Set-7

1. A School student was asked to do basic mathematical operations. Implement a LEX program to implement the same.

Code:

```
%{

#include <stdio.h>

%}


%%

// Match numbers

[0-9]+ { printf("Number: %s\n", yytext); }


// Match operators

[+\-*/] { printf("Operator: %s\n", yytext); }


// Ignore spaces and new lines

[ \t\n]+   ;


%%

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

   FILE *file;

   if (argc > 1) {

      file = fopen(argv[1], "r");

      if (!file) {

         printf("Cannot open file %s\n", argv[1]);

         return 1;

      }

      yyin = file;
```

```
    }
    yylex();
    return 0;
}
```

2. Write a LEX program to count the frequency of the given word in a given sentence.

Code:

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

char target[100];
int count = 0;
%}

%%
// Match the target word
[a-zA-Z]+ {
    if (strcmp(yytext, target) == 0) {
        count++;
    }
}

// Ignore spaces and new lines
[ \t\n]+   ;

%%
int main() {
    printf("Enter the word to count: ");
    scanf("%s", target);
    printf("Enter the sentence: ");
    yylex();
    printf("Frequency of '%s': %d\n", target, count);
    return 0;
}
```

3. Write a LEX code to replace a word with another word in a file.

Code:

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

char find[100], replace[100];
%}

%%
// Match the target word and replace it
[a-zA-Z]+ {
    if (strcmp(yytext, find) == 0) {
        printf("%s", replace);
    } else {
        printf("%s", yytext);
    }
}

// Preserve spaces and new lines
[ \t\n]+ { printf("%s", yytext); }
```

```
%%
int main() {
    printf("Enter the word to find: ");
    scanf("%s", find);
    printf("Enter the replacement word: ");
    scanf("%s", replace);
    printf("Enter the text:\n");
    yylex();
    return 0;
}
```

4. Write a C program to implement the back end of the compiler.

Code:

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

// Structure for an intermediate code instruction
typedef struct {
    char op[10];
    char arg1[10];
    char arg2[10];
    char result[10];
} Instruction;

// Function to generate assembly code from intermediate code
void generateAssembly(Instruction ic[], int n) {
    printf("Generated Assembly Code:\n");
    for (int i = 0; i < n; i++) {
```

```
    if (strcmp(ic[i].op, "+") == 0) {
      printf("MOV R0, %s\n", ic[i].arg1);
      printf("ADD R0, %s\n", ic[i].arg2);
      printf("MOV %s, R0\n", ic[i].result);
    } else if (strcmp(ic[i].op, "-") == 0) {
      printf("MOV R0, %s\n", ic[i].arg1);
      printf("SUB R0, %s\n", ic[i].arg2);
      printf("MOV %s, R0\n", ic[i].result);
    } else if (strcmp(ic[i].op, "*") == 0) {
      printf("MOV R0, %s\n", ic[i].arg1);
      printf("MUL R0, %s\n", ic[i].arg2);
      printf("MOV %s, R0\n", ic[i].result);
    } else if (strcmp(ic[i].op, "/") == 0) {
      printf("MOV R0, %s\n", ic[i].arg1);
      printf("DIV R0, %s\n", ic[i].arg2);
      printf("MOV %s, R0\n", ic[i].result);
    } else {
      printf("Unsupported operation: %s\n", ic[i].op);
    }
  }
}

int main() {
  int n;
  printf("Enter number of intermediate code instructions: ");
  scanf("%d", &n);
  Instruction ic[n];
```

```
    printf("Enter intermediate code in format (op arg1 arg2 result):\n");
    for (int i = 0; i < n; i++) {
        scanf("%s %s %s %s", ic[i].op, ic[i].arg1, ic[i].arg2, ic[i].result);
    }

    generateAssembly(ic, n);
    return 0;
}
```