**Name:** Paryusha Shah

**Roll No:** 22bce234

**Course:** Principles of Compiler Design

**Course Code:** 4CS501CC25

***Practical 8: Implement getreg() to allocate registers to variables in given three address code.***

**File:** expr.h

#ifndef EXPR\_H

#define EXPR\_H

typedef struct Expr {

    char\* place;

} Expr;

/\* Label & temporary counters \*/

static int temp\_count = 0;

static int label\_count = 0;

static char\* newtemp() {

    static char temp[10];

    sprintf(temp, "t%d", ++temp\_count);

    return strdup(temp);

}

static char\* newlabel() {

    static char label[10];

    sprintf(label, "L%d", ++label\_count);

    return strdup(label);

}

#endif

**File:** prog8.l

%{

#include "expr.h"

#include "y.tab.h"

#include <stdlib.h>

#include <string.h>

%}

%%

"if"            { return IF; }

"else"          { return ELSE; }

"while"         { return WHILE; }

"int"           { return INT; }

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

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

"{"                       { return LBRACE; }

"}"                       { return RBRACE; }

"("                       { return LPAREN; }

")"                       { return RPAREN; }

"="                       { return ASSIGN; }

";"                       { return SEMI; }

"+"                       { return PLUS; }

"-"                       { return MINUS; }

"\*"                       { return MUL; }

"/"                       { return DIV; }

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

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

%%

int yywrap() { return 1; }

**File:** prog8.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "expr.h"

int yylex(void);

extern FILE \*yyin;

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

int reg\_count = 0;

/\* Store register contents for printing later \*/

char\* registers[10];

/\* Register allocator \*/

char\* getreg() {

    static char reg[5];

    sprintf(reg, "R%d", reg\_count);

    if (!registers[reg\_count]) registers[reg\_count] = strdup("EMPTY");

    reg\_count++;

    if (reg\_count >= 10) reg\_count = 0;  // wrap-around reuse

    return strdup(reg);

}

/\* Update register content \*/

void setreg(char\* reg, char\* val) {

    int index = reg[1] - '0'; // get register index

    if (index >= 0 && index < 10) {

        if (registers[index]) free(registers[index]);

        registers[index] = strdup(val);

    }

}

%}

%union {

    int num;

    char\* id;

    Expr\* expr;

    char\* label;

}

%token <id> ID

%token <num> NUM

%token INT IF ELSE WHILE

%token PLUS MINUS MUL DIV ASSIGN SEMI LPAREN RPAREN LBRACE RBRACE

%type <expr> E

%type <label> S

%left PLUS MINUS

%left MUL DIV

%%

program:

    stmt\_list {

        printf("\n=== REGISTER CONTENTS ===\n");

        for (int i = 0; i < 10; i++) {

            if (registers[i])

                printf("R%d = %s\n", i, registers[i]);

            else

                printf("R%d = EMPTY\n", i);

        }

    }

    ;

stmt\_list:

    stmt\_list S

    | /\* empty \*/

    ;

S:

    ID ASSIGN E SEMI {

        char\* r = getreg();

        printf("%s = %s\n", r, $3->place);

        setreg(r, $3->place);

        printf("%s = %s\n", $1, r);

    }

    | IF LPAREN E RPAREN LBRACE stmt\_list RBRACE ELSE LBRACE stmt\_list RBRACE {

        char\* L1 = newlabel();

        char\* L2 = newlabel();

        printf("ifFalse %s goto %s\n", $3->place, L1);

        printf("goto %s\n", L2);

        printf("%s:\n", L1);

        printf("%s:\n", L2);

    }

    | WHILE LPAREN E RPAREN LBRACE stmt\_list RBRACE {

        char\* Lstart = newlabel();

        char\* Lend = newlabel();

        printf("%s:\n", Lstart);

        printf("ifFalse %s goto %s\n", $3->place, Lend);

        printf("goto %s\n", Lstart);

        printf("%s:\n", Lend);

    }

    ;

E:

    E PLUS E {

        $$ = malloc(sizeof(Expr));

        char\* r1 = getreg();

        char\* r2 = getreg();

        printf("%s = %s\n", r1, $1->place);

        printf("%s = %s\n", r2, $3->place);

        printf("ADD %s, %s\n", r1, r2);

        $$->place = newtemp();

        printf("%s = %s\n", $$->place, r1);

        setreg(r1, $$->place);

    }

    | E MINUS E {

        $$ = malloc(sizeof(Expr));

        char\* r1 = getreg();

        char\* r2 = getreg();

        printf("%s = %s\n", r1, $1->place);

        printf("%s = %s\n", r2, $3->place);

        printf("SUB %s, %s\n", r1, r2);

        $$->place = newtemp();

        printf("%s = %s\n", $$->place, r1);

        setreg(r1, $$->place);

    }

    | E MUL E {

        $$ = malloc(sizeof(Expr));

        char\* r1 = getreg();

        char\* r2 = getreg();

        printf("%s = %s\n", r1, $1->place);

        printf("%s = %s\n", r2, $3->place);

        printf("MUL %s, %s\n", r1, r2);

        $$->place = newtemp();

        printf("%s = %s\n", $$->place, r1);

        setreg(r1, $$->place);

    }

    | E DIV E {

        $$ = malloc(sizeof(Expr));

        char\* r1 = getreg();

        char\* r2 = getreg();

        printf("%s = %s\n", r1, $1->place);

        printf("%s = %s\n", r2, $3->place);

        printf("DIV %s, %s\n", r1, r2);

        $$->place = newtemp();

        printf("%s = %s\n", $$->place, r1);

        setreg(r1, $$->place);

    }

    | LPAREN E RPAREN {

        $$ = malloc(sizeof(Expr));

        $$->place = $2->place;

    }

    | NUM {

        $$ = malloc(sizeof(Expr));

        $$->place = malloc(20);

        sprintf($$->place, "%d", $1);

    }

    | ID {

        $$ = malloc(sizeof(Expr));

        $$->place = strdup($1);

    }

    ;

%%

int main() {

    FILE \*fp = fopen("input.c", "r");

    if(!fp) { perror("Cannot open file input.c"); return 1; }

    yyin = fp;

    yyparse();

    fclose(fp);

    return 0;

}

**File:** input.c

a = b + c \* 2;

if (a) {

    x = y + 1;

} else {

    y = y + 2;

}

while (x) {

    x = x - 1;

}

**Commands for run program:**



