Acknowledgement

I would like to express my hearty thankfulness to my faculty members

**Dr. Diptendu Bhattacharya (Associate Professor), Mr Ardhendu Gupta**

(LAB/Technical Assistant) for their continuous efforts to help us grasp this

subject and also for generating interest in it. I would also like to thank my

dear friends to share their experience while doing the experiment which has

helped me to muster the experimental knowledge and also to confirm

theoretical knowledge from the same. It has been my pleasure in

performing these experiments of Compiler Designlab. My thanks and

appreciation also goes to all other teaching staffs of Computer Science and

Engineering department for their kind and timely support.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Name of Experiment** | **Date** |
| 1 | File handling | 19/01/2022 |
| 2 | Design a lexical analyzer for a given language and the lexical analyzer. | 02/02/2022 |
| 3 | Write a C program to recognize strings under 'a', 'a\*b+', 'abb'. | 09/02/2022 |
| 4 | Write programs to show installation of lex code & execution. | 16/02/2022 |
| 5 | writing lex code & execution. | 16/02/2022 |
| 6 | LL 1 & Recursive Descent Parser implementation. | 08/03/2022 |
| 7 | Write a program to implement shift reduce parser. | 08/03/2022 |
| 8 | Construction of a recursive decent parser. | 04/04/2022 |
| 9 | Write A C program to implement operator precedence parsing | 04/04/2022 |
| 10 | C program to Design LALR Bottom up Parser | 11/04/2022 |

**EXPERIMENT – 1.1**

* 1. **OBJECTIVE:**

**Write a C program to read names and marks of n number of students from users and store them in a file.**

* 1. **RESOURCE:**

VS CODE

* 1. **PROCEDURE:**

Go to debug -> run or press CTRL + ALT+N to run the program.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

char name[30];

int marks,i,n;

printf("Enter number of students: ");

scanf("%d",&n);

FILE \*fptr;

fptr=(fopen("student.txt","w"));

if(fptr==NULL) {

printf("Error!");

exit(1);

}

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

printf("For student%d\nEnter name: ",i+1);

scanf("%s",name);

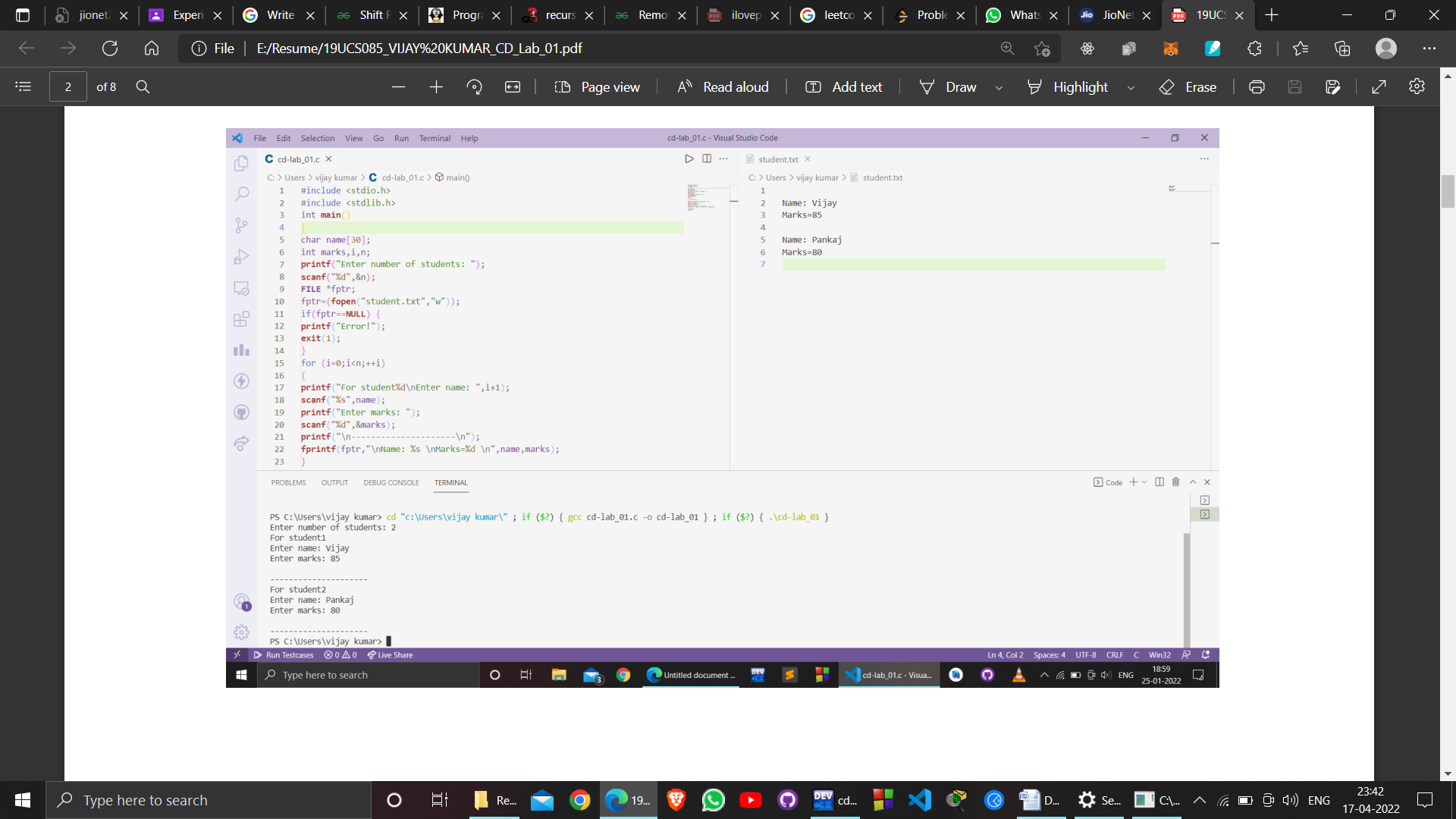
printf("Enter marks: ");

scanf("%d",&marks);

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

fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks); }

**OUTPUT:**

****

* 1. **EXPERIMENT – 1.2**
  2. **OBJECTIVE:**

**Write a C program to illustrate to read a file stored on the disk.**

* 1. **RESOURCE:**

VS CODE

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

FILE \*fptr;

char filename[15];

char ch;

printf("Enter the filename to be opened \n");

scanf("%s", filename);

/\* open the file for reading \*/

fptr = fopen(filename, "r");

if (fptr == NULL)

{

printf("Cannot open file \n");

exit(0);

}

ch = fgetc(fptr);

while (ch != EOF)

{

printf ("%c", ch);

ch = fgetc(fptr);

}

fclose(fptr);

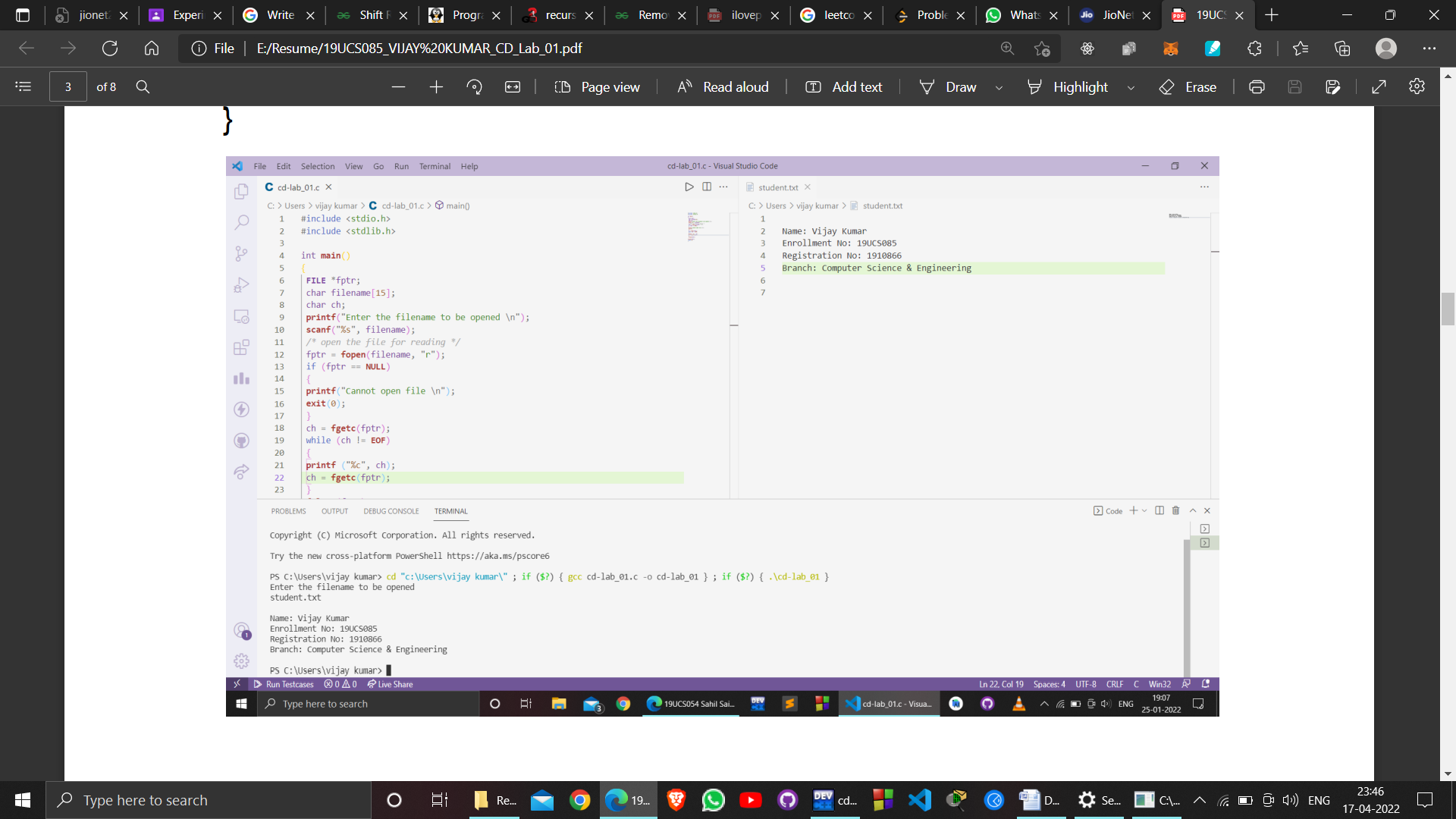
}

fclose(fptr);

return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 1.3**

* 1. **OBJECTIVE:**
  2. **Write a C Program Delete a specific Line from a Text File.**
  3. **RESOURCE:**

VS CODE

**PROGRAM:**

#include <stdio.h>

int main()

{

FILE\* fp1;

FILE\* fp2;

char ch;

int line = 0;

int temp = 1;

fp1 = fopen("disk.txt", "r");

if (fp1 == NULL) {

printf("\nUnable to open file\n");

return -1;

}

while (!feof(fp1)) {

ch = getc(fp1);

printf("%c", ch);

}

rewind(fp1);

printf("\nEnter line number to delete the line: ");

scanf("%d", &line);

fp2 = fopen("modify.txt", "w");

while (!feof(fp1)) {

ch = getc(fp1);

if (ch == '\n')

temp++;

if (temp != line)

putc(ch, fp2);

}

fclose(fp1);

fclose(fp2);

remove("disk.txt");

rename("modify.txt", "disk.txt");

printf("\nModified file:\n");

fp1 = fopen("disk.txt", "r");

if (fp1 == NULL) {

printf("\nUnable to open file\n");

return -1;

}

while (!feof(fp1)) {

ch = getc(fp1);

printf("%c", ch);

}

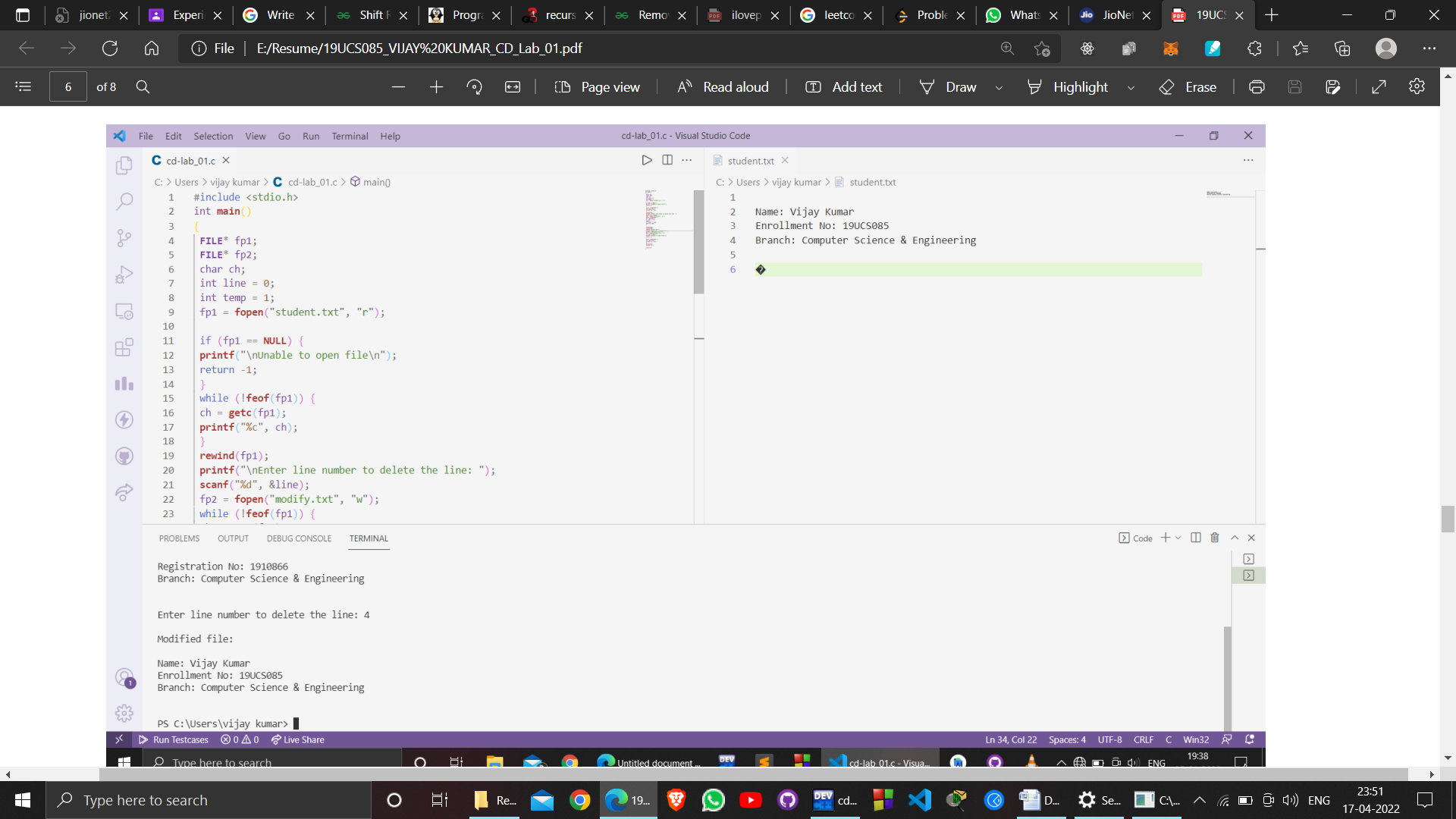
fclose(fp1);

printf("\n");

return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 1.4**

* 1. **OBJECTIVE:**
  2. **Write a C Program copy content from one file to another.**
  3. **RESOURCE:**

VS CODE

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

char c, filename[20];

int main()

{

FILE \*fptr1, \*fptr2;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

// Open one file for reading

fptr1 = fopen(filename, "r");

7

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

// Open another file for writing

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

// Read contents from file

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

printf("\nFile Contents copied to %s", filename);

fclose(fptr1);

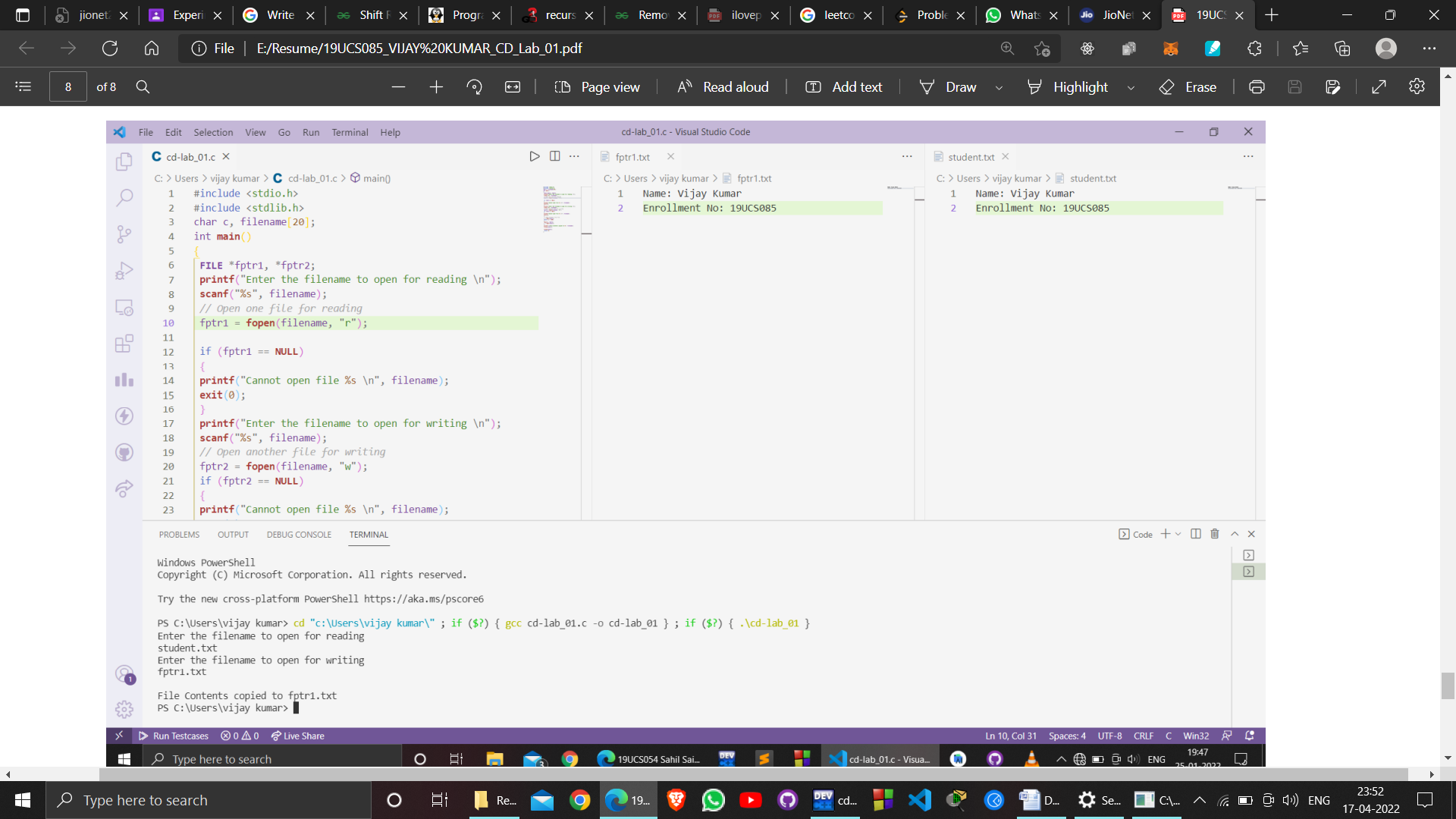
8

fclose(fptr2);

return 0;

}

**OUTPUT:**

****

**EXPERIMENT – 02**

* 1. **OBJECTIVE:**

Design a lexical analyser for a given language and the lexical analyser should ignore redundant spaces, tabs, and newlines. 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. Simulate the same in C language.

* 1. **RESOURCE:**

VS CODE

**PROGRAM:**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

bool isValidDelimiter(char ch)

{

if (ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

ch == ' ' || ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

ch == '[' || ch == ']' || ch == '{' || ch == '}')

return (true);

return (false);

}

bool isValidOperator(char ch){

if (ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == '>' || ch == '<' ||

ch == '=')

return (true);

return (false);

}

bool isvalidIdentifier(char\* str){

if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||

str[0] == '3' || str[0] == '4' || str[0] == '5' ||

str[0] == '6' || str[0] == '7' || str[0] == '8' ||

str[0] == '9' || isValidDelimiter(str[0]) == true)

return (false);

return (true);

}

bool isValidKeyword(char\* str) {

if (!strcmp(str, "if") || !strcmp(str, "else") ||

!strcmp(str, "while") ||

!strcmp(str, "do") || !strcmp(str, "break") ||

!strcmp(str, "continue") ||

!strcmp(str, "int")

|| !strcmp(str, "double") || !strcmp(str, "float") ||

!strcmp(str,

"return") || !strcmp(str, "char") || !strcmp(str, "case")

|| !strcmp(str,

"char")

|| !strcmp(str, "sizeof") || !strcmp(str, "long") ||

!strcmp(str, "short")

|| !strcmp(str, "typedef") || !strcmp(str, "switch") ||

!strcmp(str,

"unsigned")

|| !strcmp(str, "void") || !strcmp(str, "static") ||

!strcmp(str, "struct")

|| !strcmp(str, "goto"))

return (true);

return (false);

}

bool isValidInteger(char\* str) {

int i, len = strlen(str);

if (len == 0)

return (false);

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

if (str[i] != '0' && str[i] != '1' && str[i] != '2'&&

str[i] != '3' &&

str[i] != '4' && str[i] != '5'

&& str[i] != '6' && str[i] != '7' && str[i] != '8' &&

str[i] != '9' ||

(str[i] == '-' && i > 0))

return (false);

}

return (true);

}

bool isRealNumber(char\* str)

{

int i, len = strlen(str);

bool hasDecimal = false;

if (len == 0)

return (false);

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

if (str[i] != '0' && str[i] != '1' && str[i] != '2' &&

str[i] != '3' &&

str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i]

!= '7' &&

str[i] != '8'

&& str[i] != '9' && str[i] != '.' || (str[i] == '-' && i >

0))

return (false);

if (str[i] == '.')

hasDecimal = true;

}

return (hasDecimal);

}

char\* subString(char\* str, int left, int right)

{

int i;

char\* subStr = (char\*)malloc( sizeof(char) \* (right - left

+ 2));

for (i = left; i <= right; i++)

subStr[i - left] = str[i];

subStr[right - left + 1] = '\0';

return (subStr);

}

void detectTokens(char\* str)

{

int left = 0, right = 0;

int length = strlen(str);

while (right <= length && left <= right) {

if (isValidDelimiter(str[right]) == false)

right++;

if (isValidDelimiter(str[right]) == true && left == right)

{

if (isValidOperator(str[right]) == true)

printf("Valid operator : '%c'\n", str[right]);

right++;

left = right;

} else if (isValidDelimiter(str[right]) == true && left !=

right ||

(right == length && left != right)) {

char\* subStr = subString(str, left, right - 1);

if (isValidKeyword(subStr) == true)

printf("Valid keyword : '%s'\n", subStr);

else if (isValidInteger(subStr) == true)

printf("Valid Integer : '%s'\n", subStr);

else if (isRealNumber(subStr) == true)

printf("Real Number : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == true

&& isValidDelimiter(str[right - 1]) == false)

printf("Valid Identifier : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == false

&& isValidDelimiter(str[right - 1]) == false)

printf("Invalid Identifier : '%s'\n", subStr);

left = right;

}

}

return;

}

int main()

{

char str[100];

printf("Enter your input: ");

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

//printf("The Program is : '%s' \n", str);

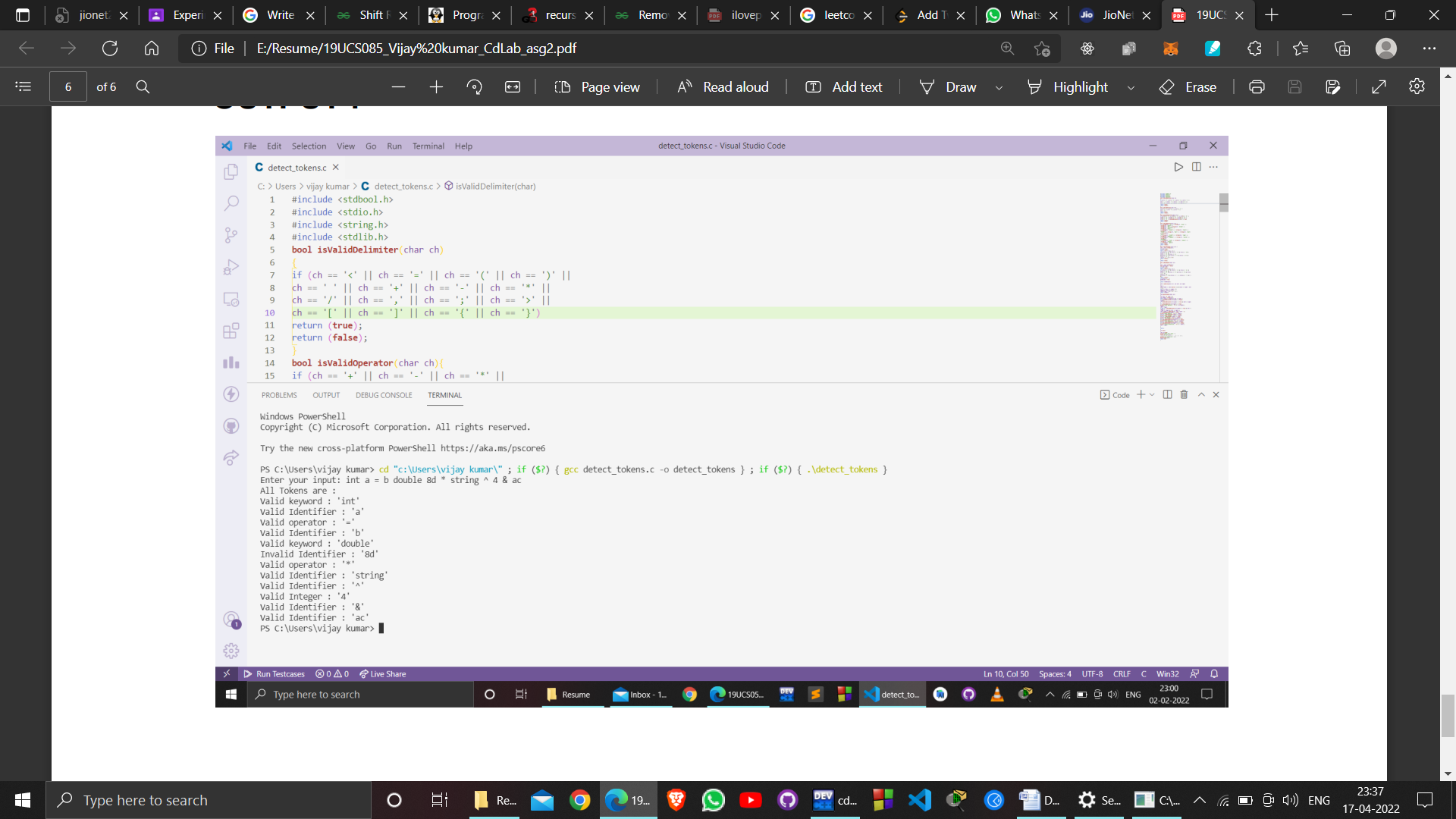
printf("All Tokens are : \n");

detectTokens(str);

return (0);

}

**OUTPUT:**

****

**EXPERIMENT – 03**

* 1. **OBJECTIVE:**
  2. **Write a C program to recognize strings under 'a', 'a\*b+', 'abb'.**
  3. **RESOURCE:**

VS CODE

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h> // For exit()

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

// Open one file for reading

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

// Open another file for writing

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

// Read contents from file

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

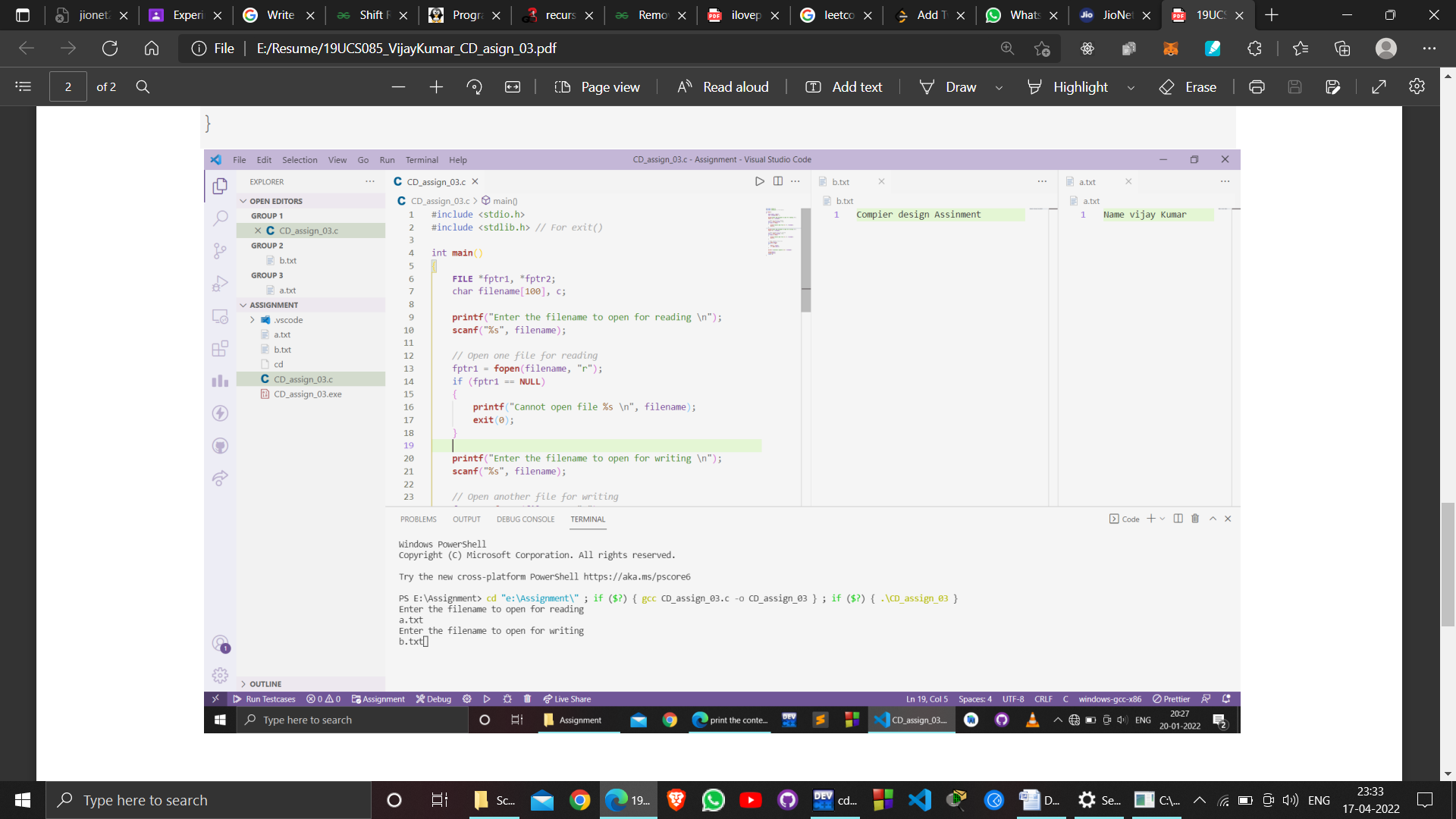
printf("\nContents copied to %s", filename);

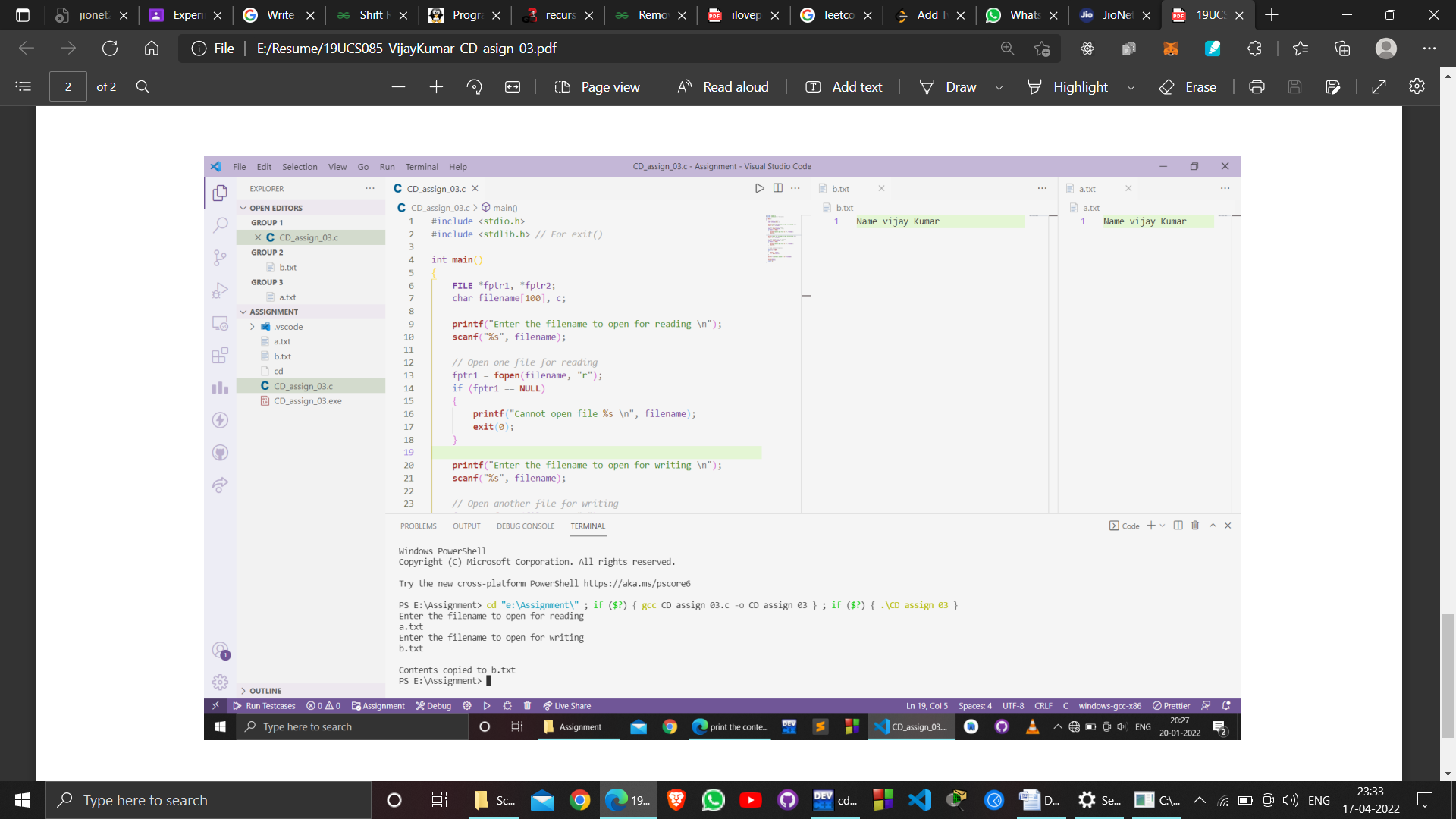
fclose(fptr1);

fclose(fptr2);

return 0; }

**OUTPUT:**

****

****

**EXPERIMENT – 04**

**OBJECTIVE: Write programs to show installation of lex code & execution**

**PROGRAM:**

%{

#include<stdio.h>

#include<string.h>

int i = 0;

%}

%%

[\n] {

printf("Hello World

}

%%

int yywrap(void){}

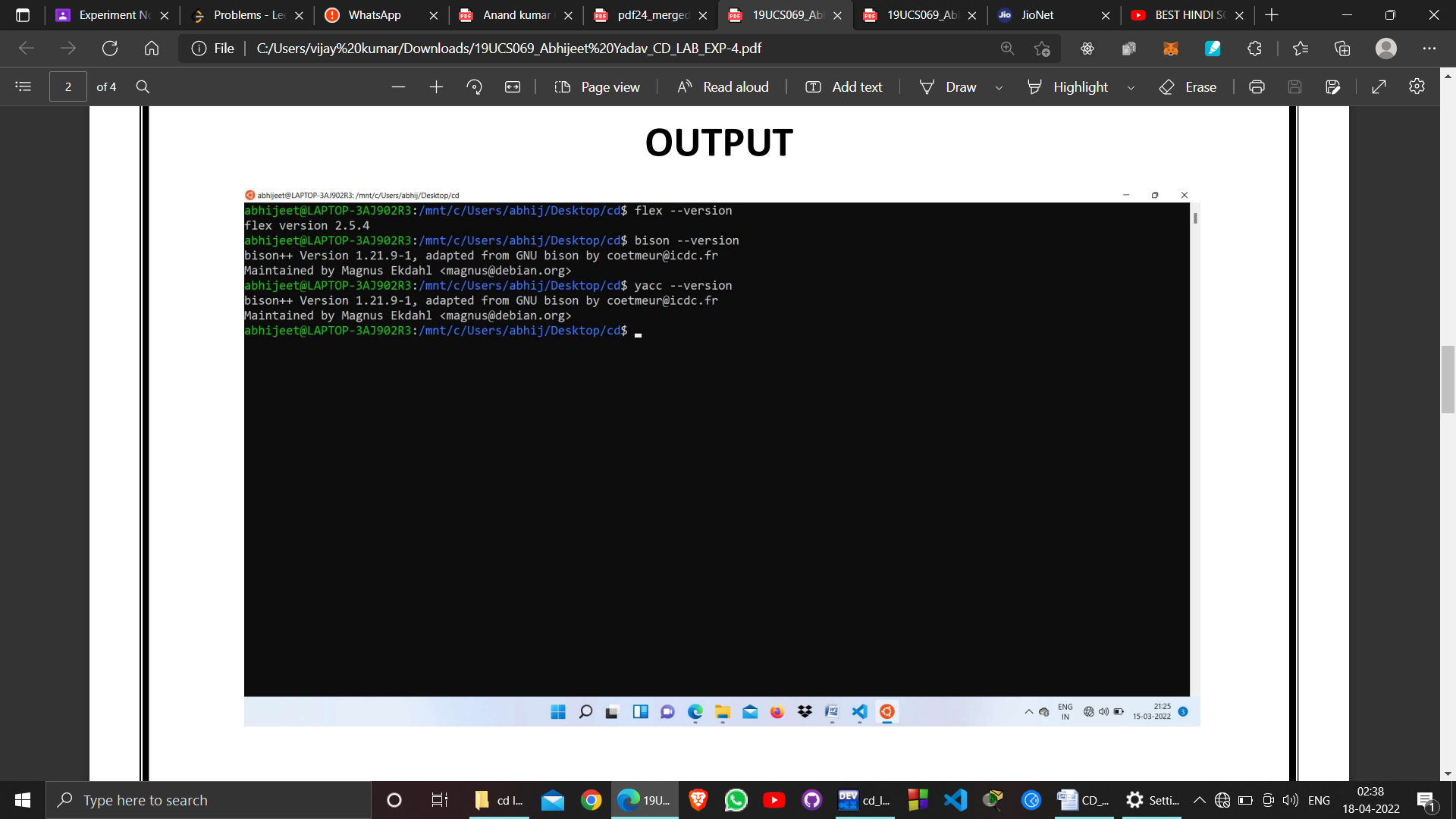
main()

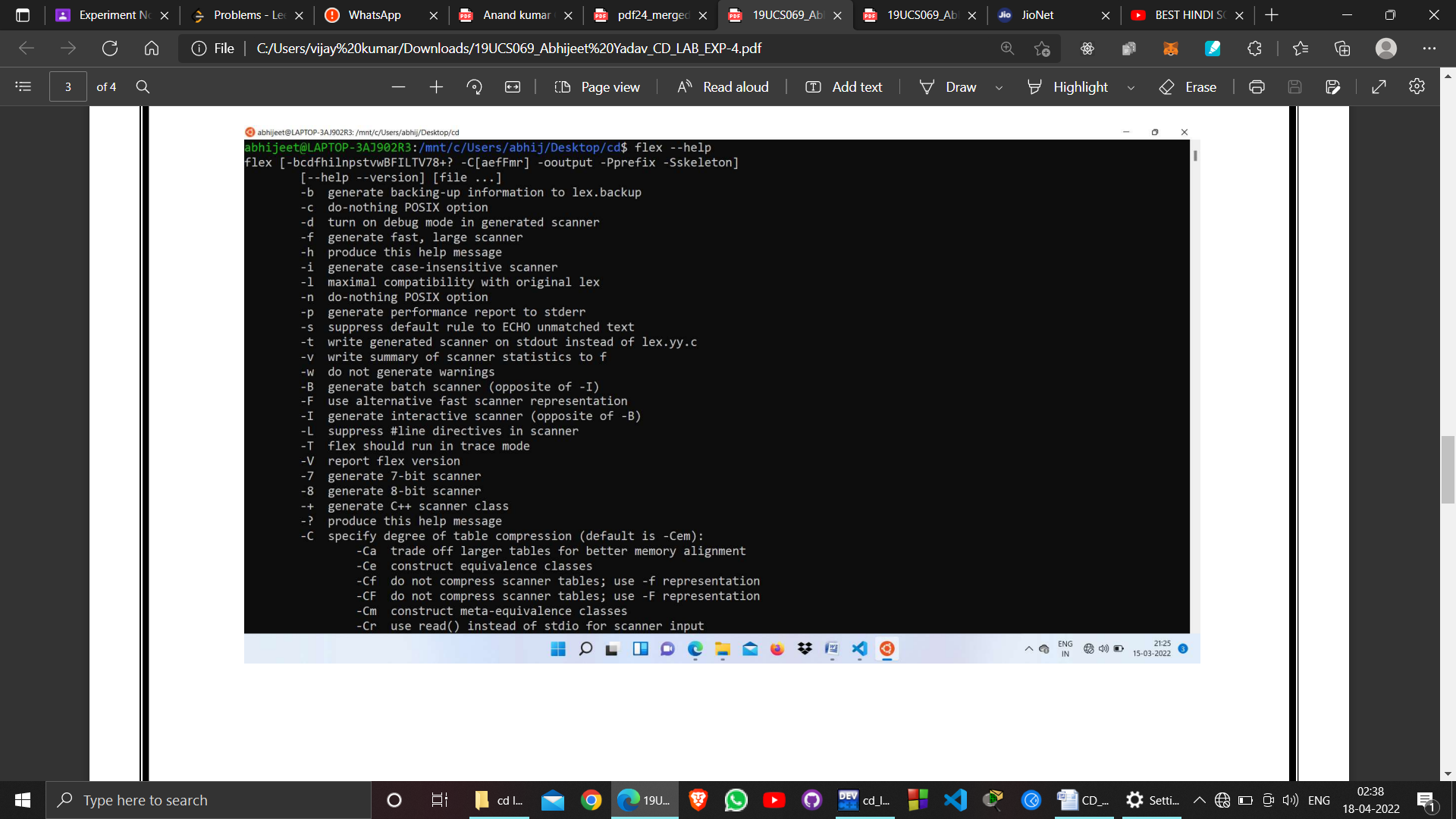
{

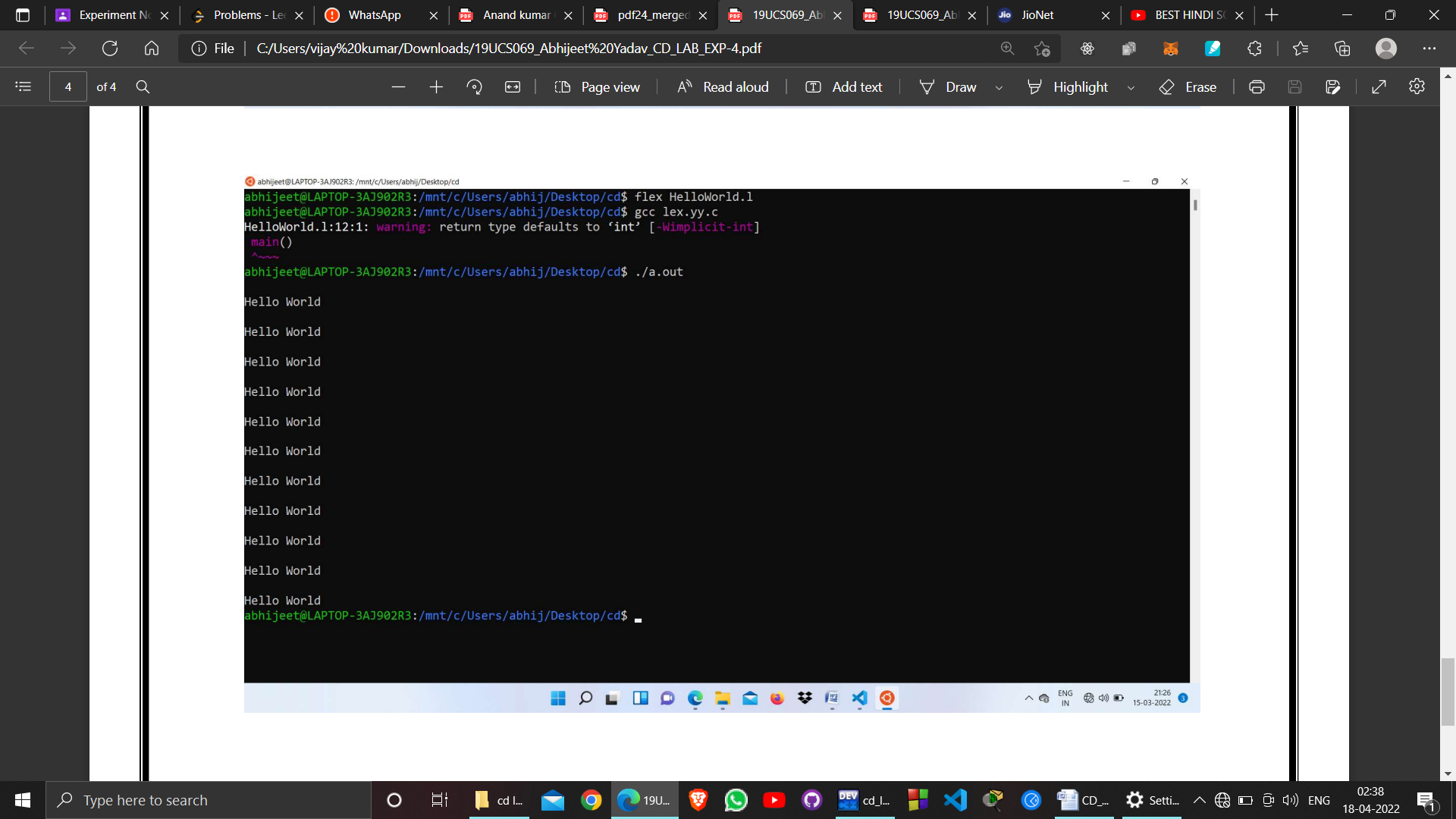
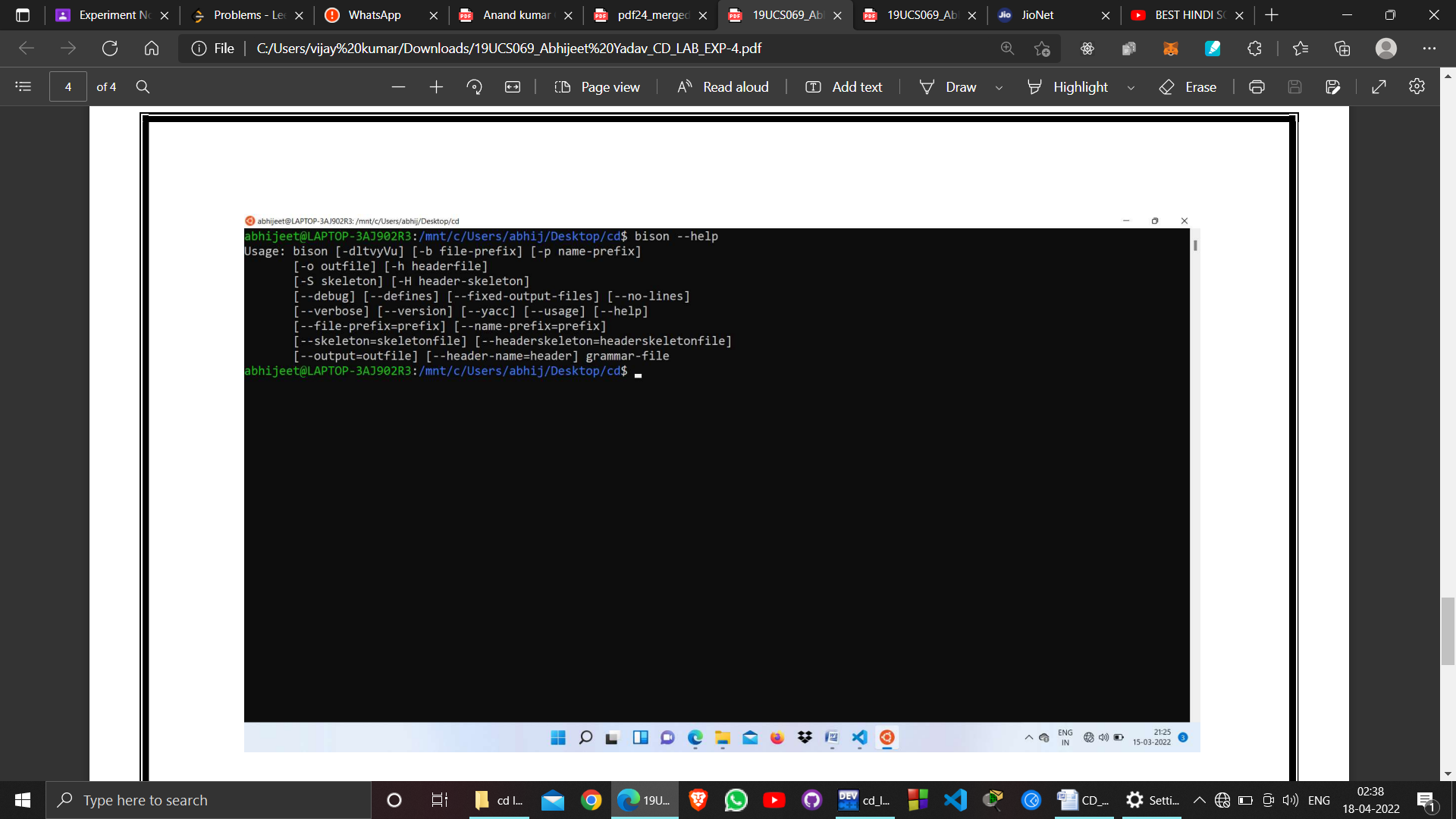
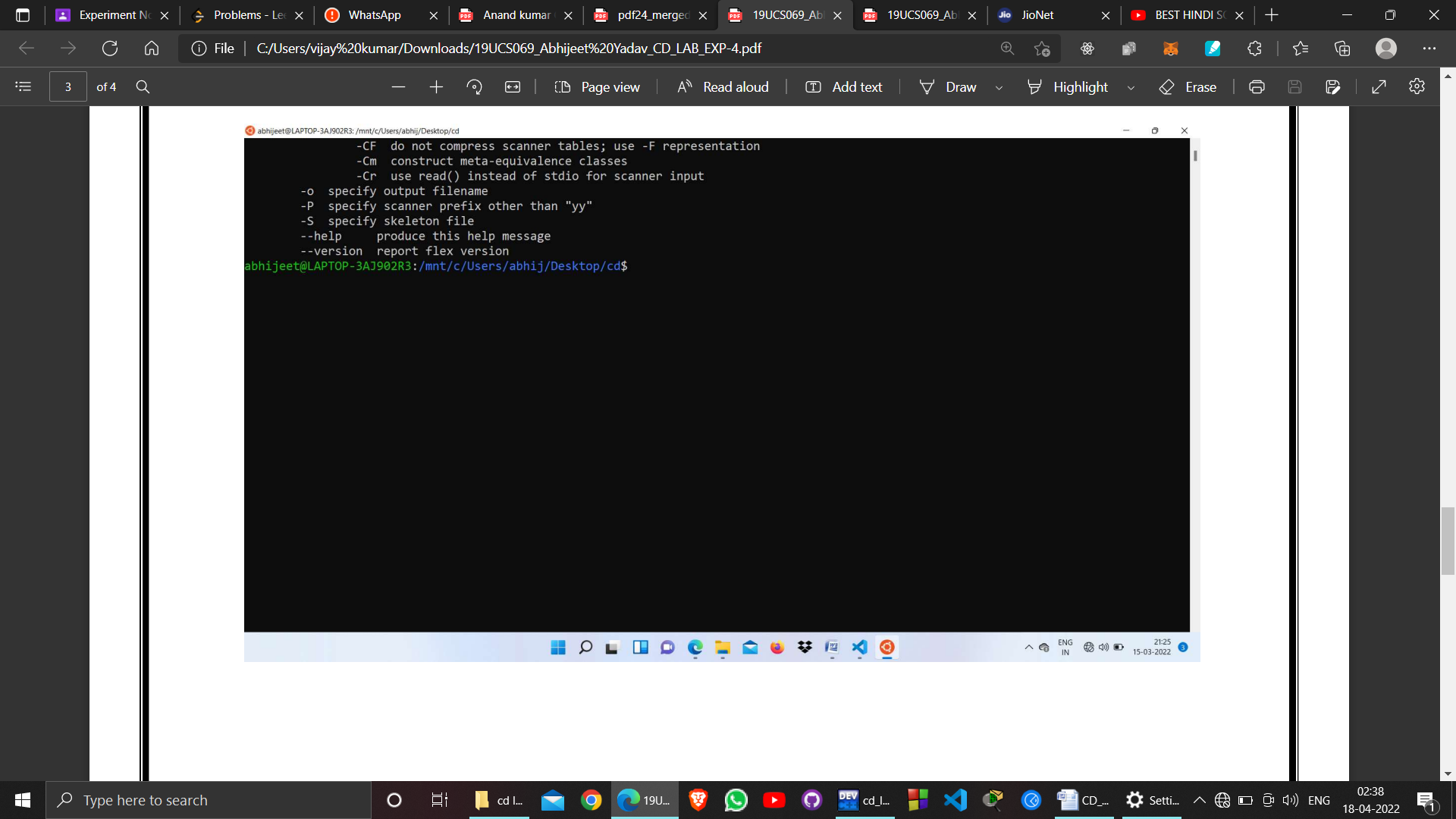
yylex();

}

**OUTPUT:**

****

****

****

**EXPERIMENT – 5.1**

**OBJECTIVE :** Write programs to check whether input letter is Vowel or Consonent.

**PROGRAM:**

%{

#include<stdio.h>

#include<string.h>

int i = 0;

%}

%%

[AEIOUaeiou] {

printf("Vowel\n");

}

[A-Za-z] printf("Consonent

%%

int yywrap(void){}

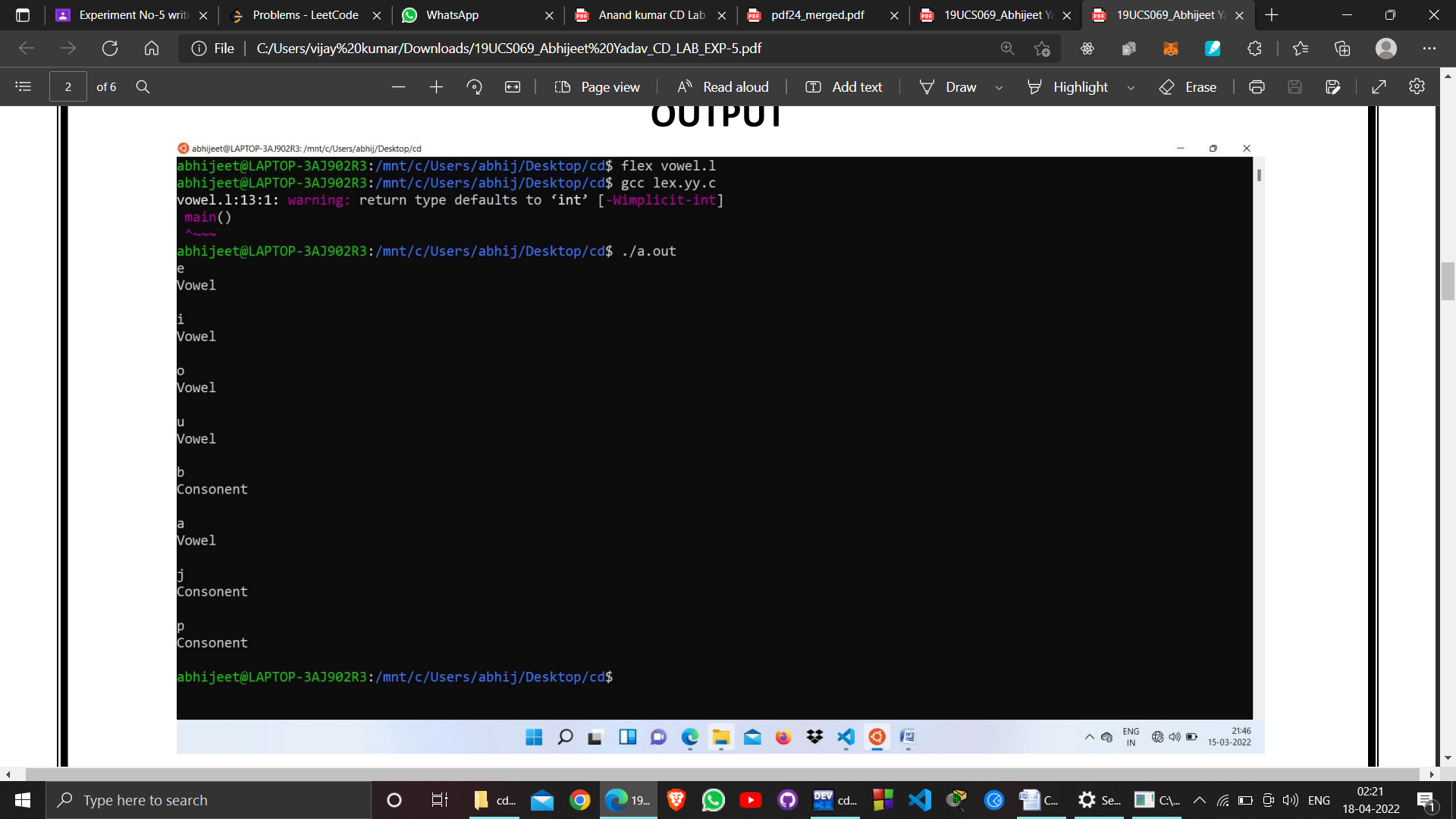
main()

{

yylex();

}

**OUTPUT:**

****

**EXPERIMENT – 5.2**

**OBJECTIVE :** Write programs to check whether input letter is capital or small.

**PROGRAM:**

%{

#include<stdio.h>

#include<string.h>

Int i =0;

%}

%%

[A-Z]{

Printf(“Capital Letter \n”);

}

[a-z] printf(“Small Letter \n”);

%%

int yywrap(void){}

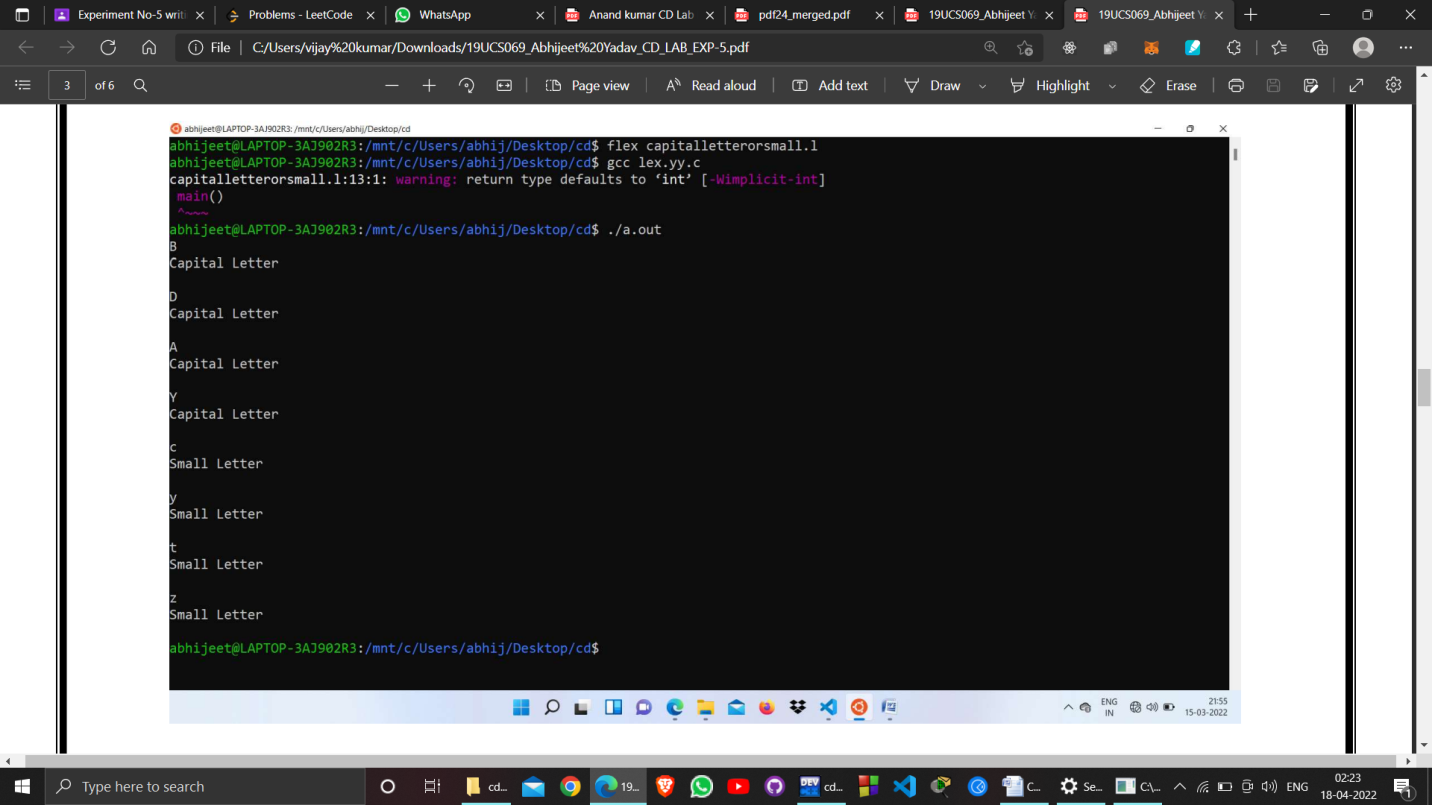
Main()

{

yylex(); // calling the rules section

}

**OUTPUT:**

****

**EXPERIMENT – 5.3**

**OBJECTIVE :** Write programs to count number of lines , tabs , spaces used in input.

**PROGRAM:**

%{

#include<stdio.h>

#include<string.h>

int lines = 0,space=0,tabs =0;

%}

%%

\t {

tabs++;

}

([ ])+ {space++;}

\n {lines++;}

%%

int yywrap(void){}

main()

{

yylex(); //calling the rules section

printf("Number of lines: %d\n Number of tabs: %d\n Number of

spaces: %d\n",lines,tabs,space); }

**EXPERIMENT – 5.4**

**OBJECTIVE :** Write programs to remove new lines, tabs, spaces used in input file and wirte in output file.

**PROGRAM:**

%{

#include<stdio.h>

extern FILE \*yyin , \*yyout;

%}

%%

[\n\t]+ {fprintf(yyout, “ “);}

. {fprintf(yyout, “%s”, yytext); }

%%

int yywrap(){}

int main()

{

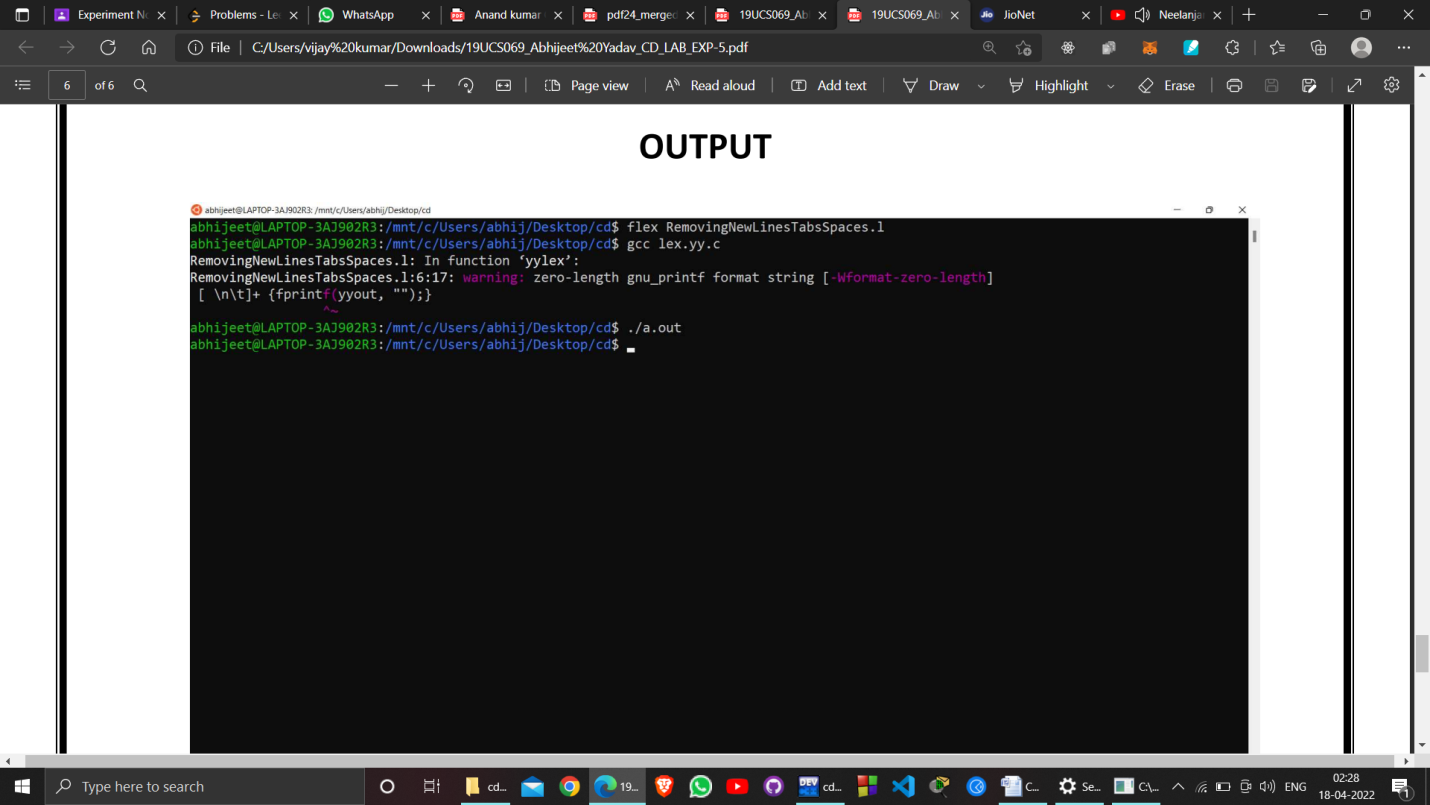
yyin = fopen(“a.text” , “r”);

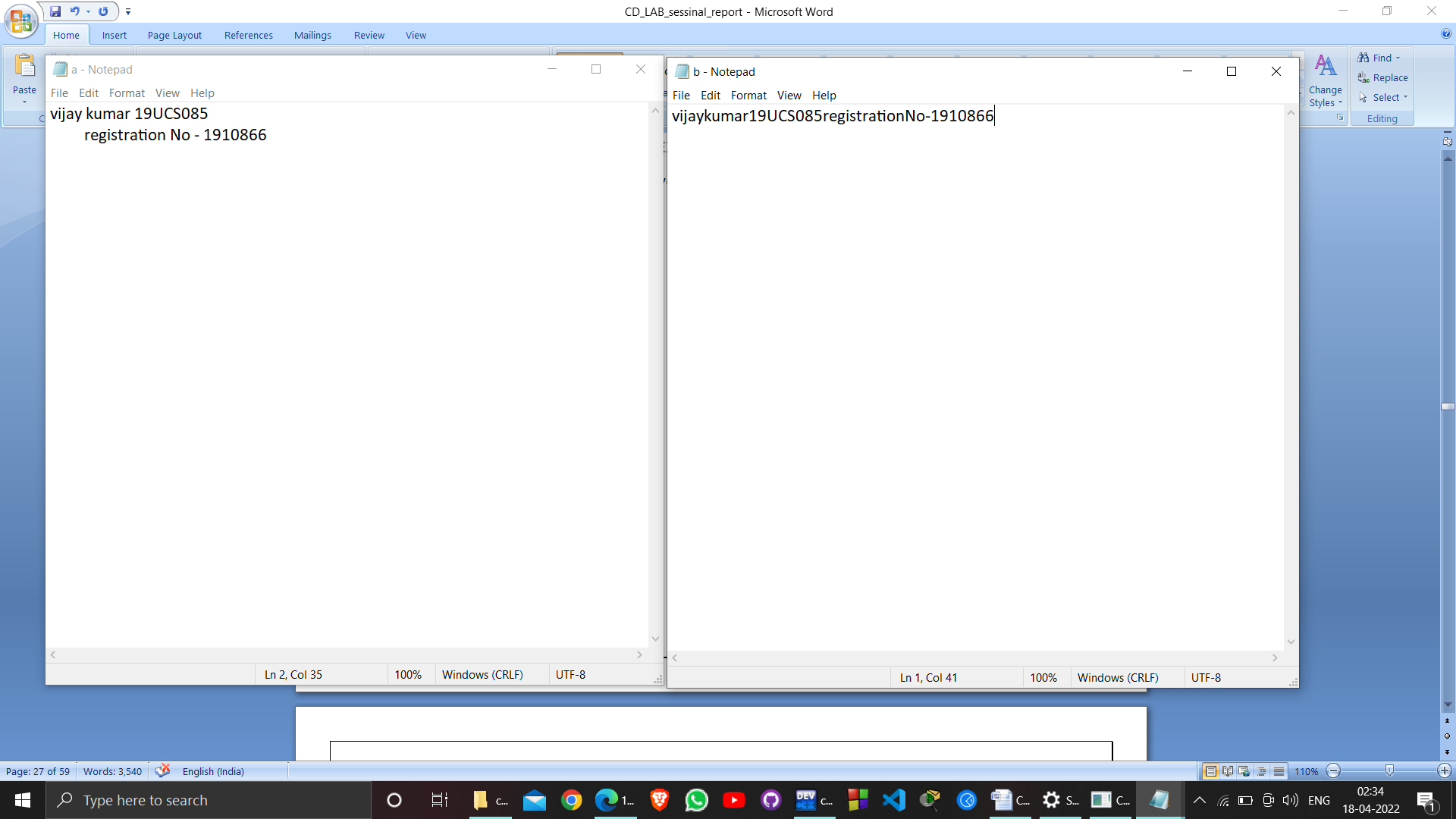
yyout = fopen( “b.text” , “w”);

yylex();

return 0; }

**OUTPUT:**

****

****

**EXPERIMENT – 06**

**OBJECTIVE:**

LL 1 & Recursive Descent Parser implementation.

**RESOURCE:**

DEV C++

**PROGRAM:**

#include <iostream>

#include <stdlib.h>

using namespace std;

int count = 0;

void E();

void Ed();

void T();

void Td();

void F();

string expr;

int main() {

cin >> expr;

int l = expr.length();

expr += "$";

E();

if (l == count)

cout << "Accepted" << endl;

else

cout << "Rejected" << endl;

}

void E() {

cout << "E->TE'" << endl;

T();

Ed();

}

void Ed() {

if (expr[count] == '+') {

count++;

cout << "E'->+TE'" << endl;

T();

Ed();

}

else if (expr[count] == '-') {

count++;

cout << "E'->-TE'" << endl;

T();

Ed();

}

else {

cout << "E'->null" << endl;

}

}

void T() {

cout << "T->FT'" << endl;

F();

Td();

}

void Td() {

if (expr[count] == '\*') {

count++;

cout << "T'->\*FT'" << endl;

F();

Td();

}

else if (expr[count] == '/') {

count++;

cout << "T'->/FT'" << endl;

F();

Td();

}

else {

cout << "T'->null" << endl;

}

}

void F() {

if (isalpha(expr[count])) {

count++;

cout << "F->id" << endl;

} else if (isdigit(expr[count])) {

count++;

cout << "F->digit" << endl;

} else if (expr[count] == '(') {

count++;

cout << "F->(E)" << endl;

E();

if (expr[count] != ')') {

cout << "Rejected" << endl;

exit(0);

}

count++;

} else {

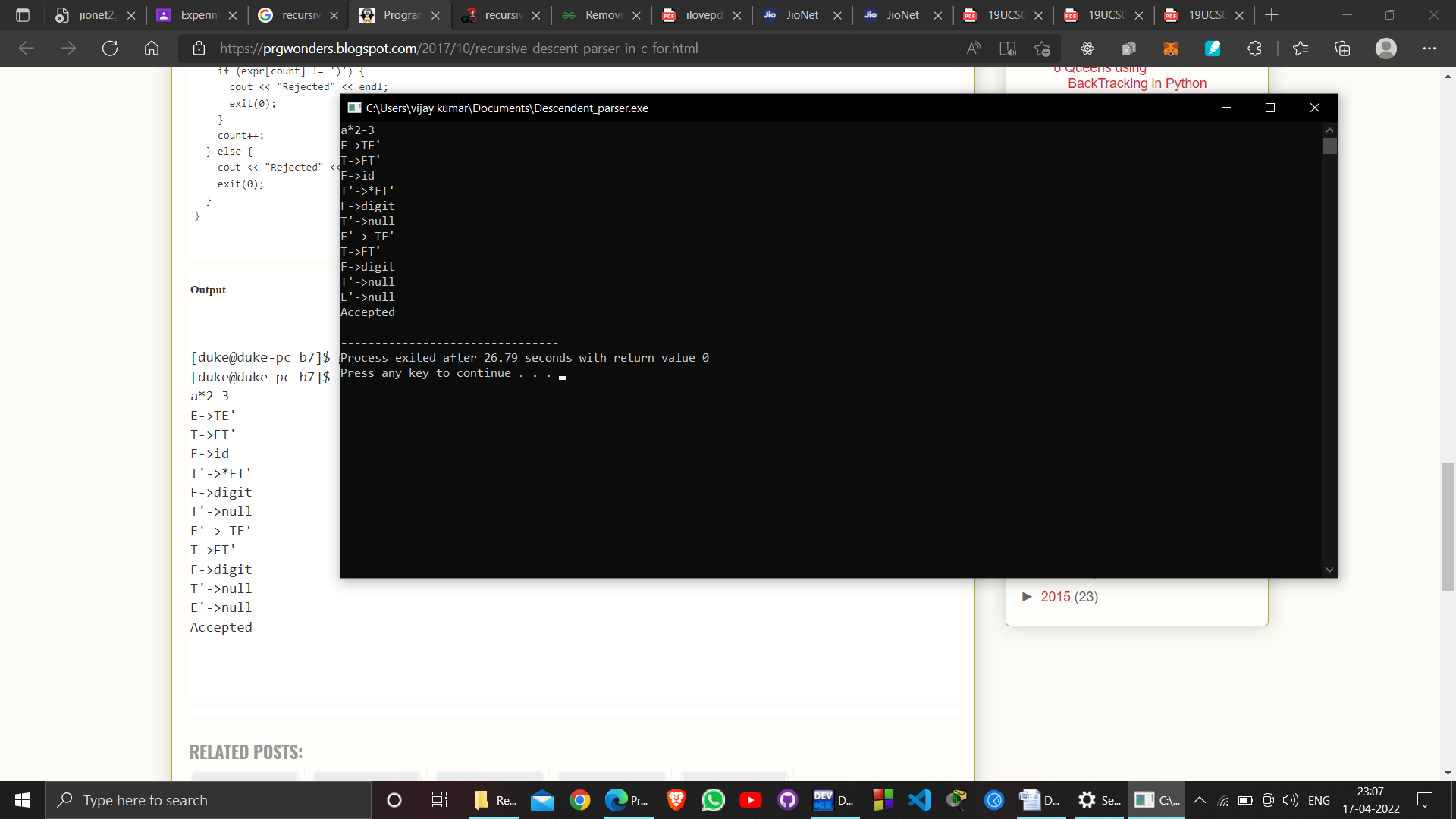
cout << "Rejected" << endl;

exit(0);

}

}

**OUPUT:**

****

**EXPERIMENT – 07**

**OBJECTIVE:**

LL 1 & Recursive Descent Parser implementation.

**RESOURCE:**

DEV C++

**PROGRAM:**

#include <bits/stdc++.h>

using namespace std;

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

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

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';

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

}

}

for(z = 0; z < c - 2; z++)

{

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 ;

}

int main()

{

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

strcpy(a,"32423");

c=strlen(a);

strcpy(act,"SHIFT");

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

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

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

{

printf("%s", act);

stk[i] = a[j];

stk[i + 1] = '\0';

a[j]=' ';

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

check();

}

check();

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

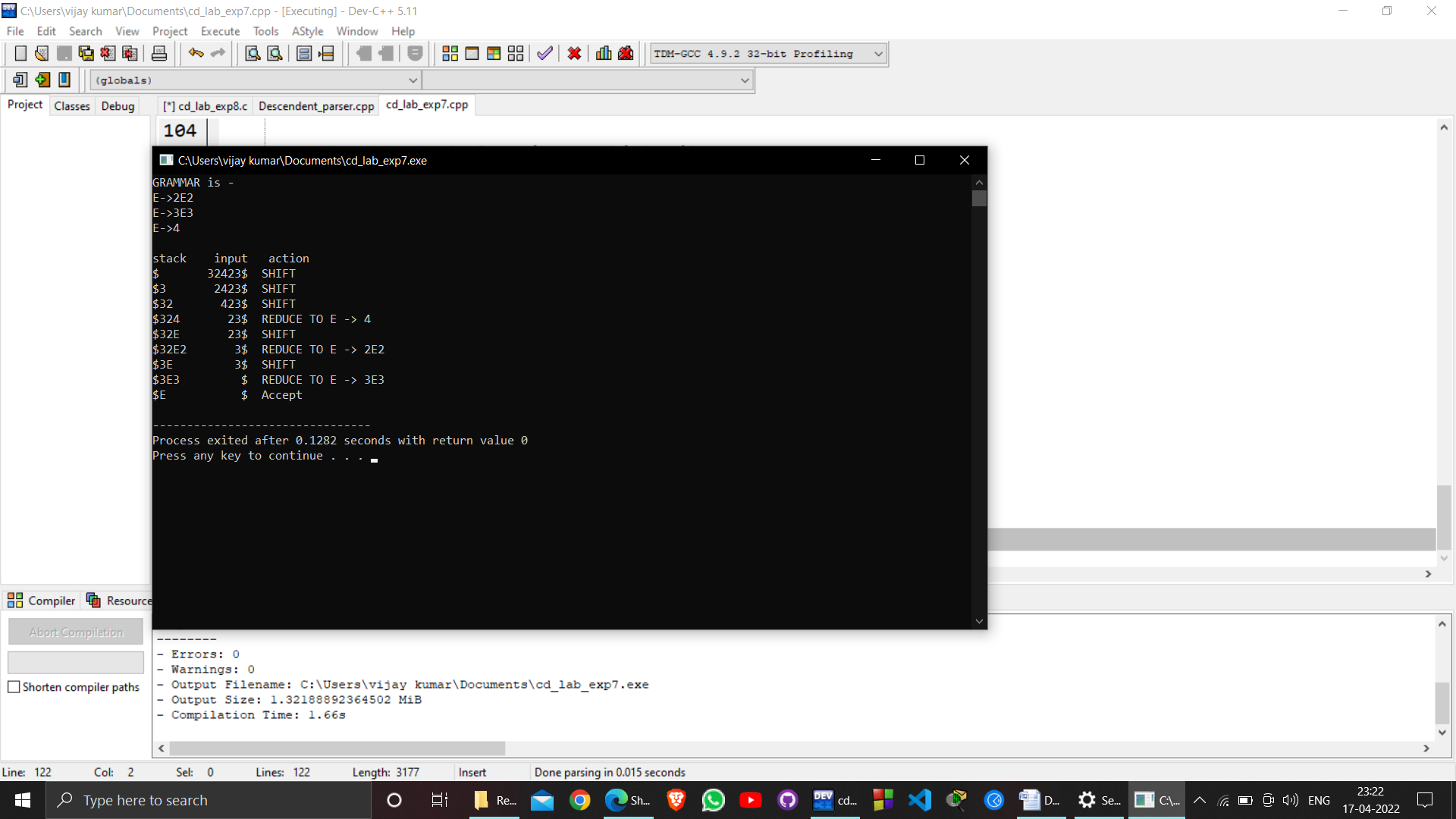
printf("Accept\n");

else //else reject

printf("Reject\n");

}

**OUTPUT:**

****

**EXPERIMENT- 08**

**Aim of Experiment :**

Construction of a recursive decent parser.

**Software :**

Ubuntu, VI editor .

**Program Logic:**

LOGIC: Read the input string.

Write procedures for the non terminals

Verify the next token equals to non terminals if it satisfies match the non terminal.

If the input string does not match print error.

**PROCEDURE:**

Go to debug -> run or press ctrl + f9 to run the program.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

int E();

int T();

int EP();

int F();

int TP();

char input[100];

int i,l;

int main()

{

printf("\nRecursive descent parsing for the following grammar\n");

printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->\*FT'/@\nF->(E)/ID\n");

printf("\nEnter the string to be checked:");

scanf("%s",&input);

if(E())

{

if(input[i+1]=='\0')

printf("\nString is accepted");

else

printf("\nString is not accepted");

}

else

printf("\nString not accepted")

}

int E()

{

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

int EP()

{

if(input[i]=='+')

{

i++;

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

int T()

{

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

int TP()

{

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

{

i++;

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

int F()

{

if(input[i]=='(')

{

i++;

if(E())

{

if(input[i]==')'

{

i++;

return(1);

}

else

return(0);

}

else

return(0);

}

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

{

i++;

return(1);

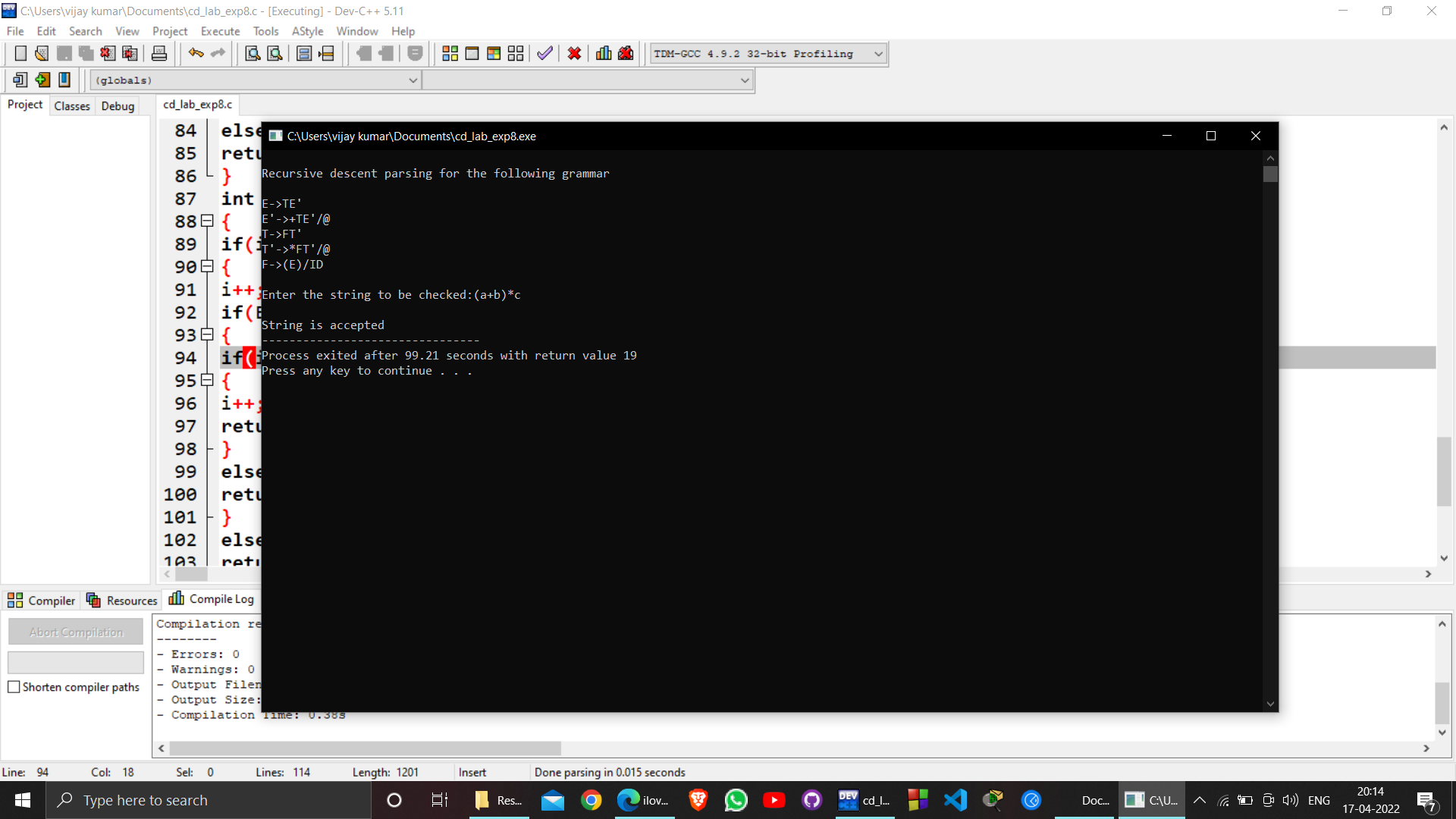
}

else

return(0);

}

**OUTPUT:**

****

**EXPERIMENT-9**

**Aim of Experiment :**

Write A C program to implement operator precedence parsing

**Software :**

Ubuntu, VI editor

**Program Logic:**

Read the arithmetic input string

Verify the precedence between terminals and symbols

Find the handle closed in < .> and reduce it to production symbol

Repeat the process till we reach the start node.

**PROCEDURE:**

Go to debug -> run or press ctrl + f9 to run the program.

**PROGRAM:**

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#define node struct tree1

int getOperatorPosition(char);

int matrix[5][5]=

{ {1,0,0,1,1},

{1,1,0,1,1},

{0,0,0,2,3},

{1,1,3,1,1},

{0,0,0,3,2} };

int tos=-1;

void matrix\_value(void);

void show\_tree(node \*);

int isOperator(char);

struct tree1

{

char data;

node \*lptr;

node \*rptr;

}\*first;

struct opr

{

char op\_name;

node \*t;

}

oprate[50];

char cur\_op[5]={'+','\*','(',')','['};

char stack\_op[5]={'+','\*','(',')',']'};

void main()

{

char exp[10];

int ssm=0,row=0,col=0;

node \*temp;

clrscr();

printf("enter exp:");

scanf("%s",exp);

matrix\_value();

while(exp[ssm]!='\0')

{

if(ssm==0)

{

tos++;

oprate[tos].op\_name=exp[tos];

}

else

{

if(isOperator(exp[ssm])==-1)

{

oprate[tos].t=(node\*)malloc(sizeof(node));

oprate[tos].t->data=exp[ssm];

oprate[tos].t->lptr='\0';

oprate[tos].t->rptr='\0';

}

else

{

row=getOperatorPosition(oprate[tos].op\_name);

col=getOperatorPosition(exp[ssm]);

if(matrix[row][col]==0)

{

tos++;

oprate[tos].op\_name=exp[ssm];

}

else if(matrix[row][col]==1)

{

temp=(node\*)malloc(sizeof(node));

temp->data=oprate[tos].op\_name;

temp->lptr=(oprate[tos-1].t);

temp->rptr=(oprate[tos].t);

tos--;

oprate[tos].t=temp;

ssm--;

}

else if(matrix[row][col]==2)

{

temp=oprate[tos].t;

tos--;

oprate[tos].t=temp;

}

else if(matrix[row][col]==3)

{

printf("\b expression is invalid...\n");

printf("%c %c can not occur simuktaneously\n", oprate[tos].op\_name,exp[ssm]);

break;

}

}

}ssm++;

}

printf("show tree \n\n\n");

show\_tree(oprate[tos].t);

printf("over");

getch();

}

int isOperator(char c)

{

int i=0;

for(i=0;i<5;i++)

{

if(c==cur\_op[i]||c==stack\_op[i])

break;

}

if(i==5)

return (-1);

else

return 1;

}

int getOperatorPosition(char c)

{

int i;

for(i=0;i<=5;i++)

{

if(c==cur\_op[i]||c==stack\_op[i])

break;

}

return i;

}

void show\_tree(node \*start)

{

if(start->lptr !=NULL)

show\_tree(start->lptr);

if(start->rptr !=NULL)

show\_tree(start->rptr);

printf("%c \n",start->data);

}

void matrix\_value(void)

{

int i,j;

printf("operator precedence matrix\n");

printf("==========================\n");

for(i=0;i<5;i++)

{

printf("%c",stack\_op[i]);

}

printf("\n");

for(i=0;i<5;i++)

{

printf("%c",cur\_op[i]);

for(j=0;j<5;j++)

{

if(matrix[i][j]==0)

printf("<");

else if(matrix[i][j]==1)

printf(">");

else if(matrix[i][j]==2)

printf("=");

else if(matrix[i][j]==3)

printf(" ");

}

printf("\n");

}

}

**OUTPUT:**

****

**EXPERIMENT-10**

**Aim of Experiment :**

C program to Design LALR Bottom up Parser

**Software :**

Ubuntu, VI editor

**Program Logic:**

LOGIC: Read the input string.

Write procedures for the non terminals

Verify the next token equals to non terminals if it satisfies match the non terminal.

If the input string does not match print error..

**PROCEDURE:**

Go to debug -> run or press ctrl + f 9 to run the program.

**PROGRAM :**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <string.h>

void push(char \*, int \*, char);

char stacktop(char \*);

void isproduct(char, char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char, char);

char pop(char \*, int \*);

void printt(char \*, int \*, char[], int);

void rep(char[], int);

struct action

{

char row[6][5];

};

const struct action A[12] = {

{"sf", "emp", "emp", "se", "emp", "emp"},

{"emp", "sg", "emp", "emp", "emp", "acc"},

{"emp", "rc", "sh", "emp", "rc", "rc"},

{"emp", "re", "re", "emp", "re", "re"},

{"sf", "emp", "emp", "se", "emp", "emp"},

{"emp", "rg", "rg", "emp", "rg", "rg"},

{"sf", "emp", "emp", "se", "emp", "emp"},

{"sf", "emp", "emp", "se", "emp", "emp"},

{"emp", "sg", "emp", "emp", "sl", "emp"},

{"emp", "rb", "sh", "emp", "rb", "rb"},

{"emp", "rb", "rd", "emp", "rd", "rd"},

{"emp", "rf", "rf", "emp", "rf", "rf"}

};

struct gotol

{

char r[3][4];

};

const struct gotol G[12] = {

{"b", "c", "d"},

{"emp", "emp", "emp"},

{"emp", "emp", "emp"},

{"emp", "emp", "emp"},

{"i", "c", "d"},

{"emp", "emp", "emp"},

{"emp", "j", "d"},

{"emp", "emp", "k"},

{"emp", "emp", "emp"},

{"emp", "emp", "emp"},

};

char ter[6] = {'i', '+', '\*', ')', '(', '$'};

char nter[3] = {'E', 'T', 'F'};

char states[12] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'm', 'j', 'k', 'l'};

char stack[100];

int top = -1;

char temp[10];

struct grammar

{

char left;

char right[5];

};

const struct grammar rl[6] = {

{'E', "e+T"},

{'E', "T"},

{'T', "T\*F"},

{'T', "F"},

{'F', "(E)"},

{'F', "i"},

};

int main()

{

char inp[80], x, p, dl[80], y, bl = 'a';

int i = 0, j, k, l, n, m, c, len;

// clrscr();

printf(" Enter the input :");

scanf("%s", inp);

len = strlen(inp);

inp[len] = '$';

inp[len + 1] = '\0';

push(stack, &top, bl);

printf("\n stack \t\t\t input");

printt(stack, &top, inp, i);

do

{

x = inp[i];

p = stacktop(stack);

isproduct(x, p);

if (strcmp(temp, "emp") == 0)

error();

if (strcmp(temp, "acc") == 0)

break;

else

{

if (temp[0] == 's')

{

push(stack, &top, inp[i]);

push(stack, &top, temp[1]);

i++;

}

else

{

if (temp[0] == 'r')

{

j = isstate(temp[1]);

strcpy(temp, rl[j - 2].right);

dl[0] = rl[j - 2].left;

dl[1] = '\0';

n = strlen(temp);

for (k = 0; k < 2 \* n; k++)

pop(stack, &top);

for (m = 0; dl[m] != '\0'; m++)

push(stack, &top, dl[m]);

l = top;

y = stack[l - 1];

isreduce(y, dl[0]);

for (m = 0; temp[m] != '\0'; m++)

push(stack, &top, temp[m]);

}

}

}

printt(stack, &top, inp, i);

} while (inp[i] != '\0');

if (strcmp(temp, "acc") == 0)

printf(" \n accept the input ");

else

printf(" \n do not accept the input ");

getch();

return 0;

}

void push(char \*s, int \*sp, char item)

{

if (\*sp == 100)

printf(" stack is full ");

else

{

\*sp = \*sp + 1;

s[\*sp] = item;

}

}

char stacktop(char \*s)

{

char i;

i = s[top];

return i;

}

void isproduct(char x, char p)

{

int k, l;

k = ister(x);

l = isstate(p);

strcpy(temp, A[l - 1].row[k - 1]);

}

int ister(char x)

{

int i;

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

if (x == ter[i])

return i + 1;

return 0;

}

int isnter(char x)

{

int i;

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

if (x == nter[i])

return i + 1;

return 0;

}

int isstate(char p)

{

int i;

for (i = 0; i < 12; i++)

if (p == states[i])

return i + 1;

return 0;

}

void error()

{

printf(" error in the input ");

exit(0);

}

void isreduce(char x, char p)

{

int k, l;

k = isstate(x);

l = isnter(p);

strcpy(temp, G[k - 1].r[l - 1]);

}

char pop(char \*s, int \*sp)

{

char item;

if (\*sp == -1)

printf(" stack is empty ");

else

{

item = s[\*sp];

\*sp = \*sp - 1;

}

return item;

}

void printt(char \*t, int \*p, char inp[], int i)

{

int r;

printf("\n");

for (r = 0; r <= \*p; r++)

rep(t, r);

printf("\t\t\t");

for (r = i; inp[r] != '\0'; r++)

printf("%c", inp[r]);

}

void rep(char t[], int r)

{

char c;

c = t[r];

switch (c)

{

case 'a':

printf("0");

break;

case 'b':

printf("1");

break;

case 'c':

printf("2");

break;

case 'd':

printf("3");

break;

case 'e':

printf("4");

break;

case 'f':

printf("5");

break;

case 'g':

printf("6");

break;

case 'h':

printf("7");

break;

case 'm':

printf("8");

break;

case 'j':

printf("9");

break;

case 'k':

printf("10");

break;

case 'l':

printf("11");

break;

default:

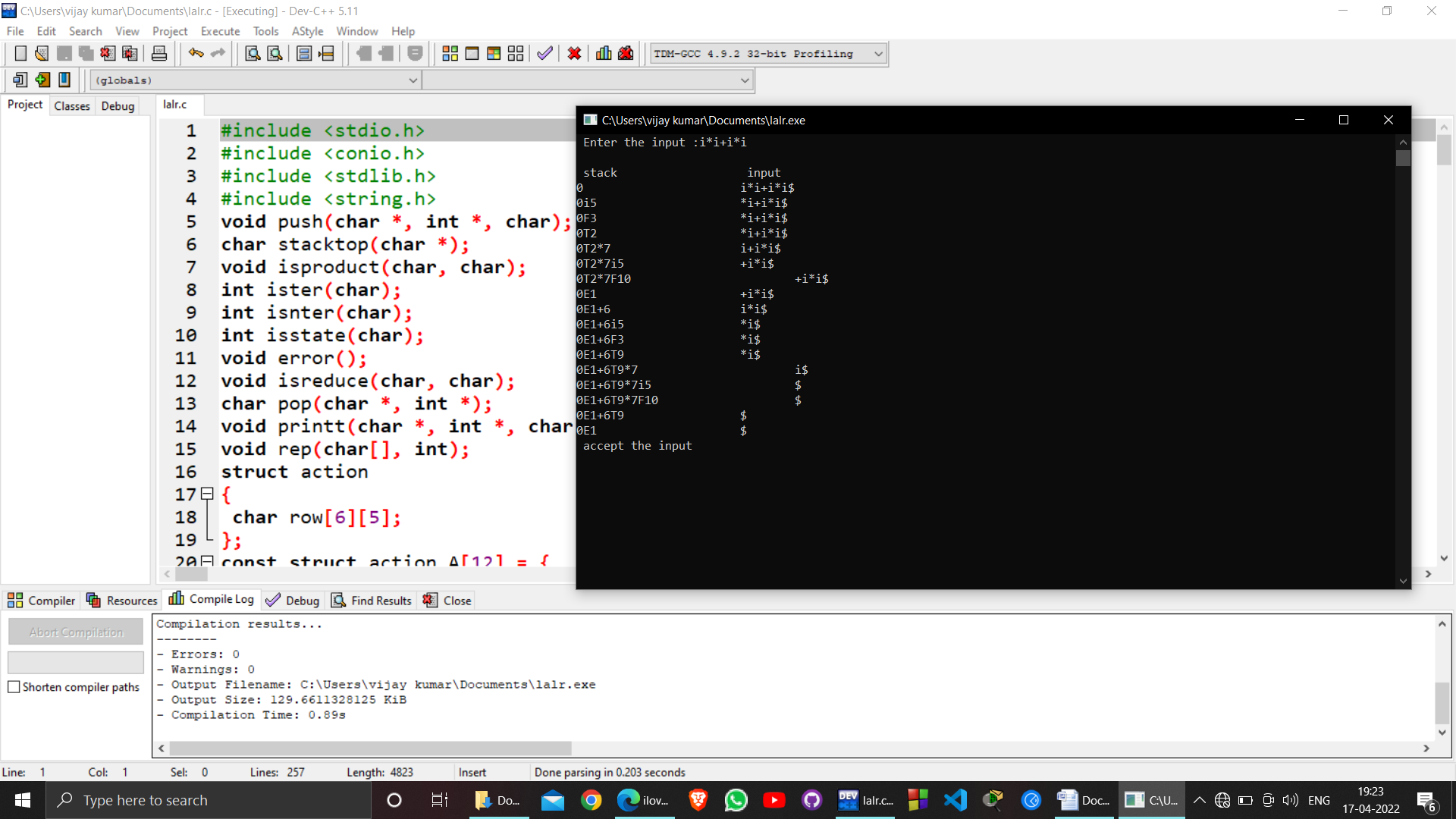
printf("%c", t[r]);

break;

}

}

**OUTPUT:**

****