Laboratory-2

QUESTION-1

Write a Lex and Yacc code for syntax analysis of arithmetic, boolean and relational operators in C.

Lex File (lex.l):

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "y.tab.h"
    extern yylval;
%}

%%
[a-zA-Z][a-zA-Z0-9]* {return id;}
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

Yacc File (yacc.y):

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    int res;
%}

%token id num

%%
stmt:expr{res=$$;};
expr:expr '+' expr {$$=$1+$3;}
    |expr '-' expr {$$=$1-$3;}
    |expr '*' expr {$$=$1*$3;}
    |expr '/' expr {$$=$1/$3;}
    |expr '<' expr {$$=($1<$3);}
    |expr '>' expr {$$=($1>$3);}
    |expr '<' '=' expr {$$=($1<=$4);}
    |expr '>' '=' expr {$$=($1>=$4);}
    |expr '=' '=' expr {$$=($1==$4);}
    |'(' expr ')'{$$=$2;}
    |id
    |num
    ;
%%

int main()
{
    printf("Enter an expression : ");
    yyparse();
    printf("\nThe result is : %d",res);
    return 0;
}
```

```
int yyerror()
{
    printf("Error!");
    exit(0);
}
```

Input/Output:

```
kal-el@mos-13:~/Desktop/Compilers/Compilers Lab/Lab2/q1$ ./a.out
Enter an expression : 12+569

The result is : 581kal-el@mos-13:~/Desktop/Compilers/Compilers Lab/Lab2/q1$
```

Result:

Lex and Yacc code to perform syntax analysis of boolean, relational and algebraic operators has been implemented successfully.

QUESTION-2

Write a program to construct a symbol table in C.

Code:

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

typedef struct {
    char name[50];
    char datatype[20];
    int offset;
    int size;
    char scope[20];
} SymbolEntry;

typedef struct {
    SymbolEntry entries[100];
    int count;
} SymbolTable;

void initSymbolTable(SymbolTable *table) {
    table->count = 0;
}

void addEntry(SymbolTable *table, const char *name, const char *datatype,
int offset, int size, const char *scope) {
    if (table->count < 100) {
        SymbolEntry *entry = &(table->entries[table->count]);
        strcpy(entry->name, name);
        strcpy(entry->datatype, datatype);
        entry->offset = offset;
        entry->size = size;
        strcpy(entry->scope, scope);
        table->count++;
    } else {
        printf("Symbol table full, cannot add entry.\n");
```

```
    }
}

void displaySymbolTable(const SymbolTable *table) {
    printf("Name\tDataType\tOffset\tSize\tScope\n");
    printf("-----------------------------------------\n");
    for (int i = 0; i < table->count; i++) {
        SymbolEntry entry = table->entries[i];
        printf("%s\t%s\t\t%d\t%d\t%s\n", entry.name, entry.datatype,
entry.offset, entry.size, entry.scope);
    }
}

int main() {
    SymbolTable symbolTable;
    initSymbolTable(&symbolTable);
    addEntry(&symbolTable, "x", "long", 4, sizeof(long), "global");
    addEntry(&symbolTable, "y", "long long", 8, sizeof(long long),
"local");
    displaySymbolTable(&symbolTable);
    return 0;
}
```

Output:

```
prajw@Prajwal_DELL MINGW64 ~/OneDrive/Desktop/NITT/Sem6/CSPC62 - Compilers/Lab-2, 27Feb2024/Q2
$ gcc file.c

prajw@Prajwal_DELL MINGW64 ~/OneDrive/Desktop/NITT/Sem6/CSPC62 - Compilers/Lab-2, 27Feb2024/Q2
$ ./a.exe
Name    DataType        Offset  Size    Scope
-----------------------------------------
x       long            4       4       global
y       long long               8       8       local
```

Result:

C program to illustrate the working of aa symbol table has been implemented successfully.