**1: No of words**

%{

#include<stdio.h>

#include<string.h>

int no\_of\_lines = 0,

no\_of\_chars = 0,

no\_of\_tabs = 0,

no\_of\_words = 0,

no\_of\_uppercase =0,

no\_of\_lowercase =0,

no\_of\_sent = 0,

no\_of\_num = 0;

%}

%%

\n {no\_of\_words++; no\_of\_lines++;};

\t {no\_of\_words++;++no\_of\_tabs;};

[.?!] {no\_of\_sent++; no\_of\_chars++;};

[0-9] {++no\_of\_num; no\_of\_chars++;};

[a-z] {++no\_of\_lowercase;no\_of\_chars++;};

[A-Z] {++no\_of\_uppercase;no\_of\_chars++;};

[' '] {no\_of\_words++;};

. no\_of\_chars++;

end return 0;

%%

int yywrap(){}

int main(int argc, char \*\*argv)

{

yylex();

printf("Count of characters = %d\nCount of words = %d\nCount of sentences = %d\nCount of lines

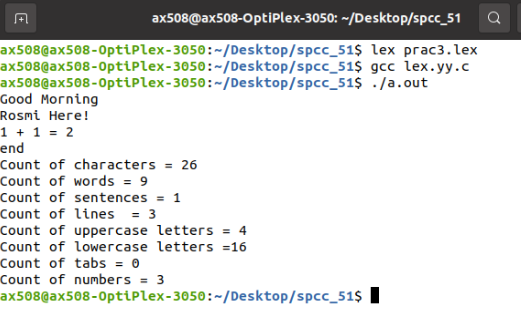
= %d\nCount of uppercase letters = %d\nCount of lowercase letters =%d\nCount of tabs = %d\

nCount of numbers = %d\n",no\_of\_chars, no\_of\_words,

no\_of\_sent,no\_of\_lines,no\_of\_uppercase,no\_of\_lowercase, no\_of\_tabs, no\_of\_num );

return 0;

}

****

**2: Uppercase and lowercase letters**

**lex.l**

%{ int count = 0;

%} %%

[A-Z] {printf("%s capital letter\n", yytext);

count++;}

. {printf("%s not a capital letter\n", yytext);}

\n {return 0;}

%%

int yywrap(){} int main(){ FILE \*fp; char filename[50]; printf("Enter the filename: \n"); scanf("%s",filename); fp = fopen(filename,"r");

yyin = fp;

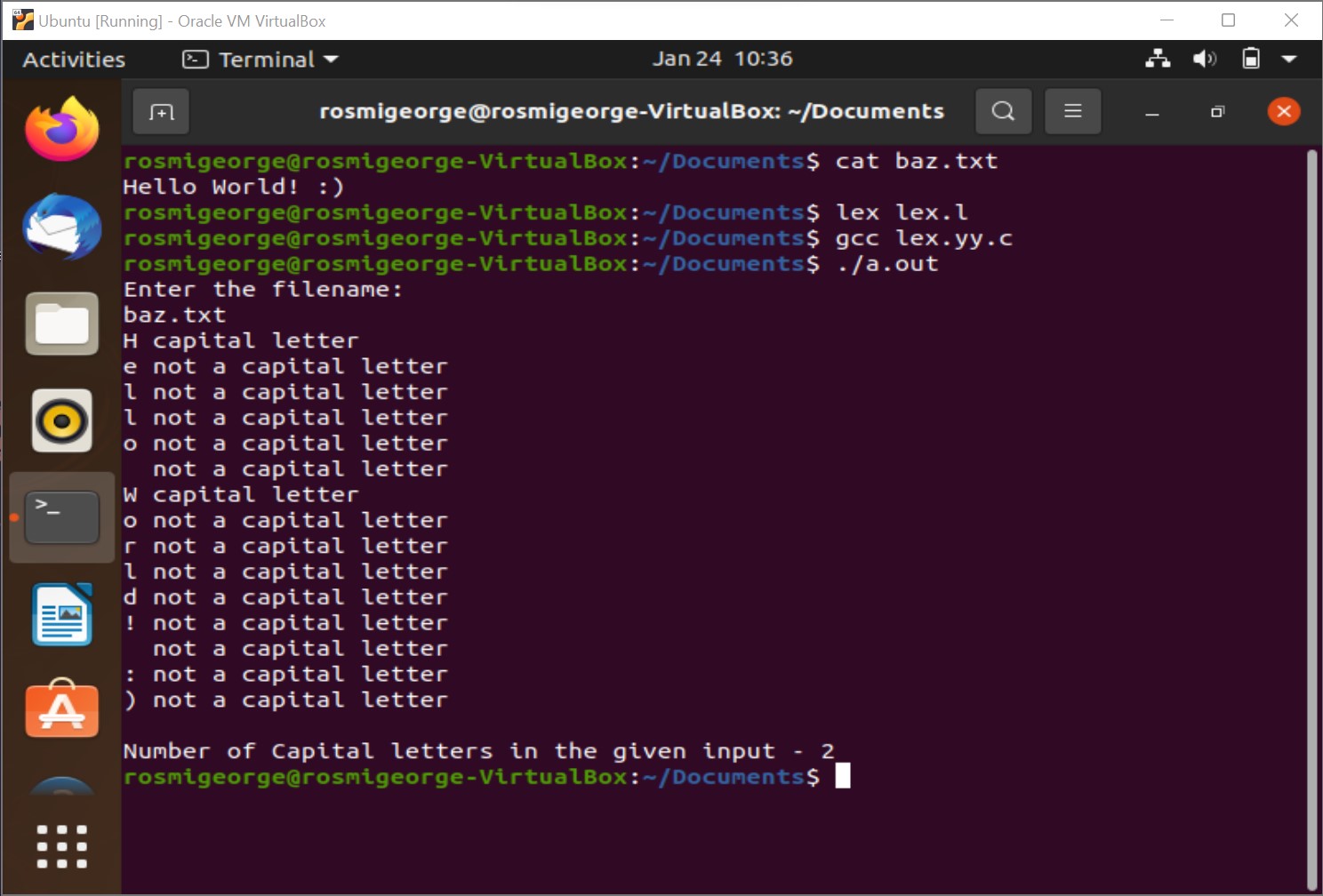
yylex();

printf("\nNumber of Capital letters "

"in the given input - %d\n", count);

return 0;

}



**3: Pass1/Pass2 assembler**

from sys import exit

motOpCode = {

"MOV": 1,

"A": 2,

"S": 3,

"M": 4,

"D": 5,

"AN": 6,

"O": 7,

"ADD": 8,

"SUB": 9,

"MUL": 10,

"DIV": 11,

"AND": 12,

"OR": 13,

"LOAD": 14,

"STORE": 15,

"DCR": 16,

"INC": 17,

"JMP": 18,

"JNZ": 19,

"HALT": 20

}

motSize = {

"MOV": 1,

"A": 1,

"S": 1,

"M": 1,

"D": 1,

"AN": 1,

"O": 1,

"ADD": 1,

"SUB": 2,

"MUL": 2,

"DIV": 2,

"AND": 2,

"OR ": 2,

"LOAD": 3,

"STORE": 3,

"DCR": 1,

"INC": 1,

"JMP": 3,

"JNZ": 3,

"HALT": 1

}

l = []

relativeAddress = []

machineCode = []

symbol = []

symbolValue = []

RA = 0

current = 0

count = 0

temp = []

n = int(input("Enter the no of instruction lines : "))

for i in range(n):

instructions = input("Enter instruction line {} : ".format(i + 1))

l.append(instructions)

l = [x.upper() for x in l]

for i in range(n):

x = l[i]

if "NEXT:" in x:

s1 = ''.join(x)

a, b, c = s1.split()

a = a[:4]

l[i] = b + " " + c

symbol.append(a)

x = l[i]

if b in motOpCode:

value = motOpCode.get(b)

size = motSize.get(b)

if len(str(size)) == 1:

temp = "000" + str(size)

elif len(str(size)) == 2:

temp = "00" + str(size)

elif len(str(size)) == 3:

temp = "0"+str(size)

else:

print("Instruction is not in Op Code.")

exit(0)

symbolValue.append(temp)

previous = size

RA += current

current = previous

relativeAddress.append(RA)

if c.isalpha() is True:

machineCode.append(str(value))

else:

temp = list(b)

for i in range(len(temp)):

if count == 2:

temp.insert(i, ',')

count = 0

else:

count = count + 1

s = ''.join(temp)

machineCode.append(str(value) + "," + s)

elif " " in x:

s1 = ''.join(x)

a, b = s1.split()

if a in motOpCode:

value = motOpCode.get(a)

size = motSize.get(a)

previous = size

RA += current

current = previous

relativeAddress.append(RA)

if b.isalpha() is True:

machineCode.append(str(value))

else:

temp = list(b)

for i in range(len(temp)):

if count == 2:

temp.insert(i, ',')

count = 0

else:

count = count + 1

s = ''.join(temp)

machineCode.append(str(value) + "," + s)

else:

print("Instruction is not in Op Code.")

exit(0)

else:

if x in motOpCode:

value = motOpCode.get(x)

size = motSize.get(x)

previous = size

RA += current

current = previous

relativeAddress.append(RA)

machineCode.append(value)

else:

print("Instruction is not in Op Code.")

exit(0)

print("Symbol Table : \n")

print("\n Symbol Value(Address)")

for i in range(len(symbol)):

print(" {} {}".format(symbol[i], symbolValue[i]))

print("\n Pass-1 machine code output without reference of the symbolic address : \n")

print("Relative Address Instruction OpCode")

for i in range(n):

if "NEXT" in l[i]:

print("{} {} {}, - ".format(

relativeAddress[i], l[i], machineCode[i]))

else:

print("{} {} {} ".format(

relativeAddress[i], l[i], machineCode[i]))

print("\n Pass-2 output: Machine code output \n ")

print("Relative Address Instruction OpCode")

for i in range(n):

if "NEXT" in l[i]:

for j in range(len(symbol)):

if "NEXT" in symbol[j]:

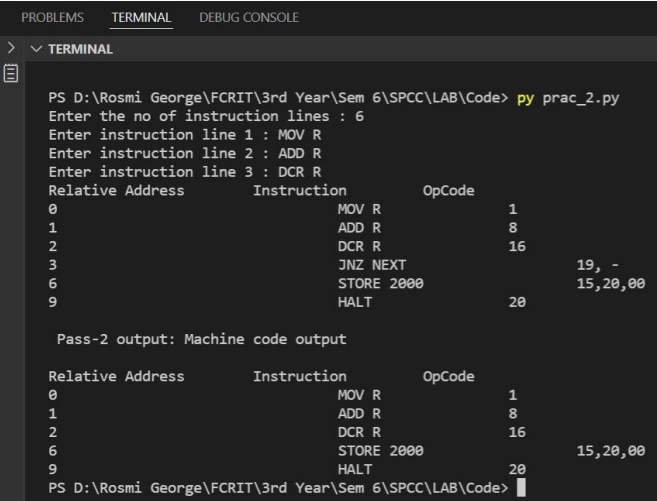
pos = j

print("{} {} {} , {}".format(

relativeAddress[i], l[i], machineCode[i], symbolValue[pos]))

else:

print("{} {} {} ".format(relativeAddress[i], l[i], machineCode[i]))



**4: Recognise identifiers**

%{

enum {

LOOKUP = 0,

INT,

FLOAT,

DOUBLE,

DEF

};

int state;

int add\_count = 1000;

int add\_word(int type, char \*word);

int lookup\_word(char \*word);

%}

%%

[\n] { state = LOOKUP; }

[(] { state = LOOKUP; }

^int { state = INT; }

^float { state = FLOAT; }

^double { state = DOUBLE; }

^def { state = DEF; }

end {return 0;}

[0-9][a-zA-Z0-9\_]\* ;

[a-zA-Z\_][a-zA-Z0-9\_][(] {

if(state != LOOKUP) {

add\_word(DEF, yytext);

}

}

[a-zA-Z\_][a-zA-Z0-9\_]\* {

if(state != LOOKUP) {

add\_word(state, yytext);

}

}

. ;

%%

int yywrap(){}

struct word \*word\_list;

struct word {

char \*word\_name;

int word\_type;

int word\_address;

struct word \*next;

};

extern void \*malloc() ;

int main(int argc, char \*\*argv)

{

yylex();

struct word \*wp = word\_list;

printf("<--------------------Symbol Table-------------------->\nIdentifier\t\tAddress\t\tType\n");

for(; wp; wp = wp->next) {

switch(wp->word\_type) {

case INT: printf("%s\t\t\t%d\t\tInteger\n", wp->word\_name, wp->word\_address); break; case

FLOAT: printf("%s\t\t\t%d\t\tFloat\n", wp->word\_name, wp->word\_address); break; case

DOUBLE: printf("%s\t\t\t%d\t\tDouble\n", wp->word\_name, wp->word\_address); break; case

DEF: printf("%s\t\t%d\t\tFunction\n", wp->word\_name, wp->word\_address); break; }

}

return 0;

}

int add\_word(int type, char \*word)

{

struct word \*wp;

if(lookup\_word(word) != LOOKUP) {

printf("!!! warning: identifier %s already defined \n", word);

return 0;

}

wp = (struct word \*) malloc(sizeof(struct word));

wp->next = word\_list;

wp->word\_name = (char \*) malloc(strlen(word)+1);

strcpy(wp->word\_name, word);

wp->word\_type = type;

add\_count += 4;

wp->word\_address = add\_count;

word\_list = wp;

return 1;

}

int lookup\_word(char \*word)

{

struct word \*wp = word\_list;

for(; wp; wp = wp->next) {

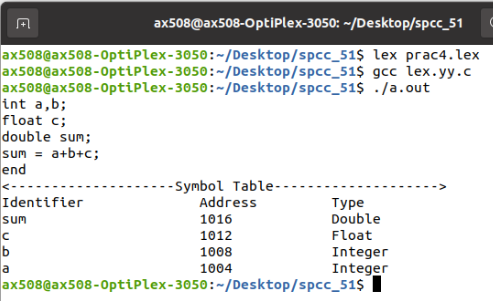
if(strcmp(wp->word\_name, word) == 0)

return wp->word\_type;

}

return LOOKUP;

}



**5: Valid arithmetic expression**

**validarith.y**

%{

#include<stdio.h>

int valid=1;

%}

%token num id op

%%

start : id '=' s ';'

s : id x

| num x

| '-' num x

| '(' s ')' x

;

x : op s

| '-' s

|

;

%%

int yyerror()

{

valid=0;

printf("\nInvalid expression!\n");

return 0;

}

int main()

{

printf("\nEnter the expression:\n");

yyparse();

if(valid)

{

printf("\nValid expression!\n");

}

}

**validarith.l**

%{

#include "y.tab.h"

%}

%%

[a-zA-Z\_][a-zA-Z\_0-9]\* return id;

[0-9]+(\.[0-9]\*)? return num;

[+/\*] return op;

. return yytext[0];

\n return 0;

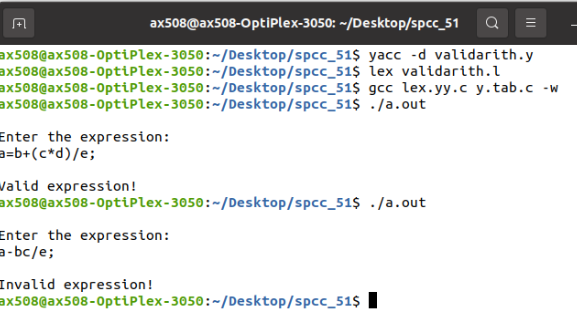
%%

int yywrap()

{

return 1;

}



**6: Single pass macro**

**macro.py**

import os

input\_file =  open("file.txt",'r')

macro\_start\_flag = 0

macro\_name\_flag = 0

mdt =dict()

mnt = dict()

ala = dict()

macro\_name = ""

mdt\_index = 0

mnt\_index= 0

ala\_index = 0

macro\_def = []

macro\_args = []

call\_count = 0

for line in input\_file:

    if(len(line.split())>0):

        line = line.replace("\n","")

        if line.split()[0] == 'MACRO':

            macro\_name\_flag= 0

            macro\_start\_flag =1

            pass

        elif line.split()[0] == 'MEND':

            mdt\_index = + mdt\_index

            macro\_line = (mdt\_index,line)

            macro\_def.append(macro\_line)

            mdt\_entry = {

                macro\_name: macro\_def

            }

            mdt.update(mdt\_entry)

            macro\_def = []

            macro\_args =[]

            macro\_start\_flag = 0

        else:

            if macro\_start\_flag !=0:

                if macro\_name\_flag ==0:

                    macro\_name = line.split()[0]

                    mdt\_ent = {

                        macro\_name: None

                    }

                    mdt.update(mdt\_ent)

                    macro\_name\_flag =1

                mdt\_index += 1

                for code in line.split():

                   if '&' in code:

                        if code not in [arg[0] for arg in macro\_args]:

                            arg\_sub = '#'+str(len(macro\_args)+1)

                            macro\_args.append((code,arg\_sub))

                        else:

                            for args in macro\_args:

                                if args[0] == code:

                                    arg\_sub = args[1]

                        line = line.replace(code,arg\_sub)

                macro\_line = (mdt\_index,line)

                macro\_def.append(macro\_line)

            else:

                macro\_ala = []

                macro\_call = line.split()[0]

                if macro\_call in list(mdt.keys()):

                    call\_count += 1

                    arguments = line.split()[1:]

                    for arg in arguments:

                        arg = arg.upper()

                        if len(arg) < 8:

                            additional\_b = 8 - len(arg)

                            for b in range(additional\_b):

                                arg += 'b'

                                macro\_ala.append(arg)

                            ala\_entry = {

                                macro\_call + '\_' +str(call\_count): macro\_ala

                            }

                            ala.update(ala\_entry)

for entry in mdt:

    mnt\_index += 1

    mnt\_entry = {

        entry : (mnt\_index,mdt[entry][0][0])

    }

    mnt.update(mnt\_entry)

ala\_final = ala.copy()

for calls in ala:

    macro\_ala = []

    macro\_call = calls.split('\_')[0]

    given = ala[calls]

    for line in mdt[macro\_call]:

        if macro\_call not in line[1] and 'MEND' not in line[1]:

            for code in line[1].split():

                if '#' in code:

                    index = int(code.replace('#',''))-1

macro\_ala.append(given[index])

next\_call = line[1].split()[0]

call\_count +=1

next\_call += '\_'+str(call\_count)

ala\_entry = {

    next\_call: macro\_ala

}

ala\_final.update(ala\_entry)

print("\nMDT")

print("Index \t Contents")

for entry in mdt:

    for lines in mdt[entry]:

        print(lines[0],'\t',lines[1])

print("\nMNT")

print("Index \t Argument")

for entry in mnt:

    print(mnt[entry][0], '\t', entry, '\t\t', mnt[entry][1])

print("ALA")

print("Index \t Argument")

for entry in ala\_final:

    for arg in ala\_final[entry]:

        ala\_index += 1

        print(ala\_index,'\t', arg)

**file.txt**

ADD1 &arg

L 1 &arg

A 1 = F'1'

ST 1 &arg

MEND

MACRO

ADDS &arg1 &arg2 &arg3

ADD1 &arg1

ADD1 &arg2

ADD1 &arg3

MEND

ADDS data1 data2 data3



**7: Sci calc**

**calc.l**

%{

#include "y.tab.h"

%}

%%

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

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

[ \t]+         ;  /\*For skipping whitespaces\*/

\n            { return 0; }

COS|cos {return cos1; }

SIN|sin {return sin1; }

TAN|tan {return tan1; }

%%

int yywrap(){

return 1;

}

**calc.y**

%{

#include<stdio.h>

#include<math.h>

%}

%union

{

   double num;

}

%token <num> cos1 sin1 tan1 number

%type <num> line exp

%%

line : exp  {

printf("\nResult = %f\n", $$);

return 0;

}

exp : number {$$=$1;}

| exp '+' number {$$=$1+$3;}

| exp '-' number {$$=$1-$3;}

| exp '\*' number {$$=$1\*$3;}

| exp '/' number {$$=$1/$3;}

| cos1 number {$$ = cos(($2/180)\*3.14);}

| sin1 number {$$ = sin(($2/180)\*3.14);}

| tan1 number {$$ = tan(($2/180)\*3.14);}

;

%%

int main(){

printf("Enter an expression: ");

yyparse();

}

int yyerror(){

exit(0);

}



|  |  |
| --- | --- |
| **Program** | ***ICG.y***  %{  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  char \* createT();  int tempcount=0;  int top=-1;  %}  %union  {  char str[30];  }  %left '+'  %left  '-'  %left '\*'  %left  '/'  %token <str> Var  %token <str> Num  %type  <str> s  %type  <str> exp  %%  s:     Var '=' exp {printf("\n%s=%s\n",$1,$3);}  exp: '(' exp ')' {strcpy($$,$2);}  | exp '+' exp {strcpy($$,createT());printf("\n%s=%s+%s",$$,$1,$3);}  |exp '-' exp {strcpy($$,createT());printf("\n%s=%s-%s",$$,$1,$3);}  | exp '\*' exp {strcpy($$,createT());printf("\n%s=%s\*%s",$$,$1,$3);}  | exp '/' exp {strcpy($$,createT());printf("\n%s=%s/%s",$$,$1,$3);}  | Num  {strcpy($$,$1);}  | Var {strcpy($$,$1);}      %%    char \* createT()  {  char snum[30],\*ptr;  sprintf(snum,"t%d",tempcount);  ptr=snum;  tempcount++;  return ptr;  }  int main()  {  yyparse();  return 0;    }  int yyerror(char \*err)  {  printf("\nInvlaid");  exit(0);  }    ***ICG.l***    %{  #include"y.tab.h"  %}  %%  [a-zA-Z]+ {strcpy(yylval.str,yytext);   return Var;}  [0-9]+ {strcpy(yylval.str,yytext);   return Num;}  \n {return 0;}  . {return yytext[0];}  %%  int yywrap()  {  return 1;  } |

Prac 9

%{

#include<stdio.h> #include<string.h> int no\_of\_lines = 0, no\_of\_chars = 0,

no\_of\_tabs = 0,

no\_of\_words = 0,

no\_of\_uppercase =0,

no\_of\_lowercase =0,

no\_of\_sent = 0,

no\_of\_num = 0;

%}

%%

\n {no\_of\_words++; no\_of\_lines++;};

\t {no\_of\_words++;++no\_of\_tabs;}; [.?!] {no\_of\_sent++; no\_of\_chars++;}; [0-9] {++no\_of\_num; no\_of\_chars++;};

[a-z] {++no\_of\_lowercase;no\_of\_chars++;}; [A-Z] {++no\_of\_uppercase;no\_of\_chars++;}; [' '] {no\_of\_words++;};

. no\_of\_chars++; end return 0;

%%

int yywrap(){}

int main(int argc, char \*\*argv)

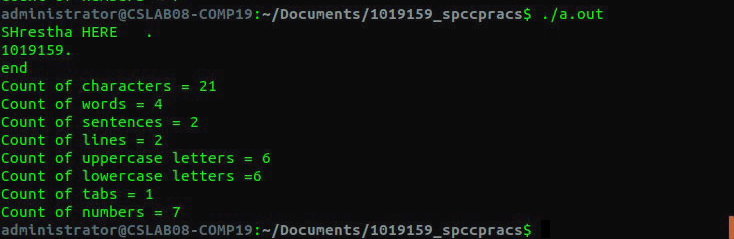
{

yylex();

printf("Count of characters = %d\nCount of words = %d\nCount of sentences = %d\nCount of lines

= %d\nCount of uppercase letters = %d\nCount of lowercase letters =%d\nCount of tabs = %d\ nCount of numbers = %d\n",no\_of\_chars, no\_of\_words, no\_of\_sent,no\_of\_lines,no\_of\_uppercase,no\_of\_lowercase, no\_of\_tabs, no\_of\_num );

return 0;

}

Prac 10

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<ctype.h>

 void main( )

{

char a[20];

int i,j=0;

FILE \*fp;

if(( fp = fopen("z.dat", "r")) != 0)

{

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

{

i=0;

if( strlen(a)==6)

{

i=i+3;

if(islower(a[i]))

printf("lw $t%d, (%c)\n", j++,a[i]);

else

{

for(i=3;i < strlen(a);i++)

{

if(isdigit(a[i]))

{

x= a[i] - '0';

k=k\*10 +x;

}

}

printf("li $t%d, %d\n", j,k);

}

i=i+2;

if(islower( a[i]))

printf("lw $t%d, (%c)\n", j++,a[i]);

else

{

for(i=3;i < strlen(a);i++)

{

if(isdigit(a[i]))

{

x= a[i] - '0';

k=k\*10 +x;

}

}

printf("li $t%d, %d\n", j,k);

}

i=i-1;

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

printf("add $t%d, $t%d, $t%d\n", j,j-1,j-2);

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

printf("sub $t%d, $t%d, $t%d\n", j,j-2,j-1);

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

printf("mul $t%d, $t%d, $t%d\n", j,j-2,j-1);

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

printf("div $t%d, $t%d, $t%d\n", j,j-2,j-1);

}

else if(strlen(a)==4)

{

i=i+3;

if( islower(a[i]))

{

printf("lw $t%d, %c\n",j,a[i]);

printf("copy %c, $t%d\n", a[i-3],j);

}

else

printf("li $t%d, %c\n",j,a[i]);

}

j=j+1;

}  } }