**Pcd lab exam**

1. **Implementation of lexical analyzer using C programming**

**PROGRAM:**

#include <string.h>

#include <ctype.h>

#include <stdio.h>

void keyword(char str[10]) {

char keywords[10][10] = {"int", "float", "char", "while", "do", "for", "if"};

for (int i = 0; i < 7; i++) {

if (!strcmp(keywords[i], str)) {

printf("\n%s is a keyword", str);

return;

}

}

printf("\n%s is an identifier", str);

}

void main() {

FILE \*f1, \*f2, \*f3, \*f4;

char c, str[10], st1[10];

int num[100], tokenvalue = 0, i = 0, j = 0, k = 0;

printf("\nEnter the C program:\n");

f1 = fopen("input", "w");

while ((c = getchar()) != EOF) {

putc(c, f1);

}

fclose(f1);

f1 = fopen("input", "r");

f2 = fopen("identifier", "w");

f3 = fopen("specialchar", "w");

f4 = fopen("operators", "w");

while ((c = getc(f1)) != EOF) {

if (isdigit(c)) {

tokenvalue = c - '0';

c = getc(f1);

while (isdigit(c)) {

tokenvalue = tokenvalue \* 10 + (c - '0');

c = getc(f1);

}

num[i++] = tokenvalue;

ungetc(c, f1);

} else if (isalpha(c)) {

putc(c, f2);

c = getc(f1);

while (isalnum(c) || c == '\_' || c == '$') {

putc(c, f2);

c = getc(f1);

}

putc(' ', f2);

ungetc(c, f1);

} else if (c == '+' || c == '-' || c == '\*' || c == '<' || c == '>' || c == '/' || c == '&' || c == '%' || c == '^' || c == '=') {

putc(c, f4);

} else {

putc(c, f3);

}

}

fclose(f4);

fclose(f2);

fclose(f3);

fclose(f1);

printf("\nThe constants are: ");

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

printf("%d ", num[j]);

}

printf("\n");

f2 = fopen("identifier", "r");

k = 0;

printf("The keywords and identifiers are:\n");

while ((c = getc(f2)) != EOF) {

if (c != ' ') {

str[k++] = c;

} else {

str[k] = '\0';

keyword(str);

k = 0;

}

}

fclose(f2);

f3 = fopen("specialchar", "r");

printf("\nSpecial characters are: ");

while ((c = getc(f3)) != EOF) {

printf("%c ", c);

}

fclose(f3);

f4 = fopen("operators", "r");

printf("Operators are: ");

while ((c = getc(f4)) != EOF) {

printf("%c ", c);

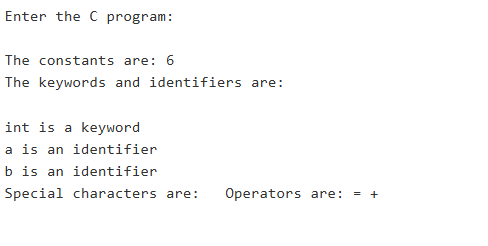
}

printf("\n");

fclose(f4);

}

**OUPUT:**

****

**INSTRUCTIONS:**

PASTE THE ABOVE PROGRAM IN ONLINE GDB COMPILER AND PASTE THE LINE LIKE

Int a=b+6 in text input box in the down panel and click run.. output will display as above..

**2.Implementation of SYMBOLTABLE**

**PROGRAM:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

struct symtab {

int lineno;

char var[25], dt[25], val[10];

} sa[20];

void main() {

int i = 0, max = 0, line = 0;

char s[25], typ[25], gar[] = "garbage";

FILE \*fn;

fn = fopen("input.txt", "r");

if (fn == NULL) {

printf("Error: Unable to open file.\n");

return;

}

printf("\n\nSYMBOL TABLE MANAGEMENT\n\n");

printf("Variable\tDatatype\tLine.no.\tValue\n");

while (!feof(fn)) {

fscanf(fn, "%s", s);

if ((strcmp(s, "int") == 0) || (strcmp(s, "float") == 0) || (strcmp(s, "char") == 0)) {

strcpy(typ, s);

line++;

while (strcmp(s, ";") != 0) {

i++;

max = i;

sa[i].lineno = line;

fscanf(fn, "%s", s);

strcpy(sa[i].var, s);

strcpy(sa[i].dt, typ);

fscanf(fn, "%s", s);

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

fscanf(fn, "%s", s);

strcpy(sa[i].val, s);

fscanf(fn, "%s", s);

} else {

strcpy(sa[i].val, gar);

}

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

continue;

} else {

break;

}

}

}

}

for (i = 1; i <= max; i++) {

printf("\n%s\t\t%s\t\t%d\t\t%s\n", sa[i].var, sa[i].dt, sa[i].lineno, sa[i].val);

}

fclose(fn);

}

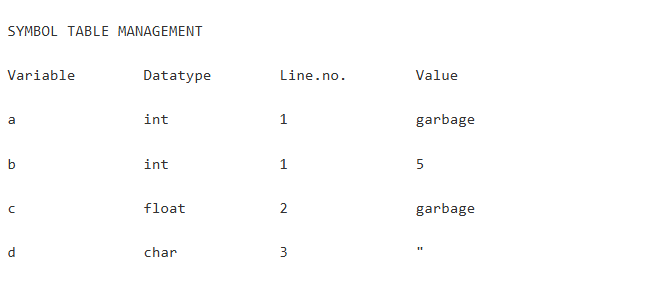
**INPUT.TXT:**

int a , b = 5 ;

float c ;

char d = " a " ;

**OUTPUT:**



**INSTRUCTIONS**

**1.**open online gdb compiler and paste the above program in code editor and at the left top panel,click on add file and give the filename as “input.txt” and paste the input.txt code in it and then run the program.

**3. Implementation of front end of a compiler**

**PROGRAM**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int ag = 0, z = 1;

void main() {

char a[50], id[50], b[50], op[50], mov[] = "MOVF", mul[] = "MULF", div[] = "DIVF", add[] = "ADDF", sub[] = "SUBF", ti = 0;

int i = 0, j = 0, k = 0, len = 0, s = 0, e = 0, r = 1, count;

FILE \*fp;

fp = fopen("out.txt", "w");

printf("\nEnter the code: ");

fgets(a, sizeof(a), stdin); // Modified to use fgets for complex input handling

strcpy(b, a);

len = strlen(a);

for (i = 0; i < strlen(b); i++) {

if (b[i] == '\*' || b[i] == '/') {

for (j = i - 1; b[j] != '-' && b[j] != '+' && b[j] != '\*' && b[j] != '/' && b[j] != '='; j--);

k = j + 1;

count = 0;

printf("\nt%d=", ti++);

for (j = j + 1; count < 2 && b[j] != '\0'; j++) {

if (b[j + 1] == '+' || b[j + 1] == '-' || b[j + 1] == '\*' || b[j + 1] == '/')

count++;

printf("%c", b[j]);

}

b[k++] = 't';

b[k++] = ti - 1 + 48;

for (j = j, k = k; k < strlen(b); k++, j++)

b[k] = b[j];

i = 0;

}

}

for (i = 0; i < strlen(b); i++) {

if (b[i] == '+' || b[i] == '-') {

for (j = i - 1; b[j] != '-' && b[j] != '+' && b[j] != '='; j--);

k = j + 1;

count = 0;

printf("\nt%d=", ti++);

for (j = j + 1; count < 2 && b[j] != '\0'; j++) {

if (b[j + 1] == '+' || b[j + 1] == '-')

count++;

printf("%c", b[j]);

}

b[k++] = 't';

b[k++] = ti - 1 + 48;

for (j = j, k = k; k < strlen(b); k++, j++)

b[k] = b[j];

}

}

printf("\n%s", b);

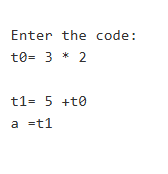
fclose(fp);

}

**INPUT(TEXT)**

a = 5 + 3 \* 2

**OUTPUT**

****

**INSTRUCTIONS**

1.Paste the c program above in the online gdb compiler and in unput text box give the input as “a = 5 + 3 \* 2” and click on run button .output will shown like above..

**4. Implementation of the back end of the compiler**

**PROGRAM:**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int ag = 0, z = 1;

void main() {

char a[50], id[50], mov[] = "MOVF", mul[] = "MULF", div[] = "DIVF", add[] = "ADDF", sub[] = "SUBF";

int i = 0, j = 0, len = 0, s = 0, e = 0, r = 1;

FILE \*fp;

fp = fopen("out.txt", "w");

printf("\nEnter the code: ");

fgets(a, sizeof(a), stdin); // Using fgets instead of gets for security

len = strlen(a);

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

if (a[i] == '=') {

for (j = i; j < len; j++) {

if (a[j] == 'i') { // Looking for 'id'

fprintf(fp, "\n%s ", mov);

fprintf(fp, "%c%c%c,R%d", a[j], a[j + 1], a[j + 2], r++);

}

}

}

else if ((a[i] <= 57) && (a[i] >= 48)) { // If the character is a digit

if ((a[i + 1] <= 57) && (a[i + 1] >= 48)) { // If next character is also a digit

fprintf(fp, "\n%s #%c%c,R%d", mov, a[i], a[i + 1], r++);

}

}

}

for (i = len - 1; i >= 0; i--) {

if (a[i] == '+') {

fprintf(fp, "\n%s ", add);

e = a[i - 1];

e--;

s = e;

if (a[i + 1] == 'i')

fprintf(fp, "R%c,R%d", e, r - 1);

}

else if (a[i] == '-') {

fprintf(fp, "\n%s ", sub);

e = a[i - 1];

e--;

s = e;

if (a[i + 1] == 'i')

fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);

else

fprintf(fp, "R%c,R%d", e, r - 1);

}

else if (a[i] == '\*') {

fprintf(fp, "\n%s ", mul);

e = a[i - 1];

e--;

s = e;

if (a[i + 1] == 'i')

fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);

else

fprintf(fp, "R%c,R%d", e, r - 1);

}

else if (a[i] == '/') {

fprintf(fp, "\n%s ", div);

e = a[i - 1];

e--;

s = e;

if (a[i + 1] == 'i')

fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);

else

fprintf(fp, "R%c,R%d", e, r - 1);

}

}

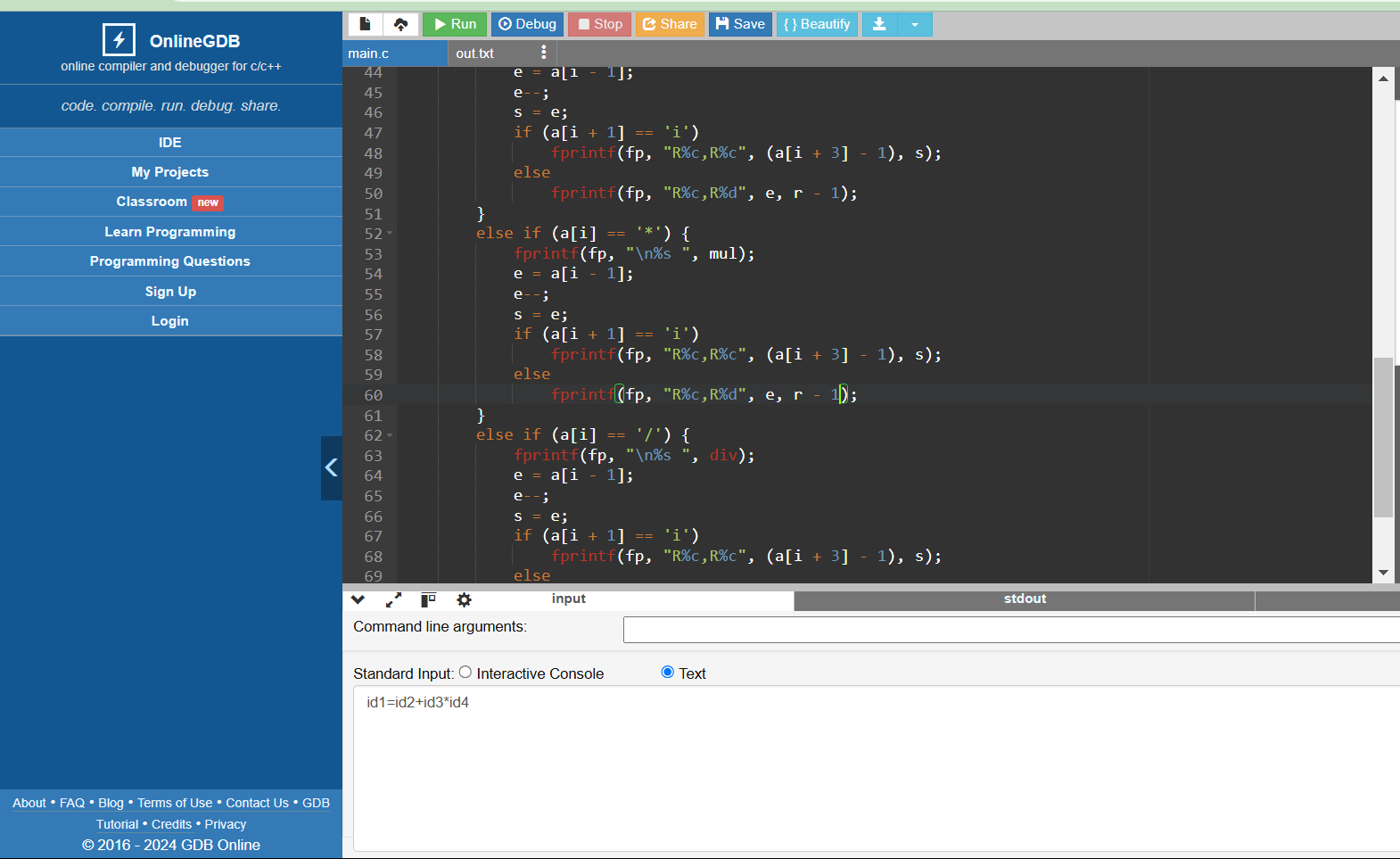
fprintf(fp, "\n%s R1,id1", mov);

fclose(fp);

}

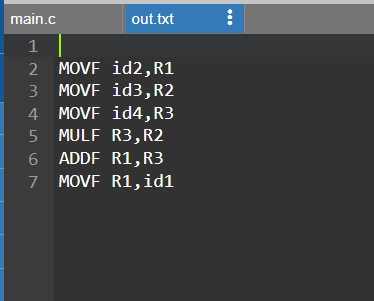
**INPUT(TEXT)**

id1=id2+id3\*id4



**OUTPUT:**

The output is shown in a generated file “out.txt”



**5. Implementation of Code optimization**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

struct op {

char l; // Left side variable

char r[20]; // Right side expression

} op[10], pr[10];

void main() {

int a, i, k, j, n, z = 0, m, q;

char \*p, \*l, \*tem, temp, t;

char nu[] = "\0";

// Input the number of assignments

printf("\nEnter the number of values: ");

scanf("%d", &n);

// Input the intermediate code (assignments)

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

printf("\nLeft ");

scanf(" %c", &op[i].l); // Take the left side variable (char)

printf("Right ");

scanf("%s", op[i].r); // Take the right side expression (string)

}

// Display the input intermediate code

printf("\nIntermediate code:\n");

for (i = 0; i < n; i++)

printf("%c = %s\n", op[i].l, op[i].r);

// Dead code elimination: Find left side variables that are not used later

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

temp = op[i].l;

p = NULL;

// Check if the left side variable is used on the right side in any future statement

for (j = 0; j < n; j++) {

p = strchr(op[j].r, temp);

if (p) {

pr[z].l = op[i].l;

strcpy(pr[z].r, op[i].r);

z++;

break;

}

}

}

// After dead code elimination

printf("\nAfter dead code elimination:\n");

for (k = 0; k < z; k++)

printf("%c = %s\n", pr[k].l, pr[k].r);

// Common subexpression elimination: Eliminate common expressions

for (m = 0; m < z; m++) {

tem = pr[m].r;

// Compare current expression with the rest

for (j = m + 1; j < z; j++) {

p = strstr(tem, pr[j].r);

if (p) {

pr[j].l = pr[m].l;

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

if (l) {

a = l - pr[i].r;

pr[i].r[a] = pr[m].l;

}

}

}

}

}

// Eliminate common expressions

printf("\nEliminated common expressions:\n");

for (i = 0; i < z; i++)

printf("%c = %s\n", pr[i].l, pr[i].r);

// Final optimized code by removing redundant assignments

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

for (j = i + 1; j < z; j++) {

q = strcmp(pr[i].r, pr[j].r);

// Eliminate redundant assignments (common subexpressions)

if ((pr[i].l == pr[j].l) && !q) {

pr[i].l = '\0'; // Mark as redundant

strcpy(pr[i].r, nu);

}

}

}

// Print the optimized code

printf("\nOptimized code:\n");

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

if (pr[i].l != '\0')

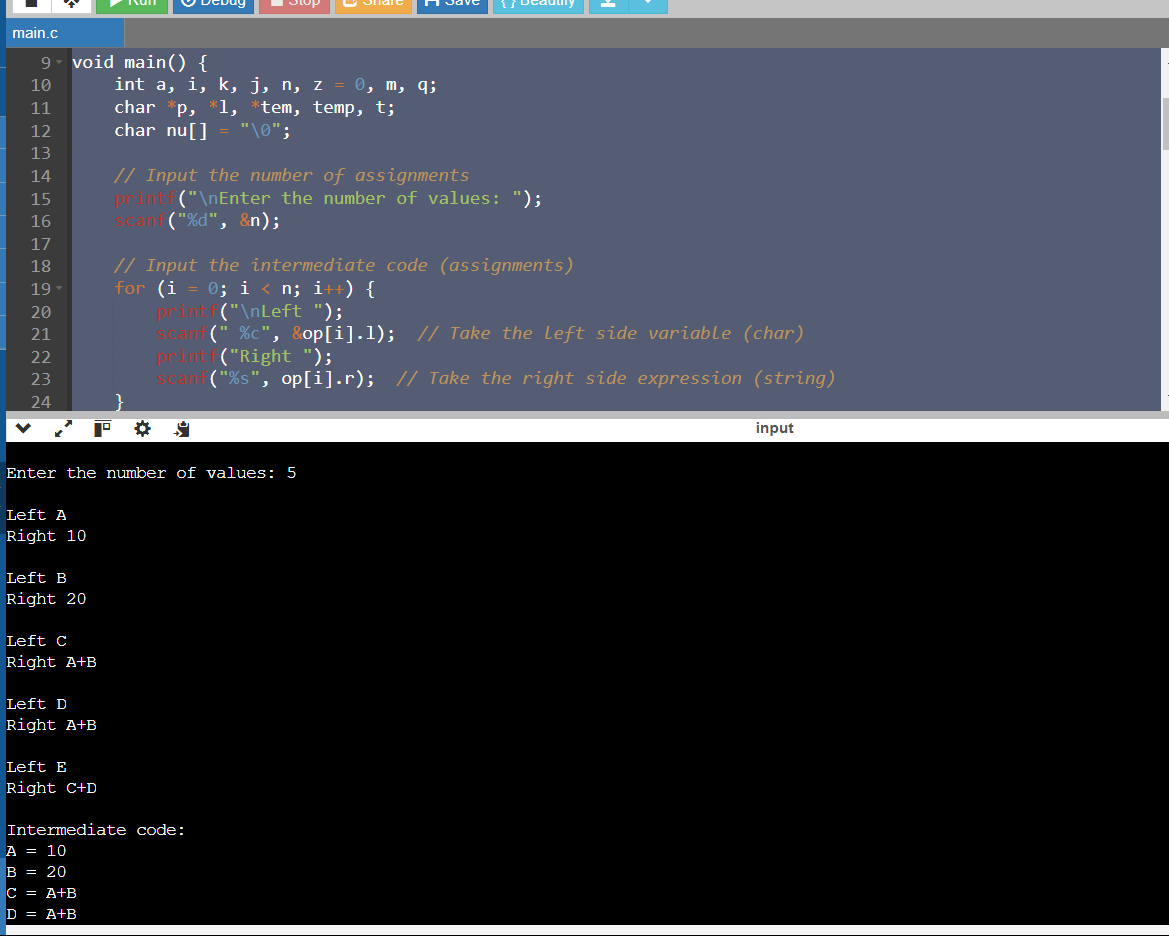
printf("%c = %s\n", pr[i].l, pr[i].r);

}

}

**INPUT(INTERACTIVE CONSOLE)**

Run the program and give inputs in the console window



**OUTPUT:**

It will be generated in the console itself

