Complete compiler design tasks

Solve the complete assignment

Compiler Design Assignment
Deadline: 6th December 2025

1. Write a C program to read a source code file and count the number of keywords, identifiers, operators, numbers, and special symbols.
Hint: Use isalpha(), isdigit(), and string comparison with keyword list.
Keyword list: printf, scanf, int, float, char, for, while, if, else, break
2. Write a C program that takes a C source file as input and removes all comments (// and /*...*/) and extra whitespaces.
3. Write a C program to extract and display all lexemes (tokens) from a given C source code.
For example,
Input: int a = b + 10;
Output: int, a, =, b, +, 10, ;
4. Write a C program that reads a line of code and checks whether each word is a keyword, identifier, or constant.
Hint: Use an array of keywords like "int", "float", "return", etc.
5. Write a C program that identifies and categorizes operators into:
 Arithmetic (+, -, *, /, %)
 Relational (==, !=, <, >, <=, >=)
 Logical (&&, ||, !)
6. Write a C program that recognizes integer, floating-point, and character constants in a given line of code.
Example:
Input: int x = 45; float y = 4.5; char c = 'a';
Output:
Integer: 45, Float: 4.5, Character: 'a'
7. Write a C program to find and display string literals in a given C program source file.
Example:
Input: printf("Hello World");
Output: "Hello World"
8. Write a C program that reads a source code file and displays:
 Total number of lines
 Total number of tokens
 Total number of comments
9. Write a C program that checks whether a given string is a valid identifier in C.
Rules:

Compiler Design Assignment
Deadline: 6th December 2025

 Must start with a letter or underscore _.

□ Can contain letters, digits, or underscores.
□ Cannot be a keyword.
Example:
Input: _count1 → Valid
Input: 2value → Invalid
10. Write a C program that scans a source file and detects lexical errors, such as:
□ Invalid identifiers (1abc)
□ Unclosed string literals ("Hello)
□ Unfinished comments (/* something)
Display error messages with line numbers.
11. Write a C program that extracts all preprocessor directives (like #include, #define, #ifdef) from a given C source file and lists them.
Input:
#include <stdio.h>
#define PI 3.14
int main() { return 0; }
Output:
#include <stdio.h>
#define PI 3.14
12. Write a C program that scans a C source file and counts how many times each keyword (e.g., int, return, if, while, etc.) appears.
Example Output:
int : 3
return : 1
if : 2
while : 1
13. Write a C program to extract and print all multi-line comments (/* ... */) from a given C source file.
If no multi-line comments are present, print "No multi-line comments found."
14. Write a C program to analyze the frequency of each character (letters, digits, symbols) in a source code file.
This helps understand how often certain symbols (like {, }, ;, (,)) appear in source text.
'{' : 4
'}' : 4
'(' : 5
')' : 5
';' : 10

Nice — here are complete, ready-to-compile C programs for **all 14 assignment problems**. Each program is self-contained, small, and uses only standard C (C89/C99 compatible). I included a short one-line description before each code block so you know what to compile and run.

If you want, I can package them into separate files or a single makefile next. Now the code:

---

1) Count keywords, identifiers, operators, numbers and special symbols (file input)

Save as `count_tokens.c`. Run: `./a.out source.c`

```c
/* count_tokens.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

const char *keywords[] =
{"printf","scanf","int","float","char","for","while","if","else","break"};
int is_keyword(const char *s){
    for(int i=0;i<10;i++) if(strcmp(s, keywords[i])==0) return 1;
    return 0;
}

int main(int argc, char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r");
    if(!f){ perror("fopen"); return 1;}
    int c;
    int in_token=0;
    char token[256]; int ti=0;
    long keywords_count=0, id_count=0, num_count=0, op_count=0, sp_count=0;
    const char *ops = "+-*/%=<>!&|^~";
    while((c=fgetc(f))!=EOF){
        if(isalpha(c) || c=='_'){
            token[ti++]=c; in_token=1;
        } else if(isdigit(c)){
            if(in_token){ token[ti++]=c; }
            else { // number token
                // collect full number
                ungetc(c,f);
                int d; int seen_dot=0;
                char num[128]; int ni=0;
                while((d=fgetc(f))!=EOF && (isdigit(d) || d=='.')){
                    if(d=='.') seen_dot=1;
                    num[ni++]=d;
                }
                num[ni]=0;
                if(d!=EOF) ungetc(d,f);
                num_count++;
            }
        } else {
            if(in_token){
                token[ti]=0;
                if(is_keyword(token)) keywords_count++;
                else id_count++;
                ti=0; in_token=0;
            }
            if(strchr(ops,c)) op_count++;
            else if(!isspace(c)) sp_count++;
        }
    }
    if(in_token){
        token[ti]=0;
        if(is_keyword(token)) keywords_count++;
        else id_count++;
    }
    fclose(f);
    printf("Keywords: %ld\nIdentifiers: %ld\nNumbers: %ld\nOperators: %ld\nSpecial symbols:
%ld\n",
           keywords_count, id_count, num_count, op_count, sp_count);
    return 0;
}
```

2) Remove all comments and extra whitespaces

Save as **`remove_comments_ws.c`**. Run: **`./a.out input.c > cleaned.c`** c

```c
/* remove_comments_ws.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c;
    int state=0; /* 0 normal, 1 slash seen, 2 line comment, 3 star comment, 4 in string */
    int prev_space=0;
    while((c=fgetc(f))!=EOF){
        if(state==0){
            if(c=='/'){ state=1; }
            else if(c=='"'){ putchar(c); state=4; prev_space=0; }
            else if(isspace(c)){
                if(!prev_space){ putchar(' '); prev_space=1; }
            } else { putchar(c); prev_space=0; }
        } else if(state==1){
            if(c=='/'){ state=2; } // line comment start
            else if(c=='*'){ state=3; } // block comment start
            else { putchar('/'); putchar(c); state=0; prev_space=0; }
        } else if(state==2){
            if(c=='\n'){ putchar('\n'); state=0; prev_space=0; }
        } else if(state==3){
            if(c=='*'){
                int d=fgetc(f);
                if(d=='/') state=0;
                else if(d==EOF) break;
                else ungetc(d,f);
            }
        } else if(state==4){ // inside string
            putchar(c);
            if(c=='\\'){ int d=fgetc(f); if(d!=EOF){ putchar(d); } else break;}
            else if(c=='"'){ state=0; prev_space=0; }
        }
    }
    fclose(f);
    return 0;
}
```

3) Extract and display all lexemes (tokens)

Save as **`lexemes.c`**. Run: **`./a.out source.c`**

c

```c
/* lexemes.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c; char tok[256]; int ti=0; int first=1;
    while((c=fgetc(f))!=EOF){
        if(isalpha(c) || c=='_'){
            tok[ti++]=c;
            while((c=fgetc(f))!=EOF && (isalnum(c) || c=='_')) tok[ti++]=c;
            if(c!=EOF) ungetc(c,f);
            tok[ti]=0; if(!first) printf(", "); printf("%s",tok); first=0; ti=0;
        } else if(isdigit(c)){
            tok[ti++]=c;
            while((c=fgetc(f))!=EOF && (isdigit(c) || c=='.')) tok[ti++]=c;
            if(c!=EOF) ungetc(c,f);
            tok[ti]=0; if(!first) printf(", "); printf("%s",tok); first=0; ti=0;
        } else if(isspace(c)) continue;
        else { // operators and punctuation, consider multi-char
            int d=fgetc(f);
            if(d!=EOF){
                if((c=='=' && d=='=') || (c=='!' && d=='=') || (c=='<' && d=='=') || (c=='>' &&
d=='=') || (c=='&' && d=='&') || (c=='|' && d=='|')){
                    char op[3]={c,d,0}; if(!first) printf(", "); printf("%s",op); first=0;
                } else { ungetc(d,f); char s[2]={c,0}; if(!first) printf(", "); printf("%s",s);
first=0; }
            } else { char s[2]={c,0}; if(!first) printf(", "); printf("%s",s); first=0; }
        }
    }
    printf("\n");
    fclose(f);
    return 0;
}
```

4) Check whether each word is keyword, identifier, or constant (single line input)

Save as **`classify_words.c`**. Run and input a line, or pipe: **`echo "int a = 5;" | ./a.out`**

```c
/* classify_words.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

const char *keywords[] =
{"int","float","return","char","for","while","if","else","break","scanf","printf"};
int is_keyword(const char *s){
    for(int i=0;i<11;i++) if(strcmp(s,keywords[i])==0) return 1;
    return 0;
}
int is_number(const char *s){
    int i=0,dots=0;
    if(s[0]=='\0') return 0;
    if(s[0]=='+'||s[0]=='-') i++;
    for(;s[i];i++){
        if(s[i]=='.') dots++;
        else if(!isdigit((unsigned char)s[i])) return 0;
    }
    return dots<=1;
}
int is_identifier(const char *s){
    if(!(isalpha((unsigned char)s[0]) || s[0]=='_')) return 0;
    for(int i=1;s[i];i++) if(!(isalnum((unsigned char)s[i])|| s[i]=='_')) return 0;
    if(is_keyword(s)) return 0;
    return 1;
}

int main(void){
    char line[1024];
    if(!fgets(line,sizeof(line),stdin)){ return 0; }
    char tok[256]; int ti=0;
    for(int i=0; ; i++){
        char ch=line[i];
        if(isalnum((unsigned char)ch) || ch=='_' || ch=='.' || ch=='+' || ch=='-'){
            tok[ti++]=ch;
        } else {
            if(ti){
                tok[ti]=0;
                if(is_keyword(tok)) printf("%s : keyword\n", tok);
                else if(is_number(tok)) printf("%s : constant\n", tok);
                else if(is_identifier(tok)) printf("%s : identifier\n", tok);
                else printf("%s : unknown\n", tok);
                ti=0;
            }
            if(ch==0) break;
        }
    }
    return 0;
}
```

5) Identify and categorize operators (input line)

Save as **`categorize_operators.c`**. Run: **`echo "a==b && c+d" | ./a.out`**

```c
/* categorize_operators.c */
#include <stdio.h>
#include <string.h>

int main(void){
    char line[1024];
    if(!fgets(line,sizeof(line),stdin)) return 0;
    for(int i=0; line[i]; ){
        if(line[i]==' '||line[i]=='\t'){ i++; continue; }
        if(line[i]=='+'||line[i]=='-'||line[i]=='*'||line[i]=='/'||line[i]=='%'){
            printf("Arithmetic: %c\n", line[i]); i++;
        } else if((line[i]=='=' && line[i+1]=='=') || (line[i]=='!'&&line[i+1]=='=') ||
                  (line[i]=='<'&&line[i+1]=='=') || (line[i]=='>'&&line[i+1]=='=')){
            printf("Relational: %c%c\n", line[i], line[i+1]); i+=2;
        } else if(line[i]=='<'||line[i]=='>'){
            printf("Relational: %c\n", line[i]); i++;
        } else if(line[i]=='&' && line[i+1]=='&'){
            printf("Logical: &&\n"); i+=2;
        } else if(line[i]=='|' && line[i+1]=='|'){
            printf("Logical: ||\n"); i+=2;
        } else if(line[i]=='!'){
            printf("Logical: !\n"); i++;
        } else i++;
    }
    return 0;
}
```

6) Recognize integer, float and character constants in a line

Save as **`recognize_constants.c`**. Run: **`echo "int x = 45; float y = 4.5; char c = 'a';" | ./a.out`**

```c
/* recognize_constants.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int main(void){
    char s[1024];
    if(!fgets(s,sizeof(s),stdin)) return 0;
    int i=0;
    while(s[i]){
        if(s[i]=='\''){ // char
            int j=i;
            j++;
            while(s[j] && s[j] != '\'') {
                if(s[j]=='\\' && s[j+1]) j+=2;
                else j++;
            }
            if(s[j]=='\''){
                printf("Character: ");
                for(int k=i;k<=j;k++) putchar(s[k]);
                printf("\n");
                i=j+1;
            } else { i++; }
        } else if(isdigit((unsigned char)s[i])){
            int j=i; int dots=0;
            while(s[j] && (isdigit((unsigned char)s[j])|| s[j]=='.')){ if(s[j]=='.') dots++;
j++; }
            if(dots) printf("Float: ");
            else printf("Integer: ");
            for(int k=i;k<j;k++) putchar(s[k]);
            printf("\n");
            i=j;
        } else i++;
    }
    return 0;
}
```

7) Find and display string literals from a source file

Save as **`string_literals.c`**. Run: **`./a.out source.c`**

```c
/* string_literals.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c; int in_string=0;
    char buf[1024]; int bi=0;
    while((c=fgetc(f))!=EOF){
        if(!in_string){
            if(c=='"'){ in_string=1; buf[bi++]=c; }
        } else {
            buf[bi++]=c;
            if(c=='\\'){ int d=fgetc(f); if(d==EOF) break; buf[bi++]=d; }
            else if(c=='"'){
                buf[bi]=0; printf("%s\n", buf); bi=0; in_string=0;
            }
            if(bi>=1020){ buf[bi]=0; printf("%s\n",buf); bi=0; }
        }
    }
    fclose(f);
    return 0;
}
```

8) Display total number of lines, total number of tokens, total number of comments

Save as **`summary_counts.c`**. Run: **`./a.out source.c`**

```c
/* summary_counts.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

```
int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c, prev=0;
    long lines=0, tokens=0, comments=0;
    int in_word=0;
    int state=0; // 0 normal,1 slash,2 line comment,3 block comment,4 string
    while((c=fgetc(f))!=EOF){
        if(c=='\n') lines++;
        if(state==0){
            if(c=='/') state=1;
            else if(c=='"') state=4;
            if(!isspace(c) && (isalnum(c) || c=='_' || strchr("+-*/%=<>&|!;:,(){}[]",c))){
                if(!in_word && (isalnum(c)||c=='_')) { tokens++; in_word=1; }
                else if(!isalnum(c) && in_word) in_word=0;
                else if(strchr("+-*/%=<>&|!;:,(){}[]",c)) { tokens++; in_word=0; }
            } else if(isspace(c)) in_word=0;
        } else if(state==1){
            if(c=='/') { comments++; state=2; }
            else if(c=='*'){ comments++; state=3; }
            else { state=0; if(!isspace(c)) { if(isalnum(c) || c=='_'){ tokens++; in_word=1;} }
}
        } else if(state==2){
            if(c=='\n') state=0;
        } else if(state==3){
            if(c=='*'){
                int d=fgetc(f);
                if(d=='/') state=0;
                else if(d==EOF) break;
                else ungetc(d,f);
            }
        } else if(state==4){
            if(c=='\\'){ int d=fgetc(f); if(d==EOF) break; }
            else if(c=='"') state=0;
        }
    }
    if(lines==0) lines=1;
    fclose(f);
    printf("Total lines: %ld\nTotal tokens (approx): %ld\nTotal comments (approx): %ld\n",
lines, tokens, comments);
    return 0;
}
```

9) Check whether a given string is a valid C identifier (and not a keyword)

Save as **`valid_identifier.c`**. Run: **`./a.out _count1`**

```c
/* valid_identifier.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

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","s
truct","switch","typedef","union","unsigned","void","volatile","while","scanf","printf","break"}
;
int is_keyword(const char *s){
    for(size_t i=0;i<sizeof(keywords)/sizeof(keywords[0]);i++) if(strcmp(s,keywords[i])==0)
return 1;
    return 0;
}

int valid_identifier(const char *s){
    if(!s || !s[0]) return 0;
    if(!(isalpha((unsigned char)s[0]) || s[0]=='_')) return 0;
    for(int i=1;s[i];i++) if(!(isalnum((unsigned char)s[i]) || s[i]=='_')) return 0;
    if(is_keyword(s)) return 0;
    return 1;
}

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s identifier\n", argv[0]); return 1;}
    if(valid_identifier(argv[1])) printf("%s → Valid\n", argv[1]);
    else printf("%s → Invalid\n", argv[1]);
    return 0;
}
```

---

10) Scan a source file and detect lexical errors: invalid identifiers, unclosed strings, unfinished comments (with line numbers)

Save as **`lexical_errors.c`**. Run: **`./a.out source.c`**

```c
c


/* lexical_errors.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

const char *keywords[]=
{"int","float","char","return","if","else","for","while","break","printf","scanf"};

int is_keyword(const char *s){
    for(int i=0;i<11;i++) if(strcmp(s,keywords[i])==0) return 1;
    return 0;
}

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c; int line=1;
    int state=0; /*0 normal,1 in string,2 in char,3 slash seen,4 block comment*/
    int token_mode=0; char tk[256]; int ti=0;
    while((c=fgetc(f))!=EOF){
        if(c=='\n') line++;
        if(state==0){
            if(isalpha(c) || c=='_'){
                tk[0]=c; ti=1;
                while((c=fgetc(f))!=EOF && (isalnum(c)|| c=='_')) tk[ti++]=c;
                if(c!=EOF) ungetc(c,f);
                tk[ti]=0;
                // check if starts with digit handled later - here fine
                if(isdigit((unsigned char)tk[0])){
                    printf("Line %d: Invalid identifier (starts with digit): %s\n", line, tk);
                }
                if(is_keyword(tk)) { /*ok*/ }
                // nothing else
            } else if(c=='/'){ int d=fgetc(f); if(d=='*'){ state=4; } else if(d=='/'){ /*line
comment consume*/ while((d=fgetc(f))!=EOF && d!='\n'); if(d=='\n') line++; } else ungetc(d,f); }
            else if(c=='"'){ state=1; }
            else if(c=='\''){ state=2; }
        } else if(state==1){ // in string
            if(c==EOF){ printf("Line %d: Unclosed string literal\n", line); break; }
            if(c=='\\'){ int d=fgetc(f); if(d==EOF){ printf("Line %d: Unclosed string
literal\n", line); break; } }
            else if(c=='"'){ state=0; }
            else if(c=='\n'){ /* string across lines allowed in some C via \ at EOL, but raw
newline inside is error */ printf("Line %d: Possibly unclosed string literal (newline
inside)\n", line); line++; state=0; }
        } else if(state==2){ // char literal
            if(c==EOF){ printf("Line %d: Unclosed character literal\n", line); break;}
            if(c=='\\'){ int d=fgetc(f); if(d==EOF){ printf("Line %d: Unclosed character
literal\n", line); break; }}
            else if(c=='\''){ state=0; }
            else if(c=='\n'){ printf("Line %d: Possibly unclosed character literal (newline
inside)\n", line); line++; state=0;}
        } else if(state==4){ // block comment
            int prev=0;
            while((c=fgetc(f))!=EOF){
                if(c=='\n') line++;
                if(prev=='*' && c=='/'){ state=0; break; }
                prev=c;
            }
            if(c==EOF){ printf("Line %d: Unfinished comment (/* ... not closed)\n", line);
break; }
        }
    }
    fclose(f);
    return 0;
}
```

11) Extract all preprocessor directives (#include, #define, #ifdef, ...)

Save as **`preprocessor_extract.c`**. Run: **`./a.out source.c`**

```c
c


/* preprocessor_extract.c */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    char line[1024];
    while(fgets(line,sizeof(line),f)){
        int i=0;
        while(isspace((unsigned char)line[i])) i++;
        if(line[i]=='#') printf("%s", line);
    }
    fclose(f);
    return 0;
}
```

12) Count how many times each keyword appears in a source file

Save as **`keyword_count.c`**. Run: **`./a.out source.c`**

```c
c


/* keyword_count.c */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

const char *keywords[] =
{"int","return","if","while","for","else","break","char","float","double","scanf","printf"};
int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    int counts[sizeof(keywords)/sizeof(keywords[0])];
    memset(counts,0,sizeof(counts));
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c;
    char tok[256]; int ti=0;
    while((c=fgetc(f))!=EOF){
        if(isalpha(c) || c=='_'){
            tok[ti++]=c;
            while((c=fgetc(f))!=EOF && (isalnum(c)||c=='_')) tok[ti++]=c;
            if(c!=EOF) ungetc(c,f);
            tok[ti]=0; ti=0;
            for(size_t i=0;i<sizeof(keywords)/sizeof(keywords[0]);i++){
                if(strcmp(tok, keywords[i])==0) counts[i]++;
            }
        }
    }
    for(size_t i=0;i<sizeof(keywords)/sizeof(keywords[0]);i++){
        printf("%s : %d\n", keywords[i], counts[i]);
    }
    fclose(f);
    return 0;
}
```

13) Extract and print all multi-line comments (/* ... */)

Save as **`extract_multiline_comments.c`**. Run: **`./a.out source.c`**

```c
/* extract_multiline_comments.c */
#include <stdio.h>
#include <string.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    int c, d;
    int found=0;
    while((c=fgetc(f))!=EOF){
        if(c=='/'){
            d=fgetc(f);
            if(d=='*'){
                found=1;
                putchar('/'); putchar('*');
                while((c=fgetc(f))!=EOF){
                    putchar(c);
                    if(c=='*'){
                        int e=fgetc(f);
                        if(e=='/'){ putchar(e); break; }
                        if(e==EOF) break;
                        ungetc(e,f);
                    }
                }
                putchar('\n');
            } else if(d!=EOF) ungetc(d,f);
        }
    }
    if(!found) printf("No multi-line comments found.\n");
    fclose(f);
    return 0;
}
```

14) Analyze frequency of each character in a source file

Save as **`char_frequency.c`**. Run: **`./a.out source.c`**

```c
/* char_frequency.c */
#include <stdio.h>
#include <ctype.h>

int main(int argc,char **argv){
    if(argc!=2){ printf("Usage: %s source.c\n", argv[0]); return 1;}
    FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;}
    long freq[256] = {0};
    int c;
    while((c=fgetc(f))!=EOF) freq[(unsigned char)c]++;
    // Print letters, digits and some common symbols
    for(int ch='a'; ch<='z'; ch++){
        if(freq[ch]) printf("'%c' : %ld\n", ch, freq[ch]);
    }
    for(int ch='A'; ch<='Z'; ch++){
        if(freq[ch]) printf("'%c' : %ld\n", ch, freq[ch]);
    }
    for(int ch='0'; ch<='9'; ch++){
        if(freq[ch]) printf("'%c' : %ld\n", ch, freq[ch]);
    }
    const char symbols[] = "{}();,[]<>+-*/=%&|!^~#:.'\"\\\n\t ";
    for(size_t i=0;i<sizeof(symbols)-1;i++){
        unsigned char ch = symbols[i];
        if(freq[ch]) {
            if(ch=='\n') printf("'\\n' : %ld\n", freq[ch]);
            else if(ch=='\t') printf("'\\t' : %ld\n", freq[ch]);
            else if(ch==' ') printf("' ' : %ld\n", freq[ch]);
            else printf("'%c' : %ld\n", ch, freq[ch]);
        }
    }
    fclose(f);
    return 0;
}
```

Notes & tips

- All programs are intentionally simple lexical-style scanners (they are not full C compilers). They work well for typical student source code and the assignment requirements.
- For the comment/string-aware tasks I implemented basic escape handling (e.g., \" or \\) but these tools are not fully robust to pathological C corner cases — they are suitable for an assignment-level solution.
- Compile with **`gcc program.c -o program`** (or **`clang`**), e.g. **`gcc count_tokens.c -o count_tokens`**.
- If you want them organized into files with a Makefile, or combined into a single interactive menu program, tell me and I'll prepare that next.

Would you like me to:

1. Put these into a zip or create all files in one paste ready to copy?
2. Create a Makefile to compile all programs?
3. Combine some related tasks into a single program with command-line options?

Pick one and I'll produce it immediately.

in Java

Got it — I rewrote the **14 assignment programs in Java**. Each program is a self-contained **`.java`** file with a **`main`** method and a usage note. These are practical, assignment-level lexical utilities (not full compilers); they handle typical student C source code and common edge cases (strings, comments, escapes) but are not guaranteed to handle every pathological C corner case.

Paste each file into its own **`.java`** file (filename matches the public class name), compile with **`javac`**, run with **`java`**. If you want, I can instead pack everything into a single multi-command program or create a Maven project — say the option number from the previous message and I'll produce that.

---

1) `CountTokens.java` — count keywords, identifiers, operators, numbers, special symbols (file input)

Usage: **`java CountTokens source.c`**

```java
// CountTokens.java
import java.io.*;
import java.util.*;

public class CountTokens {
    static final Set<String> KEYWORDS = new HashSet<>(Arrays.asList(
        "printf","scanf","int","float","char","for","while","if","else","break"
    ));

    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java CountTokens source.c"); return; }
        FileInputStream fis = new FileInputStream(args[0]);
        Reader r = new InputStreamReader(fis);
        int ch;
        StringBuilder token = new StringBuilder();
        long kw=0, id=0, num=0, op=0, special=0;
        String ops = "+-*/%=<>!&|^~";
        while ((ch = r.read()) != -1) {
            char c = (char) ch;
            if (Character.isLetter(c) || c=='_') {
                token.append(c);
                while ((ch = r.read())!=-1) {
                    c=(char)ch;
                    if (Character.isLetterOrDigit(c) || c=='_') token.append(c);
                    else { r.reset(); break; }
                    r.mark(1);
                }
                String s = token.toString();
                token.setLength(0);
                if (KEYWORDS.contains(s)) kw++; else id++;
            } else if (Character.isDigit(c)) {
                StringBuilder number = new StringBuilder();
                number.append(c);
                while ((ch = r.read())!=-1) {
                    c=(char)ch;
                    if (Character.isDigit(c) || c=='.') number.append(c);
                    else { r.reset(); break; }
                    r.mark(1);
                }
                num++;
            } else {
                if (ops.indexOf(c)>=0) op++;
                else if (!Character.isWhitespace(c)) special++;
            }
            r.mark(1);
        }
        r.close();
        System.out.println("Keywords: "+kw);
        System.out.println("Identifiers: "+id);
        System.out.println("Numbers: "+num);
        System.out.println("Operators: "+op);
        System.out.println("Special symbols: "+special);
    }
}
```

2) `RemoveCommentsWhitespace.java` — remove `//`, `/*...*/` comments and compress extra whitespace

Usage: **`java RemoveCommentsWhitespace source.c > cleaned.c`**

```java
// RemoveCommentsWhitespace.java
import java.io.*;

public class RemoveCommentsWhitespace {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java RemoveCommentsWhitespace
source.c"); return; }
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        StringBuilder out = new StringBuilder();
        int state = 0; // 0 normal,1 slash,2 line-comment,3 block-comment,4 string
        int ch;
        while ((ch = br.read()) != -1) {
            char c = (char) ch;
            if (state==0) {
                if (c=='/') { state=1; }
                else if (c=='"') { out.append(c); state=4; }
                else if (Character.isWhitespace(c)) {
                    if (out.length()>0 && !Character.isWhitespace(out.charAt(out.length()-1)))
out.append(' ');
                } else out.append(c);
            } else if (state==1) {
                if (c=='/') { state=2; }
                else if (c=='*') { state=3; }
                else { out.append('/'); out.append(c); state=0; }
            } else if (state==2) {
                if (c=='\n') { out.append('\n'); state=0; }
            } else if (state==3) {
                if (c=='*') {
                    br.mark(1);
                    int nx = br.read();
                    if (nx== -1) break;
                    if ((char)nx == '/') state=0;
                    else br.reset();
                }
            } else if (state==4) {
                out.append(c);
                if (c=='\\') { int nx = br.read(); if (nx!=-1) out.append((char)nx); }
                else if (c=='"') state=0;
            }
        }
        br.close();
        System.out.print(out.toString());
    }
}
```

3) `Lexemes.java` — extract and print lexemes/tokens

Usage: **`java Lexemes source.c`**

```java
// Lexemes.java
import java.io.*;
import java.util.*;

public class Lexemes {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java Lexemes source.c"); return; }
        FileReader fr = new FileReader(args[0]);
        StreamTokenizer st = new StreamTokenizer(fr);
        st.ordinaryChars('0','9'); // let numbers be tokens we handle manually
        st.wordChars('_', '_');
        st.wordChars('0','9'); // allow words with digits
        List<String> tokens = new ArrayList<>();
        int t;
        while ((t = st.nextToken()) != StreamTokenizer.TT_EOF) {
            if (t==StreamTokenizer.TT_WORD) tokens.add(st.sval);
            else if (t==StreamTokenizer.TT_NUMBER) tokens.add(String.valueOf(st.nval));
            else { tokens.add(String.valueOf((char)st.ttype)); }
        }
        fr.close();
        System.out.println(String.join(", ", tokens));
    }
}
```

---

4) `ClassifyWords.java` — for a single input line, classify each word as keyword, identifier, or constant

Usage: **`echo "int a = 5;" | java ClassifyWords`**

```java
// ClassifyWords.java
import java.io.*;
import java.util.*;

public class ClassifyWords {
    static final Set<String> KEYWORDS = new HashSet<>(Arrays.asList(
        "int","float","return","char","for","while","if","else","break","scanf","printf"
    ));
    private static boolean isNumber(String s){
        try { Double.parseDouble(s); return true; } catch(Exception e) { return false; }
    }
    private static boolean isIdentifier(String s){
        if (s.isEmpty()) return false;
        if (!(Character.isLetter(s.charAt(0)) || s.charAt(0)=='_')) return false;
        for (int i=1;i<s.length();i++) {
            char c=s.charAt(i);
            if (!(Character.isLetterOrDigit(c) || c=='_')) return false;
        }
        return !KEYWORDS.contains(s);
    }
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String line = br.readLine();
        if (line==null) return;
        String[] parts = line.split("\\s+|(?=[;=+\\-*/%(){}\\[\\],])|(?<=[;=+\\-*/%(){}\\
[\\],])");
        for (String p: parts) {
            if (p.trim().isEmpty()) continue;
            if (KEYWORDS.contains(p)) System.out.println(p+" : keyword");
            else if (isNumber(p)) System.out.println(p+" : constant");
            else if (isIdentifier(p)) System.out.println(p+" : identifier");
            else System.out.println(p+" : unknown");
        }
    }
}
```

---

5) `CategorizeOperators.java` — categorize operators into arithmetic, relational, logical

Usage: **`echo "a==b && c+d" | java CategorizeOperators`**

```java
// CategorizeOperators.java
import java.io.*;

public class CategorizeOperators {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String line = br.readLine(); if (line==null) return;
        for (int i=0;i<line.length(); ){
            char c = line.charAt(i);
            if (Character.isWhitespace(c)) { i++; continue; }
            if ("+-*/%".indexOf(c)>=0) {
                System.out.println("Arithmetic: "+c); i++;
            } else if ((c=='=' || c=='!' || c=='<' || c=='>') && i+1<line.length() &&
line.charAt(i+1)=='=') {
                System.out.println("Relational: "+c+line.charAt(i+1)); i+=2;
            } else if (c=='<'||c=='>') {
                System.out.println("Relational: "+c); i++;
            } else if (c=='&' && i+1<line.length() && line.charAt(i+1)=='&') {
                System.out.println("Logical: &&"); i+=2;
            } else if (c=='|' && i+1<line.length() && line.charAt(i+1)=='|') {
                System.out.println("Logical: ||"); i+=2;
            } else if (c=='!') { System.out.println("Logical: !"); i++; }
            else i++;
        }
    }
}
```

---

6) `RecognizeConstants.java` — recognize integer, float, character constants in a line

Usage: **`echo "int x = 45; float y = 4.5; char c = 'a';" | java RecognizeConstants`**

```java
// RecognizeConstants.java
import java.io.*;

public class RecognizeConstants {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String s = br.readLine(); if (s==null) return;
        for (int i=0;i<s.length();){
            char c = s.charAt(i);
            if (c=='\'') {
                int j=i+1;
                if (j<s.length() && s.charAt(j)=='\\') j+=2;
                else j++;
                if (j<s.length() && s.charAt(j)=='\'') {
                    System.out.println("Character: "+s.substring(i,j+1));
                    i=j+1; continue;
                } else { i++; }
            } else if (Character.isDigit(c)) {
                int j=i; boolean dot=false;
                while (j<s.length() && (Character.isDigit(s.charAt(j)) || s.charAt(j)=='.')) {
                    if (s.charAt(j)=='.') dot=true;
                    j++;
                }
                if (dot) System.out.println("Float: "+s.substring(i,j)); else
System.out.println("Integer: "+s.substring(i,j));
                i=j; continue;
            } else i++;
        }
    }
}
```

7) `StringLiterals.java` — extract string literals from a source file

Usage: **`java StringLiterals source.c`**

```java
// StringLiterals.java
import java.io.*;

public class StringLiterals {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java StringLiterals source.c"); return;
}
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        int ch; boolean inString=false; StringBuilder sb = new StringBuilder();
        while ((ch = br.read())!=-1) {
            char c = (char) ch;
            if (!inString) {
                if (c=='"') { inString=true; sb.append(c); }
            } else {
                sb.append(c);
                if (c=='\\') { int nx=br.read(); if (nx!=-1) sb.append((char)nx); else break; }
                else if (c=='"') { System.out.println(sb.toString()); sb.setLength(0);
inString=false; }
            }
        }
        br.close();
    }
}
```

8) `SummaryCounts.java` — total lines, tokens (approx), comments (approx)

Usage: **`java SummaryCounts source.c`**

```java
// SummaryCounts.java
import java.io.*;

public class SummaryCounts {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java SummaryCounts source.c"); return;
}
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        String line; long lines=0, tokens=0, comments=0;
        while ((line = br.readLine())!=null) {
            lines++;
            String t = line;
            // remove string literals to avoid counting tokens inside them
            t = t.replaceAll("\"(\\\\.|[^\"\\\\])*\"", " ");
            // count // comments
            int idx = t.indexOf("//");
            if (idx!=-1) { comments++; t = t.substring(0, idx); }
            // count /* */ occurrences (simple)
            if (t.contains("/*")) comments++;
            // approximate tokens: split by whitespace and punctuation
            String[] parts = t.split("\\s+|(?=[{}();,\\[\\]])|(?<=[{}();,\\[\\]])");
            for (String p: parts) if (p.trim().length()>0) tokens++;
        }
        br.close();
        System.out.println("Total number of lines: "+lines);
        System.out.println("Total number of tokens (approx): "+tokens);
        System.out.println("Total number of comments (approx): "+comments);
    }
}
```

9) `ValidIdentifier.java` — check if a string is a valid C identifier and not a keyword

Usage: **`java ValidIdentifier _count1`**

```java
// ValidIdentifier.java
import java.util.*;

public class ValidIdentifier {
    static final Set<String> KEYWORDS = new HashSet<>(Arrays.asList(

"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","scanf","printf","break
"
    ));
    public static boolean isValid(String s) {
        if (s==null || s.isEmpty()) return false;
        char c = s.charAt(0);
        if (!(Character.isLetter(c) || c=='_')) return false;
        for (int i=1;i<s.length();i++) {
            char ch = s.charAt(i);
            if (!(Character.isLetterOrDigit(ch) || ch=='_')) return false;
        }
        return !KEYWORDS.contains(s);
    }
    public static void main(String[] args) {
        if (args.length!=1) { System.out.println("Usage: java ValidIdentifier identifier");
return; }
        System.out.println(args[0] + (isValid(args[0]) ? " → Valid" : " → Invalid"));
    }
}
```

10) `LexicalErrors.java` — detect lexical errors: invalid identifiers, unclosed strings, unfinished comments (with line numbers)

Usage: **`java LexicalErrors source.c`**

```java
// LexicalErrors.java
import java.io.*;
import java.util.regex.*;

public class LexicalErrors {
    static final Pattern ID_PATTERN = Pattern.compile("[A-Za-z_][A-Za-z0-9_]*");
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java LexicalErrors source.c"); return;
}
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        String line; int lineNo=0;
        boolean inBlockComment=false;
        while ((line = br.readLine())!=null) {
            lineNo++;
            String s = line;
            // check unfinished block comment across lines
            if (inBlockComment) {
                if (s.contains("*/")) { inBlockComment=false; s =
s.substring(s.indexOf("*/")+2); }
                else continue;
            }
            if (s.contains("/*")) {
                if (!s.contains("*/")) {
                    System.out.println("Line "+lineNo+": Unfinished comment (/* ... not
closed)");
                    inBlockComment=true;
                } else {
                    // ok, remove comment
                    s = s.substring(0, s.indexOf("/*")) + s.substring(s.indexOf("*/")+2);
                }
            }
            // check unclosed string literal in the line
            int quoteCount = 0;
            for (int i=0;i<s.length();i++) if (s.charAt(i)=='"') {
                if (i>0 && s.charAt(i-1)=='\\') continue;
                quoteCount++;
            }
            if (quoteCount%2!=0) System.out.println("Line "+lineNo+": Unclosed string literal");
            // find tokens starting with digit letter combos for invalid identifiers like 1abc
            String[] parts = s.split("\\s+|(?=[;=+\\-*/%(){}\\[\\],])|(?<=[;=+\\-*/%(){}\\
[\\],])");
            for (String p: parts) {
                if (p.isEmpty()) continue;
                if (p.matches("[0-9]+[A-Za-z_].*")) {
                    System.out.println("Line "+lineNo+": Invalid identifier (starts with digit):
"+p);
                } else if (p.matches("[A-Za-z_].*")) {
                    // verify token is valid identifier when it is identifier-like
                    if (!ID_PATTERN.matcher(p).matches()) {
                        System.out.println("Line "+lineNo+": Invalid identifier characters:
"+p);
                    }
                } else if (p.startsWith("\"") && !p.endsWith("\"")) {
                    System.out.println("Line "+lineNo+": Possibly unclosed string: "+p);
                }
            }
        }
        br.close();
    }
}
```

11) `PreprocessorExtract.java` — extract # preprocessor directives

Usage: **`java PreprocessorExtract source.c`**

```java
// PreprocessorExtract.java
import java.io.*;

public class PreprocessorExtract {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java PreprocessorExtract source.c");
return; }
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        String line;
        while ((line = br.readLine())!=null) {
            String t = line.trim();
            if (t.startsWith("#")) System.out.println(line);
        }
        br.close();
    }
}
```

12) `KeywordCount.java` — count occurrences of each keyword in source file

Usage: **`java KeywordCount source.c`**

```java
// KeywordCount.java
import java.io.*;
import java.util.*;

public class KeywordCount {
    static final String[] KEYWORDS =
{"int","return","if","while","for","else","break","char","float","double","scanf","printf"};
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java KeywordCount source.c"); return; }
        Map<String,Integer> counts = new LinkedHashMap<>();
        for (String k: KEYWORDS) counts.put(k,0);
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        String line;
        while ((line = br.readLine())!=null) {
            line = line.replaceAll("\"(\\\\.|[^\"\\\\])*\"", " "); // remove strings
            String[] tokens = line.split("\\W+");
            for (String t: tokens) {
                if (counts.containsKey(t)) counts.put(t, counts.get(t)+1);
            }
        }
        br.close();
        for (String k: KEYWORDS) System.out.println(k+" : "+counts.get(k));
    }
}
```

13) `ExtractMultilineComments.java` — print all `/* ... */` blocks or message if none

Usage: **`java ExtractMultilineComments source.c`**

```java
// ExtractMultilineComments.java
import java.io.*;
```

```
public class ExtractMultilineComments {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java ExtractMultilineComments
source.c"); return; }
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        StringBuilder sb = new StringBuilder();
        String line;
        boolean inComment = false;
        boolean found = false;
        while ((line = br.readLine())!=null) {
            if (!inComment) {
                int start = line.indexOf("/*");
                if (start!=-1) {
                    inComment=true;
                    found=true;
                    sb.append(line.substring(start));
                    sb.append('\n');
                    if (line.indexOf("*/", start+2)!=-1) {
                        inComment=false;
                    }
                }
            } else {
                sb.append(line).append('\n');
                if (line.contains("*/")) inComment=false;
            }
            if (!inComment && found) {
                // print and reset buffer for next comment
                System.out.print(sb.toString());
                sb.setLength(0);
                found = false;
            }
        }
        br.close();
        if (sb.length()>0) { System.out.print(sb.toString()); }
        if (sb.length()==0 && !inComment) {
            // check if any output printed; if none, print fallback by scanning file
            // Simpler: re-open to check existence quickly
            // Instead, if nothing printed earlier, say no multi-line comments.
            // We'll reopen to be safe:
            BufferedReader br2 = new BufferedReader(new FileReader(args[0]));
            boolean any=false;
            while ((line=br2.readLine())!=null) { if (line.contains("/*")) { any=true; break; }
}
            br2.close();
            if (!any) System.out.println("No multi-line comments found.");
        }
    }
}
```

14) `CharFrequency.java` — frequency of each character in source file

Usage: **`java CharFrequency source.c`**

```java
// CharFrequency.java
import java.io.*;
import java.util.*;

public class CharFrequency {
    public static void main(String[] args) throws Exception {
        if (args.length!=1) { System.out.println("Usage: java CharFrequency source.c"); return;
}
        long[] freq = new long[256];
        FileInputStream fis = new FileInputStream(args[0]);
        int b;
        while ((b = fis.read())!=-1) freq[b & 0xFF]++;
        fis.close();
        // Print letters
        for (char c='a'; c<='z'; c++) if (freq[c]>0) System.out.println("'" + c + "' : " +
freq[c]);
        for (char c='A'; c<='Z'; c++) if (freq[c]>0) System.out.println("'" + c + "' : " +
freq[c]);
        for (char c='0'; c<='9'; c++) if (freq[c]>0) System.out.println("'" + c + "' : " +
freq[c]);
        char[] symbols = {'{','}','(',')',';','[',']',',','<','>','+','-
','*','/','=','%','&','|','!','^','~','#',':','\'','\"','\\','\n','\t',' '};
        for (char ch: symbols) {
            long count = freq[ch & 0xFF];
            if (count>0) {
                String label = (ch=='\n')? "'\\n'": (ch=='\t')? "'\\t'": (ch==' ')? "' '": "'" +
ch + "'";
                System.out.println(label + " : " + count);
            }
        }
    }
}
```

Final notes & next steps

- These Java programs follow the same behavior as the C versions I previously gave. They're written to be readable and easily modifiable for extra robustness.
- If you want:
  1. I can create a single **`AssignmentRunner.java`** that bundles all tasks with command-line options (e.g., **`java AssignmentRunner 1 source.c`**) — convenient for testing.
  2. Or I can generate a **`Makefile`**-style shell script to compile all **`.java`** files and run tests.
  3. Or I can zip all files and give them as one downloadable package.

Tell me which of (1), (2), or (3) you want and I'll produce it immediately.

Got it — I rewrote the **14 assignment programs in Java**. Each program is a self-contained **`.java`** file with a **`main`** method and a usage note. These are practical, assignment-level lexical utilities (not full compilers); they handle typical student C source code and common edge cases (strings, comments, escapes) but are not guaranteed to handle every pathological C corner case.

Paste each file into its own **`.java`** file (filename matches the public class name), compile with **`javac`**, run with **`java`**. If you want, I can instead pack everything into a single multi-command program or create a Maven project — say the option number from the previous message and I'll produce that.

# 1) **`CountTokens.java`** — count keywords, identifiers, operators, numbers, special symbols (file input)

Usage: **`java CountTokens source.c`**