# CD LAB 5

NAME: Janvii RV

SRN: PES2UG22CS232

DATE: 20/03/2025

Code:

lexer.l :

```
%{
   #define YYSTYPE char*
   #include <unistd.h>
   #include "parser.tab.h"
   #include <stdio.h>
   extern void yyerror(const char *); // declare the error handling function
%}

/* Regular definitions */
digit     [0-9]
letter    [a-zA-Z]
id        {letter}({letter}|{digit})*
digits    {digit}+
opFraction       (\.{digits})?
opExponent       ([Ee][+-]?{digits})?
number           {digits}{opFraction}{opExponent}
%option yylineno

%%
\/\/(.*) ; // ignore comments
[\t\n] ; // ignore whitespaces
```

```
"("             {return *yytext;}
")"             {return *yytext;}
"."             {return *yytext;}
","             {return *yytext;}
"*"             {return *yytext;}
"+"             {return *yytext;}
";"             {return *yytext;}
"-"             {return *yytext;}
"/"             {return *yytext;}
"="             {return *yytext;}
">"             {return *yytext;}
"<"             {return *yytext;}
{number}        {
                        yylval = strdup(yytext);  //stores the value of the number to be used
later for symbol table insertion
                        return T_NUM;
                }
{id}            {
                                        yylval = strdup(yytext); //stores the identifier to be
used later for symbol table insertion
                                        return T_ID;
                                }
.               {} // anything else => ignore
%%
```

parser.y

```
%{
        #include "quad_generation.c"
        #include <stdio.h>
```

```
            #include <stdlib.h>

            #include <string.h>


            #define YYSTYPE char*


            void yyerror(char* s);
                        // error handling function

            int yylex();
                                    // declare the function performing lexical analysis

            extern int yylineno;
                        // track the line number


            int yywrap() {

    return 1;  // Signals end of input

}




            FILE* icg_quad_file;

            int temp_no = 1;

%}



%token T_ID T_NUM


/* specify start symbol */

%start START




%%
```

```
START : ASSGN{
                                        printf("Valid syntax\n");
                                        YYACCEPT;
                                // If program fits the grammar, syntax is valid
                                }

/* Grammar for assignment */
ASSGN : T_ID '=' E      {       //call quad_code_gen with appropriate parameters
                                                quad_code_gen($1, $3, "=", "");
                                        }
        ;

/* Expression Grammar */
E : E '+' T     {       //create a new temporary and call quad_code_gen with appropriate
parameters
                                        $$= new_temp();
                                        char* op =strdup("+");
                                        quad_code_gen($$,$1,op,$3);
                                }
        | E '-' T       {       //create a new temporary and call quad_code_gen with
appropriate parameters
                                        $$= new_temp();
                                        char* op =strdup("-");
                                        quad_code_gen($$,$1,op,$3);
                                }
        | T
        ;
```

```
T : T '*' F         {          //create a new temporary and call quad_code_gen with appropriate
parameters
                                                $$= new_temp();
                                                char* op =strdup("*");
                                                quad_code_gen($$,$1,op,$3);
                                       }
          | T '/' F          {          //create a new temporary and call quad_code_gen with
appropriate parameters
                                                $$= new_temp();
                                                char* op =strdup("/");
                                                quad_code_gen($$,$1,op,$3);
                                       }
          | F
          ;

F : '(' E ')'        {          $$= strdup($2);        }
          | T_ID           {          $$= strdup($1);          }
          | T_NUM        {          $$= strdup($1);          }
          ;

%%


/* error handling function */
void yyerror(char* s)
{
          printf("Error :%s at %d \n",s,yylineno);
}
```

```
/* main function - calls the yyparse() function which will in turn drive yylex() as well */

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

{

        icg_quad_file = fopen("icg_quad.txt","w");

        yyparse();

        fclose(icg_quad_file);

        return 0;

}
```

quad_generation.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "quad_generation.h"

void quad_code_gen(char* a, char* b, char* op, char* c)
{
    //use fprintf to output the quadruple code to icg_quad_file
    printf("%s, %s, %s, %s\n", op, b, c, a);
    fprintf(icg_quad_file, "%s %s %s %s\n", op, b, c, a);
}

char* new_temp()    //returns a pointer to a new temporary
{
    char* temp = (char*)malloc(sizeof(char)*4);
    sprintf(temp, "t%d", temp_no);
    ++temp_no;
    return temp;
}
```

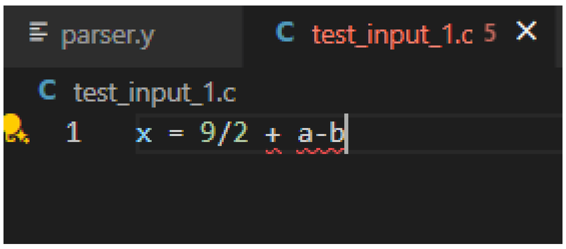
quad_generation.h

```c
extern FILE* icg_quad_file;     //pointer to the output file
extern int temp_no;             //variable to keep track of current temporary
count

void quad_code_gen(char* a, char* b, char* op, char* c);
char* new_temp();
```

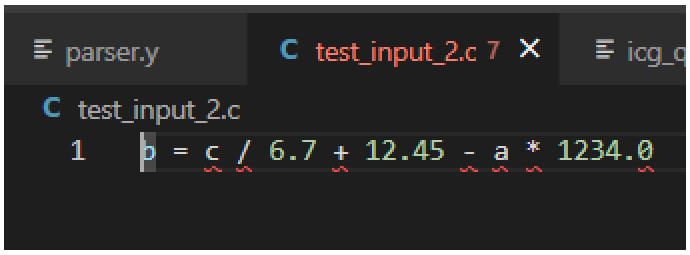
Input:

1:



```
x = 9/2 + a-b
```

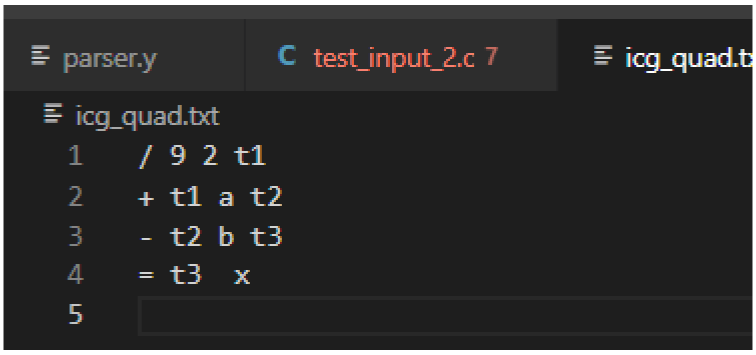
2:



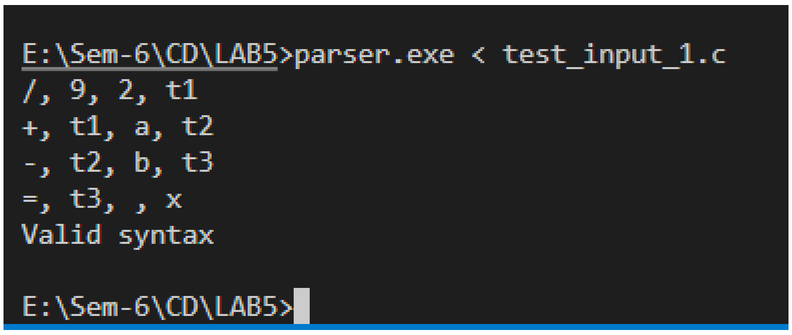
```
b = c / 6.7 + 12.45 - a * 1234.0
```

Output Screenshot:

1:



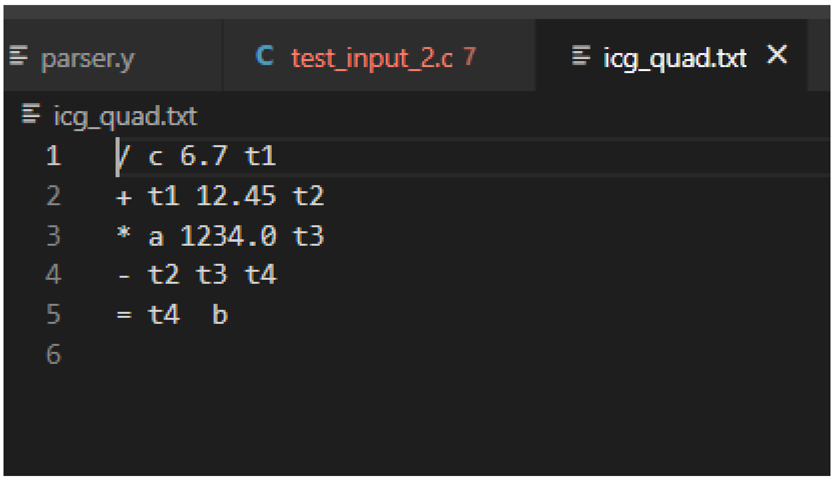
```
/ 9 2 t1
+ t1 a t2
- t2 b t3
= t3  x
```



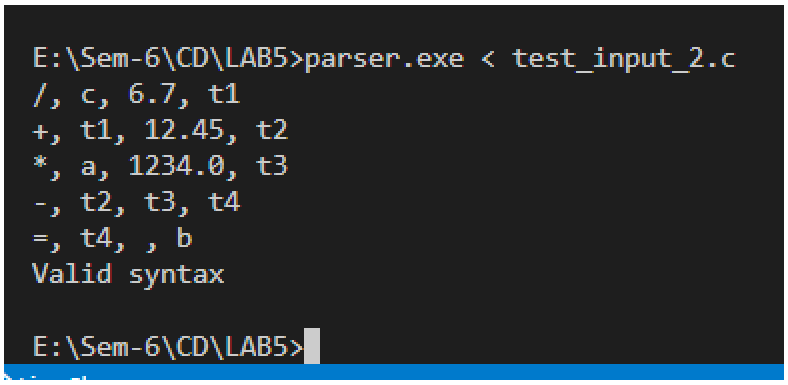
```
E:\Sem-6\CD\LAB5>parser.exe < test_input_1.c
/, 9, 2, t1
+, t1, a, t2
-, t2, b, t3
=, t3, , x
Valid syntax

E:\Sem-6\CD\LAB5>
```

2:



```
/ c 6.7 t1
+ t1 12.45 t2
* a 1234.0 t3
- t2 t3 t4
= t4  b
```



```
E:\Sem-6\CD\LAB5>parser.exe < test_input_2.c
/, c, 6.7, t1
+, t1, 12.45, t2
*, a, 1234.0, t3
-, t2, t3, t4
=, t4, , b
Valid syntax

E:\Sem-6\CD\LAB5>
```