SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Symbiosis International (Deemed University)

(Established under section 3 of the UGC Act, 1956)

Re-accredited by NAAC with ‘A’ grade (3.58/4) | Awarded Category – I by UGC

Founder: Prof. Dr. S. B. Mujumdar, M. Sc., Ph. D. (Awarded Padma Bhushan and Padma Shri by President of India)

Assignment No. 10

Subject:	Compiler Construction Lab
Name of Student	**Onkar Mendhapurkar**
PRN No.	**22070122135**
Branch	CSE B2, Batch (2022-26)
Academic Year & Semester	2022-26
Date of Performance	09/10/2025
Title of Assignment:	Parser for Intermediate code (IC) generator for arithmetic expression.
Practice Questions	1. YACC program for Intermediate code (IC) generator for arithmetic expression. 2. YACC program for IC generation for the expression involving parenthesis. **PostLab Question** 3. YACC program for IC generation for the expression involving programming constructs.

Source Code	**1.**
	ic.l (Lex)
	`%{`
	`#include "ic.tab.h"`
	`%}`
	
	`%%`
	
	`[0-9]+                  { yylval.ival = atoi(yytext); return NUMBER; }`
	`[a-zA-Z_][a-zA-Z0-9_]*  { yylval.sval = strdup(yytext); return ID; }`
	`"+"                     { return '+'; }`
	`"-"                     { return '-'; }`

```
"*"                    { return '*'; }
"/"                   { return '/'; }
"("                   { return '('; }
")"                   { return ')' ;}
";"                   { return ';'; }
"\n"                   { return '\n'; }
[ \t]+                 { /* ignore whitespace */ }
.                     { printf("Unknown character: %s\n", yytext); }

%%

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

ic.y

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

int tempCount = 0;
char tempName[10];

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

char* newTemp() {
    sprintf(tempName, "t%d", tempCount++);
    return strdup(tempName);
}
%}

%union {
    char* sval;
    int ival;
}

%token <sval> ID
%token <ival> NUMBER

%left '+' '-'
%left '*' '/'

%type <sval> expr
```

	%%
	
	program:
	    expr '\n' { printf("Final Result: %s\n", $1); }
	  | expr ';' { printf("Final Result: %s\n", $1); }
	;
	
	expr:
	    expr '+' expr {
	      char* t = newTemp();
	      printf("%s = %s + %s\n", t, $1, $3);
	      $$ = t;
	    }
	  \| expr '-' expr {
	      char* t = newTemp();
	      printf("%s = %s - %s\n", t, $1, $3);
	      $$ = t;
	    }
	  | expr '*' expr {
	      char* t = newTemp();
	      printf("%s = %s * %s\n", t, $1, $3);
	      $$ = t;
	    }
	  \| expr '/' expr {
	      char* t = newTemp();
	      printf("%s = %s / %s\n", t, $1, $3);
	      $$ = t;
	    }
	  | '(' expr ')' { $$ = $2; }
	  \| ID {
	      char* t = newTemp();
	      printf("%s = %s\n", t, $1);
	      $$ = t;
	    }
	  | NUMBER {
	      char* t = newTemp();
	      printf("%s = %d\n", t, $1);
	      $$ = t;
	    }
	;
	
	%%

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

int main() {
    printf("Enter arithmetic expression:\n");
    while(!feof(stdin)) yyparse();
    return 0;
}
```

2.
LEX File (ic_paren.l)

```
%{
#include "ic_paren.tab.h"
%}

%%

[0-9]+                  { yylval.ival = atoi(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { yylval.sval = strdup(yytext); return ID; }
"+"                     { return '+'; }
"-"                     { return '-'; }
"*"                     { return '*'; }
"/"                     { return '/'; }
"("                     { return '('; }
")"                     { return ')' ;}
";"                     { return ';'; }
"\n"                    { return '\n'; }
[ \t]+                  { /* ignore whitespace */ }
.                       { printf("Unknown character: %s\n", yytext); }

%%

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

IC Generator with Parentheses (ic_paren.y)

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

int tempCount = 0;
char tempName[10];
```

```
int yylex();
void yyerror(const char *s);

// Function to create new temporary variable
char* newTemp() {
    sprintf(tempName, "t%d", tempCount++);
    return strdup(tempName);
}
%}

%union {
    char* sval;
    int ival;
}

%token <sval> ID
%token <ival> NUMBER

%left '+' '-'
%left '*' '/'
%type <sval> expr

%%

program:
      expr '\n' { printf("Final Result: %s\n", $1); }
    | expr ';' { printf("Final Result: %s\n", $1); }
;

expr:
      expr '+' expr {
          char* t = newTemp();
          printf("%s = %s + %s\n", t, $1, $3);
          $$ = t;
      }
    | expr '-' expr {
          char* t = newTemp();
          printf("%s = %s - %s\n", t, $1, $3);
          $$ = t;
      }
    | expr '*' expr {
          char* t = newTemp();
          printf("%s = %s * %s\n", t, $1, $3);
          $$ = t;
```

	```
	 }
	   | expr '/' expr {
	        char* t = newTemp();
	        printf("%s = %s / %s\n", t, $1, $3);
	        $$ = t;
	     }
	   | '(' expr ')' { $$ = $2; }   // Handle parentheses
	   | ID {
	        char* t = newTemp();
	        printf("%s = %s\n", t, $1);
	        $$ = t;
	     }
	   | NUMBER {
	        char* t = newTemp();
	        printf("%s = %d\n", t, $1);
	        $$ = t;
	     }
	;

	%%

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

	int main() {
	   printf("Enter arithmetic expression (with parentheses allowed):\n");
	   while(!feof(stdin)) yyparse();
	   return 0;
	}
	```
	
	3. PostLab Experiment
	ic_constructs.l – LEX File
	```
	 %{
	#include "ic_constructs.tab.h"
	%}

	%%

	"if"                   { return IF; }
	"while"                  { return WHILE; }
	[0-9]+                  { yylval.ival = atoi(yytext); return NUMBER; }
	[a-zA-Z_][a-zA-Z0-9_]*  { yylval.sval = strdup(yytext); return ID; }
	```

```
"="                 { return '='; }
";"                 { return ';'; }
"\("                { return '('; }
"\)"                { return ')'; }
"\{"                { return '{'; }
"\}"                { return '}'; }

"+"                 { return '+'; }
"-"                 { return '-'; }
"*"                 { return '*'; }
"/"                 { return '/'; }

[ \t\n]+            { /* ignore whitespace */ }

.                   { printf("Unknown character: %s\n", yytext); }

%%

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

ic_constructs.y – YACC File

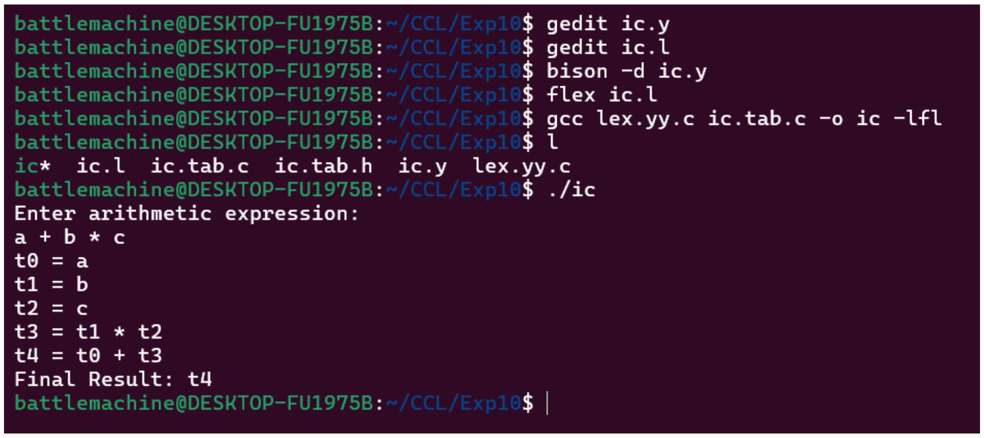
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int tempCount = 0;
int labelCount = 0;

char tempName[10];
char labelName[10];

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

// Generate new temporary variable
char* newTemp() {
    sprintf(tempName, "t%d", tempCount++);
    return strdup(tempName);
}

// Generate new label
```

```
char* newLabel() {
    sprintf(labelName, "L%d", labelCount++);
    return strdup(labelName);
}
%}

%union {
    char* sval;
    int ival;
}

%token <sval> ID
%token <ival> NUMBER
%token IF WHILE

%left '+' '-'
%left '*' '/'
%type <sval> expr stmt program

%%

program:
      /* empty */
    | program stmt
;

stmt:
      ID '=' expr ';' {
          printf("%s = %s\n", $1, $3);
      }
    | IF '(' expr ')' '{' program '}' {
          char* L1 = newLabel();
          printf("if %s == 0 goto %s\n", $3, L1);
          // Statements inside IF already printed
          printf("%s:\n", L1);
      }
    | WHILE '(' expr ')' '{' program '}' {
          char* L1 = newLabel();
          char* L2 = newLabel();
          printf("%s:\n", L1);
          printf("if %s == 0 goto %s\n", $3, L2);
          // Statements inside WHILE already printed
          printf("goto %s\n", L1);
          printf("%s:\n", L2);
    }
  | expr ';' { /* just evaluate expression */ }
;

expr:
    expr '+' expr {
        char* t = newTemp();
        printf("%s = %s + %s\n", t, $1, $3);
        $$ = t;
    }
  | expr '-' expr {
        char* t = newTemp();
        printf("%s = %s - %s\n", t, $1, $3);
        $$ = t;
    }
  | expr '*' expr {
        char* t = newTemp();
        printf("%s = %s * %s\n", t, $1, $3);
        $$ = t;
    }
  | expr '/' expr {
        char* t = newTemp();
        printf("%s = %s / %s\n", t, $1, $3);
        $$ = t;
    }
  | '(' expr ')' { $$ = $2; }
  | ID {
        char* t = newTemp();
        printf("%s = %s\n", t, $1);
        $$ = t;
    }
  | NUMBER {
        char* t = newTemp();
        printf("%s = %d\n", t, $1);
        $$ = t;
    }
;

%%

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

```
int main() {
    printf("Enter statements (assignments, if, while):\n");
    while(!feof(stdin)) yyparse();
    return 0;
}
```

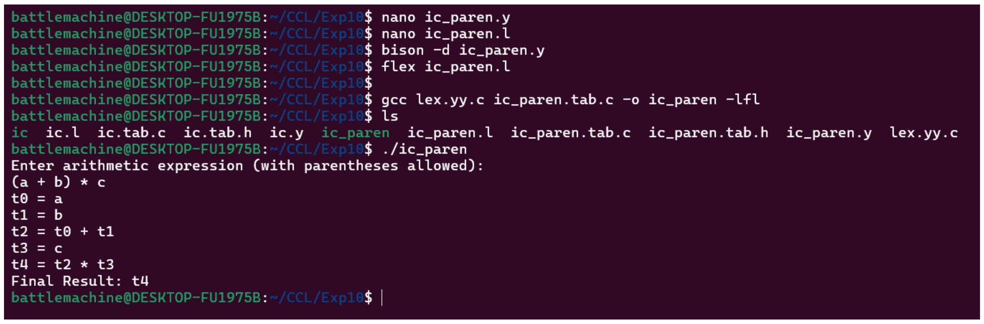
Output Screenshot

1.



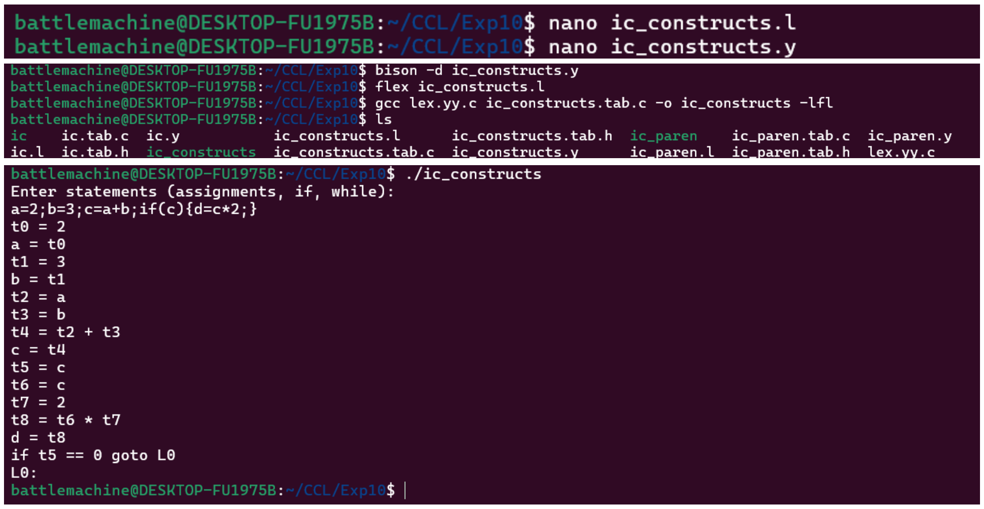
```
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ gedit ic.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ gedit ic.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ bison -d ic.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ flex ic.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ gcc lex.yy.c ic.tab.c -o ic -lfl
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ l
ic*  ic.l  ic.tab.c  ic.tab.h  ic.y  lex.yy.c
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ ./ic
Enter arithmetic expression:
a + b * c
t0 = a
t1 = b
t2 = c
t3 = t1 * t2
t4 = t0 + t3
Final Result: t4
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ |
```

2.



```
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ nano ic_paren.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ nano ic_paren.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ bison -d ic_paren.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ flex ic_paren.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ gcc lex.yy.c ic_paren.tab.c -o ic_paren -lfl
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ ls
ic  ic.l  ic.tab.c  ic.tab.h  ic.y  ic_paren  ic_paren.l  ic_paren.tab.c  ic_paren.tab.h  ic_paren.y  lex.yy.c
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ ./ic_paren
Enter arithmetic expression (with parentheses allowed):
(a + b) * c
t0 = a
t1 = b
t2 = t0 + t1
t3 = c
t4 = t2 * t3
Final Result: t4
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ |
```

3. PostLab Experiment



```
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ nano ic_constructs.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ nano ic_constructs.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ bison -d ic_constructs.y
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ flex ic_constructs.l
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ gcc lex.yy.c ic_constructs.tab.c -o ic_constructs -lfl
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ ls
ic    ic.tab.c  ic.y             ic_constructs.l      ic_constructs.tab.h  ic_paren    ic_paren.tab.c  ic_paren.y
ic.l  ic.tab.h  ic_constructs    ic_constructs.tab.c  ic_constructs.y      ic_paren.l  ic_paren.tab.h  lex.yy.c
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ ./ic_constructs
Enter statements (assignments, if, while):
a=2;b=3;c=a+b;if(c){d=c*2;}
t0 = 2
a = t0
t1 = 3
b = t1
t2 = a
t3 = b
t4 = t2 + t3
c = t4
t5 = c
t6 = c
t7 = 2
t8 = t6 * t7
d = t8
if t5 == 0 goto L0
L0:
battlemachine@DESKTOP-FU1975B:~/CCL/Exp10$ |
```

Conclusion	These experiments demonstrate the use of YACC and LEX to parse, evaluate, and generate intermediate code for arithmetic expressions and programming constructs, highlighting syntax checking, expression evaluation, and three-address code generation in a single integrated workflow.