CD ASSIGNMENT-2

Name:Kushagra Agarwal
SRN:PES2UG22CS275
SEC :E

AST:

```
%{
   #define YYSTYPE char*
   #include "parser.tab.h"
   #include <stdio.h>
   #include <string.h>
   extern void yyerror(const char *);
%}

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

%option yylineno

%%

\/\/.*                ;
[\t\n ]               ;

"if"                  { return IF; }
"else"                  { return ELSE; }
"do"                    { return DO; }
"while"                  { return WHILE; }

"<="                    { yylval = strdup("<="); return RELOP; }
">="                    { yylval = strdup(">="); return RELOP; }
"=="                    { yylval = strdup("=="); return RELOP; }
"!="                   { yylval = strdup("!="); return RELOP; }
"<"                    { yylval = strdup("<"); return RELOP; }
">"                    { yylval = strdup(">"); return RELOP; }

"("                   { return *yytext; }
")"                   { return *yytext; }
"{"                   { return *yytext; }
"}"                   { return *yytext; }
```

```
"="                     { return *yytext; }
";"                     { return *yytext; }
"+"                     { return *yytext; }
"-"                     { return *yytext; }
"*"                     { return *yytext; }
"/"                     { return *yytext; }

{number}                {
                          yylval = strdup(yytext);
                          return T_NUM;
                        }

{id}                    {
                          yylval = strdup(yytext);
                          return T_ID;
                        }

.                       ;  // ignore other characters
%%
```

Parser.y:

```
%{
   #include "abstract_syntax_tree.c"
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   void yyerror(char* s);
   int yylex();
   extern int yylineno;
%}

%union {
   char* text;
   expression_node* exp_node;
}

%token <text> T_ID T_NUM IF ELSE DO WHILE RELOP
%type <exp_node> E T F START ASSGN S C SEQ

%start START

%%

START : SEQ {
   display_exp_tree($1);
```

```
    printf("\nValid syntax\n");
    YYACCEPT;
};

SEQ : S SEQ { $$ = init_exp_node("seq", $1, $2); }
    | S     { $$ = $1; }
;

S : IF '(' C ')' '{' SEQ '}' ELSE '{' SEQ '}' {
        $$ = init_exp_node(strdup("if-else"), $3, init_exp_node("", $6, $10));
    }
  | IF '(' C ')' '{' SEQ '}' {
        $$ = init_exp_node(strdup("if"), $3, $6);
    }
  | DO '{' SEQ '}' WHILE '(' C ')' ';' {
        $$ = init_exp_node(strdup("do-while"), $7, $3);
    }
  | ASSGN { $$ = $1; }
;

C : F RELOP F {
    $$ = init_exp_node(strdup($2), $1, $3);
}
;

ASSGN : T_ID '=' E ';' {
    $$ = init_exp_node(strdup("="), init_exp_node(strdup($1), NULL, NULL), $3);
}
;

E : E '+' T { $$ = init_exp_node(strdup("+"), $1, $3); }
  | E '-' T { $$ = init_exp_node(strdup("-"), $1, $3); }
  | T       { $$ = $1; }
;

T : T '*' F { $$ = init_exp_node(strdup("*"), $1, $3); }
  | T '/' F { $$ = init_exp_node(strdup("/"), $1, $3); }
  | F       { $$ = $1; }
;

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

```
%%

void yyerror(char* s)
{
   printf("Error: %s at line %d\n", s, yylineno);
}

int yywrap() {
   return 1;
}

int main(int argc, char* argv[])
{
   printf("Preorder:\n");
   yyparse();
   return 0;
}
```

abstract_syntax_tree.c

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

expression_node* init_exp_node(char* val, expression_node* left,
expression_node* right)
{
   // function to allocate memory for an AST node and set the left and right children
of the nodes
   expression_node* node = (expression_node*)malloc(sizeof(expression_node));
   node->left = left;
   node->right = right;
   node->val = val;
   return node; // Return the allocated node
}

void helper(expression_node* exp_node)
{
   if (exp_node == NULL)
               return;
   printf("%s ", exp_node->val);
   helper(exp_node->left);
   helper(exp_node->right);
}

void display_exp_tree(expression_node* exp_node)
{
   // traversing the AST in postorder and displaying the nodes
   helper(exp_node);
   printf("\n");
}
```

abstract_syntax_tree.h

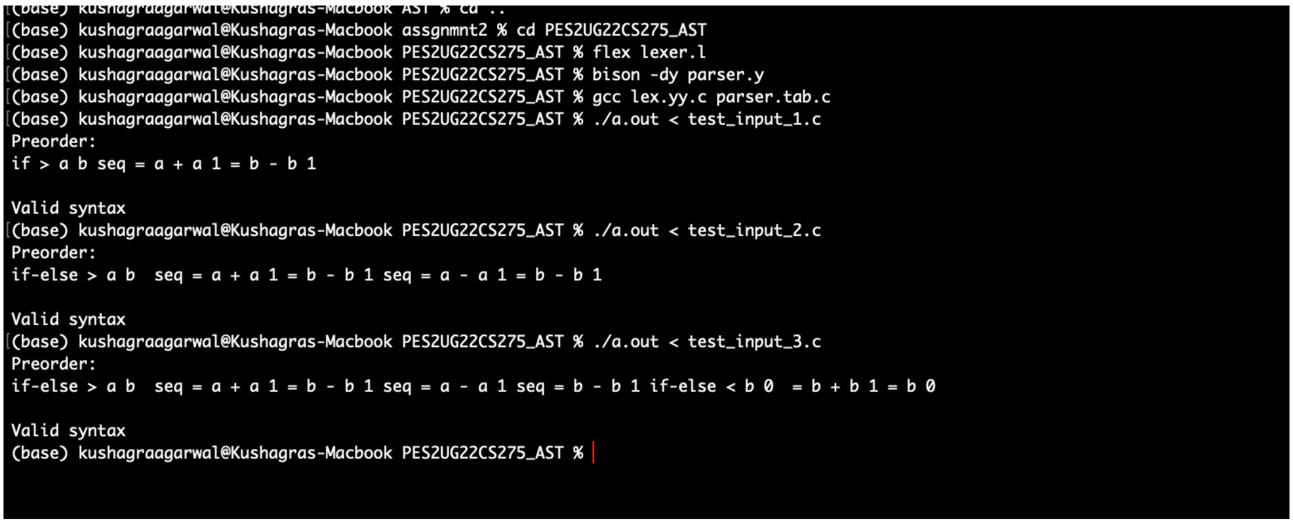
```
#ifndef ABSTRACT_SYNTAX_TREE_H
#define ABSTRACT_SYNTAX_TREE_H

typedef struct expression_node
{
        struct expression_node* left;
        char* val;
        struct expression_node* right;
}expression_node;

expression_node* init_exp_node(char* val, expression_node* left,
expression_node* right);
void display_exp_tree(expression_node* exp_node);

#endif // ABSTRACT_SYNTAX_TREE_H%
```

Output:



```
(base) kushagraagarwal@Kushagras-Macbook AST % cd ..
(base) kushagraagarwal@Kushagras-Macbook assgnmnt2 % cd PES2UG22CS275_AST
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % flex lexer.l
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % bison -dy parser.y
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % gcc lex.yy.c parser.tab.c
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % ./a.out < test_input_1.c
Preorder:
if > a b seq = a + a 1 = b - b 1

Valid syntax
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % ./a.out < test_input_2.c
Preorder:
if-else > a b  seq = a + a 1 = b - b 1 seq = a - a 1 = b - b 1

Valid syntax
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST % ./a.out < test_input_3.c
Preorder:
if-else > a b  seq = a + a 1 = b - b 1 seq = a - a 1 seq = b - b 1 if-else < b 0  = b + b 1 = b 0

Valid syntax
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_AST %
```

ICG:

lexer.l

```
%{
   #define YYSTYPE char*
   #include <unistd.h>
   #include "parser.tab.h"
   #include <stdio.h>
   #include <string.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 LTEQ;}
">="            {return GTEQ;}
"=="            {return EQQ;}
"!="            {return NEQ;}
"{"             {return OC;}
"}"             {return CC;}
"("             {return *yytext;}
")"             {return *yytext;}
"."             {return *yytext;}
","             {return *yytext;}
"*"             {return *yytext;}
"+"             {return *yytext;}
";"             {return *yytext;}
"-"             {return *yytext;}
"/"             {return *yytext;}
"="             {return *yytext;}
">"             {return GT;}
"<"             {return LT;}
{number}     {
```

```
                        yylval = strdup(yytext);  //stores the value of the number to be
used later for symbol table insertion
                        return T_NUM;
                }
"if"            {return T_IF;}
"else"          {return T_ELSE;}
{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

        FILE* icg_quad_file;
        int temp_no = 1;
        int label_no=1;
%}


%token T_ID T_NUM T_IF T_ELSE GTEQ LTEQ EQQ NEQ GT LT OC CC

/* specify start symbol */
%start START

%nonassoc T_IF
%nonassoc T_ELSE
```

```
%%
START : S {
                printf("----------------------------------------------------\n");
                printf("Valid syntax\n");
                YYACCEPT;
        };



/* Grammar for assignment */
ASSGN : T_ID '=' E        {

                                quad_code_gen($1, $3, "=", " ");

                        }
        ;

/* Expression Grammar */
E : E '+' T    {

                        $$ = new_temp();
                        quad_code_gen($$, $1, "+", $3);
                }
        | E '-' T       {

                                $$ = new_temp();
                                quad_code_gen($$, $1, "-", $3);
                        }
        | T
        ;


T : T '*' F     {
                        $$ = new_temp();
                        quad_code_gen($$, $1, "*", $3);
                }
        | T '/' F       {


                                $$ = new_temp();
                                quad_code_gen($$, $1, "/", $3);
                        }
        | F
        ;
```

```
F : '(' E ')'    {

                        $$=strdup($2);
                 }
        l T_ID                {
                                      $$=strdup($1);
                              }
        l T_NUM     {
                              $$=strdup($1);
                    }
        ;

S : T_IF '('C')' OC S CC {quad_code_gen($3,"","Label","");} S
      l T_IF '('C')' OC S CC {
                        $2 = new_label();
                              quad_code_gen($2,"","goto","");
                              quad_code_gen($3,"","Label","");} T_ELSE OC S CC
{quad_code_gen($2,"","Label","");}S
      l ASSGN ';' S
      l'{'S'}'
      l
      ;

C : E rel E  {$$ = new_temp();
                 quad_code_gen($$, $1, $2, $3);
                 $1 = new_label();
                 quad_code_gen($1,$$,"if","");
                 $$ = new_label();
                 quad_code_gen($$,"","goto","");
                 quad_code_gen($1,"","Label","");

                 };

rel :  GT {strcpy($$,">");}
    l LT {strcpy($$,"<");}
    l LTEQ {strcpy($$,"<=");}
    l GTEQ {strcpy($$,">=");}
    l EQQ {strcpy($$,"==");}
    l NEQ {strcpy($$,"!=");}
    ;

%%


/* error handling function */
```

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

int yywrap() {
   return 1;
}

/* main function - calls the yyparse() function which will in turn drive yylex() as well
*/
int main(int argc, char* argv[])
{

        printf("Generated Intermediate Code \n");
        printf("---------------------------------------------------\n");
        printf("| %-10s | %-10s | %-10s | %-10s |\n", "op", "arg1", "arg2", "result");
        printf("---------------------------------------------------\n");
        yyparse();
        return 0;
}%
```

<u>quad_generation.c</u>

```
#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)
{
        printf("| %-10s | %-10s | %-10s | %-10s |\n",op,b,c,a);
}

char* new_temp()
{
                char* tempNew = (char*)malloc(sizeof(char)*4);
                sprintf(tempNew,"t%d",temp_no);
                temp_no++;
                return tempNew;
}

char* new_label()
{
                char* label = (char*)malloc(sizeof(char)*4);
```

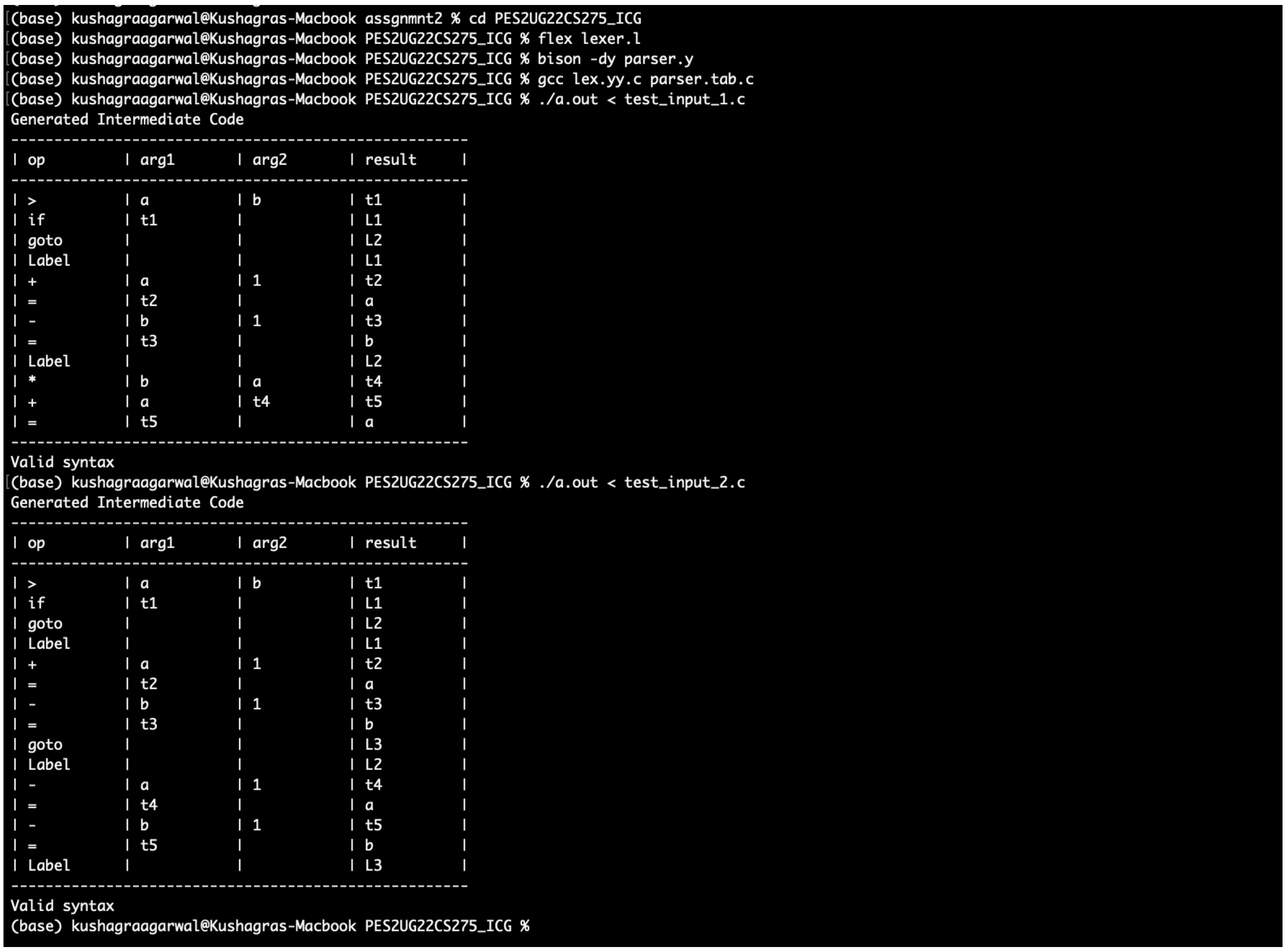
```
            sprintf(label,"L%d",label_no);
            label_no++;
            return lab
}
```

quad_generation.h

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

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

Output:



```
(base) kushagraagarwal@Kushagras-Macbook assgnmnt2 % cd PES2UG22CS275_ICG
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG % flex lexer.l
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG % bison -dy parser.y
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG % gcc lex.yy.c parser.tab.c
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG % ./a.out < test_input_1.c
Generated Intermediate Code
-----------------------------------------------------
| op        | arg1      | arg2      | result    |
-----------------------------------------------------
| >         | a         | b         | t1        |
| if        | t1        |           | L1        |
| goto      |           |           | L2        |
| Label     |           |           | L1        |
| +         | a         | 1         | t2        |
| =         | t2        |           | a         |
| -         | b         | 1         | t3        |
| =         | t3        |           | b         |
| Label     |           |           | L2        |
| *         | b         | a         | t4        |
| +         | a         | t4        | t5        |
| =         | t5        |           | a         |
-----------------------------------------------------
Valid syntax
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG % ./a.out < test_input_2.c
Generated Intermediate Code
-----------------------------------------------------
| op        | arg1      | arg2      | result    |
-----------------------------------------------------
| >         | a         | b         | t1        |
| if        | t1        |           | L1        |
| goto      |           |           | L2        |
| Label     |           |           | L1        |
| +         | a         | 1         | t2        |
| =         | t2        |           | a         |
| -         | b         | 1         | t3        |
| =         | t3        |           | b         |
| goto      |           |           | L3        |
| Label     |           |           | L2        |
| -         | a         | 1         | t4        |
| =         | t4        |           | a         |
| -         | b         | 1         | t5        |
| =         | t5        |           | b         |
| Label     |           |           | L3        |
-----------------------------------------------------
Valid syntax
(base) kushagraagarwal@Kushagras-Macbook PES2UG22CS275_ICG %
```