**Name**: D. Ashwanth Ram

**Reg No:** 3122 21 5001 013

—----------------------------------------------------------------------------------------------------------------------------

**Assignment-07: Implementation of code optimization techniques:**

-------------------------------------------------------------------------------------------------------------------------------

**Aim**:

Consider the Three Address Code sequences and apply the following techniques to optimize the code.

1. Constant folding

2. Algebraic identities

3. Strength reduction

4. Dead code elimination

**Code**:

**Optimize.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void yyerror(const char \*s);

int yylex();

typedef struct {

char \*name;

char \*expression;

} entry;

entry symbol\_table[100];

int symbol\_table\_index = 0;

int lookup(char \*expr);

void add\_expression(char \*name, char \*expr);

char \*concatenate(char \*a, char \*op, char \*b);

int is\_constant(char \*name);

%}

%union {

int intval;

char \*strval;

}

%token <intval> NUMBER

%token <strval> VARIABLE

%token ASSIGN MULTIPLY ADD POWER NEWLINE

%left ADD

%left MULTIPLY

%right POWER

%type <strval> expression line

%%

input:

| input line NEWLINE

;

line:

VARIABLE ASSIGN expression {

int index = lookup($3);

if (index == -1) {

// Expression not yet seen, so we add it

add\_expression($1, $3);

printf("%s = %s\n", $1, $3);

} else {

// Expression already exists, skip duplicate

printf("// Duplicate of %s; skipping %s\n",

symbol\_table[index].name, $1);

}

}

;

expression:

NUMBER {

char buffer[12];

sprintf(buffer, "%d", $1);

$$ = strdup(buffer);

}

| VARIABLE {

$$ = strdup($1);

}

| expression MULTIPLY expression {

if (is\_constant($1) && is\_constant($3)) {

char buffer[12];

sprintf(buffer, "%d", atoi($1) \* atoi($3));

$$ = strdup(buffer);

} else if (strcmp($1, "1") == 0) $$ = $3;

else if (strcmp($3, "1") == 0) $$ = $1;

else {

$$ = concatenate($1, "\*", $3);

}

}

| expression ADD expression {

if (is\_constant($1) && is\_constant($3)) {

char buffer[12];

sprintf(buffer, "%d", atoi($1) + atoi($3));

$$ = strdup(buffer);

} else if (strcmp($1, "0") == 0) $$ = $3;

else if (strcmp($3, "0") == 0) $$ = $1;

else {

$$ = concatenate($1, "+", $3);

}

}

| expression POWER expression {

if (strcmp($3, "2") == 0) $$ = concatenate($1, "\*", $1);

else $$ = concatenate($1, "\*\*", $3);

}

;

%%

int main() {

yyparse();

return 0;

}

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

int lookup(char \*expr) {

for (int i = 0; i < symbol\_table\_index; i++) {

if (strcmp(symbol\_table[i].expression, expr) == 0) {

return i;

}

}

return -1;

}

void add\_expression(char \*name, char \*expr) {

symbol\_table[symbol\_table\_index].name = strdup(name);

symbol\_table[symbol\_table\_index].expression = strdup(expr);

symbol\_table\_index++;

}

int is\_constant(char \*name) {

for (int i = 0; name[i] != '\0'; i++) {

if (name[i] < '0' || name[i] > '9') return 0;

}

return 1;

}

char \*concatenate(char \*a, char \*op, char \*b) {

char \*result = malloc(strlen(a) + strlen(op) + strlen(b) + 1);

sprintf(result, "%s%s%s", a, op, b);

return result;

}

**Optimize.l**

%{

#include "Optimize.tab.h"

%}

digit [0-9]+

variable [a-zA-Z][a-zA-Z0-9]\*

%%

{digit} { yylval.intval = atoi(yytext); return NUMBER; }

{variable} { yylval.strval = strdup(yytext); return VARIABLE; }

"=" { return ASSIGN; }

"\*\*" { return POWER; }

"\*" { return MULTIPLY; }

"+" { return ADD; }

\n { return NEWLINE; }

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

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

Input.txt

t1 = 5 \* 3

t2 = a + 0

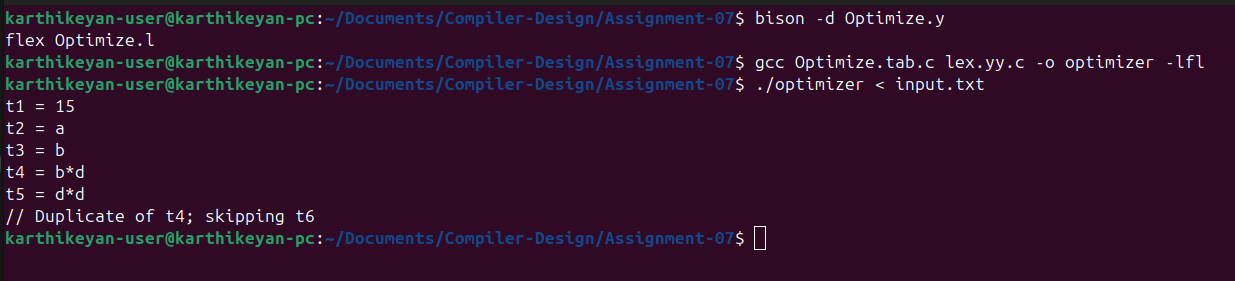
t3 = b \* 1

t4 = b \* d

t5 = d \*\* 2

t6 = b \* d

**Output:**



**Learning Outcome:**

1. Understand the role of optimizer in the compilation process.

2. Implemented lexer and parser components using Flex and Yacc.

3. Generate optimized code for three address code.

4. Handle syntax errors and improve debugging skills in optimized code generation