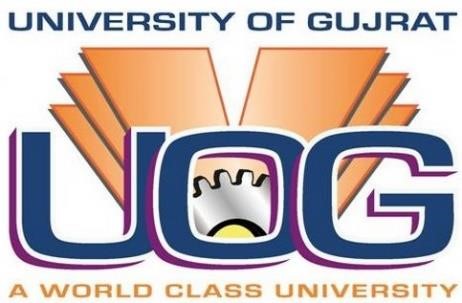
Assignment # 01

# Topic: LEXICAL ANALYSIS



**Submitted To**

Sir Toqueer

**Submitted By**

**Ahmad Pasha**

**(17271519-002)**

**Section**

Human Computer Interaction

**Due Date**

Thursday, October 15, 2020

Fall – 2020

Department of Computer Science

Faculty of Computing and Information Technology

University of Gujrat, Hafiz Hayat Campus

Lexical Analysis:

# Lex Code:

%{

#include<stdio.h>

%}

DIGIT [0-9]

ID [A-Za-z\_][A-Za-z0-9\_]\*

CLASS\_POINT [A-Za-z\_][A-Za-z0-9\_]\*[.][A-Za-z\_][A-Za-z0-9\_]\*

SYM [+|-|\*|/|>|=|<]

N\_OP !=

NUMBER [0-9]+

KEYWORD False|await|else|import|pass|None|break|except|in|raise|True|class|finally|is|return|and|continue|for|lambda|try|as|def|from|nonlocal|while|assert|del|global|not|with|async|elif|if|or|yield

PUNCTUATION [:|,|"|'|.]

SPECIAL [!|&|%|$|?]

DECIMAL [0-9]+[.][0-9]+

LITERAL ["][^"\n]\*["]

LITERAL\_2 ['][^'\n]\*[']

%option noyywrap

%%

^[ ]\* {fprintf(yyout,"\nINDENTATION: %s\n",yytext);}

"#"[^\n]\* {fprintf(yyout,"\nCOMMENT: %s\n",yytext);}

"]"|"["|"}"|"{"|"("|")" {fprintf(yyout,"\nDELIMITER: %s\n",yytext);}

{LITERAL}|{LITERAL\_2} {fprintf(yyout,"\nLITERAL: %s\n",yytext);}

{NUMBER} {fprintf(yyout,"\nNUMBER: %s\n",yytext);}

{KEYWORD} {fprintf(yyout,"\nKEYWORD: %s\n",yytext);}

{SYM}|{N\_OP}|{SPECIAL} {fprintf(yyout,"\nOPERATOR: %s\n",yytext);}

{ID}|{CLASS\_POINT} {fprintf(yyout,"\nIDENTIFIER: %s\n",yytext);}

{DECIMAL} {fprintf(yyout,"\nDECIMAL: %s\n",yytext);}

{PUNCTUATION} {fprintf(yyout,"\nPUNCTUATION: %s\n",yytext);}

<<EOF>> {fprintf(yyout,"\nEnd of File %s\n",yytext); return 0;}

%%

int main()

{

extern FILE \*yyin, \*yyout;

yyin = fopen("Input.txt", "r");

yyout = fopen("Output.txt", "w");

yylex();

return 0;

}

# Source Code (content of “Input.txt”):

# Weather Forcasting Project

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

weather\_data=pd.read\_csv(r'C:\Users\Talha Chuhdary\Downloads\weather\_data\_test.csv',

parse\_dates=['datetime'], sep=';', decimal=',',infer\_datetime\_format=True)

print(weather\_data.shape)

temp\_df=weather\_data[["datetime","T\_mu"]]

print(weather\_data.shape)

temp\_df.head(10)

mask=(temp\_df['datetime'] >= '2016-01-01')&(temp\_df['datetime']<='2019-05-01')

temp\_df=temp\_df.loc[mask]

temp\_df.set\_index("datetime", inplace=True)

from IPython.display import display

display(temp\_df.head(10))

display(temp\_df.tail(10))

temp\_df.describe()

print(temp\_df.loc[temp\_df['T\_mu'] == temp\_df['T\_mu'].max()])

print(temp\_df.loc[temp\_df['T\_mu'] == temp\_df['T\_mu'].min()])

plt.figure(figsize=(16,10),dpi=100)

plt.plot(temp\_df.index,temp\_df.T\_mu, color='tab:red')

plt.gca().set(title="Daily Temperature from 2016 - 2019",xlabel='Date',ylabel="Dagree(in celsius)")

plt.show()

temp\_df\_mean=temp\_df.T\_mu.rolling(window=30).mean()

temp\_df\_mean.plot(figsize=(20,15))

from statsmodels.tsa.seasonal import seasonal\_decompose

result\_add=seasonal\_decompose(temp\_df.T\_mu,model='additive',extrapolate\_trend='freq',freq=365)

plt.rcParams.update({'figure.figsize':(10,10)})

result\_add.plot().suptitle('Additive Decomposition',fontsize=22)

plt.show()

predicted\_df=temp\_df["T\_mu"].to\_frame().shift(1).rename(columns={"T\_mu":"T\_mu\_pred"})

actual\_df=temp\_df["T\_mu"].to\_frame().rename(columns={"T\_mu":"T\_mu\_actual"})

one\_step\_df=pd.concat([actual\_df,predicted\_df],axis=1)

one\_step\_df=one\_step\_df[1:]

one\_step\_df.head(10)

from sklearn.metrics import mean\_squared\_error as MSE

from math import sqrt

temp\_pred\_err=sqrt(MSE(one\_step\_df.T\_mu\_actual,one\_step\_df.T\_mu\_pred))

print("The RMSE is ",temp\_pred\_err)

import itertools

p=d=q=range(0,2)

pdq=list(itertools.product(p,d,q))

seasonal\_pdq=[(x[0],x[1],x[2],12) for x in list (itertools.product(p,d,q))]

print('Example of parametes combinations od seasinal ARIMA....')

print('SARIMAX: {} X {} '.format(pdq[1],seasonal\_pdq[1]))

print('SARIMAX: {} X {} '.format(pdq[1],seasonal\_pdq[2]))

print('SARIMAX: {} X {} '.format(pdq[2],seasonal\_pdq[3]))

print('SARIMAX: {} X {} '.format(pdq[2],seasonal\_pdq[4]))

import warnings

warnings.filterwarnings("ignore")

import statsmodels.api as sm

for param in pdq:

for param\_seasonal in seasonal\_pdq:

try:

mod = sm.tsa.statespace.SARIMAX(one\_step\_df.T\_mu\_actual,

order=param,

seasonal\_order=param\_seasonal,

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

print('SARIMA{}x{}12 - AIC:{}'.format(param, param\_seasonal, results.aic))

except:

continue

import statsmodels.api as sm

mod = sm.tsa.statespace.SARIMAX(one\_step\_df.T\_mu\_actual,order=(1, 1, 1),

seasonal\_order=(1, 1, 1, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

results.plot\_diagnostics(figsize=(15,12))

plt.show()

pred=results.get\_prediction(start=pd.to\_datetime('2017-05-19'),dynamic=False)

pred\_ci=pred.conf\_int()

ax=one\_step\_df.T\_mu\_actual['2015':].plot(label='observed')

pred.predicted\_mean.plot(ax=ax, label='Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1],

color='k', alpha=0.2)

ax.set\_xlabel('Date')

ax.set\_ylabel('Temperature (in celsius)')

plt.ylim([-20,30])

plt.legend()

plt.show()

y\_forecasted=pred.predicted\_mean

y\_truth=one\_step\_df.T\_mu\_actual['2017-05-19':]

print(y\_forecasted.shape)

print(y\_truth.shape)

mse=sqrt(MSE(y\_truth, y\_forecasted).mean())

print('The Mean squared Error of our Forecast is {}'.format(round(mse, 2)))

pred\_dynamic=results.get\_prediction(start=pd.to\_datetime('2017-05-19'),dynamic=True,full\_results=True)

pred\_dynamic\_ci=pred\_dynamic.conf\_int()

ax=one\_step\_df.T\_mu\_actual['2015':].plot(label='observed',figsize=(20,15))

pred\_dynamic.predicted\_mean.plot(label='Dynamic forecast',ax=ax)

ax.fill\_between(pred\_dynamic\_ci.index,

pred\_dynamic\_ci.iloc[:, 0],

pred\_dynamic\_ci.iloc[:, 1],

color='k', alpha=0.25)

ax.set\_xlabel('Date')

ax.set\_ylabel('Temperature (in celsius)')

plt.ylim([-20,30])

plt.legend()

plt.show()

# OUTPUT:

