Assignment 1:

1. Write a program to identify delimiters in an input file

Code:

```
#include <stdio.h>
int main()
{
        char c;
        FILE *file;
        file = fopen("input.txt","r");
        if(file==NULL)
        {
                printf("Error opening file");
                return 1;
        }
        while((c=getc(file)) != EOF)
        {
                if(c==' ' || c==',' || c=='.' || c==';' || c=='?' || c=='!' ||
c=='\n')
                        printf("%c is a delimiter\n",c);
        }
        fclose(file);
        return 0;
}
```

2. Write a C program to identify space, newline and tab in an input file.

Code:

```
#include <stdio.h>
int main()
{
        char c;
        FILE *file;
        file = fopen("input.txt","r");
        if(file==NULL)
        {
                printf("Error opening file");
                return 1;
        }
        while((c=getc(file)) != EOF)
        {
                if(c==' ')
                        printf("%c is a space\n",c);
                else if(c=='\t')
                        printf("%c is a tab character\n",c);
                else if(c=='\n')
                        printf("%c is a new line feed\n",c);
        }
        fclose(file);
        return 0;
}
```

3. Write a C program to identify operators in an input file.

Code:

```
#include <stdio.h>
int main()
{
        char c;
        FILE *file;
        file = fopen("input.txt","r");
        if(file==NULL)
        {
                printf("Error opening file");
                return 1;
        }
        while((c=getc(file)) != EOF)
        {
                if(c=='+' || c=='-' || c=='*' || c=='/' || c=='%' || c=='>' ||
c=='<' || c=='=' || c=='!')
                        printf("%c is an operator\n",c);
        }
        fclose(file);
        return 0;
}
```

4. Write a C program to identify all the delimiters, operators in an input C program. (Consider single character processing)

Code:

```
#include <stdio.h>
int main()
{
        char c;
        FILE *file;
        file = fopen("input.txt","r");
        if(file==NULL)
        {
                printf("Error opening file");
                return 1;
        }
        while((c=getc(file)) != EOF)
        {
                if(c==' ' || c==',' || c=='.' || c==';' || c=='\n' || c=='\t' ||
c=='(' || c==')' || c=='{' || c=='}' || c=='[' || c==']')
                        printf("%c is a delimiter\n",c);
                else if(c=='+' || c=='-' || c=='*' || c=='/' || c=='%' || c=='>' ||
c=='<' || c=='=' || c=='!' || c=='&' || c=='|' || c=='^' || c=='~' || c=='?')
                        printf("%c is an operator\n",c);
        }
        fclose(file);
        return 0;
}
```

Assignment 2:

1. Write a C program to identify and/or count the occurrences of all two character long operators

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    FILE *file = fopen("code.txt", "r");
    if (!file) {
        printf("Error: could not open file\n");
        return 1;
    }

    const char *operators[] = {"++", "--", "==", "!=", ">=", "<=", "&&", "||"};
    int counts[8] = {0};
    int i;
    char buffer[3];
    while (fread(buffer, sizeof(char), 2, file) == 2) {
        buffer[2] = '\0';
        for (i = 0; i < 8; i++) {
            if (strcmp(buffer, operators[i]) == 0) {
                counts[i]++;
            }
        }
        fseek(file, -1, SEEK_CUR);
    }

    for (i = 0; i < 8; i++)
        {
                if(counts[i]>0)
                printf("%s: %d\n", operators[i], counts[i]);
    }
    fclose(file);
    return 0;
}
```

Assignment 3:

1. Write a Lex Program to count the number of words in a string.

Code:

```
/*Lex Program to count the number of words*/
/*Declarations*/
%{
        #include <stdio.h>
        #include <string.h>
        int i = 0;
%}
/*Translation Rules*/
%%
[a-zA-Z]+   { i++; }
"\n"        {printf("The number of Words in the string are: %d\n",i); i = 0;}
%%
/*Auxiliary Functions*/
int yywrap(void){}
```

```
int main()
{
        printf("Enter the String\n");
        yylex();
        return 0;
}
```

2. Write a lex program to identify single and multiline comments using regular expressions.

```
/*Declarations*/
%{
        #include <stdio.h>
%}
/*Translation Rules*/
%%
"//".*                  {printf("Contains single Line Comment");}
"/*".*"\n".*"*/"  {printf("Contains multiline Comment");}
%%
/*Auxiliary Functions*/
int yywrap(void){}
int main()
{
        yyin = fopen("code.txt","r");
        if (!yyin)
        {
        perror("Error opening file");
        return 1;
        }
        yylex();
        fclose(yyin);
        return 0;
}
```

3. Write a lex program to count printf and scanf lines in a program and replace it with writef and readf respectively.

Code:

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

int printf_count = 0;
int scanf_count = 0;
%}

%%
"printf"    { printf_count++; printf("writef"); }
"scanf"     { scanf_count++; printf("readf"); }
.           { printf("%s", yytext); }
%%
int yywrap(void){}
int main() {
    yyin = fopen("code.txt", "r");
    yylex();
```

```
    fclose(yyin);

    printf("\nNumber of printf statements: %d\n", printf_count);
    printf("Number of scanf statements: %d\n", scanf_count);

    return 0;
}
```

Assignment 4:

1. Write a C program to identify keywords and identifiers in a program.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_IDENTIFIER_LENGTH 32
#define NUM_KEYWORDS 32

const char *keywords[] = {"auto", "break", "case", "char", "const", "continue",
                          "default", "do", "double", "else", "enum", "extern",
                          "float", "for", "goto", "if", "inline", "int",
                          "long", "register", "restrict", "return", "short",
                          "signed", "sizeof", "static", "struct", "switch",
                          "typedef", "union", "unsigned", "void", "volatile",
                          "while"};

int is_keyword(const char *word) {
    int i;
    for (i = 0; i < NUM_KEYWORDS; i++) {
        if (strcmp(word, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

int is_valid_identifier_char(char c) {
    return isalnum(c) || c == '_';
}

int main() {
    char identifier[MAX_IDENTIFIER_LENGTH];
    int identifier_len = 0;

    FILE *fp = fopen("code.txt", "r");
    if (!fp) {
        perror("Error opening file");
        return 1;
    }
```

```
    int c;
    while ((c = fgetc(fp)) != EOF) {
        if (is_valid_identifier_char(c)) {
            if (identifier_len < MAX_IDENTIFIER_LENGTH - 1) {
                identifier[identifier_len++] = c;
            }
        } else {
            if (identifier_len > 0) {
                identifier[identifier_len] = '\0';
                if (is_keyword(identifier)) {
                    printf("%s is a keyword\n", identifier);
                } else {
                    printf("%s is an identifier\n", identifier);
                }
                identifier_len = 0;
            }
        }
    }

    fclose(fp);
    return 0;
}
```

2. Write a program in C to identify keywords and identifiers in an input file with each instance only being once.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_IDENTIFIER_LENGTH 32
#define NUM_KEYWORDS 32

const char *keywords[] = {"auto", "break", "case", "char", "const", "continue",
                          "default", "do", "double", "else", "enum", "extern",
                          "float", "for", "goto", "if", "inline", "int",
                          "long", "register", "restrict", "return", "short",
                          "signed", "sizeof", "static", "struct", "switch",
                          "typedef", "union", "unsigned", "void", "volatile",
                          "while"};

int is_keyword(const char *word) {
    int i;
    for (i = 0; i < NUM_KEYWORDS; i++) {
        if (strcmp(word, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}
```

```
int is_valid_identifier_char(char c) {
    return isalnum(c) || c == '_';
}

int main(int argc, char **argv) {
    char identifier[MAX_IDENTIFIER_LENGTH];
    int identifier_len = 0;

    char *seen_identifiers[MAX_IDENTIFIER_LENGTH];
    int num_seen_identifiers = 0;

    FILE *fp = fopen("code.txt", "r");
    if (!fp) {
        perror("Error opening file");
        return 1;
    }

    int c;
    while ((c = fgetc(fp)) != EOF) {
        if (is_valid_identifier_char(c)) {
            if (identifier_len < MAX_IDENTIFIER_LENGTH - 1) {
                identifier[identifier_len++] = c;
            }
        } else {
            if (identifier_len > 0) {
                identifier[identifier_len] = '\0';
                if (is_keyword(identifier)) {
                    printf("%s is a keyword\n", identifier);
                } else {
                    int i;
                    for (i = 0; i < num_seen_identifiers; i++) {
                        if (strcmp(identifier, seen_identifiers[i]) == 0) {
                            break;
                        }
                    }
                    if (i == num_seen_identifiers) {
                        seen_identifiers[num_seen_identifiers++] =
strdup(identifier);
                        printf("%s is an identifier\n", identifier);
                    }
                }
                identifier_len = 0;
            }
        }
    }

    fclose(fp);
    return 0;
}
```

Assignment 5:
1. Write a C program to implement a sample symbol table/terminal table when an arithmatical expression having identifiers of single character are fed as input.
2. Write a C program to implement a sample symbol table/terminal table when an arithmatical expression having identifiers of single character are fed as input and show moemory location
Assignment 6:
1. Write a lex program to recognize a vaild arithmetic expression.
Code:

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

%%
[0-9]+                  printf("Operand: %s\n", yytext);
[ \t\n]                 /* ignore whitespace */
[()+*/%-]               printf("Operator: %c\n", yytext[0]);
.                       printf("Invalid character: %c\n", yytext[0]);
%%
int yywrap(void) {}
int main() {
        printf("Enter the expression\n");
    yylex();
    return 0;
}
```

2. Write a lex program to recognize a valid variable which starts with a letter followed by any number of letters or digits

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

%%
[a-zA-Z][_]*[a-zA-Z0-9]*     printf("Valid variable: %s\n", yytext);
.                       printf("Invalid character: %c\n", yytext[0]);
%%
int yywrap(void) {}
int main() {
        printf("Enter the variable name\n");
    yylex();
    return 0;
}
```

Assignment 7:
1. Write a lex program to implement type checking:
Code:

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

%%
int                     printf("Variable Type Used: int\n");
```

```
float                   printf("Variable Type Used: float\n");
double                  printf("Variable Type Used: double\n");
char                    printf("Variable Type Used: char\n");
void                    printf("Variable Type Used: void\n");
[0-9]+                  ;
[ \t\n]                 /* ignore whitespace */
[()+*/%-]               ;
[a-zA-Z_][a-zA-Z0-9_]*  ;
.                       ;
%%
int yywrap(void) {}
int main() {
    yyin = fopen("code.txt", "r");
    yylex();
    fclose(yyin);
    return 0;
}
```

Assignment 8:

1. Write a C program to implement Control Flow Analysis & Data Flow Analysis: