Cd1a.l – Lex Program to Recognize Digits, Characters, and Unknown Tokens

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

%%
[0-9]+        { printf("Token: %s --> DIGIT\n", yytext); }
[a-zA-Z]+     { printf("Token: %s --> CHARACTER\n", yytext); }
[ \t\n]+      { /* ignore whitespace */ }
.             { printf("Token: %s --> UNKNOWN\n", yytext); }
%%

int main() {
   printf("Enter input:\n");
   yylex();
   return 0;
}
```

Execution commands –

```
flex cd1a.l
gcc lex.yy.c -o cd1a -lfl
./cd1a
```

Cd1b.l – Lex Program for Part-of-Speech (POS) Tagging of English Words

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

char *nouns[] = {"dog", "cat", "car", "tree", "book", "city"};
char *pronouns[] = {"he", "she", "it", "they", "we", "i", "you"};
char *verbs[] = {"run", "eat", "sleep", "write", "go", "see"};
```

```
char *adjectives[] = {"big", "small", "fast", "red", "happy", "blue"};
char *adverbs[] = {"quickly", "slowly", "well", "badly", "very"};
char *prepositions[] = {"in", "on", "at", "with", "by", "for"};

#define N_NOUNS (sizeof(nouns)/sizeof(nouns[0]))
#define N_PRON  (sizeof(pronouns)/sizeof(pronouns[0]))
#define N_VERBS (sizeof(verbs)/sizeof(verbs[0]))
#define N_ADJ   (sizeof(adjectives)/sizeof(adjectives[0]))
#define N_ADV   (sizeof(adverbs)/sizeof(adverbs[0]))
#define N_PREP  (sizeof(prepositions)/sizeof(prepositions[0]))

int is_in_list(char *word, char *list[], int size) {
   for (int i = 0; i < size; i++) {
      if (strcmp(word, list[i]) == 0)
         return 1;
   }
   return 0;
}
%}

%%
[a-zA-Z]+ {
   char word[100];
   strcpy(word, yytext);
   for (int i = 0; word[i]; i++) word[i] = tolower(word[i]);

   printf("Token: %s --> ", yytext);

   if (is_in_list(word, nouns, N_NOUNS)) printf("NOUN\n");
   else if (is_in_list(word, pronouns, N_PRON)) printf("PRONOUN\n");
   else if (is_in_list(word, verbs, N_VERBS)) printf("VERB\n");
```

```
    else if (is_in_list(word, adjectives, N_ADJ)) printf("ADJECTIVE\n");
    else if (is_in_list(word, adverbs, N_ADV)) printf("ADVERB\n");
    else if (is_in_list(word, prepositions, N_PREP)) printf("PREPOSITION\n");
    else printf("UNKNOWN\n");
}

[ \t\n]+ { /* ignore */ }
. { /* ignore */ }
%%

int main() {
    printf("Enter input text:\n");
    yylex();
    return 0;
}
```

Exection Commands –

```
flex cd1b.l
 gcc lex.yy.c -o cd1b -lfl
./cd1b
```

Cd1c.l – Lex Program to Count Number of Words, Vowels, Lines, and Characters

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

int word_count = 0;
int vowel_count = 0;
int line_count = 0;
int char_count = 0;

int is_vowel(char c) {
```

```
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
%}

%%
[a-zA-Z]+ {
    word_count++;
    for (int i = 0; yytext[i] != '\0'; i++) {
        char_count++;
        if (is_vowel(yytext[i]))
            vowel_count++;
    }
}

[0-9]+ {
    word_count++;
    for (int i = 0; yytext[i] != '\0'; i++)
        char_count++;
}

\n {
    line_count++;
    char_count++;
}

. {
    char_count++;
}
%%
```

```
int main() {
    printf("Enter input:\n");
    yylex();
    printf("Number of words: %d\n", word_count);
    printf("Number of vowels: %d\n", vowel_count);
    printf("Number of lines: %d\n", line_count);
    printf("Number of characters: %d\n", char_count);
    return 0;
}
```

Execution commands –

```
flex cd1c.l
 gcc lex.yy.c -o cd1c -lfl
./cd1c
```

Cd1d.l – Lex Program to Perform Find and Replace Operation in a Text File

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

char find_str[100];
char replace_str[100];
size_t find_len, replace_len;
char *buffer = NULL;
size_t bufsize = 0;

void find_and_replace(char *input);
%}

%%
.|\n {
```

```
    size_t old_len = bufsize;
    bufsize += yyleng;
    buffer = realloc(buffer, bufsize + 1);
    if (!buffer) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    memcpy(buffer + old_len, yytext, yyleng);
    buffer[bufsize] = '\0';
}
%%

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s inputfile\n", argv[0]);
        return 1;
    }

    FILE *fp = fopen(argv[1], "r");
    if (!fp) {
        perror("File open error");
        return 1;
    }

    yyin = fp;
    yylex();
    fclose(fp);

    printf("Enter string to find: ");
    scanf("%99s", find_str);
    printf("Enter string to replace with: ");
```

```
    scanf("%99s", replace_str);

    find_len = strlen(find_str);
    replace_len = strlen(replace_str);

    find_and_replace(buffer);
    free(buffer);
    return 0;
}

void find_and_replace(char *input) {
    char *pos = input;
    char *result = malloc(strlen(input) * 2 + 1); // Extra space
    if (!result) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    result[0] = '\0';

    while ((pos = strstr(input, find_str)) != NULL) {
        strncat(result, input, pos - input);
        strcat(result, replace_str);
        input = pos + find_len;
    }
    strcat(result, input);

    printf("\nModified text:\n%s\n", result);
    free(result);
}
```

Sample.txt –

Hello World from Lex

Execution commands –

flex cd1d.l

gcc lex.yy.c -o cd1d -lfl

./cd1d sample.txt

Cd1e.l – Lex Program to Perform Case Conversion (Uppercase, Lowercase, Sentence Case, and Toggle Case)

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

enum CaseMode { NONE, UPPER, LOWER, SENTENCE, TOGGLE } mode = NONE;
int sentence_start = 1;

void to_upper(char *text);
void to_lower(char *text);
void to_sentence(char *text);
void to_toggle(char *text);
%}

%%
.+ {
   switch (mode) {
      case UPPER:
         to_upper(yytext);
         printf("%s", yytext);
         break;
      case LOWER:
         to_lower(yytext);
```

```
            printf("%s", yytext);
            break;
        case SENTENCE:
            to_sentence(yytext);
            printf("%s", yytext);
            break;
        case TOGGLE:
            to_toggle(yytext);
            printf("%s", yytext);
            break;
        default:
            printf("%s", yytext);
    }
}
%%

void to_upper(char *text) {
    for (int i = 0; text[i]; i++)
        text[i] = toupper(text[i]);
}

void to_lower(char *text) {
    for (int i = 0; text[i]; i++)
        text[i] = tolower(text[i]);
}

void to_sentence(char *text) {
    for (int i = 0; text[i]; i++) {
        if (sentence_start && isalpha(text[i])) {
            text[i] = toupper(text[i]);
            sentence_start = 0;
```

```
        } else {
            text[i] = tolower(text[i]);
        }
        if (text[i] == '.' || text[i] == '!' || text[i] == '?')
            sentence_start = 1;
    }
}

void to_toggle(char *text) {
    for (int i = 0; text[i]; i++) {
        if (islower(text[i]))
            text[i] = toupper(text[i]);
        else if (isupper(text[i]))
            text[i] = tolower(text[i]);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s [upper|lower|sentence|toggle]\n", argv[0]);
        return 1;
    }

    if (strcmp(argv[1], "upper") == 0) mode = UPPER;
    else if (strcmp(argv[1], "lower") == 0) mode = LOWER;
    else if (strcmp(argv[1], "sentence") == 0) mode = SENTENCE;
    else if (strcmp(argv[1], "toggle") == 0) mode = TOGGLE;
    else {
        printf("Invalid mode. Use: upper, lower, sentence, toggle\n");
        return 1;
    }
```

```
    yylex();
    return 0;
}
```

Sample.txt –

```
Hello World from Lex
```

Execution commands –

```
flex cd1e.l
 gcc lex.yy.c -o cd1e -lfl
 ./cd1e sentence < sample.txt
./cd1e upper < sample.txt
./cd1e lower < sample.txt
./cd1e toggle < sample.txt
```

Cd2.l – Lex Program to Identify Tokens in a C Source Code (Keywords, Identifiers, Operators, Numbers, Strings, and Delimiters)

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

int line_num = 1;
extern FILE *yyin;
%}

%%
"int"|"float"|"char"|"double"|"if"|"else"|"while"|"for"|"return" {
    printf("KEYWORD\t\t%s\t(line %d)\n", yytext, line_num);
}

[a-zA-Z_][a-zA-Z0-9_]* {
    printf("IDENTIFIER\t%s\t(line %d)\n", yytext, line_num);
```

```
}

[0-9]+(\.[0-9]+)? {
   printf("NUMBER\t\t%s\t(line %d)\n", yytext, line_num);
}

"=="|"="|"+"|"-"|"*"|"/"|"<"|">" {
   printf("OPERATOR\t%s\t(line %d)\n", yytext, line_num);
}

[(){};,.\[\]] {
   printf("DELIMITER\t%s\t(line %d)\n", yytext, line_num);
}

\"([^\\\"]|\\.)*\" {
   printf("STRING\t\t%s\t(line %d)\n", yytext, line_num);
}

"//".*              { /* ignore single-line comments */ }
"/*"([^*]|\*+[^*/])*\*+\/   { /* ignore multi-line comments */ }

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

\n                  { line_num++; }

.                   { printf("UNKNOWN\t\t%s\t(line %d)\n", yytext, line_num); }
%%

int main(int argc, char *argv[]) {
   if (argc > 1) {
      FILE *f = fopen(argv[1], "r");
```

```
        if (!f) {
            perror("Cannot open file");
            return 1;
        }
        yyin = f;
    }
    yylex();
    return 0;
}
```

Cd2sample.c –

```
#include <stdio.h>
int main() {
    int a = 10, b = 5;
    int sum, diff, prod, quot;
    sum = a + b;
    diff = a - b;
    prod = a * b;
    quot = a / b;
    printf("Sum = %d\n", sum);
    printf("Difference = %d\n", diff);
    printf("Product = %d\n", prod);
    printf("Quotient = %d\n", quot);
    return 0;
}
```

Execution commands –

```
 flex cd2.l
 gcc lex.yy.c -o cd2 -lfl
 ./cd2 cd2sample.c
```

Cd3.y - **YAAC specifications and implement Parser for specified grammar single file (Implement Calculator)**

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include "cd3.tab.h"

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

%union {
   double dval;
}

/* Tokens */
%token <dval> NUMBER
%token SIN COS TAN SQRT LOG

/* Types */
%type <dval> expr

/* Operator precedence and associativity */
%left '+' '-'
%left '*' '/'
%right '^'
%right UMINUS

%%
```

```
input:
   /* empty */
   | input expr '\n'   { printf("Result = %lf\n", $2); }
   ;

expr:
    NUMBER                { $$ = $1; }
   | expr '+' expr        { $$ = $1 + $3; }
   | expr '-' expr        { $$ = $1 - $3; }
   | expr '*' expr        { $$ = $1 * $3; }
   | expr '/' expr        {
                            if ($3 == 0) {
                               yyerror("Division by zero");
                               $$ = 0;
                            } else $$ = $1 / $3;
                          }
   | expr '^' expr        { $$ = pow($1, $3); }
   | '(' expr ')'         { $$ = $2; }
   | '-' expr %prec UMINUS    { $$ = -$2; }
   | SIN '(' expr ')'     { $$ = sin($3 * M_PI / 180.0); }
   | COS '(' expr ')'     { $$ = cos($3 * M_PI / 180.0); }
   | TAN '(' expr ')'     { $$ = tan($3 * M_PI / 180.0); }
   | SQRT '(' expr ')'    { $$ = sqrt($3); }
   | LOG '(' expr ')'     { $$ = log($3); }
   ;
%%
/* Lexical analyzer for calculator */
int yylex(void) {
   int c;

   while ((c = getchar()) == ' ' || c == '\t');
```

```
if (c == EOF)
    return 0;
if (c == '\n')
    return '\n';

if (isdigit(c) || c == '.') {
    ungetc(c, stdin);
    double val;
    scanf("%lf", &val);
    yylval.dval = val;
    return NUMBER;
}

if (isalpha(c)) {
    char func[32];
    int i = 0;
    while (isalpha(c)) {
        func[i++] = c;
        c = getchar();
    }
    func[i] = '\0';
    ungetc(c, stdin);

    if (strcmp(func, "sin") == 0) return SIN;
    if (strcmp(func, "cos") == 0) return COS;
    if (strcmp(func, "tan") == 0) return TAN;
    if (strcmp(func, "sqrt") == 0) return SQRT;
    if (strcmp(func, "log") == 0) return LOG;

    yyerror("Unknown function");
```

```
        return 0;
    }

    return c;
}

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

/* Main function */
int main() {
    printf("Enter expression (press Enter to evaluate, Ctrl+D to exit)\n");
    yyparse();
    return 0;
}
```

EXECUTION COMMANDS -

```
bison -d cd3.y
gcc cd3.tab.c -lm -o calc
./calc
```

Cd5.l – Lex and Yacc Program to Validate Arithmetic Expressions

```
%{
#include "cd5.tab.h"
#include <stdlib.h>
%}

%%
[0-9]+      { return NUMBER; }
```

```
[ \t]      ;       /* Ignore spaces and tabs */
\n         { return '\n'; }
"+"        { return '+'; }
"-"        { return '-'; }
"*"        { return '*'; }
"/"        { return '/'; }
"("        { return '('; }
")"        { return ')'; }
.          { printf("Invalid character: %s\n", yytext); }
%%

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

cd4.l - YAAC specifications -Implement Scientific Calculator with separate LEX and YACC file (OR) Implement Parser for C programming language.

```
%{
#include "cd4.tab.h"
#include <stdio.h>
#include <stdlib.h>
%}

digit   [0-9]
number  {digit}+(\.{digit}*)?([eE][+-]?{digit}+)?

%%
"sin"      { yylval.sval = "sin"; return FUNC; }
"cos"      { yylval.sval = "cos"; return FUNC; }
"tan"      { yylval.sval = "tan"; return FUNC; }
"log"      { yylval.sval = "log"; return FUNC; }
"sqrt"     { yylval.sval = "sqrt"; return FUNC; }
```

```
"pi"       { yylval.dval = 3.141592653589793; return CONST; }
"e"        { yylval.dval = 2.718281828459045; return CONST; }

{number}    { yylval.dval = atof(yytext); return NUMBER; }

"+"        { return '+'; }
"-"        { return '-'; }
"*"        { return '*'; }
"/"        { return '/'; }
"^"        { return '^'; }
"("        { return '('; }
")"        { return ')'; }
\n         { return '\n'; }

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

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

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

cd4.y:

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

double call_func(char *name, double val);
int yylex(void);
int yyerror(const char *s);
```

```
%}

%union {
   double dval;
   char *sval;
}

/* Tokens */
%token <dval> NUMBER CONST
%token <sval> FUNC

/* Non-terminal type */
%type <dval> input expr

/* Operator precedence */
%left '+' '-'
%left '*' '/'
%right '^'
%right UMINUS

%%
input:
     /* empty */
   | input expr '\n'  { printf("Result = %lf\n", $2); }
   ;

expr:
     expr '+' expr       { $$ = $1 + $3; }
   | expr '-' expr       { $$ = $1 - $3; }
   | expr '*' expr       { $$ = $1 * $3; }
   | expr '/' expr       {
```

```
                        if ($3 == 0) {
                            yyerror("Division by zero");
                            $$ = 0;
                        } else $$ = $1 / $3;
                    }
    | expr '^' expr       { $$ = pow($1, $3); }
    | '-' expr %prec UMINUS { $$ = -$2; }
    | '(' expr ')'        { $$ = $2; }
    | FUNC '(' expr ')'    { $$ = call_func($1, $3); }
    | CONST               { $$ = $1; }
    | NUMBER              { $$ = $1; }
    ;
%%

double call_func(char *name, double val) {
    if (strcmp(name, "sin") == 0)  return sin(val);
    if (strcmp(name, "cos") == 0)  return cos(val);
    if (strcmp(name, "tan") == 0)  return tan(val);
    if (strcmp(name, "log") == 0)  return log(val);
    if (strcmp(name, "sqrt") == 0) return sqrt(val);

    printf("Unknown function: %s\n", name);
    return 0;
}

int yyerror(const char *s) {
    printf("Error: %s\n", s);
    return 0;
}

int main() {
```

```
    printf("Enter an expression (press Enter to evaluate, Ctrl+D to exit):\n");
    yyparse();
    return 0;
}
```

EXECUTION COMMANDS -

bison -d cd4.y (will give a warning, ignore and proceed)

flex cd4.l

gcc cd4.tab.c lex.yy.c -lfl -lm -o calc

./calc

cd5.y –

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

int yylex();
void yyerror(const char *s);
int valid = 1;   // flag to track if expression is valid
%}

%token NUMBER

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

%%
input :
```

```
    E '\n'    {
        if (valid) printf("Valid expression.\n");
        valid = 1;  /* Reset flag for next input */
    }
  ;

E : E '+' T
  | E '-' T
  | T
 ;

T : T '*' F
  | T '/' F
  | F
 ;

F : '(' E ')'
  | NUMBER
 ;
%%

void yyerror(const char *s) {
    valid = 0;
    printf("Error: syntax error\n");
}

int main() {
    printf("Enter an arithmetic expression:\n");
    yyparse();
    return 0;
}
```

Execution commands –

bison -d cd5.y -o cd5.tab.c

flex cd5.l

gcc lex.yy.c cd5.tab.c -o cd5 -lfl

./cd5

Cd6.l – Lex and Yacc Program to Generate Intermediate Code (Quadruple and Triple Representation) for Arithmetic Expressions

```
%{
#include "cd6.tab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}

%%
[a-zA-Z][a-zA-Z0-9]*   { yylval.str = strdup(yytext); return ID; }
[0-9]+                 { yylval.str = strdup(yytext); return NUM; }
"="                    { return ASSIGN; }
"*"                    { return MUL; }
"/"                    { return DIV; }
"+"                    { return PLUS; }
"-"                    { return MINUS; }
"("                    { return LPAREN; }
")"                    { return RPAREN; }
[ \t\n]                ;
.                      { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

cd6.y –

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

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

/* --- Structures --- */
struct Quadruple {
    char op[5], arg1[20], arg2[20], result[20];
};
struct Triple {
    char op[5], arg1[20], arg2[20];
};

struct Quadruple q[50];
struct Triple t[50];
int qIndex = 0, tempCount = 1;

/* --- Helper Functions --- */
char *newTemp() {
    static char temp[10];
    sprintf(temp, "t%d", tempCount++);
    return strdup(temp);
}

char *indexRef(int index) {
    static char ref[10];
    sprintf(ref, "(%d)", index);
```

```
    return strdup(ref);
}
%}

%union { char *str; }

%token <str> ID NUM
%token ASSIGN PLUS MINUS MUL DIV LPAREN RPAREN
%left PLUS MINUS
%left MUL DIV
%right UMINUS
%type <str> expr
%start statement

%%
statement:
    ID ASSIGN expr {
        /* --- Quadruple for assignment --- */
        strcpy(q[qIndex].op, "=");
        strcpy(q[qIndex].arg1, $3);
        strcpy(q[qIndex].arg2, "-");
        strcpy(q[qIndex].result, $1);

        /* --- Triple for assignment --- */
        strcpy(t[qIndex].op, "=");
        /* Replace temporaries (t1, t2) with their index refs */
        if ($3[0] == 't')
            strcpy(t[qIndex].arg1, indexRef(atoi($3 + 1) - 1));
        else
            strcpy(t[qIndex].arg1, $3);
        strcpy(t[qIndex].arg2, "-");
```

```
        qIndex++;

        printf("\n=== Quadruple Representation ===\n");
        printf("Index\tOp\tArg1\tArg2\tResult\n");
        printf("-----\t--\t----\t----\t------\n");
        for (int i = 0; i < qIndex; i++)
            printf("%d\t%s\t%s\t%s\t%s\n", i, q[i].op, q[i].arg1, q[i].arg2, q[i].result);

        printf("\n=== Triple Representation ===\n");
        printf("Index\tOp\tArg1\tArg2\n");
        printf("-----\t--\t----\t----\n");
        for (int i = 0; i < qIndex; i++)
            printf("%d\t%s\t%s\t%s\n", i, t[i].op, t[i].arg1, t[i].arg2);
    }
  ;

expr:
    expr PLUS expr {
        char *temp = newTemp();
        strcpy(q[qIndex].op, "+");
        strcpy(q[qIndex].arg1, $1);
        strcpy(q[qIndex].arg2, $3);
        strcpy(q[qIndex].result, temp);

        strcpy(t[qIndex].op, "+");
        /* Replace temp references in Triple */
        if ($1[0] == 't')
            strcpy(t[qIndex].arg1, indexRef(atoi($1 + 1) - 1));
        else
            strcpy(t[qIndex].arg1, $1);
```

```
    if ($3[0] == 't')
        strcpy(t[qIndex].arg2, indexRef(atoi($3 + 1) - 1));
    else
        strcpy(t[qIndex].arg2, $3);

    $$ = strdup(temp);
    qIndex++;
 }
| expr MINUS expr {
    char *temp = newTemp();
    strcpy(q[qIndex].op, "-");
    strcpy(q[qIndex].arg1, $1);
    strcpy(q[qIndex].arg2, $3);
    strcpy(q[qIndex].result, temp);

    strcpy(t[qIndex].op, "-");
    if ($1[0] == 't')
        strcpy(t[qIndex].arg1, indexRef(atoi($1 + 1) - 1));
    else
        strcpy(t[qIndex].arg1, $1);

    if ($3[0] == 't')
        strcpy(t[qIndex].arg2, indexRef(atoi($3 + 1) - 1));
    else
        strcpy(t[qIndex].arg2, $3);

    $$ = strdup(temp);
    qIndex++;
 }
| expr MUL expr {
```

```
    char *temp = newTemp();
    strcpy(q[qIndex].op, "*");
    strcpy(q[qIndex].arg1, $1);
    strcpy(q[qIndex].arg2, $3);
    strcpy(q[qIndex].result, temp);

    strcpy(t[qIndex].op, "*");
    if ($1[0] == 't')
       strcpy(t[qIndex].arg1, indexRef(atoi($1 + 1) - 1));
    else
       strcpy(t[qIndex].arg1, $1);

    if ($3[0] == 't')
       strcpy(t[qIndex].arg2, indexRef(atoi($3 + 1) - 1));
    else
       strcpy(t[qIndex].arg2, $3);

    $$ = strdup(temp);
    qIndex++;
 }
| expr DIV expr {
    char *temp = newTemp();
    strcpy(q[qIndex].op, "/");
    strcpy(q[qIndex].arg1, $1);
    strcpy(q[qIndex].arg2, $3);
    strcpy(q[qIndex].result, temp);

    strcpy(t[qIndex].op, "/");
    if ($1[0] == 't')
       strcpy(t[qIndex].arg1, indexRef(atoi($1 + 1) - 1));
    else
```

```
        strcpy(t[qIndex].arg1, $1);

    if ($3[0] == 't')
        strcpy(t[qIndex].arg2, indexRef(atoi($3 + 1) - 1));
    else
        strcpy(t[qIndex].arg2, $3);

    $$ = strdup(temp);
    qIndex++;
 }
| LPAREN expr RPAREN { $$ = $2; }
| MINUS expr %prec UMINUS {
    char *temp = newTemp();
    strcpy(q[qIndex].op, "~");
    strcpy(q[qIndex].arg1, $2);
    strcpy(q[qIndex].arg2, "-");
    strcpy(q[qIndex].result, temp);

    strcpy(t[qIndex].op, "~");
    if ($2[0] == 't')
        strcpy(t[qIndex].arg1, indexRef(atoi($2 + 1) - 1));
    else
        strcpy(t[qIndex].arg1, $2);
    strcpy(t[qIndex].arg2, "-");

    $$ = strdup(temp);
    qIndex++;
 }
| ID  { $$ = strdup($1); }
| NUM { $$ = strdup($1); }
;
```

```
%%

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

int main() {
   printf("Enter an arithmetic expression:\n");
   yyparse();
   return 0;
}
```

Execution commands –

```
bison -d cd6.y -o cd6.tab.c
flex cd6.l
gcc lex.yy.c cd6.tab.c -o cd6 -lfl
./cd6
```

Cd7.c – C Program for Intermediate Code Optimization using Constant Folding, Common Subexpression Elimination, and Copy Propagation

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

struct Quadruple {
   char op[5], arg1[20], arg2[20], res[20];
} q[20];

int n;

int isNumber(char *s) {
   for (int i = 0; s[i]; i++)
      if (!isdigit(s[i])) return 0;
```

```
    return 1;
}

void constantFolding() {
    for (int i = 0; i < n; i++) {
        if (isNumber(q[i].arg1) && isNumber(q[i].arg2)) {
            int a = atoi(q[i].arg1);
            int b = atoi(q[i].arg2);
            int result;

            if (strcmp(q[i].op, "+") == 0) result = a + b;
            else if (strcmp(q[i].op, "-") == 0) result = a - b;
            else if (strcmp(q[i].op, "*") == 0) result = a * b;
            else if (strcmp(q[i].op, "/") == 0 && b != 0) result = a / b;
            else continue;

            sprintf(q[i].arg1, "%d", result);
            strcpy(q[i].arg2, "-");
            strcpy(q[i].op, "=");
        }
    }
}

void commonSubexprElim() {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(q[i].op, q[j].op) == 0 &&
                strcmp(q[i].arg1, q[j].arg1) == 0 &&
                strcmp(q[i].arg2, q[j].arg2) == 0) {
                strcpy(q[j].op, "=");
                strcpy(q[j].arg1, q[i].res);
```

```
            strcpy(q[j].arg2, "-");
        }
    }
  }
}

void copyPropagation() {
  for (int i = 0; i < n; i++) {
    if (strcmp(q[i].op, "=") == 0 && strcmp(q[i].arg2, "-") == 0) {
      char source[20];
      strcpy(source, q[i].arg1);
      char target[20];
      strcpy(target, q[i].res);
      for (int j = i + 1; j < n; j++) {
        if (strcmp(q[j].arg1, target) == 0)
          strcpy(q[j].arg1, source);
        if (strcmp(q[j].arg2, target) == 0)
          strcpy(q[j].arg2, source);
      }
    }
  }
}

int main() {
  printf("Enter number of Quadruples: ");
  scanf("%d", &n);
  printf("Enter Quadruples in format: op arg1 arg2 result\n");

  for (int i = 0; i < n; i++) {
    scanf("%s %s %s %s", q[i].op, q[i].arg1, q[i].arg2, q[i].res);
  }
```

```
    printf("\nIntermediate Code:\n");
    for (int i = 0; i < n; i++)
        printf("%s\t%s\t%s\t%s\n", q[i].op, q[i].arg1, q[i].arg2, q[i].res);

    constantFolding();
    commonSubexprElim();
    copyPropagation();

    printf("\nOptimized Code:\n");
    for (int i = 0; i < n; i++)
        printf("%s\t%s\t%s\t%s\n", q[i].op, q[i].arg1, q[i].arg2, q[i].res);

    return 0;
}
```

Execution commands –

```
 gcc cd7.c -o cd7
 ./cd7
Enter number of Quadruples: 6
Enter Quadruples in format: op arg1 arg2 result
* 2 3 t1
+ t1 5 t2
= t2 - a
* 2 3 t3
+ t3 5 t4
= t4 - b
```