**var.l**

%{

#include "y.tab.h"

%}

%%

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

[0-9] return digit;

. return yytext[0];

\n return 0;

%%

int yywrap()

{

return 1;

}

**var.y**

%{

#include<stdio.h>

int valid=1;

%}

%token digit letter

%%

start : letter s

s : letter s

| digit s

|

;

%%

int yyerror()

{

printf("Its not an identifier!\n");

valid=0;

return 0;

}

int main()

{

printf("\nEnter a name to tested for identifier : ");

yyparse();

if(valid)

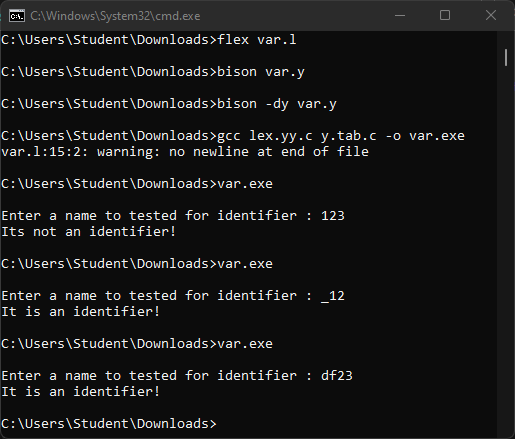
{

printf("It is an identifier!\n");

}

}

**OUTPUT :**



**calci.l**

%{

#include "y.tab.h"

#include<math.h>

extern double vbltable[26];

%}

%%

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

[\t];

[a-z] { yylval.vblno=yytext[0] - 'a'; return NAME; }

"$" { return 0;}

\n |

. return yytext[0];

%%

int yywrap(void)

{

return 0;

}

int main(void)

{

yyparse();

return 0;

}

int yyerror(void)

{

printf("error");

exit(1);

}

**calci.y**

%{

double vbltable[26];

int yylex(void);

%}

%union{

double dval;

int vblno;

}

%token <vblno> NAME

%token <dval> NUMBER

%left '-' '+'

%left '\*' '/'

%nonassoc UMINUS

%type <dval> expression

%%

statement\_list: statement '\n'

| statement\_list statement '\n'

;

statement: NAME '=' expression { vbltable[$1] = $3; }

| expression { printf(" = %g\n", $1); }

;

expression: expression '+' expression { $$ = $1 + $3; }

| expression '-' expression { $$ = $1 - $3; }

| expression '\*' expression { $$ = $1 \* $3; }

| expression '/' expression {

if($3 == 0.0)

yyerror("divide by zero");

else

$$ = $1 / $3;

}

| '-' expression %prec UMINUS { $$ = -$2; }

| '(' expression ')' { $$ = $2; }

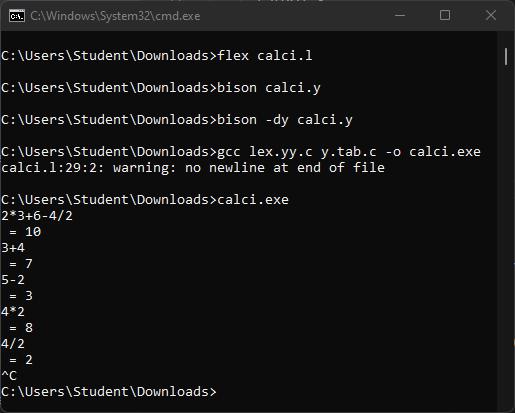
| NUMBER

| NAME { $$ = vbltable[$1]; }

;

%%

**OUTPUT :**



**3ac.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;}

[ \t] {}

. {return yytext[0];}

%%

int yywrap()

{

return 1;

}

**3ac.y**

%{

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char \* createT(); // Declaration for creating the temporary variables

int tempcount=0; // Global variable to track the number of temporary variables

%}

%union { char str[30]; }

%left '+'

%left '-'

%left '\*'

%left '/'

%token <str> Var // Defining the datatype of Tokens as str

%token <str> Num

%type <str> s // Defining the datatypes of Non-Terminals

%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; // Declaring the string array and pointer variable

sprintf(snum,"t%d",tempcount); // Returning a formatted String

ptr=snum; // Intializing the pointer with formatted stringaddress

tempcount++; // Temporary count

return ptr; // Returning the pointer

}

int main()

{

yyparse();

return 0;

}

int yyerror(char \*err)

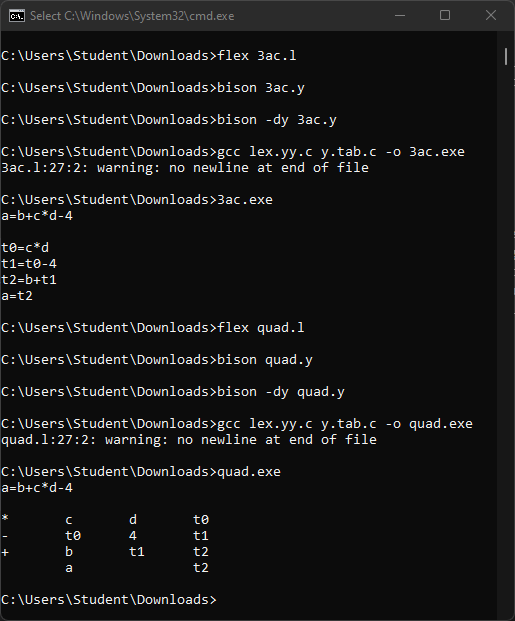
{

printf("\nInvlaid");

exit(0);

}

**OUTPUT :**



**quad.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;}

[ \t] {}

. {return yytext[0];}

%%

int yywrap()

{

return 1;

}

**quad.y**

%{

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char \* createT(); // Declaration for creating the temporary variables

int tempcount=0; // Global variable to track the number of temporary variables

%}

%union { char str[30]; }

%left '+'

%left '-'

%left '\*'

%left '/'

%token <str> Var // Defining the datatype of Tokens as str

%token <str> Num

%type <str> s // Defining the datatypes of Non-Terminals

%type <str> exp

%%

s : Var '=' exp {printf("\n \t%s \t\t%s\n",$1,$3);}

exp : '(' exp ')' {strcpy($$,$2);}

| exp '+' exp {strcpy($$,createT()); printf("\n+\t%s\t%s\t%s",$1,$3,$$);}

| exp '-' exp {strcpy($$,createT()); printf("\n-\t%s\t%s\t%s",$1,$3,$$);}

| exp '\*' exp {strcpy($$,createT()); printf("\n\*\t%s\t%s\t%s",$1,$3,$$);}

| exp '/' exp {strcpy($$,createT()); printf("\n/\t%s\t%s\t%s",$1,$3,$$);}

| Num {strcpy($$,$1);}

| Var {strcpy($$,$1);}

;

%%

char \* createT()

{

char snum[30],\*ptr; // Declaring the string array and pointer variable

sprintf(snum,"t%d",tempcount); // Returning a formatted String

ptr=snum; // Intializing the pointer with formatted stringaddress

tempcount++; // Temporary count

return ptr; // Returning the pointer

}

int main()

{

yyparse();

return 0;

}

int yyerror(char \*err)

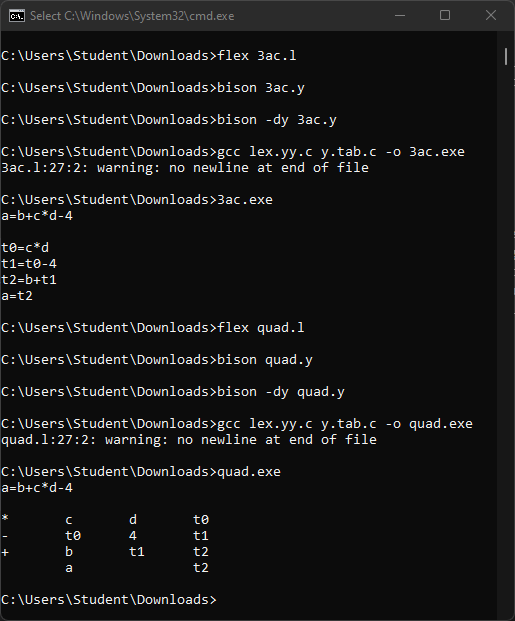
{

printf("\nInvlaid");

exit(0);

}

**OUTPUT :**



**INPUT :**

from tabulate import tabulate

ipCode = []

contents = []

print("Enter code (Ctrl-Z to save it) : \n")

while True:

try:

line = input()

except EOFError:

break

contents.append(line)

MNT = []

ALA = []

MDT = []

alaList = []

macroName = False

mntIndex = 0

mdtIndex = 1

alaIndex = 1

for i in contents:

if i[0:5].upper() == 'MACRO':

macroName = True

temp = i.upper().replace('MACRO', '').strip().split()

MNT.append([mdtIndex, temp[0], mntIndex])

mdtIndex += 1

alas = temp[1].split('=', 1)[0].split(',')

for j in alas:

alaList.append(j)

ALA.append([f'#{alaIndex}', j])

alaIndex += 1

MDT.append([mntIndex, f'{temp[0]} {temp[1]}'])

mntIndex += 1

elif i[0:4].upper() == 'MEND':

macroName = False

MDT.append([mntIndex, 'MEND'])

mntIndex += 1

else:

if macroName == True:

for j in alaList:

if j in i:

i = i.replace(j, f'#{alaList.index(j)+1}')

MDT.append([mntIndex, i])

mntIndex += 1

else:

ipCode.append(i)

print('\n===========PASS 1===========\n-----------MNT-----------')

print(tabulate(MNT, tablefmt="simple\_grid", headers=['Index', 'Macro Name', 'MDT Index']))

print('\n-----------ALA-----------')

print(tabulate(ALA, tablefmt="simple\_grid", headers=['Positional Arguments', 'Dummy Arguments']))

print('\n-----------MDT-----------')

print(tabulate(MDT, tablefmt="simple\_grid", headers=['Index', 'Code']))

i = 0

ala\_ip = []

ala\_code = []

ala\_index = []

while i < len(ipCode):

j = 0

while j < len(MNT):

if ipCode[i].split()[0] == MNT[j][1]:

startIndex = MNT[j][2]

try:

endIndex = MNT[j+1][2] - 1

except IndexError:

endIndex = len(MDT) - 1

ipIndex = i

ala\_ip += ipCode[i].split()[1].split(',')

ala\_code += MDT[startIndex][1].split()[1].split(',')

for k in MDT[startIndex][1].split()[1].split(','):

for x in ALA:

if x[1] == k:

temp = x[0]

ala\_index.append(temp)

del ipCode[i]

for k in range(startIndex, endIndex):

temp = MDT[k][1]

for x in range(0, len(ala\_ip)):

temp = temp.replace(ala\_index[x], ala\_ip[x])

temp = temp.replace(ala\_code[x], ala\_ip[x])

ipCode.insert(ipIndex, temp)

ipIndex += 1

break

j += 1

i += 1

pass2 = []

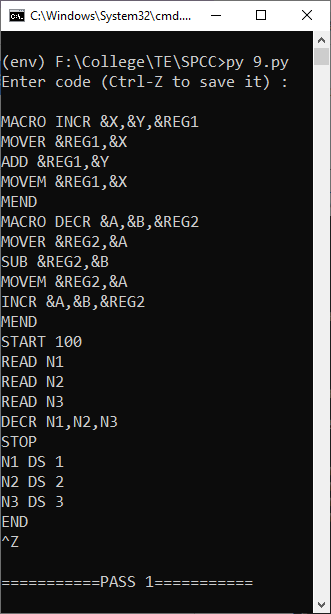
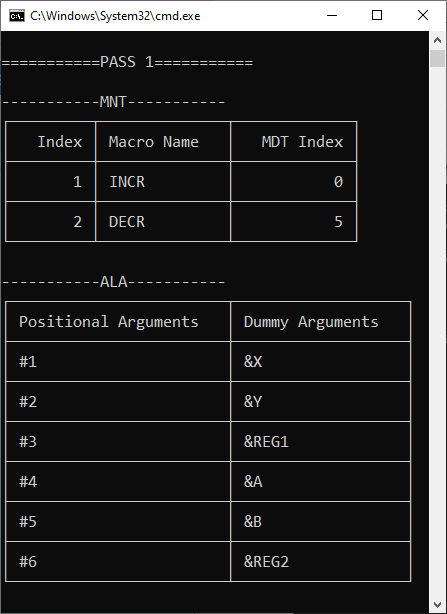
for i, index in enumerate(ipCode):

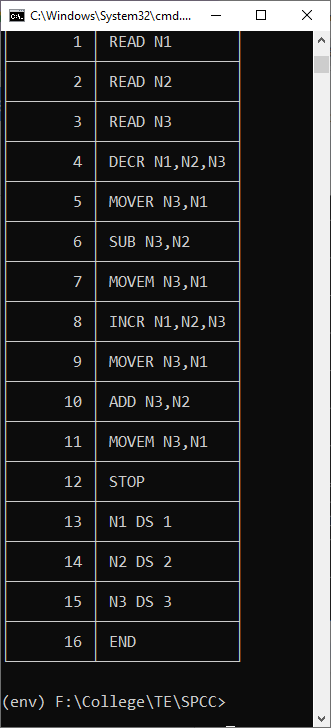
pass2.append([i, index])

print('\n===========PASS 2===========')

print(tabulate(pass2, tablefmt="simple\_grid", headers=['Index', 'Code']))

**OUTPUT :**

**INPUT :**

from tabulate import tabulate

no\_of\_prod = int(input("Enter no. of productions : "))

startSymbol = input("Enter start symbol : ")

productions = []

for i in range(no\_of\_prod):

temp = input(f"Production {i+1} : ").replace(" ", "").split('->')

try:

vals = temp[1].split('|')

for j in vals:

productions.append(f"{temp[0]}->{j}")

except Exception as e:

print(e)

operators = input('Enter all operators separated by commas : ')

operatorsList = operators.replace(" ", "").split(",")

operatorsList.append('$')

operatorPrecedenceTable = []

for i in range(0, len(operatorsList)):

temp = []

for j in range(0, len(operatorsList)):

temp.append(input(f'Operator precedence of {operatorsList[i]}, {operatorsList[j]} : '))

operatorPrecedenceTable.append(temp)

temp = []

for i in operatorPrecedenceTable:

temp.append([operatorsList[operatorPrecedenceTable.index(i)]] + i)

print('\n---------Operator Precedence Table---------')

print(tabulate(temp, tablefmt="simple\_grid", headers=operatorsList))

ip = input('\nEnter string to parse : ')

ip += ',$'

ip = ip.split(',')

prodRHS = [x.split('->')[1] for x in productions]

stack = ['$']

track = []

while True:

x = stack[-1]

y = ip[0]

op = operatorPrecedenceTable[operatorsList.index(x)][operatorsList.index(y)]

if x == startSymbol and y == '$':

if len(stack) == 2:

track.append([''.join(stack), ''.join(ip), 'ACCEPT'])

break

else:

for j in reversed(range(1, len(stack))):

eq = ''.join(stack[j:len(stack)])

if eq in prodRHS:

temp = prodRHS.index(eq)

track.append([''.join(stack), ''.join(ip), f'Reduce {productions[temp]}'])

stack[j:len(stack)] = productions[temp].split('->')[0]

break

else:

if op == '<' or op == '=':

track.append([''.join(stack), ''.join(ip), 'Shift'])

stack.append(y)

ip = ip[1:]

elif op == '>':

for j in reversed(range(1, len(stack))):

eq = ''.join(stack[j:len(stack)])

if eq in prodRHS:

temp = prodRHS.index(eq)

track.append([''.join(stack), ''.join(ip), f'Reduce {productions[temp]}'])

stack[j:len(stack)] = productions[temp].split('->')[0]

break

else:

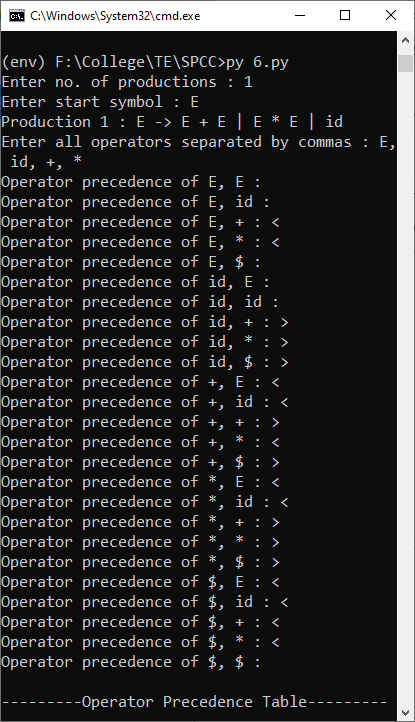
track.append([''.join(stack), ''.join(ip), 'REJECT'])

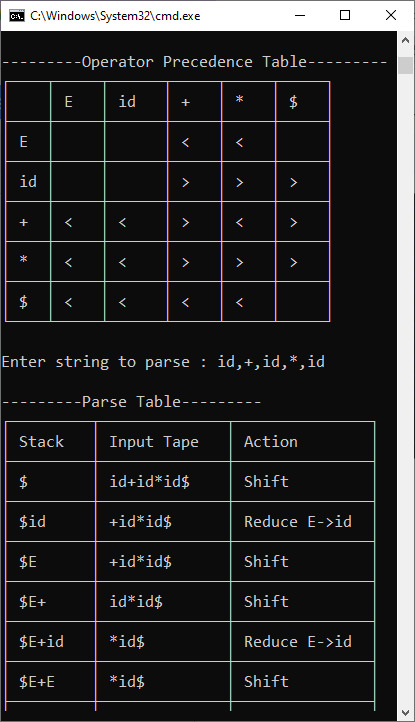
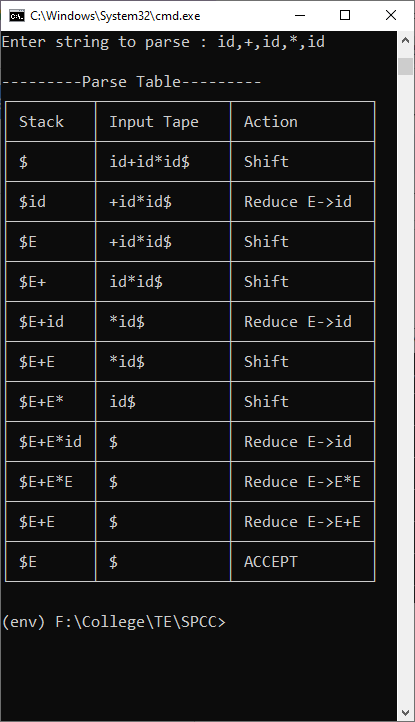
break

print('\n---------Parse Table---------')

print(tabulate(track, tablefmt="simple\_grid", headers=['Stack', 'Input Tape', 'Action']))

**OUTPUT :**



If.l

%{

#include "y.tab.h"

%}

%%

"if" {return IF;}

[sS][0-9]\* {return S;}

"<"|">"|"=="|"<="|">="|"!=" {return RELOP;}

[0-9]+ {return NUMBER;}

[a-z][a-zA-Z0-9\_]\* {return ID;}

\n {return NL;}

. {return yytext[0];}

%%

int yywrap()

{

return 1;

}

If.y

%{

#include<stdio.h>

#include<stdlib.h>

int count=0;

%}

%token IF RELOP S NUMBER ID NL

%%

stmt: if NL {printf("No. of nested if statements : %d\n",count);exit(0);};

if : IF'('cond')''{'if'}else{'S'}' {count++;}

|IF'('cond')''{'if'}' {count++;}

|S;

cond: x RELOP x ;

x:ID | NUMBER;

%%

int yyerror(char \*msg)

{

printf("The statement is Invalid\n");

exit(0);

}

main()

{

printf("Enter the statement : ");

yyparse();

}

1, 3, 5, 7, 9, 11, 13, 15

2, 4, 6, 8, 10, 12, 14, 16