**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 1**

#include <stdio.h>

#include <string.h>

int state = 0, bcount = 0;

FILE \*fp;

int reset() { bcount = 0; state = 0; fseek(fp, -1, SEEK\_CUR); }

int isDigi(char ch) {

if(ch >= '0' && ch <= '9') return 1;

return 0;

}

int isIden(char ch) {

if((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || ch == '\_') return 1;

return 0;

}

int isOper(char ch) {

if(ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '%' || ch == '=') return 1;

return 0;

}

int isSymb(char ch) {

if(ch == '{' || ch == '}' || ch == '(' || ch == ')' || ch == ';' || ch == ',') return 1;

return 0;

}

int isWhit(char ch) {

if(ch == ' ' || ch == '\t' || ch == '\n') return 1;

return 0;

}

int isKeyw(char \*b) {

if(!strcmp(b, "main") || !strcmp(b, "void") || !strcmp(b, "int")) return 1;

return 0;

}

void main() {

char buffer[128], ch;

fp = fopen("input\_1.txt", "r");

while((ch = fgetc(fp)) != EOF) {

buffer[bcount++] = ch;

switch(state) {

case 0:

if(isDigi(ch)) state = 1;

if(isIden(ch)) state = 2;

if(isOper(ch)) state = 3;

if(isSymb(ch)) state = 5;

if(isWhit(ch)) bcount = 0;

break;

case 1:

if(!isDigi(ch)) {

buffer[bcount - 1] = '\0';

printf("Number: \t%s\n", buffer); reset();

}

break;

case 2:

if(!isIden(ch) && !isDigi(ch)) {

buffer[bcount - 1] = '\0';

if(isKeyw(buffer)) printf("Keyword: \t%s\n", buffer);

else printf("Identifier: \t%s\n", buffer);

reset();

}

break;

case 3:

if(ch == '/' && buffer[bcount - 2] == '/') state = 5;

if(!isOper(ch)) {

buffer[bcount - 1] = '\0';

printf("Operator: \t%s\n", buffer); reset();

}

break;

case 4:

if(ch == '\n') reset();

case 5:

buffer[bcount - 1] = '\0';

printf("Symbol: \t%s\n", buffer); reset(); break;

}

}

fclose(fp);

}

**// Input File**

void main()

{

// hello world

int a = 10, b;

a = a \* b / 5; // Simple Operation comment

}

**// Output**

Keyword: void

Keyword: main

Symbol: (

Symbol: )

Symbol: {

Keyword: int

Identifier: a

Operator: =

Number: 10

Symbol: ,

Identifier: b

Symbol: ;

Identifier: a

Operator: =

Identifier: a

Operator: \*

Identifier: b

Operator: /

Number: 5

Symbol: ;

Symbol: }

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 2**

%{

#include <stdio.h>

int comment = 0;

%}

%%

"//".\*\n { comment++; }

[ \n\t] { ; }

[+-]?[0-9]+(\.[0-9]+)?(E[+-]?[0-9]+) { printf("%s\tExponent Number\n", yytext); }

([+-]?[0-9]+)\.[0-9]+ { printf("%s\tFloating Number\n", yytext); }

[+-]?[0-9]+ { printf("%s\tNumber\n", yytext); }

(void|main|printf|int|float) { printf("%s\tReserved Keyword\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("%s\tLiteral\n", yytext); }

[=\*/+\-%] { printf("%s\tArithmetic Operator\n", yytext); }

(==|<|<=|>|>=|!=) { printf("%s\tRelational Operator\n", yytext); }

[(){};,] {printf("%s\tSpecial Operator\n", yytext); }

%%

void main() {

yyin = fopen("input.c", "r"); yylex();

printf("\n%d Comments Ignored\n", comment); fclose(yyin);

}

**// Input File**

void main(){

// hello world

int a = 7, b = 7.35, c = 7.35E2 \* 7E10;

a = a \* b;

}

**// Output**

void Reserved Keyword

main Reserved Keyword

( Special Operator

) Special Operator

{ Special Operator

int Reserved Keyword

a Literal

= Arithmetic Operator

7 Number

, Special Operator

b Literal

= Arithmetic Operator

7.35 Floating Number

, Special Operator

c Literal

= Arithmetic Operator

7.35E2 Exponent Number

\* Arithmetic Operator

7E10 Exponent Number

; Special Operator

a Literal

= Arithmetic Operator

a Literal

\* Arithmetic Operator

b Literal

; Special Operator

} Special Operator

1 Comments Ignored

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 3**

%{

#include <stdio.h>

int lines = 0, char\_count = 0, words = 0;

%}

%%

\n { lines++; char\_count++; }

[a-zA-Z\_]\* {words++; char\_count += yyleng; }

. { char\_count++; }

%%

void main() {

yyin = fopen("input.txt", "r"); yylex();

printf("Statistics:\n%d\tLines\n%d\tCharacters\n%d\tWords\n", lines, char\_count, words);

fclose(yyin);

}

**// Input File**

hello World

aeiou

abcf gejk

**// Output**

Statistics:

4 Lines

32 Characters

5 Words

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 4**

%{

#include <stdio.h>

%}

%%

abc { printf("ABC"); }

.|\n { printf("%s", yytext); }

%%

void main() {

yyin = fopen("input.txt", "r"); yylex();

fclose(yyin);

}

**// Input**

helloabccdabfc

**// Output**

helloABCcdabfc

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 5**

%{

#include <stdio.h>

int vowels = 0, cons = 0;

%}

%%

[aeiouAEIOU] { vowels++; }

[a-zA-Z] { cons++; }

(.|\n) { ; }

%%

void main() {

yyin = fopen("input.txt", "r"); yylex();

printf("Statistics:\n%d\tVowels\n%d\tConsonants\n", vowels, cons);

fclose(yyin);

}

**// Input**

hello World

aeiou

abcf gejk

**// Output**

Statistics:

10 Vowels

13 Consonants

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 6**

**// Lex File**

%{

#include <stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ { return NUM; }

[a-zA-Z] { return ID; }

[+\-\*/()] { return yytext[0]; }

. { return other; }

\n { return '\n'; }

%%

**// YACC File**

%{

#include <stdio.h>

int yylex(); void yyerror();

%}

%token NUM ID other

%left '+' '-'

%left '\*' '/'

%%

start : T '\n' { printf("Valid Arithmetic Expression\n"); return 0; };

T : T '+' T

| T '-' T

| T '\*' T

| T '/' T

| '(' T ')'

| NUM

| ID

;

%%

void yyerror() {

printf("Error. Failed to parse.\n");

}

void main() {

printf("Enter arithmetic expression: ");

yyparse();

}

**// Output #1**

Enter arithmetic expression: (a+b\*(a/2+3))

Valid Arithmetic Expression

**// Output #2**

Enter arithmetic expression: (a+b(c+/

Error. Failed to parse.

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 7**

**// Lex File**

%{

#include <stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9] { return digit; }

[a-zA-Z] { return alpha; }

. { return other; }

\n { return '\n'; }

%%

**// YACC File**

%{

#include <stdio.h>

int yylex();

void yyerror();

%}

%token digit alpha other

%%

start : T '\n' { printf("Valid Computer Identifier\n"); return 0; };

T : alpha U

;

U : U alpha

| U digit

|

;

%%

void yyerror() {

printf("Error. Failed to parse.\n");

}

void main() {

printf("Enter identifier: ");

yyparse();

}

**// Output #1**

Enter identifier: abc123

Valid Computer Identifier

**// Output #2**

Enter identifier: 12ac

Error. Failed to parse.

**// Output #3**

Enter identifier: abc$#123

Error. Failed to parse.

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 8**

**// Lex File**

%{

#include <stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ { sscanf(yytext, "%d", &yylval); return NUM; }

[+\-\*/()] { return yytext[0]; }

. { return other; }

\n { return '\n'; }

%%

**// YACC File**

%{

#include <stdio.h>

int yylex(); void yyerror();

%}

%token NUM other

%left '+' '-'

%left '\*' '/'

%%

start : T '\n' { printf("Result: %d\n", $$); return 0; };

T : T '+' T { $$ = $1 + $3; }

| T '-' T { $$ = $1 - $3; }

| T '\*' T { $$ = $1 \* $3; }

| T '/' T { $$ = $1 / $3; }

| '(' T ')' { $$ = $2; }

| NUM { $$ = $1; }

;

%%

void yyerror() {

printf("Error. Failed to parse.\n");

}

void main() {

printf("Enter arithmetic expression: ");

yyparse();

}

**// Output #1**

Enter arithmetic expression: 2+3\*5

Result: 17

**// Output #2**

Enter arithmetic expression: (2+3)\*5+(6\*7+(2\*3))

Result: 73

**// Output #3**

Enter arithmetic expression: 2+3/

Error. Failed to parse.

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 9**

#include <stdio.h>

#define st 10 // Maximum handle limit

int arr[st][st];

int visited[st], seen[st];

void eps(int node) {

if(visited[node]) return;

visited[node] = 1;

printf("%d ", node);

for(int i = 0; i < st; i++) if(arr[node][i]) eps(i);

}

void main() {

FILE \*fp = fopen("td.txt", "r");

int s1, s2; char in;

while(fscanf(fp, "%d %c %d", &s1, &in, &s2) != EOF) {

seen[s1] = 1;

seen[s2] = 1;

if(in == 'e') arr[s1][s2] = 1;

}

fclose(fp);

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

if(!seen[i]) continue;

printf("E-CLOSURE(%d) = ", i);

for(int j = 0; j < st; j++) visited[j] = 0;

eps(i);

printf("\n");

}

}

**// Transition Diagram**

**// Input File**

0 a 0

0 e 1

1 b 1

1 e 2

2 c 2

**// Output**

E-CLOSURE(0) = 0 1 2

E-CLOSURE(1) = 1 2

E-CLOSURE(2) = 2

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 10**

#include <stdio.h>

#define st 10 // MAX STATES

#define ch 10 // MAX INPUTS

// DS

int tra[st][st][ch], inv[st][ch][st], nfa[st][ch][st], ecl[st][st];

char inp[st];

int vis[st], see[st], inpcnt = 0;

int mapInp(char in) {

for(int i = 0; i <= inpcnt; i++) if(inp[i] == in) return i;

inp[inpcnt++] = in; return inpcnt - 1;

}

void eps(int org, int s) {

if(vis[s]) return;

vis[s] = 1;

ecl[org][s] = 1;

for(int i = 0; i < st; i++) if(tra[s][i][0]) eps(org, i);

}

void main() {

inp[inpcnt++] = 'e';

FILE \*fp = fopen("td.txt", "r");

int s1, s2; char in;

while(fscanf(fp, "%d %c %d", &s1, &in, &s2) != EOF) {

see[s1] = 1;

see[s2] = 1;

int index = mapInp(in);

tra[s1][s2][index] = 1;

inv[s1][index][s2] = 1;

}

fclose(fp);

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

for(int j = 0; j < st; j++) vis[j] = 0;

eps(i, i);

}

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

if(!see[i]) continue;

for(int j = 1; j < inpcnt; j++) {

for(int k = 0; k < st; k++) {

if(!see[k]) continue;

if(ecl[i][k]) {

for(int l = 0; l < st; l++) {

if(inv[k][j][l]) {

for(int m = 0; m < st; m++) {

if(ecl[l][m]) {

nfa[i][j][m] = 1;

}

}

}

}

}

}

}

}

printf("\nNFA Transitions\n");

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

for(int j = 1; j < inpcnt; j++) {

for(int k = 0; k < st; k++) {

if(nfa[i][j][k] && see[i] && see[k]) {

printf("%d - %c - %d\n", i, inp[j], k);

}

}

}

}

}

**// Input Transition Diagram**

**// Output Transition Diagram**

**// Input File**

0 1 1

0 e 2

1 1 0

2 0 3

2 1 4

3 0 2

4 0 2

**// Output**

NFA Transitions

0 - 1 - 1

0 - 1 - 4

0 - 0 - 3

1 - 1 - 0

1 - 1 - 2

2 - 1 - 4

2 - 0 - 3

3 - 0 - 2

4 - 0 – 2

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 11**

#include <stdio.h>

#define max\_tc 20 // MAXIMUM TRANSITION COUNT

#define max\_tl 10 // MAXIMUM TRANSITION LENGTH

#define max\_nt 10 // MAXIMUM ALLOWED NON TERMINAL

#define max\_te 10 // MAXIMUM ALLOWED TERMINAL

char trans[max\_tc][max\_tl];

int trcnt = -1;

char nonterm[max\_nt], term[max\_te];

int ntcnt = 0, tecnt;

int res[max\_te];

int isNonTerminal(char ch) { return (ch >= 'A' && ch <='Z') ? 1 : 0; }

int mapSymbol(char ch) {

if(isNonTerminal(ch)) {

for(int i = 0; i < ntcnt; i++)

if(nonterm[i] == ch)

return i;

nonterm[ntcnt++] = ch;

return ntcnt - 1;

}

for(int i = 0; i < tecnt; i++)

if(term[i] == ch)

return i;

term[tecnt++] = ch;

return tecnt - 1;

}

void analyze(FILE \*fp) {

char s[32], left;

int i;

while(fscanf(fp, "%s", s) != EOF) {

int tlcnt = 0;

mapSymbol(s[0]);

for(i = 1; i < 32 && s[i] != '\0'; i++) {

if(s[i] == '=' || s[i] == '|') {

if(i == '|')

trans[trcnt][tlcnt++] = '\0';

trcnt++; tlcnt = 0;

trans[trcnt][tlcnt++] = s[0];

trans[trcnt][tlcnt++] = '=';

continue;

}

mapSymbol(s[i]);

trans[trcnt][tlcnt++] = s[i];

}

trans[trcnt][tlcnt++] = '\0';

}

trcnt++;

printf("Transitions Read: (%d)\n", trcnt + 1);

for(i = 0; i < trcnt; i++) printf(" %s\n", trans[i]);

printf("Non-Terminals Encountered: (%d)\n ", ntcnt);

for(i = 0; i < ntcnt; i++) printf("%c ", nonterm[i]);

printf("\nTerminals Encountered: (%d)\n ", ntcnt);

for(i = 0; i < tecnt; i++) printf("%c ", term[i]);

fclose(fp); printf("\n\n");

}

int first(char T) {

int i, j, flag = 0;

for(i = 0; i < trcnt; i++) {

if(trans[i][0] == T) {

for(j = 2; j < max\_tl && trans[i][j] != T && trans[i][j] != '\0'; j++) {

if(isNonTerminal(trans[i][j])) {

if(!first(trans[i][j])) {

flag = 0;

break;

}

flag = 1;

}

else {

res[mapSymbol(trans[i][j])] = 1;

if(trans[i][j] == '#') flag = 1;

break;

}

}

}

}

return flag;

}

void follow(char T) {

int i, j, k;

for(i = 0; i <= trcnt; i++) {

for(j = 2; j < max\_tl && trans[i][j] != '\0'; j++) {

if(trans[i][j] == T) {

for(k = j + 1; k < max\_tl && trans[i][k] != '\0'; k++) {

if(isNonTerminal(trans[i][k])) {

if(!first(trans[i][k]))

break;

}

else {

if(trans[i][k] != '#') {

res[mapSymbol(trans[i][k])] = 1;

break;

}

}

}

if(trans[i][k] == '\0' && trans[i][0] != T) follow(trans[i][0]);

}

}

}

}

void main() {

FILE \*fp = fopen("gram.txt", "r");

analyze(fp);

int i, j;

for(i = 0; i < ntcnt; i++) {

for(j = 0; j < tecnt; j++) res[j] = 0;

first(nonterm[i]);

printf("First(%c) = { ", nonterm[i]);

for(j = 0; j < tecnt; j++) if(res[j]) printf("%c, ", term[j]);

printf("}\n");

}

sprintf(trans[trcnt], "%c=%c$", trans[0][0], trans[0][0]);

for(i = 0; i < ntcnt; i++) {

for(j = 0; j < tecnt; j++) res[j] = 0;

follow(nonterm[i]);

printf("Follow(%c) = { ", nonterm[i]);

for(j = 0; j < tecnt; j++) if(res[j] && term[j] != '#') printf("%c, ", term[j]);

printf("}\n");

}

}

**// Output**

Transitions Read: (9)

E=TR

R=+TR

R=#

T=FY

Y=\*FY

Y=#

F=(E)

F=i

Non-Terminals Encountered: (5)

E T R F Y

Terminals Encountered: (5)

+ # \* ( ) i

First(E) = { (, i, }

First(T) = { (, i, }

First(R) = { +, #, }

First(F) = { (, i, }

First(Y) = { #, \*, }

Follow(E) = { ), $, }

Follow(T) = { +, ), $, }

Follow(R) = { ), $, }

Follow(F) = { +, \*, ), $, }

Follow(Y) = { +, ), $, }

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 12**

#include <stdio.h>

#include <ctype.h>

char s[64];

int cnt = 0;

int E(), E\_(), T(), T\_(), F();

int main() {

printf("Enter Expression to Validate: ");

scanf("%s", s);

if(E())

printf("Expression is Valid.\n");

else

printf("Expression is Invalid.\n");

return 0;

}

int E() {

if(T())

if(E\_())

return 1;

return 0;

}

int E\_() {

if(s[cnt] == '+') {

cnt++;

if(T())

if(E\_())

return 1;

return 0;

}

return 1;

}

int T() {

if(F())

if(T\_())

return 1;

return 0;

}

int T\_() {

if(s[cnt] == '\*') {

cnt++;

if(F())

if(T\_())

return 1;

return 0;

}

return 1;

}

int F() {

if(s[cnt] == '(') {

cnt++;

if(E()) {

if(s[cnt] == ')') {

cnt++;

return 1;

}

}

}

else if(isalnum(s[cnt])) {

cnt++;

return 1;

}

return 0;

}

**// Input Grammar**

E -> E+T|T

T -> T\*F|F

F -> (E)|id

// Without left recursion and after left factoring

E -> TE'

E' -> +TE'|∈

T -> FT'

T' -> \*FT'|∈

F -> (E)|id

**// Output #1**

Enter Expression to Validate: a+b

Expression is Valid.

**// Output #2**

Enter Expression to Validate: a+(b\*c)

Expression is Valid.

**// Output #3**

Enter Expression to Validate: a+(b++c)

Expression is Invalid.

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 13**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

char stk[64], inp[64];

int scnt = 0, icnt = 0;

int reduce(char ch, int index, char \*trans) {

scnt = index;

stk[scnt] = ch;

stk[scnt + 1] = '\0';

printf("%s\t|\t%s\n", stk, trans);

return 1;

}

int check() {

if(stk[scnt] == 'E' && stk[scnt - 1] == '+' && stk[scnt - 2] == 'E') return reduce('E', scnt - 2, "E -> E + E");

if(stk[scnt] == 'E' && stk[scnt - 1] == '\*' && stk[scnt - 2] == 'E') return reduce('E', scnt - 2, "E -> E \* E");

if(stk[scnt] == ')' && stk[scnt - 1] == 'E' && stk[scnt - 2] == '(') return reduce('E', scnt - 2, "E -> ( E )");

if(isalnum(stk[scnt]) && stk[scnt] != 'E') return reduce('E', scnt, "E -> id");

return 0;

}

int main() {

printf("Enter Expression to Validate: ");

scanf("%s", inp);

stk[scnt] = '$';

stk[scnt + 1] = '\0';

printf("STACK\t|\tREDUCTION\n");

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

for(int i = 0; i < 64 && inp[i] != '\0'; i++) {

stk[++scnt] = inp[icnt++];

printf("%s\t|\n", stk);

while(check());

}

if(!strcmp(stk, "$E"))

printf("\nValid Expression.\n");

else

printf("\nInvalid Expression.\n");

return 0;

}

**// Input Grammar**

// E -> E+E|E\*E|(E)|id

**// Output**

Enter Expression to Validate: a+(b\*c)

STACK | REDUCTION

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

$a |

$E | E -> id

$E+ |

$E+( |

$E+(b |

$E+(E | E -> id

$E+(E\* |

$E+(E\*c |

$E+(E\*E | E -> id

$E+(E | E -> E \* E

$E+(E)E |

$E+E | E -> ( E )

$E | E -> E + E

Valid Expression.

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 14**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

char stk[64];

int top = -1;

char expr[32], pex[32];

int pcnt = 0;

void push(char ch) { stk[++top] = ch; }

char pop() { return stk[top--]; } // Assuming valid expression

int rank(char ch, int stk\_rank) {

if(ch == '+' || ch == '-') return 1;

if(ch == '\*' || ch == '/') return 2;

if(ch == '^') return stk\_rank ? 3 : 4;

if(ch == '(') return stk\_rank ? 0 : 5;

}

void inpos() {

char ch, temp;

push('('); strcat(expr, ")");

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

ch = expr[i];

if(isalpha(ch))

pex[pcnt++] = ch;

else if(ch == ')')

while((temp = pop()) != '(') pex[pcnt++] = temp;

else {

if(ch != '(') while(rank(stk[top], 1) >= rank(ch, 0)) pex[pcnt++] = pop();

push(ch);

}

}

printf("\nPostfix Expression: %s\n\n", pex);

}

void icg() {

char ch, a1, a2, res = '0';

printf("OPR\tA1\tA2\tRES\n");

for(int i = 0; pex[i] != 0; i++) {

ch = pex[i];

if(isalpha(ch)) push(ch);

else {

a2 = pop(); a1 = pop();

printf("%c\t", ch);

if(isdigit(a1)) printf("t");

printf("%c\t", a1);

if(isdigit(a2)) printf("t");

printf("%c\tt%c\n", a2, res);

push(res++);

}

}

}

int main() {

printf("Enter Expression: ");

scanf("%s", expr);

inpos(); icg();

return 0;

}

**// Output**

Enter Expression: a+b\*c

Postfix Expression: abc\*+

OPR A1 A2 RES

\* b c t0

+ a t0 t1

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 15**

#include <stdio.h>

#include <string.h>

struct statement {

char op[2], a1[4], a2[4], res[4];

int flag;

};

struct statement st[32];

int cnt = 0;

void replace(char \*targ, char \*repl, int i) {

// replace targ with repl

st[i].flag = 1;

for(int j = i + 1; j < cnt; j++) {

if(!strcmp(targ, st[j].a1)) sprintf(st[j].a1, "%s", repl);

if(!strcmp(targ, st[j].a2)) sprintf(st[j].a2, "%s", repl);

}

}

int main() {

FILE \*fp;

fp = fopen("in.txt", "r");

while(fscanf(fp, "%s %s %s %s", st[cnt].op, st[cnt].a1, st[cnt].a2, st[cnt].res) != EOF) cnt++;

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

if(st[i].op[0] == '=')

replace(st[i].res, st[i].a1, i);

}

printf("Constant propagated code: \n");

printf("OPR\tA1\tA2\tRES\n");

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

if(st[i].flag == 1) continue;

printf("%s\t%s\t%s\t%s\n", st[i].op, st[i].a1, st[i].a2, st[i].res);

}

return 0;

}

**// Input File**

= 3 - a

+ a b t1

+ a c t2

+ t1 t2 t3

**// Output**

Constant propagated code:

OPR A1 A2 RES

+ 3 b t1

+ 3 c t2

+ t1 t2 t3

**// Emmanuel Jojy**

**// 53 S7 CSE A**

**// Experiment 16**

#include <stdio.h>

int main() {

char op[2], arg1[5], arg2[5], res[5];

FILE \*fp; fp = fopen("in.txt", "r");

while (fscanf(fp, "%s%s%s%s", op, arg1, arg2, res) != EOF) {

printf("MOV R0,%s\n", arg1);

switch (op[0]) {

case '+': printf("ADD R0,%s\n", arg2); break;

case '-': printf("SUB R0,%s\n", arg2); break;

case '\*': printf("MUL R0,%s\n", arg2); break;

case '/': printf("DIV R0,%s\n", arg2); break;

}

printf("MOV %s,R0\n", res);

}

fclose(fp); return 0;

}

**// Input File**

+ a b t1

\* c d t2

- t1 t2 t

= t ? x

**// Output**

MOV R0,a

ADD R0,b

MOV t1,R0

MOV R0,c

MUL R0,d

MOV t2,R0

MOV R0,t1

SUB R0,t2

MOV t,R0

MOV R0,t

MOV x,R0