# Scanner for C - language

**Report -1 for Compiler Design lab**

**Submitted by:**

1. **Korra Jagan Babu - 181CO228**
2. **Sasidhar Swarangi - 181CO245**
3. **Arjun A - 181CO109**
4. **Adithi Srinath - 181CO103**

# Table of contents

# ABSTRACT

This report contains the details of the task-1 finished as a part of Four phases of the C- Compiler. The lexical analyzer phase generates and identifies the tokens and classifies them. This task requires us to code a lex file with .l extension which does the same.

We have used the flex tool ( faster version of the traditional lex tool) to implement this lexical analysis. We have used regular expressions to analyse each token and perform actions accordingly.

We have also shown the screenshots of 8 test-cases (but actually ran 15 test cases) which has been executed in ubuntu.

**Input:**
We have used bash script for automation of the 15 test cases written in the C-language , where some of them also include the errors that are to be identified by the analyzer.

**Output:**
We have printed the class of every token encountered by the analyzer along with the Symbol table and Constant table.

**To run the test cases:**

Bash ./run.sh

# Introduction

The aim of the project is to use lex/flex tools to implement a lexical analyzer that generates and identifies tokens and classifies them.

The grammar of the source language is studied and the lexical components of the language are identified.These tokens are later returned to the scanner.

A symbol table is used to keep track and store information about the occurrence of various entities such as variable names, function names, objects, classes, etc. And a constant table is used to store the constants similarly. Both these tables use the hash organization and are also printed out as output.

The lex file program is of the format:

```
{ definitions }   // include declarations of constant, variable and regular definitions.
%%
 { rules }   // define the statement of form p1 {action1} p2 {action2}....pn {action}.
%%
```

```
{ user subroutines }
```

# Code explanation

## Definitions section

```
%{
    // Declaration of symbol table structure which contains the
name , type and length of a particular symbol

    // Declaration of constant table structure which contains name ,
type and length of each constant


    int hash(char *str)
    {
        // A hash function which calculates the value of a
particular string that is to be stored in the value index.
    }

    int symbolTable_check(char *str)
    {
        /* a function which inputs a symbol and checks if the
value index in the symbol table is empty or already has the same
symbol or if the symbol is
            elsewhere in the hash table and returns appropriate
flag */
    }
    int constantTable_check(char *str)
    {
        // checks if the value index in the constant table is empty
or if it already contains the constant or if the constant is found at
another index value and returns a flag accordingly.
    }
    void insert_symbol(char *str1, char *str2)
```

```
        {
            /*This function stores the symbol in the symbol      table if
the symbol is not found already by using the symbolTable_check
function. It puts the symbol at the
 VALUE index if it is empty and puts it in the next empty    index if
VALUE is already taken. */
}
        void insert_constant(char *str1, char *str2)
        {

          /*This function stores the constant in the constant table if the
constant is not found already by using the constantTable_check
function. It puts the constant at the
VALUE index if it is empty and puts it in the next empty index if
VALUE is already taken. */

        }
        void printST()
        {
            //This function prints the symbol table
        }
        void printCT()
        {
            //This function prints the constant table
        }

      %}
```

## This section includes the regular expressions for all the tokens

%%

- \n  {yylineno++;} // increments line number
- ([#]["  "]*({IN})[ ]*([<]?)([A-Za-z]+)[.]?([A-Za-z]*)([>]?))/["\n"|\/|" "|"\t"]//Matches #include<stdio.h> and other header files
- ([#]["  "]*({DE})["  "]*([A-Za-z]+)(" ")*[0-9]+)/["\n"|\/|" "|"\t"] //Matches the #define statements
- \/\/(.*) // Checks for single line comments
- \/*([^*]|[\r\n]|(*+([^*/]|[\r\n])))**+\/ //checks for multi line comments
- [ \n\t] ;
- ; // SEMICOLON DELIMITER
- , //COMMA DELIMITER
- \{ //OPENING BRACES
- \} //CLOSING BRACES
- \( //OPENING BRACKETS
- \) //CLOSING BRACKETS
- \[ //SQUARE OPENING BRACKETS
- \] //SQUARE CLOSING BRACKETS
- \: //COLON DELIMITER
- \\ //FSLASH
- \. //DOT DELIMITER
- auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|while|main/[\(|" "|\{|;|:|"\n"|"\t"] //KEYWORDS
- \"[^\n]*\"/[;|,|\)] //STRING CONSTANT
- \'[A-Z|a-z]\'/[;|,|\)|:] //Character CONSTANT
- [a-z|A-Z]([a-z|A-Z]|[0-9])*/\[ //ARRAY IDENTIFIER
-
- {operator}/[a-z]|[0-9]|;|" "|[A-Z]|\(|\"|\'|\)|\n|\t //OPERATOR

- [1-9][0-9]*|0/[;|,|" "|\)|<|>|=|\!|\||&|\+|\-|*|\/|\%|~|\]|\}|:|\n|\t|\^] //NUMBER CONSTANT
- ([0-9]*)\.([0-9]+)/[;|,|" "|\)|<|>|=|\!|\||&|\+|\-|*|\/|\%|~|\n|\t|\^] //Floating CONSTANT
- [A-Za-z_][A-Za-z_0-9]*/[" "|;|,|\(|\)|<|>|=|\!|\||&|\+|\-|*|\/|\%|~|\n|\.|\{|\^|\t] //IDENTIFIERS

- (.?) {

```
            if(yytext[0]=='#')
            //error in declaring the header files
        }

        else if(yytext[0]=='/')
        {
            //error in a single line comment
    }
    else if(yytext[0]=="")
    {
            //error- incomplete string
    }
    else
    {
            //other errors
    }
    printf("%s\n", yytext); // Print the tokens
    }
%%
```

# Code Implementation

```
%{
	#include <stdio.h>
	#include <string.h>


	struct symboltable
	{
		char name[100];
		char type[100];
		int length;
	}ST[1001];

	struct constanttable
	{
		char name[100];
		char type[100];
		int length;
	}CT[1001];

	int hash(char *str)
	{
		int value = 0;
		for(int i = 0 ; i < strlen(str) ; i++)
		{
			value = 10*value + (str[i] - 'A');
			value = value % 1001;
			while(value < 0)
				value = value + 1001;
		}
		return value;
	}

	int lookupST(char *str)
	{
		int value = hash(str);
		if(ST[value].length == 0)
		{
			return 0;
		}
		else if(strcmp(ST[value].name,str)==0)
		{
			return 1;
		}
```

```
        else
        {
                for(int i = value + 1 ; i!=value ; i = (i+1)%1001)
        {
                if(strcmp(ST[i].name,str)==0)
                {
                        return 1;
                }
        }
        return 0;
    }
}

int lookupCT(char *str)
{
        int value = hash(str);
        if(CT[value].length == 0)
                return 0;
        else if(strcmp(CT[value].name,str)==0)
                return 1;
        else
        {
                for(int i = value + 1 ; i!=value ; i = (i+1)%1001)
                {
                        if(strcmp(CT[i].name,str)==0)
                        {
                                return 1;
                        }
                }
                return 0;
        }
}

void insertST(char *str1, char *str2)
{
        if(lookupST(str1))
        {
            return;
    }
        else
        {
                int value = hash(str1);
                if(ST[value].length == 0)
                {
                        strcpy(ST[value].name,str1);
                        strcpy(ST[value].type,str2);
                        ST[value].length = strlen(str1);
```

```
                return;
        }

        int pos = 0;

        for (int i = value + 1 ; i!=value ; i = (i+1)%1001)
        {
                if(ST[i].length == 0)
                {
                        pos = i;
                        break;
                }
        }

        strcpy(ST[pos].name,str1);
        strcpy(ST[pos].type,str2);
        ST[pos].length = strlen(str1);
    }
}

void insertCT(char *str1, char *str2)
{
    if(lookupCT(str1))
        return;
    else
    {
        int value = hash(str1);
        if(CT[value].length == 0)
        {
            strcpy(CT[value].name,str1);
            strcpy(CT[value].type,str2);
            CT[value].length = strlen(str1);
            return;
        }

        int pos = 0;

        for (int i = value + 1 ; i!=value ; i = (i+1)%1001)
        {
            if(CT[i].length == 0)
            {
                pos = i;
                break;
            }
        }

        strcpy(CT[pos].name,str1);
```

```
                strcpy(CT[pos].type,str2);
                CT[pos].length = strlen(str1);
        }
}

void printST()
{
        for(int i = 0 ; i < 1001 ; i++)
        {
                if(ST[i].length == 0)
                {
                        continue;
                }

                printf("\t%s\t%s\n",ST[i].name, ST[i].type);
        }
}

void printCT()
{
        for(int i = 0 ; i < 1001 ; i++)
        {
                if(CT[i].length == 0)
                        continue;

                printf("\t%s\t%s\n",CT[i].name, CT[i].type);
        }
}

%}

DE "define"
IN "include"

operator
[[<][=]|[>][=]|[=][=]|[!][=]|[>]|[<]|[\|][\|]|[&][&]|[\!]|[=]|[\^]|[\+][=]|[\-][=]|[\*][=]|[\/][=]|[\%][=]|[\+][\+]|[\
-][\-]|[\+]|[\-]|[\*]|[\/]|[\%]|[&]|[\|]|[~]|[<][<]|[>][>]]


%%
yyleng
([#][" "]*({IN})[ ]*([<]?)([A-Za-z]+)[.]?([A-Za-z]*)([>]?))/["\n"|\/|" "|"\t"] {printf("\t%s
\t\t----------Pre Processor directive\n",yytext);}     //Matches #include<stdio.h>
([#][" "]*({DE})[" "]*([A-Za-z]+)(" ")*[0-9]+)/["\n"|\/|" "|"\t"] {printf("\t%s
\t\t---------Macro\n",yytext);} //Matches macro
\/\/(.*) {printf("\t%s \t\t---------- Single line comment\n", yytext);}
\/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+\/  {printf("\t%s \t\t--------- Multi line comment\n", yytext);}
```

```
[ \n\t] ;
; {printf("\t%s \t\t--------- SemiColon delimiter\n", yytext);}
, {printf("\t%s \t\t--------- Comma delimiter\n", yytext);}
\{ {printf("\t%s \t\t--------- Opening braces\n", yytext);}
\} {printf("\t%s \t\t--------- Closing braces\n", yytext);}
\( {printf("\t%s \t\t--------- Opening brackets\n", yytext);}
\) {printf("\t%s \t\t--------- Closing brackets\n", yytext);}
\[ {printf("\t%s \t\t--------- Square Opening brackets\n", yytext);}
\] {printf("\t%s \t\t--------- Square Closing brackets\n", yytext);}
\: {printf("\t%s \t\t--------- Colon Delimiter\n", yytext);}
\\ {printf("\t%s \t\t--------- Fslash\n", yytext);}
\. {printf("\t%s \t\t--------- Dot Delimiter\n", yytext);}
auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|
long|register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|vol
atile|while|main/[\(|" "|\{|;|:|"\n"|"\t"] {printf("\t%s \t\t---------- Keyword\n", yytext);
insertST(yytext, "KEYWORD");}
\"[^\n]*\"/[;|,|\)] {printf("\t%s \t\t--------- String Constant\n", yytext); insertCT(yytext,"String
Constant");}
\'[A-Z|a-z]\'/[;|,|\)|:] {printf("\t%s \t\t---------- Character Constant\n", yytext);
insertCT(yytext,"Character Constant");}
[a-z|A-Z]([a-z|A-Z]|[0-9])*/\[ {printf("\t%s \t\t--------- Array Identifier\n", yytext);
insertST(yytext, "Identifier");}

{operator}/[a-z]|[0-9]|;|" "|[A-Z]|\(|\"|\'|\)|\n|\t {printf("\t%s \t\t--------- Operator\n", yytext);}

[1-9][0-9]*|0/[;|,|" "|\)|<|>|=|\!|\||&|\+|\-|\*|\/|\%|~|\]|\}|:|\n|\t|\^] {printf("\t%s \t\t--------- Number
Constant\n", yytext); insertCT(yytext, "Nunmber Constant");}
([0-9]*)\.([0-9]+)/[;|,|" "|\)|<|>|=|\!|\||&|\+|\-|\*|\/|\%|~|\n|\t|\^] {printf("\t%s \t\t--------- Floating
Constant\n", yytext); insertCT(yytext, "Floating Constant");}
[A-Za-z_][A-Za-z_0-9]*/[" "|;|,|\(|\)|<|>|=|\!|\||&|\+|\-|\*|\/|\%|~|\n|\.|\{|\^|\t] {printf("\t%s
\t\t--------- Identifier\n", yytext); insertST(yytext, "Identifier");}


(.?) {
		if(yytext[0]=='#')
		{
		printf("\tError in Pre-Processor directive at line no. %d\n",yyleng);
	}
	    else if(yytext[0]=='/')
	    {
		printf("\tERR_UNMATCHED_COMMENT at line no. %d\n",yyleng);
	}
	else if(yytext[0]=="")
	{
		printf("\tERR_INCOMPLETE_STRING at line no. %d\n",yyleng);
	}
	else
```

```
	{
		printf("\tERROR at line no. %d\n",yyleng);
	}
	printf("\t%s\n", yytext);
	return 0;
}

%%

int main(int argc , char **argv){


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

	int i;
	for (i=0;i<1001;i++){
		ST[i].length=0;
		CT[i].length=0;
	}

	yyin = fopen(argv[1],"r");
	yylex();

	printf("\n\n\tSYMBOL TABLE\n\n");
	printST();
	printf("\n\n\tCONSTANT TABLE\n\n");
	printCT();
}

int yywrap(){
    return 1;
}
```

## Run.sh file:

```
#!/bin/bash

function run() {
	flex scanner.l && gcc lex.yy.c
	local total_testcases="$1"
	echo "Running: $total_testcases"
	local start=1
	while [ $start -le $total_testcases ]
	do
		printf
"\n\n-----------------------------------------------------------------------------------------------------------------------\n\t\t\
t\t\t"
		echo TestCase number :$start
		local filename="tests/test"$start".c"
		./a.out $filename
		((start++))
	done
}

number_of_files=`ls -l ./tests/ | egrep -c '^-'`
run $number_of_files
```

# LIST OF TOKENS THE CODE RECOGNISES

## Identifiers

- Variables
- Functions

## Keywords

- `Auto`
- `Break`
- `Case`
- `Char`
- `Const`
- `Continue`
- `Default`
- `Do`
- `Double / Float`
- `Else`
- `Enum`
- `Extern`
- `For`
- `Goto`
- `If`
- `Int`
- `Long`
- `Register`
- `Return`
- `Short`
- `Signed`
- `Sizeof`
- `Static`
- `Struct`
- `Switch`
- `Typedef`
- `Union`
- `Unsigned`
- `Void`
- `While`
- `Main`

# Operators

- [<][=] - Less than or equal to
- [>][=] - greater than or equal to
- [=][=] - equal to
- [!][=] - not equal to
- [>] - greater than
- [<] - less than
- [\|][\|] - or
- [&][&] - and
- [\!] - not
- [=] - assignment
- [\^] - xor
- [\+][=] - shorthand op
- [\-][=] - shorthand op
- [*][=] - shorthand op
- [\/][=] - shorthand op
- [\%][=] - shorthand op
- [\+][\+] - increment op
- [\-][\-] - decrement op
- [\+] - addition
- [\-] - subtraction
- [*] - multiplication
- [\/] - division
- [\%] - remainder
- [&] - bitwise and
- [\|] - bitwise or
- [~] - tilde op
- [<][<] - left shift
- [>][>] - right shift

# Code inputs/Test cases

This is the sample program , test case-5 in the submitted folder.

**Test Case 5 : Error free Code**

```
#include<stdio.h>
// While Loop
int main()
{
	int a = 5;
	while(a>0)
	{
		printf("Hello world");
		a--;
	}

	a=4;
	while(a>0)
	{
		printf("%d",a);
		a--;
		int b;
		b= 4;
		while(b>0)
		{
			printf("%d", a*b);
			b--;
		}
	}
}
```

# Output screenshots

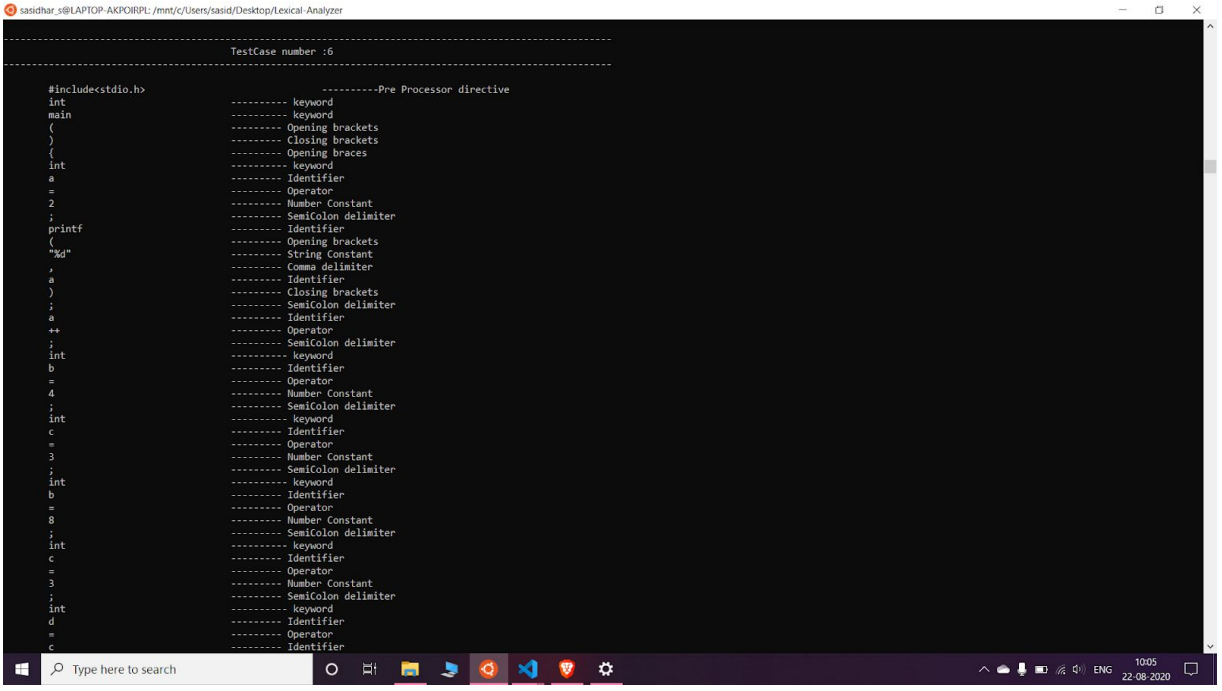
```
--------------------------------------------------------------------------------
                                TestCase number :5
--------------------------------------------------------------------------------

#include<stdio.h>           ----------Pre Processor directive
// While Loop          ---------- Single line comment
int             ---------- Keyword
main            ---------- Keyword
(               --------- Opening brackets
)               --------- Closing brackets
{               --------- Opening braces
int             ---------- Keyword
a               --------- Identifier
=               --------- Operator
5               --------- Number Constant
;               --------- SemiColon delimiter
while           ---------- Keyword
(               --------- Opening brackets
a               --------- Identifier
>               --------- Operator
0               --------- Number Constant
)               --------- Closing brackets
{               --------- Opening braces
printf          --------- Identifier
(               --------- Opening brackets
"Hello world"          --------- String Constant
)               --------- Closing brackets
;               --------- SemiColon delimiter
a               --------- Identifier
--              --------- Operator
;               --------- SemiColon delimiter
}               --------- Closing braces
a               --------- Identifier
=               --------- Operator
4               --------- Number Constant
;               --------- SemiColon delimiter
while           ---------- Keyword
(               --------- Opening brackets
a               --------- Identifier
>               --------- Operator
0               --------- Number Constant
)               --------- Closing brackets
{               --------- Opening braces
printf          --------- Identifier
(               --------- Opening brackets
"%d"            --------- String Constant
,               --------- Comma delimiter
a               --------- Identifier
)               --------- Closing brackets
;               --------- SemiColon delimiter
a               --------- Identifier
--              --------- Operator
;               --------- SemiColon delimiter
int             ---------- Keyword
b               --------- Identifier
;               --------- SemiColon delimiter
b               --------- Identifier
```

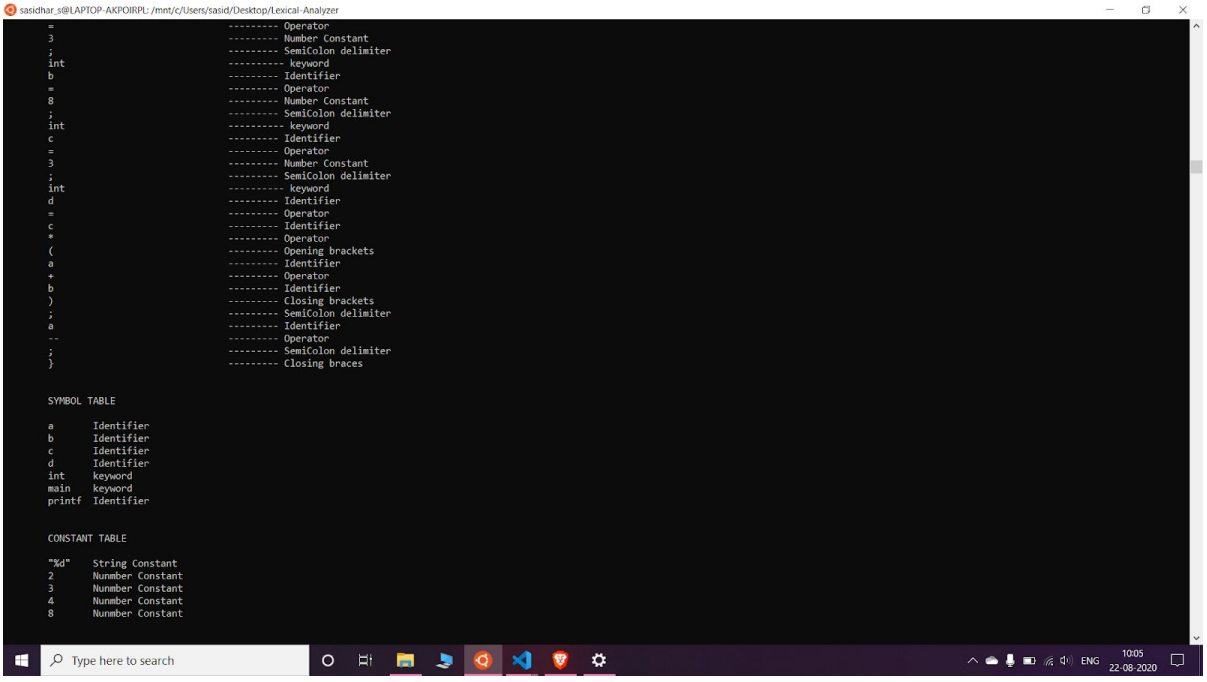
```
SYMBOL TABLE

a        Identifier
b        Identifier
int      KEYWORD
main     KEYWORD
printf   Identifier
while    KEYWORD


CONSTANT TABLE

"Hello world"   String Constant
"%d"     String Constant
0        Number Constant
4        Number Constant
5        Number Constant
```

This is the sample program , test case-6 in the submitted folder.

**Test Case 6 : Error free Code**.

```
#include<stdio.h>

int main()
{
    int a = 2;
    printf("%d",a);
    a++;
    int b = 4;
    int c = 3;
    int b = 8;
    int c = 3;
    int d = c*(a+b);
    a--;}
```

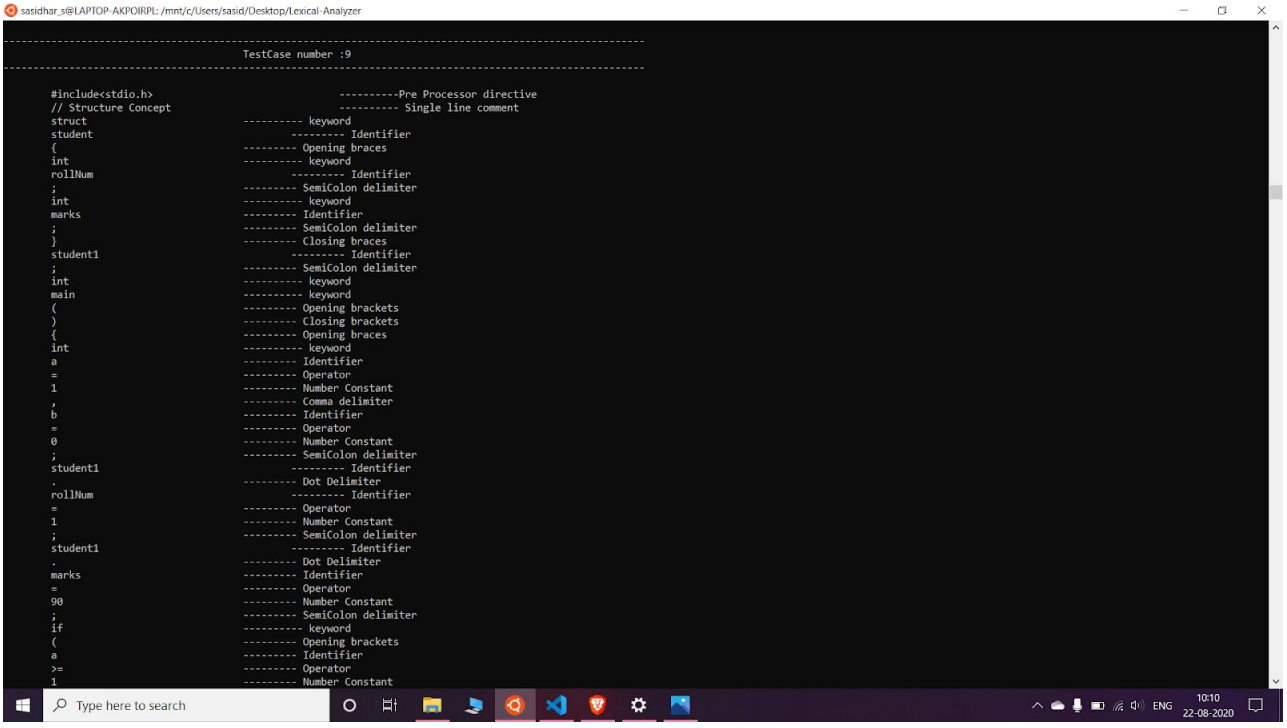


This is the sample program , test case-9 in the submitted folder.

**Test case 9 : Error Code.**

```
#include<stdio.h>
// Structure Concept
struct student
{
  int rollNum;
  int marks;
}student1;

int main()
{
 int a = 1, b=0;

 student1.rollNum = 1;
 student1.marks = 90;

 if(a >= 1 && a <= 10)
      b++;

 else
     {  b--;
     /* }
}
```



```
sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer
=                                   --------- Operator
90                                  --------- Number Constant
;                                   --------- SemiColon delimiter
if                                  --------- keyword
(                                   --------- Opening brackets
a                                   --------- Identifier
>=                                  --------- Operator
1                                   --------- Number Constant
&&                                  --------- Operator
a                                   --------- Identifier
<=                                  --------- Operator
10                                  --------- Number Constant
)                                   --------- Closing brackets
b                                   --------- Identifier
++                                  --------- Operator
;                                   --------- SemiColon delimiter
else                                ---------- keyword
{                                   --------- Opening braces
b                                   --------- Identifier
--                                  --------- Operator
;                                   --------- SemiColon delimiter
ERR_UNMATCHED_COMMENT at line no. 1
/


SYMBOL TABLE

struct  keyword
a       Identifier
b       Identifier
if      keyword
int     keyword
student Identifier
main    keyword
else    keyword
rollNum Identifier
marks   Identifier
student1        Identifier


CONSTANT TABLE

10      Nunmber Constant
90      Nunmber Constant
0       Nunmber Constant
1       Nunmber Constant


-------------------------------------------------------------------------------------------------------
                                    TestCase number :10
```

This is the sample program , test case-14 in the submitted folder.

**Test case 14 : Error Code.**

```
// Implicit Error that our Language doesn't support
#include<stdio.h>

int main() {
      char !hey;
      !hey = 'chai';
}

int fun(x);
{
      return 2020;;
}
```

```
sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer
CONSTANT TABLE

"%d"    String Constant
"Compiler Design"       String Constant
10      Nunmber Constant
0       Nunmber Constant
5       Nunmber Constant


-------------------------------------------------------------------------------------------------------
                                    TestCase number :14
-------------------------------------------------------------------------------------------------------

// Implicit Error that our Language doesn't support                    ---------- Single line comment
#include<stdio.h>                           ----------Pre Processor directive
int                                 ---------- keyword
main                                ---------- keyword
(                                   --------- Opening brackets
)                                   --------- Closing brackets
{                                   --------- Opening braces
char                                ---------- keyword
!                                   --------- Operator
hey                                 --------- Identifier
;                                   --------- SemiColon delimiter
!                                   --------- Operator
hey                                 --------- Identifier
=                                   --------- Operator
ERROR at line no. 1
'


SYMBOL TABLE

char    keyword
hey     Identifier
int     keyword
main    keyword


CONSTANT TABLE


-------------------------------------------------------------------------------------------------------
                                    TestCase number :15
-------------------------------------------------------------------------------------------------------

#include<stdio.h>                           ----------Pre Processor directive
#define X 100                       ---------Macro
```

This is the sample program , test case-2 in the submitted folder.

**Test case 2 : Error Code.**

```
#include<stdio.h>
#include<stdlib.h>
/*struct pair{
    int a;
    int b;
};*/

int fun(int x){
    return x*x;
}

int main(){
    int a=2,b,c,d,e,f,g,h;
    char x[20]='Compiler Design'
    c=a+b;
    d=a*b;
    e=a/b;
    f=a%b;
    g=a&&b;
    h=a||b;
    h=a*(a+b);
    h=a*a+b*b;
    h=fun(b);

    //This Test case contains
operator,structure,delimeters,Function,string;
}
```

sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer

```
-----------------------------------------------------------------------------------------------------------
                                        TestCase number :2
-----------------------------------------------------------------------------------------------------------

        #include<stdio.h>                                  ----------Pre Processor directive
        #include<stdlib.h>                                 ----------Pre Processor directive
        /*struct pair{
    int a;
    int b;
};*/                                    --------- Multi line comment
        int                                     ---------- keyword
        fun                                     --------- Identifier
        (                                       --------- Opening brackets
        int                                     ---------- keyword
        x                                       --------- Identifier
        )                                       --------- Closing brackets
        {                                       --------- Opening braces
        return                                  ---------- keyword
        x                                       --------- Identifier
        *                                       --------- Operator
        x                                       --------- Identifier
        ;                                       --------- SemiColon delimiter
        }                                       --------- Closing braces
        int                                     ---------- keyword
        main                                    ---------- keyword
        (                                       --------- Opening brackets
        )                                       --------- Closing brackets
        {                                       --------- Opening braces
        int                                     ---------- keyword
        a                                       --------- Identifier
        =                                       --------- Operator
        2                                       --------- Number Constant
        ,                                       --------- Comma delimiter
        b                                       --------- Identifier
        ,                                       --------- Comma delimiter
        c                                       --------- Identifier
        ,                                       --------- Comma delimiter
        d                                       --------- Identifier
        ,                                       --------- Comma delimiter
        e                                       --------- Identifier
        ,                                       --------- Comma delimiter
        f                                       --------- Identifier
        ,                                       --------- Comma delimiter
        g                                       --------- Identifier
        ,                                       --------- Comma delimiter
        h                                       --------- Identifier
        ;                                       --------- SemiColon delimiter
        char                                    ---------- keyword
```

Type here to search

ENG 10:51 22-08-2020

sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer

```
        2                                       --------- Number Constant
        ,                                       --------- Comma delimiter
        b                                       --------- Identifier
        ,                                       --------- Comma delimiter
        c                                       --------- Identifier
        ,                                       --------- Comma delimiter
        d                                       --------- Identifier
        ,                                       --------- Comma delimiter
        e                                       --------- Identifier
        ,                                       --------- Comma delimiter
        f                                       --------- Identifier
        ,                                       --------- Comma delimiter
        g                                       --------- Identifier
        ,                                       --------- Comma delimiter
        h                                       --------- Identifier
        ;                                       --------- SemiColon delimiter
        char                                    ---------- keyword
        x                                       --------- Array Identifier
        [                                       --------- Square Opening brackets
        20                                      --------- Number Constant
        ]                                       --------- Square Closing brackets
        =                                       --------- Operator
        ERROR at line no. 1
        '


        SYMBOL TABLE

        a       Identifier
        b       Identifier
        c       Identifier
        d       Identifier
        e       Identifier
        f       Identifier
        g       Identifier
        h       Identifier
        x       Identifier
        char    keyword
        fun     Identifier
        return  keyword
        int     keyword
        main    keyword


        CONSTANT TABLE

        20      Nunmber Constant
        2       Nunmber Constant
```

Type here to search

ENG 10:51 22-08-2020

**Test case 7:- Error free code**

```
#include<stdio.h>
// Cube of a Number
int cube(int a)
{
     return(a*a*a);
}

/*struct abc
{
     int a;
     char b;
};*/

int main()
{
     int num = 2;
     int num2 = cube(num);

     printf("Cube of %d is %d", num, num2);

     return 0;
}
```

sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer

```
-----------------------------------------------------------------------------------------------------
                                        TestCase number :7
-----------------------------------------------------------------------------------------------------

        #include<stdio.h>                                  ----------Pre Processor directive
        // Cube of a Number                                ---------- Single line comment
        int                             ---------- keyword
        cube                            --------- Identifier
        (                               --------- Opening brackets
        int                             ---------- keyword
        a                               --------- Identifier
        )                               --------- Closing brackets
        {                               --------- Opening braces
        return                          ---------- keyword
        (                               --------- Opening brackets
        a                               --------- Identifier
        *                               --------- Operator
        a                               --------- Identifier
        *                               --------- Operator
        a                               --------- Identifier
        )                               --------- Closing brackets
        ;                               --------- SemiColon delimiter
        }                               --------- Closing braces
        /*struct abc
{
        int a;
        char b;
};*/                            --------- Multi line comment
        int                             ---------- keyword
        main                            ---------- keyword
        (                               --------- Opening brackets
        )                               --------- Closing brackets
        {                               --------- Opening braces
        int                             ---------- keyword
        num                             --------- Identifier
        =                               --------- Operator
        2                               --------- Number Constant
        ;                               --------- SemiColon delimiter
        int                             ---------- keyword
        num2                            --------- Identifier
        =                               --------- Operator
        cube                            --------- Identifier
        (                               --------- Opening brackets
        num                             --------- Identifier
        )                               --------- Closing brackets
        ;                               --------- SemiColon delimiter
        printf                          --------- Identifier
        (                               --------- Opening brackets
```

Type here to search

ENG 10:58 22-08-2020

sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer

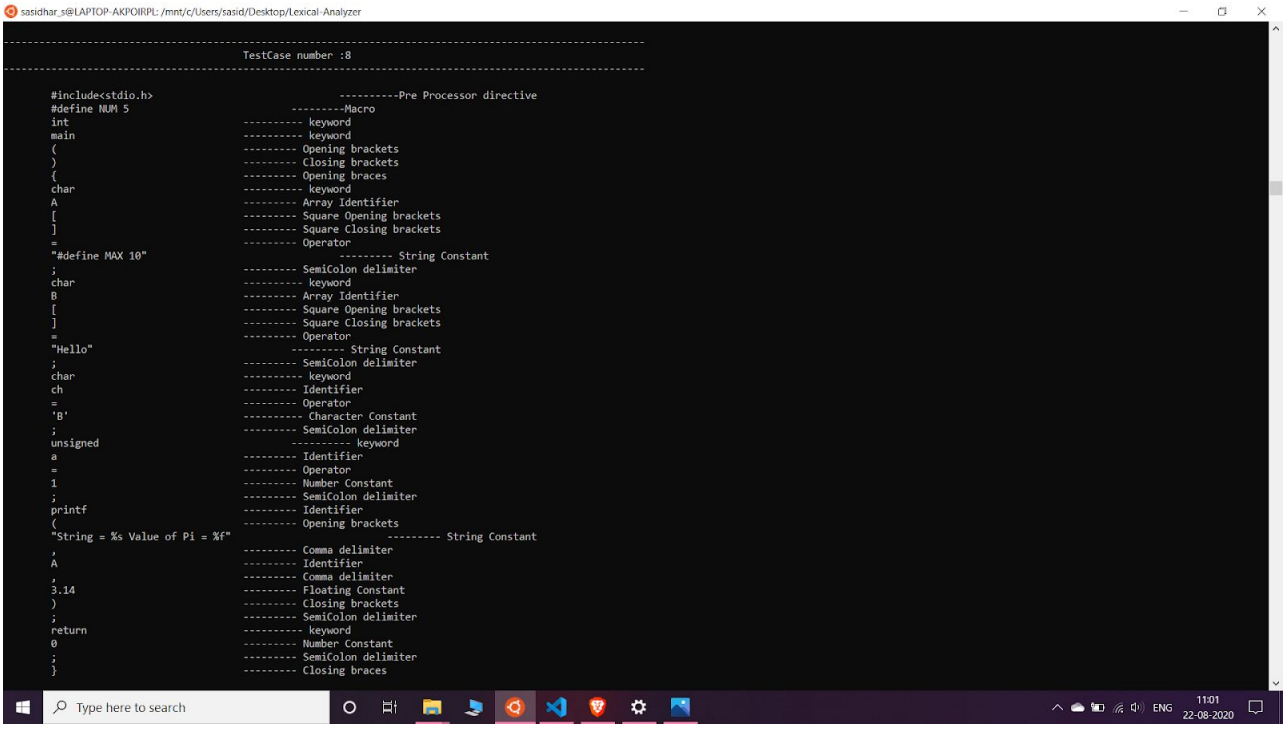
```
        num                             --------- Identifier
        =                               --------- Operator
        2                               --------- Number Constant
        ;                               --------- SemiColon delimiter
        int                             ---------- keyword
        num2                            --------- Identifier
        =                               --------- Operator
        cube                            --------- Identifier
        (                               --------- Opening brackets
        num                             --------- Identifier
        )                               --------- Closing brackets
        ;                               --------- SemiColon delimiter
        printf                          --------- Identifier
        (                               --------- Opening brackets
        "Cube of %d is %d"                          --------- String Constant
        ,                               --------- Comma delimiter
        num                             --------- Identifier
        ,                               --------- Comma delimiter
        num2                            --------- Identifier
        )                               --------- Closing brackets
        ;                               --------- SemiColon delimiter
        return                          ---------- keyword
        0                               --------- Number Constant
        ;                               --------- SemiColon delimiter
        }                               --------- Closing braces


        SYMBOL TABLE

        a       Identifier
        num     Identifier
        return  keyword
        int     keyword
        cube    Identifier
        num2    Identifier
        main    keyword
        printf  Identifier


        CONSTANT TABLE

        "Cube of %d is %d"      String Constant
        0       Nunmber Constant
        2       Nunmber Constant


-----------------------------------------------------------------------------------------------------
                                        TestCase number :8
-----------------------------------------------------------------------------------------------------
```

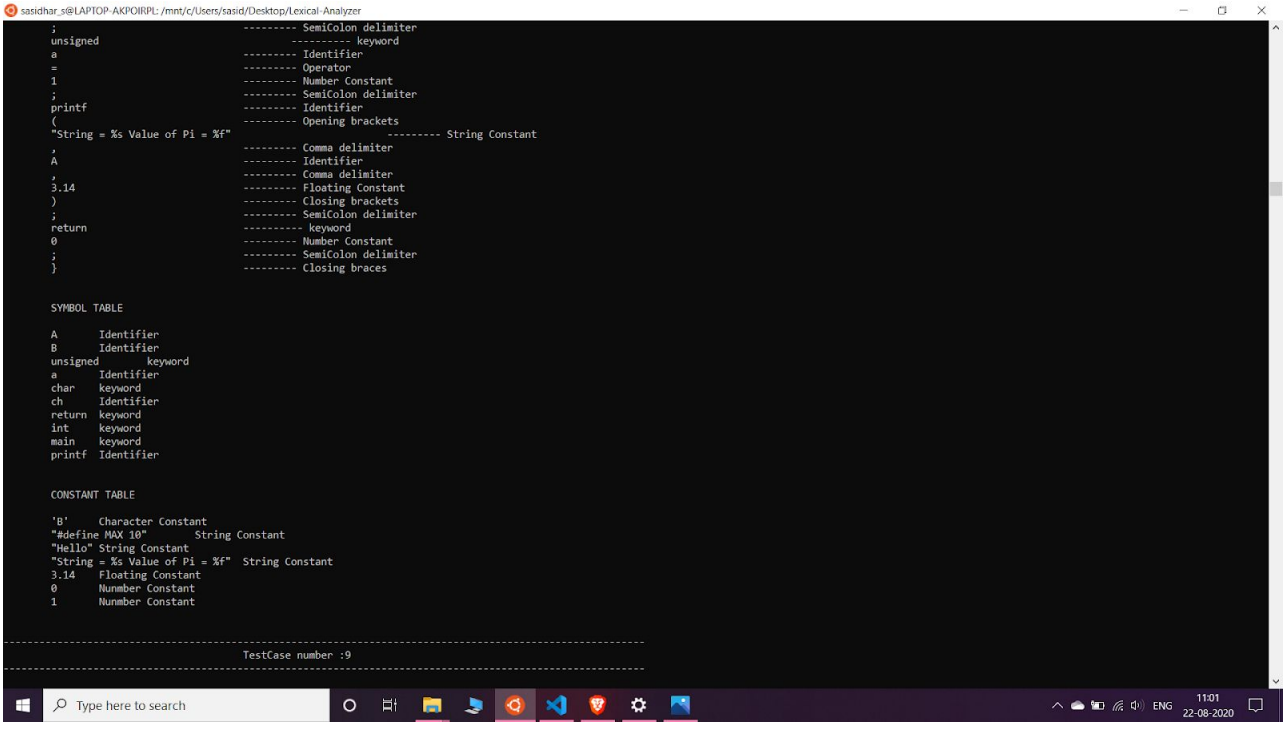
Type here to search

ENG 10:58 22-08-2020

## Test case 8:- Error free code

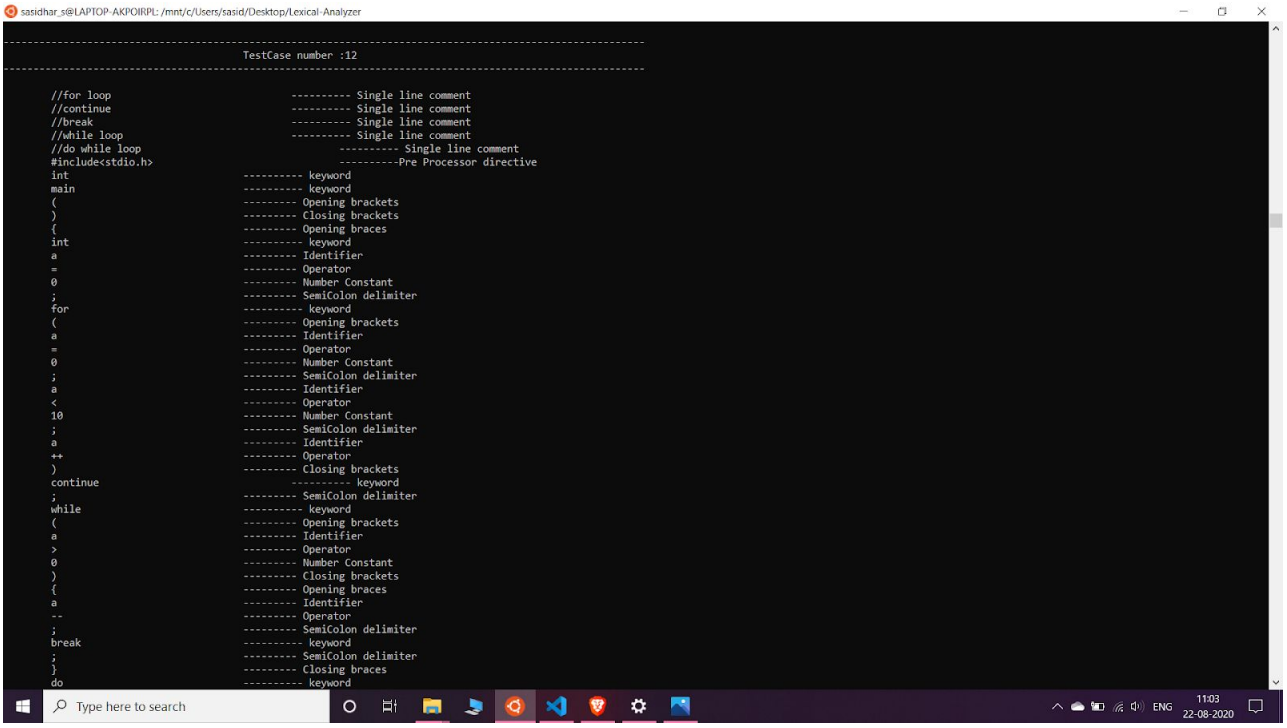
```
#include<stdio.h>
#define NUM 5

int main()
{
char A[] = "#define MAX 10";
char B[ ] = "Hello";
char ch  = 'B';
unsigned a = 1;
printf("String = %s Value of Pi = %f", A, 3.14);

     return 0;
}
```



**Test case 12: error free code**

```
//for loop
//continue
//break
//while loop
//do while loop

#include<stdio.h>

int main()
{
    int a=0;
    for (a = 0; a < 10; a++)
        continue;

    while(a>0) {
        a--;
        break;
    }

    do {
        a++;
    }while(a<10);
}
```



sasidhar_s@LAPTOP-AKPOIRPL: /mnt/c/Users/sasid/Desktop/Lexical-Analyzer

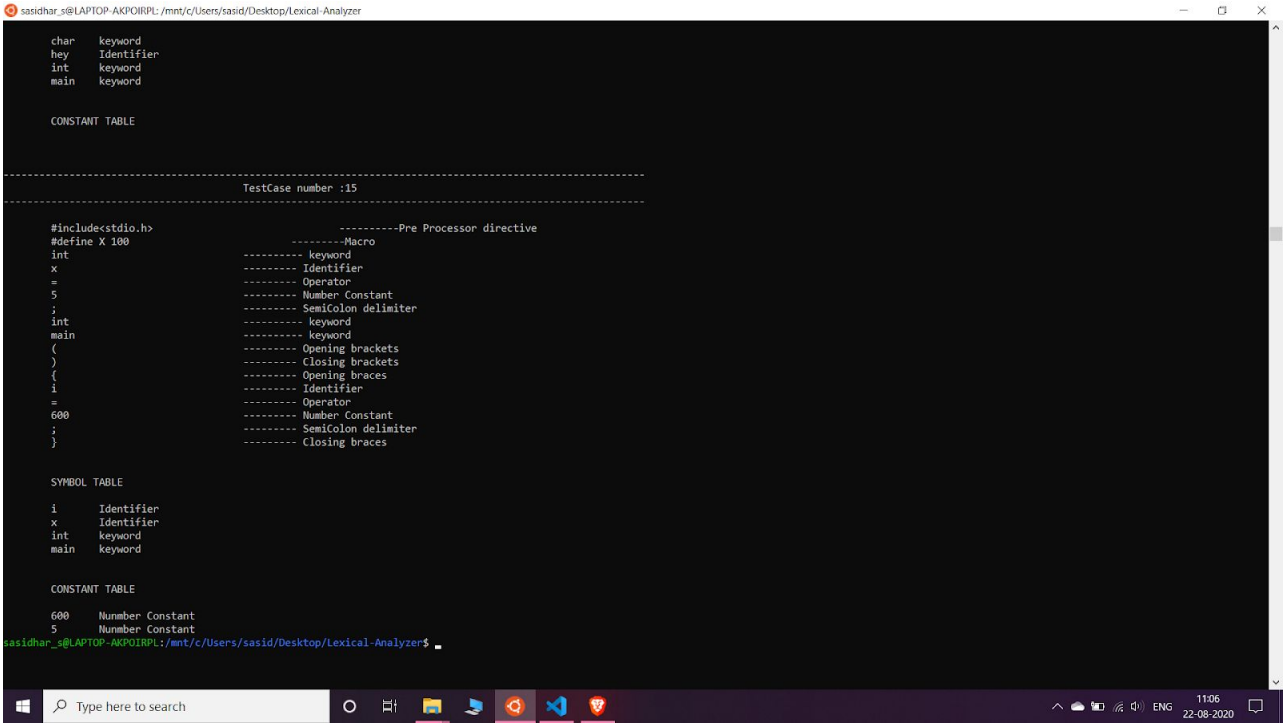
```
>                                   --------- Operator
0                                   --------- Number Constant
)                                   --------- Closing brackets
{                                   --------- Opening braces
a                                   --------- Identifier
--                                  --------- Operator
;                                   --------- SemiColon delimiter
break                               ---------- keyword
;                                   --------- SemiColon delimiter
}                                   --------- Closing braces
do                                  ---------- keyword
{                                   --------- Opening braces
a                                   --------- Identifier
++                                  --------- Operator
;                                   --------- SemiColon delimiter
}                                   --------- Closing braces
while                               ---------- keyword
(                                   --------- Opening brackets
a                                   --------- Identifier
<                                   --------- Operator
10                                  --------- Number Constant
)                                   --------- Closing brackets
;                                   --------- SemiColon delimiter
}                                   --------- Closing braces


SYMBOL TABLE

a       Identifier
for     keyword
continue        keyword
do      keyword
int     keyword
break   keyword
main    keyword
while   keyword


CONSTANT TABLE

10      Nunmber Constant
0       Nunmber Constant


-------------------------------------------------------------------------------------------------------
                                    TestCase number :13
-------------------------------------------------------------------------------------------------------

// Combination of  tests, comments                          ---------- Single line comment
#include<stdio.h>                           ----------Pre Processor directive
```

Type here to search

ENG 11:03 22-08-2020

**Test case 15:-**

```
#include<stdio.h>
#define X 100
int x = 5;
int main()
{
      i = 600;

}
```




# CONCLUSION

In this project, we learnt how to build a scanner for C-language and an Lexical Analyzer that supports nested comments, symbols and other constants and also return errors, using the flex tool and our knowledge in automata theory to write the regular expressions for the tokens.