PSET: César (*Caesar*)

Harvard CS50 Staff

2023-10-01

Resumo

Esta atividade corresponde ao PSET *Caesar* (César) original da disciplina Harvard CS50, em sua versão integral, traduzido e adaptado para o português pelo prof. Abrantes Araújo Silva Filho.

Sumário

1 Introdução

Esta atividade corresponde ao PSET "*Caesar*" (César) da disciplina **Harvard CS50**, e deve ser feito pelos alunos que estão estuando o conteúdo sobre *arrays*.

O objetivo deste exercício é que você pratique manipulação de strings e arrays, fazendo acesso direto à diversas posições do array através de seu índice, e pratique a passagem de argumentos via linha de comando.

A tradução e adaptação para o português foram feita com base na versão de 2023 do PSET, conforme o PSET original[1].

[1] https://cs50.harvard.edu/x/2023/psets/2/caesar/

2 César (*Caesar*)

Nesta atividade você implementará um programa que criptografa mensagens utilizando a ténica conhecida por Cifra de César[2], como no exemplo abaixo:

```
1   $ ./cesar 13
2   Texto puro: OLA
3   Texto cifrado: BYN
4   $
```

Supostamente, César (sim, aquele César) costumava "criptografar" (ou seja, ocultar de maneira reversível) mensagens confidenciais deslocando cada letra do alfabeto por algum número de posições. Por exemplo, se o deslocamento for de 1 posição, "A" será escrito como "B", "B" como "C", "C" como "D" e assim por diante, até que "Z" seria escrito como "A" (ao terminar a seqüência alfabética, se ainda houverem posições a serem descoladas, inicia-se novamente da letra "A"). Com deslocamento de 1 posição, para escrever "OLA" César teria que escrever "PMB". Quando o destinatário receber as mensagens terá que descriptografá-las fazendo o mesmo deslocamento na direção oposta.

A segurança desse sistema de criptografia baseava-se no fato de que apenas César e os destinatários da mensagem sabiam e compartilhavam um segredo, o número do deslocamento, ou seja, a quantidade de posições que César deslocava as letras. Obviamente isso não é nada seguro pelos padrões atuais mas, ei!, se você foi a primeira pessoa do mundo a usar esse sistema, ele era totalmente seguro!

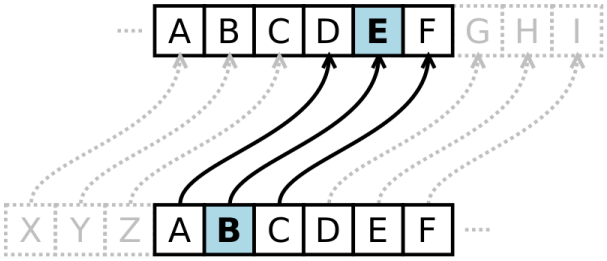
Textos não criptografados são geralmente chamados de **texto puro**, e textos criptografados são geralmente chamados de **texto cifrado**. O segredo, que no caso da Cifra de César é o deslocamento, é chamado de **chave**.

Para isso ficar bem claro, veja como criptografar a palavra "COMPUTADOR" com uma chave de 1 resulta na palavra "DPNQVUBEPS":

Texto puro	C	O	M	P	U	T	A	D	O	R
+ **Chave**	1	1	1	1	1	1	1	1	1	1
= **Texto cifrado**	D	P	N	Q	V	U	B	E	P	S

A rotação pode ser entendida na figura abaixo, onde o texto puro (na parte inferior) é transformado com uma chave de 3 no texto cifrado (na parte de cima):

Figura 1: Cifra de César com deslocamento de 3 posições



Adaptado de: Matt Crypto, na Wikipedia[3]

De modo mais formal, o algoritmo de César criptografa as mensagens fazendo uma rotação (deslocamento) de cada letra por k posições. Mais formalmente ainda, se p é algum texto puro (uma mensagem não criptografada), p_i é o i-ésimo caractere em p, e k é a chave (um inteiro não negativo), então cada letra c_i no texto cifrado c é calculada como

$$c_i = (p_i + k) \% 26 \quad (1)$$

[2] https://en.wikipedia.org/wiki/Caesar_cipher
[3] https://en.wikipedia.org/wiki/File:Caesar_cipher_left_shift_of_3.svg

onde "$\%\ 26$" indica o resto da divisão por 26 (pois existem 26 letras no alfabeto latino). Essa fórmula parece fazer a cifra de César ser mais complicada do que realmente é, mas a fórmula é apenas uma maneira concisa de expressar nosso algoritmo de forma precisa.

Como exemplo, pense na letra "A" (ou "a") como 0, "B" (ou "b") como 1, ..., "H" (ou "h") como 7, "I" (ou "i") como 8, ..., e "Z" (ou "z") como 25, conforme a tabela a seguir.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Suponha agora que César quer escrever "Oi" para alguém de forma confidencial usando uma chave, k, de 3 ($k = 3$). O texto puro, p, seria "Oi", e as letras p_i nesse texto seriam: $p_0 =$ O (correspondente a 14) e $p_1 =$ i (correspondente a 8). O texto crifado, c, seria "Rl", e os caracteres na mensagem cifrada c seriam: $c_0 =$ R (correspondente a 17) e $c_1 =$ l (correspondente a 11).

Escreva um programa chamado `cesar` que permite criptografar mensagens usando a Cifra de César. No momento em que o usuário executa o programa, ele deve decidir, fornecendo um **argumento de linha de comando**, qual deve ser a chave na mensagem secreta (a chave k). Não devemos necessariamente assumir que a chave que o usuário digitar será um número inteiro, mas você pode assumir que, se for um número, será um número interio positivo.

Aqui estão alguns exemplos de como o programa pode funcionar. Por exemplo, se o usuário inserir uma chave de 1 e o texto puro "HELLO":

```
$ ./cesar 1
Texto puro: HELLO
Texto cifrado: IFMMP
$
```

Veja como o programa pode funcionar se o usuário fornecer uma chave de 13 e um o texto simples "hello, world":

```
$ ./cesar 13
Texto puro: hello, world
Texto cifrado: uryyb, jbeyq
$
```

Observe que nem a vírgula nem o espaço foram deslocados pela cifra. Faça o deslocamento apenas em caracteres alfabéticos (letras)!

Que tal mais um exemplo? Veja como o programa pode funcionar se o usuário fornecer uma chave de 13, com um texto puro mais complexo:

```
$ ./cesar 13
Texto puro: be sure to drink your Ovaltine
Texto cifrado: or fher gb qevax lbhe Binygvar
$
```

Por que Ovaltine?

Christmas Story, no YouTube[4]

Note que as letras maiúsculas permaneceram maiúsculas, assim como as minúsculas permaneceram minúsculas.

E se um usuário não cooperar, fornecendo um argumento de linha de comando que não é um número? O programa deve lembrar o usuário como usar o programa:

```
$ ./cesar HELLO
Uso: ./cesar chave
$
```

Se o usuário não fornecer nenhum argumento de linha de comando, o programa deve lembrar o usuário como usar o programa:

```
$ ./cesar
Uso: ./cesar chave
$
```

Se o usuário fornecer mais de um argumento de linha de comando, o programa deve lembrar o usuário como usar o programa:

```
$ ./cesar 1 2 3
Uso: ./cesar chave
$
```

Aqui está o comportamente esperado quando o usuário comete alguns erros e depois utiliza o programa corretamente:

```
$ ./cesar
Uso: ./cesar chave
$ ./cesar HELLO
Uso: ./cesar chave
$ ./cesar 1 2 3
Uso: ./cesar chave
$ ./cesar 13
Texto puro: Hi there!
Texto cifrado: Uv gurer!
$ ./cesar 26
Texto puro: This is CS50!
Texto cifrado: This is CS50!
```

3 Especificação

Crie e implemente um programa chamado "`cesar`", que faz a criptografia de mensagens utilizando a Cifra de César, de acordo com as seguintes instruções:

[4] https://www.youtube.com/watch?v=9K4FsAHB-C8

- Utilize o *starter file* com o nome "`cesar.c`", preencha as informações de identificação e honestidade acadêmica e escreva seu programa;
- Seu programa deve aceitar um único argumento de linha de comando, um número inteiro não negativo. Vamos chamar esse número de k;
- Se o seu programa for executado sem nenhum argumento de linha de comando, ou for executado com mais de um argumento de linha de comando, ele deve imprimir uma mensagem de erro de sua escolha (com o `printf`) e retornar imediatamente o valor de 1 a partir da função `main` (o que tende a significar um erro);
- Se algum dos caracteres do argumento de linha de comando não for um dígito decimal, seu programa deve imprimir a mensagem:

  ```
  Uso: ./cesar chave
  ```

  e retornar imediatamente o valor de 1 a partir da função `main`;
- Não assuma que $k <= 26$. Seu programa deve funcionar para todos os valores de $k < 2^{31} - 26$. Em outras palavras, você não precisa se preocupar se o seu programa travará se o usuário escolher um valor de k que é que é muito grande ou quase muito grande para caber em um `int` (lembre-se de que um `int` pode sofrer *overflow*). Mas mesmo se k for maior do que 26, os caracteres alfabéticos no input do seu programa devem continuar caracteres alfabéticos no output, por exemplo: se $k = 27$, `A` não deve se tornar `\`, mesmo que `\` esteja 27 posições na frente do `A` na tabela ASCII; nesse caso o `A` deve se tornar `B`, já que `B` está 27 posições na frente de `A` (desde que você reinicie a contagem em `A` após o `Z`);
- Seu programa deve imprimir `Texto puro: ` (com um espaço após os dois pontos, mas não uma quebra de linha) e aguardar a string fornecida pelo usuário (não se esqueça que para obter uma string do usuário você deve usar a função `get_string`);
- Seu programa deve imprimir `Texto cifrado: ` (com um espaço após os dois pontos), seguido do correspondente texto cifraqdo, com cada um dos caracteres alfabéticos rotacionados por k posições; caracteres não alfabéticos devem permanecer inalterados;
- Seu programa deve manter as letras maiúsculas e minúsculas: letras maiúsculas devem permanecer maiúsculas após a rotação; letras minúsculas devem permanecer minúsculas após a rotação;
- Após mostrar o texto cifrado seu programa deve imprimir uma nova linha.
- Seu programa deve terminar e retornar o status `0` pela função `main`.

4 Dicas

Como começar? Vamos abordar esse problema computacional passo a passo.

4.1 Pseudocódigo

Uma das primeiras coisa que você deve fazer é escrever a função `main` no arquivo `cesar.c` utilizando pseudocódigo, ou seja, você deve utilizar suas habilidades de pensamento computacional para pensar em como resolver o problema, antes de escrever qualquer código real.

Aqui está uma maneira que ilustra essa abordagem (obviamente existem várias maneiras de escrever o pseudocódigo, aqui está apenas um dos exemplos possíveis):

```
int main(void)
{
    // Verificar que o programa foi executado com um, e apenas um,
    // argumento de linha de comando

    // Garantir que todos os caracteers em argv[1] sejam dígitos

    // Converter argv[1] de string para int

    // Pedir para o usuário informar o texto puro

    // Para cada caracteres no texto puro, rotacionar o caractere se
    // for uma letra
}
```

Atenção: você pode editar seu próprio pseudocódigo a partir desse modelo, mas não copie e cole o nosso pseudocógido como se fosse o seu!

4.2 Contagem dos argumentos de linha de comando

Qualquer que seja seu pseudocódigo, vamos escrever primeiro o código C que verifica se o programa foi executado com um único argumento de linha de comando, antes de fazer qualquer outra coisa.

Especificamente, modifique a função `main` no arquivo `cesar.c` de modo que, se o usuário não fornecer nenhum argumento de linha de comando ou se ele fornecer dois ou mais argumentos, a função imprima

Uso: ./cesar chave

e então retorne 1, saindo do programa. Se o usuário fornecer um, e apenas um, argumento de linha de comando, o programa não deve imprimir nada e simplesmente retornar 0. Nesse ponto seu programa deve se comportar como o seguinte:

```
$ ./cesar
Uso: ./cesar chave
```

```
$ ./cesar 1 2 3
Uso: ./cesar chave
```

```
$ ./cesar 1
```

Dicas:

- Lembre-se de que você deve usar a função `printf`;
- Lembre-se de que uma função retorna alguma valor com o comando `return`; e
- Lembre-se de que `argc` contém o número de argumentos de linha de comando passados a um programa, acrescido do próprio nome do programa.

4.3 Verificando a chave

Agora que seu programa está aceitando e tratando de modo correto os argumentos de linha de comando, é hora de mais um passo: a verificação da chave.

Adicione ao arquivo `cesar.c` uma função chamada, por exemplo, `apenas_digitos`, que recebe uma `string` como argumento e retorna `true` se essa `string` contém apenas

dígitos na faixa de (0–9), ou retorna `false` caso contrário. Não se esqueça do protótipo da função.

Dica: após criar a função acima, modifique a função função `main` de modo que ela chame a função `apenas_digitos` usando `argv[1]` como argumento. Se essa função retornar `false`, então seu programa deve imprimir `Uso: ./cesar chave` e retornar 1. Se tudo correr bem seu programa deve retornar 0. Nesse ponto seu programa deve se comportar como o seguinte:

```
$ ./cesar 42
```

```
$ ./cesar banana
Uso: ./cesar chave
```

4.4 Usando a chave

Agora modifique a função `main` de modo que ela converta `argv[1]` para um `int`. Talvez você ache a função `atoi`, declarada em `stdlib.h`, útil para isso (consulte a documentação em manual.cs50.io[5]. Depois utilize a função `get_string` para solicitar que o usuário informe o texto puro a ser cifrado. Lembre-se que o *prompt* a ser exibido deve ser: "`Texto puro: `".

Depois implemente uma função chamada, por exemplo, de `rotacionar`, que recebe um `char` e um `int`, e rotaciona o `char` por `int` posições se for uma letra (um caractere alfabético), recomeçando a contagem em `A` após a letra `Z` (ou recomeçando em `a` após a letra `z`), se necessário. Se `char` não for alfabético, não deve ser alterado.

Algumas dicas:

- Talvez você precise de um protótipo como:

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

- A chamada de uma função como

  ```
  rotacionar('A', 1);
  ```

  ou até mesmo

  ```
  rotacionar('A', 27);
  ```

  deve retornar `B`. E uma chamada de função como

  ```
  rotacionar('!', 13);
  ```

  deve retornar `!`.

- Lembre-se de que você pode fazer um *cast* para explicitamente converter um `char` em um `int` com `(int)`, e um `int` em um `char` com `(char)`. Você também pode fazer isso implicitamente simplesmente tratando um como o outro;

- Talvez você queira subtrair o valor ASCII do `A` de qualquer caractere maiúsculo, considerando assim o `A` como 0, o `B` como 1 e assim por diante, no momento de fazer os cálculos necessários. No final, quando tudo estiver pronto, você pode somar o valor de volta;

[5] https://manual.cs50.io/

- Talvez você queira subtrair o valor ASCII do `a` de qualquer caractere minúsculo, considerando assim o `a` como 0, o `b` como 1 e assim por diante, no momento de fazer os cálculos necessários. No final, quando tudo estiver pronto, você pode somar o valor de volta; e
- Você pode encontrar algumas funções úteis na biblioteca `ctype.h`. Consulte a documentação em manual.cs50.io.

Modifique a função `main` de forma que ela imprima `Texto cifrado:` e faça uma iteração sobre cada `char` no texto puro, chamando a função `rotacionar` em cada um, e imprimindo o valor de retorno.

Mais algumas dicas:

- Lembre-se de que `printf` imprime um `char` usando `%c`;
- Se você não estiver vendo nenhum output quando usar a função `printf`, talvez você esteja imprimindo caracteres cujo valor ASCII estejam fora da faixa de 0–127. Nesse caso, tente imprimir temporariamente os caracteres como números inteiros (usando `%i`) para ver o que você está tentando imprimir.

5 Passo a passo

Se precisar de mais dicas, assista ao vídeo abaixo:

Caesar[6]

6 Como testar seu código?

Teste seu programa tentando reproduzir todos os exemplos listados neste documento e tenha certeza de que o output é exatamente o esperado.

Lembre-se também de que seu código deve seguir todas as normas de estilo de programação C da disciplina Harvard CS50: Harvard CS50 C Style Guide[7].

7 Como enviar seu código?

Utilize o starter file "`cesar.c`" (mantenha esse padrão de nome), preencha as informações de identificação e envie o arquivo no Autolab, no PSET denominado "César)".

[6] https://www.youtube.com/watch?v=V2uusmv2wxI
[7] https://cs50.readthedocs.io/style/c/