Preparations (do these first)

Open the Ubuntu terminal in your Oracle VM and run:

```
sudo apt update

sudo apt install -y flex bison gcc make

sudo apt install -y gdb nano
```

EXP 1 – Count comments, keywords, identifiers, words, lines, spaces (Lex)

```
mkdir ~/lex_programs

cd ~/lex_programs

gedit count.l
```

Paste this:

```
%{

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int comment_count=0,keyword_count=0,identifier_count=0,word_count=0,line_count=1,space_count=0;

char
*keywords[]={"int","float","double","char","if","else","for","while","return","break","continue","void","switch","ca
se","default",NULL};

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

%}

%x COMMENT
```

```
%%
"/*"            { BEGIN(COMMENT); comment_count++; }
<COMMENT>[^*]+      { }
<COMMENT>"*/"       { BEGIN(INITIAL); }
"//".*          { comment_count++; }
[ \t]+          { space_count += yyleng; }
\n              { line_count++; }
[A-Za-z_][A-Za-z0-9_]* { word_count++; if(is_keyword(yytext)) keyword_count++; else identifier_count++; }
[0-9]+(\.[0-9]+)?   { word_count++; }
.               { }
%%

int main(int argc,char **argv){
   if(argc>1){ FILE *f=fopen(argv[1],"r"); if(!f){ perror("fopen"); return 1;} yyin=f;}
   yylex();
   printf("Lines: %d\nSpaces/Tabs: %d\nWords: %d\nKeywords: %d\nIdentifiers: %d\nComments: %d\n",
line_count,space_count,word_count,keyword_count,identifier_count,comment_count);
   return 0;
}
```

Create input file:

```
gedit sample.c
```

Paste sample C code:

```
#include <stdio.h>
int main() {
   int a=5; // comment
   /* comment */
```

```
    int num=a+10;
    return 0;
}
```

Compile & run:

```
flex count.l
gcc lex.yy.c -lfl -o count
./count sample.c
```

EXP 2 – Count words starting with ‘A’ (Lex)

```
gedit startA.l
```

Paste:

```
%{
#include <stdio.h>
int countA=0;
%}

%%
([Aa][A-Za-z0-9_]*) { countA++; }
.|\n                { }
%%

int main(int argc,char **argv){
    if(argc>1) yyin=fopen(argv[1],"r");
    yylex();
    printf("Words starting with A/a: %d\n", countA);
    return 0;
}
```

Input file:

```
gedit words.txt
```

```
Apple apple aardvark Ball A2 12A
```

Compile & run:

```
flex startA.l
gcc lex.yy.c -lfl -o startA
./startA words.txt
```

EXP 3 – Conversion of lowercase ↔ uppercase (C)

```
gedit swapcase.c
```

Paste:

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

int main(int argc,char **argv){
   FILE *in=stdin;
   if(argc>1) in=fopen(argv[1],"r");
   if(!in){ perror("fopen"); return 1;}
   int c;
   while((c=fgetc(in))!=EOF){
      if(islower(c)) putchar(toupper(c));
```

```
        else if(isupper(c)) putchar(tolower(c));
        else putchar(c);
    }
    if(in!=stdin) fclose(in);
    return 0;
}
```

Input file:

```
gedit text.txt
```

```
Hello World abc DEF
```

Compile & run:

```
gcc swapcase.c -o swapcase
./swapcase text.txt
```

EXP 4 – Decimal → Hexadecimal (C)

```
gedit dec2hex.c
```

Paste:

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

int main(int argc,char **argv){
    FILE *in=stdin;
    if(argc>1) in=fopen(argv[1],"r");
    if(!in){ perror("fopen"); return 1;}
    int c;
    while((c=fgetc(in))!=EOF){
        if(isdigit(c)){
            long num=c-'0';
            while((c=fgetc(in))!=EOF && isdigit(c)) num=num*10+(c-'0');
            printf("0x%lX",num);
            if(c==EOF) break;
            putchar(c);
        }else putchar(c);
    }
    if(in!=stdin) fclose(in);
    return 0;
}
```

Input file:

```
gedit nums.txt
```

```
val1=15; val2=255; x=10 apples 1234
```

Compile & run:

```
gcc dec2hex.c -o dec2hex
./dec2hex nums.txt
```

EXP 5 – Lines ending with "com" or ".org" (Lex)

```
gedit endings.l
```

Paste:

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

%%
^[^\n]*com[ \t]*$   { printf("Line ends with 'com': %s\n", yytext); }
^[^\n]*\.org[ \t]*$ { printf("Line ends with '.org': %s\n", yytext); }
[^\n]+              { }
\n                  { }
%%

int main(int argc,char **argv){
    if(argc>1) yyin=fopen(argv[1],"r");
    yylex();
    return 0;
}
```

Input file:

```
gedit lines.txt
```

```
hello.com
example.org
notmatch.comx
```

Compile & run:

```
flex endings.l
gcc lex.yy.c -lfl -o endings
./endings lines.txt
```

EXP 6 – Postfix Expression Evaluation (C)

```
gedit postfix.c
```

Paste:

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

#define MAX 100
int stack[MAX],top=-1;
void push(int v){ stack[++top]=v; }
int pop(){ return stack[top--]; }

int main(int argc,char **argv){
    char buf[1024];
    FILE *f=stdin;
    if(argc>1) f=fopen(argv[1],"r");
    while(fgets(buf,sizeof(buf),f)){
        top=-1;
        char *tok=strtok(buf," \t\n");
        while(tok){
            if(strcmp(tok,"+")==0){ int b=pop(),a=pop(); push(a+b); }
            else if(strcmp(tok,"-")==0){ int b=pop(),a=pop(); push(a-b); }
            else if(strcmp(tok,"*")==0){ int b=pop(),a=pop(); push(a*b); }
            else if(strcmp(tok,"/")==0){ int b=pop(),a=pop(); push(a/b); }
            else push(atoi(tok));
            tok=strtok(NULL," \t\n");
        }
        if(top==0) printf("Result: %d\n",pop());
        else printf("Error\n");
    }
    if(f!=stdin) fclose(f);
    return 0;
}
```

Input file:

```
gedit pf.txt
```

```
3 4 + 2 * 7 /
```

Compile & run:

```
gcc postfix.c -o postfix
./postfix pf.txt
```

EXP 7 – Desk Calculator with Error Recovery (YACC)

Already done in previous message.

EXP 8 – YACC Parser for FOR Loop Statements

```
gedit forloop.l
```

Lex file:

```
%{
#include "y.tab.h"
%}
%%
for      { return FOR; }
[ \t\n]+ { }
.        { return *yytext; }
%%
int yywrap(){ return 1; }
```

```
gedit forloop.y
```

YACC file:

```
%{
#include <stdio.h>
int yylex();
void yyerror(const char *s){ printf("Error: %s\n",s); }
%}
%token FOR
%%
start: statements ;
statements: statements statement | statement ;
statement: FOR { printf("FOR keyword found\n"); } ;
%%
int main(){ yyparse(); return 0; }
```

Compile & run:

```
flex forloop.l
```

```
yacc -d forloop.y
gcc lex.yy.c y.tab.c -o forloop -ll
echo "for(int i=0;i<10;i++){}" > forsample.c
./forloop < forsample.c
```

EXP 9 – YACC Parser IC Generator for Arithmetic Expressions

gedit ic.l

```
%{
#include "y.tab.h"
%}
%%
[0-9]+   { yylval=atoi(yytext); return NUM; }
[\+\-\*/] { return *yytext; }
[ \t\n]+ { }
.        { }
%%
int yywrap(){return 1;}
```

gedit ic.y

```
%{
#include <stdio.h>
int temp=1;
int yylex();
void yyerror(const char *s){ printf("Error: %s\n",s);}
%}
%token NUM
%%
start: expr { }
expr: NUM { printf("LOAD t%d,%d\n",temp++,$1); }
    | expr '+' expr { printf("+\n"); }
    | expr '-' expr { printf("-\n"); }
    | expr '*' expr { printf("*\n"); }
    | expr '/' expr { printf("/\n"); }
    ;
%%
int main(){ yyparse(); return 0; }
```

Compile & run:

```
flex ic.l
yacc -d ic.y
gcc lex.yy.c y.tab.c -o icgen -ll
echo "3 + 4 * 2" > expr.txt
./icgen < expr.txt
```

EXP 10 – YACC Simple Calculator

Similar to EXP 7 — you can reuse YACC desk calculator for 8–10 style arithmetic parsing.

EXP 11 – LEX Email Checker

```
gedit email.l
```

```
%{
#include <stdio.h>
%}
%%
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,} { printf("Valid Email: %s\n",yytext);}
[^\n]+                                  { }
\n                                      { }
%%
int main(int argc,char **argv){
    if(argc>1) yyin=fopen(argv[1],"r");
    yylex();
    return 0;
}
```

Input:

```
gedit emails.txt
```

```
test@example.com
invalid-email@
abc.def@org
```

Compile & run:

```
flex email.l
gcc lex.yy.c -lfl -o email
./email emails.txt
```

EXP 12 – LEX Simple Calculator

```
gedit simplecalc.l
```

```
%{
#include <stdio.h>
%}
%%
[0-9]+      { printf("%s ",yytext);}
[\+\-\*/]   { printf("%s ",yytext);}
[ \t\n]+    { }
.           { }
```

```
%%
int main(int argc,char **argv){
    if(argc>1) yyin=fopen(argv[1],"r");
    yylex();
    return 0;
}
```

Input:

```
gedit scalc.txt
```

```
3 + 4 * 2
```

Compile & run:

```
flex simplecalc.l
gcc lex.yy.c -lfl -o scalc
./scalc scalc.txt
```