OBJETIVO: Manipulação de Strings em C através da implementação de um sistema de geração de palavras a partir de um alfabeto e uso desse programa para quebrar senhas no Linux.

Atenção: A primeira questão desse trabalho pode ser feita no Windows/Mac. Mas a segunda questão só é possível de ser feita usando o **Linux**. Use uma máquina do laboratório, um Linux, ou alguma máquina virtual (e.g., VirtualBox).

QUESTÃO 1 (6 PONTOS)

Dado um *tamanho máximo das palavras* e um *conjunto de caracteres* (alfabeto), seu programa deverá gerar todas as palavras possíveis usando o conjunto de caracteres até o tamanho máximo definido. Nas "dicas", mais a seguir, tem um algoritmo explicando como fazer isso.

O *tamanho máximo* das palavas será colocado em uma variável global chamada `tamanhoMaximo`. Por exemplo:

```
int tamanhoMaximo = 3;
```

O *conjunto de caracteres* será colocado em uma variável global chamada `caracteres`. Por exemplo:

```
char caracteres[] = "abc";
```

Nesta última, cada letra da string é um caractere que faz parte do conjunto de caracteres. Por exemplo, a string `"abc"` gerará o conjunto de caracteres formado pelas letras `a`, `b` e `c`. Neste trabalho, há a diferenciação de maiúsculas e minúsculas, ou seja, a string `"abcABC"` gerará o conjunto de caracteres formados pelas letras `a`, `b`, `c`, `A`, `B`, `C`. Números e caracteres especiais também poderão fazer parte da string.

Exemplo de Execução usando os valores anteriores (`tamanhoMaximo = 3, caracteres[] = "abc"`):

```
$ ./palavras
a
b
c
aa
ab
ac
ba
bb
bc
ca
cb
cc
aaa
```

```
aab
aac
aba
abb
abc
aca
acb
acc
baa
bab
bac
bba
bbb
bbc
```

```
bca
bcb
bcc
caa
cab
cac
cba
cbb
cbc
cca
ccb
ccc
```

DICAS – LEIA COM ATENÇÃO!

Na seguinte referência:

> Flor, Nick V. and Shannon, Haile (2011) "Technology Corner: Brute Force Password Generation -- Basic Iterative and Recursive Algorithms," Journal of Digital Forensics, Security and Law: Vol. 6 : No. 3 , Article 7.
>
> PDF: https://commons.erau.edu/cgi/viewcontent.cgi?article=1102&context=jdfsl

é explicado como implementar esse algoritmo de duas formas diferentes: usando iteração e usando recursão. Você pode (e deve) se basear nele. Mais especificamente, a Figura 2 (pág. 5) do artigo mostra a implementação usando *recursão*, que é a mais fácil e prática de se implementar. Entretanto, note que as palavras geradas pela função possuem exatamente o tamanho pedido no parâmetro dela. Como neste trabalho, está sendo pedido todas as palavras "até" o tamanho pedido, você precisará executar esta função (no seu *main*) para cada um dos tamanhos, começando de 1 (um) até `tamanhoMaximo`.

QUESTÃO 2 (4 PONTOS + 2 EXTRAS)

No Linux, as senhas dos usuários são criptografadas usando algoritmos *hash* como o MD5, SHA-256, SHA-512 ou YesCrypt. A figura abaixo mostra uma linha do arquivo `/etc/shadow`, responsável por armazenar as senhas dos usuários.

Como você pode ver, o segundo campo da linha (a parte que começa com `$6$ECL`) corresponde à senha criptografada do usuário `fulano`. A principal característica dos algoritmos de *hash* é que é praticamente impossível reverter o processo, ou seja, dada uma assinatura criptografada, não é possível gerar o texto original. Entretanto, dado dois textos iguais, o *hash* deles serão sempre iguais.

Então, quando você faz login, como que o Linux sabe se a sua senha está correta ou não? Basicamente, ele gera o *hash* da senha que você acabou de digitar e o compara com o armazenado no `/etc/shadow`. Se for igual, você digitou a senha correta. Em outras palavras, a sua senha nunca é "descriptografada", pois isso é impossível (acredita-se).

Portanto, dado um *hash* (senha criptografada), a única forma de se "descobrir" a senha original (se você for um atacante) é testando várias ou todas as possibilidades. Por exemplo, podemos começar com a senha "aaa", gerar o *hash* dessa string e compará-lo com o armazenado no `/etc/shadow`. Se for igual, você descobriu a senha. Caso contrário, vamos tentar outra possibilidade (e.g., aab). Esse tipo de ataque é conhecido como *quebra de senha por força-bruta*. Entretanto, como a quantidade possível de senhas (com 8 caracteres) é um número absurdamente grande ($72^8 = 7{,}2*10^{14}$), essa senha não seria descoberta em tempo de ser útil, pois poderia demorar séculos para descobri-la.

De qualquer forma, em alguns cenários que você possui algumas informações que limitam as possibilidades de senhas, é possível descobrir uma senha por força-bruta. Por exemplo, se você olhar de lado um colega digitando a senha e perceber que a mesma possui apenas números e alguns caracteres, isso já tornaria possível descobrir a senha. Isso é o que faremos a seguir.

Para facilitar o desenvolvimento de programas, o Linux provê uma função na biblioteca GLibC chamada `crypt`. Essa função tem como argumentos uma senha em texto limpo (sem estar criptografada) e também uma senha criptografada que queremos comparar. O retorno dele é a forma criptografada da senha em texto limpo, usando a mesma técnica de criptografia que a senha criptografada, uma vez que essa técnica muda de um sistema para outro (e.g., MD5, SHA-512, YesCrypt). Em seguida, precisamos comparar o retorno da função `crypt` novamente com a senha criptografada. Se as duas senhas criptografadas forem iguais, isso significa que a senha em texto limpo corresponde à senha criptografada.

Como exemplo, o programa a seguir verifica se a `senhaCriptografada` corresponde à senha `"FParad0x?"`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <crypt.h>

/**
* @brief Verifica se uma senha em texto é a mesma de uma senha criptografada.
*
* @param senhaTeste Senha em texto que queremos ver se corresponde à senha criptografada.
* @param senhaCriptografada Senha criptografada como no arquivo /etc/shadow.
* @return int Retorna 0 (zero) se as senhas forem iguais. Outro valor, caso contrário.
*/
int verificaSenha(char* senhaTeste, char* senhaCriptografada) {
    char *senhaTesteCriptografada = crypt(senhaTeste, senhaCriptografada);
    return strcmp(senhaTesteCriptografada, senhaCriptografada);
}

void main() {
    char *senhaTeste = "FParad0x?";
    char *senhaCriptografada = "$6$OByk0oaCjQMb9ApC$Z4Zm.GP6/I/t2iJROFazxXEwTiVzdDJZ2"
                               "FdyPdSaRnkPmH9RXdT7xjr30.uy484yx61WsqEUWuvpcqa7X/0oN1";

    if (!verificaSenha(senhaTeste, senhaCriptografada))
        printf("A senha está correta!\n");
    else
        printf("Senhas diferentes.\n");
}
```

Esse programa precisa ser compilado com a **biblioteca crypt**, como mostra o exemplo a seguir (note o `-lcrypt` no final):

```
$ gcc verifica_senha.c -o verifica_senha -lcrypt
```

Usando essas informações, modifique seu programa da Questão 1 para descobrir a senha de um usuário. Considere que a senha criptografada seja uma variável global como no exemplo abaixo:

```c
char *senhaCriptografada = "$6$7yCakIXevncmT6se$m002Lkf2BK6Qgyhc.c/PxMTvcmBAXYtIo"
                           "kUKvWwvB5H5zCt5HhhPOlV8ygebOcSsgNqG74whVwN.8UF9WaGfs/";
```

Seu programa precisa ir testando todas as combinações possíveis de senhas até encontrar a senha correta. Quando a senha for encontrada, o programa deve terminar. Por exemplo, para as variáveis globais abaixo:

```c
int  tamanhoMaximo = 3;
char caracteres[]  = "abc";
char *senhaCriptografada = "$6$7yCakIXevncmT6se$m002Lkf2BK6Qgyhc.c/PxMTvcmBAXYtIo"
                           "kUKvWwvB5H5zCt5HhhPOlV8ygebOcSsgNqG74whVwN.8UF9WaGfs/";
```

Seu programa deve gerar a seguinte saída:

```
$ ./palavras
a --> não
b --> não
c --> não
aa --> não
ab --> não
ac --> não
ba --> não
bb --> não
```

```
bc --> não
ca --> não
cb --> não
cc --> não
aaa --> não
aab --> não
aac --> não
aba --> não
abb --> não
```

```
abc --> não
aca --> não
acb --> não
acc --> não
baa --> não
bab --> não
bac --> sim! Senha encontrada!
```

OUTROS TESTES

No e-mail que você for enviar para o professor, além do código-fonte, responda:

1. Sabendo que uma senha tem, no máximo, 4 caracteres e contém apenas **números**, descubra a senha abaixo:

```
char *senhaCriptografada = "$6$LrSF5BAseToYYHJ0$SYY1avj8FRoRGpn.1kPXuZ6Xn5WTl2kL3"
                           "hxc3yMWdDUyz4c/Ac3Av3WO8Q9LciP8o4c9WaeLcgxIXWaHpJMFb.";
```

2. Sabendo que uma senha tem, no máximo, 3 caracteres e contém apenas **letras minúsculas**, descubra a senha abaixo:

```
char *senhaCriptografada = "$6$rMAk28dVkWjpYoA3$SkWbPYqEB8O/10ryvvjm1qN9BOrkeBOXp"
                           "JScVSGDL5L88OIs0UCBuP.pnd9TQ6SBx60dLKwR9WAzfnLtvjGvj.";
```

3. Sabendo que uma senha tem, no máximo, 5 caracteres e contém apenas **números**, descubra a senha abaixo:

```
char *senhaCriptografada = "$6$l2xE4w9twgjtnZBz$9YK9krslZFraLffy5VNiahAfT.xZNvB54"
                           "j91DMCMIoVFvj335ZKxb11qgVMn.KzU2GqVPPyS2FTBqPSciYq761";
```

ENTREGA DO LABORATÓRIO

O laboratório é presencial e deve ser entregue durante o horário da aula. Para entregar, responda as perguntas acima, anexe o código-fonte e envie o e-mail para horacio@icomp.ufam.edu.br com o assunto “Entrega do 5o Laboratório de LPA”.

Depois de entregue, você pode sair ou, se desejar, pode ajudar algum(a) colega a terminar o trabalho dele(a), desde que não haja cópia do seu código (plágio).