**Name**: Karthikeyan A

**Reg No**: 3122 21 5001 041

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

**Assignment-06: Implementation of code optimization techniques**

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

**Aim:**

Develop an intermediate code generator to generate three address code for the following statements by writing suitable syntax directed translation rules

1. Assignment statements

2. Boolean expressions

**Code:**

**intermediate.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int temp\_count = 0; // Counter for temporary variables

int label\_count = 100; // Counter for labels

// Function to generate a new temporary variable

int new\_temp() {

return temp\_count++;

}

// Function to generate a new label

int new\_label() {

return label\_count++;

}

// Emit an intermediate code statement

void emit(const char\* op, const char\* arg1, const char\* arg2, const char\* result) {

if (strcmp(op, "=") == 0) {

printf("%s = %s\n", result, arg1);

} else if (arg2) {

printf("%s = %s %s %s\n", result, arg1, op, arg2);

} else {

printf("%s = %s %s\n", result, op, arg1);

}

}

// Emit an 'if-goto' statement

void emit\_if\_goto(const char\* op, const char\* arg1, const char\* arg2, const char\* label) {

printf("if %s %s %s goto %s\n", arg1, op, arg2, label);

}

// Emit a label

void emit\_label(const char\* label) {

printf("%s:\n", label);

}

// Emit a goto statement

void emit\_goto(const char\* label) {

printf("goto %s\n", label);

}

void yyerror(const char\* s) { }

int yylex();

%}

%union {

int num;

char\* id;

}

%token <id> ID

%token <num> NUM

%token PLUS MINUS MULT DIV ASSIGN LT AND OR

%token LPAREN RPAREN SEMICOLON

%type <id> expr assignment boolean\_expr

%left OR

%left AND

%nonassoc LT

%left PLUS MINUS

%left MULT DIV

%right ASSIGN UMINUS

%%

stmt\_list:

stmt\_list stmt SEMICOLON

| stmt SEMICOLON

;

stmt:

assignment

| boolean\_expr

;

assignment:

ID ASSIGN expr {

emit("=", $3, NULL, $1);

}

;

expr:

expr PLUS expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("+", $1, $3, temp);

$$ = strdup(temp);

}

| expr MINUS expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("-", $1, $3, temp);

$$ = strdup(temp);

}

| expr MULT expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("\*", $1, $3, temp);

$$ = strdup(temp);

}

| expr DIV expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("/", $1, $3, temp);

$$ = strdup(temp);

}

| MINUS expr %prec UMINUS {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("-", $2, NULL, temp);

$$ = strdup(temp);

}

| ID {

$$ = strdup($1);

}

| NUM {

char temp[10];

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

$$ = strdup(temp);

}

| LPAREN expr RPAREN {

$$ = $2;

}

;

boolean\_expr:

expr LT expr {

char label\_true[10], label\_end[10];

sprintf(label\_true, "L%d", new\_label());

sprintf(label\_end, "L%d", new\_label());

emit\_if\_goto("<", $1, $3, label\_true);

emit\_goto(label\_end);

emit\_label(label\_true);

printf("1\n");

emit\_label(label\_end);

}

| expr OR expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("or", $1, $3, temp);

$$ = strdup(temp);

}

| expr AND expr {

int t = new\_temp();

char temp[10];

sprintf(temp, "t%d", t);

emit("and", $1, $3, temp);

$$ = strdup(temp);

}

;

%%

int main() {

return yyparse();

}

**scanner.l**

%{

#include "y.tab.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

%}

%%

"and" { return AND; }

"or" { return OR; }

":=" { return ASSIGN; }

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return MULT; }

"/" { return DIV; }

"<" { return LT; }

"(" { return LPAREN; }

")" { return RPAREN; }

";" { return SEMICOLON; }

[0-9]+ { yylval.num = atoi(yytext); return NUM; }

[a-zA-Z][a-zA-Z0-9]\* { yylval.id = strdup(yytext); return ID; }

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

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

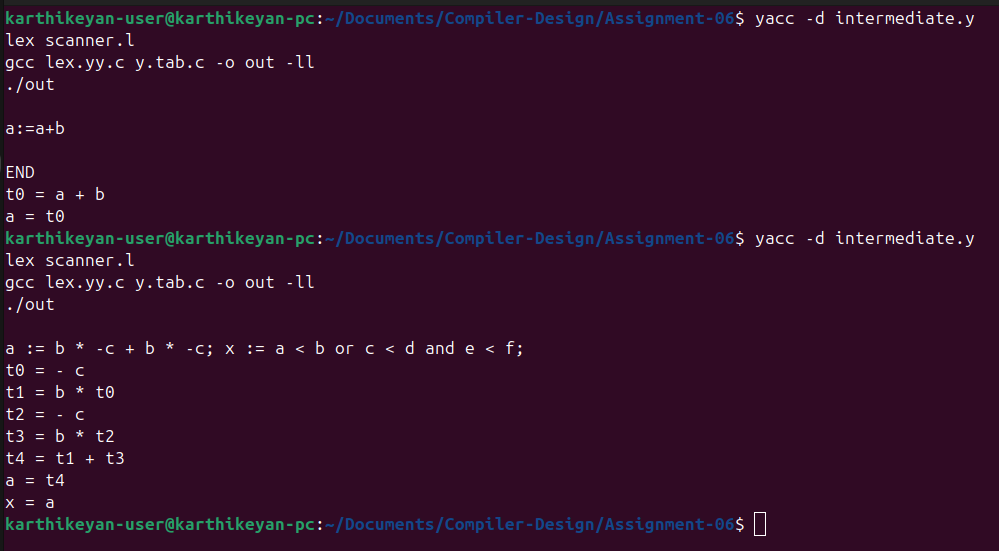
%%

int yywrap() {

return 1;

}

**Output:**



**Learning Outcome:**

1. Understand the role of intermediate code in the compilation process.
2. Apply syntax-directed translation rules for generating intermediate code.
3. Implemented lexer and parser components using Flex and Yacc.