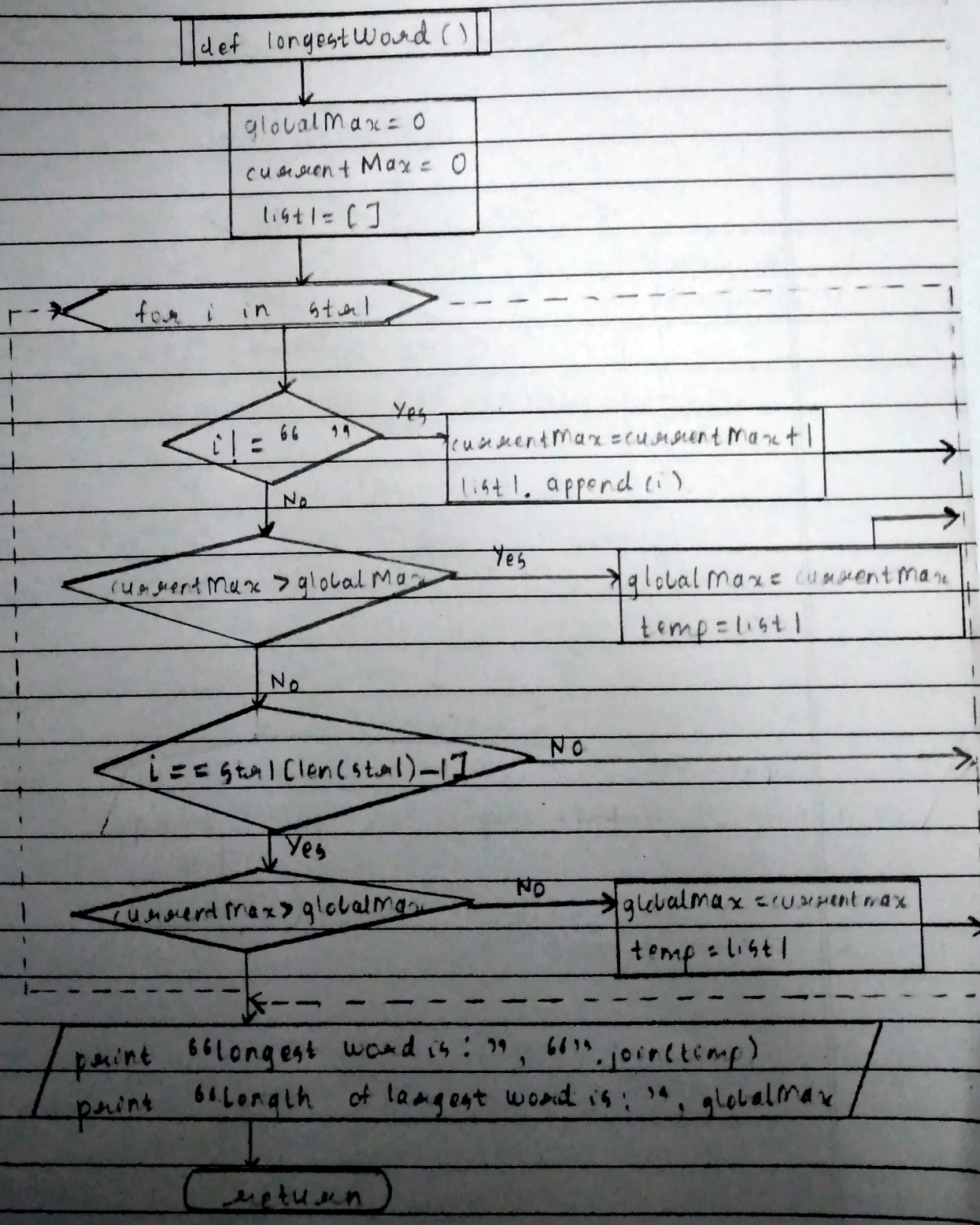
```
#Input:
str1 = input("Enter string: ")
substr1 = input("Enter substring: ")
occur1 = input("Enter character whose occurence is to be calculated: ")
occur2 = str1.split()
index1 = []
for i in occur2:
    for j in occur2:
        if(i == j and j not in index1):
            index1.append(i)
        else:
            continue

def longestWord():
    globalMax = 0
    currentMax = 0
    list1 = []
    for i in str1:
        if (i != " "):
            currentMax = currentMax + 1
            list1.append(i)
        else:
            if (currentMax > globalMax):
                globalMax = currentMax
                temp = list1
            currentMax = 0
            list1 = []
        if (i == str1[len(str1) - 1]):
            if (currentMax > globalMax):
                globalMax = currentMax
                temp = list1
    print("Longest word is: ", "".join(temp))
    print("Length of largest word is: ", globalMax)


def palindrome():
    rev1 = str1
    if (str1[::-1] == rev1):
        print("String is palindrome")
    else:
        print("String is not palindrome")


def charfrequency():
    temp = 0
    for i in str1:
        max1 = 0
        for j in range(0, len(str1)):
            if (str1[j] != " " and str1[j] == occur1):
                max1 = max1 + 1
            else:
                continue
```

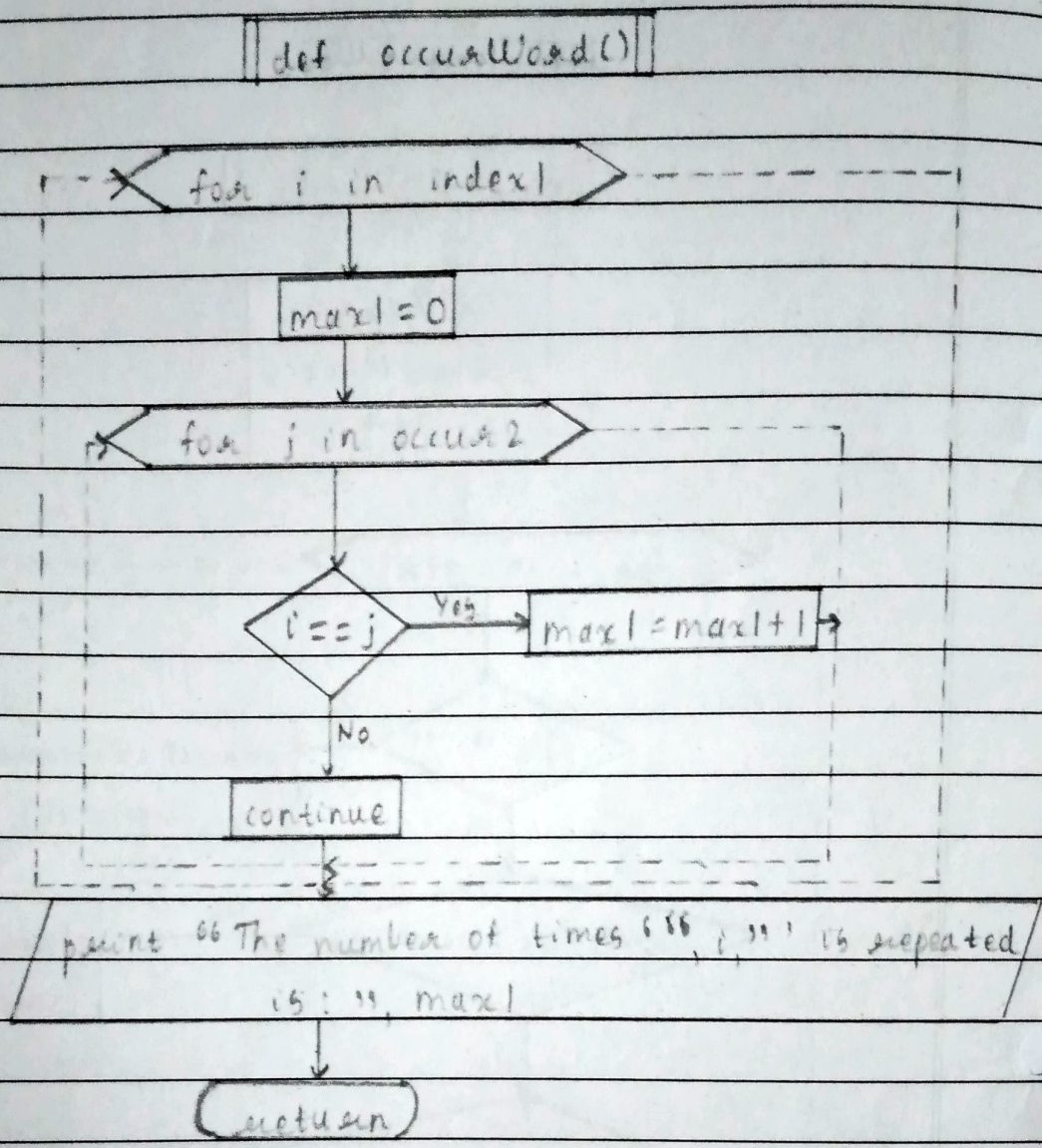
```
        if (max1 > temp):
            temp = max1
        else:
            continue
    print("Number of times the character '", occur1, "' occurs: ", temp)


def indexSubstring():
    for i in range(len(str1) - len(substr1) + 1):
        if str1[i:i + len(substr1)] == substr1:
            print("Substring present in string at index: ", i)
            break
    else:
        print('Substring not present')

def occurWord():
    for i in index1:
        max1 = 0
        for j in occur2:
            if(i == j):
                max1 = max1 + 1
            else:
                continue
        print("The number of times '",i,"' is repeated is: ",max1)

flag = 1
while flag == 1:
    print("1. Display word with longest word: \n")
    print("2. The occurence of '",occur1,"' in the string is: \n")
    print("3. Check whether is string is palindrome or not: \n")
    print("4. Display index of first appearance of the substring: \n")
    print("5. Occurrence of each word in the given string: \n")
    print("6. Exit")
    ch = int(input("Enter your choice: "))
    if(ch == 1):
        longestWord()
    elif(ch == 2):
        charfrequency()
    elif(ch == 3):
        palindrome()
    elif(ch == 4):
        indexSubstring()
    elif(ch == 5):
        occurWord()
    elif(ch>6 or ch<1):
        print("Enter valid choice")
    elif(ch == 6):
        flag = 0
```

→ Flowchart for def longestWord():

~~def longestWo~~

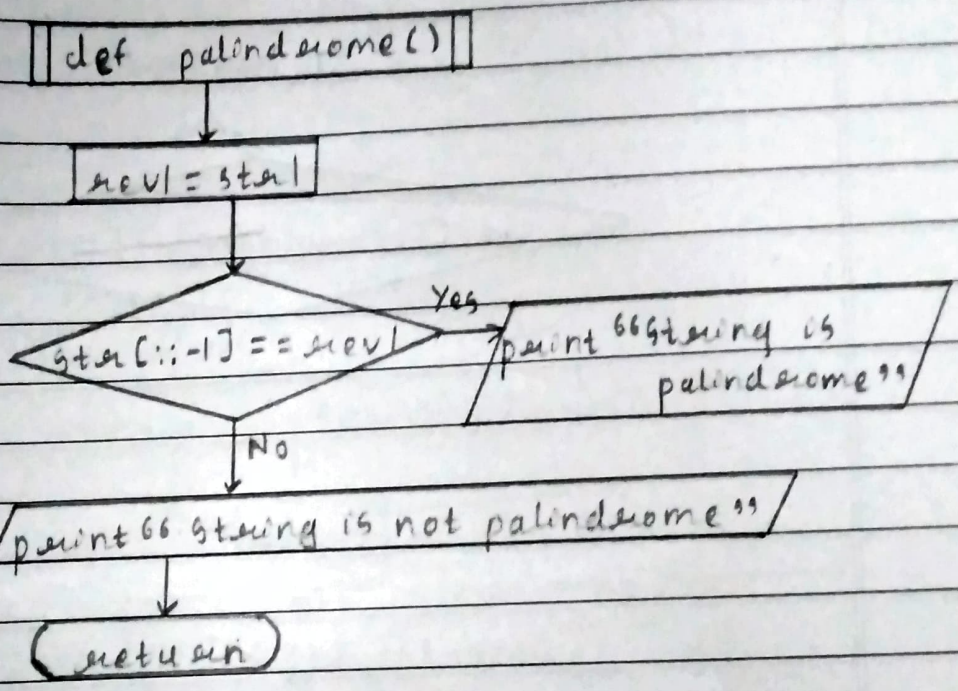


→ Flowchart for def occurWord()

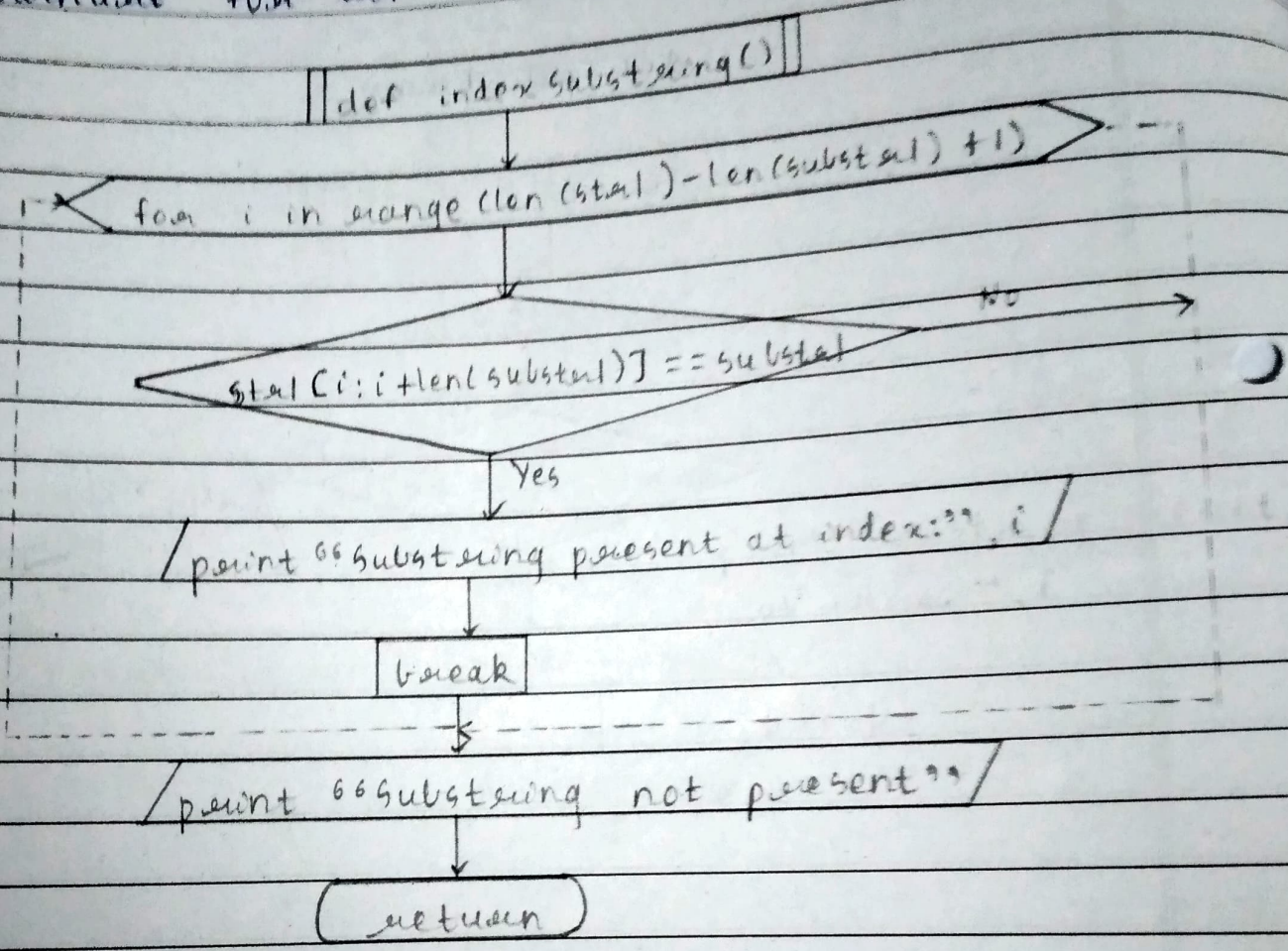


→ Flowchart for def palindrome()

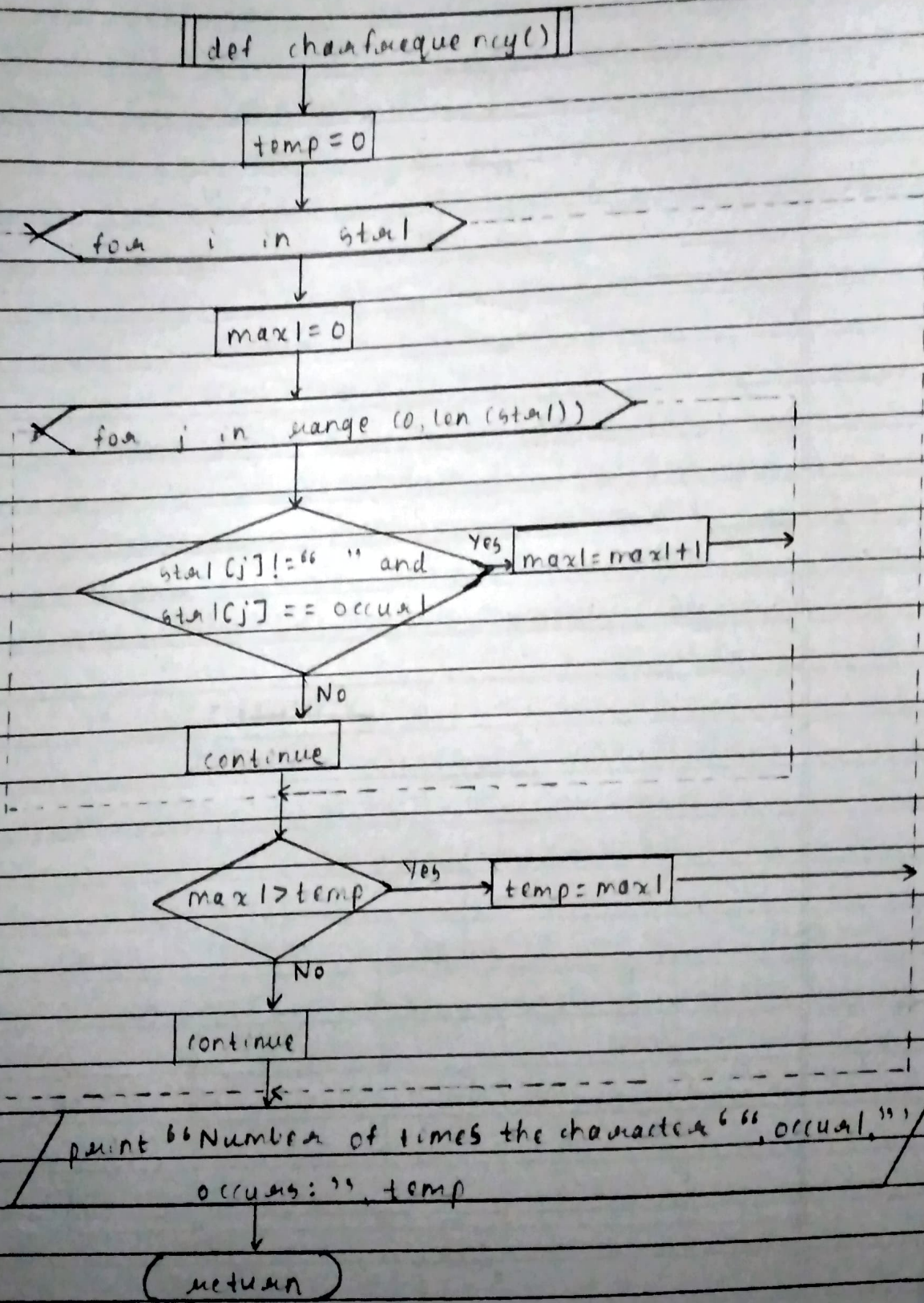
~~def palindrom~~



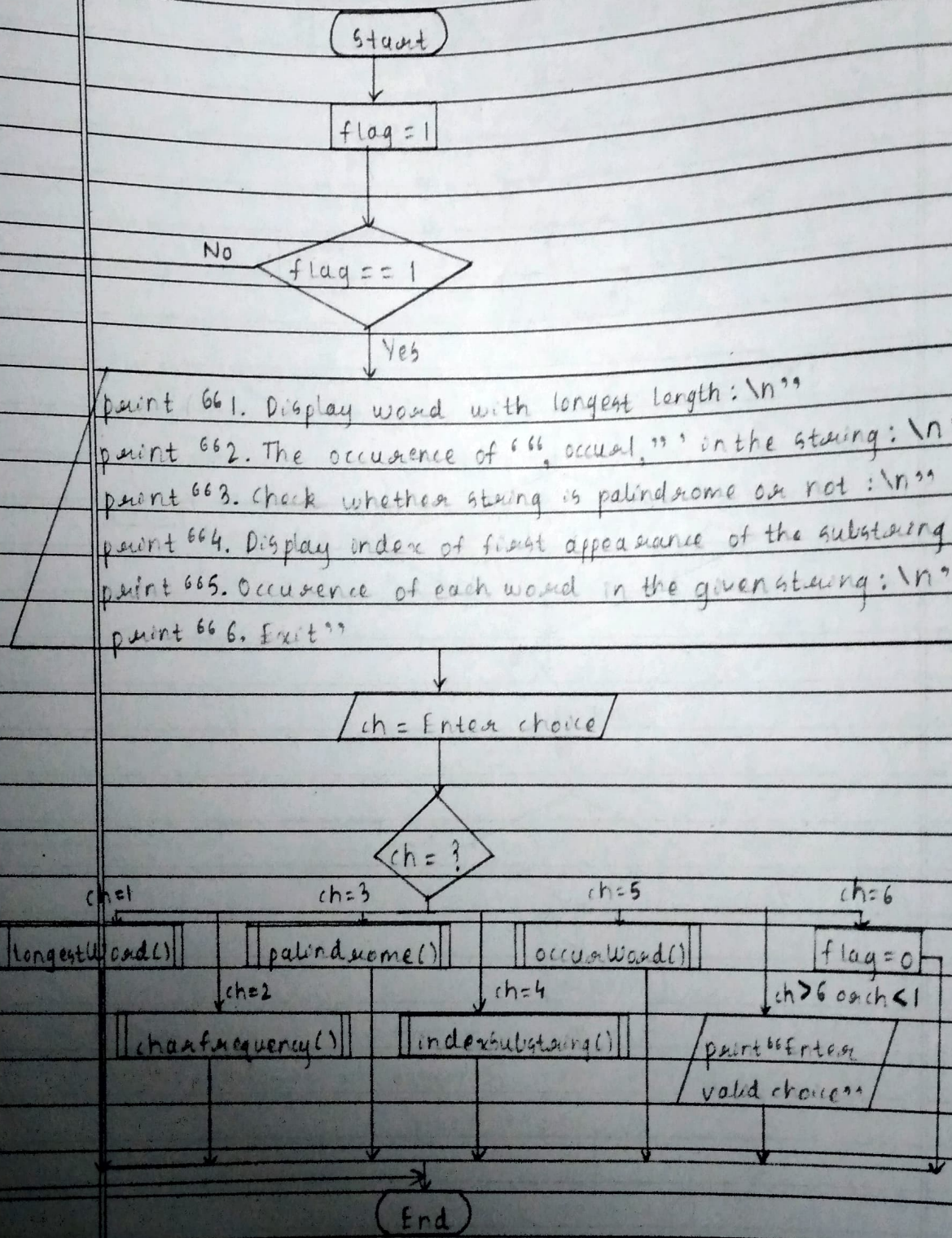
→ Flowchart for def indexSubstring()



→ Flowchart for def charfrequency()



→ Flowchart for menu



→ Pseudocode for def longestWord()

1. Initialize globalMax = 0
   Initialize currentMax = 0
   list1 = []
2. for i in str1 do
   begin
   if (i != " ") then
   currentMax = currentMax + 1
   list1.append(i)
   else
   if (currentMax > globalMax) then
   globalmax = currentmax
   store temp = list1
   initialize currentmax = 0
   list1 = []
   if (i == str1[len(str1) - 1] then
   if (currentMax > globalMax) then
   globalMax = currentMax
   store temp = list1
3. end
4. Display longest word is , temp
   Display length of largest word is , globalMax
5. return

→ Pseudocode for def charfrequency()

1. Initialize temp=0
2. for i in str1 do
begin
    initialize max1=0
        for j in range (0, len(str1)) do
        begin
            if (str1[j] != " " and str1[j]==occur1) then
                max1=max1+1
            else
                continue
        end
    if (max1 > temp) then
        store temp=max1
    else
        continue
end
3. Display Number of times the character, occur1, occurs :, temp

→ Pseudocode for def palindrome

1. Store rev1=str1
2. if (str1[::-1] == rev1) then
        display string is palindrome
else
        Display string is not palindrome
3. return

→ Pseudocode for def indexSubstring()

1. for i in range(len(str1) - len(substr1) + 1) do
begin
    if str1[i : i + len(substr1)] == substr1 then
        Display substring is present in string at
        index: , i
        break
    else
        Display substring is not present
end
2. return

→ Pseudocode for def occurWord()

1. for i in index1 do
begin
    initialize max1 = 0
    for j in occur2 do
    begin
        if (i == j) then
            max1 = max1 + 1
        else
            continue
    end
end
2. Display The number of times, i is repeated is: , max1
3. return

→ Pseudocode ~~flowchart~~ for menu

1. Start
2. Initialize flag = 1
3. while flag = 1 do

begin

Display 1. Display word with longest length:
Display 2. The occurence of, occuar, in the string is:
Display 3. check whether the string is palindrome or not:
Display 4. Display index of first appearance of the substring:
Display 5. Occurence of each word in the goven string
Display 6. Exit
ch = Enter your choice
if (ch==1) then
    call function longestWord()
elif (ch==2) then
    call function charfrequency()
elif (ch==3) then
    call function palindrome()
elif (ch==4) then
    call function indexSubstring()
elif (ch==5) then
    call function occurWord()
elif (ch>6 or ch<1) then
    Display Enter valid choice
elif (ch==6) then
    initialize flag = 0

end
4. Stop

Q1. What is frequency count? Why is frequency count important in the analysis of algorithm?

Ans. • Efficiency of an algorithm can be measured by inserting a counter in the algorithm in order to count the number of times the basic operation is executed that counter is called frequency count (FC).

• Frequency count in algorithm gives you that how many times your program is run that is it is dependent on the loop used in the program.

• However, the loop defines the time complexity of your program.

• So, it means that the frequency count defines the time complexity of the program.

• Also, the frequency count gives us the order of growth of an algorithm.

Q2. Write an algorithm to compute the sum of the digits of the given number. Justify that your algorithm satisfies all the given characteristics of an algorithm.

Ans. → Algorithm :-

1. Read n = Enter number
2. Initialize total = 0
3. while (n != 0) do
   begin
   Store total = total + n % 10
   Store n = n // 10
4. Display total
5. Stop

→ Justification:

1) Input: The given algorithm has a well defined input 'n'.
2) Output: The algorithm has a well defined output 'total'.
3) Finiteness: The given ~~counter~~ algorithms terminates when the input "n" is given; the looping is done as per the condition and the output 'total' is displayed.
4) Definiteness: There are no complex and ambiguous steps in the above algorithm.
5) Effectiveness: The above algorithm has an input statement, output statement, assigning statement, conditional statement and can be easily converted into a program hence satisfying the effectiveness property of the algorithm.

Q3. Write an algorithm to perform sparse matrix addition and state its time complexity.

Ans. Algorithm:-

1. Initialize i=1, j=1, k=1
2. Assign l1 = c[0][2]
   Assign l2 = d[0][2]
3. Let e[0][0] = c[0][0]
   let e[0][1] = c[0][1]
4. while i<=l1 and j<=l2 go to next step, otherwise go to step 21
5. if c[i][0] == d[j][0] go to next step, otherwise go to step 15

6. If c[i][1] == d[j][1] go to next step, otherwise go to step 9
7. Assign e[k][0] = c[i][0]
   e[k][1] = c[i][1]
   e[k][2] = c[i][2] + d[j][2]
8. increment i, j, k
9. if c[i][1] > d[j][1] go to next step, otherwise go to step 12
10. Assign e[k][0] = d[j][0]
    e[k][1] = d[j][1]
    e[k][2] = d[j][2]
11. increment j, k
12. if c[i][1] < d[j][1] go to next step, otherwise go to step 4
13. Assign e[k][0] = ~~d[j]~~ c[i][0]
    e[k][1] = c[i][1]
    e[k][2] = c[i][2]
14. increment i, k
15. if c[i][0] > d[j][0] go to next step, otherwise go to step 18
16. Assign e[k][0] = d[j][0]
    e[k][1] = d[j][1]
    e[k][2] = d[j][2]
17. increment j, k
18. if c[i][0] < d[j][0] go to next step, otherwise go ~~st~~ to step 4
19. Assign e[k][0] = c[i][0]
    e[k][1] = c[i][1]
    e[k][2] = c[i][2]
20. increment i, k

21. while i<=l2 and i>=l1 go to next step, otherwise go to step 24
22. Assign e[k][0] = d[j][0]
e[k][1] = d[j][1]
e[k][2] = d[j][2]
23. increment j, k
24. while j>=l2 and i<=l1 go to next step, otherwise go to step 26
25. Assign e[k][0] = c[i][0]
e[k][1] = c[i][1]
e[k][2] = c[i][2]
24. increment i, k
26. Assign e[0][2] = k-1
27. Print array e
28. stop

→ Time complexity:

• If l1 and l2 are number of non-zero elements in first and second sparse matrix respectively, then time complexity is O(l1+l2).