CS1010E: Programming Methodology

Take Home Lab 5: String

05 Apr 2017

Preliminary: Stringification [#1]

Problem Description

This is a simple exercise of finding `string` length, concatenating `string`, and copying `string` without the use of `string` library.

Final Objective

Given **three (3)** `string` s_1, s_2, s_3, find the length of all the `string`, concatenate s_1 to s_2, and copy s_3 to s_1.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ 3 $\leq$ `length(s1)` $\leq$ 100 (*number of characters in* `string` s_1)
- ▷ 3 $\leq$ `length(s2)` $\leq$ 100 (*number of characters in* `string` s_2)
- ▷ 3 $\leq$ `length(s3)` $\leq$ 100 (*number of characters in* `string` s_3)

Tasks

The problem is split into 1 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/1

Write a program that reads **three (3)** `string` `s1, s2, s3` and print the length of all **three (3)** `string` in one line, the string `s1, s2, s3` after the operations above in the next line.

Sample Run:

Inputs:

```
adiyoga sidi prabawa
```

Outputs:

```
7 11 7↩                          | length after operation
prabawa sidiadiyoga prabawa↩     | concatenation and copy
```

Save your program in the file named `stringify1.c`. Submit your program to CodeCrunch.

Easy: Case Insensitive Search [#2]

Problem Description

Case insensitive search is a type of search where the character in the search term matches both *uppercase* and *lowercase* character in the text.

Final Objective

Given **two (2)** **`string`** *text* and *term* corresponding to the text and the search term, find the first index where the *term* is found in *text*.

Example

Consider the text `"SearchMeIfYouCan"`:

- `"Arch"` is found at index 2
- `"can"` is found at index 13
- `"EAR"` is found at index 1

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ 3 ≤ `length(text)` ≤ 100 (*number of characters in the* `text`)

▷ 3 ≤ `length(term)` ≤ 20 (*number of characters in the search* `term`)

Tasks

The problem is split into 1 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[`**`newline`**`]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/1

Write a program that reads **two (2)** **`string`** `text` and `term` and print the first index of the occurrences of `term` in `text`. If `term` is **NOT** found in `text`, print `-1` instead.

Sample Run:

```
Inputs:                   Outputs:

SearchMeIfYouCan Arch     2↩
```

Sample Run:

```
Inputs:                   Outputs:

SearchMeIfYouCan can      13↩
```

Sample Run:

```
Inputs:                   Outputs:

SearchMeIfYouCan EAR      1↩
```

Sample Run:

```
Inputs:                   Outputs:

SearchMeIfYouCan Adi      -1↩
```

Save your program in the file named `search1.c`. Submit your program to CodeCrunch.

Easy: Palindrome [#3]

Problem Description

"A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as madam or racecar. Sentence-length palindromes may be written when allowances are made for adjustments to capital letters, punctuation, and word dividers, such as "A man, a plan, a canal, Panama!", "Was it a car or a cat I saw?" or "No 'x' in Nixon"." – Wikipedia

In this question, we will be dealing with standard word palindrome. There are several ways to describe a palindrome and they are all equal:

- Word that reads the same backward and forward
- Word that is its own reverse
- Word such that the first and last character is the same and when you remove them, the inner word is also a palindrome (*recursive*)

Final Objective

Given a word, determine if it is a palindrome.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $3 \leq$ `length(word)` ≤ 20 (*number of characters in the word*)

▷ The *word* consists only of lowercase alphabet

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3

Write a program to read a word as a `string` and print the word back.

Sample Run:

Inputs:	Outputs:
`madam`	`madam`↩

Sample Run:

Inputs:	Outputs:
`nixon`	`nixon`↩

Save your program in the file named `palindrome1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp palindrome1.c palindrome2.c
```

Task 2/3
Write a program to read a word as a **`string`** and print the word in reverse.

Sample Run:

Inputs:	Outputs:
`madam`	`madam`↩

Sample Run:

Inputs:	Outputs:
`nixon`	`noxin`↩

Save your program in the file named `palindrome2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp palindrome2.c palindrome3.c
```

Task 3/3
Write a program to read a word as a **`string`** and print `"true"` if the word is a palindrome and `"false"` if the word is not a palindrome.

Sample Run:

Inputs:	Outputs:
`madam`	`true`↩

Sample Run:

Inputs:	Outputs:
`nixon`	`false`↩

Save your program in the file named `palindrome3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp palindrome3.c palindrome4.c
```

Medium: Regular Expression [#4]

Problem Description

"A regular expression, regex or regexp (sometimes called a rational expression) is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings." – Wikipedia

Regular expression search is similar to normal *case-sensitive* search except that certain characters matches multiple characters. In this simplified regular expression question, we will introduce **three (3)** wildcards as follows:

- `*`: matches all *lowercase* characters `a-z`
- `^`: matches all *uppercase* characters `A-Z`
- `#`: matches all *numeric* characters `0-9`

Final Objective

Given **two (2)** **`string`** corresponding to the *text* and *term* where *term* may contain the wildcards, find the first *substring* that matches the regular expression. If no match is found, print "NONE" instead.

Example

Given a **`string`** `AdiYogaCS1010E`, the following regular expressions below match the given substring:

- `^di^og*`: `AdiYoga`
- `CS####E`: `CS1010E`
- `^^####^`: `CS1010E`
- `^^####*`: does not match any part of the **`string`**
- `cs1010E`: does not match any part of the **`string`**

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $3 \leq$ `length(text)` ≤ 100 (*number of characters in the* `text`)
▷ $1 \leq$ `length(term)` ≤ 20 (*number of characters in the* `term`)
▷ `text` and `term` consists of only alphanumeric characters and wildcards

Tasks

The problem is split into 4 task(s). In the sample run, please note the following:

- ↩ is the *invisible* [**`newline`**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/4

Write a program that reads **two (2)** **`string`** `text` and `term`, and print the `text`.

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E ^di^og*`	`AdiYogaCS1010E↩`

Save your program in the file named `regexp1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp regexp1.c regexp2.c
```

Task 2/4

Write a program that reads **two (2)** `string` `text` and `term`, and print the `text` such that **ALL** *lowercase* character is replaced with `*`, *uppercase* with `^`, and *numeric* with `#`.

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E ^di^og*`	`^**^***^^####^`↩

Save your program in the file named `regexp2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp regexp2.c regexp3.c
```

Task 3/4

Write a program that reads **two (2)** `string` `text` and `term`, and print the starting index of first occurrences of `term` in `text`. If `term` is **NOT** found in `text`, print `-1`.

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E ^di^og*`	`0`↩

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E CS####E`	`7`↩

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E cs1010E`	`-1`↩

Save your program in the file named `regexp3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp regexp3.c regexp4.c
```

Task 4/4

Write a program that reads **two (2)** `string` `text` and `term`, and print the first occurrences of `term` in `text`. If `term` is **NOT** found in `text`, print "NONE".

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E ^di^og*`	`AdiYoga`↩

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E CS####E`	`CS1010E`↩

Sample Run:

Inputs:	Outputs:
`AdiYogaCS1010E cs1010E`	`NONE`↩

Save your program in the file named `regexp4.c`. Submit your program to CodeCrunch.

Medium: Caesar Cipher [#5]

Problem Description

"In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence." – Wikipedia

For instance, given $Key = 23$, the cipher is as follows:

```
Plain:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher: XYZABCDEFGHIJKLMNOPQRSTUVW
```

The cipher is created by replacing the alphabet with the alphabet 23 to its right. Using it in an encryption, we will get:

```
Plain:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Cipher: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

Final Objective

Given a Key and a sequence of *word* as **`string`**, encrypt every *word* in a case-sensitive way.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ 0 ≤ `key` ≤ 1000 (*the encryption key*)

▷ 1 ≤ `length(word)` ≤ 20 (*number of characters in the* `word`)

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* [**`newline`**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3

Write a program that reads the *key* and the sequence of *word* and the list of words, print the sequence of words. The input is given as:

- The first line are **one (1)** **`integer`** numbers *key* corresponding to the encryption key
- The next lines consist of a **`string`** corresponding to the word to be encrypted
  - Read until there are no more inputs

Sample Run:

Inputs:

```
23
The Quick Brown Fox
```

Outputs:

```
The Quick Brown Fox ↩
```

Sample Run:

Inputs:

```
23
Jumps Over the Lazy Dog
```

Outputs:

```
Jumps Over the Lazy Dog ↩
```

Save your program in the file named `caesar1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp caesar1.c caesar2.c
```

Task 2/3

Write a program that reads the *key* and the sequence of *word* and the list of words, print the uppercase and lowercase ciphers based on the *key*. The input is given as:

- The first line are **one (1)** `integer` numbers *key* corresponding to the encryption key
- The next lines consist of a `string` corresponding to the word to be encrypted
  - Read until there are no more inputs

Sample Run:

Inputs:

```
23
The Quick Brown Fox
```

Outputs:

```
XYZABCDEFGHIJKLMNOPQRSTUVW↩
xyzabcdefghijklmnopqrstuvw↩
```

Sample Run:

Inputs:

```
23
Jumps Over the Lazy Dog
```

Outputs:

```
XYZABCDEFGHIJKLMNOPQRSTUVW↩
xyzabcdefghijklmnopqrstuvw↩
```

Save your program in the file named `caesar2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp caesar2.c caesar3.c
```

Task 3/3

Write a program that reads the *key* and the sequence of *word* and the list of words, print the encrypted *word* based on Caesar cipher using the encryption *key*. The input is given as:

- The first line are **one (1)** `integer` numbers *key* corresponding to the encryption key
- The next lines consist of a `string` corresponding to the word to be encrypted
  - Read until there are no more inputs

Sample Run:

Inputs:

```
23
The Quick Brown Fox
```

Outputs:

```
Qeb Nrfzh Yoltk Clu ↩
```

Sample Run:

Inputs:

```
23
Jumps Over the Lazy Dog
```

Outputs:

```
Grjmp Lsbo qeb Ixwv Ald ↩
```

Save your program in the file named `caesar3.c`. Submit your program to CodeCrunch.

Hard: Pig Latin [#6]

Problem Description

“Pig Latin is a language game in which words in English are altered. The objective is to conceal the words from others not familiar with the rules. The reference to Latin is a deliberate misnomer, as it is simply a form of jargon, used only for its English connotations as a strange and foreign-sounding language.” – Wikipedia

The rules to transform a word into pig latin is simple. For every words beginning with *consonants*, all letters before the initial vowel are placed at the end of the word sequence. Then add `"way"` to the end of the word if the word starts with a vowel or `"ay"` if the word starts with a consonant. Lastly, ensure that only the first character is capitalized.

Final Objective

Given a sequence of **`string`**, transform each **`string`** into its pig latin equivalent.

Example

Below are some example of pig latin:

- Pig = Igpay
- Latin = Atinlay
- Trash = Ashtray
- Omelet = Omeletway

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $1 \leq \texttt{length(word)} \leq 15$ (*number of characters in the* `word`, *English words aren't that long*)
▷ There are an *arbitrary* number of `word`
▷ The `word` consists only of alphabets
▷ First letter of the `word` is always in uppercase, the rest in lowercase

Tasks

The problem is split into 5 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[`**`newline`**`]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/5

Write a program that reads a sequence of **`string`** `word`, and print the all `word` back in a single line.
Note: there **IS** an additional space at the end.

Sample Run:

Inputs:	Outputs:
`Adi Is Handsomest`	`Adi Is Handsomest` ↩

Save your program in the file named `latin1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp latin1.c latin2.c
```

Task 2/5

Write a program that reads a sequence of **string** word, and print the all word back from the first vowel in a single line. *Note:* there **IS** an additional space at the end.

Sample Run:

Inputs:	Outputs:
`Adi Is Handsomest`	`Adi Is andsomest ↩`

Save your program in the file named `latin2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp latin2.c latin3.c
```

Task 3/5

Write a program that reads a sequence of **string** word, and print the all word back from the first vowel and append the skipped consonants to the end in a single line. *Note:* there **IS** an additional space at the end.

Sample Run:

Inputs:	Outputs:
`Adi Is Handsomest`	`Adi Is andsomestH ↩`

Save your program in the file named `latin3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp latin3.c latin4.c
```

Task 4/5

Write a program that reads a sequence of **string** word, and print the all word back from the first vowel and append the skipped consonants as well as "ay" to the end in a single line. *Note:* there **IS** an additional space at the end.

Sample Run:

Inputs:	Outputs:
`Adi Is Handsomest`	`Adiay Isay andsomestHay ↩`

Save your program in the file named `latin4.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 5*), copy your program using the following command:

```
cp latin4.c latin5.c
```

Task 5/5

Write a program that reads a sequence of **string** word, and print the all word back from the first vowel and append the skipped consonants as well as "ay" to the end, then perform the proper lower- and upper-case conversion, in a single line. *Note:* there **IS** an additional space at the end.

Sample Run:

Inputs:	Outputs:
`Adi Is Handsomest`	`Adiay Isay Andsomesthay ↩`

Save your program in the file named `latin5.c`. Submit your program to CodeCrunch.