**The CFG**

S -> AcB

A -> aAb

A->ab

B->dBe

B->de

**The code**

|  |
| --- |
| def checkPattern(string, pattern):      l = len(pattern)      if len(string) < l:  return False    for i in range(l - 1):     x = pattern[i]  y = pattern[i + 1]     last = string.rindex(x)          if last == -1 or first == -1 or last > first:  return False      return True def parser():  f = open("two.txt","r")  for word in f:  ab=0  de=0  c=0  if checkPattern(word,"abcde")==True:  for letter in word:    if letter == "a" or letter == "b":  if letter == "a":  ab+=1  else:  ab-=1  elif letter == "c":  c+=1  elif letter == "d" or letter == "e":  if letter == "d":  de+=1  else:  de-=1  else:  if letter.isspace()==True:  pass  else:  print("unknown word")   if ab!=0 or de!=0 or c!=1:  print(word," cannot be parsed")  else:  print(word," parsed")   else:  print(word,"cannot be parsed")    parser() |
|  |
|  |

**How to run the code**

We need to add the strings we want to parse inside the file two.txt

Then we need to run the command “python3 two.py” on the cmd line

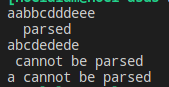
**Two.txt file contents**

aabbcdddeee

abcdedede

A

**Output:**



**Methodology**

In this system we will be designing and using two functions parser() and checkPattern().

Parser is the main function that checks the number of a,b,c,d,e in the string and checkPattern() is used to validate if the sequence a>b>c>d> e maintained.

**Parser function**

File is opened using the open() command and for each strings are checked using a for loop.

In each iteration three variables are initialized ab, de, c to keep track of the occurrences of a,b,c,d,e

def parser():

f = open("two.txt","r")

for word in f:

a=0

b=0

c=0

CheckPattern function is used to see if the pattern is followed. (More discussed in the next checkPattern paragraph)

if checkPattern(word,"abcde")==True:

If the letter ‘a’ or ’b’ is encountered the command enters this if else statement where if a is encountered the variable ab increments by 1 and if ab is encountered the variable decrements by 1.

if letter == "a" or letter == "b":

if letter == "a":

ab+=1

else:

ab-=1

Same is followed for the characters d and e in an elif condition where if d is encountered the variable de is incremented and if e is encountered de is decremented.

elif letter == "d" or letter == "e":

if letter == "d":

de+=1

else:

de-=1

Another elif condition checks for the occurence of the character c and increments the variable c if it encounters the char.

elif letter == "c":

c+=1

The command is then passed to another if-else statement which check the values of ab, de and c to determined if the string can be parsed or not.

The variable ab should be zero at the end of parsing becausing only when the number of occurrences of the characters a and b are the same should the string be accepted.

The variable de should be zero at the end of parsing becausing only when the number of occurrences of the characters d and e are the same should the string be accepted.

The variable c should be one at the end of parsing becausing only when the number of occurrences of the character c is one should the string be accepted.

if ab!=0 or de!=0 or c!=1:

print(word," cannot be parsed")

else:

print(word," parsed")

**Check Pattern function**

Two parameters are passed into this function string and pattern. string holds the string that we want to parse and pattern is the sequence of characters that we want our parsed string to follow. In our case it is a>b>c>d>e.

def checkPattern(string, pattern):

Length of pattern is stored in the variable l. If the length of string is less than the length of pattern the string is rejected because the string needs to have as many if not more number of characters as the pattern

l = len(pattern)

if len(string) < l:

return False

A for loop is run that iterates until there is no character left to parse in pattern.

for i in range(l - 1):

Two variables x and y are instantiated in every loop. Variable x holds the ith character in the pattern and variable y holds the (i+1)th character. For instance, in our case, if x hold ‘a’ y will hold ‘b’. If x holds ‘d’ y will hold ‘e’.

x = pattern[i]

y = pattern[i + 1]

In each for loop two other variables are declared last and first. The ‘first’ variable holds the first occurrence of the character in variable y and the ‘last’ variable holds the last occurrence of variable y in string. For instance, in the string ‘aaabbbcddee ’, if the value of x is ‘a’ the last occurrence of a is in the index 2 and if the value of y is b the first occurrence of b in the index 3. Therefore the values of last and first will be 2 and 3.

last = string.rindex(x)

first = string.index(y)

Then we enter an if statement. If the value of last and first is ‘-1’ it means the character of the pattern doesnt exist in the string which means the string is invalid. If the value of the last is greater than the value of first the string will also become invalid. For instance in the string “ababcdede” the last occurrence of a is in the index 2 but the first occurrence of b is in the index 1. Last > first. Which means a exists after b and this makes the string invalid.

if last == -1 or first == -1 or last > first:

return False

Other all the conditions are met the string should be deemed valid and True should be returned

return True