Warning

This page is located in archive. Go to the latest version of this course pages. Go the latest version of this page.

HW 05 - Caesarova šifra

Termín odevzdání	~~13~~20.11.2021 23:59 PST[1)]
Povinné zadání	3b kontrola Coding Stylu [/b211/courses/b0b36prp/resources/tessun/start]
Volitelné zadání	2b kontrola Coding Stylu [/b211/courses/b0b36prp/resources/tessun/start]
Bonusové zadání	Není
Počet uploadů	20
Podpůrné soubory	b0b36prp-hw05.zip [/b211/_media/courses/b0b36prp/hw/b0b36prp-hw05.zip]

Povinné zadaní

Naším úkolem je rozluštit zprávu zakódovanou Caesarovou šifrou [https://en.wikipedia.org/wiki/Caesar_cipher], která funguje na principu záměny písmene z abecedy za písmeno posunuté o pevně určený počet míst. V našem případě máme k dispozici nejen zašifrovaný text, ale také nespolehlivě odposlechnuté originální znění zprávy, tj. některé znaky originální zprávy jsou chybně zapsány. Řešení tak můžeme založit na testování všech možností kódování Caesarovou šifrou a porovnání s odposlechnutým textem. Posun s největší shodou dekódovaného textu s odposlechnutou zprávou budeme považovat za správný a takto dekódovaný text je náš požadovaný výstup. Obecně nemusí tento postup vést na unikátní správné řešení, nicméně v testovaných případech vždy existuje unikátní řešení.

Jedním z cílů tohoto domácího úkolu je procvičení dynamické alokace paměti na základě velikosti vstupu. Proto délka vstupního textu není dopředu známa a je nutné v případě vyčerpání počáteční velikosti (např. 10-100 znaků) dynamicky alokovat prostor větší, např. funkcí `realloc` () pro dvojnásobnou délku. V programu používejte pouze množství paměti, které řádově odpovídá zadanému vstupu. Odevzdávací systém kontroluje velikost dynamicky alokované paměti.

Použitá abeceda se skládá pouze z malých a velkých písmen bez diakritiky, tj. znaky v rozsahu [a-zA-Z].

Pro načítání vstupu používejte pouze funkci `getchar()` nebo `scanf("%c",...)` tak, abyste si vyzkoušeli dynamickou alokaci paměti podle aktuálně potřeby odpovídající načítanému vstupu. Každý řádek na vstupu je zakončen znakem **'\n'**. Na testovacím serveru běží OS Debian, takže se jedná o jeden znak LF **0x0a**. (Na Windows se používají dva znaky CR+LF.)

Použití metody dynamického načítání textového řetězce prostřednictvím rozšíření scanf("%ms") nebo scanf("%as") není z tohoto důvodu povoleno.
Pro testování funkčnosti program před jeho odevzdáním lze využít přiložené vstupní a referenční výstupní soubory. Dále je možné testovat také generátorem a referečním řešení viz Testování HW programů před odevzdáním [/b211/courses/b0b36prp/tutorials/testing]. Pro generování volitelného zadání použijte dodatečný přepínač `-optional` .

Vstup

Na standardním vstupu očekávejte dvě posloupnosti znaků (texty) na samostatných řádcích. První text je zakódovaná zpráva a druhý text je nespolehlivě odposlechnutý text.

- Pokud vstupní text neodpovídá abecedě [a-zA-Z] vypíše program na standardní chybový výstup " `Error: Chybny vstup!` " a skončí s návratovou hodnotou `100` .
- V případě povinné části jsou oba vstupní texty stejné délky a pokud mají délku různou, program vypíše na `stderr` " `Error: Chybna delka vstupu!` " a skončí s návratovou hodnotou `101` .
- Pokud nastanou obě chyby `100` a `101` současně, program vypíše pouze chybovou hlášku " `Error: Chybny vstup!` " a skončí s návratovou hodnotou `100` .
- V případě řešení volitelné části, kdy je program spuštěn s parametrem " `-prp-optional` ", mohou být délky vstupních textů různé a program tak chybou nekončí.

Výstup

Na standardní výstup vypište dekódovanou zprávu jako posloupnost znaků zakončenou znakem konce řádku. Program končí s návratovou hodnotou `0` .

Implementance

Program vhodně dekomponujte na jednotlivé funkce, např. funkce pro dekódování textu o definovaný posun a výpočet vzdálenosti dvou textových řetězců:

```
void shift(const char *src, char *dst, int offset);

int compare(const char *str1, const char *str2);
```

Dále může být vhodné napsat funkci pro šifrování jednoho písmene, která bude respektovat zadanou abecedu [a-zA-Z]. Například:

```
char rotate(char original, int offset);
```

Příklad 1 - pub01-m

V prvním příkladu je zašifrován text “ `Helloworld` ”. Posun je zde o 42 písmen. (16 míst je mezi 'h' a 'x'; 26 pak mezi malými a velkými písmeny). Zašifrovaný text je tedy “ `xUbbemehbT` ”.

Odposlechnutý text může být například “ `??lloworld` ”, kde písmena na místě otazníku špatně odposlechneme a dostaneme jiné náhodné písmeno, například “ `XYlloworld` ”.

Standardní vstup	Očekávaný výstup	Očekávaný chybový výstup	Návratová hodnota
`xUbbemehbT` `XYlloworld`	`Helloworld`	žádný	0

Příklad vyhodnocení. Největší počet shodných písmen (8/10) má posun o 10, proto jej vyhodnotíme jako správný výsledek.

```
00: xUbbemehbT ~ XYlloworld > 0 correct letters
01: yVccfnficU ~ XYlloworld > 0 correct letters
02: zWddgogjdV ~ XYlloworld > 0 correct letters
03: AXeehphkeW ~ XYlloworld > 0 correct letters
04: BYffiqilfX ~ XYlloworld > 1 correct letters
05: CZggjrjmgY ~ XYlloworld > 0 correct letters
06: DahhksknhZ ~ XYlloworld > 0 correct letters
07: Ebiiltloia ~ XYlloworld > 0 correct letters
08: Fcjjmumpjb ~ XYlloworld > 0 correct letters
09: Gdkknvnqkc ~ XYlloworld > 0 correct letters
10: Helloworld ~ XYlloworld > 8 correct letters -- result
11: Ifmmpxpsme ~ XYlloworld > 0 correct letters
12: Jgnnqyqtnf ~ XYlloworld > 0 correct letters
...
50: vSZZckcfZR ~ XYlloworld > 0 correct letters
51: wTaadldgaS ~ XYlloworld > 0 correct letters
52: xUbbemehbT ~ XYlloworld > 0 correct letters
```

Příklad 2 - pub02-m

Standardní vstup	Očekávaný výstup	Očekávaný chybový výstup	Návratová hodnota
`mnoXYhnJLJ` `JCudvgtXRi`	`studentPRP`	žádný	0

Příklad 3 - pub03-m

Standardní vstup	Očekávaný výstup	Očekávaný chybový výstup	Návratová hodnota
`fghQRa-CEC` `scbdeMKARZ`	žádný	`Error: Chybny vstup!`	100

Příklad 4 - pub04-m

Standardní vstup	Očekávaný výstup	Očekávaný chybový výstup	Návratová hodnota
`fghQRa` `scbdeMK`	žádný	`Error: Chybna delka vstupu!`	101

Jednou z častých chyb je zjišťování délky textu v každé iteraci. Výsledný program pak má časovou složitost $\mathcal{O}(n^2)$, viz jedno studentské řešení:

```
for(int i = 0; i < strlen(text); i++) { ... }
```

Volitelné zadání (optXX) [-prp-optional]

U volitelné části úlohu rozšiřujeme o možnost úplné ztráty písmene nebo naopak odposlechnutí písmene, které nebylo vůbec vysláno. Vstupní texty tak mohou mít různou délku. V tomto případě věrohodný text určíme na základě Levenštejnovy vzdálenosti [https://en.wikipedia.org/wiki/Levenshtein_distance] vypočtené pomocí Wagner-Fisher algoritmu [https://en.wikipedia.org/wiki/Wagner%E2%80%93Fischer_algorithm]. Program je ve volitelné části spuštěn s argumentem '-prp-optional'. Vstup pro volitolné zadání může obsahovat dva texty různé délky, a proto při spuštění s argumentem " `-prp-optional` " nevypisujeme chybu `101` . Použití Levenštejnovy vzdálenosti by fungovalo i pro povinné zadání, ale bylo by výrazně pomalejší, používalo více paměti a odevzdávací systém by ho neakceptoval. Používejte proto prosím

Levenštejnovu vzdálenost výhradně pro testovací vstupy z volitelného zadání. Používejte dynamickou alokaci paměti.

Příklad 1 - pub01-o

Standardní vstup	Očekávaný výstup	Očekávaný chybový výstup	Návratová hodnota
`xUbbemehbT` `Hellwooorld`	`Helloworld`	žádný	0

Na jiném příkladu si demonstrujme vzdálenostní matici při výpočtu vzdálenosti Wagner-Fisher algoritmem [https://en.wikipedia.org/wiki/Wagner%E2%80%93Fischer_algorithm], kde je matice vyplňována postupně po řádcích s využitím předchozích hodnot (dynamické programování).

	-	H	e	l	l	o	w	o	o	r	l	d
-	0	1	2	3	4	5	6	7	8	9	10	11
H	1	0	1	2	3	4	5	6	7	8	9	10
e	2	1	0	1	2	3	4	5	6	7	8	9
l	3	2	1	0	1	2	3	4	5	6	7	8
l	4	3	2	1	0	1	2	3	4	5	6	7
o	5	4	3	2	1	0	1	2	3	4	5	6
w	6	5	4	3	2	1	0	1	2	3	4	5
o	7	6	5	4	3	2	1	0	1	2	3	4
r	8	7	6	5	4	3	2	1	1	1	2	3
l	9	8	7	6	5	4	3	2	2	2	1	2
d	10	9	8	7	6	5	4	3	3	3	2	1

Levenštejnova vzdálenost je zde rovna jedné, protože stačí pouze jedna operace odebrání písmena.

Odevzdání

Veřejné příklady + Makefile: b0b36prp-hw05.zip [/b211/_media/courses/b0b36prp/hw/b0b36prp-hw05.zip]

	Povinné zadání	Volitelné zadání
Název v BRUTE	HW05	
Odevzdávané soubory	main.c	
Argumenty při spuštění	žádné	`-prp-optional`

Návratová hodnota	`0` ; Program skončil úspěšně
	`100` ; "Error: Chybny vstup!" → `stderr`
	`101` ; "Error: Chybna delka vstupu!" → `stderr`
		`0` ; Program skončil úspěšně
		`100` ; "Error: Chybny vstup!" → `stderr`

Kompilace pomocí	clang -pedantic -Wall -Werror -std=c99	
Očekávaná časová složitost [2]	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Očekávaná paměťová složitost	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Paměťový limit (stack) [b]	50 000	50 000
Paměťový limit (heap) [b] [3]	INPUT * 20 + 20000	INPUT^2 * 10 + 10000
Procvičované oblasti	práce s textem, ASCII tabulka, dynamická alokace paměti podle velikosti vstupu	dynamické programování Levenštajnova vzdálenost

---

1)

PST [https://www.timeanddate.com/time/zones/pst]

2)

V závislosti na počtu písmen n zašifrovaného textu

3)

INPUT vyjadřuje velikost vstupního souboru v bytech.

courses/b0b36prp/hw/hw05.txt · Last modified: 2021/11/25 08:25 by faiglj