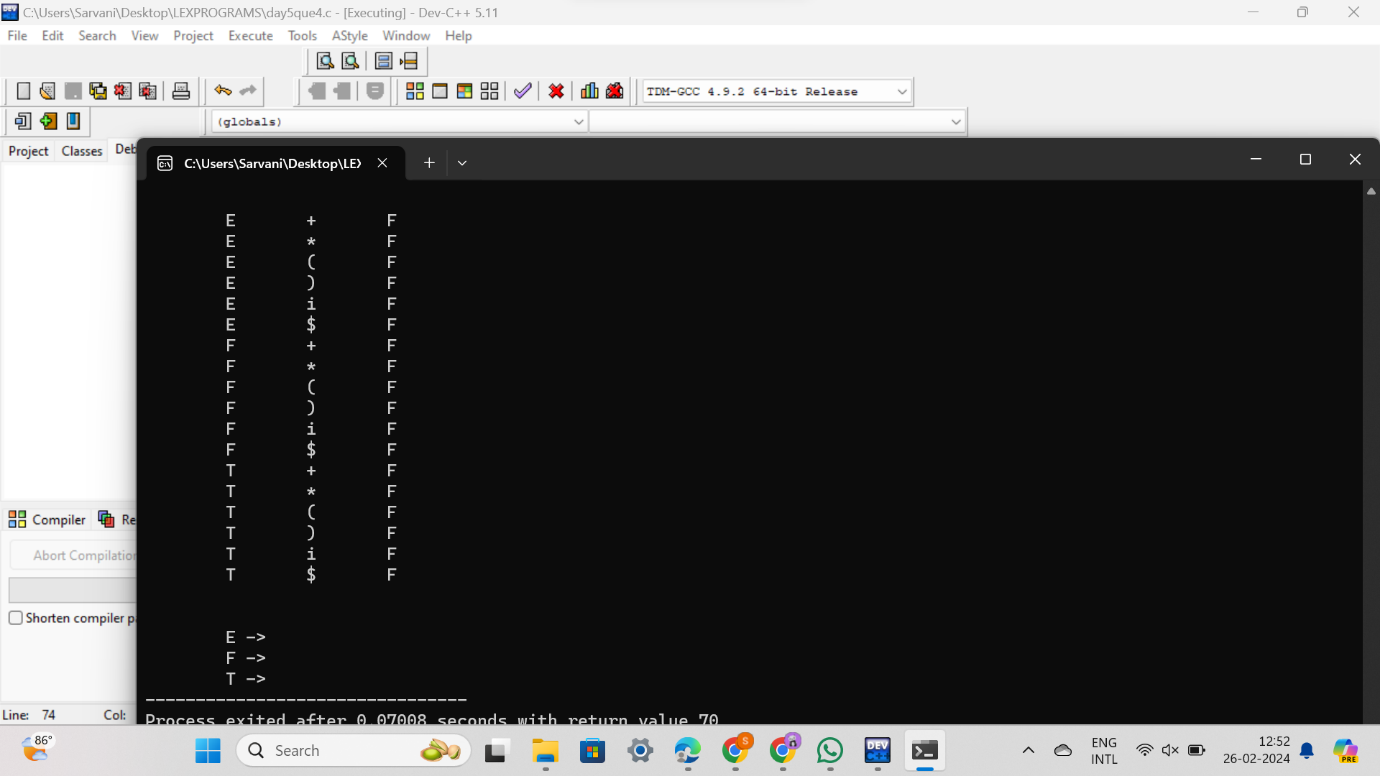
}

if (arr[i][2] == 'T')

printf("%c ", arr[i][1]);}

}

**OUTPUT:**



**QUE 5**

**The main function of the Intermediate code generation is producing three address code statements for a given input expression. The three address codes help in determining the sequence in which operations are actioned by the compiler. The key work of Intermediate code generators is to simplify the process of Code Generator. Write a C Program to Generate the Three address code representation for the given input statement.**

**CODE:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct three

{

char data[10],temp[7];

}s[30];

int main()

{

char d1[7],d2[7]="t";

int i=0,j=1,len=0;

FILE \*f1,\*f2;

//clrscr();

f1=fopen("sum.txt","r");

f2=fopen("out.txt","w");

while(fscanf(f1,"%s",s[len].data)!=EOF)

len++;

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

strcpy(d1,"");

strcpy(d2,"t");

if(!strcmp(s[3].data,"+"))

{

fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

else if(!strcmp(s[3].data,"-"))

{

fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

for(i=4;i<len-2;i+=2)

{

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

if(!strcmp(s[i+1].data,"+"))

fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);

else if(!strcmp(s[i+1].data,"-"))

fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);

strcpy(d1,"");

strcpy(d2,"t");

j++;

}

fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);

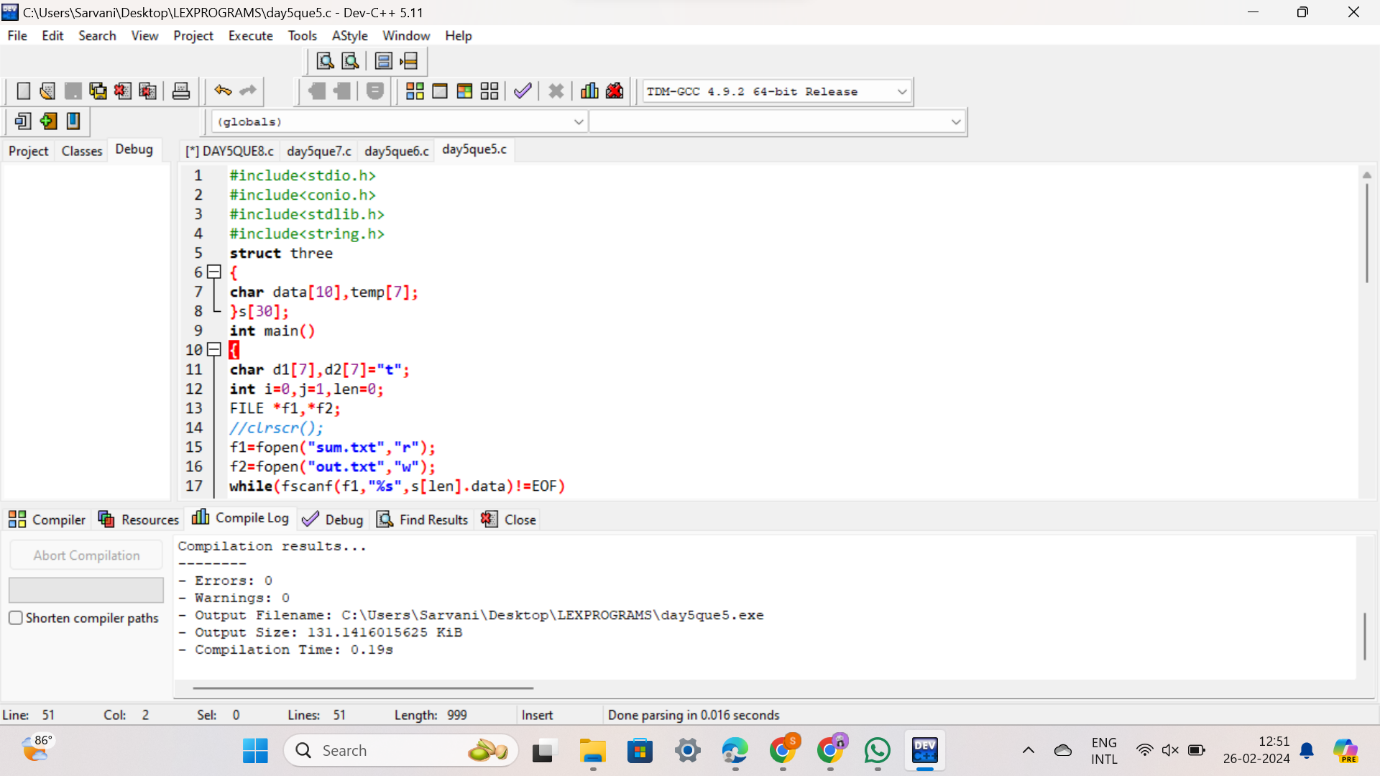
fclose(f1);

fclose(f2);

getch();

}

**OUTPUT:**



**QUE 6**

**Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int characters = 0, words = 0, lines = 0;

char ch;

// Open the file in read mode

FILE \*fp = fopen("file.txt", "r");

// Check if the file exists

if (fp == NULL)

{

printf("Error: Could not open file\n");

exit(1);

}

// Read the file character by character

while ((ch = fgetc(fp)) != EOF)

{

// Increment the character count

characters++;

// Check if the character is a space or a newline

if (ch == ' ' || ch == '\n')

{

// Increment the word count

words++;

}

// Check if the character is a newline

if (ch == '\n')

{

// Increment the line count

lines++;

}

}

// Close the file

fclose(fp);

// Print the results

printf("Characters: %d\n", characters);

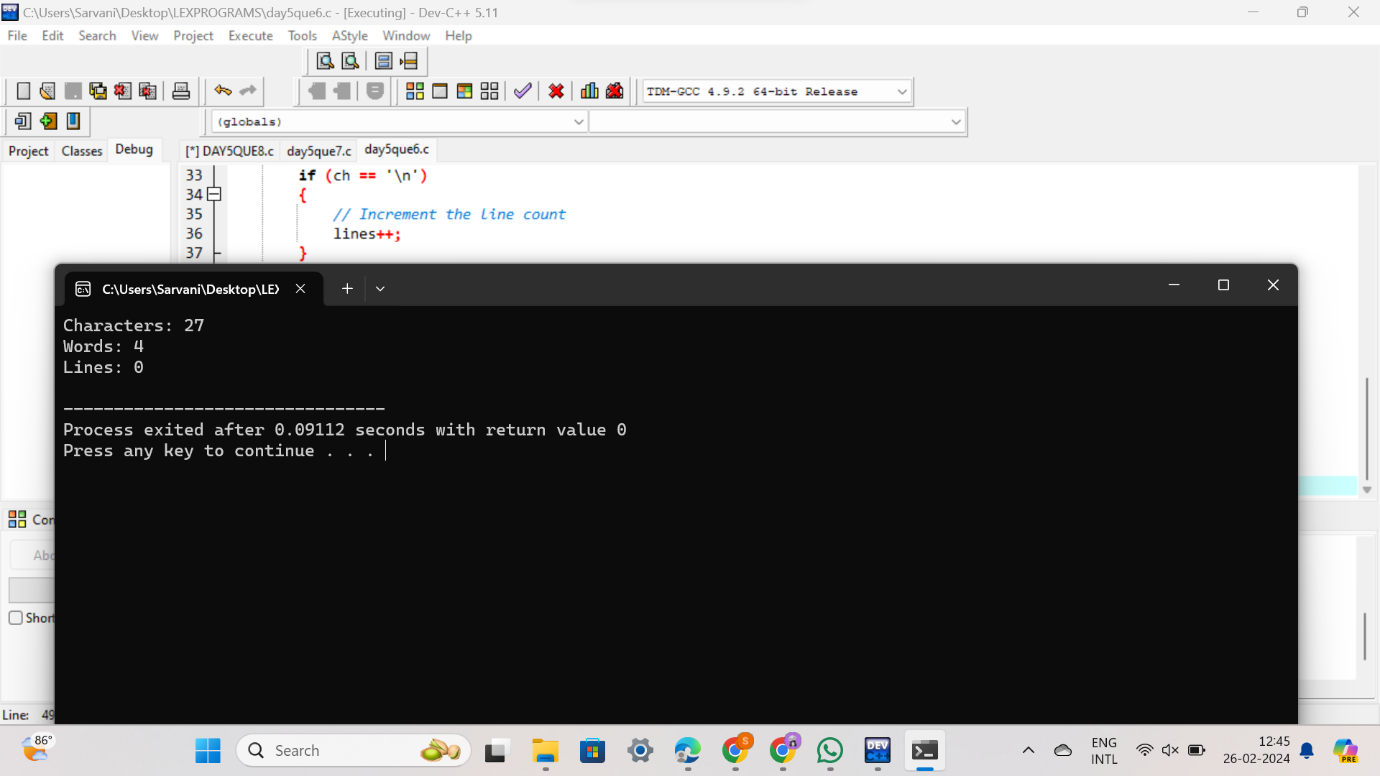
printf("Words: %d\n", words);

printf("Lines: %d\n", lines);

return 0;

}

**OUTPUT:**



**QUE 7**

**Write a C Program for code optimization to eliminate common subexpression.**

**CODE:**

#include <stdio.h>

int main() {

int a = 10;

int b = 20;

int c = a + b;

int d = a + b \* 2;

// Common subexpression elimination

int temp = a + b;

c = temp;

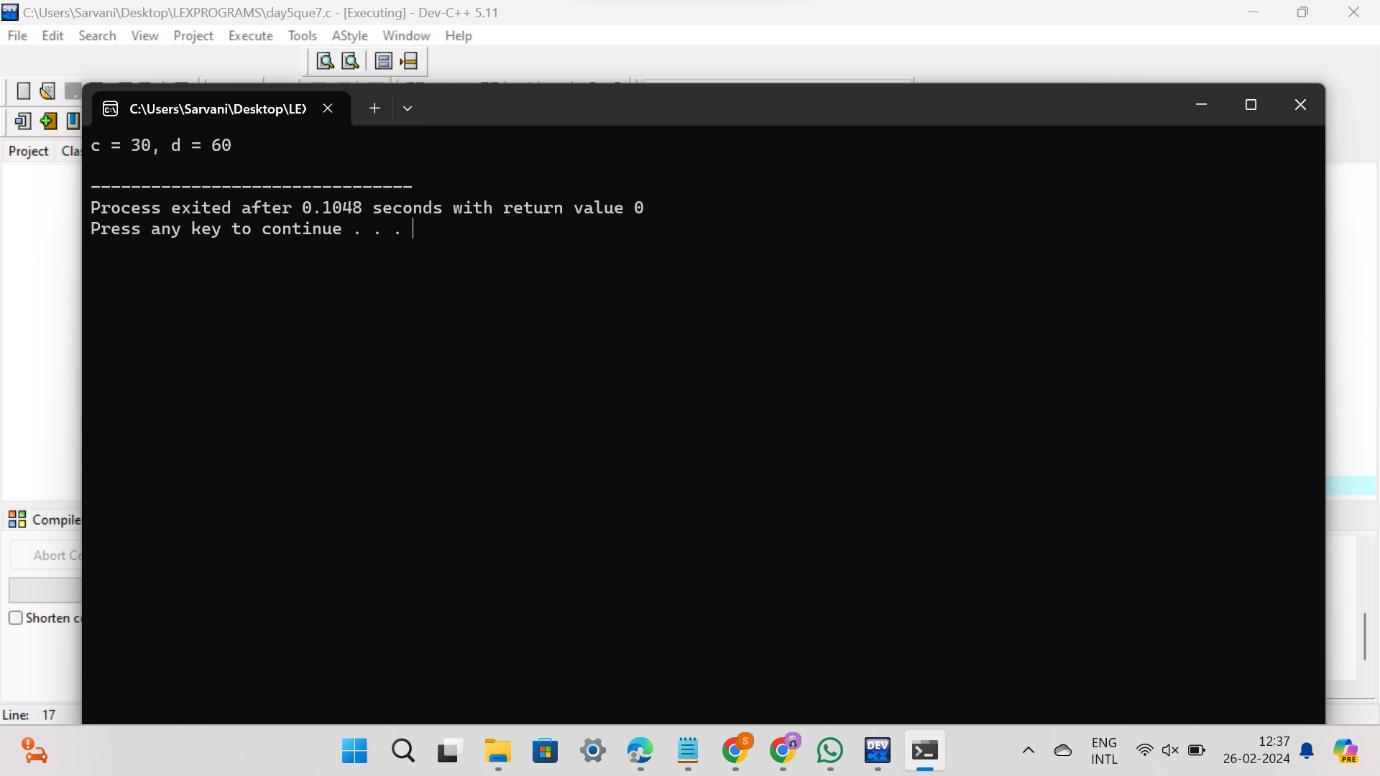
d = temp \* 2;

printf("c = %d, d = %d\n", c, d);

return 0;

}

**OUTPUT:**



**QUE 8**

**Write a C program to implement the back end of the compiler.**

**CODE:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int n,i,j;

char a[50][50];

printf("enter the no: intermediate code:");

scanf("%d",&n);

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

{

printf("enter the 3 address code:%d:",i+1);

for(j=0;j<6;j++)

{

scanf("%c",&a[i][j]);

}

}

printf("the generated code is:");

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

{

printf("\n mov %c,R%d",a[i][3],i);

if(a[i][4]=='-')

{

printf("\n sub %c,R%d",a[i][5],i);

}

if(a[i][4]=='+')

{

printf("\n add %c,R%d",a[i][5],i);

}

if(a[i][4]=='\*')

{

printf("\n mul %c,R%d",a[i][5],i);

}

if(a[i][4]=='/')

{

printf("\n div %c,R%d",a[i][5],i);

}

printf("\n mov R%d,%c",i,a[i][1]);

printf("\n");

}

return 0;

}

**OUTPUT:**

