EXP - 1 : Write a simple calculator program in C/C++/JAVA.

```
#include <iostream>
using namespace std;
int main()
{
	char op;
	float num1, num2;
	cin >> op;
	cin >> num1 >> num2;
	switch (op) {
	case '+':
		cout << num1 + num2;
		break;
	case '-':
		cout << num1 - num2;
		break;
	case '*':
		cout << num1 * num2;
		break;
	case '/':
		cout << num1 / num2;
		break;
	default:
		cout << "Error! operator is not correct";
	}
	return 0;
}
```

Run:

```
Sudo apt install g++ -y
g++ calc.cpp
./a.out
```

Exp - 2 : Write a FLEX Program

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

```
%}

%%

"FLEX" { flex_count++; }

%%
int yywrap(){}
int main()
{
   yylex();
   printf("Occurrences of 'FLEX': %d\n", flex_count);
   return 0;
}
```

Run :
ubuntu :
exp2.l
flex exp2.l
gcc lex.yy.c -lfl
./a.out < input.txt

Windows:
exp2.l
flex exp2.l
gcc lex.yy.c
a.exe < input2.txt

EXP - 3 : Implementation of scanner by specifying Regular Expressions.

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

%%

[0-9]+                { printf("INTEGER: %s\n", yytext); }
[0-9]*"."[0-9]+       { printf("FLOAT: %s\n", yytext); }
```

```
[a-zA-Z_][a-zA-Z0-9_]*  { printf("IDENTIFIER: %s\n", yytext); }
"+"|"-"|"*"|"/"         { printf("OPERATOR: %s\n", yytext); }
[ \t\n]+                /* ignore whitespace */
.                       { printf("UNKNOWN: %s\n", yytext); }

%%
int yywrap(){}
int main()
{
   yylex();
   return 0;
}
```

Run:

```
exp3.l
flex exp3.l
gcc lex.yy.c -lfl
./a.out < input.txt
Input: 123 + abc
```

EXP - 4 : write a simple program in bison

Lex code:

```
%{
#include <stdio.h>
#include "exp4.tab.h"
int flex_count = 0;

%}

%option noyywrap

%%

"FLEX" { flex_count++; }

%%
```

Bison Code:

```
%{
#include <stdio.h>
#include <stdlib.h> // Add this line
extern int flex_count;
int yylex(void);
void yyerror(const char *s);
%}

%token FLEX

%%

flexes:
    | flexes FLEX { flex_count++; }
    ;

%%

int main()
{
    yyparse(); // Add this line
    printf("Occurrences of 'FLEX': %d\n", flex_count);
    return 0;
}

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

Run :

Linux:

```
exp4.y
Exp4_lex.l
flex exp4.l
bison -d exp4.y
gcc exp4.tab.c lex.yy.c -lfl
./a.out < input.txt
```

FLEX FLEX FLEX
FLEX FLEX
FLEX
Occurances of Flex : 6

EXP - 5 : Top Down Parser

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

char lookahead;

void match(char expected) {
    if (lookahead == expected)
        lookahead = getchar();
}

void expression();
void term();
void factor();

void expression() {
    term();
    while (lookahead == '+' || lookahead == '-') {
        char op = lookahead;
        match(op);
        term();
        if (op == '+')
            printf("ADD\n");
        else if (op == '-')
            printf("SUB\n");
    }
}

void term() {
    factor();
    while (lookahead == '*' || lookahead == '/') {
        char op = lookahead;
        match(op);
        factor();
        if (op == '*')
            printf("MUL\n");
```

```
        else if (op == '/')
            printf("DIV\n");
    }
}

void factor() {
    if (isdigit(lookahead)) {
        printf("PUSH %c\n", lookahead);
        match(lookahead);
    } else if (lookahead == '(') {
        match('(');
        expression();
        match(')');
    }
    }
}

int main() {
    printf("Enter an arithmetic expression: ");
    lookahead = getchar();
    expression();
    }
    return 0;
}
```

EXP - 7 : Introduction to basic Java - Programs in java

```
public class TestClass {
    public static void main (String []args) {
        System.out.println ("Hello World is my first Java Program.");
    }
}
```

Run:
Class name and filename should be same
TestClass.java
javac TestClass.java
java TestClass

Exp - 8 :Write a program to traverse syntax trees and perform action arithmetic operations.

```
#include <iostream>
```

```
using namespace std;
struct Node {
    char value;
    Node* left = nullptr;
    Node* right = nullptr;

    Node(char value) : value(value) {}
};

void printTree(Node* node, int depth = 0) {
    if (!node) return;

    printTree(node->right, depth + 1);

    for (int i = 0; i < depth; ++i)
        cout << "    ";

    cout << node->value << endl;

    printTree(node->left, depth + 1);
}

double evaluate(Node* node) {
    if (!node) return 0;
    if (!node->left && !node->right) return node->value - '0';

    double left = evaluate(node->left);
    double right = evaluate(node->right);

    switch (node->value) {
        case '+': return left + right;
        case '-': return left - right;
        case '*': return left * right;
        case '/': return left / right;
    }

    return 0;
}


int main() {
    Node nodes[] = { '+', '1', '*', '2', '3' };
    nodes[0].left = &nodes[1];
    nodes[0].right = &nodes[2];
```

```
    nodes[2].left = &nodes[3];
    nodes[2].right = &nodes[4];

    printTree(&nodes[0]);

    cout << "Result: " << evaluate(nodes) << endl;

    return 0;
}
```

Run :
exp8.cpp
g++ exp8.cpp
./a.out

windows
a.exe

EXP - 9 : Write an Intermediate code generation for If/While.

```
#include <iostream>
#include <string>
using namespace std;

string gifc(string cond, string act) {
    string code;
    code += "if (" + cond + ") {\n";
    code += "\t" + act + "\n";
    code += "}";
    return code;
}

int main() {
    string c = "x > 5";
    string a = "y = x * 2;";
    string gc = gifc(c, a);
    cout << "Generated code:\n";
    cout << gc << endl;
    return 0;
}
```

EXP - 10 : Write a program for MIPS Assembly language- (Teach spim mips simulator)

```
.data
msg: .asciiz "Hello, World!"

.text
main:
   # Print the hello message
   li $v0, 4         # Load system call code for printing a string
   la $a0, msg  # Load the address of the hello message
   syscall

   # Exit program
   li $v0, 10        # Load system call code for exit
   syscall
```

Run :

`hello_world.asm`

Open SPIM, load the program file (File > Load File), and execute it (Run > Go).

EXP - 11 : Write a program to generate machine code for a simple statement.

```
.data
a : .word 10
b : .word 20
r : .word 0

.text
main:
lw $t0,a
lw $t1,b
add $t2, $t0,$t1
sw $t2, r
li $v0,1
move $a0 , $t2
syscall
li $v0, 10
syscall
```

EXP - 12 : Write a program to generate machine code for an indexed assignment statement.

```
.data
a : .word 10,20,30,40,50

.text
main:
li $t0 , 2
li $t1,100
la $t2 , a
sll $t0, $t0,2
add $t2, $t2, $t0
sw $t1, 0($t2)
lw $a0 , 0($t2)
li $v0, 1
syscall
li $v0 , 10
syscall
```

4 : print
10 : exit
2 : index in the array where we want to store a value.
Sll : shift left logical

1: Print integer. The integer to print should be in the `$a0` register.

4: Print string. The address of the null-terminated string to print should be in the `$a0` register.

5: Read integer. The integer read from input will be placed in the `$v0` register.

8: Read string. The address to store the input string should be in the `$a0` register, and the maximum length of the string should be in the `$a1` register.

10: Exit. This system call terminates the program.