1)Write a c/c++ program to identify whether the given lexeme is a valid identifier.

Code:

```
 #include <iostream>
#include <vector>
#include <cctype>
using namespace std;

bool isKeyword(const string &lexeme)
{
   vector<string> 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", "struct", "switch", "typedef",
      "union", "unsigned", "void", "volatile", "while", "class", "namespace", "public",
      "private", "protected", "virtual", "friend", "inline", "operator", "template"
   };

   for (const string & keyword : keywords)
   {
      if (lexeme == keyword)
      {
         return true;
      }
   }
   return false;
}


bool isValidIdentifier(const string &lexeme)
{
   if (lexeme.empty() || (!isalpha(lexeme[0]) && lexeme[0] != '_'))
   {
      return false;
   }
   for (char ch : lexeme)
   {
      if (!isalnum(ch) && ch != '_')
      {
         return false;
      }
```

```
    }
    return !isKeyword(lexeme);
}

int main()
{
    string lexeme;
    while (true)
    {
        cout << "Enter a lexeme : ";
        cin >> lexeme;

        if (lexeme == "exit")
        {
            break;
        }

        if (isValidIdentifier(lexeme))
        {
            cout << lexeme << " is a valid identifier." << endl;
        }
        else
        {
            cout << lexeme << " is NOT a valid identifier." << endl;
        }
    }

    return 0;
}
```

OUTPUT:

```
 Enter a lexeme : _variableName
_variableName is a valid identifier.
Enter a lexeme : myVar123
myVar123 is a valid identifier.
Enter a lexeme : data_value
data_value is a valid identifier.
Enter a lexeme : CamelCase
CamelCase is a valid identifier.
Enter a lexeme : 123variable
123variable is NOT a valid identifier.
Enter a lexeme : my-var
my-var is NOT a valid identifier.
Enter a lexeme : data@value
data@value is NOT a valid identifier.
```

2) Write a c/c++ program to identify whether the given lexeme is a valid floating point.

Code:

```
#include <iostream>
#include <cctype>
using namespace std;
bool isValidFloatingPoint(const string &lexeme)
{
   bool hasDecimalPoint = false;
   bool hasDigits = false;

   for (size_t i = 0; i < lexeme.size(); ++i)
   {
      if (isdigit(lexeme[i]))
      {
         hasDigits = true;
      }
      else if (lexeme[i] == '.')
      {
         if (hasDecimalPoint)
         {
            return false;
         }
         hasDecimalPoint = true;
      }
      else if ((lexeme[i] == '+' || lexeme[i] == '-') && i == 0)
      {
         continue;
      }
      else
      {
         return false;
      }
   }

   return hasDigits && hasDecimalPoint;
}

int main()
{
   string lexeme;
   while (true)
   {
      cout << "Enter a lexeme : ";
```

```
        cin >> lexeme;

        if (lexeme == "exit")
        {
            break;
        }

        if (isValidFloatingPoint(lexeme))
        {
            cout << lexeme << " is a valid floating-point number." << endl;
        }
        else
        {
            cout << lexeme << " is NOT a valid floating-point number." << endl;
        }
    }

    return 0;
}
```

OUTPUT:

```
Enter a lexeme : 3.6
3.6 is a valid floating-point number.
Enter a lexeme : 3..6
3..6 is NOT a valid floating-point number.
Enter a lexeme : .099
.099 is a valid floating-point number.
Enter a lexeme : 808.
808. is a valid floating-point number.
Enter a lexeme : avv
avv is NOT a valid floating-point number.
Enter a lexeme : -4.6
-4.6 is a valid floating-point number.
Enter a lexeme : 10
10 is NOT a valid floating-point number.
```

3) Write a flex program to count total number of tokens and Identify tokens (Integer Number, Floating Points, Exponential Number, Identifiers, Keywords, Operators, Symbols).

```
%{
#include <stdio.h>
#include <string.h>
int total_tokens = 0;
```

```
void print_token(const char *type, const char *value) {
   total_tokens++;
   printf("%-20s: %s\n", type, value);
}
%}

%%
[ \t\n]+            ;
"int"|"float"|"char"|"if"|"else"|"return"|"void" { print_token("Keyword", yytext); }
[+-]?[0-9]+         { print_token("Integer", yytext); }
[+-]?[0-9]*\.[0-9]+   { print_token("Floating Point", yytext); }
[+-]?[0-9]+\.[0-9]+[eE][+-]?[0-9]+ { print_token("Exponential Number", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { print_token("Identifier", yytext); }
"+"|"-"|"*"|"/"|"="   { print_token("Operator", yytext); }
"("|")"|"{"|"}"|";"   { print_token("Symbol", yytext); }
.                   { print_token("Unknown", yytext); }

%%

int main() {
   printf("Token Type         : Token\n");
   printf("--------------------:---------------\n");
   yylex(); // Call the Flex scanner
   printf("\nTotal Tokens: %d\n", total_tokens);
   return 0;
}
```

OUTPUT:

```
int x=10;
Token Type      : Token
--------------------:----------------
Keyword          : int
Identifier       : x
Operator         : =
Integer          : 10
Symbol           : ;

Total Tokens    : 5
```

4) Write a flex program to Identify and Count Positive, Negative Numbers, Even Numbers, Odd Numbers and Fractions.

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

```
#include <string.h>

int total_tokens = 0, pos_count = 0, neg_count = 0, even_count = 0, odd_count = 0, frac_count = 0;

void print_token(const char *type, const char *value) {
   total_tokens++;
   printf("%-20s: %s\n", type, value);
   if (strcmp(type, "Positive") == 0) pos_count++;
   else if (strcmp(type, "Negative") == 0) neg_count++;
   else if (strcmp(type, "Even") == 0) even_count++;
   else if (strcmp(type, "Odd") == 0) odd_count++;
   else if (strcmp(type, "Fraction") == 0) frac_count++;
}
%}

%%

[ \t\n]+                    ;
[+-]?[0-9]+/[0-9]+          { print_token("Fraction", yytext); frac_count++; }
[+-]?[0-9]+                 {
   int num = atoi(yytext);
   if (num > 0) {
      print_token("Positive", yytext);
      if (num % 2 == 0) {
         print_token("Even", yytext);
      } else {
         print_token("Odd", yytext);
      }
   } else if (num < 0) {
      print_token("Negative", yytext);
      if (num % 2 == 0) {
         print_token("Even", yytext);
      } else {
         print_token("Odd", yytext);
      }
   } else {
      print_token("Zero", yytext);
   }
}

.|\n                        { /* Ignore other characters */ }

%%

int main() {
   char input[1024];
   printf("Enter a list of numbers (positive, negative, even, odd, fractions) on the same line:\n");
   printf("Example input: -5 4 2/3 3 -7.8 8 0 1/2\n");
```

```
    fgets(input, sizeof(input), stdin);
    YY_BUFFER_STATE buffer = yy_scan_string(input);
    yylex();
    yy_delete_buffer(buffer);
    printf("\nSummary:\n");
    printf("Total Tokens: %d\n", total_tokens);
    printf("Positive: %d\n", pos_count);
    printf("Negative: %d\n", neg_count);
    printf("Even: %d\n", even_count);
    printf("Odd: %d\n", odd_count);
    printf("Fractions: %d\n", frac_count);

    return 0;
}
```

OUTPUT:

```
Token Type      : Token
-----------------:----------------
Negative         : -5
Odd              : -5
Positive         : 4
Even             : 4
Fraction         : 2/3
Positive         : 3
Negative         : -7.8
Positive         : 8
Even             : 8
Zero             : 0
Fraction         : 1/2

Summary:
Total Tokens: 10
Positive: 4
Negative: 2
Even: 3
Odd: 3
Fractions: 2
```

5) Write a YACC program to evaluate a given arithmetic expression.

Lab.l => File

```
%{
#include "y.tab.h"
%}
```

```
%%
[0-9]+  { yylval = atoi(yytext); return NUMBER; }
[ \t]   { /* ignore whitespace */ }
\n      { return '\n'; }
.       { return yytext[0]; }
%%

int yywrap(void) {
   return 1;
}
```

Lab.y => File

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

void yyerror(const char *s);
int yylex(void);
%}

%token NUMBER

%left '+' '-'
%left '*' '/'
%right UMINUS

%%
input:
   | input line
   ;

line:
   '\n'
   | expr '\n' { printf("Result: %d\n", $1); }
   ;

expr:
   NUMBER
   | expr '+' expr { $$ = $1 + $3; }
   | expr '-' expr { $$ = $1 - $3; }
   | expr '*' expr { $$ = $1 * $3; }
   | expr '/' expr { $$ = $1 / $3; }
   | '-' expr %prec UMINUS { $$ = -$2; }
```

```
   | '(' expr ')' { $$ = $2; }
   ;

%%
void yyerror(const char *s) {
   fprintf(stderr, "Error: %s\n", s);
}

int main(void) {
   printf("Enter an arithmetic expression:\n");
   return yyparse();
}
```

OUTPUT:

```
Enter an arithmetic expression:
 4 + 2*3
Result: 10
```

6) Write a YACC program to convert a given hexadecimal value to decimal value.

Lab.l=>File

```
 %{
/* Definition section */
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h" /* Include the header file generated by YACC */
extern int yylval; /* `yylval` is used to store token values */
%}

/* Token Rules Section */
%%

[0-9]                { yylval = yytext[0] - '0'; return DIGIT; } /* Match digits 0-9 */
[a-fA-F]             { yylval = (yytext[0] | 32) - 'a' + 10; return DIGIT; } /* Match A-F/a-f and convert */
[ \t]       { /* Ignore whitespace */ } /* Skip spaces and tabs */
\n          { return 0; } /* End input on a newline */
.           { return yytext[0]; } /* Return any other character as-is */

%%

/* Helper function for Flex */
int yywrap() {
          return 1; /* Signals end of input */
}
```

Lab.y=>File

```
/* Definition section */
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex(void); /* Declare yylex() function for YACC to use */
void yyerror(const char *s); /* Error handling function */
%}

/* Token definitions */
%token DIGIT

/* Rule Section */
%%

/* Start rule */
N:
   L { printf("\nDecimal Value: %d\n", $1); } /* Output the decimal value */
   ;

/* Hexadecimal digit sequence */
L:
   L B { $$ = $1 * 16 + $2; } /* Multiply by 16 for conversion to decimal */
   | B { $$ = $1; } /* Single digit */
   ;

/* Base rule for a single hexadecimal digit */
B:
   DIGIT { $$ = $1; } /* The value of the digit */
   ;

%%

/* Error handling function */
void yyerror(const char *s) {
        fprintf(stderr, "Error: %s\n", s); /* Print error message */
}

/* Driver code */
int main() {
        printf("Enter a hexadecimal sequence (e.g., 1A3F): ");
        return yyparse(); /* Call the parser */
}
```

OUTPUT:

```
Enter a hexadecimal sequence (e.g., 1A3F): 1A3F
```

Decimal Value: 6719

Enter a hexadecimal sequence (e.g., 1A3F): ABC
Decimal Value: 2748

Enter a hexadecimal sequence (e.g., 1A3F): F
Decimal Value: 15

Enter a hexadecimal sequence (e.g., 1A3F): 123
Decimal Value: 291