Este é o CS50

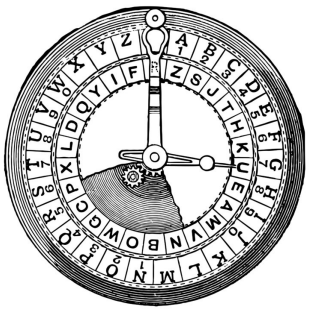
Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

César



Problema a resolver

Supostamente, César (sim, aquele César) costumava "criptografar" (ou seja, ocultar de forma reversível) mensagens confidenciais, deslocando cada letra em algumas posições. Por exemplo, ele poderia escrever A como B, B como C, C como D, …, e, seguindo a ordem alfabética, Z como A. Assim, para dizer OLÁ a alguém, César poderia escrever IFMMP. Ao receber tais mensagens de César, os destinatários teriam que "decifrá-las", deslocando as letras na direção oposta pelo mesmo número de posições.

O sigilo desse "criptossistema" dependia do fato de apenas César e os destinatários conhecerem um segredo: o número de posições que César havia deslocado em suas cartas (por exemplo, 1). Não era particularmente seguro para os padrões modernos, mas, ei, se você fosse talvez o primeiro no mundo a fazer isso, era bem seguro!

Texto não criptografado é geralmente chamado de *texto plano* . Texto criptografado é geralmente chamado *de texto cifrado* . E o segredo usado é chamado de *chave* .

Para ficar claro, então, aqui está como criptografar `HELLO` com uma chave de1rendimentos `IFMMP` :

texto simples	`H`	`E`	`L`	`L`	`O`
+ tecla	1	1	1	1	1
= texto cifrado	`I`	`F`	`M`	`M`	`P`

De forma mais formal, o algoritmo de César (ou seja, a cifra) criptografa mensagens "rotacionando" cada letra por$k$posições. Mais formalmente, se$p$é um texto simples (ou seja, uma mensagem não criptografada),p_ié oi^{th}personagem em$p$, eké uma chave secreta (ou seja, um número inteiro não negativo), então cada letra,c_i, no texto cifrado,c, é calculado como

$$c_i = (p_i + k) \ \% \ 26$$

em que$\% \ 26$Aqui, significa “resto da divisão por 26”. Essa fórmula talvez faça a cifra parecer mais complicada do que realmente é, mas na verdade é apenas uma maneira concisa de expressar o algoritmo com precisão. De fato, para fins de discussão, pense em A (ou a) como0, B (ou b) como 1, …, H (ou h) como7, eu (ou i) como8, …, e Z (ou z) como25Suponha que César queira dizer `Hi` a alguém confidencialmente, usando, desta vez, uma chave:k, de 3. E assim seu texto claro,p, é `Hi` , caso em que o primeiro caractere do seu texto original,p_0, é `H` (também conhecido como 7), e o segundo caractere de seu texto original,p_1, é `i` (também conhecido como 8). O primeiro caractere do seu texto cifrado,c_0, é, portanto `K` , , e o segundo caractere de seu texto cifrado,c_1, é assim `L` . Faz sentido?

Em um arquivo chamado `<nome_do_arquivo>` `caesar.c` em uma pasta chamada ` `caesar` <nome_da_pasta>`, escreva um programa que permita criptografar mensagens usando a cifra de César. No momento da execução do programa, o usuário deverá escolher, por meio de um argumento de linha de comando, qual será a chave da mensagem secreta que será fornecida em tempo de execução. Não devemos necessariamente assumir que a chave do usuário será um número; embora você possa assumir que, se for um número, será um inteiro positivo.

Demonstração

```
Usage: ./caesar key
$ ./caesar HELLO
Usage: ./caesar key
$ ./caesar 1 2 3
Usage: ./caesar key
$ ./caesar 13
plaintext:  Hi there!
ciphertext: Uv gurer!
$ ./caesar 26
plaintext:  This is CS50.
ciphertext: This is CS50.
$
```

Recorded with **asciinema**

Especificação

Projete e implemente um programa, `caesar`, que criptografe mensagens usando a cifra de César.

- Implemente seu programa em um arquivo chamado `caesar.c` em um diretório chamado `caesar`.
- Seu programa deve aceitar um único argumento de linha de comando, um número inteiro não negativo. Vamos chamá-lo de kPara fins de discussão.
- Se o seu programa for executado sem nenhum argumento de linha de comando ou com mais de um argumento de linha de comando, ele deverá imprimir uma mensagem de erro de sua escolha (com `printf`) e retornar imediatamente `main` um valor de `1` (que geralmente indica um erro).
- Se algum dos caracteres do argumento da linha de comando não for um dígito decimal, seu programa deverá imprimir a mensagem `Usage: ./caesar key` e retornar `main` um valor de `1`.
- Não presuma que $k$será menor ou igual a 26. Seu programa deve funcionar para todos os valores inteiros não negativos de $k$menor que $2^{31} - 26$Em outras palavras, você não precisa se preocupar se o seu programa eventualmente falhar caso o usuário escolha um valor para kIsso é grande demais ou quase grande demais para caber em um recipiente `int`. (Lembre-se de que um recipiente `int` pode transbordar.) Mas, mesmo que ké maior que 26Os caracteres alfabéticos na entrada do seu programa devem permanecer alfabéticos na saída do seu programa. Por exemplo, se ké 27, `A` não deveria se tornar, `\` mesmo que

`\` seja 27 posições distantes de `A` em ASCII, de acordo com asciitable.com (https://www.asciitable.com/) ; `A` deveria se tornar `B`, já que `B` é 27 posições longe de `A`, desde que você dê a volta de `Z` para `A`.

- Seu programa deve exibir `plaintext:` (com dois espaços, mas sem nova linha) e, em seguida, solicitar ao usuário um `string` texto simples (usando `get_string`).
- Seu programa deve exibir `ciphertext:` (com um espaço, mas sem quebra de linha) seguido do texto cifrado correspondente, com cada caractere alfabético no texto original "rotacionado" em *k* posições; caracteres não alfabéticos devem ser exibidos sem alterações.
- Seu programa deve preservar as maiúsculas e minúsculas: letras maiúsculas, mesmo que rotacionadas, devem permanecer maiúsculas; letras minúsculas, mesmo que rotacionadas, devem permanecer minúsculas.
- Após imprimir o texto cifrado, você deve imprimir uma nova linha. Seu programa deve então terminar retornando `0` da função `main`.

Conselho

Por onde começar? Vamos abordar esse problema passo a passo.

Pseudocódigo

Primeiro escreva, tente escrever uma `main` função `caesar.c` que implemente o programa usando apenas pseudocódigo, mesmo que você (ainda!) não tenha certeza de como escrevê-la em código real.

▼ **Dica**

Existem várias maneiras de fazer isso, então aqui está apenas uma!

```
int main(int argc, string argv[])
{
    // Make sure program was run with just one command-line argument

    // Make sure every character in argv[1] is a digit

    // Convert argv[1] from a `string` to an `int`

    // Prompt user for plaintext

    // For each character in the plaintext:

        // Rotate the character if it's a letter
}
```

Você pode editar seu próprio pseudocódigo depois de ver o nosso aqui, mas não copie e cole o nosso no seu!

Contagem de argumentos da linha de comando

Independentemente do seu pseudocódigo, vamos primeiro escrever apenas o código C que verifica se o programa foi executado com um único argumento de linha de comando antes de adicionar funcionalidades adicionais.

Especificamente, modifique `main` o código `caesar.c` de forma que, se o usuário não fornecer nenhum argumento de linha de comando, ou dois ou mais, a função imprima nada `"Usage: ./caesar key\n"` e retorne ` `1` false`, encerrando o programa. Se o usuário fornecer exatamente um argumento de linha de comando, o programa não deverá imprimir nada e simplesmente retornar `false` `0`. O programa deverá, portanto, comportar-se conforme descrito abaixo.

```
$ ./caesar
Usage: ./caesar key
```

```
$ ./caesar 1 2 3
Usage: ./caesar key
```

```
$ ./caesar 1
```

▼ **Dicas**

- Lembre-se que você pode imprimir com `printf`.
- Lembre-se de que uma função pode retornar um valor com `return`.
- Lembre-se que isso `argc` contém o número de argumentos da linha de comando passados para um programa, mais o nome do próprio programa.

Verificando a chave

Agora que seu programa está (esperamos!) aceitando entradas conforme o esperado, é hora de dar o próximo passo.

Adicione `caesar.c` abaixo `main` uma função chamada, por exemplo, `only_digits` que recebe `string` um argumento e retorna verdadeiro `true` se ele `string` contiver apenas dígitos, `0` de 0 a 1 `9`, caso contrário, retorna falso `false`. Certifique-se de adicionar `main` também o protótipo da função acima.

▼ **Dicas**

- Provavelmente você vai querer um protótipo como este:

  ```
  bool only_digits(string s);
  ```

  E certifique-se de incluir `cs50.h` no topo do seu arquivo, para que o compilador reconheça `string` (e `bool`).

- Lembre-se que `a` `string` é simplesmente uma matriz de `char` elementos.
- Lembre-se que `strlen`, declarado em `string.h`, calcula o comprimento de um `string`.
- Você pode achar útil o parâmetro `is` `isdigit`, declarado em `in`, conforme manual.cs50.io (https://manual.cs50.io/). Mas observe que ele verifica apenas um parâmetro por vez! `ctype.h` (https://manual.cs50.io/) `char`

Em seguida, modifique o código `main` de forma que ele chame `only_digits` a função `argv[1]`. Se essa função retornar ` true` `false`, então `. Caso contrário` deve simplesmente retornar `false`. O programa deve, portanto, se comportar conforme o exemplo abaixo: `main` `"Usage: ./caesar key\n"` `1` `main` `0`

```
$ ./caesar 42
```

```
$ ./caesar banana
Usage: ./caesar key
```

Usando a chave

Agora modifique `main` de forma que ele se converta `argv[1]` em um `int`. Você pode achar útil o `atoi`, declarado em , conforme manual.cs50.io (https://manual.cs50.io/). E então use para solicitar ao usuário algum texto simples com . `stdlib.h` (https://manual.cs50.io/) `get_string` `"plaintext: "`

Em seguida, implemente uma função chamada, por exemplo, `rotate` `rotate`, que recebe um `x` `char` como entrada e também um `y` `int`, e rotaciona esse `y` `char` por esse número de posições se for uma letra (ou seja, alfabética), girando de `x` para ` `Z` y` `A` (e de `y` `z` para `z` `a`) conforme necessário. Se o ` `char` y` não for uma letra, a função deve retornar o mesmo ` `char` y` sem alterações.

▼ **Dicas**

- Provavelmente você vai querer um protótipo como este:

  ```
  char rotate(char c, int n);
  ```

  Uma chamada de função como

  ```
  rotate('A', 1)
  ```

  ou até mesmo

  ```
  rotate('A', 27)
  ```

  deve, portanto, retornar `'B'`. E uma chamada de função como

  ```
  rotate('!', 13)
  ```

deve retornar `'!'`.

- Lembre-se de que você pode "converter" explicitamente um `a` `char` para um `b` `int` com `a` `(int)`, e um `b` `int` para um `a` `char` com `b` `(char)`. Ou você pode fazer isso implicitamente, simplesmente tratando um como o outro.
- Provavelmente você vai querer subtrair o valor ASCII de `'A'` qualquer letra maiúscula, para tratar `'A'` como `0`, `'B'` como `1`, e assim por diante, ao realizar operações aritméticas. E então, adicione o valor de volta ao terminar.
- Provavelmente você vai querer subtrair o valor ASCII de `'a'` qualquer letra minúscula, para tratar `'a'` como `0`, `'b'` como `1`, e assim por diante, ao realizar operações aritméticas. E então, adicione o valor de volta ao final da operação.
- `ctype.h` Você pode achar úteis algumas outras funções declaradas em manual.cs50.io (https://manual.cs50.io/).
- É provável que você ache `%` isso útil ao "dar a volta" aritmeticamente de um valor como `25` para `0`.

Em seguida, modifique o código `main` de forma que ele imprima `"ciphertext: "` e itere sobre cada elemento `char` no texto simples do usuário, chamando `rotate` cada um deles e imprimindo o valor de retorno correspondente.

▼ **Dicas**

- Lembre-se que `printf` é possível imprimir um `char` usando `%c`.
- Se você não estiver vendo nenhuma saída ao chamar a função `printf`, provavelmente é porque está imprimindo caracteres fora do intervalo ASCII válido, de 0 a 127. Tente imprimir os caracteres temporariamente como números (usando `number` `%i` em vez de `number` `%c`) para ver quais valores você está imprimindo!

Passo a passo

Como testar

Correção

No seu terminal, execute o comando abaixo para verificar se o seu trabalho está correto.

```
check50 cs50/problems/2025/x/caesar
```

▼ **Como usar** `debug50`

Deseja executar o comando `debug50`? Você pode fazê-lo da seguinte forma, após compilar seu código com sucesso usando o comando `make`:

```
debug50 ./caesar KEY
```

onde `<key> KEY` é a chave que você fornece como argumento de linha de comando para o seu programa. Observe que a execução

```
debug50 ./caesar
```

(Idealmente!) fará com que seu programa termine solicitando uma chave ao usuário.

Estilo

Execute o comando abaixo para avaliar o estilo do seu código usando `style50`.

```
style50 caesar.c
```

Como enviar

No seu terminal, execute o comando abaixo para submeter seu trabalho.

```
submit50 cs50/problems/2025/x/caesar
```