

Prova 1 (2021/01)

Ao concordar em fazer esta prova, eu juro que seguirei o **código de honra**:

1. Não ajudarei e nem pedirei ajuda a ninguém durante o período de submissão do exame;
2. Não divulgarei qualquer informação sobre as minhas soluções durante o período de submissão do exame.

É normal não entender completamente uma questão. Se você tiver dúvidas sobre qualquer questão, envie uma mensagem ***PRIVADA*** no Teams ou um e-mail para olmo@dcc.ufmg.br com o assunto **[PROVA1]** e explique a sua dúvida para mim. Tentarei responder o mais rápido possível, mas a resposta pode demorar se a mensagem for enviada fora do horário comercial.

Importante #1: você não deve imprimir nenhuma mensagem diferente daquelas pedidas nos enunciados das questões.

Importante #2: você pode usar qualquer função de bibliotecas padrões da linguagem C (o que inclui a `math.h`) e criar funções e recursos adicionais sempre que precisar.

Importante #3: Para os exercícios abaixo serem pontuados corretamente, você DEVE usar os nomes das funções que são descritos em cada exercício.

Importante #4: Você não pode implementar a função `"main()"`, pois ela já foi implementada e fará a chamada das funções pedidas na prova.

Importante #5: Assim como na Prática 3 (VPL), você pode avaliar a sua prova quantas vezes quiser.

Importante #6: Leia informações importantes sobre a prova no fórum de avisos do Moodle ou no Teams. Um exemplo de informação importante descrita nesses canais é: **uma prova que não compila é avaliada com a nota zero.**

Sumário

Nesta prova, você vai implementar funções que compõem parte de um sistema bancário. Mais especificamente, você vai implementar funções do módulo de cadastro e de *login* do usuário. No cadastro, inicialmente, você deve registrar um usuário informando somente uma **chave** de cadastro e uma **senha**, ambos inteiros. A **chave** é um identificador único do usuário, como é o CPF, por exemplo. Para se conectar ao sistema (*login*), o usuário deve informar a **chave** e a **senha**. O número de tentativas **n** disponível para o usuário conseguir logar no sistema depende do valor da **chave**. O processo de *login* termina quando o usuário acertar a **senha** ou quando ele realizar as **n** tentativas disponíveis.

Assuma que o usuário irá sempre entrar com dados do mesmo tipo que é pedido no exercício (ex: não entrará com um valor **float** quando é pedido um **int**), sem também extrapolar a quantidade que é pedida (ex: se são pedidos dois valores inteiros, o usuário sempre entrará com dois valores inteiros).

Exercícios

1) O primeiro passo para trabalhar com os clientes consiste em cadastrá-los no sistema. Implemente uma função de nome **cadastraUsuario** que retorna um inteiro (**int**) e que recebe dois endereços de memória inteiros como parâmetros. No primeiro endereço você deve armazenar a **chave** de cadastro do seu cliente e no segundo uma **senha**. Para isso, leia a **chave** e a **senha** do teclado dentro da sua função **cadastraUsuario**, nessa ordem (primeiro a **chave**, depois a **senha**). Além disso, o cadastro deve seguir algumas regras:

- **Regra para o número de tentativas de login:** clientes com chaves maiores que **2000** são clientes mais recentes e, por terem acesso a mais recursos digitais do banco, possuem somente **3** chances de acesso antes do bloqueio do cartão. Em contrapartida, clientes mais antigos possuem **5** chances para acessar.
- **Regra para os valores das chaves e senhas:** o valor das **chaves e das senhas** deve estar limitado dentro de uma faixa, podendo variar de **4 a 6 dígitos** (inclusive). Caso qualquer um dos valores informados (**chave** ou **senha**) esteja fora dessa faixa, a sua função deve pedir novos valores, tanto de **chave** quanto de **senha**. Por exemplo, se a **chave** estiver incorreta e a **senha** correta, tanto a **chave** quanto a **senha** devem ser lidas novamente. **Esse processo deve se repetir enquanto o usuário entrar com valores inválidos. #Importante:** considere que o usuário não irá entrar com valores de **chave** e **senha** iniciados com **0(s)** (ex: **0029**) e nem valores negativos..

A sua função **deve retornar** o número **n** de tentativas de *login* de acordo com a **regra para o número de tentativas de login** descrita acima. Além disso, você não precisa imprimir nenhuma mensagem informativa na sua função! Protótipo:

```
int cadastraUsuario (int *chave, int *senha);
```

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

2) Um sistema de banco deve ser capaz de identificar senhas corretas e validar ou não o acesso de um usuário. Implemente uma função de nome **verificaSenha** que recebe uma **tentativa** (inteiro) e uma **senha** (inteiro), que é a senha correta, e verifica se a **tentativa** é igual à **senha**. Se forem iguais, retorne 1. Se forem diferentes, retorne 0. Protótipo:

```
int verificaSenha(int tentativa, int senha);
```

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

3) Agora precisamos garantir que o cliente só tenha acesso à conta caso consiga memorizar a senha cadastrada. Implemente uma função de nome **acessoConta** que recebe uma **senha** (inteiro) como parâmetro e o número **n** de tentativas (inteiro) que o usuário tem para se conectar. A sua função deve pedir tentativas do usuário enquanto ele não acertar a senha e enquanto não ultrapassar **n**. Em cada tentativa do usuário, você deve usar a função **verificaSenha** para conferir se a tentativa está correta. Caso o usuário acerte a senha com um número de tentativas menor ou igual ao limite atribuído a sua conta (**n**), retorne 1, caso contrário, retorne 0.

```
int acessoConta(int senha, int n);
```

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

4) O seu sistema deve ser capaz de indicar ao gerente do banco se clientes possuem acesso a cartão, segmentando-os de acordo com o score de crédito individual.

Implemente uma função de nome **cartaoCredito** que recebe como parâmetro o **score** do cliente (inteiro) e retorna um **código de crédito** (char). O **score** é um valor inteiro entre 0 e 1000. Clientes com score até 400 não devem ter acesso ao cartão. Nesse caso, imprima “**Cartao negado**” e retorne o caractere ‘N’. Clientes com score entre 401 e 600 necessitam de uma **análise mais detalhada** por parte do banco. Nesse caso, imprima “**Necessita de mais detalhes**” e retorne o caractere ‘X’. Clientes com score entre 601 e 800 têm acesso ao cartão do tipo “B”. Nesse caso, imprima “**Cartao basico**” e retorne o caractere ‘B’. Por fim, clientes com score entre 801 e 1000 têm acesso ao cartão do tipo “A”. Nesse caso, imprima “**Cartao gold**” e retorne o caractere ‘A’. Como o score é um valor entre 0 e 1000, caso seja inserido um valor fora da faixa imprima “**Score invalido**” e retorne o caractere ‘X’. **Importante: note que as impressões não usam caracteres especiais (ex: ó, ã, ç etc)!** Protótipo:

```
char cartaoCredito(int score);
```

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

5) Implemente uma função de nome **banco** que não recebe nenhum parâmetro e vai simular o seu procedimento **main**. Nesta função você deve chamar a função **cadastraUsuario** para ler a **chave** de cadastro e a **senha** do usuário, e obter o número **n** de tentativas que ele tem direito. Depois disso, você deve chamar a função **acessoConta**, enviando a **senha** e **n** como parâmetros. Caso o usuário consiga acessar a conta, imprima: **“Acesso realizado com sucesso\n”** e imprima logo em seguida o código de crédito do usuário, que é retornado pela função **cartaoCredito**. Para simplificar, assuma que o **score** de crédito do usuário é dado pelo resto da divisão da sua **chave** por **1000**. Exemplo: se a chave é **2345**, o **score** de crédito desse usuário é **345**, que é o resto de **(2345 / 1000)**. Caso o cliente não tenha conseguido logar, imprima: **“Cliente bloqueado\n”**. Protótipo:

```
void banco();
```

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

6) Na sua função **acessoConta**, adicione a funcionalidade para permitir que, quando o usuário digitar **-999** como tentativa, ele poderá acertar a senha usando **divisores maiores que 10** da sua senha para acessar a conta. Em outras palavras, qualquer divisor da senha (maior que 10) poderá ser aceito como senha correta. Exemplo: se a **senha** é **4422** e o usuário digitar **-999** como tentativa, o acesso em sua próxima tentativa poderá ser através do próprio número (**4422**), ou de um divisor maior que **10**, por exemplo **2211**. Note que o divisor **2** não funciona (ou seja, é considerada uma tentativa incorreta), pois ele não é maior que **10**. **Importante #1**: entrar com a tentativa **-999** conta como uma tentativa válida, ou seja, **reduz** o número de tentativas restantes do usuário normalmente. **Importante #2**: lembre que a senha tem no mínimo **4** dígitos.

Número de testes para avaliação: 4 de 30 (~13.33% da nota).

7) Finalizado o cadastro, será gerado um **código verificador** da **chave** do usuário. O **código verificador** corresponde ao *número de bits 1* da representação binária da chave. Por exemplo: se a chave for **5**, a representação binária é **“101”**, e o **número de bits 1** é **2**, que é também o **código verificador**. Caso o **número de bits 1** seja igual ou maior a **10**, o **código verificador** é o **dígito mais à direita desse número menos significativo desse número**. Por exemplo, se o **número de bits 1** é **13**, então o **código verificador** é **3**. Implemente uma função denominada **codigoVerificador** para gerar e retornar este **código verificador** a partir de uma **chave**. Protótipo:

```
int codigoVerificador(int chave);
```

Número de testes para avaliação: 6 de 30 (~20.0% da nota).

Execução no VPL

Para conseguir compilar o seu programa, você deve implementar uma versão sintaticamente correta de todas as funções pedidas. Sugiro fortemente que faça isso antes de pensar nas soluções. Além disso, sugiro que avalie o programa dessa forma para poder visualizar os testes de desenvolvimento. Para o exercício **2**, por exemplo, você pode implementar a seguinte função:

```
int verificaSenha(int tentativa, int senha) {return 0;}
```

Para testar um exercício, inicie a execução no VPL e digite o número da função que gostaria de testar (**1, 2, 3, 4, 5, 6** ou **7**). Para as funções que exigem parâmetros (**2, 3, 4, 7**), digite também os valores dos parâmetros. Por exemplo, se quiser testar a função **acessoConta** (exercício **3**) com os parâmetros **senha = 100** e **n = 2**, digite:

3

100 2

Lembre que durante a execução das funções **1, 3, 5** e **6**, valores são pedidos para o usuário e você deve inserir tais valores para testá-las completamente.