University of Asia Pacific

Department of Computer Science & Engineering

Course code: CSE 430

Course Title: Compiler Design Lab

Submitted To	**Submitted By**
Baivab Das Lecturer, CSE, UAP	Asma Sultana Id: 20101084 Section: B

Lab 6

Problem

Create a lexical analyzer using lex on Linux environment

Description of the problem

Lex is a tool or software which automatically generates a lexical analyzer (finite Automata). It takes as its input a LEX source program and produces lexical Analyzer as its output. Lexical Analyzer will convert the input string entered by the user into tokens as its output. We define regular expressions by using lex, where has keywords, digits, identifiers and comments.

File format of Lex-

```
{ definitions }
%%
{ rules }
%%
{ user subroutines }
```

Definitions include declarations of constant, variable and regular definitions.
Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.
Where p describes the regular expression and action describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

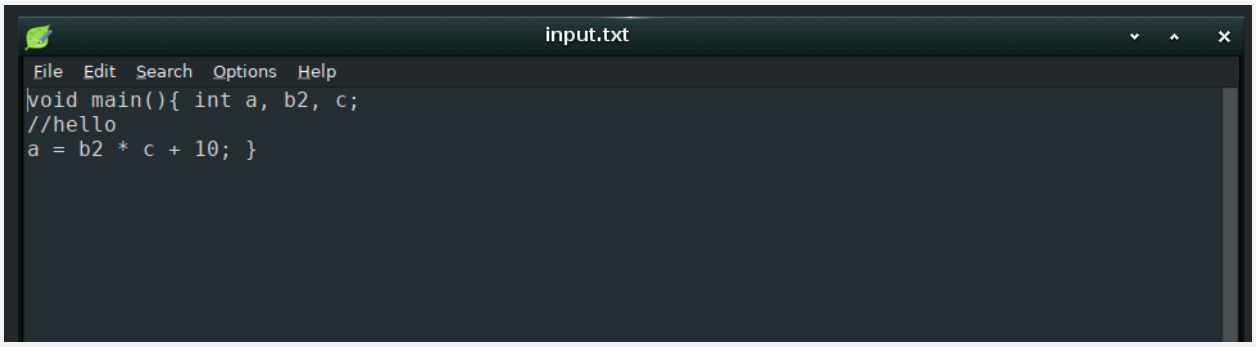
Lex Code

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

#define MAX_TOKENS 100

char printed_tokens[MAX_TOKENS][50];
int token_count = 0;

int is_in_set(const char *token) {
    for (int i = 0; i < token_count; i++) {
        if (strcmp(printed_tokens[i], token) == 0) {
            return 1;
        }
    }
    return 0;
}

void add_to_set(const char *token) {
    if (token_count < MAX_TOKENS) {
        strcpy(printed_tokens[token_count++], token);
    }
}

int is_keyword(const char *buffer) {
    const char *keywords[] = {
        "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"
    };
    int num_keywords = sizeof(keywords) / sizeof(keywords[0]);
    for (int i = 0; i < num_keywords; i++) {
        if (strcmp(keywords[i], buffer) == 0) {
            return 1;
        }
    }
    return 0;
}
%}
```

```
%%

[a-zA-Z_][a-zA-Z0-9_]* {
                    char token[100];
                    if (is_keyword(yytext)) {
                        snprintf(token, sizeof(token), "%s - keyword", yytext);
                    } else {
                        snprintf(token, sizeof(token), "%s - identifier", yytext);
                    }
                    if (!is_in_set(token)) {
                        printf("%s\n", token);
                        add_to_set(token);
                    }
                  }
"//".*            { if (!is_in_set(yytext)) { printf("%s - comment\n", yytext); add_to_set(yytext); } }
[0-9]+            { if (!is_in_set(yytext)) { printf("%s - digit\n", yytext); add_to_set(yytext); } }
[ \t\n]+           /* Ignore whitespace */
.                 /* Ignore any other character */

%%

int main() {
    yylex();
    return 0;
}

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

Sample Input:

```
void main(){ int a, b2, c; //hello a = b2 * c + 10; }
```



Observed Output