# Compiler Construction Assignment #3

Name: Jawad Ahmed
Section: BCS-6A
Instructor: Usman Wajid

## Instructions

- Rewrite the lexical analyzer code (in assignment 02) in Lex tool (Latest version Flex)
- Submit only the Lex file as your assignment
- Also submit the screen shot of your output
- Your assignment 02 and assignment 03 should link with each other

## Lex Code

The lex code is shown in fiqure 1.

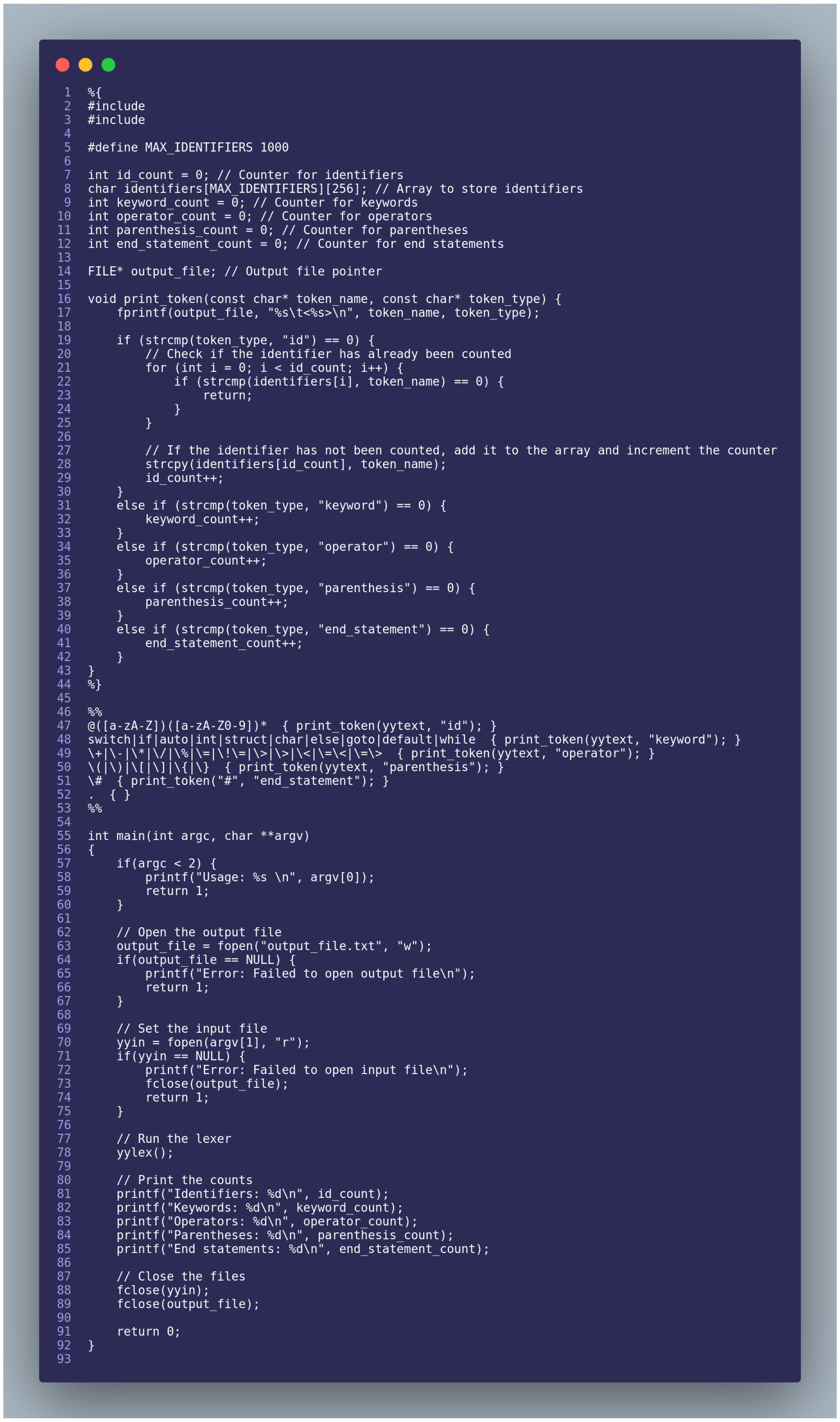
## Output

### Inputfile

The contents of the input file is shown in figure 2.

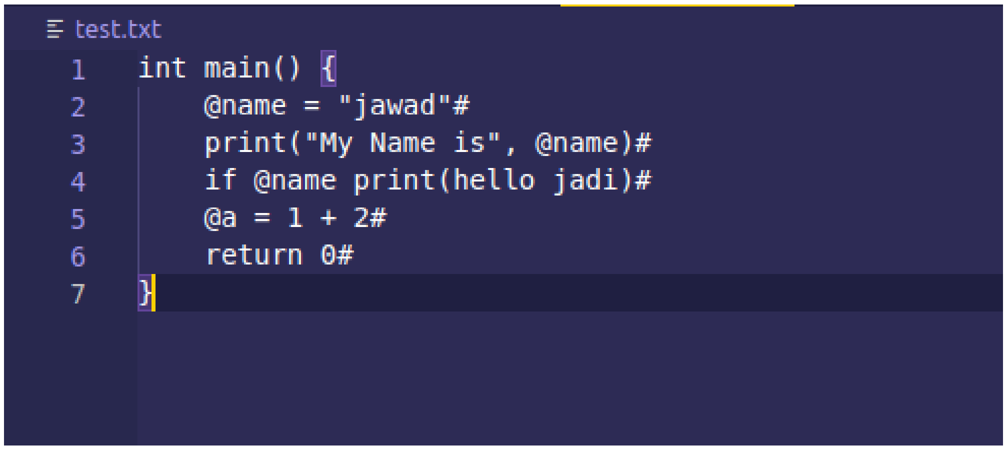
### Output Of the code

The output given by the above lex code is given in figure 3.



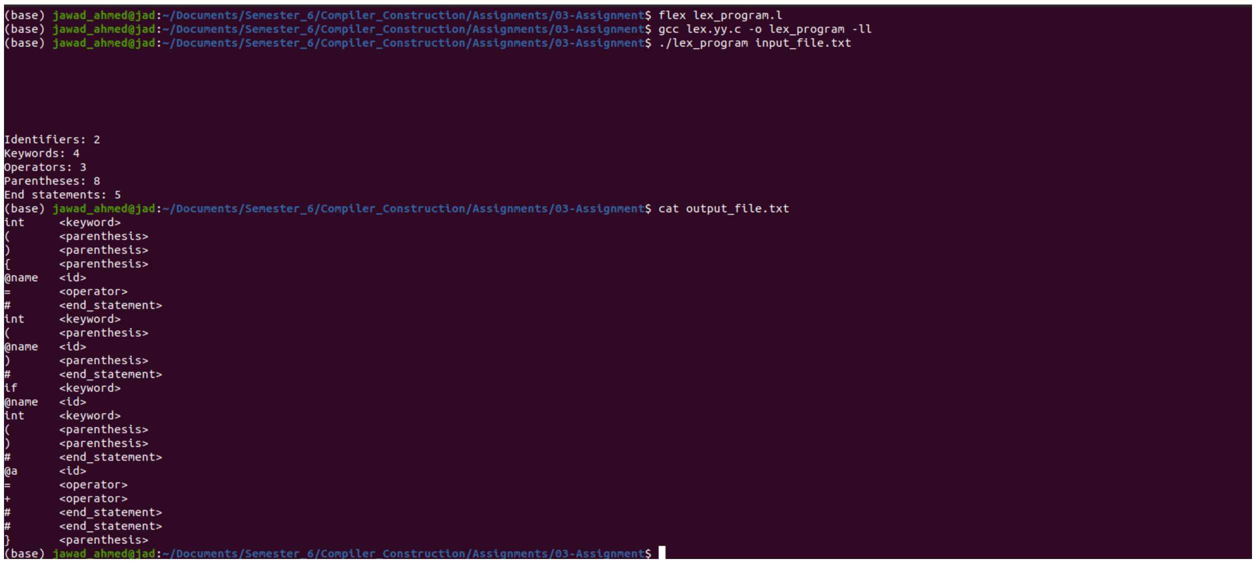
```
%{
#include
#include

#define MAX_IDENTIFIERS 1000

int id_count = 0; // Counter for identifiers
char identifiers[MAX_IDENTIFIERS][256]; // Array to store identifiers
int keyword_count = 0; // Counter for keywords
int operator_count = 0; // Counter for operators
int parenthesis_count = 0; // Counter for parentheses
int end_statement_count = 0; // Counter for end statements

FILE* output_file; // Output file pointer

void print_token(const char* token_name, const char* token_type) {
    fprintf(output_file, "%s\t<%s>\n", token_name, token_type);

    if (strcmp(token_type, "id") == 0) {
        // Check if the identifier has already been counted
        for (int i = 0; i < id_count; i++) {
            if (strcmp(identifiers[i], token_name) == 0) {
                return;
            }
        }

        // If the identifier has not been counted, add it to the array and increment the counter
        strcpy(identifiers[id_count], token_name);
        id_count++;
    }
    else if (strcmp(token_type, "keyword") == 0) {
        keyword_count++;
    }
    else if (strcmp(token_type, "operator") == 0) {
        operator_count++;
    }
    else if (strcmp(token_type, "parenthesis") == 0) {
        parenthesis_count++;
    }
    else if (strcmp(token_type, "end_statement") == 0) {
        end_statement_count++;
    }
}
%}

%%
@([a-zA-Z])([a-zA-Z0-9])*  { print_token(yytext, "id"); }
switch|if|auto|int|struct|char|else|goto|default|while  { print_token(yytext, "keyword"); }
\+|\-|\*|\/|\%|\=|\!\=|\>|\>|\<|\=\<|\=\>  { print_token(yytext, "operator"); }
\(|\)|\[|\]|\{|\}  { print_token(yytext, "parenthesis"); }
\#  { print_token("#", "end_statement"); }
.  { }
%%

int main(int argc, char **argv)
{
    if(argc < 2) {
        printf("Usage: %s \n", argv[0]);
        return 1;
    }

    // Open the output file
    output_file = fopen("output_file.txt", "w");
    if(output_file == NULL) {
        printf("Error: Failed to open output file\n");
        return 1;
    }

    // Set the input file
    yyin = fopen(argv[1], "r");
    if(yyin == NULL) {
        printf("Error: Failed to open input file\n");
        fclose(output_file);
        return 1;
    }

    // Run the lexer
    yylex();

    // Print the counts
    printf("Identifiers: %d\n", id_count);
    printf("Keywords: %d\n", keyword_count);
    printf("Operators: %d\n", operator_count);
    printf("Parentheses: %d\n", parenthesis_count);
    printf("End statements: %d\n", end_statement_count);

    // Close the files
    fclose(yyin);
    fclose(output_file);

    return 0;
}

```

Figure 1: Lex Code



```
int main() {
    @name = "jawad"#
    print("My Name is", @name)#
    if @name print(hello jadi)#
    @a = 1 + 2#
    return 0#
}
```

Figure 2: Input File



```
(base) jawad_ahmed@jad:~/Documents/Semester_6/Compiler_Construction/Assignments/03-Assignment$ flex lex_program.l
(base) jawad_ahmed@jad:~/Documents/Semester_6/Compiler_Construction/Assignments/03-Assignment$ gcc lex.yy.c -o lex_program -ll
(base) jawad_ahmed@jad:~/Documents/Semester_6/Compiler_Construction/Assignments/03-Assignment$ ./lex_program input_file.txt






Identifiers: 2
Keywords: 4
Operators: 3
Parentheses: 8
End statements: 5
(base) jawad_ahmed@jad:~/Documents/Semester_6/Compiler_Construction/Assignments/03-Assignment$ cat output_file.txt
int     <keyword>
(       <parenthesis>
)       <parenthesis>
{       <parenthesis>
@name   <id>
=       <operator>
#       <end_statement>
int     <keyword>
(       <parenthesis>
@name   <id>
)       <parenthesis>
#       <end_statement>
if      <keyword>
@name   <id>
int     <keyword>
(       <parenthesis>
)       <parenthesis>
#       <end_statement>
@a      <id>
=       <operator>
+       <operator>
#       <end_statement>
#       <end_statement>
}       <parenthesis>
(base) jawad_ahmed@jad:~/Documents/Semester_6/Compiler_Construction/Assignments/03-Assignment$
```

Figure 3: Output