Esse é o **CC50.**
O Curso de Harvard, no Brasil 2012/01

Cfrg 2: Pelcgb

n fre ragerthr ngr: 19:00, frk 09/03

Graun pregrmn qr dhr frh pbqvtb r orz pbzragnqb
qr sbezn dhr n shapvbanyvqnqr frwn ncneragr ncranf cryn yrvghen qbf pbzragnevbf.

Bowrgvibf.

- Zryube snzvyvnevmn-yb pbz nythznf shapbrf r ovoyvbgrpnf.
- Fr niraghene hz cbhpb ab pnzcb qn pevcgbtensvn.

Yrvghen erpbzraqnqn.

- Frpbrf 11 n 14 r 39 qr uggc://vasbezngvpn.ufj.hby.pbz.oe/cebtenznpnb-rz-p.ugz

Esse é o **CC50.**
O Curso de Harvard, no Brasil 2012/01

Pset 2: Crypto

a ser entregue até: 19:00, sex 09/03

Tenha certeza de que seu código é bem comentado
de forma que a funcionalidade seja aparente apenas pela leitura dos comentários.

Objetivos.

- Melhor familiarizá-lo com algumas funções e bibliotecas.
- Se aventurar um pouco no campo da criptografia.

Leitura recomendada.

- Seções 11 a 14 e 39 de `http://informatica.hsw.uol.com.br/programacao-em-c.htm`

Honestidade Acadêmica.

Todo o trabalho feito no sentido do cumprimento das expectativas deste curso deve ser exclusivamente seu, a não ser que a colaboração seja expressamente permitida por escrito pelo instrutor do curso. A colaboração na realização de Psets não é permitida, salvo indicação contrária definida na especificação do Set.

Ver ou copiar o trabalho de outro indivíduo do curso ou retirar material de um livro, site ou outra fonte, mesmo em parte e apresentá-lo como seu próprio constitui desonestidade acadêmica, assim como mostrar ou dar a sua obra, mesmo em parte, a um outro estudante. Da mesma forma é desonestidade acadêmica apresentação dupla: você não poderá submeter o mesmo trabalho ou similar a este curso que você enviou ou vai enviar para outro. Nem poderá fornecer ou tornar as soluções disponíveis para os Psets para os indivíduos que fazem ou poderão fazer este curso no futuro.

Você está convidado a discutir o material do curso com os outros, a fim de melhor compreendê-lo. Você pode até discutir sobre os Psets com os colegas, mas você não pode compartilhar o código. Em outras palavras, você poderá se comunicar com os colegas em Português, mas você não pode comunicar-se em, digamos, C. Em caso de dúvida quanto à adequação de algumas discussões, entre em contato com o instrutor.

Você pode e deve recorrer à Web para obter referências na busca de soluções para os Psets, mas não por soluções definitivas para os problemas. No entanto, deve-se citar (como comentários) a origem de qualquer código ou técnica que você descubra fora do curso.

Todas as formas de desonestidade acadêmica são tratadas com rigor.

Licença.

Notas:

Seu trabalho neste Pset será avaliado em três quesitos principais:

Exatidão. Até que ponto o seu código é consistente com as nossas especificações e livre de bugs?
Design. Até que ponto o seu código é bem escrito*(*escrito claramente, funcionando de forma eficiente, elegante, e / ou lógica)?
Estilo. Até que ponto o seu código é legível (comentado e indentado, com nomes de variáveis apropriadas)?

I need somebody... Help!

☐ Vá para

`http://cc50.com.br`

e faça o login, se solicitado. Depois siga até a lista de discussões.

Daqui em diante, considere a `Lista de Discussões do CC50` *o* lugar para ir quando você tiver alguma dúvida ou pergunta. Além de postar perguntas suas, você também pode procurar respostas às perguntas já feitas por outros.

Espera-se, é claro, que você respeite as políticas do curso de honestidade acadêmica. Postagem de trechos de código sobre o qual você tiver dúvidas não é geralmente um problema. Mas postagem de programas inteiros, mesmo que não funcionem, é definitivamente um problema! Em caso de dúvida sobre se você deve postar ou não a sua pergunta, simplesmente mande um e-mail para `ajuda@cc50.com.br`, especialmente se você precisa mostrar grande parte do seu código. Mas quanto mais perguntas você fizer publicamente, mais os outros também se beneficiarão!

Não se preocupe com o que os outros pensam das suas perguntas, você não será julgado nem pelo Instrutor, nem pelos colegas sobre o que você está perguntando. Certamente, não hesite em postar uma pergunta, porque você acha que ela é "burra". Ela não é!

Começando

☐ Tudo certo, aqui vamos nós! Abra o seu Terminal e crie um diretório chamado `pset2` dentro de cc50, e, em seguida, navegue para dentro desse diretório. Lembra-se como? Não? Sem problemas. Uma vez que você tiver o Terminal aberto, execute

```
mkdir ~/cc50/pset2
```

para criar um diretório chamado `pset2` em seu diretório cc50. Em seguida, execute

```
cd ~/cc50/pset2
```

ou apenas

```
cd cc50/pset2
```

para abrir esse diretório. O prompt agora deve ser semelhante ao abaixo.

```
username@computer (~/cc50/pset2):
```

Se não for, refaça os seus passos e veja se você pode descobrir onde está o erro. Você também pode executar

```
history
```

para ver os seus últimos comandos em ordem cronológica, se você gostaria de fazer alguma investigação. Você também pode percorrer os comandos do seu histórico a qualquer momento usando as setas do seu teclado para cima e para baixo; pressione Enter para reexecutar qualquer comando que você gostaria. Se você ainda não tem certeza de como corrigir, lembre-se que `http://cc50.com.br/forum` é o seu novo amigo!

☐ Por fim copie a Makefile que veio junto com esse documento para dentro do diretório pset2 que você criou.

Todo o trabalho que você fizer para este Pset deve, no fim, residir no diretório `pset2` .

Vamos aquecer com uma música.

☐ Lembre-se da seguinte canção de bar

```
10 garrafas de cerveja no muro, bebo uma, jogo no lixo, 9 garrafas no
muro...
9 garrafas de cerveja no muro, bebo uma, jogo no lixo, 8 garrafas no
muro...
8 garrafas de cerveja no muro, bebo uma, jogo no lixo, 7 garrafas no
muro...
7 garrafas de cerveja no muro, bebo uma, jogo no lixo, 6 garrafas no
muro...
6 garrafas de cerveja no muro, bebo uma, jogo no lixo, 5 garrafas no
muro...
5 garrafas de cerveja no muro, bebo uma, jogo no lixo, 4 garrafas no
muro...
4 garrafas de cerveja no muro, bebo uma, jogo no lixo, 3 garrafas no
muro...
3 garrafas de cerveja no muro, bebo uma, jogo no lixo, 2 garrafas no
muro...
2 garrafas de cerveja no muro, bebo uma, jogo no lixo, 1 garrafa no
muro...
1 garrafa de cerveja no muro, bebo uma, jogo no lixo, nenhuma garrafa no
muro...
```

Seu primeiro desafio desta semana é escrever, em `beer.c`, um programa que escreve, na íntegra, essa versão de "Garrafas de Cerveja no muro". Sua versão deve ser escrita exatamente como a nossa é.

Observe, no entanto, a repetição nos versos desta canção. Talvez você possa utilizar um loop que itera de 1 a 10 (ou de 0 a 9) para gerá-los? Mas eles variam um pouco, então você pode precisar de algumas condições if? Parece que você poderia até implementar algumas funções que recebem, como input, um inteiro e retornam, como output, uma string? Ou talvez você pode armazenar todas as strings em arrays? Hm. Tantas possibilidades!

Existem, como você pode perceber cada vez mais, muitas maneiras diferentes de resolver problemas como este. Escolha uma abordagem, implemente-a, teste-a, e então volte e veja se é

possível melhorá-la antes de ir para o próximo problema! O seu código não deve estar somente correto *(*funcionando), mas deve também apresentar um bom design e um bom estilo. Fique tranquilo pois existem muitas maneiras de se implementar esta canção.

O estilo é fácil. Pergunte a si mesmo algo como: O meu código é bem comentado, mas sem excessos? O meu código é "bonito" *(*consistentemente indentado e não maior do que 80 caracteres de diâmetro)? Dei nomes apropriados às minhas variáveis?

Para o design, pergunte a si mesmo algo como: O meu código é fácil de ler? Estou desperdiçando ciclos da CPU desnecessariamente? O meu código, apesar de funcionar, é mais complicado do que o necessário para resolver esse problema?

Considere o problema resolvido quando você sentir que não há mais espaço para melhorias!

Seu programa terá, pelo menos, uma função `main`. Cabe a você decidir se deseja ou não escrever uma ou mais funções adicionais além de `main`. Você provavelmente vai achar mais fácil compilar o seu programa com:

```
make beer
```

Aí então você pode executá-lo com:

```
./beer
```

Como você vê, `beer` não precisa aceitar quaisquer argumentos de linha de comando. E assim basta declarar `main` com

```
int
main(void)
```

sem qualquer menção a `argc` ou `argv`.

Tudo bem, agora é com você!

Ave, César!

☐ Lembre-se do final da Semana 2, que a Cifra de César criptografa (embaralha de forma reversível) mensagens "girando" cada letra *k* posições, passando de 'Z' para 'A', conforme necessário. Nós utilizaremos o alfabeto padrão de 26 letras `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

`http://pt.wikipedia.org/wiki/Cifra_de_C%C3%A9sar`

Em outras palavras, se *p* é um texto não criptografado, p_i é o *igésimo* caractere de *p,* e *k* é a chave secreta *(*um número inteiro não negativo), então cada letra, c_i no texto criptografado, *c,* é computada como:

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

Esta fórmula talvez faz com que a Cifra de César pareça mais complicada do que é, mas ela é na verdade apenas uma maneira matemática precisa e concisa de expressar o algoritmo. E cientistas da computação amam precisão e concisão.

Por exemplo, suponha que a chave *k (*secreta) seja 13 e que o texto *p* seja "Nao se esqueca de beber o seu Nescau". Agora criptografamos esse *p* com aquela *k,* a fim de obter o texto *c,* cifrado, girando cada uma das letras 13 posições:

```
p: Nao se esqueca de beber o seu Nescau
c: Anb fr rfdhrpn qr ororе b frh Arfpnh
```

Observe como `A` (a primeira letra do texto cifrado) vem 13 letras depois de `N` (a primeira letra do texto original). Da mesma forma `n` (a segunda letra do texto cifrado) está a 13 letras de distância de `a` (a segunda letra do texto original). Enquanto isso, `b` (a terceira letra do texto cifrado) está 13 letras distante de `o` (a terceira letra do texto original), nesse caso (e no primeiro também) tivemos que passar por 'Z' e começar de novo do 'A' para chegar lá. E assim por diante. Esse não é o sistema criptográfico mais seguro, com certeza, mas é divertido para implementar!

Aliás, uma cifra de César com uma chave de 13 é geralmente chamado ROT13:

```
http://pt.wikipedia.org/wiki/ROT13
```

No mundo real, porém, é provavelmente melhor usar ROT26, que se acredita ser duas vezes mais seguro do que ROT13.[1]

De qualquer forma, sua próxima meta é escrever, em `caesar.c`, um programa que criptografa mensagens usando a Cifra de César. Seu programa deve aceitar um argumento de linha de comando único: um número inteiro não negativo, *k.* Se o seu programa é executado sem nenhum argumento de linha de comando ou com mais de um argumento de linha de comando, o programa deve reclamar com o usuário e retornar um valor de `1` (que tende a significar um erro), através do comando abaixo:

```
return 1;
```

Caso contrário, seu programa deve proceder para solicitar ao usuário um texto e depois o output é o texto com cada caractere alfabético "girado" *k* posições; caracteres não-alfabéticos (assim como letras com acento, etc) devem ser emitidos inalterados. Após o print desse texto cifrado, seu programa deve acabar, com a função `main` retornando `0`.

Apesar de existirem apenas 26 letras no alfabeto padrão, você não pode assumir que *k* será menor ou igual a 26; seu programa deve funcionar para todos os valores inteiros não-negativos de *k* menores do que $2^{31} - 26$. Em outras palavras, você não precisa se preocupar se o seu programa, eventualmente quebrar, se o usuário escolher um valor para *k* que é muito grande ou quase grande demais para caber em um `int`. Agora, mesmo que *k* seja maior do que 26, caracteres

[1] `http://www.urbandictionary.com/define.php?term=ROT26`

alfabéticos no input do seu programa devem permanecer caracteres alfabéticos no output. Por exemplo, se *k* é 27, 'A' não deve tornar-se '[' embora o valor ASCII de '[' se encontra 27 posições acima do valor ASCII de 'A'. 'A' deve tornar-se 'B', pois 27 modulo 26 é 1, como um cientista da computação diria. Em outras palavras, valores como *k* = 1 e *k* = 27 são efetivamente equivalentes.

Seu programa deve preservar a capitalização: letras maiúsculas, embora rodadas, devem permanecer letras maiúsculas; letras minúsculas, embora rodadas, devem permanecer letras minúsculas.

Por onde começar? Bem, este programa tem de aceitar um argumento de linha de comando *k*, por isso desta vez você vai querer declarar `main` com:

```
int
main(int argc, char *argv[])
```

Lembre-se que `argv` é um "array" de strings (que são também conhecidas como "char estrela" por razões que veremos em breve). Na verdade, `string` é apenas um sinônimo para `char *`, graças à Biblioteca do CC50, você também poderia declarar `main` com:

```
int
main(int argc, string argv[])
```

se você acha que assim a sintaxe é mais clara. De qualquer maneira, você pode imaginar um array como uma linha de armários da escola, dentro de cada um dos quais está algum valor (e talvez algumas meias). Neste caso, dentro de cada armário se encontra uma `string`. Para abrir o primeiro armário, você pode usar uma sintaxe como `argv`[0], já que arrays sempre possuem o índice 0 como primeiro índice. Para abrir o armário ao lado, você pode usar sintaxe como `argv`[1]. E assim por diante. Claro que, se há `n`, armários é melhor você parar de abrir armários quando encontrar `argv`[n-1], uma vez que `argv[n]` não existe! (Ou isso ou ele pertence a outra pessoa, caso em que você não deve abri-lo)

E assim você pode acessar *k* usando um código parecido com:

```
string k = argv[1];
```

assumindo que k esteja realmente lá! Lembre-se que `argc` é um `int` que é igual ao número de strings que estão em `argv`, então é melhor você verificar o valor do `argc` antes de abrir um armário que possa não existir! Idealmente, `argc` será 2. Por quê? Bem, lembre-se que dentro de `argv`[0], por padrão, se encontra o nome do próprio programa. Então `argc` sempre será pelo menos `1`. Mas para este programa você quer que o usuário forneça um argumento de linha de comando *k,* logo `argc` deve ser `2`. Claro, se o usuário fornece mais de um argumento de linha de comando no prompt, `argc` pode ser superior a `2`, neste caso, é hora para algumas reclamações.

Agora, só porque o usuário digita um número inteiro no prompt, isso não significa que o seu input será automaticamente armazenado em um `int`. Ao contrário, ele será armazenado como uma `string` que só se parece com um `int!` E por isso você vai precisar converter essa `string` para um verdadeiro `int`. Como você tem sorte! A função `atoi`, existe exatamente para este fim. Veja como você pode usá-la:

```
int k = atoi(argv[1]);
```

Perceba que desta vez nós declaramos *k* como um verdadeiro `int` para que você realmente possa fazer alguma aritmética com ele. Ah, muito melhor agora. Aliás, você pode supor que o usuário só conseguirá digitar inteiros na linha de comando. Você não tem que se preocupar se eles digitarem, por exemplo, `foo`, apenas para serem difíceis; `atoi` retornará `0` nesses casos. Aliás, você precisará de um `#include` além de `cc50.h` e `stdio.h` para usar `atoi` sem que o gcc grite com você. Deixamos para você descobrir qual é esse arquivo![1]

Ok, então você tem *k* armazenado como um `int`, agora você terá que pedir ao usuário algum texto. Achamos que a função `GetString` do CC50 pode te ajudar com isso.

Uma vez que você tem tanto k quanto o texto, é hora de criptografar. Lembre-se que você pode iterar sobre todos os caracteres em uma string, imprimindo um de cada vez, com um código como o abaixo:

```
for (int i = 0, n = strlen(p); i < n; i++)
{
    printf("%c", p[i]);
}
```

Em outras palavras, assim como `argv` é um array de strings, uma `string` é um array de caracteres. E assim você pode usar índices dentro de colchetes para acessar caracteres individuais em strings assim como você pode usá-los para acessar strings individuais em `argv`. Legal, né? É claro, a impressão de cada um dos caracteres de uma string não é exatamente criptografia. Bem, talvez tecnicamente, se *k* = 0. Mas o código acima deve ajudá-lo a implementar a sua Cifra de César! Por César!

Aliás, você ainda precisará incluir outro arquivo (com `#include`) para usar `strlen`.[2]

Para que possamos automatizar alguns testes do seu código, seu programa deve se comportar de acordo com o exemplo abaixo. Assumindo que o texto em negrito é o que algum usuário digitou.

```
username@cloud (~/cc50/pset2): ./caesar 13
Nao se esqueca de beber o seu Nescau
Anb fr rfdhrpn qr orore b frh Arfpnh
```

Além de `atoi`, você pode encontrar algumas funções úteis documentadas em:

`http://www.cs50.net/resources/cppreference.com/stdstring/`

Por exemplo, `isdigit` soa interessante. E, quanto à continuação de 'Z' para 'A', não se esqueça do operador de módulo (`%`). Você também pode querer verificar `http://asciitable.com/`,

[1] man `atoi`

[2] man `strlen`

que revela os códigos ASCII para vários caracteres, não somente os alfabéticos, apenas no caso do seu programa acabar imprimindo alguns símbolos estranhos acidentalmente.

☐ Anl sqzazkgn.

Parlez-vous français?

☐ Bem, aquela última cifra não era lá muito segura. Felizmente há um algoritmo mais sofisticado por aí: Vigenère. É, naturalmente, francês:[1]

`http://pt.wikipedia.org/wiki/Cifra_de_Vigen%C3%A8re`

A Cifra de Vigenère é uma melhoria da Cifra de César, criptografando as mensagens usando uma sequência de chaves (ou, dito de outra forma, uma palavra-chave). Em outras palavras, se *p* é um texto simples e *k* é uma palavra-chave *(*uma string alfabética, na qual 'A' e 'a' representam 0, e 'Z' e 'z' representam 25), então cada caractere, c_i, no texto cifrado, *c,* é computado como:

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

Note a utilização de k_j ao invés de apenas *k.* E lembre que se *k* tem menos letras do que *p* as letras de *k* devem ser reutilizadas ciclicamente quantas vezes for preciso para criptografar *p.*

O seu último desafio nesta semana é escrever, em `vigenere.c`, um programa que criptografa mensagens usando a cifra de Vigenère. Este programa deve aceitar um argumento de linha de comando único: a palavra-chave *k,* composta inteiramente de caracteres alfabéticos. Se o seu programa é executado sem nenhum argumento de linha de comando, com mais de um argumento de linha de comando, ou com um argumento de linha de comando que contém qualquer caractere não-alfabético, seu programa deve reclamar e sair imediatamente, com main retornando `1` (assim significando um erro que nossos testes podem detectar). Caso contrário, seu programa deve proceder para solicitar ao usuário por uma sequência de texto *p,* que deve ser então criptografado de acordo com a cifra de Vigenère utilizando a palavra-chave k*,* no fim imprimindo o texto resultante e saindo, com a função `main` retornando `0`.

Quanto aos caracteres em *k,* você deve tratar 'A' e 'a' como 0, 'B' e 'b', como 1 ,. . . , e 'Z' e 'z' como 25. Além disso, seu programa deve apenas aplicar a cifra de Vigenère a um caractere em *p* se esse caractere é uma letra. Todos os outros caracteres (números, símbolos, espaços, sinais de pontuação, *etc)* devem ser emitidas inalterados. Além disso, se seu código está prestes a aplicar o *jésimo* caractere de *k* no *iésimo* caractere *p,* mas esse não parece ser um caractere alfabético, você deve esperar para aplicar esse *jésimo* caractere de *k* no próximo caractere alfabético de *p;* você não deve avançar para o próximo caractere em *k.* Finalmente, o programa deve preservar a capitalização de cada letra de *p.*

[1] Não seja enganado pela discussão do artigo sobre *tabula recta*. Cada c_i pode ser calculado com uma aritmética relativamente simples! Você não precisa de uma matriz bidimensional.

Não sabe por onde começar? Você tem muita sorte, este programa é muito semelhante ao de `César!` Só que desta vez, você precisa decidir qual caractere em *k* você deve usar a medida que você itera por todos os caracteres de *p.*

Para que possamos automatizar alguns testes do seu código, seu programa deve se comportar de acordo com o exemplo abaixo; destaque em negrito são alguns inputs de exemplo.

```
username@computer (~/cc50/pset2): ./vigenere FOOBAR
HELLO, WORLD
MSZMO, NTFZE
```

Como testar o programa, além de prever o que deveria ser impresso, dado um certo input? Bem, lembre que nós somos pessoas agradáveis e queremos facilitar o seu trabalho. Talvez utilizando algo como

http://sharkysoft.com/misc/vigenere/

assim você não terá tanto trabalho encriptando e decriptando textos utilizando papel e caneta.

☐ Esse foi o Set de Problemas 2.