**Name**: Karthikeyan A

**Reg No**: 3122 21 5001 041

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

**Assignment-09: Mini Compiler**

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

**Aim:**

Develop a end to end mini compiler using lex and yacc tool.

**Code:**

**Q.l**

%{

#include <stdio.h>

#include "y.tab.h"

%}

%%

"#include"[ ]\*"<"[^>]\*">" { printf("%s - preprocessor directive\n", yytext); return INCLUDE; }

"int" { printf("int - keyword\n"); return INT\_TYPE; }

"float" { printf("float - keyword\n"); return FLOAT\_TYPE; }

"if" { printf("if - keyword\n"); return IF; }

"else" { printf("else - keyword\n"); return ELSE; }

"while" { printf("while - keyword\n"); return WHILE; }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("%s - identifier\n", yytext); yylval.str = strdup(yytext); return ID; }

[0-9]+ { printf("%s - integer constant\n", yytext); yylval.intval = atoi(yytext); return INT; }

[0-9]+\.[0-9]+ { printf("%s - float constant\n", yytext); yylval.fltval = atof(yytext); return FLOAT; }

">" { printf("> - relational operator\n"); return GT; }

"<" { printf("< - relational operator\n"); return LT; }

">=" { printf(">= - relational operator\n"); return GE; }

"<=" { printf("<= - relational operator\n"); return LE; }

"==" { printf("== - relational operator\n"); return EQ; }

"!=" { printf("!= - relational operator\n"); return NE; }

"=" { printf("= - assignment operator\n"); return EQUAL; }

"{" { printf("{ - special character\n"); return LBRACE; }

"}" { printf("} - special character\n"); return RBRACE; }

"(" { printf("( - special character\n"); return LPAREN; }

")" { printf(") - special character\n"); return RPAREN; }

";" { printf("; - special character\n"); return SEMI; }

"," { printf(", - special character\n"); return yytext[0]; }

"//".\* { printf("%s - multiline comment\n",yytext);}

"+" { printf("+ - Arithmetic Operator\n"); return PLUS; }

"-" { printf("- - Arithmetic Operator\n"); return MINUS; }

"\*" { printf("\* - Arithmetic Operator\n"); return STAR; }

"/" { printf("/ - Arithmetic Operator\n"); return DIV; }

[ \t\n] { ; }

. { printf("%s - special character\n", yytext); return yytext[0]; }

%%

**Q.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

void yyerror(const char \*s);

int yylex(void);

int yywrap();

int flag = 0;

extern FILE \*yyin;

int labelCounter() {

static int labels = 0;

return labels++;

}

int varCounter() {

static int variables = 0;

return variables++;

}

typedef struct Node {

int intval;

float fltval;

char \*code;

char \*optcode;

char \*tac;

char \*opttac;

char \*gen;

} Node;

Node \*makeNode() {

Node \*n = (Node \*)malloc(sizeof(Node));

n->intval = 0;

n->fltval = 0.0;

n->code = (char \*)malloc(sizeof(char) \* 1024);

n->optcode = (char \*)malloc(sizeof(char) \* 1024);

n->tac = (char \*)malloc(sizeof(char) \* 1024);

n->opttac = (char \*)malloc(sizeof(char) \* 1024);

n->gen = (char \*)malloc(sizeof(char) \* 1024);

return n;

}

%}

%union {

int intval;

float fltval;

char \*str;

struct Node\* node;

}

%token <str> ID IF ELSE WHILE INT\_TYPE FLOAT\_TYPE INCLUDE

%token <intval> INT

%token <fltval> FLOAT

%token GT LT GE LE EQ NE

%left '+' '-'

%left '\*' '/' '%'

%nonassoc UMINUS

%token PLUS MINUS STAR DIV LBRACE RBRACE LPAREN RPAREN EQUAL SEMI

%type <node> Program GlobalStmts GlobalStmt Block Stmts Stmt Decl Type Assign Expr WhileStmt IfStmt Condition Term Factor

%%

Program : GlobalStmts {

printf("\n---INPUT CODE---\n");

system("cat q.c");

printf("\n---SYNTAX CHECK---\n");

printf("\nSyntactically Correct.\n");

printf("\n---TAC CODE---\n");

printf("%s\n", $1->tac);

printf("\n---OPTIMIZED CODE---\n");

printf("%s\n", $1->opttac);

printf("\n---MACHINE CODE---\n");

printf("%s\n", $1->gen);

}

;

GlobalStmts : GlobalStmt GlobalStmts {

$$ = makeNode();

sprintf($$->tac, "%s%s", $1->tac, $2->tac);

sprintf($$->opttac, "%s%s", $1->opttac, $2->opttac);

sprintf($$->gen, "%s%s", $1->gen, $2->gen);

}

| GlobalStmt { $$ = $1; }

;

GlobalStmt : INCLUDE { $$ = makeNode(); }

| Decl SEMI { $$ = $1; }

| WhileStmt { $$ = $1; }

| IfStmt { $$ = $1; }

;

Block : LBRACE Stmts RBRACE { $$ = $2; }

;

Stmts : Stmt Stmts {

$$ = makeNode();

sprintf($$->tac, "%s%s", $1->tac, $2->tac);

sprintf($$->opttac, "%s%s", $1->opttac, $2->opttac);

sprintf($$->gen, "%s%s", $1->gen, $2->gen);

}

| Stmt { $$ = $1; }

;

Stmt : Decl SEMI { $$ = $1; }

| Assign SEMI { $$ = $1; }

| WhileStmt { $$ = $1; }

| IfStmt { $$ = $1; }

;

Decl : Type ID {

$$ = makeNode();

sprintf($$->tac, "declare %s\n", $2);

sprintf($$->opttac, "declare %s\n", $2);

sprintf($$->gen, "ALLOC %s\n", $2);

}

| Type Assign {

$$ = makeNode();

sprintf($$->tac, "declare %s\n%s", $2->code, $2->tac);

sprintf($$->opttac, "declare %s\n%s", $2->code, $2->opttac);

sprintf($$->gen, "ALLOC %s\n%s", $2->code, $2->gen);

}

;

Type : INT\_TYPE { $$ = makeNode(); }

| FLOAT\_TYPE { $$ = makeNode(); }

;

WhileStmt : WHILE LPAREN Condition RPAREN Block {

$$ = makeNode();

int label1 = labelCounter();

int label2 = labelCounter();

char temp[1024];

sprintf(temp, "L%d:\n%sif not %s goto L%d\n%sgoto L%d\nL%d:\n",

label1, $3->tac, $3->code, label2, $5->tac, label1, label2);

sprintf($$->tac, "%s", temp);

sprintf($$->opttac, "%s", temp);

sprintf($$->gen, "L%d:\n%sCMP %s, #0\nJZ L%d\n%sJMP L%d\nL%d:\n",

label1, $3->gen, $3->code, label2, $5->gen, label1, label2);

}

;

IfStmt : IF LPAREN Condition RPAREN Block ELSE Block {

$$ = makeNode();

int label1 = labelCounter();

int label2 = labelCounter();

char temp[1024];

sprintf(temp, "%sif not %s goto L%d\n%sgoto L%d\nL%d:\n%sL%d:\n",

$3->tac, $3->code, label1, $5->tac, label2, label1, $7->tac, label2);

sprintf($$->tac, "%s", temp);

sprintf($$->opttac, "%s", temp);

sprintf($$->gen, "%sCMP %s, #0\nJZ L%d\n%sJMP L%d\nL%d:\n%sL%d:\n",

$3->gen, $3->code, label1, $5->gen, label2, label1, $7->gen, label2);

}

| IF LPAREN Condition RPAREN Block {

$$ = makeNode();

int label1 = labelCounter();

char temp[1024];

sprintf(temp, "%sif not %s goto L%d\n%sL%d:\n",

$3->tac, $3->code, label1, $5->tac, label1);

sprintf($$->tac, "%s", temp);

sprintf($$->opttac, "%s", temp);

sprintf($$->gen, "%sCMP %s, #0\nJZ L%d\n%sL%d:\n",

$3->gen, $3->code, label1, $5->gen, label1);

}

;

Condition : Expr GT Expr {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

sprintf($$->tac, "%s%s%s = %s > %s\n",

$1->tac, $3->tac, $$->code, $1->code, $3->code);

sprintf($$->opttac, "%s%s%s = %s > %s\n",

$1->opttac, $3->opttac, $$->code, $1->code, $3->code);

sprintf($$->gen, "%s%sMOV %s, R0\nCMP %s, R0\nSGT %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Expr LT Expr {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

sprintf($$->tac, "%s%s%s = %s < %s\n",

$1->tac, $3->tac, $$->code, $1->code, $3->code);

sprintf($$->opttac, "%s%s%s = %s < %s\n",

$1->opttac, $3->opttac, $$->code, $1->code, $3->code);

sprintf($$->gen, "%s%sMOV %s, R0\nCMP %s, R0\nSLT %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Expr { /\* Add this rule to handle single expressions in conditions \*/

$$ = makeNode();

sprintf($$->code, "%s", $1->code);

sprintf($$->tac, "%s", $1->tac);

sprintf($$->opttac, "%s", $1->opttac);

sprintf($$->gen, "%s", $1->gen);

}

;

Assign : ID EQUAL Expr {

$$ = makeNode();

sprintf($$->code, "%s", $1);

char temp[100];

sprintf(temp, "%s := %s\n", $$->code, $3->code);

sprintf($$->tac, "%s%s", $3->tac, temp);

sprintf($$->opttac, "%s%s", $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nMOV R0, %s\n",

$3->gen, temp, $3->code, $$->code);

}

;

Expr : Expr PLUS Term {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = %s + %s\n", $$->code, $1->code, $3->code);

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);

sprintf($$->opttac, "%s%s%s", $1->opttac, $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nADD %s, R0\nMOV R0, %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Expr MINUS Term {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = %s - %s\n", $$->code, $1->code, $3->code);

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);

sprintf($$->opttac, "%s%s%s", $1->opttac, $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nSUB %s, R0\nMOV R0, %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Term { $$ = $1; }

;

Term : Term STAR Factor {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = %s \* %s\n", $$->code, $1->code, $3->code);

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);

sprintf($$->opttac, "%s%s%s", $1->opttac, $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nMUL %s, R0\nMOV R0, %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Term DIV Factor {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = %s / %s\n", $$->code, $1->code, $3->code);

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);

sprintf($$->opttac, "%s%s%s", $1->opttac, $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nDIV %s, R0\nMOV R0, %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Term '%' Factor {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = %s %% %s\n", $$->code, $1->code, $3->code);

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);

sprintf($$->opttac, "%s%s%s", $1->opttac, $3->opttac, temp);

sprintf($$->gen, "%s%sMOV %s, R0\nMOD %s, R0\nMOV R0, %s\n",

$1->gen, $3->gen, $1->code, $3->code, $$->code);

}

| Factor { $$ = $1; }

;

Factor : INT {

$$ = makeNode();

sprintf($$->code, "%d", $1);

sprintf($$->tac, "");

sprintf($$->opttac, "");

sprintf($$->gen, "");

}

| FLOAT {

$$ = makeNode();

sprintf($$->code, "%.2f", $1);

sprintf($$->tac, "");

sprintf($$->opttac, "");

sprintf($$->gen, "");

}

| ID {

$$ = makeNode();

sprintf($$->code, "%s", $1);

sprintf($$->tac, "");

sprintf($$->opttac, "");

sprintf($$->gen, "");

}

| LPAREN Expr RPAREN { $$ = $2; }

| MINUS Factor %prec UMINUS {

$$ = makeNode();

int vc = varCounter();

sprintf($$->code, "T%d", vc);

char temp[100];

sprintf(temp, "%s = -%s\n", $$->code, $2->code);

sprintf($$->tac, "%s%s", $2->tac, temp);

sprintf($$->opttac, "%s%s", $2->opttac, temp);

sprintf($$->gen, "%sMOV %s, R0\nNEG R0\nMOV R0, %s\n",

$2->gen, $2->code, $$->code);

}

;

%%

void yyerror(const char \*s) {

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

flag = 1;

}

int yywrap() {

return 1;

}

int main(void) {

FILE \*file = fopen("q.c", "r");

if (!file) {

printf("Error opening file\n");

return 1;

}

yyin = file;

printf("\n----------LEXICAL ANALYSIS----------\n\n");

printf("Token - Type\n");

printf("--------------------\n");

yyparse();

if (flag) {

printf("\nSyntactically wrong\n");

}

fclose(file);

return 0;

}

**Q.c**

#include <stdio.h>

int a = 10;

int b = 20;

float c = 10.00;

while(1) {

if(a > b) {

c = a - b;

}

else {

c = b - a;

}

a = a - 1;

}

Output:







