

**ABESIT**

**COLLEGE CODE – 290**

# Lab File

|  |  |
| --- | --- |
| **NAME** | SANDEEP KUMAR SHUKLA |
| **BRANCH** | CSE |
| **UNIVERSITY ROLL NO.** | 1729010140 |
| **SESSION** | 2019-20 |
| **NAME OF LAB** | Compiler Design Lab (RCS 652) |

**Index**

|  |  |
| --- | --- |
| **S. No.** | **Program** |
| 1 | Write a C program to print a message with the help of getchar() and putchar() of a given line. |
| 2 | Write a C program for identifying keywords for lexical analyzer |
| 3 | Write a C program to check whether a string is constant or not with the help of getchar() function. |
| 4 | Write a C program to simulate lexical analyzer for validating operators. |
| 5 | Write a C program to implement lexical analyzer. |
| 6 | Write a C Program for construction of NFA From regular expression. |
| 7 | Write compiler construction tool Yacc (yet another compiler compiler**)** for unambiguous grammar |
| 8 | Write compiler construction tool Yacc for string (id+id\*id) using shift reduce parsing technique for ambiguous grammar |
| 9 | Write compiler construction tool lex for string (id+id\*id) |
| 10 | Write a C program for operator precedence parsing. |
| 11 | Write a C program for implementation of code generator |

**Program - 1**

**Aim :-** *Write a C program to print a message with help of getchar() and putchar() in a line.*

**Code :-**

#include <stdio.h>

int main() {

int i = 0;

char msg[100], ch;

printf("Enter a message: ");

while (ch = getchar()) {

if (ch == '\n') {

break;

}

msg[i++] = ch;

}

i = 0;

while (msg[i] != NULL) {

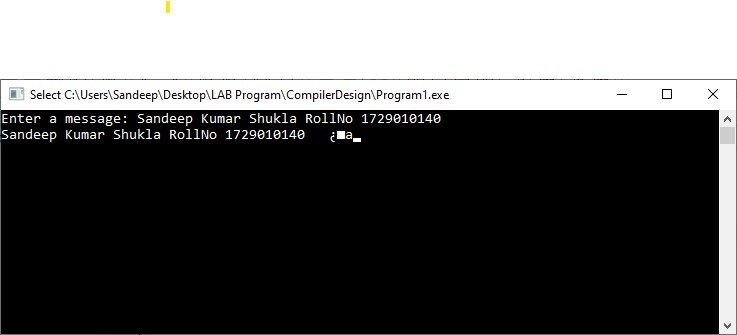
putchar(msg[i++]);

}

return 0;

}

**Output :-**

****

**Program - 2**

**Aim :-** *Write a C program for identifying keywords for lexical analyzer.*

**Code :-**

#include <stdio.h>

#include<conio.h>

#include<string.h>

#include <stdbool.h> /\*C90 does not support the boolean data type.C99 does include it with this include\*/

bool isKeyword(char\* str)

{

if (!strcmp(str, "if") || !strcmp(str, "else") ||

!strcmp(str, "while") || !strcmp(str, "do") ||

!strcmp(str, "break") ||

!strcmp(str, "continue") || !strcmp(str, "int")

|| !strcmp(str, "double") || !strcmp(str, "float")

|| !strcmp(str, "return") || !strcmp(str, "char")

|| !strcmp(str, "case") || !strcmp(str, "char")

|| !strcmp(str, "sizeof") || !strcmp(str, "long")

|| !strcmp(str, "short") || !strcmp(str, "typedef")

|| !strcmp(str, "switch") || !strcmp(str, "unsigned")

|| !strcmp(str, "void") || !strcmp(str, "static")

|| !strcmp(str, "struct") || !strcmp(str, "goto")){

return (true);

}

return (false);

}

int main() {

char str[20];

puts("Enter your string:");

gets(str);

if (isKeyword(str) == true)

printf("%s IS A KEYWORD\n",str);

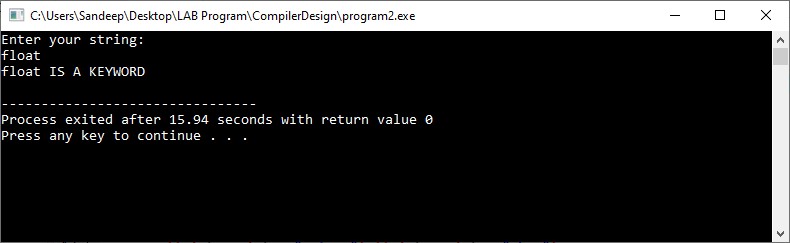
else

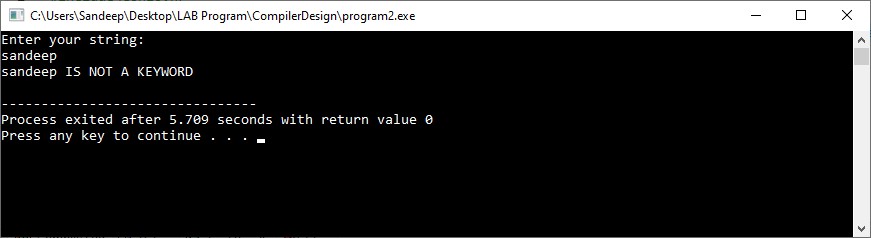
printf("%s IS NOT A KEYWORD\n",str);

return 0;

}

**Output :-**

****



**Program - 3**

**Aim :-** *Write a C program to check whether a string is constant or not .*

**Code :-**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

int main() {

char str[10];

int len, a;

printf("Input a string :");

gets(str);

len = strlen(str);

a = 0;

while (a < len) {

if (isdigit(str[a])) {

a++;

}

else {

printf(" It is not a Constant");

break;

}

}

if (a == len) {

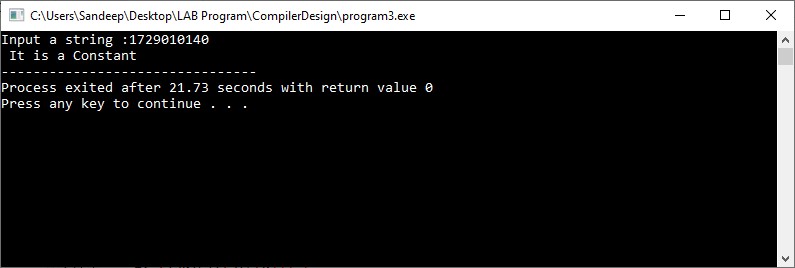
printf(" It is a Constant");

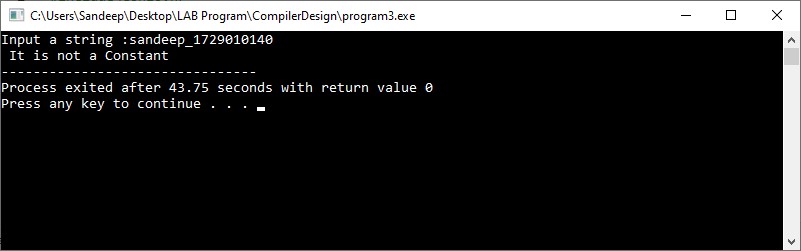
}

return 0;

}

**Output :-**

****

****

**Program - 4**

**Aim :-** *Write a C program to simulate lexical analyzer for validating operators.*

**Code :-**

#include<stdio.h>

#include<conio.h>

int main() {

char s[5];

printf("\n Enter any operator: ");

gets(s);

switch (s[0]) {

case '>':

if (s[1] == '=') printf("\n Greater than or equal");

else printf("\n Greater than");

break;

case '<':

if (s[1] == '=') printf("\n Less than or equal");

else printf("\nLess than");

break;

case '=':

if (s[1] == '=') printf("\nEqual to");

else printf("\nAssignment");

break;

case '!':

if (s[1] == '=') printf("\nNot Equal");

else printf("\n Bit Not");

break;

case '&':

if (s[1] == '&') printf("\nLogical AND");

else printf("\n Bitwise AND");

break;

case '|':

if (s[1] == '|') printf("\nLogical OR");

else printf("\nBitwise OR");

break;

case '+':

printf("\n Addition");

break;

case '-':

printf("\nSubstraction");

break;

case '\*':

printf("\nMultiplication");

break;

case '/':

printf("\nDivision");

break;

case '%':

printf("Modulus");

break;

default:

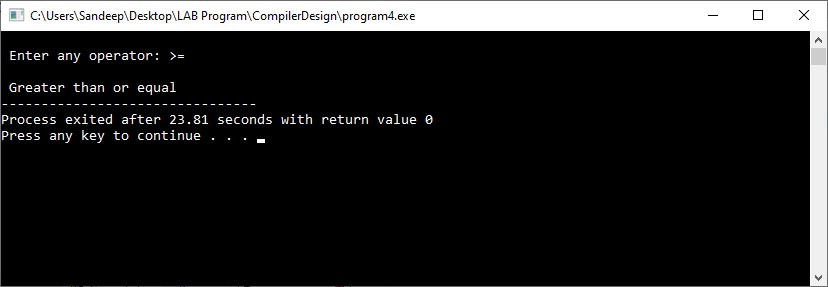
printf("\n Not a operator");

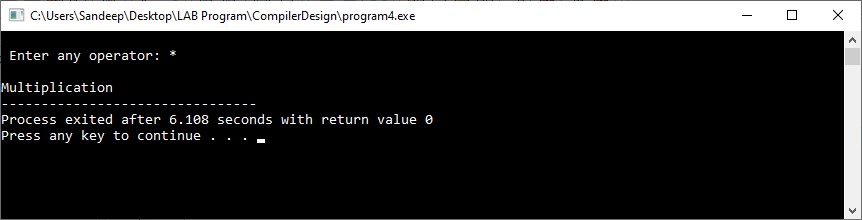
}

return 0;

}

**Output :-**

****

****

**Program - 5**

**Aim :-** *Write a C program to implement lexical analyzer.*

**Code :-**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

int isKeyword(char buffer[]){

char keywords[32][10] = {"auto","break","case","char","const","continue","default", "do","double","else","enum","extern","float","for","goto", "if","int","long","register","return","short","signed", "sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

FILE \*fp;

int i,j=0;

fp = fopen("program5.txt","r");

if(fp == NULL){

printf("error while opening the file\n");

exit(0);

}

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

for(i = 0; i < 6; ++i){

if(ch == operators[i])

printf("%c is operator\n", ch);

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

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

else

printf("%s is indentifier\n", buffer);

}

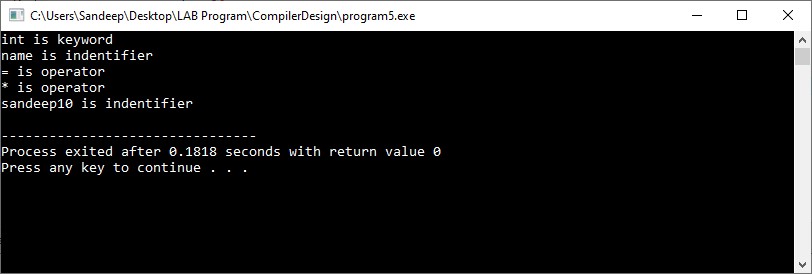
}

fclose(fp);

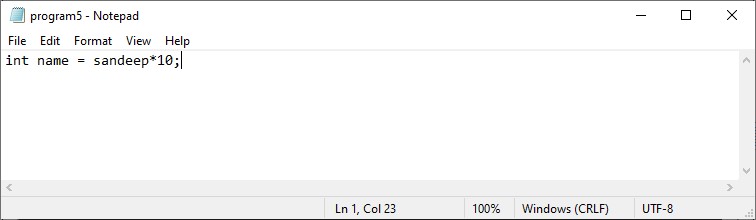
return 0;

}

**Output :-**

****

This is program5.txt file **👇**

****

**Program - 6**

**Aim :-** *Write a C Program for construction of NFA From regular expression.*

**Code :-**

#include<stdio.h>

#include<conio.h>

int main() {

char m[20], t[10][10];

int n, i, j, r = 0, c = 0;

printf("\n\t\t\t\tSIMULATION OF NFA");

printf("\n\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

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

for (j = 0; j < 10; j++) {

t[i][j] = ' ';

}

}

printf("\n\nEnter a regular expression:");

scanf("%s", m);

n = strlen(m);

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

switch (m[i]) {

case '|':

{

t[r][r + 1] = 'E';t[r + 1][r + 2] = m[i - 1];t[r + 2][r + 5] = 'E';t[r][r + 3] = 'E';t[r + 4][r + 5] = 'E';t[r + 3][r + 4] = m[i + 1];

r = r + 5;

break;

}

case '\*':

{

t[r - 1][r] = 'E';t[r][r + 1] = 'E';

t[r][r + 3] = 'E';t[r + 1][r + 2] = m[i - 1];t[r + 2][r + 1] = 'E';t[r + 2][r + 3] = 'E';r = r + 3;

break;

}

case '+':

{

t[r][r + 1] = m[i - 1];

t[r + 1][r] = 'E';r = r + 1;

break;

}

default:

{

if (c == 0) {

if ((isalpha(m[i])) && (isalpha(m[i + 1]))) {

t[r][r + 1] = m[i];

t[r + 1][r + 2] = m[i + 1];

r = r + 2;

c = 1;

}

c = 1;

} else if (c == 1) {

if (isalpha(m[i + 1])) {

t[r][r + 1] = m[i + 1];

r = r + 1;

c = 2;

}

} else {

if (isalpha(m[i + 1])) {

t[r][r + 1] = m[i + 1];

r = r + 1;

c = 3;

}

}

}

break;

}

}

printf("\n");

for (j = 0; j <= r; j++) printf(" %d", j);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n");

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

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

printf(" %c", t[i][j]);

}

printf(" | %d", i);

printf("\n");

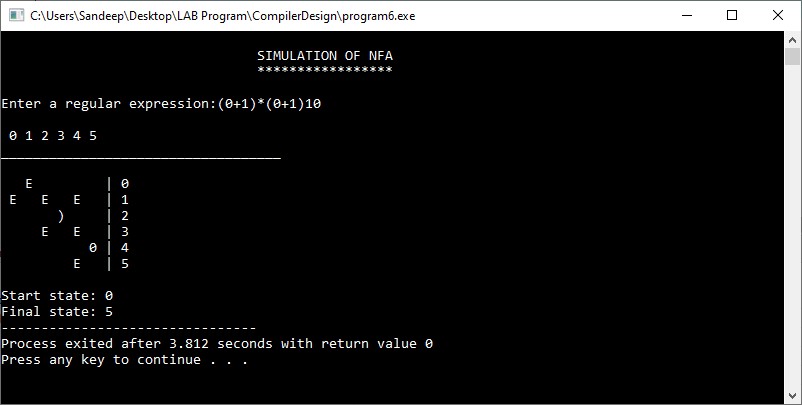
}

printf("\nStart state: 0\nFinal state: %d", i - 1);

return 0;

}

**Output :-**

****

**Program - 7**

**Aim :-** *Write compiler construction tool Yacc (yet another compiler compiler) for unambiguous grammar E->E+T|T*

*T->T/F|F*

*F->(E)|id*

*of string(id+id/id)*

**Code :-**

% {

#include <stdio.h>

int regs[26];int base; %

} %

start list

%

union {

int a;

} %

type < a > expr number % token DIGIT LETTER

list:

|

list stat '\n' |

list error '\n' {

yyerrok;

};

stat: expr {

printf("%d\n", $1);

} |

LETTER '='

expr {

regs[$1] = $3;

};

factor: '('

expr ')' {

$$ = $2;

} |

|

term: term '/'

factor {

$$ = $1 / $3;

} |

term: factor |

expr: expr '+'

term {

$$ = $1 + $3;

} |

expr: term |

Factor: number |

;

number: DIGIT {

$$ = $1;

base = ($1 == 0) ? 8 : 10;

} |

number DIGIT {

$$ = base \* $1 + $2;

}; %

%

main() {

return (yyparse());

}

yyerror(s) char \* s; {

fprintf(stderr, "%s\n", s);

}

yywrap() {

return (1);

}

**Output :-**

Input = (2+2/2)

Output =3

**Program - 8**

**Aim :-** *Write compiler construction tool Yacc for string (id+id\*id) using shift reduce parsing technique for ambiguous grammar*

*E->E+E|E-E|-E|€|E\*E|E/E|id*

**Code :-**

% {

#include <stdio.h>

int regs[26];int base; %

} %

start list

%

union {

int a;

} %

type < a > expr number %

token DIGIT LETTER %

left‘ | ’

%

left‘ & ’

%

left‘ + ’‘ -‘

%

left‘ \* ’‘/’ ‘%’ %

left UMINUS %

%

list:

|

list stat‘\ n’ |

list error‘\ n’ {

yyerrok;

};

stat: expr {

printf(“ % d\ n”, $1);

} |

LETTER‘ = ’expr {

regs[$1] = $3;

};

expr: ‘(‘expr‘)’ {

$$ = $2;

} |

expr‘ \* ’expr {

$$ = $1 \* $3;

} |

expr‘ / ’expr {

$$ = $1 / $3;

} |

expr‘ % ’expr {

$$ = $1 % $3;

} |

expr‘ + ’expr {

$$ = $1 + $3;

} |

expr‘ - ‘expr {

$$ = $1 - $3;

} |

expr‘ & ’expr {

$$ = $1 & $3;

} |

expr‘ | ’expr {

$$ = $1 | $3;

} | ‘-‘expr % prec UMINUS {

$$ = -$2;

} |

LETTER {

$$ = regs[$1];

} |

number;

number: DIGIT {

$$ = $1;

base = ($1 == 0) ? 8 : 10;

} |

number DIGIT {

$$ = base \* $1 + $2;

}; %

%

main() {

return (yyparse());

}

yyerror(s) char \* s; {

fprintf(stderr, “ % s\ n”, s);

}

yywrap() {

return (1);

}

**Output :-**

Input = 2+2

Output=4

**Program - 9**

**Aim :-** *Write compiler construction tool lex for string (id+id\*id).*

**Code :-**

% {

#include <stdio.h>

#include "y.tab.h"

int c;

extern int yylval;

%}

%%

" ";

[a-z] {

c = yytext[0];

yylval = c - 'a';

return(LETTER);

}

[0-9] {

c = yytext[0];

yylval = c - '0';

return(DIGIT);

}

[^a-z0-9\b] {

c = yytext[0];

return(c);

}

**Output :-**

(2+2\*2)

=6

**Program - 10**

**Aim :-** *Write a C program for operator precedence parsing.*

**Code :-**

#include<conio.h>

#include<ctype.h>

#include<stdio.h>

int main()

{

int i, j, x, x1, k, k1, z1, z;

char a[20], b[20][20], c[20], p[10] = "\*+$";

char fp[10][10] = {

"id1",

"id2",

"id3"

}, d[10];

printf("ENTER INPUT STRING: ");

gets(a);

i = 0;

j = 0;

x = 0;

x1 = 0;

printf("$ ");

while (a[i] != '\0') {

j = 0;

while (1) {

if (a[i] != '+' && a[i] != '\*' && a[i] != '$') {

b[x][j] = a[i];

i++;

j++;

} else {

c[x] = a[i];

break;

}

}

b[x][j] = '\0';

for (k = 0; k < 3; k++)

if (!strcmp(b[x], fp[k]))

for (k1 = 0; k1 < 3; k1++)

if (c[x] == p[k1]) printf("< %s > %c ", b[x], c[x]);

x++;

i++;

}

c[x] = '\0';

i = 0;

printf("\n$ <");

while (c[i] != '\0') {

for (k = 0; k < 3; k++)

if (c[i] == p[k]) z = k;

i++;

printf(" %c", c[i - 1]);

for (k = 0; k < 3; k++)

if (c[i] == p[k]) z1 = k;

i++;

if (z > z1) printf(" <");

else printf(" >");

printf(" %c", c[i - 1]);

i++;

}

printf(" > $");

for (i = 0; i < 3; i++) {

if (c[i] == p[0]) c[i] = ' ';

}

i = 0;

printf("\n$ <");

for (k = 0; k < 3; k++)

if (c[i] == p[k]) z = k;

printf(" %c", c[i]);

i++;

i++;

for (k = 0; k < 3; k++)

if (c[i] == p[k]) z1 = k;

if (z > z1) printf(" <");

else

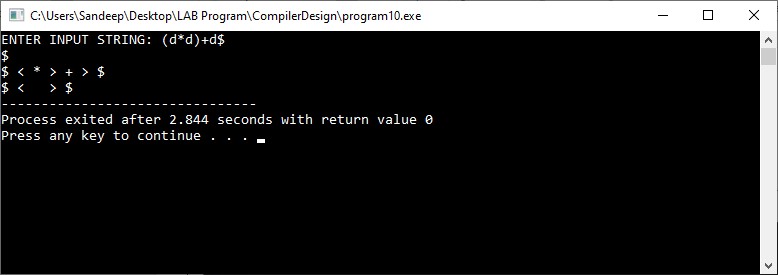
printf(" >");

printf(" %c", c[i]);

return 0;

}

**Output :-**

****

**Program - 11**

**Aim :-** *Write a C program for implementation of code generator.*

**Code :-**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int i=1,j=0,no=0,tmpch=90;

char str[100],left[15],right[15];

void findopr();

void explore();

void fleft(int);

void fright(int);

struct exp{

int pos;

char op;

}k[15];

int main() {

printf("\t\tINTERMEDIATE CODE GENERATION\n\n");

printf("Enter the Expression :");

scanf("%s", str);

printf("The intermediate code:\t\tExpression\n");

findopr();

explore();

return 0;

}

void findopr() {

for (i = 0; str[i] != '\0'; i++)

if (str[i] == ':') {

k[j].pos = i;

k[j++].op = ':';

}

for (i = 0; str[i] != '\0'; i++)

if (str[i] == '/') {

k[j].pos = i;

k[j++].op = '/';

} for (i = 0; str[i] != '\0'; i++)

if (str[i] == '\*') {

k[j].pos = i;

k[j++].op = '\*';

} for (i = 0; str[i] != '\0'; i++)

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

k[j].pos = i;

k[j++].op = '+';

} for (i = 0; str[i] != '\0'; i++)

if (str[i] == '-') {

k[j].pos = i;

k[j++].op = '-';

}

}

void explore() {

i = 1;

while (k[i].op != '\0') {

fleft(k[i].pos);

fright(k[i].pos);

str[k[i].pos] = tmpch--;

printf("\t%c := %s%c%s\t\t", str[k[i].pos], left, k[i].op, right);

for (j = 0; j < strlen(str); j++)

if (str[j] != '$') printf("%c", str[j]);

printf("\n");

i++;

}

fright(-1);

if (no == 0) {

fleft(strlen(str));

printf("\t%s := %s", right, left);

getch();

exit(0);

}

printf("\t%s := %c", right, str[k[--i].pos]);

}

void fleft(int x) {

int w = 0, flag = 0;

x--;

while (x != -1 && str[x] != '+' &&

str[x] != '\*' && str[x] != '=' && str[x] != '\0' && str[x] != '-' && str[x] != '/' && str[x] != ':') {

if (str[x] != '$' && flag == 0) {

left[w++] = str[x];

left[w] = '\0';

str[x] = '$';

flag = 1;

}

x--;

}

}

void fright(int x) {

int w = 0, flag = 0;

x++;

while(x != -1 && str[x] != '+' && str[x] != '\*' && str[x] != '\0' && str[x] != '=' && str[x] != ':' && str[x] != '-'&&str[x]!=' / ') {

if (str[x] != '$' && flag == 0) {

right[w++] = str[x];

right[w] = '\0';

str[x] = '$';

flag = 1;

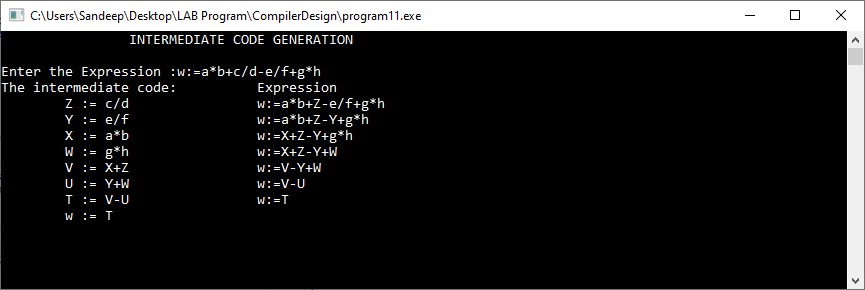
}

x++;

}

}

**Output :-**

****