program for lexical analyser

```
#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);
}
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);
}
```

strings under a, a*b+,abb

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
```

```
{
char s[20],c;
int state=0,i=0;
clrscr();
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); getch();
}
```

is a comment

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

// Function to check if a line is a comment
int isComment(char *line) {
   int i = 0;
   int len = strlen(line);
   int isWhiteSpace = 1;

   // Skip leading white spaces
   while (i < len && (line[i] == ' ' || line[i] == '\t')) {
      i++;
   }

   // Check if line starts with '//'
   if (line[i] == '/' && line[i + 1] == '/') {
      return 1; // Line is a single-line comment
   }

   // Check if line starts with '/*'
   if (line[i] == '/' && line[i + 1] == '*') {
      return 2; // Line is a multi-line comment
```

```
    }

    // Check if line ends with '*/'
    if (i > 0 && line[i - 1] == '*' && line[i] == '/') {
        return 3; // Line ends a multi-line comment
    }

    // Check if line is empty or only white spaces
    while (i < len) {
        if (line[i] != ' ' && line[i] != '\t') {
            isWhiteSpace = 0;
            break;
        }
        i++;
    }

    if (isWhiteSpace) {
        return 4; // Line is empty or contains only white spaces
    }

    // If none of the above conditions met, line is not a comment
    return 0;
}

int main() {
    char line[1000];

    printf("Enter your C code (press Ctrl + D to end input):\n");

    while (fgets(line, sizeof(line), stdin) != NULL) {
        int commentType = isComment(line);
```

```
        if (commentType == 1) {
            printf("Single-line comment: %s", line);
        } else if (commentType == 2) {
            printf("Start of multi-line comment: %s", line);
        } else if (commentType == 3) {
            printf("End of multi-line comment: %s", line);
        } else if (commentType == 4) {
            printf("Empty line: %s", line);
        } else {
            printf("Not a comment: %s", line);
        }
    }

    return 0;
}
```

is a comment1

```
#include<stdio.h>
#include<conio.h>
void main()
{
char com[30];
int i=2,a=0;
clrscr();
printf("\n Enter comment:");
gets(com);
if(com[0]=='/')
{
if(com[1]=='/')
printf("\n It is a comment");
```

```
else if(com[1]=='*')
{
for(i=2;i<=30;i++)
{
if(com[i]=='*'&&com[i+1]=='/')
{
printf("\n It is a comment");
a=1;
break;
}
else
continue;
}
if(a==0)
printf("\n It is not a comment");
}
else
printf("\n It is not a comment");
}
else printf("\n It is not a comment");
getch();
}
```

whether a valid identifier

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
char a[10];
int flag, i=1;
```

```
clrscr();
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();
}
```

validating operators

```
#include<stdio.h>
#include<conio.h>
void main()
{
char s[5];
clrscr();
printf("\n Enter any operator:");
gets(s);
switch(s[0])
```

```
{
case'>': if(s[1]=='=')
printf("\n Greater than or equal");
else
printf("\n Greater than");
break;
case'<': if(s[1]=='=')
printf("\n Less than or equal");
else
printf("\nLess than");
break;
case'=': if(s[1]=='=')
printf("\nEqual to");
else
printf("\nAssignment");
break;
case'!': if(s[1]=='=')
printf("\nNot Equal");
else
printf("\n Bit Not");
break;
case'&': if(s[1]=='&')
printf("\nLogical AND");
else
printf("\n Bitwise AND");
break;
case'|': if(s[1]=='|')
printf("\nLogical OR");
else
printf("\nBitwise OR");
break;
case'+': printf("\n Addition");
```

```
break;
case'-': printf("\nSubstraction");
break;
case'*': printf("\nMultiplication");
break;
case'/': printf("\nDivision");
break;
case'%': printf("Modulus");
break;
default: printf("\n Not a operator");
}
getch();
}
```

validating operators1

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

// Function to check if a string is a valid operator
int isValidOperator(char *str) {
   // List of valid operators
   char operators[] = {"+", "-", "", "/", "%", "=", "==", "<", ">", "<=", ">="};

   // Check if the given string is in the list of valid operators
   for (int i = 0; i < sizeof(operators) / sizeof(operators[0]); i++) {
      if (strcmp(str, operators[i]) == 0) {
         return 1; // Valid operator
      }
   }
```

```
    return 0; // Not a valid operator
}

// Function to tokenize and validate operators in an input string
void validateOperators(char *input) {
    char token[20]; // Assuming maximum length of an operator is 20 characters

    for (int i = 0; input[i] != '\0'; i++) {
        if (isalnum(input[i]) || input[i] == '_') {
            // If the character is alphanumeric or underscore, add it to the current token
            strncat(token, &input[i], 1);
        } else {
            // Check if the token is a valid operator
            if (strlen(token) > 0 && isValidOperator(token)) {
                printf("Valid operator: %s\n", token);
            }

            // Reset the token for the next iteration
            token[0] = '\0';
        }
    }
}

int main() {
    char input[100];

    // Get input from the user
    printf("Enter a string: ");
    fgets(input, sizeof(input), stdin);

    // Remove trailing newline character
    input[strcspn(input, "\n")] = '\0';
```

```
    // Call the function to validate operators
    validateOperators(input);

    return 0;
}
```

lex program

```
// lexer.l
%{
#include <stdio.h>
%}

DIGIT [0-9]
LETTER [a-zA-Z]
ID ({LETTER}({LETTER}|{DIGIT})*)
INT {DIGIT}+
WS [ \t\n]

%%

{INT}      { printf("Integer: %s\n", yytext); }
{ID}       { printf("Identifier: %s\n", yytext); }
[-+*/=]    { printf("Operator: %s\n", yytext); }
{WS}       ;  // Ignore whitespace

.          { printf("Invalid character: %s\n", yytext); }

%%

int main(int argc, char* argv[]) {
```

```
    if (argc != 2) {
        printf("Usage: %s <input_file>\n", argv[0]);
        return 1;
    }

    FILE* input = fopen(argv[1], "r");
    if (!input) {
        perror("Error opening file");
        return 1;
    }

    yyin = input;
    yylex();

    fclose(input);

    return 0;
}
```