1.6 Strings

All the inputs to the problems that we have encountered so far involved numerical information. What if we want to work with strings? Or in layman's terms, words like "you", "are", "awesome", or "no devil lived on". Yes, they are called strings in programming parlance. Strings are composed of symbols or characters, both printable and non-printable. Characters are the letters from a-z, A-Z, digits from 0-9, the punctuation marks, other symbols that you can type and can be printed, and those that cannot be displayed or printed like the new line character (when you press the enter key), even the space, etc.

This means that of the following lines are strings as well:

```
Philippines
Central Visayas
VII
123456
390ADF
PADAYON
p@ssw0rd
    (\_/)
   (='.'=)
   (")_(")
...
```

Like numbers, strings have positions also. Take the number 1092 and the string ciriaco. For the number input, we the ones digit, the tens, hundreds and thousands digits. For ciriaco, we say that the first character c is found in position 1. i is in position 2, r in position 3, and o is in position 7.

Strings have lengths. The length is simply the number of characters/symbols found in the string. In the case of ciriaco, its length is 7, since there are 7 characters in it.

When working with integers, it quite challenging to extract the digits. Again take 1092. How will you extract 9 from it and assign it to a variable? Or take any number n, what is its hundreds digit? We will solve this problem in c programming. We promise!

For strings, that is pretty much straight forward. This operation is called indexing. The positions that started with 1 that we mentioned early on, are called the indices. Take the string ciriaco again. The index of the symbol a is 5. Or we say that the symbol a is found in index 5.

Indexing requires a variable that represents a string. Let's say str. If we want to initialize str with a string, we use the same assignment operator (=) and assign it a literal string or another string variable. Literal strings are enclosed in double quotes. This is how to assign the lliteral string ciriaco to str: `str = "ciriaco"`.

Indexing requires a pair of square brackets and enclosed in the square bracket is the index of the character/symbol we want to access. This is shown below.

```
str[1] -> refers to the first symbol of the string, also known as the leftmost
character
str[2] -> refers to the 2rd symbol of the string
str[6] -> refers to the 6th symbol of the string
```

To assign a new symbol at given an index, indexing is used together with the assignment operator. So if we want to change the symbol o of ciriaco to the symbol a via str, we say `str[7] = 'a'`. Character literals in flowcharts are enclosed in single quotes. This is to distinguish them from variable names. If we simply did `str[7] = a`, this would mean that there is a variable called a and we want it assigned to $str[7]$. Since a is variable, it could be anything, even a number!

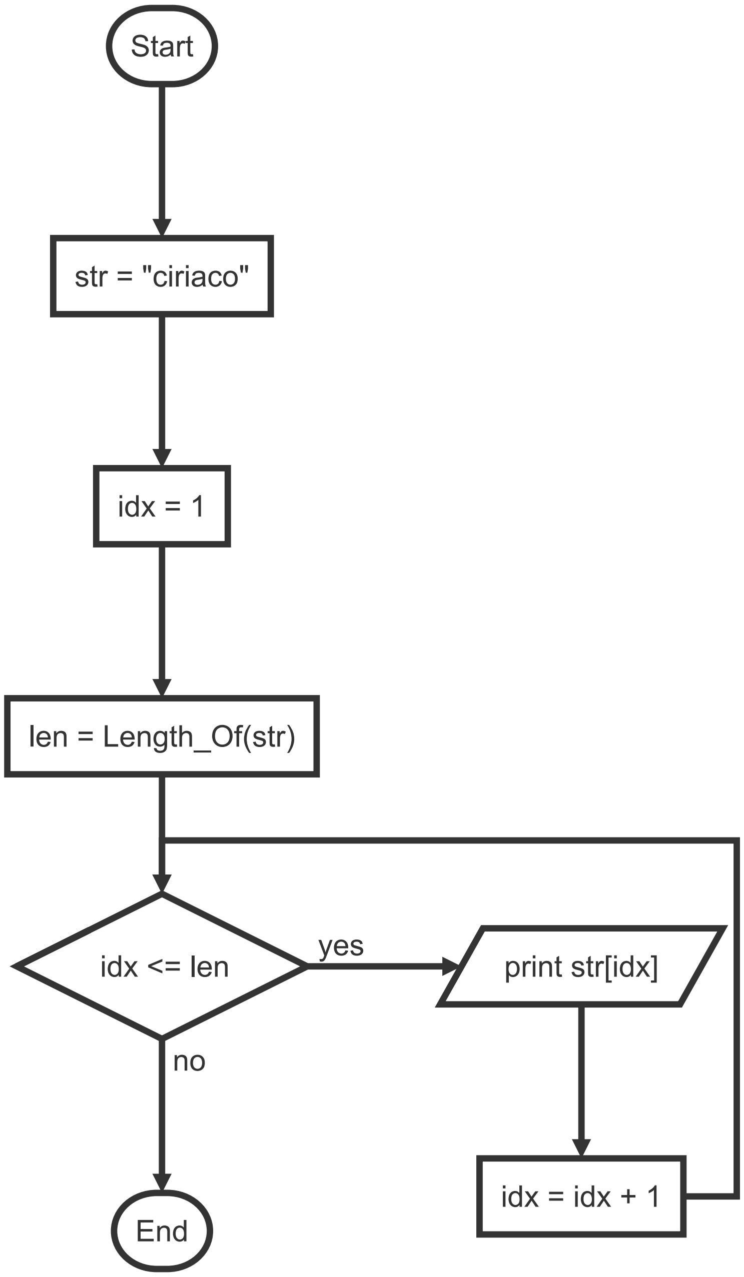
To get the length of the string str, there is an operator we can use called Length_Of. This is how you use it `Length_Of(str)` and it will give you a positive integer that represents the length of the string str. Normally, lengths are assigned to a variable as shown below.

```
len = Length_Of(str)
```

In our case, len will be assigned the value 7 since this is the length of ciriaco.

1.6.1 Printing the characters of the string

Shown below is a simple flowchart that displays the characters of the string one by one.

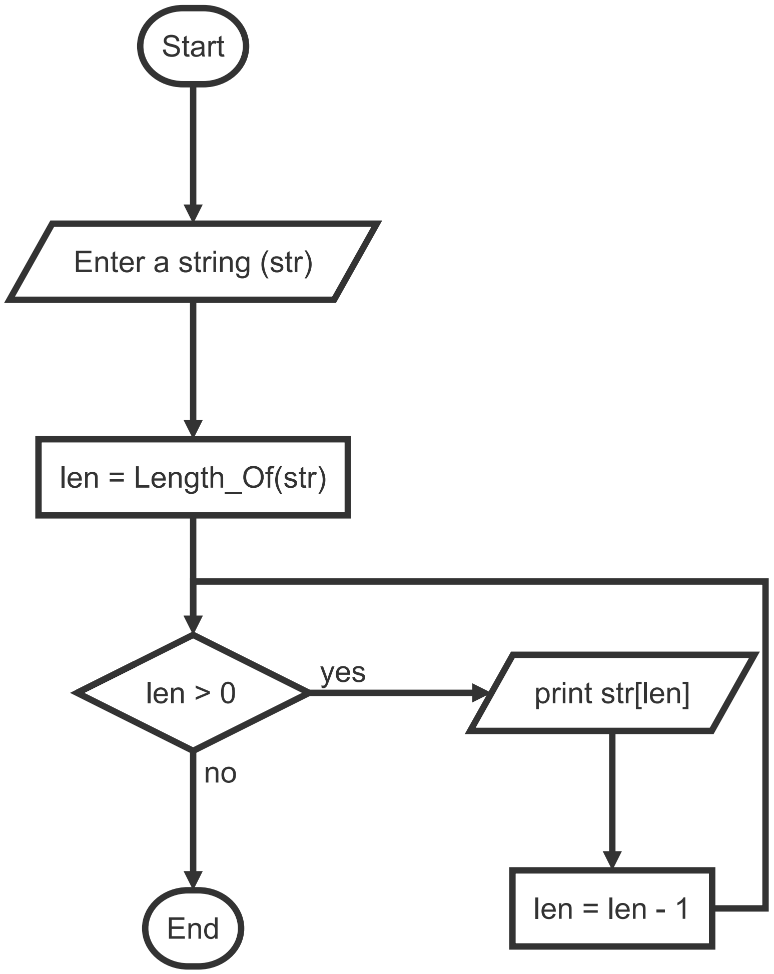


As was discussed, the first symbol or character of the string is found at index 1. The last character at $length$. To be able to scan the string character by character, we need an iterator that begins in 0. In this case, idx. We keep incrementing idx until it reaches len. This will happen when the loop has exhausted all the characters of str.

Trace time!

```
str: ciriaco
idx: 1
1 <= 7 ??   Yes
print c
idx: 1 + 1: 2
2 <= 7 ??   Yes
print i
idx: 2 + 1: 3
3 <= 7 ??   Yes
print r
idx: 3 + 1: 4
4 <= 7 ??   Yes
print i
idx: 4 + 1: 5
5 <= 7 ??   Yes
print a
idx: 5 + 1: 6
6 <= 7 ??   Yes
print c
idx: 6 + 1: 7
7 <= 7 ??   Yes
print o
idx: 7 + 1: 8
8 <= 7 ??   No
end
```

We can also print the string in reverse order by starting from len all the way down to 1.

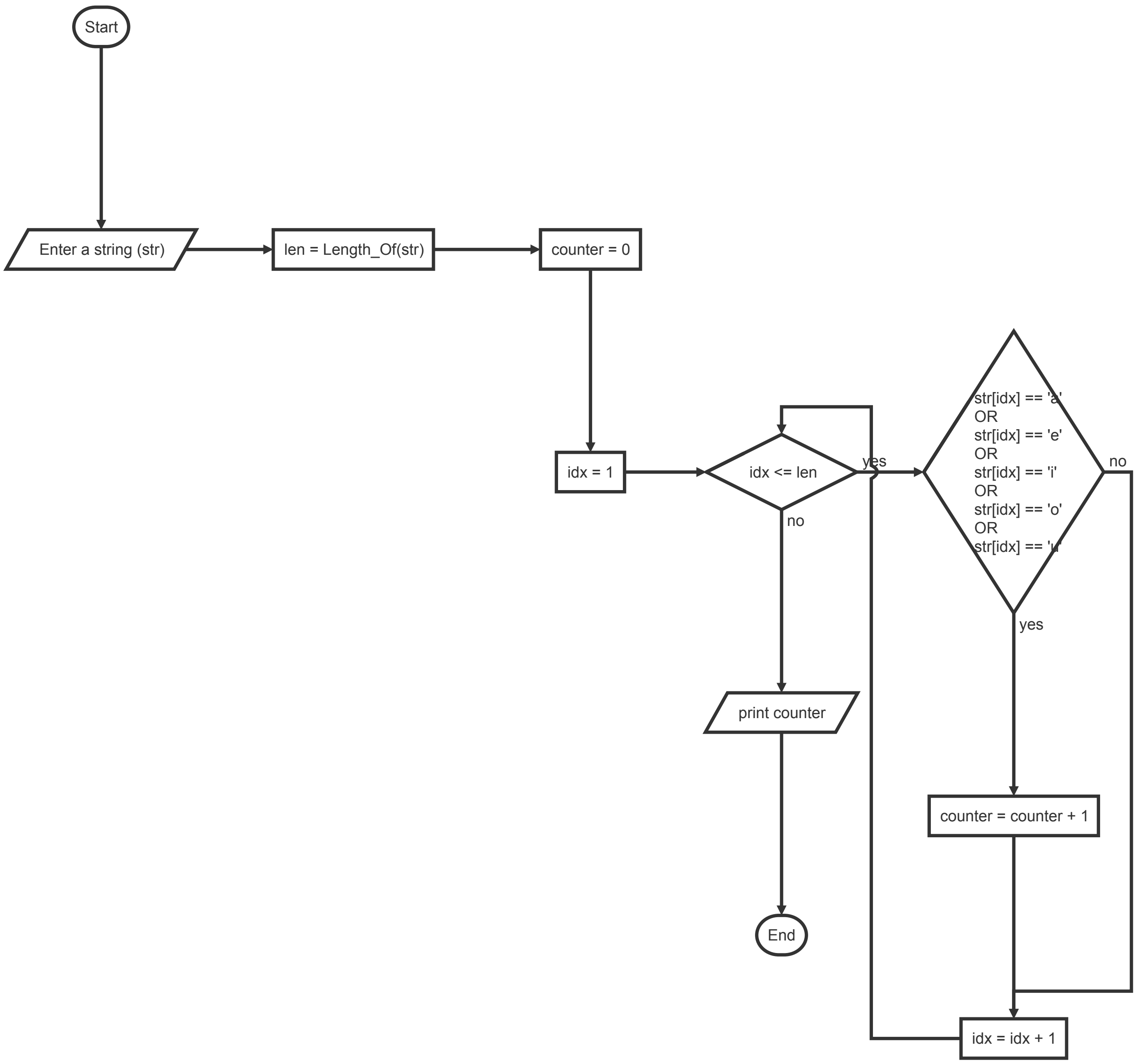


Doing so has eliminated the need for idx. The iteration will stop when len has been reduced to 0. I also took the liberty of asking the string from user instead of assigning it to a literal string.

1.6.2 Count Vowels

Let's say we want to count how many vowels there are in an input string. We would need a counter that would keep track of the number of vowels there are in the string. Let's call this $counter$. Since we are counting, initially we do not have any count yet, so set `counter = 0`. We also need an index that goes from 1 to len since we need to scan the entire string from left to right. The vowels are 'a', 'e', 'i', o', 'u'.

Let's jump right in!


Start
Enter a string (str)
len = Length_Of(str)
counter = 0
idx = 1
idx <= len
yes
no
str[idx] == 'a'
OR
str[idx] == 'e'
OR
str[idx] == 'i'
OR
str[idx] == 'o'
OR
str[idx] == 'u'
no
yes
print counter
End
counter = counter + 1
idx = idx + 1

Trace time! Assume the input string is "mass testing". But we leave this to you as an exercise.

Head on to the video entitled **"Problem Solving: Numerical String Only"**

1.6.3 Palindromes

One of most common string problems is the palindrome. Palindromes are strings when read backwards is the string itself. Below are some palindromes:

```
anna
level
yehey
nodevillivedon
racecar
vvvvv
z
l
```

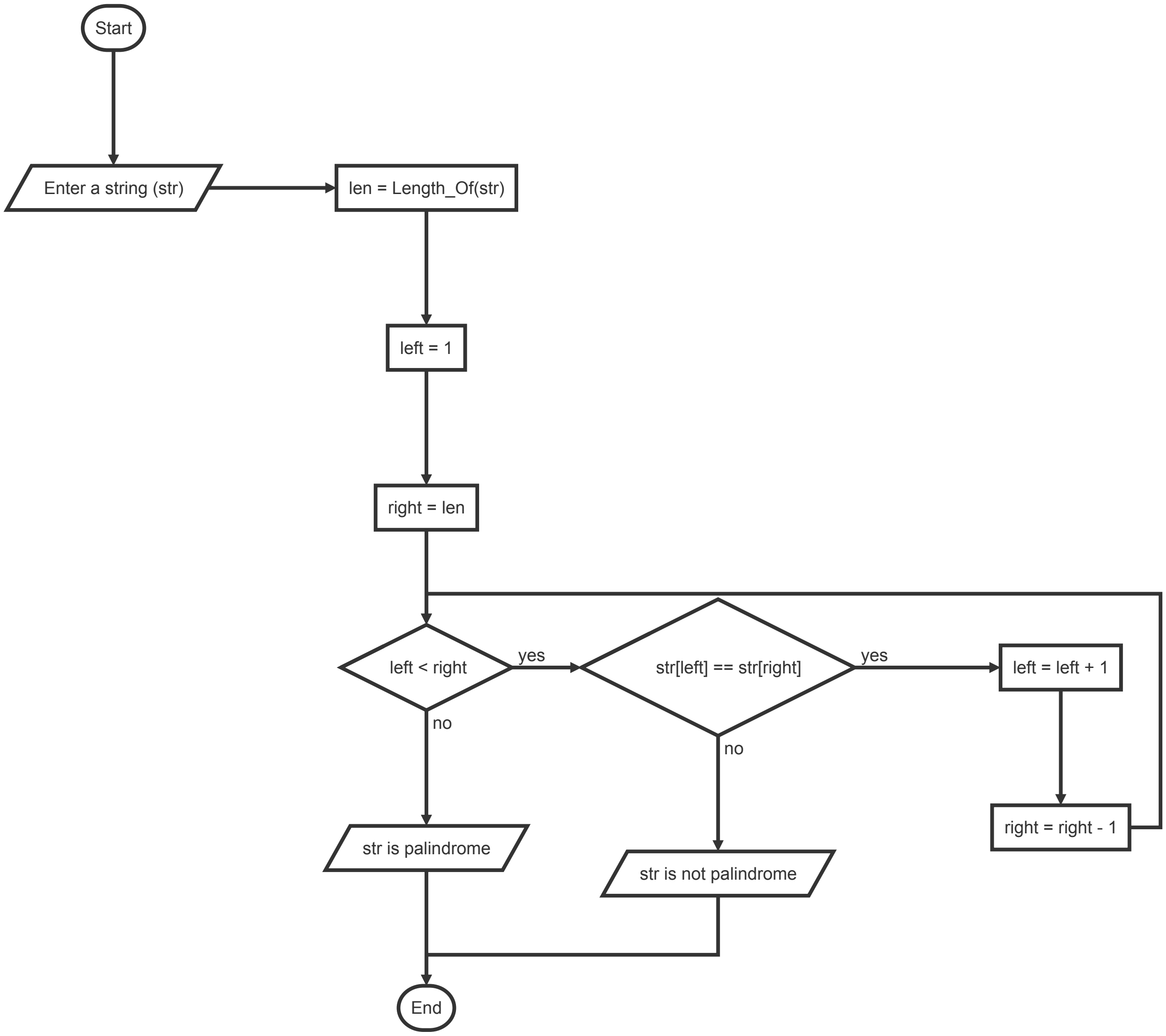
There is only one way for a string to be a palindrome. Take "yehey". The first letter is the same as the fifth letter. Also, the second letter is the same as the 4th letter. And of course, the 3rd letter is the same as itself.

What we did here was to compare a pair of characters. One of the pairs come the left part of the string, the other from the right. It starts from the leftmost and rightmost characters then both pairs move inwards until they meet in the middle (when length of the string is odd) or they cross values.

This means we need an index that starts from the left, and another index that starts from the right.

The left index is 1 i.e. `left = 1` and the right index is $length$ i.e. `right = length`. $left$ increments while $right$ decrements. At the onset, $left$ is always less than $right$. If the process of increment and decrement is applied to $left$ and $right$, respectively, then $left$ and $right$ will eventually be equal (when the length of the string is odd) or $right$ will be less than $left$. They would have "crossed" values.

Let's dive right into it!


Start
Enter a string (str)
len = Length_Of(str)
left = 1
right = len
left < right
yes
no
str[left] == str[right]
yes
no
left = left + 1
right = right - 1
str is palindrome
str is not palindrome
End

Trace time!

```
str: spans
len: 5
left: 1
right: 5
1 < 5   ??    Yes
s == s  ??    Yes
left: 1 + 1:  2
right: 5 - 1: 4
2 < 4   ??    Yes
p == n  ??    No
spans is not a palindrome.
end
```

Inspecting the iteration, just because the leftmost and rightmost characters match it does not mean that it is palindrome right away. We must check the next pair. But the moment they do not match, we can readily say that the string is not prime and therefore the flowchart can safely terminate.

One last trace!

```
str: nodevillivedon
len: 14
left: 1
right: 14
1 < 14  ??      Yes
n == n  ??      Yes
left: 1 + 1:    2
right: 14 - 1:  13
2 < 13  ??      Yes
o == o  ??      Yes
left: 2 + 1:    3
right: 13 - 1:  12
3 < 12  ??      Yes
d == d  ??      Yes
left: 3 + 1:    4
right: 12 - 1:  11
4 < 11  ??      Yes
e == e  ??      Yes
left: 4 + 1:    5
right: 11 - 1:  10
5 < 10  ??      Yes
v == v  ??      Yes
```

```
left: 5 + 1:    6
right: 10 - 1:  9
6 < 9   ??      Yes
i == i  ??      Yes
left: 6 + 1:    7
right: 9 - 1:   8
7 < 8   ??      Yes
l == l  ??      Yes
left: 7 + 1:    8
right: 8 - 1:   7
8 < 7   ??      No
nodevillivedon is a palindrome
```

Notice the last check. `8 < 7`, left and right have crossed values already. Now $left$ is bigger than $right$. This signals us that the comparisons are done. We do not have proceed because we would be checking what has checked already.

Congratulations! We are now ready to use the c programming language as a tool for problem solving!