**North Western University**



**Lab Report**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| SN | Topic | Page no. |
| 1. | Design a lexical analyzer |  |
| 2. | Identify whether the statement is comment or not |  |
| 3. | Identify strings from the input strings. |  |
| 4. | Recognize Token |  |
| 5. | Valid Identifier |  |

**North Western University**



**Lab-1**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Objectives:**

Design a lexical analyzer for given language and the lexical analyzer 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. Simulate the same in C language.

Write a C/C++ program to implement the design of a Lexical analyzer to recognize the tokens defined by the given grammar.

**Introduction:**

**Lexical Analysis** is the very first phase in the compiler designing. A Lexer takes the modified source code which is written in the form of sentences . In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyzer breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.

Programs that perform Lexical Analysis in compiler design are called lexical analyzers or lexers. A lexer contains tokenizer or scanner. If the lexical analyzer detects that the token is invalid, it generates an error. The role of Lexical Analyzer in compiler design is to read character streams from the source code, check for legal tokens, and pass the data to the syntax analyzer when it demands.

**Description:**

**#include <stdio.h>**

**int main() {**

**char arr[150];**

**int vowels, consonant, digit, space,special\_\_character;**

**// initialize all variables to 0**

**vowels = consonant = digit = space = special\_\_character = 0;**

**// get full line of string input**

**printf("Ent;er a line of string: ");**

**//fflush(stdin);**

**gets(arr);**

**//int sizeofArray = sizeof(arr);**

**//arr[i] != '\0'**

**// loop through each character of the string**

**for (int i = 0; arr[i] != '\0' ; ++i) {**

**// convert character to lowercase**

**arr[i] = tolower(arr[i]);**

**// check if the character is a vowel**

**if (arr[i] == 'a' || arr[i] == 'e' || arr[i] == 'i' ||**

**arr[i] == 'o' || arr[i] == 'u') {**

**// increment value of vowels by 1**

**++vowels;**

**}**

**// if it is not a vowel and if it is an alphabet, it is a consonant**

**else if ((arr[i] >= 'a' && arr[i] <= 'z')) {**

**++consonant;**

**}**

**// check if the character is a digit**

**else if (arr[i] >= '0' && arr[i] <= '9') {**

**++digit;**

**}**

**// check if the character is an empty space**

**else if (arr[i] == ' ') {**

**++space;**

**}**

**else**

**{ ++ special\_\_character; }**

**}**

**printf("Vowels: %d", vowels);**

**printf("\nConsonants: %d", consonant);**

**printf("\nDigits: %d", digit);**

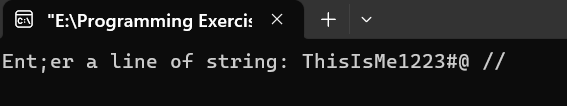
**printf("\nWhite spaces: %d", space);**

**printf("\nSpecial Character: %d", special\_\_character);**

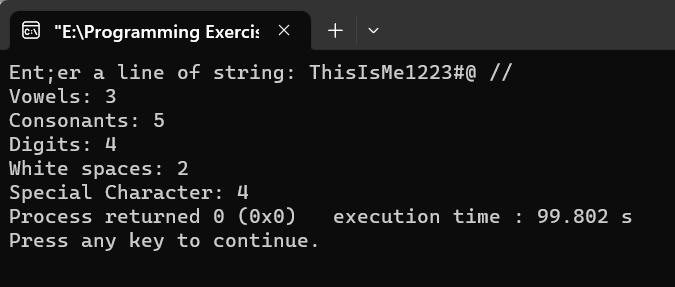
**return 0;**

**}**

**Input:**

****

**Output:**

****

**Conclusion:**

* Lexical analysis is the very first phase in the compiler design
* Lexemes and Tokens are the sequence of characters that are included in the source program according to the matching pattern of a token
* Lexical analyzer is implemented to scan the entire source code of the program
* Lexical analyzer helps to identify token into the symbol table
* A character sequence which is not possible to scan into any valid token is a lexical error
* Removes one character from the remaining input is useful Error recovery method
* Lexical Analyser scan the input program while parser perform syntax analysis
* It eases the process of lexical analysis and the syntax analysis by eliminating unwanted tokens
* Lexical analyzer is used by web browsers to format and display a web page with the help of parsed data from JavsScript, HTML, CSS
* The biggest drawback of using Lexical analyzer is that it needs additional runtime overhead is required to generate the lexer tables and construct the tokens

**North Western University**



**Lab-2**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Objectives:**

Write a C program to identify whether a given line is a comment or not.

**Introduction:**

Given a string S, representing a line in a program, the task is to check if the given string is a comment or not.

Types of comments in programs:

Single Line Comment: Comments preceded by a Double Slash (‘//’)

Multi-line Comment: Comments starting with (‘/\*’) and ending with (‘\*/’).

The idea is to check whether the input string is a comment or not. Below are the steps:

Check if at the first Index(i.e. index 0) the value is ‘/’ then follow below steps else print “It is not a comment”.

If line[0] == ‘/’:

If line[1] == ‘/’, then print “It is a single line comment”.

If line[1] == ‘\*’, then traverse the string and if any adjacent pair of ‘\*’ & ‘/’ is found then print “It is a multi-line comment”.

Otherwise, print “It is not a comment”.

Description:

#include<stdio.h>

#include<string.h>

#include<conio.h>

int main()

{

char str[100];

int i=2,a=0;

printf("Enter the String: ");

gets(str);

printf("\nOutput:\n");

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

{

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

{

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

}

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

{

for(i=2; i<=100; i++)

{

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

{

printf("It is a Multi-Line comment\n");

a=1;

break;

}

else

{

continue; }

}

if(a==0)

{

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

}

}

else

{

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

}

}

else

{

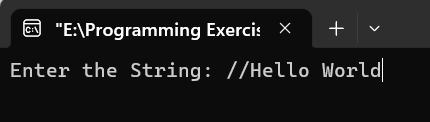
printf("It is not a comment\n");

}

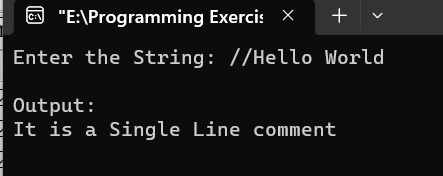
getch();

}

Input:



Output:



**Conclusion:**

A comment statement is where the programmer place a remark in the source code. The content of the comment is ignored by the compiler.

**North Western University**



**Lab-3**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Objectives:** Write a C program to recognize strings under 'a\*', 'a\*b+', 'abb'.

Identify strings from the input strings.

Introduction:

By using transition diagram we verify input of the state. If the state recognize the given pattern rule. Then print string is accepted under a\*/ a\*b+/ abb. Else print string not accepted.

**Description:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

char s[20],c;

int state=0,i=0;

printf("\n Enter a string:");

gets(s);

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

{

switch(state)

{

case 0: c=s[i++];

if(c=='a')

state=1;

else if(c=='b')

state=2;

else

state=6;

break;

case 1: c=s[i++];

if(c=='a')

state=3;

else if(c=='b')

state=4;

else

state=6;

break;

case 2: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=2;

else

state=6;

break;

case 3: c=s[i++];

if(c=='a')

state=3;

else if(c=='b')

state=2;

else

state=6;

break;

case 4: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=5;

else

state=6;

break;

case 5: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=2;

else

state=6;

break;

case 6: printf("\n %s is not recognised.",s);

exit(0);

}

}

if(state==1)

printf("\n %s is accepted under rule 'a'",s);

else if((state==2)||(state==4))

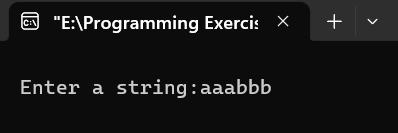
printf("\n %s is accepted under rule 'a\*b+'",s);

else if(state==5)

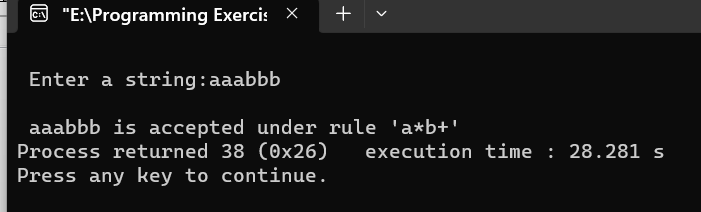
printf("\n %s is accepted under rule 'abb'",s);

}

**Input:**



**Output:**



**Conclusion:**

* If the first character is always a, then traverse the remaining string and check if any of the characters is a or ab.
* If the first character in the string is ‘b’ then it prints “string is rejected”.
* If there exists any character other than a or b while traversing, then it prints “entered value is wrong”.
* Otherwise, print “string is accepted”.

**North Western University**



**Lab-4**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Objectives:**

Write a C program to develop a lexical analyzer to recognize a few patterns in C.

Introduction:

**Introduction :**

Lexical Analysis is the first step of the compiler which reads the source code one character at a time and transforms it into an array of tokens. The token is a meaningful collection of characters in a program. These tokens can be keywords including do, if, while etc. and identifiers including x, num, count, etc.

Description:

#include<string.h>

#include<ctype.h>

#include<stdio.h>

void keyword(char str[10])

{

if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",str )==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0)

printf("\n%s is a keyword",str);

else

printf("\n%s is an identifier",str);

printf("\n%s is an identifier",str);

}

int main()

{

FILE \*f1,\*f2,\*f3;

char c,str[10],st1[10];

int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;

printf("\nEnter the c Program: ");/\*gets(st1);\*/

f1=fopen("input","w");

while((c=getchar())!=EOF)

putc(c,f1);

fclose(f1);

f1=fopen("input","r");

f2=fopen("identifier","w");

f3=fopen("specialchar","w");

while((c=getc(f1))!=EOF)

{

if(isdigit(c))

{

tokenvalue=c-'0';

c=getc(f1);

while(isdigit(c))

{

tokenvalue\*=10+c-'0';

c=getc(f1);

}

num[i++]=tokenvalue;

ungetc(c,f1);

}

else if(isalpha(c))

{

putc(c,f2);

c=getc(f1);

while(isdigit(c)||isalpha(c)||c=='\_'||c=='$')

{

putc(c,f2);

c=getc(f1);

}

putc(' ',f2);

ungetc(c,f1);

}

else if(c==' '||c=='\t')

printf(" ");

else if(c=='\n')

lineno++;

else

putc(c,f3);

}

fclose(f2);

fclose(f3);

fclose(f1);

printf("\nThe no's in the program are");

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

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

printf("\n");

f2=fopen("identifier","r");

k=0;

printf("The keywords and identifiersare:");

while((c=getc(f2))!=EOF)

{

if(c!=' ')

str[k++]=c;

else

{

str[k]='\0';

keyword(str);

k=0;

}

}

fclose(f2);

f3=fopen("specialchar","r");

printf("\nSpecial characters are");

while((c=getc(f3))!=EOF)

printf("%c",c);

printf("\n");

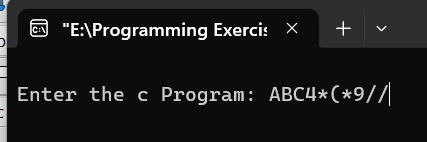
fclose(f3);

printf("Total no. of lines are:%d",lineno);

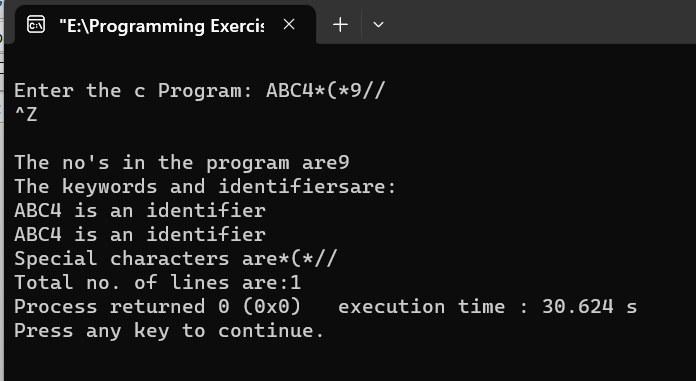
return 0;

}

**Input:**



**Output:**



**Conclusion:**

Input preprocessing: This stage involves cleaning up the input text and preparing it for lexical analysis. This may include removing comments, whitespace, and other non-essential characters from the input text.

Tokenization: This is the process of breaking the input text into a sequence of tokens. This is usually done by matching the characters in the input text against a set of patterns or regular expressions that define the different types of tokens.

Token classification: In this stage, the lexer determines the type of each token. For example, in a programming language, the lexer might classify keywords, identifiers, operators, and punctuation symbols as separate token types.

Token validation: In this stage, the lexer checks that each token is valid according to the rules of the programming language. For example, it might check that a variable name is a valid identifier, or that an operator has the correct syntax.

Output generation: In this final stage, the lexer generates the output of the lexical analysis process, which is typically a list of tokens. This list of tokens can then be passed to the next stage of compilation or interpretation.

**North Western University**



**Lab-5**

Course Title: Compiler Design Sessional

Course Code: CSE-4104

|  |  |
| --- | --- |
| Submitted by:  **Sajib Bhattacharjee**  **ID:20201070010**  Department of CSE  North Western University, Khulna. | Submitted to:  **Name: Abu Naim Khan**  Lecturer,  Department of CSE.  North Western University, Khulna. |

Submission Date: 01/06/2023 Teacher’s Signature

**Objectives:**

Write a C program to test whether a given identifier is valid or not .

**Introduction:**

 Read the given input string. Check the initial character of the string is numerical or any special character except ‘\_’ then print it is not a valid identifier. Otherwise print it as valid identifier if remaining characters of string doesn’t contains any special characters except ‘\_’.

**Description:**

#include <stdio.h>

#include<stdlib.h>

void 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++;

}

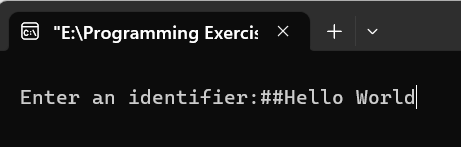
if(flag==1)

printf("\n Valid identifier");

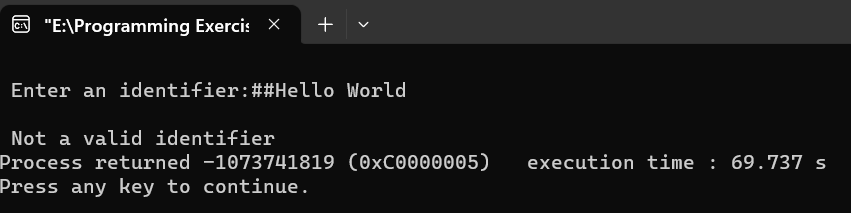
getch();

}

**Input:**



**Output:**



**Conclusion:**

It must start with an either underscore(\_) or any of the characters from the ranges [‘a’, ‘z’] and [‘A’, ‘Z’].

There must not be any white space in the string.

And, all the subsequent characters after the first character must not consist of any special characters like $, #, % etc.