SET - B
Task -1

// input.txt

input.txt U

SET B > task_1 > input.txt

```
int main()
{
    int num, original, reversed = 0, digit;
    printf("Enter an integer: ");
    scanf("%d", &num);
    original = num;
    do
    {
        digit = num % 10;
        reversed = reversed * 10 + digit;
        num = num / 10;
    } while (num != 0);
    printf("Reversed number: %d\n", reversed);
    if (original == reversed)
    {
        printf("The number is a palindrome.\n");
    }
    else
    {
        printf("The number is not a palindrome.\n");
    }
    return 0;
}
```

// cal.l

SET B > task_1 > cal.l

```
%option noyywrap

%{
#include "cal.tab.h"
#include <string.h>
%}

digit   [0-9]
number  {digit}+
id      [a-zA-Z_][a-zA-Z0-9_]*
ws      [ \t\n]+
string  \"([^\"\n]*)\"

%%

{ws}                {/* skip whitespace */}
{number}            { yylval.num = atoi(yytext); return NUM; }
{string}            { yylval.id = strdup(yytext); return STRING; }

"int"               { return INT; }
"main"              { return MAIN; }
"printf"            { return PRINTF; }
"scanf"             { return SCANF; }
"return"            { return RETURN; }
"if"                { return IF; }
"else"              { return ELSE; }
"do"                { return DO; }
"while"             { return WHILE; }
```

```
"=="                { return EQ; }
"!="                { return NEQ; }
"="                 { return ASSIGN; }
"%"                 { return MOD; }
"/"                 { return DIV; }
"*"                 { return MULT; }
"+"                 { return PLUS; }

"&"                 { return AMP; }
";"                 { return SEMI; }
","                 { return COMMA; }
"("                 { return LPAREN; }
")"                 { return RPAREN; }
"{"                 { return LBRACE; }
"}"                 { return RBRACE; }

{id}                { yylval.id = strdup(yytext); return ID; }

.                   { /* ignore unknown chars */ }

%%
```

//cal.y

cal.y U

SET B > task_1 > cal.y

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

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

%union {
    int num;
    char* id;
}

%token <num> NUM
%token <id> ID
%token INT MAIN PRINTF SCANF RETURN IF ELSE DO WHILE
%token EQ NEQ ASSIGN MOD DIV MULT PLUS
%token LPAREN RPAREN LBRACE RBRACE SEMI COMMA
%token AMP STRING

%start program

%%

program:
    INT MAIN LPAREN RPAREN LBRACE statements RBRACE
    ;

statements:
    statements statement
    | statement
    ;
```

```
statement:
    declaration
    | assignment
    | print_stmt
    | scan_stmt
    | do_while_loop
    | if_else
    | return_stmt
    ;

declaration:
    INT id_list SEMI
    ;

id_list:
    ID
    | ID ASSIGN expression
    | id_list COMMA ID
    | id_list COMMA ID ASSIGN expression
    ;

assignment:
    ID ASSIGN expression SEMI
    ;

print_stmt:
    PRINTF LPAREN STRING RPAREN SEMI
    | PRINTF LPAREN STRING COMMA ID RPAREN SEMI
    ;

scan_stmt:
    SCANF LPAREN STRING COMMA AMP ID RPAREN SEMI
    ;
```

cal.y U

SET B > task_1 > cal.y

```
expression:
    NUM
    | ID
    | expression PLUS expression
    | expression MULT expression
    | expression DIV expression
    | expression MOD expression
    ;

do_while_loop:
    DO LBRACE statements RBRACE WHILE LPAREN condition RPAREN SEMI
    ;

condition:
    expression EQ expression
    | expression NEQ expression
    ;

if_else:
    IF LPAREN condition RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE
    ;

return_stmt:
    RETURN NUM SEMI
    ;

%%

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

int main() {
    yyparse();
    printf("Task-1 Parsing Done.\n");
    return 0;
}
```

//output.txt

output.txt U

SET B > task_1 > output.txt

```
1    Task-1 Parsing Done.
2
```

```
ACER@mdyasin MINGW64 /d/course-work/CSE_MU/CSE415-416_Compiler-Construction/Lab 09 - Test/SET B/task_1 (main)
$ make
bison -d cal.y
cal.y: conflicts: 16 shift/reduce
flex cal.l
gcc cal.tab.c lex.yy.c
./a.exe <input.txt> output.txt
```

Task -2

// input.txt

input.txt U

SET B > task_2 > input.txt

```c
int main()
{
    char op;
    double a, b, result;
    printf("Enter operation (+, -, *, /, ^): ");
    scanf(" %c", &op);
    printf("Enter two numbers: ");
    scanf("%lf %lf", &a, &b);
    switch (op)
    {
    case '+':
        result = a + b;
        printf("Result: %.2lf\n", result);
        break;
    case '-':
        result = a - b;
        printf("Result: %.2lf\n", result);
        break;
    case '*':
        result = a * b;
        printf("Result: %.2lf\n", result);
        break;
    case '/':
        if (b != 0)
            printf("Result: %.2lf\n", a / b);
        else
            printf("Error: Division by zero!\n");
        break;
    case '^':
        result = pow(a, b);
        printf("Result: %.2lf\n", result);
        break;
    default:
        printf("Invalid operator!\n");
    }
    return 0;
}
```

// cal.l

cal.l U

SET B > task_2 > cal.l

```
%{
#include "cal.tab.h"
#include <stdlib.h>
#include <string.h>
%}

%%

"int"           { return INT; }
"char"          { return CHAR; }
"double"        { return DOUBLE; }
"main"          { return MAIN; }
"return"        { return RETURN; }
"scanf"         { return SCANF; }
"printf"        { return PRINTF; }
"switch"        { return SWITCH; }
"case"          { return CASE; }
"default"       { return DEFAULT; }
"break"         { return BREAK; }
"if"            { return IF; }
"else"          { return ELSE; }

"!="            { return NEQ; }
"^"             { return POWER; }
"&"             { return AMP; }

":"             { return COLON; }
";"             { return SEMICOLON; }
"="             { return ASSIGN; }
","             { return COMMA; }
"("             { return LPAREN; }
")"             { return RPAREN; }
"{"             { return LBRACE; }
"}"             { return RBRACE; }

[0-9]+\.[0-9]+  { yylval.dval = atof(yytext); return NUMBER; }
[0-9]+          { yylval.dval = atof(yytext); return NUMBER; }
"\'."           { yylval.cval = yytext[1]; return CHAR_CONST; }

[a-zA-Z_][a-zA-Z0-9_]* { yylval.id = strdup(yytext); return ID; }

\".*\"          { return STRING; }

"+"             { return PLUS; }
"-"             { return MINUS; }
"*"             { return MUL; }
"/"             { return DIV; }

[ \t\r\n]+      { /* skip whitespace */ }

.               { return yytext[0]; }

%%

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

// cal.y

cal.y U

SET B > task_2 > cal.y

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

double a, b, result;
char op;

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

%union {
    int ival;
    char cval;
    char *id;
    double dval;
}

%token <ival> INT CHAR DOUBLE RETURN MAIN
%token <id> ID
%token <cval> CHAR_CONST
%token <dval> NUMBER
%token <id> STRING

%token IF ELSE SCANF PRINTF
%token SWITCH CASE DEFAULT BREAK
%token ASSIGN PLUS MINUS MUL DIV POWER
%token NEQ AMP
%token LPAREN RPAREN LBRACE RBRACE SEMICOLON COLON COMMA

%start program

%%

program:
      INT MAIN LPAREN RPAREN LBRACE declarations statements RBRACE
    ;

declarations:
      declarations declaration
    | declaration
    ;

declaration:
```

Ln 21, Col 15 Spaces: 2 UTF-8 CRLF {} Plain Text

cal.y U

SET B > task_2 > cal.y

```

declarations:
      declarations declaration
    | declaration
    ;

declaration:
      CHAR ID SEMICOLON
    | DOUBLE id_list SEMICOLON
    ;

id_list:
      ID
    | id_list COMMA ID
    ;

statements:
      statements statement
    | statement
    ;

statement:
      scan_stmt
    | print_stmt
    | assign_stmt
    | switch_stmt
    | return_stmt
    | if_stmt
    ;

scan_stmt:
      SCANF LPAREN STRING COMMA AMP ID RPAREN SEMICOLON
    ;

print_stmt:
      PRINTF LPAREN STRING RPAREN SEMICOLON
    | PRINTF LPAREN STRING COMMA ID RPAREN SEMICOLON
    ;

assign_stmt:
      ID ASSIGN expr SEMICOLON
    ;

```

cal.y U

SET B > task_2 > cal.y

```
print_stmt:

assign_stmt:
    ID ASSIGN expr SEMICOLON
    ;

switch_stmt:
    SWITCH LPAREN ID RPAREN LBRACE case_list default_case RBRACE
    ;

case_list:
    case_list case_stmt
    | case_stmt
    ;

case_stmt:
    CASE CHAR_CONST COLON statements BREAK SEMICOLON
    ;

default_case:
    DEFAULT COLON statements
    ;

return_stmt:
    RETURN NUMBER SEMICOLON
    ;

if_stmt:
    IF LPAREN expr NEQ NUMBER RPAREN statement
    | IF LPAREN expr NEQ NUMBER RPAREN statement ELSE statement
    ;
```

```
expr:
    expr PLUS expr
    | expr MINUS expr
    | expr MUL expr
    | expr DIV expr
    | POWER LPAREN expr COMMA expr RPAREN
    | ID
    | NUMBER
    ;

%%

int main() {
    yyparse();
    printf("Task-2 Parsing Done.\n");
    return 0;
}

int yyerror(char *s) {
    fprintf(stderr, "Error: %s\n", s);
    return 0;
}
```

// output

output.txt U

SET B > task_2 > output.txt

```
1   Task-2 Parsing Done.
2
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
$ make
bison -d cal.y
cal.y: conflicts: 17 shift/reduce
flex cal.l
gcc cal.tab.c lex.yy.c -o a -lm
./a < input.txt > output.txt
Error: syntax error
```