

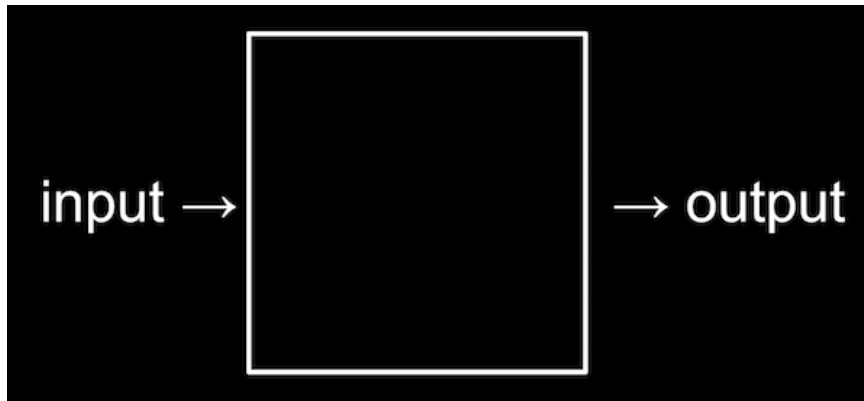
TODAS AS ANOTAÇÕES E EXERCÍCIOS DO CS50

ANOTAÇÕES MÓDULO 0

O que é Ciência da Computação?

A ciência da computação é fundamentalmente sobre resolução de problemas.

Podemos pensar na resolução de problemas como o processo de pegar algumas informações (detalhes sobre nosso problema) e gerar alguns resultados (a solução para nosso problema). A “caixa preta” no meio é a ciência da computação, ou o código que aprenderemos a escrever.



Para começar a fazer isso, precisaremos de uma maneira de representar entradas (inputs) e saídas (outputs), para que possamos armazenar e trabalhar com informações de forma padronizada.

Representando números

Podemos começar com a tarefa de marcar presença, contando o número de pessoas em uma sala. Com a nossa mão, podemos levantar um dedo de cada vez para representar cada pessoa, mas não poderemos contar muito alto. Este sistema é denominado unário, onde cada dígito representa um único valor de um.

Provavelmente aprendemos um sistema mais eficiente para representar números, onde temos dez dígitos, de 0 a 9:

0 1 2 3 4 5 6 7 8 9

- Este sistema é denominado decimal, ou base 10, uma vez que existem dez valores diferentes que um dígito pode representar.

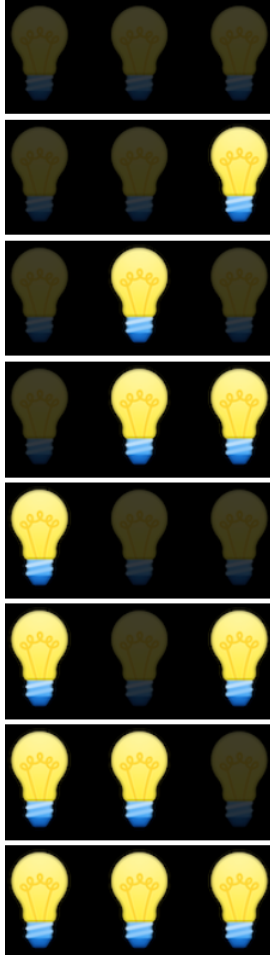
Os computadores usam um sistema mais simples chamado **binário**, ou base dois, com apenas dois dígitos possíveis, **0 e 1**.

- Cada dígito binário também é chamado de **bit**.

Como os computadores funcionam com eletricidade, que pode ser ligada ou desligada, podemos convenientemente representar um bit ligando ou desligando alguma chave para representar 0 ou 1.

- Com uma lâmpada, por exemplo, podemos ligá-la para contar até 1.

Com três lâmpadas, podemos acendê-las em padrões diferentes e contar de 0 (com as três apagadas) a 7 (com as três acesas):



Dentro dos computadores modernos, não existem lâmpadas, mas milhões de pequenos interruptores chamados **transistores** que podem ser ligados e desligados para representar valores diferentes. Por exemplo, sabemos que o seguinte número em decimal representa cento e vinte e três.

1 2 3

- O **3** está na coluna das unidades, o **2** está na coluna das dezenas e o **1** está na coluna das centenas.
-
- Portanto, **123** é $100 \times 1 + 10 \times 2 + 1 \times 3 = 100 + 20 + 3 = 123$.
-
- Cada casa de um dígito representa uma potência de dez, pois há dez dígitos possíveis para cada casa. O lugar mais à direita é para 10^0 , o do meio 10^1 e o lugar mais à esquerda 10^2

$$\begin{matrix} 10^2 & 10^1 & 10^0 \\ 1 & 2 & 3 \end{matrix}$$

Em binário, com apenas dois dígitos, temos potências de dois para cada valor de casa:

$$\begin{matrix} 2^2 & 2^1 & 2^0 \\ \text{Equivalente a:} & \mathbf{4} & \mathbf{2} & \mathbf{1} \end{matrix}$$

Com todas as lâmpadas ou interruptores desligados, ainda teríamos um valor de 0:

$$\begin{matrix} 2^2 & 2^1 & 2^0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{matrix}$$

Agora, se mudarmos o valor binário para, digamos, 0 1 1, o valor decimal seria 3, uma vez que somamos o 2 e o 1:

$$\begin{matrix} 4 & 2 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \end{matrix}$$

Se tivéssemos mais lâmpadas, poderíamos ter um valor binário de 110010, que teria o valor decimal equivalente a 50:

$$\begin{matrix} 32 & 16 & 8 & 4 & 2 & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{matrix}$$

Observe que $32 + 16 + 2 = 50$. Com mais bits, podemos contar até números ainda maiores.

Texto

Para representar as letras, tudo o que precisamos fazer é decidir como os números são mapeados para as letras. Alguns humanos, muitos anos atrás, decidiram coletivamente um mapeamento padrão de números em letras. A letra “A”, por exemplo, é o número 65, e “B” é 66 e assim por diante. Ao usar o contexto, como quando estamos olhando uma planilha ou um e-mail, diferentes programas podem interpretar e exibir os mesmos bits como números ou texto.

O mapeamento padrão, [ASCII](#), também inclui letras minúsculas e pontuação.

Se recebêssemos uma mensagem de texto com um padrão de bits que tivesse os valores decimais **72**, **73** e **33**, esses bits seriam mapeados para as letras **HI!**. Cada letra é normalmente representada com um padrão de oito bits, ou um **byte**, então as sequências de bits que receberíamos são **01001000**, **01001001** e **00100001**.

- Podemos já estar familiarizados com o uso de bytes como uma unidade de medida para dados, como em megabytes ou gigabytes, para milhões ou bilhões de bytes.

Com oito bits, ou um byte, podemos ter 2^8 ou 256 valores diferentes (incluindo zero). (O valor mais alto que podemos contar seria 255.)

Outros caracteres, como letras com acentos e símbolos em outros idiomas, fazem parte de um padrão chamado [Unicode](#), que usa mais bits do que ASCII para acomodar todos esses caracteres.

- Quando recebemos um emoji, nosso computador está apenas recebendo um número binário que mapeia para a imagem do emoji baseado no padrão Unicode. Por exemplo, o emoji “rosto com lágrimas de alegria” tem apenas os bits **000000011111011000000010**:

Imagem, vídeo e sons

Uma imagem, como a imagem do emoji, é composta de cores. Com apenas bits, podemos mapear números para cores também. Existem muitos sistemas diferentes para representar cores, mas um comum é **RGB**, que representa cores diferentes indicando a quantidade de vermelho, verde e azul dentro de cada cor.

Por exemplo, nosso padrão de bits anterior, **72, 73 e 33** pode indicar a quantidade de vermelho, verde e azul em uma cor. (E nossos programas saberiam que esses bits são mapeados para uma cor se abríssemos um arquivo de imagem, em vez de recebê-los em uma mensagem de texto.)

Cada número pode ser um byte, com 256 valores possíveis, portanto, com três bytes, podemos representar milhões de cores. Nossos três bytes de cima representariam um tom escuro de amarelo:

Os pontos, ou quadrados, em nossas telas são chamados de **pixels**, e as imagens são compostas por muitos milhares ou milhões desses pixels também. Então, usando três bytes para representar a cor de cada pixel, podemos criar imagens. Podemos ver os pixels em um emoji se aumentarmos o zoom, por exemplo:

A **resolução** de uma imagem é o número de pixels que existem, horizontalmente e verticalmente, portanto, uma imagem de alta resolução terá mais pixels e exigirá mais bytes para ser armazenada.

Os vídeos são compostos de muitas imagens, mudando várias vezes por segundo para nos dar a aparência de movimento, como um [flipbook](#) antigo faria.

A música também pode ser representada com bits, com mapeamentos de números para notas e durações, ou mapeamentos mais complexos de bits para frequências de som em cada momento transcorrido.

Os formatos de arquivo, como JPEG e PNG, ou documentos do Word ou Excel, também são baseados em algum padrão com o qual alguns humanos concordaram, para representar informações com bits.

Algoritmos

Agora que podemos representar inputs e outputs, podemos trabalhar na resolução de problemas.

Os humanos também podem seguir algoritmos, como receitas para cozinhar. Ao programar um computador, precisamos ser mais precisos com nossos algoritmos para que nossas instruções não sejam ambíguas ou mal interpretadas.

Podemos ter um aplicativo em nossos telefones que armazena nossos contatos, com seus nomes e números de telefone classificados em ordem alfabética. O equivalente “old-school” pode ser uma lista telefônica, uma cópia impressa de nomes e números de telefone.

Nossa contribuição para o problema de encontrar o número de alguém seria a lista telefônica e um nome a ser procurado. Podemos abrir o livro e começar da primeira página, procurando um nome uma página de cada vez. Este algoritmo estaria **correto**, já que eventualmente encontraremos o nome que buscamos se ele estiver no livro.

Podemos folhear o livro duas páginas por vez, mas esse algoritmo não estará correto, pois podemos pular a página com nosso nome nela. Podemos consertar esse bug, ou engano, voltando uma página se formos longe demais, pois sabemos que a lista telefônica está classificada em ordem alfabética.

Outro algoritmo seria abrir a lista telefônica ao meio, decidir se nosso nome estará na metade esquerda ou na metade direita do livro (porque o livro está em ordem alfabética) e reduzir o tamanho do nosso problema pela metade. Podemos repetir isso até encontrar nosso nome, dividindo o problema pela metade a cada vez. Com 1.024 páginas para começar, precisaríamos apenas de 10 etapas de divisão ao meio antes de termos apenas uma página restante para verificar. Podemos ver isso visualizado em uma [animação de dividir uma lista telefônica ao meio repetidamente](#), em comparação com a [animação de pesquisar uma página por vez](#).

Na verdade, podemos representar a eficiência de cada um desses algoritmos com um gráfico:

•

Nossa primeira solução, pesquisar uma página por vez, pode ser representada pela linha vermelha: nosso tempo para resolver aumenta linearmente à medida que o tamanho do problema aumenta. n é um número que representa o tamanho do problema, portanto, com n páginas em nossas listas telefônicas, temos que realizar até n etapas para encontrar um nome.

A segunda solução, pesquisar duas páginas por vez, pode ser representada pela linha amarela: nossa inclinação é menos acentuada, mas ainda linear. Agora, precisamos apenas de (aproximadamente) $n/2$ etapas, já que viramos duas páginas de cada vez.

Nossa solução final, dividindo a lista telefônica ao meio a cada vez, pode ser representada pela linha verde, com uma relação fundamentalmente diferente entre o tamanho do problema e o tempo de resolvê-lo: [logarítmica](#), já que nosso tempo de resolução aumenta cada vez mais lentamente conforme o tamanho do problema aumenta.

Em outras palavras, se a lista telefônica fosse de 1.000 para 2.000 páginas, precisaríamos apenas de mais uma etapa para encontrar nosso nome. Se o tamanho dobrasse novamente de 2.000 para 4.000 páginas, ainda precisaríamos de apenas mais uma etapa. A linha verde é rotulada $\log_2 n$, ou log base 2 de n , já que estamos dividindo o problema por dois em cada etapa.

Quando escrevemos programas usando algoritmos, geralmente nos preocupamos não apenas com o quão corretos eles são, mas também com o quão bem projetados eles são, considerando fatores como eficiência.

Pseudocódigo

Podemos escrever **pseudocódigo**, que é uma representação de nosso algoritmo em inglês preciso (ou alguma outra linguagem humana):

- 1 Pegue a lista telefônica
- 2 Abra no meio da lista telefônica
- 3 Olhe para a página
- 4 Se a pessoa estiver na página
- 5 Ligar para pessoa
- 6 Caso contrário, se a pessoa estiver mais para o início do livro
- 7 Abrir no meio da metade esquerda do livro
- 8 Volte para a linha 3
- 9 Caso contrário, se a pessoa estiver mais para o final do livro
- 10 Abrir no meio da metade direita do livro
- 11 Volte para a linha 3
- 12 Caso contrário
- 13 Desistir

- Com essas etapas, verificamos a página do meio, decidimos o que fazer e repetimos. Se a pessoa não estiver na página e não houver mais páginas sobrando no livro, paramos. E esse caso final é particularmente importante para lembrar. Quando outros programas em nossos computadores esquecem esse caso final, eles podem travar ou parar de responder, uma vez que encontraram um caso que não foi contabilizado, ou continuar a repetir o mesmo trabalho continuamente nos bastidores, sem fazer nenhum progresso.

Algumas dessas linhas começam com verbos ou ações. Começaremos chamando estas *funções*:

- 1 **Pegue** a lista telefônica
- 2 **Abra** no meio da lista telefônica
- 3 **Olhe** para a página
- 4 Se a pessoa estiver na página
- 5 **Ligar** para pessoa
- 6 Caso contrário, se a pessoa estiver mais para o início do livro
- 7 **Abrir** no meio da metade esquerda do livro
- 8 Volte para a linha 3
- 9 Caso contrário, se a pessoa estiver mais para o final do livro
- 10 **Abrir** no meio da metade direita do livro
- 11 Volte para a linha 3
- 12 Caso contrário
- 13 **Desistir**

Também temos ramificações que levam a caminhos diferentes, como bifurcações na estrada, que chamaremos de *condições*:

- 1 Pegue a lista telefônica
- 2 Abra no meio da lista telefônica
- 3 Olhe para a página
- 4 **Se** a pessoa estiver na página
- 5 Ligar para pessoa
- 6 **Caso contrário**, se a pessoa estiver mais para o início do livro
- 7 Abrir no meio da metade esquerda do livro
- 8 Volte para a linha 3
- 9 **Caso contrário**, se a pessoa estiver mais para o final do livro
- 10 Abrir no meio da metade direita do livro
- 11 Volte para a linha 3

- 12 **Caso contrário**
- 13 **Desistir**

E as perguntas que decidem para onde vamos são chamadas de *expressões booleanas*, que eventualmente resultam em um valor de sim ou não, verdadeiro ou falso:

- 1 Pegue a lista telefônica
- 2 Abra no meio da lista telefônica
- 3 Olhe para a página
- 4 Se **a pessoa estiver na página**
- 5 **Ligar para pessoa**
- 6 **Caso contrário, se a pessoa estiver mais para o início do livro**
- 7 **Abrir no meio da metade esquerda do livro**
- 8 **Volte para a linha 3**
- 9 **Caso contrário, se a pessoa estiver mais para o final do livro**
- 10 **Abrir no meio da metade direita do livro**
- 11 **Volte para a linha 3**
- 12 **Caso contrário**
- 13 **Desistir**

Por último, temos palavras que criam ciclos, onde podemos repetir partes de nosso programa, chamadas *loops*:

- 1 Pegue a lista telefônica
- 2 Abra no meio da lista telefônica
- 3 Olhe para a página
- 4 Se a pessoa estiver na página
- 5 **Ligar para pessoa**
- 6 **Caso contrário, se a pessoa estiver mais para o início do livro**
- 7 **Abrir no meio da metade esquerda do livro**
- 8 **Volte para a linha 3**
- 9 **Caso contrário, se a pessoa estiver mais para o final do livro**
- 10 **Abrir no meio da metade direita do livro**
- 11 **Volte para a linha 3**
- 12 **Caso contrário**
- 13 **Desistir**

EXERCÍCIOS MÓDULO 0

Como mexer no Scratch?

Podemos escrever programas com os blocos de construção que acabamos de descobrir:

- funções
-
- condições
-
- Expressões booleanas
-
- rotações

E descobriremos recursos adicionais, incluindo:

- variáveis
-
- tópicos
-
- eventos ...

Antes de aprendermos a usar uma linguagem de programação baseada em texto chamada C, usaremos uma linguagem de programação gráfica chamada [Scratch](#), onde arrastaremos e soltaremos blocos que contêm instruções.

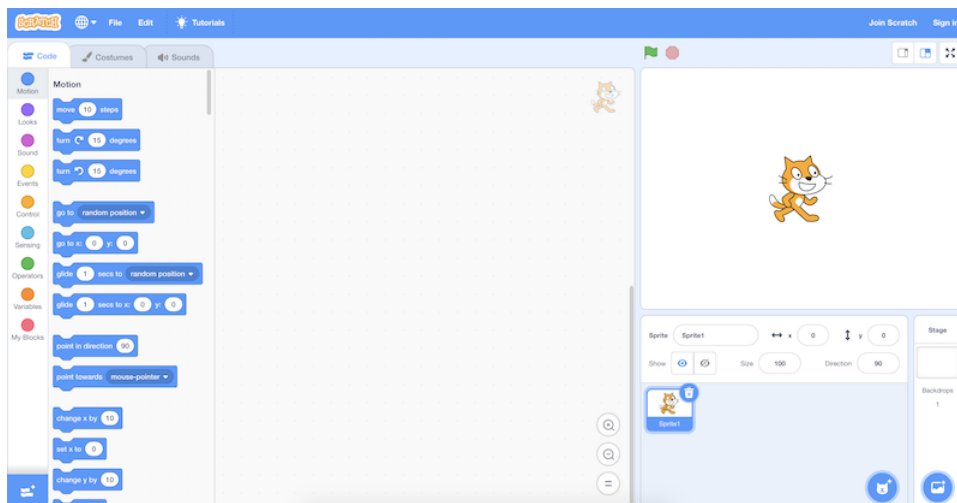
Um programa simples em C que imprime “olá, mundo”, ficaria assim:

```
#include <stdio.h>

int main(void)
{
    printf("oi, mundo\n");
}
```

- Há muitos símbolos e sintaxe, ou seja, o arranjo desses símbolos, que teríamos que descobrir.

O ambiente de programação do Scratch é um pouco mais amigável:



- No canto superior direito, temos um “palco” que será mostrado pelo nosso programa, onde podemos adicionar ou alterar planos de fundo, personagens (chamados de sprites no Scratch) e muito mais.
-

- À esquerda, temos peças de quebra-cabeça que representam funções ou variáveis, ou outros conceitos, que podemos arrastar e soltar em nossa área de instrução no centro.
-
- No canto inferior direito, podemos adicionar mais personagens para nosso programa usar.

Podemos arrastar alguns blocos para fazer o Scratch dizer “olá, mundo”. Ao adicionar o bloco “**green flag clicked**” (“quando a bandeira verde é clicada”) refere-se ao início do nosso programa (já que há uma bandeira verde acima do palco que podemos usar para iniciá-lo), e abaixo dela encaixamos um bloco “**say**” (“dizer”) e digitamos “hello, world” (“olá mundo”). E podemos descobrir o que esses blocos fazem explorando a interface e experimentando.

Também podemos arrastar o bloco “**ask _ and wait**” (“perguntar _ e esperar”), com uma pergunta como “what’s your name” (“qual é o seu nome?”), e combiná-lo com um bloco “**say**” para a resposta. O bloco “**answer**” (“responder”) é uma variável, ou valor, que armazena o que o usuário do programa digita, e podemos colocá-lo em um bloco “**say**” arrastando e soltando também.

Mas não pausamos depois de dizer “Olá” com o primeiro bloco, então podemos usar o bloco “**join**” (“juntar”) para combinar duas frases para que nosso gato possa dizer “olá, David”:

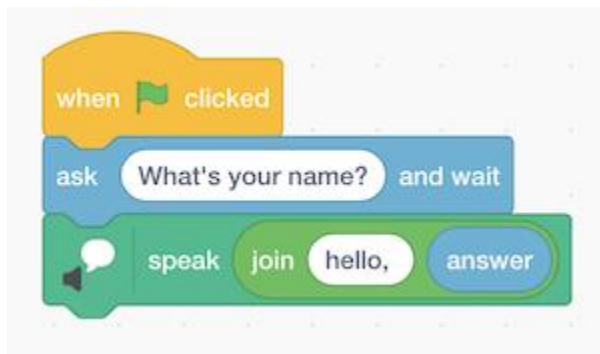


Quando tentamos aninhar blocos ou colocá-los uns dentro dos outros, o Scratch nos ajudará a expandir os locais onde eles podem ser usados. Na verdade, o bloco “**say**” em si é como um algoritmo, onde fornecemos um input de “olá, mundo” e ele produziu a output de Scratch (o gato) “dizendo” essa frase.

O bloco “**ask**” também recebe um input (a pergunta que queremos fazer) e produz um output do bloco “**answer**”.

Podemos então usar o bloco “**answer**” junto com nosso próprio texto, “hello,“, como duas entradas para o algoritmo de junção... o output do qual podemos passar como input para o bloco “**say**”.

No canto inferior esquerdo da tela, vemos um ícone para extensões, e um deles é chamado “**text-to-speech**”. Depois de adicioná-lo, podemos usar o bloco “**say**” para ouvir nosso gato falar:



A extensão "**text-to-speech**", graças à nuvem, ou servidores de computador na internet, está convertendo nosso texto em áudio. Podemos tentar fazer o gato dizer miau:

```
when green flag clicked
play sound: Meow until done
wait 1 seconds
play sound: Meow until done
wait 1 seconds
```

Podemos dizer miau três vezes, mas agora estamos repetindo blocos indefinidamente. Vamos usar um loop ou um bloco de "**repeat**" ("repetição"):



Agora nosso programa atinge os mesmos resultados, mas com menos blocos. Podemos considerar que ele tem um design melhor: se há algo que queremos mudar, só precisaríamos mudar em um lugar ao invés de três.

Podemos fazer com que o gato aponte para o mouse e se mova em direção a ele:

```
when green flag clicked
forever
  point towards mouse-pointer
  move 1 steps
```

Experimentamos a extensão da caneta, usando o bloco "**pen down**" ("caneta para baixo") com uma condição:

```
when green flag clicked
forever
  go to mouse-pointer
  if mouse down? then
    pen down
  else
    pen up
```

Aqui, movemos o gato até o ponteiro do mouse, e se o mouse for clicado, ou para baixo, colocamos o "**pen down**", que desenha. Caso contrário, colocamos a caneta para cima. Repetimos isso muito rapidamente, uma e outra vez, e então produzimos o efeito de desenhar sempre que mantemos o mouse pressionado.

Scratch também tem diferentes fantasias, ou imagens, que podemos usar para nossos personagens. Faremos um programa que pode contar:

```
when green flag clicked
set counter to 1
forever
  say counter for 1 seconds
  change counter by 1
```

Aqui, “counter” é uma variável, cujo valor podemos definir, usar e alterar. Vemos mais alguns programas, como o [salto](#), em que o gato se move para frente e para trás na tela para sempre, girando sempre que estivermos na borda da tela.

Podemos melhorar a animação fazendo com que o gato mude para uma roupa diferente a cada 10 passos no [bounce1](#). Agora, quando clicamos na bandeira verde para executar nosso programa, vemos o gato alternar o movimento de suas pernas.

Podemos até gravar nossos próprios sons com o microfone de nosso computador e reproduzi-los em nosso programa.

Para construir programas cada vez mais complexos, começamos com cada um desses recursos mais simples e os colocamos em camadas um dentro do outro. Também podemos fazer o Scratch miar se tocarmos com o ponteiro do mouse, no [pet0](#).

Na [bark](#), não temos um, mas dois programas no mesmo projeto Scratch. Ambos os programas serão executados ao mesmo tempo depois que a bandeira verde for clicada. Um deles tocará um som de leão-marinho se a variável **silenciada** for configurada como **falsa**, e o outro configurará a variável silenciada de **verdadeiro** para **falso** ou de **falso** para **verdadeiro**, se a tecla de espaço for pressionada.

Outra extensão olha para o vídeo conforme capturado pela webcam do nosso computador e reproduz o som de miado se o vídeo tiver movimento acima de algum limite.

Com vários sprites ou personagens, podemos ter diferentes conjuntos de blocos para cada um deles:

```
when green flag clicked
forever
  if key [space] pressed? then
    say Marco! for 2 seconds
    broadcast event
```

Para um fantoche, temos esses blocos que dizem “Marco!” E, em seguida, um bloco de “**broadcast**” (“transmitir”). Este “evento” é usado para nossos dois sprites se comunicarem, por exemplo enviando uma mensagem nos bastidores. Portanto, o nosso outro fantoche pode simplesmente esperar por este “evento” para dizer “Polo!”:

```
when I receive event
say PolO! for 2 seconds
```

Também podemos usar a extensão “Translate” para dizer algo em outros idiomas:

```
when green flag clicked
ask Qual é seu nome? and wait
```

say **translate** join [olá] answer to *English*

Aqui, o resultado do bloco “**join**” é usado como entrada para o bloco “**translate**”, cujo resultado é passado como entrada para o bloco “**say**”.

Agora que sabemos algumas noções básicas, podemos pensar sobre o design ou a qualidade de nossos programas. Por exemplo, podemos querer que o gato mia três vezes com o bloco “**repeat**”(repetir):

```
when green flag clicked
repeat 3
  play sound Meow until done
  wait 1 seconds
```

Podemos usar abstração, o que simplifica um conceito mais complexo. Neste caso, podemos definir nosso próprio bloco “miau” no Scratch e reutilizá-lo em outro lugar em nosso programa, como visto em [miau3](#). A vantagem é que não precisamos saber como o miado é implementado ou escrito em código, mas apenas usá-lo em nosso programa, tornando-o mais legível.

Podemos até definir um bloco com uma entrada em [meow4](#), onde temos um bloco que faz o gato miar um certo número de vezes. Agora podemos reutilizar esse bloco em nosso programa para miar qualquer número de vezes, da mesma forma como podemos usar os blocos “traduzir” ou “falar”, sem saber os detalhes de implementação ou como o bloco realmente funciona.

Vamos dar uma olhada em mais algumas demos, incluindo [Gingerbread tales remix](#) e [Oscartime](#), que combinam loops, condições e movimento para criar um jogo interativo. Oscartime foi na verdade feito por David muitos anos atrás, e ele começou adicionando um sprite, então um recurso de cada vez, e assim por diante, até que eles acabassem formando um programa mais complicado.

Um ex-aluno, Andrew, criou o [Raining Men](#). Embora Andrew tenha acabado não seguindo a ciência da computação como profissão, as habilidades de resolução de problemas, algoritmos e ideias que aprenderemos no curso são aplicáveis em todos os lugares.

Scratch

É hora de escolher sua própria aventura! Sua tarefa, muito simplesmente, é implementar no **Scratch** qualquer projeto de sua escolha, seja uma história interativa, jogo, animação ou qualquer outra coisa, sujeito apenas aos seguintes requisitos:

- Seu projeto deve ter pelo menos dois sprites, pelo menos um deles deve se parecer com algo diferente de um gato.
-
- Seu projeto deve ter pelo menos três scripts no total (ou seja, não necessariamente três por sprite).
-
- Seu projeto deve usar pelo menos uma condição.
-
- Seu projeto deve usar pelo menos um loop.
-

- Seu projeto deve usar pelo menos uma variável.
-
- Seu projeto deve usar pelo menos um som.

Seu projeto deve ser mais complexo do que a maioria dos demonstrados na aula (muitos dos quais, embora instrutivos, foram bastante curtos), mas pode ser menos complexo do que o Ivy's Hardest Game . Como tal, seu projeto provavelmente deve usar algumas dezenas de peças do quebra-cabeça no geral.

ANOTAÇÕES MÓDULO 1

C

Hoje vamos aprender uma nova linguagem, **C** : uma linguagem de programação que tem todos os recursos do Scratch e muito mais, porém talvez um pouco menos amigável, por ser puramente em texto:

```
#include <stdio.h>
int main(void)
{
    printf("olá, mundo");
}
```

Embora a princípio tentar absorver todos esses novos conceitos possa parecer como beber de uma mangueira de incêndio - pegando emprestado uma frase do MIT - , tenha certeza de que, no final do semestre, estaremos capacitados e experientes em aprender e aplicar esses conceitos.

Podemos comparar muitos dos recursos de programação em C aos blocos que já vimos e usamos no Scratch. Os detalhes da sintaxe são muito menos importantes do que as ideias, às quais já fomos apresentados.

Em nosso exemplo, embora as palavras sejam novas, as ideias são exatamente as mesmas que os blocos "quando a bandeira verde for clicada" e "diga (olá, mundo)" no Scratch:

```
when green flag clicked
say Olá Mundo!
```

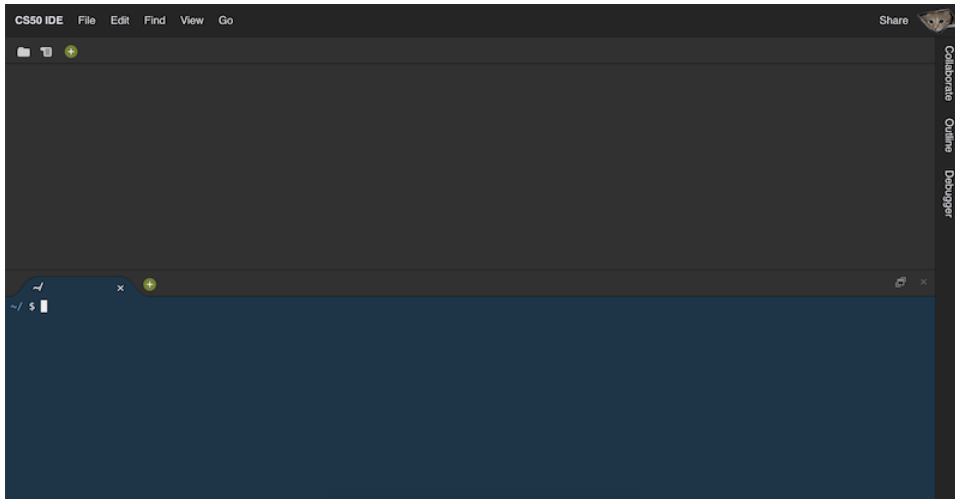
Ao escrever o código, podemos considerar as seguintes qualidades:

- **Correção**, ou se nosso código funciona corretamente, conforme planejado.
-
- **Design**, ou uma medida subjetiva de quão bem escrito nosso código é, com base em quão eficiente, elegante ou logicamente legível ele é, sem repetição desnecessária.
-
- **Estilo**, ou o quão esteticamente formatado nosso código é, em termos de indentação consistente e outra colocação de símbolos. As diferenças de estilo não afetam a exatidão ou o significado do nosso código, mas afetam o quão legível é visualmente.

CS50 IDE

Para começar a escrever nosso código rapidamente, usaremos uma ferramenta para o curso, o [CS50 IDE](#), um ambiente de desenvolvimento integrado que inclui programas e recursos para escrever código. CS50 IDE é construído sobre um IDE baseado em nuvem muito popular, usado por programadores gerais, mas com recursos educacionais adicionais e personalização.

Abriremos o IDE e, após o login, veremos uma tela como esta:



- O painel superior, em branco, conterá arquivos de texto nos quais podemos escrever nosso código.
-
- O painel inferior, uma janela de **terminal**, nos permitirá digitar vários comandos e executá-los, incluindo programas do nosso código acima.

Nosso IDE é executado na nuvem e vem com um conjunto padrão de ferramentas, mas saiba que também existem muitos IDEs baseados em desktop, oferecendo mais personalização e controle para diferentes propósitos de programação, ao custo de maior tempo e esforço para configurá-los.

No IDE, iremos para Arquivo > Novo arquivo e, em seguida, Arquivo > Salvar para salvar nosso arquivo como **hello.c**, indicando que nosso arquivo será um código escrito em C. Veremos que o nome de nossa guia de fato mudou para **hello.c**, e agora vamos colar o código que vimos acima:

```
#include <stdio.h>
int main(void)
{
    printf("olá, mundo");
}
```

Para executar nosso programa, usaremos uma **CLI**, ou **interface de linha de comando**, um *prompt* (um “gatilho”, por assim dizer) ao qual respondemos inserindo comandos de texto. Isso contrasta com a **interface gráfica do usuário**, ou GUI, como o Scratch, onde temos imagens, ícones e botões além do texto.

Compilação

No terminal no painel inferior de nosso IDE, iremos **compilar** nosso código antes de podermos executá-lo. Os computadores só entendem binário, que também é usado para representar instruções como imprimir algo na tela. Nosso **código-fonte** foi escrito em caracteres que podemos ler, mas precisa ser compilado: convertido em **código de máquina**, padrões de zeros e uns que nosso computador possa entender diretamente.

Um programa chamado **compilador** pegará o código-fonte como entrada e produzirá o código de máquina como saída. No IDE CS50, já temos acesso a um compilador, por meio de um comando chamado **make**. Em nosso terminal, digitaremos `make hello`, que encontrará automaticamente nosso arquivo `hello.c` com nosso código-fonte e o compilará em um programa chamado `hello`. Haverá alguma saída, mas nenhuma mensagem de erro em amarelo ou vermelho, então nosso programa foi compilado com sucesso.

Para executar nosso programa, digitaremos outro comando, `./hello`, que procura na pasta atual, `.`, para um programa chamado `hello` e o executa.

Funções e argumentos

Usaremos as mesmas ideias que exploramos no Scratch.

Funções são pequenas ações ou verbos que podemos usar em nosso programa para fazer algo, e as entradas para funções são chamadas de **argumentos**.

- Por exemplo, o bloco “say” (“dizer”) no Scratch pode ter considerado algo como “olá, mundo” como um argumento. Em C, a função de imprimir algo na tela é chamada de **printf** (com **f** significando texto “formatado”, que veremos em breve). E em C, passamos os argumentos entre parênteses, como em **printf (“hello, world”)**; . As aspas duplas indicam que queremos imprimir as letras **hello, world** literalmente, e o ponto-e-vírgula no final indica o fim de nossa linha de código.

As funções também podem ter dois tipos de saídas:

- **efeitos colaterais**, como algo impresso na tela,
-
- e **valores de retorno**, um valor que é passado de volta ao nosso programa que podemos usar ou armazenar para mais tarde.
 -
 - O bloco “ask” (“perguntar”) no Scratch, por exemplo, criou um bloco “answer” (“responder”).
 -
-

Para obter a mesma funcionalidade do bloco “ask”, usaremos uma **biblioteca** ou um conjunto de código já escrito. A Biblioteca CS50 incluirá algumas funções básicas e simples que podemos usar imediatamente. Por exemplo, **get_string** pedirá ao usuário uma string, ou alguma sequência de

texto, e a retornará ao nosso programa. **get_string** recebe algum input e o usa como prompt para o usuário, como “Qual é o seu nome?”, e nós teremos que salvá-lo em uma variável com:

```
string answer = get_string("Qual é o seu nome?");
```

- Em C, o “=” indica **atribuição** ou configuração do valor à direita para a variável à esquerda. E o programa chamará a função **get_string** primeiro para então obter seu output.
-
- E também precisamos indicar que nossa variável chamada **answer** é do **tipo** string, então nosso programa saberá interpretar os zeros e uns como texto.
-
- Finalmente, precisamos nos lembrar de adicionar um ponto-e-vírgula para encerrar nossa linha de código.

No Scratch, também usamos o bloco “answer” dentro de nossos blocos “join” (“juntar”) e “say”. Em C, faremos isso:

```
printf("olá,% s", resposta);
```

- O %s é chamado de **código de formatação**, o que significa apenas que queremos que a função **printf** substitua uma variável onde está o marcador %s. E a variável que queremos usar é **answer**, que passamos para **printf** como outro argumento, separado do primeiro por uma vírgula. (**printf** ("hello, answer")) iria literalmente imprimir hello, answer sempre.)

De volta ao IDE CS50, nós implementaremos o que descobrimos:

```
#include <cs50.h>
#include <stdio.h>
int main(void)
{
    string answer = get_string("Qual é o seu nome?");
    printf("olá, %s", resposta);
}
```

- Precisamos dizer ao compilador para incluir a Biblioteca CS50, com **#include <cs50.h>**, para que possamos usar a função **get_string**.
-
- Também temos a oportunidade de escrever o código usando um “estilo” que favoreça intuitividade, já que poderíamos nomear nossa variável de resposta com qualquer coisa, mas um nome mais descritivo nos ajudará a entender sua finalidade melhor do que um nome mais curto como a ou x.

Depois de salvar o arquivo, precisaremos recompilar nosso programa com **make hello**, já que alteramos apenas o código-fonte, mas não o código de máquina compilado. Outras linguagens ou IDEs podem não exigir que recompilemos manualmente nosso código depois de alterá-lo, mas aqui temos a oportunidade de ter mais controle e compreensão do que está acontecendo nos bastidores.

Agora, ./hello executará nosso programa e solicitará nosso nome conforme pretendido. Podemos notar que o próximo prompt é impresso imediatamente após a saída de nosso programa, como em hello, Brian ~ / \$. Podemos adicionar uma nova linha após a saída de nosso programa, de modo que o próximo prompt esteja em sua própria linha, com \n:

```
printf("olá, %s\n" , resposta);
```

\n é um exemplo de **sequência de escape** ou algum texto que na verdade representa algum outro texto.

Função principal(*main*) e arquivos de cabeçalho

O bloco “quando a bandeira verde for clicada” no Scratch inicia o que consideramos ser o programa principal. Em C, a primeira linha para o mesmo é `int main (void)`, sobre a qual aprenderemos mais nas próximas semanas, seguida por uma chave aberta { e uma chave fechada }, envolvendo tudo o que deveria estar em nosso programa.

```
int main(void)
{

}
```

- Aprenderemos mais sobre como podemos modificar essa linha nas próximas semanas, mas, por enquanto, simplesmente usaremos isso para iniciar nosso programa.

Arquivos de cabeçalho que terminam com .h referem-se a algum outro conjunto de código, como uma biblioteca, que podemos usar em nosso programa. Nós os incluímos com linhas como `#include <stdio.h>` , por exemplo, para a biblioteca de entrada / saída padrão, que contém a função **printf**.

Ferramentas

Com toda a nova sintaxe, é fácil cometer erros ou esquecer algo. Temos algumas ferramentas criadas pela equipe para nos ajudar.

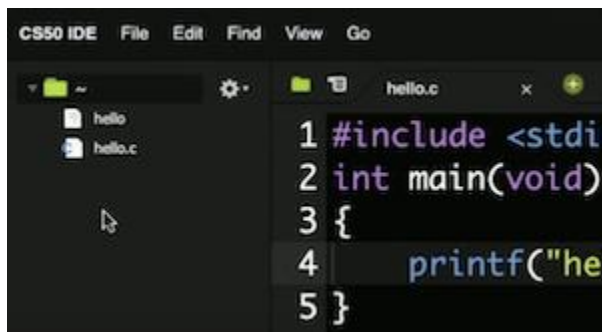
Podemos esquecer de incluir uma linha de código e, quando tentamos compilar nosso programa, vemos muitas linhas de mensagens de erro que são difíceis de entender, pois o compilador pode ter sido projetado para um público mais técnico. **help50** é um comando que podemos executar para explicar problemas em nosso código de uma forma mais amigável. Podemos executá-lo adicionando `help50` à frente de um comando que estamos tentando, como `help50 make hello` , para obter conselhos que possam ser mais compreensíveis.

Acontece que, em C, novas linhas e indentação geralmente não afetam a forma como nosso código é executado. Por exemplo, podemos alterar nossa **função principal(main)** para uma linha, `int main (void) {printf ("hello, world");}`, mas é muito mais difícil de ler, então consideramos que tem um estilo ruim. Podemos executar `style50` , como `style50 hello.c`, com o nome do arquivo de nosso código-fonte, para ver sugestões de novas linhas e recuo.

Além disso, podemos adicionar **comentários**, notas em nosso código-fonte para nós mesmos ou para outras pessoas que não afetem a forma como nosso código é executado. Por exemplo, podemos adicionar uma linha como `// Cumprimentar o usuário`, com duas barras `//` para indicar que a linha é um comentário e, em seguida, escrever o propósito do nosso código ou programa para nos ajudar a lembrar mais tarde.

check50 irá verificar a exatidão do nosso código com alguns testes automatizados. A equipe escreve testes especificamente para alguns dos programas que escreveremos no curso, e as instruções para usar o check50 serão incluídas em cada conjunto de problemas ou laboratório, conforme necessário. Depois de executar check50, veremos algum output nos informando se nosso código passou nos testes relevantes.

O IDE CS50 também nos dá o equivalente a nosso próprio computador na nuvem, em algum lugar da internet, com nossos próprios arquivos e pastas. Se clicarmos no ícone da pasta no canto superior esquerdo, veremos uma árvore de arquivos, uma GUI dos arquivos em nosso IDE:



- Para abrir um arquivo, podemos apenas clicar duas vezes nele. `hello.c` é o código-fonte que acabamos de escrever, e `hello` em si terá muitos pontos vermelhos, cada um dos quais são caracteres não imprimíveis, pois representam instruções binárias para nossos computadores.

Comandos

Como o IDE CS50 é um computador virtual na nuvem, também podemos executar comandos disponíveis no Linux, um sistema operacional como o macOS ou Windows.

No terminal, podemos digitar `ls`, abreviação de `list`, para ver uma lista de arquivos e pastas na pasta atual:

```
~ / $ ls
ola* ola.c
```

- **ola** está em verde com um asterisco para indicar que podemos executá-lo como um programa.

Também podemos remover arquivos com `rm`, com um comando como `rm ola`. Isso nos solicitará uma confirmação e podemos responder com `y` ou `n` para sim ou não.

Com **mv**, ou **move**, podemos renomear arquivos. Com `mv hello.c goodbye.c`, renomeamos nosso arquivo `ola.c` com o nome `goodbye.c`.

Com **mkdir**, ou *diretório make*, podemos criar pastas ou diretórios. Se executarmos `mkdir lecture`, veremos uma pasta chamada `lecture` e podemos mover arquivos para diretórios com um comando como `mv ola.c lecture/`.

Para *mudar os diretórios* em nosso terminal, podemos usar **cd**, como em `cd lecture /`. Nosso prompt mudará de `~/` para `~/ lecture /`, indicando que estamos no diretório de palestras(`lecture`) dentro de `~`. `~` representa nosso diretório inicial ou a pasta padrão de nível superior de nossa conta.

Também podemos usar `..` como uma abreviação para a “pasta-mae”, ou a pasta que contém aquela na qual estamos. Dentro de `~/ lecture /`, podemos executar `mv ola.c ..` para movê-lo de volta para `~`, já que é a pasta-mae de `lecture /`. `cd ..`, da mesma forma, mudará o diretório do nosso terminal para a mae atual. Um único ponto `..`, refere-se ao diretório atual, como em `./ola`.

Agora que nossa pasta `lecture/` está vazia, podemos removê-la com `rmdir lecture/` também.

Tipos e Códigos de Formato

Existem muitos **tipos** de dados que podemos usar para nossas variáveis, que indicam ao computador que tipo de dados eles representam:

- `bool`, uma expressão booleana **verdadeira** ou **falsa**
-
- `char`, um único caractere ASCII como `a` ou `2`
-
- `double`, um valor de vírgula flutuante com mais dígitos do que um **float**
-
- `float`, um valor de vírgula flutuante ou número real com um valor decimal
-
- `int`, inteiros até um certo tamanho ou número de bits
-
- `long`, inteiros com mais bits, para que possam contar mais do que um **int**
-
- `string`, uma linha de caracteres

E a biblioteca CS50 tem funções correspondentes para obter entrada de vários tipos:

- `get_char`
-
- `get_double`
-
- `get_float`
-
- `get_int`
-

- `get_long`
-
- `get_string`

Para **printf**, também, existem diferentes marcadores de posição para cada tipo:

- `%c` para caracteres
-
- `%f` para flutuadores, duplos
-
- `%i` para ints
-
- `%li` para longos
-
- `%s` para strings

Operadores, limitações, truncamento

Existem vários operadores matemáticos que podemos usar também:

- `+` para adição
-
- `-` para subtração
-
- `*` para multiplicação
-
- `/` para divisão
-
- `%` para calcular o resto

Faremos um novo programa, **additional.c**:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    int x = get_int("x: ");

    int y = get_int("y: ");

    printf("%i\n", x + y);
}
```

- Vamos incluir arquivos de cabeçalho para as bibliotecas que sabemos que iremos usar, e então vamos chamar `get_int` para obter inteiros do usuário, armazenando-os em variáveis nomeadas **x** e **y**.
-
- Em seguida, em **printf**, imprimiremos um espaço reservado para um inteiro, **%i**, seguido por uma nova linha. Já que nós queremos imprimir a soma de **x** e **y**, vamos passar em **x + y** para **printf** para substituir na string.
-
- Vamos salvar, executar `make add` no terminal e depois `./addition` para ver nosso programa funcionando. Se digitarmos algo que não seja um inteiro, veremos `get_int` nos pedindo um inteiro novamente. Se digitarmos um número muito grande, como **4000000000**, `get_int` nos alertará novamente. Isso ocorre porque, como em muitos sistemas de computador, um **int** no CS50 IDE é de 32 bits, que pode conter apenas cerca de quatro bilhões de valores diferentes. E uma vez que os inteiros podem ser positivos ou negativos, o maior valor positivo para um **int** só pode ser cerca de dois bilhões, com um valor negativo mais baixo de cerca de dois bilhões negativos, para um total de cerca de quatro bilhões de valores totais.

Podemos mudar nosso programa para usar o tipo **long**:

```
#include <cs50.h>
#include <stdio.h>

int main (void)
{
    long x = get_long("x: ");

    long y = get_long("y: ");

    printf("%li\n", x + y);
}
```

- Agora podemos digitar inteiros maiores e ver um resultado correto conforme o esperado.

Sempre que obtivermos um erro durante a compilação, é uma boa ideia rolar para cima para ver o primeiro erro e corrigi-lo primeiro, já que às vezes um erro no início do programa fará com que o resto do programa seja interpretado com erros também.

Vejamos outro exemplo, **truncation.c**:

```
#include <cs50.h>
#include <stdio.h>

int main (void)
{
    // Pega os números do usuário
    int x = get_int("x: ");
    int y = get_int("y: ");

    // Divide x por y
```

```
float z = x / y;
printf("%li\n", x + y);
}
```

- Vamos armazenar o resultado de **x** dividido por **y** em **z**, um valor de virgula flutuante ou número real, e imprimi-lo também como um valor flutuante.
-
- Mas quando compilamos e executamos nosso programa, vemos **z** impresso como números inteiros como **0,000000** ou **1,000000**. Acontece que, em nosso código, **x / y** é dividido como dois inteiros primeiro, portanto, o resultado fornecido pela operação de divisão também é um inteiro. O resultado é **truncado**, com o valor após a vírgula perdida. Mesmo que **z** seja um **float**, o valor que estamos armazenando nele já é um número inteiro.

Para corrigir isso, vamos fazer o **casting**, ou seja, converter nossos números inteiros para float antes de dividi-los:

```
float z = (float) x / (float) y;
```

O resultado será um float como esperamos e, na verdade, podemos lançar apenas um de **x** ou **y** e obter um float também.

Variáveis e Açúcar Sintático(baixas práticas)

No Scratch, tínhamos blocos como “set [counter] to (0)” que definem uma variável para algum valor. Em C, escreveríamos `int contador = 0;` para o mesmo efeito.

Podemos aumentar o valor de uma variável com `contador = contador + 1;`, onde olhamos primeiro para o lado direito, pegando o valor original do contador, adicionando 1 e, em seguida, armazenando-o no lado esquerdo (de volta ao contador, neste caso).

C também suporta **açúcar sintático** ou expressões abreviadas para a mesma funcionalidade. Nesse caso, poderíamos dizer de maneira equivalente `contador += 1;` para adicionar um ao contador antes de armazená-lo novamente. Também poderíamos escrever `contador++;`, e podemos aprender isso (e outros exemplos) examinando a documentação ou outras referências online.

Condições

Podemos traduzir condições, ou blocos “se”, com:

```
if (x < y)
{
    printf (“x é menor que y\n”);
}
```

- Observe que em C, usamos `{ e }` (bem como indentação) para indicar como as linhas de código devem ser aninhadas.

Podemos ter condições “if” e “else”:

```
if (x < y)
{
    printf("x é menor que y\n");
}
else
{
    printf("x não é menor que y\n");
}
```

E até mesmo “senão se(else if)”:

```
if (x < y)
{
    printf("x é menor que y\n");
}
else if (x > y)
{
    printf("x é maior que y\n");
}
else if (x == y)
{
    printf("x é igual a y\n");
}
```

- Observe que, para comparar dois valores em C, usamos ==, dois sinais de igual.
-
- E, logicamente, não precisamos de if (x == y) na condição final, já que esse é o único caso restante, então podemos apenas dizer o contrário com **else**:

```
if (x < y)
{
    printf("x é menor que y\n");
}
else if (x > y)
{
    printf("x é maior que y\n");
}
else
{
    printf("x é igual a y\n");
}
```

Vamos dar uma olhada em outro exemplo, **conditions.c**:

```
#include <cs50.h>
#include <stdio.h>
```

```

int main(void)
{
    // Usuário entra com o valor de x
    int x = get_int("x: ");

    // Usuário entra com o valor de y
    int y = get_int("y: ");

    // Compara x e y
    if (x < y)
    {
        printf("x é menor que y\n");
    }
    else if (x > y)
    {
        printf("x é maior que y\n");
    }
    else
    {
        printf("x é igual a y\n");
    }
}

```

- Nós incluímos as condições que acabamos de ver, juntamente com duas “chamadas”, ou usos, de `get_int` para obter `x` e `y` do usuário.
-
- Vamos compilar e executar nosso programa para ver se ele realmente funciona conforme o planejado.

Em **concorda.c**, podemos pedir ao usuário para confirmar ou negar algo:

```

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Solicita um caracter para o usuário
    char c = get_char("Você concorda?");

    // Verifica se concordou
    if (c == 'S' || c == 's')
    {
        printf("Concordo.\n");
    }
    else if (c == 'N' || c == 'n')
    {
        printf("Não concordo..\n");
    }
}

```



```
}  
}
```

- Com **get_char**, podemos obter um único caractere e, como só temos um em nosso programa, parece razoável chamá-lo de **c**.
-
- Usamos duas barras verticais, **||**, para indicar um “ou” lógico (matemático), onde qualquer uma das expressões pode ser verdadeira para que a condição seja seguida. (**&&**, por sua vez, indica um “e” lógico, onde ambas as condições deveriam ser verdadeiras.) E observe que usamos dois sinais de igual, **==**, para comparar dois valores, bem como aspas simples, **'**, para envolver nossos valores de caracteres únicos.
-
- Se nenhuma das expressões for verdadeira, nada acontecerá, pois nosso programa não tem um loop.

Expressões booleanas, loops

Podemos traduzir um bloco “para sempre” no Scratch com:

```
while (true)  
{  
    printf (“Oi mundo!\n”);  
}
```

- A palavra-chave **while** (enquanto) requer uma condição, então usamos **true** como a expressão booleana para garantir que nosso loop seja executado para sempre. **while** dirá ao computador para verificar se a expressão é avaliada como **true**(verdadeira) e, em seguida, executar as linhas dentro das chaves. Em seguida, ele repetirá isso até que a expressão não seja mais verdadeira. Nesse caso, **true** sempre será true, então nosso loop é um **loop infinito** ou que será executado para sempre.

Poderíamos fazer algo um certo número de vezes com **while**:

```
int i = 0;  
while (i < 50)  
{  
    printf(“Oi mundo!\n”);  
    i++;  
}
```

- Criamos uma variável, **i**, e a definimos como 0. Então, enquanto **i** é menor que 50, executamos algumas linhas de código, incluindo uma em que adicionamos 1 a **i** a cada passagem. Dessa forma, nosso loop acabará eventualmente, quando **i** atingir um valor de 50.

-
- Nesse caso, estamos usando a variável `i` como contador, mas como ela não tem nenhum propósito adicional, podemos simplesmente chamá-la de `i`.

Mesmo que possamos iniciar a contagem em 1, como demonstrado abaixo, por convenção devemos começar em 0:

```
int i = 1;
while (i <= 50)
{
    printf("Oi mundo!\n");
    i++;
}
```

Outra solução correta, mas possivelmente menos bem projetada, pode começar com o contador em 50 e contar para trás:

```
int i = 50;
while (i > 0)
{
    printf("Oi mundo!\n");
    i--;
}
```

- Nesse caso, a lógica do nosso loop é mais difícil de raciocinar sem servir a nenhum propósito adicional e pode até mesmo confundir os leitores.

Finalmente, mais comumente, podemos usar a palavra-chave `for`:

```
int i = 0;
for (int i = 0; i < 50; i++)
{
    printf("Oi mundo!\n");
}
```

- Novamente, primeiro criamos uma variável chamada `i` e a definimos como 0. Em seguida, verificamos que `i < 50` toda vez que alcançamos o topo do loop, antes de executar qualquer código interno. Se essa expressão for verdadeira, executamos o código interno. Finalmente, depois de executar o código interno, usamos `i++` para adicionar um a `i`, e o loop se repete.
-
- O loop do tipo **for** é mais elegante do que o loop do tipo **while** nesse caso, uma vez que tudo relacionado ao loop está na mesma linha, e somente o código que realmente desejamos executar múltiplas vezes está dentro do loop.

Observe que para muitas dessas linhas de código, como condições do tipo **if** e loops do tipo **for**, não colocamos um ponto e vírgula no final. É assim que a linguagem C foi projetada, muitos anos atrás, e uma regra geral é que apenas as linhas para ações ou verbos têm ponto e vírgula no final.

Abstração

Podemos escrever um programa que imprime **miau**(meow) três vezes:

```
#include <stdio.h>

int main(void)
{
    printf("miau.\n");
    printf("miau.\n");
    printf("mmiau.\n");
}
```

Poderíamos usar um loop **for**, para não ter que copiar e colar tantas linhas:

```
#include <stdio.h>

int main(void)
{
    for(int i = 0; i < 3; i++)
    {
        printf("miau.\n");
        printf("miau.\n");
        printf("miau.\n");
    }
}
```

Podemos mover a linha **printf** para sua própria função, como nossa própria peça de quebra-cabeça:

```
#include <stdio.h>

void miau(void)
{
    printf("miau.\n");
}

int main(void)
{
    for(int i = 0; i < 3; i++)
    {
        miau();
    }
}
```

- Definimos uma função, **miau**, acima de nossa função principal(main).

Mas, convencionalmente, nossa **função principal**(main) deve ser a primeira função em nosso programa, então precisamos de mais algumas linhas:

```
#include <stdio.h>

void miau(void);

int main(void)
{
    for(int i = 0; i < 3; i++)
    {
        miau();
    }
}

void miau(void)
{
    printf("miau.\n");
}
```

- Acontece que precisamos declarar nossa função **miau** primeiro com um **protótipo**, antes de usá-lo em **main**, e realmente defini-lo depois. O compilador lê nosso código-fonte de cima para baixo, então ele precisa saber que o **miau** existirá posteriormente no arquivo.

Podemos até mesmo alterar nossa função de **miau** para obter alguma entrada, **n** e miau **n** vezes:

```
#include <stdio.h>

void miau(int n);

int main(void)
{
    miau(3);
}

void miau(int n)
{
    for(int i = 0; i < 3; i++)
    {
        printf("miau.\n");
    }
}
```

- O **void** antes da função **miau** significa que ela não retorna um valor e, da mesma forma, no geral, não podemos fazer nada com o resultado de **miau**, então apenas a chamamos.

A abstração aqui leva a um design melhor, já que agora temos a flexibilidade de reutilizar nossa função **miau** em vários lugares no futuro.

Vejamos outro exemplo de abstração, `get_positive_int.c`:

```
#include <cs50.h>
#include <stdio.h>

int get_positive_int(void);

int main(void)
{
    int i = get_positive_int();
    printf("%i\n");
}

// Solicita um número inteiro positivo ao usuário
int get_positive_int(void)
{
    int n;
    do
    {
        n = get_int("Número positivo: \n");
    }
    while(n < 1);
    return n;
}
```

- Temos nossa própria função que chama **get_int** repetidamente até que tenhamos algum número inteiro que não seja menor que 1. Com um loop do-while, nosso programa fará algo primeiro, depois verificará alguma condição e repetirá enquanto a condição for verdadeira. Um loop while, por outro lado, verificará a condição primeiro.
-
- Precisamos declarar nosso inteiro **n** fora do loop do-while, pois precisamos usá-lo após o término do loop. O **escopo** de uma variável em C se refere ao contexto, ou linhas de código, dentro do qual ela existe. Em muitos casos, serão as chaves ao redor da variável.
-
- Observe que a função **get_positive_int** agora começa com **int**, indicando que ela tem um valor de retorno do tipo **int** e, em principal, nós o armazenamos em **i** após chamar **get_positive_int()**. Em **get_positive_int**, temos uma nova palavra-chave, **return**, para retornar o valor **n** para onde quer que a função foi chamada.

Mario

Podemos querer um programa que imprima parte de uma tela de um videogame como Super Mario Bros. Em **mario.c**, podemos imprimir quatro pontos de interrogação, simulando blocos:

```
#include <stdio.h>

int main(void)
{
    printf("????\n");
}
```

Com um loop, podemos imprimir vários pontos de interrogação, seguindo-os com uma única nova linha após o loop:

```
#include <stdio.h>

int main(void)
{
    for(int i = 0; i < 3; i++)
    {
        printf("?");
    }
    printf ("\n");
}
```

Podemos obter um número inteiro positivo do usuário e imprimir esse número de pontos de interrogação, usando **n** para o nosso loop:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Pega o valor de n com o usuário
    int n;
    do
    {
        n = get_int("Largura: ");
    }
    while (n < 1);

    // Imprima pontos de interrogação
    for(int i = 0; i < n; i++)
    {
        printf("?");
    }
    printf("\n");
}
```

E podemos imprimir um conjunto bidimensional de blocos com loops aninhados, um dentro do outro:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            printf("#");
        }
        printf("\n");
    }
}
```

- Temos dois loops aninhados, onde o loop externo usa **i** para fazer tudo que contem 3 vezes, e o loop interno usa **j**, uma variável diferente, para fazer algo 3 vezes para cada um desses tempos. Em outras palavras, o loop externo imprime 3 linhas, terminando cada uma delas com uma nova linha, e o loop interno imprime 3 colunas, ou caracteres tipo **#**, *sem* uma nova linha.

Memória, imprecisão e estouro

Nosso computador tem memória, em chips de hardware chamados RAM, memória de acesso aleatório. Nossos programas usam essa RAM para armazenar dados enquanto estão em execução, mas essa memória é finita.

Com **imprecision.c**, podemos ver o que acontece quando usamos valores flutuantes:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    float x = get_float("x: ");
    float y = get_float("y: ");

    printf("%.50f\n", x / y);
}
```

- Com **%.50f**, podemos especificar o número de casas decimais exibidas.
-
- Hmm, agora nós temos ...

x: 1

y: 10

0,100000001490116119384765625000000000000000000000000

- Acontece que isso é chamado de **imprecisão de vírgula flutuante**, em que não temos bits suficientes para armazenar todos os valores possíveis. Com um número finito de bits para um **float**, não podemos representar todos os números reais possíveis (dos quais existe um número *infinito* de), então o computador tem que armazenar o valor mais próximo que puder. E isso pode levar a problemas em que mesmo pequenas diferenças no valor se somam, a menos que o programador use alguma outra maneira para representar os valores decimais com a precisão necessária.

Na semana passada, quando tínhamos três bits e precisávamos contar mais do que sete (ou 111), adicionamos outro bit para obter oito, 1000. Mas se tivéssemos apenas três bits disponíveis, não teríamos lugar para o 1 extra. Ele desapareceria e estaríamos de volta a 000. Esse problema é chamado de **overflow (“vazamento”) de inteiro**, pois um inteiro só pode atingir um tamanho específico antes de ficar sem bits.

O problema Y2K surgiu porque muitos programas armazenavam o ano civil com apenas dois dígitos, como 98 para 1998 e 99 para 1999. Mas quando o ano 2000 se aproximou, os programas tiveram que armazenar apenas 00, levando a confusão entre os anos 1900 e 2000.

Em 2038, também ficaremos sem bits para rastrear o tempo, já que há muitos anos alguns humanos decidiram usar 32 bits como o número padrão de bits para contar o número de segundos desde 1º de janeiro de 1970. Mas com 32 bits representando apenas números positivos, só podemos contar até cerca de quatro bilhões e, em 2038, atingiremos esse limite, a menos que atualizemos o software em todos os nossos sistemas de computador.

EXERCÍCIOS MÓDULO 1

Exercício 1: Mario (versão fácil)

Mundo 1-1

Perto do final do Mundo 1-1 no Super Mario Brothers da Nintendo, Mario deve ascender a pirâmide de blocos alinhada à direita, como demonstrado abaixo.



Vamos recriar essa pirâmide em C, ainda que em texto, usando hashes (#) para tijolos, como visto a seguir. Cada hash é um pouco mais alto do que largo, então a pirâmide em si também é mais alta do que larga.

```
#
  ##
   ###
    ####
     #####
      ######
       #######
        #####
```

O programa que escreveremos se chamará **mario**. E vamos permitir que o usuário decida qual deve ser a altura da pirâmide, primeiro solicitando um número inteiro positivo entre, digamos, 1 e 8, inclusive.

Veja como o programa pode funcionar se o usuário inserir **8** quando solicitado:

```
$ ./mario
Tamanho: 8
```

```
#
  ##
   ###
    ####
     #####
      ######
       #######
        #####
```

Veja como o programa pode funcionar se o usuário inserir **4** quando solicitado:

```
$ ./mario
Tamanho: 4
```

```
#
  ##
```

```
###
####
```

Veja como o programa pode funcionar se o usuário inserir **2** quando solicitado:

```
$ ./mario
Tamanho: 2
#
##
```

Veja como o programa pode funcionar se o usuário inserir **1** quando solicitado:

```
$ ./mario
Tamanho: 1
#
```

Se o usuário não inserir, de fato, um número inteiro positivo entre 1 e 8, inclusive, quando solicitado, o programa deve solicitar novamente ao usuário até que ele coopere:

```
$ ./mario
Tamanho: -1
Tamanho: 0
Tamanho: 42
Tamanho: 9
Tamanho: 4
#
##
###
####
```

[Quero resolver este exercício agora, clique aqui para ir para o IDE.](#)

Pseudocódigo

Primeiro, crie um novo diretório (ou seja, pasta) chamado mario dentro do seu diretório pset1, executando

```
~/ $ mkdir ~/pset1/mario
```

Adicione um novo arquivo chamado **pseudocodigo.txt** dentro do seu diretório mario.

Escreva em **pseudocodigo.txt** algum pseudocódigo que implemente este programa, mesmo que não tenha (ainda!) certeza de como escrevê-lo em código. Não existe uma maneira certa de escrever pseudocódigo, mas frases curtas são suficientes. É provável que seu pseudocódigo use (ou implique o uso!) de uma ou mais funções, condições, expressões booleanas, loops e/ou variáveis.

Clique aqui para ver o Spoiler ;)

Existe mais de uma forma para resolver esse exercício, esse spoiler aqui é apenas uma delas!

- 1- Peça ao usuário o tamanho da altura.
- 2- Se o tamanho da altura for menor que 1 ou maior que 8(ou não inteiro), fique nesse passo até que o usuário insira uma entrada válida.

3- Itere a variável *i* até o tamanho da altura.

4- Imprima os #. [Não se esqueça da quebra de linha!]

Como testar seu código no IDE do CS50?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/mario/less
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 mario.c
```

Exercício 2: Mario (desafio)

Mundo 1-1

No início de World 1-1 em Super Mario Brothers, da Nintendo, Mario deve pular pirâmides de blocos adjacentes, conforme mostrado abaixo.



Vamos recriar essas pirâmides em C, ainda que em texto, usando hashes (#) para tijolos, a la a seguir. Cada hash é um pouco mais alto do que largo, então as pirâmides em si também são mais altas do que largas.

```
# #
## ##
### ###
#### ####
```

O programa que escreveremos se chamará **mario**. E vamos permitir que o usuário decida a altura das pirâmides, primeiro solicitando um número inteiro positivo entre, digamos, 1 e 8, inclusive.

Veja como o programa pode funcionar se o usuário inserir **8** quando solicitado:

```
$ ./mario
```

```
Altura: 8
```

```
  # #
 ## ##
### ###
#### ####
##### #####
```

```
#####  
#####  
#####
```

Veja como o programa pode funcionar se o usuário inserir **4** quando solicitado:

```
$ ./mario  
Altura: 4  
  # #  
  ## ##  
  ### ###  
  #### ####
```

Veja como o programa pode funcionar se o usuário inserir **2** quando solicitado:

```
$ ./mario  
Altura: 8  
  # #  
  ## ##
```

Veja como o programa pode funcionar se o usuário inserir **1** quando solicitado:

```
$ ./mario  
Altura: 8  
  # #
```

Se o usuário não inserir, de fato, um número inteiro positivo entre 1 e 8, inclusive, quando solicitado, o programa deve solicitar novamente ao usuário até que ele escreva o valor correto:

```
$ ./mario  
Altura: -1  
Altura: 0  
Altura: 32  
Altura: 10  
Altura: 4  
  # #  
  ## ##  
  ### ###  
  #### ####
```

Observe que a largura da “lacuna” entre as pirâmides adjacentes é igual à largura de dois hashes, independentemente da altura das pirâmides. Crie um novo diretório (ou seja, pasta) chamado **mario** dentro do seu diretório **pset1**, executando:

```
~/ $ mkdir ~/pset1/mario
```

Crie um novo arquivo chamado **mario.c** dentro do seu diretório **mario**. Modifique **mario.c** de forma que implemente este programa conforme descrito!

Como testar seu código no IDE do CS50?

Seu código funciona conforme prescrito quando você insere:

- -1 (ou outros números negativos)?
-
- 0 ?
-
- 1 a 8 ?
-
- 9 ou outros números positivos?
-
- letras ou palavras?
-
- nenhuma entrada, quando você apenas pressiona Enter?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/mario/more
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 mario.c
```

Exercício 3: Dinheiro (versão fácil)

Algoritmos Gulosos ou Algoritmos Ambiciosos

Ao dar o troco, é provável que você queira minimizar o número de moedas que está distribuindo para cada cliente, para não acabar com o estoque (ou irritar o cliente!). Felizmente, a ciência da computação deu aos caixas em todos os lugares maneiras de minimizar o número de moedas devidas: algoritmos ambiciosos, também conhecidos como gulosos ou gananciosos.

De acordo com o Instituto Nacional de Padrões e Tecnologia (NIST), um algoritmo ambicioso é aquele “que sempre pega a melhor solução imediata, ou local, enquanto encontra uma resposta. Algoritmos ambiciosos encontram a solução geral ou globalmente ideal para alguns problemas de otimização, mas podem encontrar soluções menos do que ideais para algumas instâncias de outros problemas.”

O que tudo isso significa? Bem, suponha que um caixa deva a um cliente algum troco e na gaveta desse caixa estejam moedas de 25, 10, 5 e 1 centavo(s). O problema a ser resolvido é decidir quais moedas e quantas de cada uma entregar ao cliente. Pense em um caixa “ganancioso” como alguém que quer tirar o maior proveito possível desse problema com cada moeda que tira da gaveta. Por exemplo, se algum cliente deve pagar 41 centavos, a maior “mordida”(ou seja, melhor “mordida” imediata ou local) que pode ser feita é 25 centavos. (Essa mordida é “melhor” na medida em que nos deixa mais perto de 0 ¢ mais rápido do que qualquer outra moeda faria.) Observe que uma mordida desse tamanho reduziria o que era um problema de 41 ¢ a um problema de 16 ¢, já que $41 - 25 = 16$. Ou seja, o restante é um problema semelhante, mas menor. Desnecessário dizer que outra mordida de 25 centavos seria muito grande (supondo que o caixa prefere não perder dinheiro), e assim nosso caixa ganancioso mudaria para uma mordida de 10 centavos, deixando-o com um problema de 6 centavos. Nesse ponto, a ganância pede uma mordida de 5 centavos seguida de uma mordida de 1 centavo, ponto em que o problema é resolvido. O cliente recebe um quarto, um

centavo, um centavo e um centavo: quatro moedas no total. Acontece que essa abordagem gananciosa (do algoritmo) não é apenas ótima localmente, mas também globalmente para a moeda dos Estados Unidos (e também da União Europeia). Ou seja, desde que o caixa tenha o suficiente de cada moeda, essa abordagem do maior para o menor renderá o menor número possível de moedas. Quão menor? Bem, diga-nos você!

Detalhes de Implementação

Implemente, em um arquivo chamado **cash.c** em um diretório **~/pset1/cash**, um programa que primeiro pergunta ao usuário quanto dinheiro é devido e depois imprime o número mínimo de moedas com as quais essa mudança pode ser feita.

Use **get_float** para obter a entrada do usuário e **printf** para gerar sua resposta. Suponha que as únicas moedas disponíveis sejam de 25, 10, 5 e 1 centavo(s).

- Pedimos que você use **get_float** para que possa lidar com reais e centavos, embora sem o cifrão. Em outras palavras, se algum cliente deve R\$9.75 (como no caso em que um jornal custa 25 centavos, mas o cliente paga com uma nota de R\$10), suponha que a entrada de seu programa será de **9.75** e não de **R\$9.75** ou **975**. No entanto, se algum cliente deve exatamente R\$9, suponha que a entrada de seu programa será **9.00** ou apenas **9**, mas, novamente, não **R\$9** ou **900**. É claro que, pela natureza dos valores de ponto flutuante, seu programa provavelmente funcionará com entradas como 9.0 e 9.000 também; você não precisa se preocupar em verificar se a entrada do usuário está “formatada” como o dinheiro deveria estar.

Você não precisa tentar verificar se a entrada de um usuário é muito grande para caber em um **float**. Usar **get_float** sozinho garantirá que a entrada do usuário seja realmente um valor de ponto flutuante (ou integral), mas não que seja não negativo.

Se o usuário não fornecer um valor não negativo, seu programa deve solicitar novamente ao usuário uma quantia válida até que o usuário concorde.

Para que possamos automatizar alguns testes do seu código, certifique-se de que a última linha de output do seu programa seja apenas o número mínimo de moedas possível: um inteiro seguido por **\n**.

Cuidado com a imprecisão inerente aos valores de ponto flutuante. Lembre do **floats.c** da aula, em que, se **x** é 2, e **y** é 10, **x / y** não é precisamente dois décimos! E assim, antes de fazer a alteração, você provavelmente desejará converter os dólares inseridos pelo usuário em centavos (ou seja, de um **float** para um **int**) para evitar pequenos erros que poderiam se acumular!

Tome cuidado para arredondar seus centavos até o último centavo mais próximo, por exemplo usando o **round**, que é declarado na **math.h**. Por exemplo, se o real é um **float** com input do usuário (por exemplo, **0.20**), então uma linha como:

```
int centavos = round(reais * 100);
```

irá converter com segurança **0.20** (ou mesmo 0.2000002980232238769531250) em 20.

Utilize o ponto final ao invés de vírgula!!

Seu programa deve se comportar de acordo com os exemplos abaixo.

```
$ ./cash
Troca devida: 0.41
4
```

```
$ ./cash
Troca devida: -0.41
Troca devida: foo
Troca devida: 0.41
4
```

Como testar seu código no IDE do CS50?

Seu código funciona conforme prescrito quando você insere:

- **-1.00** (ou outros números negativos)?
-
- **0.00** ?
-
- **0.01** (ou outros números positivos)?
-
- letras ou palavras?
-
- nenhuma entrada, quando você apenas pressiona Enter?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/cash
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 cash.c
```

Exercício 4: Crédito (desafio)

Um cartão de crédito (ou débito), é claro, é um cartão de plástico com o qual você pode pagar por bens e serviços. Impresso nesse cartão está um número que também é armazenado em um banco de dados em algum lugar, de modo que, quando o cartão for usado para comprar algo, o credor saiba a quem cobrar. Há muitas pessoas com cartões de crédito no mundo, então esses números são bem longos: American Express usa números de 15 dígitos, MasterCard usa números de 16 dígitos e Visa usa números de 13 e 16 dígitos. E esses são números decimais (0 a 9), não binários, o que significa, por exemplo, que a American Express poderia imprimir até $10^{15} = 1.000.000.000.000.000$ de cartões exclusivos! (Isso é, hum, um quatrilhão.)

Na verdade, isso é um pouco exagerado, porque os números de cartão de crédito têm alguma estrutura. Todos os números American Express começam com 34 ou 37; a maioria dos números do MasterCard começa com 51, 52, 53, 54 ou 55 (eles também têm alguns outros números iniciais potenciais com os quais não nos preocupamos neste problema); e todos os números Visa começam

com 4. Mas os números de cartão de crédito também têm um “checksum” embutido, uma relação matemática entre pelo menos um número e outros. Essa soma de verificação permite que os computadores (ou humanos que gostam de matemática) detectem erros de digitação (por exemplo, transposições), se não números fraudulentos, sem ter que consultar um banco de dados, que pode ser lento. É claro que um matemático desonesto certamente poderia criar um número falso que, no entanto, respeite a restrição matemática, portanto, uma pesquisa no banco de dados ainda é necessária para verificações mais rigorosas.

Algoritmo de Luhn

Então, qual é a fórmula secreta? Bem, a maioria dos cartões usa um algoritmo inventado por Hans Peter Luhn, da IBM. De acordo com o algoritmo de Luhn, você pode determinar se um número de cartão de crédito é (sintaticamente) válido da seguinte maneira:

1. Multiplique cada segundo dígito por 2, começando com o penúltimo dígito do número e, em seguida, some os dígitos desses produtos.
- 2.
3. Adicione essa soma à soma dos dígitos que não foram multiplicados por 2.
- 4.
5. Se o último dígito do total for 0 (ou, mais formalmente, se o módulo total 10 for congruente com 0), o número é válido!

Isso é meio confuso, então vamos tentar um exemplo com o cartão Visa do David:
4003600000000014.

1- Para fins de discussão, vamos primeiro sublinhar todos os outros dígitos, começando com o penúltimo dígito do número:

4003600000000014

Ok, vamos multiplicar cada um dos dígitos sublinhados por 2:

$$1 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 6 \cdot 2 + 0 \cdot 2 + 4 \cdot 2$$

Isso nos dá:

$$2 + 0 + 0 + 0 + 0 + 12 + 0 + 8$$

Agora vamos adicionar os dígitos desses produtos (ou seja, não os próprios produtos):

$$2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$$

2- Agora vamos adicionar essa soma (13) à soma dos dígitos que não foram multiplicados por 2 (começando do final):

$$13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$$

3- Sim, o último dígito dessa soma (20) é 0, então o cartão de David é legítimo!

Portanto, validar números de cartão de crédito não é difícil, mas se torna um pouco tedioso manualmente. Vamos escrever um programa.

Detalhes de Implementação

Em um arquivo chamado **credit.c** em um diretório **~/pset1/credit/**, escreva um programa que solicite ao usuário um número de cartão de crédito e, em seguida, informe (via **printf**) se é um número de cartão American Express, MasterCard ou Visa válido, de acordo com as definições de formato de cada um neste documento. Para que possamos automatizar alguns testes do seu código, pedimos que a última linha de saída do seu programa seja **AMEX\n** ou **MASTERCARD\n** ou **VISA\n** ou **INVALID\n**, nada mais, nada menos. Para simplificar, você pode assumir que o input do usuário será inteiramente numérica (ou seja, sem hífen, como pode ser impresso em um cartão real). Mas não presuma que o input do usuário caberá em um **int**! Melhor usar **get_long** da biblioteca do CS50 para obter o input dos usuários. (Por que?)

Considere o seguinte exemplo de como seu próprio programa deve se comportar quando um número de cartão de crédito válido é fornecido (sem hífen).

```
$ ./credit
Número: 4003600000000014
VISA
```

Agora, **get_long** em si rejeitará hífen (e mais) de qualquer maneira:

```
$ ./credit
Número: 4003-6000-0000-0014
Número: foo
Número: 4003600000000014
VISA
```

Mas depende de você pegar entradas que não sejam números de cartão de crédito (por exemplo, um número de telefone), mesmo que sejam numéricos:

```
$ ./credit
Número: 6176292929
INVALID
```

Teste seu programa com um monte de entradas, válidas e inválidas. (Certamente o faremos!) Aqui estão alguns números de cartão que o PayPal recomenda para teste.

Se o seu programa se comporta incorretamente com alguns inputs(ou não compila), é hora de depurar!

Como testar seu código no IDE do CS50?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/credit
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 credit.c
```

Lab 1: População

Laboratório 1: crescimento populacional

Determine quanto tempo leva para uma população atingir um determinado tamanho.

```
$ ./population
Start size: 100
End size: 200
Years: 9
```

Background

Digamos que temos uma população de n lhamas. A cada ano, nascem $n / 3$ novas lhamas e $n / 4$ morrem.

Por exemplo, se começarmos com $n = 1.200$ lhamas, no primeiro ano, $1.200 / 3 = 400$ novas lhamas nascerão e $1.200 / 4 = 300$ lhamas morrerão. No final daquele ano, teríamos $1.200 + 400 - 300 = 1.300$ lhamas.

Para tentar outro exemplo, se começarmos com $n = 1000$ lhamas, no final do ano teremos $1000/3 = 333,33$ novas lhamas. Não podemos ter uma parte decimal de uma lhama, entretanto, vamos truncar o decimal para que **333** novas lhamas nasçam. $1000/4 = 250$ lhamas passarão, então terminaremos com um total de $1000 + 333 - 250 = 1083$ lhamas no final do ano.

Começando

Copie o “código de distribuição” (ou seja, código inicial) a seguir em um novo arquivo em seu IDE chamado `population.c`.

```
#include
#include

int main(void)
{
    // TODO: Solicite o valor inicial ao usuário

    // TODO: Solicite o valor final ao usuário

    // TODO: Calcule o número de anos até o limite

    // TODO: Imprima o número de anos
}
```

Detalhes de Implementação

Conclua a implementação de `population.c`, de forma que calcule o número de anos necessários para que a população cresça do tamanho inicial ao tamanho final.

Seu programa deve primeiro solicitar ao usuário um tamanho inicial da população.

Se o usuário inserir um número menor que 9 (o tamanho mínimo permitido da população), o usuário deve ser solicitado novamente a inserir um tamanho inicial da população até inserir um número maior ou igual a 9. (Se começarmos com menos de 9 lhamas, a população de lhamas ficará estagnada rapidamente!)

Seu programa deve então solicitar ao usuário o tamanho final da população.

Se o usuário inserir um número menor que o tamanho da população inicial, ele deverá ser solicitado novamente a inserir um tamanho da população final até inserir um número que seja maior ou igual ao tamanho da população inicial. (Afinal, queremos que a população de lhamas cresça!)

Seu programa deve então calcular o número (inteiro) de anos necessários para que a população atinja pelo menos o tamanho do valor final.

Finalmente, seu programa deve imprimir o número de anos necessários para que a população de lhama alcance esse tamanho final, como ao imprimir no terminal **Years: n** , onde **n** é o número de anos.

Dicas

Se você deseja solicitar repetidamente ao usuário o valor de uma variável até que alguma condição seja atendida, você pode usar um loop do ... while. Por exemplo, recupere o seguinte código da palestra, que avisa o usuário repetidamente até que ele insira um número inteiro positivo.

```
int n;
do
{
    n = get_int("Inteiro positivo: ");
}
while (n < 1);
```

Como você pode adaptar este código para garantir um tamanho inicial de pelo menos 9, além de um tamanho final que seja pelo menos o tamanho inicial?

Para declarar uma nova variável, certifique-se de especificar seu tipo de dado, um nome para a variável e (opcionalmente) qual deve ser seu valor inicial.

Por exemplo, você pode querer criar uma variável para controlar quantos anos se passaram.

Para calcular quantos anos a população levará para atingir o tamanho final, outro ciclo pode ser útil! Dentro do loop, você provavelmente desejará atualizar o tamanho da população de acordo com a fórmula em Background e atualizar o número de anos que se passaram.

Para imprimir um inteiro **n** no terminal, lembre-se de que você pode usar uma linha de código como `printf("O número é %i\n", n);`

para especificar que a variável **n** deve ser preenchida para o espaço reservado **%i** .

Como testar seu código

Seu código deve resolver os seguintes casos de teste:

- `$./population`
Start size: 1200
End size: 1300
Years: 1
- `$./population`
Start size: -5
Start size: 3
Start size: 9
End size: 5

- End size: 18
Years: 8
- \$./population
Start size: 20
End size: 1
End size: 10
End size: 100
Years: 20
- \$./population
Start size: 100
End size: 1000000
Years: 115

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50** . Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/population
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50** .

```
style50 population.c
```

ANOTAÇÕES MÓDULO 2

Compilação

Da última vez, aprendemos a escrever nosso primeiro programa em C, imprimindo “olá, mundo” na tela.

Nós o compilamos com **make hello** primeiro, transformando nosso código-fonte em código de máquina antes de podermos executar o programa compilado com **./hello**.

make é na verdade apenas um programa que chama **clang**, um compilador, com opções. Poderíamos compilar nosso arquivo de código-fonte, **hello.c**, nós mesmos executando o comando **clang hello.c**. Parece que nada aconteceu, o que significa que não houve erros. E se executarmos **ls**, agora vemos um arquivo **a.out** em nosso diretório. O nome do arquivo ainda é o padrão, então podemos executar um comando mais específico: **clang -o hello hello.c**.

Adicionamos outro **argumento de linha de comando** ou uma entrada para um programa na linha de comando como palavras extras após o nome do programa. **clang** é o nome do programa e **-o, hello e hello.c** são argumentos adicionais. Estamos dizendo ao **clang** para usar **hello** como o nome do arquivo de saída e usar **hello.c** como código-fonte. Agora, podemos ver **hello** sendo criado como output.

Se quisermos usar a biblioteca do CS50, via **#include <cs50.h>**, para a função **get_string**, também temos que adicionar um sinalizador: **clang -o hello hello.c -lcs50**:

```
#include <cs50.h>
#include <stdio.h>
```

```
int main(void)
```

```
{
    string nome = get_string("Qual o seu nome?");
    printf("oi, %s\n", nome);
}
```

- O sinalizador **-l** vincula o arquivo **cs50**, que já está instalado no CS50 IDE, e inclui o código de máquina para **get_string** (entre outras funções) que nosso programa pode consultar e usar também.

Com o **make**, esses argumentos são gerados para nós, uma vez que a equipe também configurou o **make** no IDE CS50.

Compilar o código-fonte em código de máquina é, na verdade, feito em etapas menores:

- pré-processamento
-
- compilação
-
- montagem
-
- linkagem/vinculação

O **pré-processamento** geralmente envolve linhas que começam com **#**, como **#include**. Por exemplo, **#include <cs50.h>** dirá ao **clang** para procurar por esse arquivo de cabeçalho, pois ele contém o conteúdo que queremos incluir em nosso programa. Então, o **clang** irá essencialmente substituir o conteúdo desses arquivos de cabeçalho em nosso programa. Por exemplo ...

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string nome = get_string("Qual o seu nome?");
    printf("oi, %s\n", nome);
}
```

... será pré-processado em:

```
...
string get_string(string prompt);
int printf(string format, ...);
...

int main(void)
{
    string nome = get_string("Qual o seu nome?");
    printf("oi, %s\n", nome);
}
```

Isso inclui os protótipos de todas as funções dessas bibliotecas que incluímos, para que possamos usá-las em nosso código.

A **compilação** pega nosso código-fonte, em C, e o converte em outro tipo de código-fonte chamado **código assembly**, que se parece com isto:

```
...
main:                                # @main
    .cfi_startproc
# BB#0:
    pushq    %rbp
.Ltmp0:
    .cfi_def_cfa_offset 16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq     %rsp, %rbp
.Ltmp2:
    .cfi_def_cfa_register %rbp
    subq     $16, %rsp
    xorl     %eax, %eax
    movl     %eax, %edi
    movabsq  $.L.str, %rsi
    movb     $0, %al
    callq    get_string
    movabsq  $.L.str.1, %rdi
    movq     %rax, -8(%rbp)
    movq     -8(%rbp), %rsi
    movb     $0, %al
    callq    printf
    ...
```

- Essas instruções são de nível inferior e estão mais próximas das instruções binárias que o processador de um computador pode entender diretamente. Eles geralmente operam nos próprios bytes, em oposição a abstrações como nomes de variáveis.

A próxima etapa é pegar o código do *assembly* e traduzi-lo em instruções em binário, **montando-o**. As instruções em binário são chamadas de **código de máquina**, que a CPU de um computador pode executar diretamente.

A última etapa é a **linkagem/vinculação**, onde versões previamente compiladas de bibliotecas que incluímos anteriormente, como **cs50.c**, são realmente combinadas com o binário de nosso programa. Portanto, terminamos com um arquivo binário, **a.out** ou **hello**, que é o código de máquina combinado para **hello.c**, **cs50.c** e **stdio.c**. (No CS50 IDE, o código de máquina pré-compilado para **cs50.c** e **stdio.c** já foi instalado, e **clang** foi configurado para encontra-los e usá-los.)

Essas quatro etapas foram abstraídas ou simplificadas pelo **make**, portanto, tudo o que precisamos implementar é o código de nossos programas.

Debugging/Depuração

Bugs são erros ou problemas em programas que fazem com que eles se comportem de maneira diferente do pretendido. E o debugging (ou depuração) é o processo de localização e correção desses bugs.

Na semana passada, aprendemos sobre algumas ferramentas que nos ajudam a escrever código que compila, tem bom estilo e está correto:

- **help50**
-
- **style50**
-
- **check50**

Podemos usar outra “ferramenta”, a função **printf**, para imprimir mensagens e variáveis para nos ajudar a depurar.

Vamos dar uma olhada em buggy0.c:

```
#include <stdio.h>
```

```
int main(void)
{
    // Imprime 10 hashes
    for (int i = 0; i <= 10; i++)
    {
        printf("# \n");
    }
}
```

- Hmm, queremos imprimir apenas 10 #s, mas há 11. Se não soubéssemos qual é o problema (já que nosso programa está compilando sem erros e agora temos um erro lógico), poderíamos adicionar outro **printf** temporariamente:

```
#include <stdio.h>
```

```
int main(void)
{
    for (int i = 0; i <= 10; i++)
    {
        printf("i vale agora %i\n",i);
        printf("# \n");
    }
}
```

- Agora, podemos ver que **i** começou em 0 e continuei até chegar a 10, mas devemos fazer nosso **loop for** parar quando estiver em 10, com **i < 10** em vez de **i <= 10**.

No IDE CS50, temos outra ferramenta, **debug50**, para nos ajudar a depurar programas. Esta é uma ferramenta escrita pela equipe que se baseia em uma ferramenta padrão chamada **gdb**. Ambos os **depuradores** são programas que executam nossos próprios programas passo a passo e nos permitem examinar as variáveis e outras informações enquanto nosso programa está em execução.

Executaremos o comando **debug50 ./buggy0**, e ele nos dirá para recompilar nosso programa desde que o alteramos. Então, ele nos dirá para adicionar um **ponto de interrupção (breakpoint)** ou indicador para uma linha de código onde o depurador deve pausar nosso programa.

- Usando as teclas para cima e para baixo no terminal, podemos reutilizar comandos do passado sem digitá-los novamente.

Clicaremos à esquerda da linha 6 em nosso código e um círculo vermelho aparecerá:

•

Agora, se executarmos **debug50 ./buggy0** novamente, veremos o painel do depurador aberto à direita:

•

Vemos que a variável que criamos, **i**, está na seção **Variáveis locais**, e vemos que há um valor **0**.

Nosso ponto de interrupção pausou nosso programa na linha 6, destacando essa linha em amarelo. Para continuar, temos alguns controles no painel do depurador. O triângulo azul continuará nosso programa até chegarmos a outro ponto de interrupção ou o fim de nosso programa. A seta curva à sua direita, Step Over, irá “passar por cima” da linha, executando-a e pausando nosso programa novamente imediatamente após.

Portanto, usaremos a seta curva para percorrer a próxima linha e ver o que muda depois. Estamos na linha **printf** e, pressionando a seta curva novamente, vemos um único **#** impresso em nossa janela de terminal. Com outro clique na seta, vemos o valor de **i** mudar para **1**. Podemos continuar clicando na seta para ver o nosso programa rodar, uma linha de cada vez.

Para sair do depurador, podemos pressionar **control + C** para interromper o programa em execução.

Vejamos outro exemplo, buggy1.c:

```
#include <cs50.h>
#include <stdio.h>

// Prototype
int get_negative_int(void);

int main(void)
{
    // Pega um número inteiro negativo do usuário
    int i = get_negative_int();
    printf ("%i \n", i);
}
```



```
int get_negative_int(void)
{
    int n;
    do
    {
        n = get_int ("Número inteiro negativo:");
    } enquanto (n < 0);
    return n;
}
```

- Implementamos outra função, **get_negative_int**, para obter um número inteiro negativo do usuário. Precisamos lembrar de colocar o protótipo antes de nossa função **principal(main)** e, em seguida, nosso código é compilado.

Mas quando executamos nosso programa, ele continua nos pedindo um número inteiro negativo, mesmo depois de fornecermos um. Vamos definir um ponto de interrupção na linha 10, **int i = get_negative_int () ;**, já que é a primeira linha de código interessante. Executaremos **debug50 ./buggy1** e veremos na seção Call stack (“pilha de comandos”) do painel de depuração que estamos na função **principal(main)**. (A “pilha de comandos” refere-se a todas as funções que foram chamadas em nosso programa no momento e ainda não retornaram. Até agora, apenas a função **principal(main)** foi chamada.)

Clicaremos na seta apontando para baixo, Step Into, e o depurador nos levará para a função chamada nessa linha, **get_negative_int**. Vemos a pilha de chamadas atualizada com o nome da função e a variável **n** com o valor **0**:

•

Podemos clicar na seta Step Over novamente e ver **n** ser atualizado com **-1**, que é realmente o que inserimos:

•

Clicamos em Step Over novamente e vemos nosso programa voltando para dentro do loop. Nosso **while** loop ainda está em execução, então a condição que ele verifica ainda deve ser **verdade(true)**. E vemos que **n < 0** é verdadeiro mesmo se inserirmos um número inteiro negativo, portanto, devemos corrigir nosso bug alterando-o para **n >= 0**.

Podemos economizar muito tempo no futuro investindo um pouco agora para aprender como usar o **debug50**!

Também podemos usar **ddb**, abreviação de “duck debugger”, uma [técnica real](#) em que explicamos o que estamos tentando fazer com um pato de borracha e, muitas vezes, percebemos nosso próprio erro de lógica ou implementação conforme o explicamos.

Memória

Em C, temos diferentes tipos de variáveis que podemos usar para armazenar dados, e cada uma delas ocupa uma quantidade fixa de espaço. Na verdade, diferentes sistemas de computador variam

na quantidade de espaço realmente usado para cada tipo, mas trabalharemos com as quantidades aqui, conforme usadas no IDE CS50:

- bool 1 byte
-
- char 1 byte
-
- double 8 bytes
-
- float 4 bytes
-
- int 4 bytes
-
- long 8 bytes
-
- string ? bytes
-
- ...

Dentro de nossos computadores, temos chips chamados RAM, **memória** de acesso aleatório , que armazena dados para uso de curto prazo, como o código de um programa enquanto está sendo executado ou um arquivo enquanto está aberto. Podemos salvar um programa ou arquivo em nosso disco rígido (ou SSD, unidade de estado sólido) para armazenamento de longo prazo, mas usar RAM porque é muito mais rápido. No entanto, a RAM é volátil, ou requer energia para manter os dados armazenados.

Podemos pensar nos bytes armazenados na RAM como se estivessem em uma grade:

•

- Na realidade, existem milhões ou bilhões de bytes por chip.
-
- Cada byte terá um local no chip, como o primeiro byte, o segundo byte e assim por diante.

Em C, quando criamos uma variável do tipo **char**, que terá o tamanho de um byte, ela será armazenada fisicamente em uma dessas caixas na RAM. Um inteiro, com 4 bytes, ocupará quatro dessas caixas.

Arrays/Vetores

Digamos que quiséssemos calcular a média de três variáveis:

```
#include <stdio.h>
```

```
int main(void)
{
    int score1 = 72;
    int score2 = 73;
    int score3 = 33;
```

```
    printf ("Média: %f \n", (score1 + score2 + score3) / 3.0);
}
```

- Dividimos não por **3**, mas por **3.0**, então o resultado também é um float. <.p>
-
- Podemos compilar e executar nosso programa e ver uma média impressa.

Enquanto nosso programa está em execução, as três variáveis **int** são armazenadas na memória:

- Cada **int** ocupa quatro caixas, representando quatro bytes, e cada byte por sua vez é composto de oito bits, 0s e 1s armazenados por componentes elétricos.

Acontece que, na memória, podemos armazenar variáveis uma após a outra, consecutivamente, e acessá-las mais facilmente com loops. Em C, uma lista de valores armazenados um após o outro de forma contígua é chamada de **array** (uma espécie de matriz) .

Para nosso programa acima, podemos usar pontuações **int scores[3]**; para declarar uma matriz de três inteiros.

E podemos atribuir e usar variáveis em uma matriz com **scores[0] = 72**. Com os colchetes, estamos indexando ou indo para a posição "0" na matriz. As matrizes são indexadas por zero, o que significa que o primeiro valor tem índice 0 e o segundo valor tem índice 1 e assim por diante.

Vamos atualizar nosso programa para usar um array:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    int scores[3];
    scores[0] = get_int("Pontuação:");
    scores[1] = get_int("Pontuação:");
    scores[2] = get_int("Pontuação:");
    // Imprimir média
    printf ("Média: %f \n", (scores[0] + scores[1] + scores[2]) / 3.0);
}
```

- Agora, estamos pedindo ao usuário três valores e imprimindo a média como antes, mas usando os valores armazenados no array.

Como podemos definir e acessar itens em uma matriz com base em sua posição, e essa posição também pode ser o valor de alguma variável, podemos usar um loop:

```
#include <cs50.h>
#include <stdio.h>
```

```
int main(void)
{
    int scores[3];
    for (int i = 0; i < 3; i++)
    {
        scores[i] = get_int("Pontuação:");
    }
    // Imprimir média
    printf ("Média:%f\n", (scores[0] + scores[1] + scores[2]) / 3,0);
}
```

- Agora, em vez de codificar permanentemente ou especificar manualmente cada elemento três vezes, usamos um **for** loop e **i** como o índice de cada elemento no array.

E repetimos o valor 3, que representa o comprimento do nosso array, em dois lugares diferentes. Portanto, podemos usar uma **constante** ou variável com um valor fixo em nosso programa:

```
#include <cs50.h>
#include <stdio.h>

const int TOTAL = 3;

int main(void)
{
    int scores[TOTAL];
    for (int i = 0; i < TOTAL; i++)
    {
        scores[i] = get_int("Pontuação:");
    }
    // Imprimir média
    printf ("Média: %f \n", (scores[0] + scores[1] + scores[2]) / TOTAL);
}
```

- Podemos usar a palavra-chave **const** para informar ao compilador que o valor de **TOTAL** nunca deve ser alterado por nosso programa. E por convenção, colocaremos nossa declaração da variável fora da função **principal(main)** e colocaremos seu nome em maiúscula, o que não é necessário para o compilador, mas mostra a outros humanos que esta variável é uma constante e torna fácil ver desde o início .
-
- Mas agora nossa média estará incorreta ou quebrada se não tivermos exatamente três valores.

Vamos adicionar uma função para calcular a média:

```
float media(int quantidade, int array[])
{
    int soma = 0;
    for (int i = 0; i < quantidade; i++)
```

```

    {
        soma += array[i];
    }
    return soma / (float) quantidade;
}

```

- Vamos passar o comprimento e uma matriz de **ints** (que pode ser de qualquer tamanho) e usar outro loop dentro de nossa função auxiliar para adicionar os valores em uma variável de **soma**. Usamos (**float**) para converter o **comprimento** em um float, então o resultado que obtemos ao dividir os dois também é um float.
-
- Agora, em nossa função **principal(main)**, podemos chamar nossa nova função **média** com **printf ("Média:%f\n", media(TOTAL, scores));**. Observe que os nomes das variáveis na função **principal(main)** não precisam corresponder aqueles usados para fazer a **média**, uma vez que apenas os valores são passados.
-
- Precisamos passar o comprimento da matriz para a função que faz a **média**, para que ela saiba quantos valores existem.

Caracteres

Podemos imprimir um único caractere com um programa simples:

```
#include <stdio.h>
```

```

int main(void)
{
    char c = '#';
    printf("%c\n", c);
}

```

Quando executamos este programa, obtemos # impresso no terminal.

Vamos ver o que acontece se mudarmos nosso programa para imprimir c como um inteiro:

```
#include <stdio.h>
```

```

int main(void)
{
    char c = '#';
    printf("%i\n", (int) c);
}

```

- Quando executamos este programa, obtemos **35** impressos. Acontece que **35** é de fato o código ASCII para um símbolo #.
-

- Na verdade, não precisamos converter **c** para um **int** explicitamente; o compilador pode fazer isso por nós neste caso.

Um **char** é um único byte, então podemos imaginá-lo como sendo armazenado em uma caixa na grade de memória acima.

Strings

Podemos imprimir uma string, ou algum texto, criando uma variável para cada caractere e imprimindo-os:

```
#include <stdio.h>
```

```
int main(void)
{
    char c1 = 'H';
    char c2 = 'i';
    char c3 = '!';
    printf("%c%c%c\n", c1, c2, c3);
}
```

- Aqui, veremos o **Hi!** impresso.

Agora vamos imprimir os valores inteiros de cada caractere:

```
#include <stdio.h>
```

```
int main(void)
{
    char c1 = 'H';
    char c2 = 'i';
    char c3 = '!';
    printf("%i%i%i\n", c1, c2, c3);
}
```

- Veremos **72 73 33** impressos e perceberemos que esses caracteres são armazenados na memória da seguinte forma:

•

Strings são, na verdade, apenas matrizes de caracteres e definidas não em C, mas pela biblioteca CS50. Se tivéssemos um array chamado **s**, cada caractere pode ser acessado com **s[0]**, **s[1]** e assim por diante.

E acontece que uma string termina com um caractere especial, **'\0'**, ou um byte com todos os bits definidos como 0. Esse caractere é chamado de **caractere nulo** ou NUL. Então, na verdade,

precisamos de quatro bytes para armazenar nossa string com três caracteres: Podemos usar

uma string como uma matriz em nosso programa e imprimir os códigos ASCII, ou valores inteiros, de cada caractere da string:

```
#include <stdio.h>
#include <cs50.h>

int main(void)
{
    string s = "Hi!";
    printf("%i %i %i %i \n", s[0], s[1], s[2], s[3]);
}
```

- E como poderíamos esperar, vemos **72 73 33 0** impressos.
-
- Na verdade, poderíamos tentar acessar **s[4]** e ver algum símbolo inesperado impresso. Com C, nosso código tem a capacidade de acessar ou alterar a memória que de outra forma não deveria, o que é poderoso e perigoso.

Podemos usar um loop para imprimir todos os caracteres em uma string:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string s = get_string("Input: ");
    printf("Saída: ");
    for (int i = 0; s[i] != '\0'; i++)
    {
        printf("%c", s[i]);
    }
    printf("\n");
}
```

Podemos alterar a condição do nosso loop para continuar independentemente do que **i** seja, mas apenas quando **s[i] != '\0'**, ou quando o caractere na posição atual em **s** não for o caractere nulo.

Podemos usar uma função que vem com a biblioteca de **strings** de C, **strlen**, para obter o comprimento da string para nosso loop:

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string("Input: ");
    printf("Saída: ");
```

```

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

```

Temos a oportunidade de aprimorar o design de nosso programa. Nosso loop foi um pouco ineficiente, pois verificamos o comprimento da string, após cada caractere ser impresso, em nossa condição. Mas como o comprimento da string não muda, podemos verificar o comprimento da string uma vez:

```

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string("Input: ");
    printf("Saída:\n");
    for (int i = 0, n = strlen(s); i < n; i++)
    {
        printf("%c\n", s[i]);
    }
}

```

- Agora, no início de nosso loop, inicializamos uma variável **i** e **n** e lembramos o comprimento de nossa string em **n**. Então, podemos verificar os valores sem ter que chamar **strlen** para calcular o comprimento da string a cada vez.
-
- E precisávamos usar um pouco mais de memória para armazenar **n**, mas isso nos economiza algum tempo por não termos que verificar o comprimento da string todas as vezes.

Podemos declarar uma matriz de duas strings:

```

string words[2];
words[0] = "HI!";
words[1] = "BYE!";

```

E na memória, a matriz de strings pode ser armazenada e acessada com:

•

- **words[0]** refere-se ao primeiro elemento, ou valor, da matriz **words**, que é uma string e, portanto, **words[0][0]** se refere ao primeiro elemento dessa string, que é um caractere.
-

- Portanto, um array de strings é apenas um array de arrays de caracteres.

Agora podemos combinar o que vimos para escrever um programa que pode capitalizar letras:

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string("Antes: ");
    printf("Depois: ");
    for (int i = 0, n = strlen(s); i < n; i++)
    {
        if (s[i] >= 'a' && s[i] <= 'z')
        {
            printf("%c", s[i] - 32);
        }
        else
        {
            printf("%c", s[i]);
        }
    }
    printf("\n");
}
```

- Primeiro, obtemos uma string `s` do usuário. Então, para cada caractere na string, se estiver em minúsculas (o que significa que tem um valor entre o de `a` e `z`), nós o convertemos em maiúsculas. Caso contrário, apenas imprimiremos.
-
- Podemos converter uma letra minúscula em seu equivalente maiúsculo subtraindo a diferença entre seus valores ASCII. (Sabemos que as letras minúsculas têm um valor ASCII mais alto do que as letras maiúsculas e a diferença é a mesma entre as mesmas letras, portanto, podemos subtrair para obter uma letra maiúscula de uma letra minúscula.)

Acontece que existe outra biblioteca, `ctype.h`, que podemos usar:

```
#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string("Antes: ");
    printf("Depois: ");
    for (int i = 0, n = strlen(s); i < n; i++)
    {
        if (islower(s[i]))
```

```

        {
            printf("%c", toupper(s[i]));
        }
        else
        {
            printf("%c", s[i]);
        }
    }
    printf("\n");
}

```

- Agora, nosso código está mais legível e provavelmente correto, já que outros escreveram e testaram essas funções para nós.

Podemos simplificar ainda mais, e apenas passar cada caractere para o **toupper**, já que ele não altera os caracteres não minúsculos:

```

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string s = get_string("Antes: ");
    printf("Depois: ");
    for (int i = 0, n = strlen(s); i < n; i++)
    {
        printf("%c", toupper(s[i]));
    }
    printf("\n");
}

```

Podemos usar as [páginas de manual do CS50](#) para encontrar e aprender sobre as funções comuns da biblioteca. Pesquisando nas páginas de manual, vemos que **toupper()** é uma função, entre outras, de uma biblioteca chamada **ctype**, que podemos usar.

Argumentos de linha de comando

Os nossos próprios programas também podem aceitar argumentos de linha de comando ou palavras adicionadas após o nome do nosso programa no próprio comando.

Em `argv.c`, mudamos a aparência de nossa função **principal(main)**:

```

#include <cs50.h>
#include <stdio.h>

int main(int argc, string argv[])

```

```

{
    if (argc == 2)
    {
        printf("oi, %s\n", argv[1]);
    }
    else
    {
        printf("olá, mundo\n");
    }
}

```

- **argc** e **argv** são duas variáveis que nossa função **main** obterá automaticamente quando nosso programa for executado a partir da linha de comando. **argc** é a contagem de argumentos, ou número de argumentos, e **argv**, vetor de argumentos (ou lista de argumentos), uma matriz de strings.
-
- O primeiro argumento, **argv[0]**, é o nome do nosso programa (a primeira palavra digitada, como **./hello**). Neste exemplo, verificamos se temos dois argumentos e imprimimos o segundo se houver.
-
- Por exemplo, se executarmos **./argv David**, receberemos **Oi, David** impresso, já que digitamos **David** como a segunda palavra em nosso comando.

Também podemos imprimir cada caractere individualmente:

```

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(int argc, string argv[])
{
    if (argc == 2)
    {
        for (int i = 0, n = strlen(argv[1]); i < n; i++)
        {
            printf("%c\n", argv[1][i]);
        }
    }
}

```

- Usaremos **argv[1][i]** para acessar cada caractere no primeiro argumento de nosso programa.

Acontece que nossa função **main** também retorna um valor inteiro. Por padrão, nossa função **main** retorna **0** para indicar que nada deu errado, mas podemos escrever um programa para retornar um valor diferente:

```
#include <cs50.h>
#include <stdio.h>

int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("Argumento ausente\n");
        return 1;
    }
    printf("oi, %s\n", argv[1]);
    return 0;
}
```

- O valor de retorno de **main** em nosso programa é chamado de **código de saída**, geralmente usado para indicar códigos de erro. (Vamos escrever o **return 0** explicitamente no final do nosso programa aqui, mesmo que tecnicamente não seja necessário.)

Conforme escrevemos programas mais complexos, códigos de erro como este podem nos ajudar a determinar o que deu errado, mesmo que não seja visível ou significativo para o usuário

Aplicações

Agora que sabemos como trabalhar com strings em nossos programas, bem como códigos escritos por outros em bibliotecas, podemos analisar parágrafos de texto quanto ao seu nível de legibilidade, com base em fatores como o comprimento e a complexidade das palavras e frases.

A **criptografia** é a arte de embaralhar ou ocultar informações. Se quisermos enviar uma mensagem a alguém, podemos **criptografar** ou, de alguma forma, embaralhar essa mensagem para que seja difícil para outras pessoas lerem. A mensagem original, ou input para nosso algoritmo, é chamada de **plaintext** (texto simples), e a mensagem criptografada, ou output, é chamada de **ciphertext** (texto cifrado). E o algoritmo que faz o embaralhamento é chamado de **cifra**. Uma cifra geralmente requer outro input além do texto simples. Uma **chave**, como um número, é um outro input que é mantida em segredo.

Por exemplo, se quisermos enviar uma mensagem como I L O V E Y O U, podemos primeiro convertê-la para ASCII: 73 76 79 86 69 89 79 85. Em seguida, podemos criptografá-la com uma chave de apenas 1 e um algoritmo simples, onde basta adicionar a chave a cada valor: 74 77 80 87 70 90 80 86. Então, o texto cifrado depois de converter os valores de volta para ASCII seria J M P W F Z P V. Para descriptografar isso, alguém teria que saber que a chave é 1, e para subtraí-lo de cada personagem!

EXERCÍCIOS MÓDULO 2

Exercício 0: Legibilidade

Implemente um programa que calcule o nível (representado a partir de uma série, como na escola) aproximado necessário para compreender algum texto, conforme a seguir.

```
$ ./readability
```

```
Texto: Parabéns! Hoje é seu dia. Você está indo para ótimos lugares! Você está com tudo!
```

```
Grade 3
```

Níveis de leitura

De acordo com a Scholastic, “Charlotte's Web” de EB White está entre o nível de leitura da segunda e quarta séries, e “The Giver” de Lois Lowry está entre um nível de leitura da oitava série e um nível de leitura da décima segunda série. No entanto, o que significa um livro estar no “nível de leitura da quarta série”?

Bem, em muitos casos, um especialista humano pode ler um livro e tomar uma decisão sobre a série para a qual acha que o livro é mais apropriado. Mas você também pode imaginar um algoritmo tentando descobrir qual é o nível de leitura de um texto.

Então, que tipo de características são características de níveis de leitura mais altos? Bem, palavras mais longas provavelmente se correlacionam com níveis de leitura mais altos. Da mesma forma, frases mais longas provavelmente se correlacionam com níveis mais altos de leitura também. Vários “testes de legibilidade” foram desenvolvidos ao longo dos anos, para fornecer um processo estereotipado para calcular o nível de leitura de um texto.

Um desses testes de legibilidade é o índice Coleman-Liau. O índice Coleman-Liau de um texto é projetado para mostrar qual nível escolar (EUA) é necessário para entender o texto. A fórmula é:

$$\text{índice} = 0,0588 * L - 0,296 * S - 15,8$$

Aqui, L é o número médio de letras por 100 palavras no texto e S é o número médio de sentenças por 100 palavras no texto.

Vamos escrever um programa chamado **readability** que pega um texto e determina seu nível de leitura. Por exemplo, se o usuário digitar uma linha do Dr. Seuss:

```
$ ./readability
```

```
Texto: Congratulations! Today is your day. You're off to Great Places!  
You're off and away!
```

```
Grade 3
```

O texto que o usuário inseriu tem 65 letras, 4 sentenças e 14 palavras. 65 letras por 14 palavras é uma média de cerca de 464,29 letras por 100 palavras. E 4 sentenças por 14 palavras é uma média de cerca de 28,57 sentenças por 100 palavras. Conectados à fórmula Coleman-Liau e arredondados para o número inteiro mais próximo, obtemos uma resposta de 3: portanto, esta passagem está em um nível de leitura da terceira série.

Vamos tentar outro:

```
$ ./readability
```

```
Text: Harry Potter was a highly unusual boy in many ways. For one thing,  
he hated the summer holidays more than any other time of year. For  
another, he really wanted to do his homework, but was forced to do it in  
secret, in the dead of the night. And he also happened to be a wizard.
```

```
Grade 5
```

Este texto contém 214 letras, 4 frases e 56 palavras. Isso resulta em cerca de 382,14 letras por 100 palavras e 7,14 frases por 100 palavras. Conectados à fórmula Coleman-Liau, chegamos ao nível de leitura da quinta série.

À medida que o número médio de letras e palavras por frase aumenta, o índice de Coleman-Liau dá ao texto um nível de leitura mais alto. Se você pegasse este parágrafo, por exemplo, que tem palavras e sentenças mais longas do que qualquer um dos dois exemplos anteriores, a fórmula daria ao texto um nível de leitura de décimo primeiro ano.

\$./readability

Text: As the average number of letters and words per sentence increases, the Coleman-Liau index gives the text a higher reading level. If you were to take this paragraph, for instance, which has longer words and sentences than either of the prior two examples, the formula would give the text an eleventh grade reading level.

Grade 11

Tente!

Especificação

Projete e implemente um programa, **readability**, que calcule o índice Coleman-Liau do texto.

- Implemente seu programa em um arquivo denominado **readability.c** em um diretório denominado **readability**.
-
- Seu programa deve solicitar ao usuário uma **string** de texto (usando **get_string**).
-
- Seu programa deve contar o número de letras, palavras e frases do texto. Você pode assumir que uma letra é qualquer caractere minúsculo de **a** a **z** ou qualquer caractere maiúsculo de **A** a **Z**, qualquer sequência de caracteres separados por espaços deve contar como uma palavra e que qualquer ocorrência de um ponto final, ponto de exclamação ou ponto de interrogação indica o final de uma frase.
-
- Seu programa deve imprimir como saída "**Grade X**", onde **X** é o nível de grau calculado pela fórmula de Coleman-Liau, arredondado para o número inteiro mais próximo.
-
- Se o número do índice resultante for 16 ou superior (equivalente ou superior ao nível de leitura de graduação sênior), seu programa deve produzir "**Grade 16+**" em vez de fornecer o número do índice exato. Se o número do índice for menor que 1, seu programa deve imprimir "**Before Grade 1**".

Como testar seu código no IDE do CS50?

Aqui alguns casos de teste. Seu código funciona conforme prescrito quando você insere....:

- One fish. Two fish. Red fish. Blue fish. (Before Grade 1)

-
- Would you like them here or there? I would not like them here or there. I would not like them anywhere. (Grade 2)
-
- Congratulations! Today is your day. You're off to Great Places! You're off and away! (Grade 3)
-
- Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard. (Grade 5)
-
- In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since. (Grade 7)
-
- Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice "without pictures or conversation?" (Grade 8)
-
- When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow. When it healed, and Jem's fears of never being able to play football were assuaged, he was seldom self-conscious about his injury. His left arm was somewhat shorter than his right; when he stood or walked, the back of his hand was at right angles to his body, his thumb parallel to his thigh. (Grade 8)
-
- There are more things in Heaven and Earth, Horatio, than are dreamt of in your philosophy. (Grade 9)
-
- It was a bright cold day in April, and the clocks were striking thirteen. Winston Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped quickly through the glass doors of Victory Mansions, though not quickly enough to prevent a swirl of gritty dust from entering along with him. (Grade 10)
-
- A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains. (Grade 16+)

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

`check50 cs50/problems/2021/x/readability`

Execute o seguinte para avaliar o style do seu código usando **style50**.

style50 readability.c

Exercício 1: Caesar

Implemente um programa que criptografa mensagens usando a cifra de César, conforme a seguir.

```
$ ./caesar 13
plaintext: HELLO
ciphertext: URYYB
```

Lembre-se que plaintext significa *texto simples* e ciphertext é o *texto cifrado*!

Background

Supostamente, César (sim, aquele César) costumava “criptografar” (ou seja, ocultar de forma reversível) mensagens confidenciais deslocando cada letra nelas em algum número de lugares. Por exemplo, ele pode escrever A como B, B como C, C como D, ..., e, agrupando em ordem alfabética, Z como A. E então, para dizer HELLO para alguém, César pode escrever IFMMP. Ao receber essas mensagens de César, os destinatários teriam que “decifrá-las” deslocando as letras na direção oposta no mesmo número de lugares.

O segredo desse “criptosistema” dependia apenas de César e dos destinatários saberem de um segredo, o número de lugares pelos quais César havia mudado suas cartas (por exemplo, 1). Não é particularmente seguro para os padrões modernos, mas, ei, se você é talvez o primeiro no mundo a fazer isso, bastante seguro!

O texto não criptografado é geralmente chamado de texto simples . O texto criptografado é geralmente chamado de texto cifrado . E o segredo usado é chamado de chave .

Para ser claro, então, aqui está como criptografar **HELLO** com uma chave de 1 resulta em **IFMMP** :

plaintext (texto simples)	H	E	L	L	O
+ key(chave)	1	1	1	1	1
= ciphertext (texto cifrado)	I	F	M	M	P

Mais formalmente, o algoritmo de César (isto é, cifra) criptografa mensagens “girando” cada letra em k posições. Mais formalmente, se p é algum texto simples (ou seja, uma mensagem não criptografada), p_i é o i -ésimo caractere em p , e k é uma chave secreta (ou seja, um inteiro não negativo), então cada letra, c_i , em o texto cifrado, c , é calculado como

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

em que **%26** aqui significa "resto ao dividir por 26." Essa fórmula talvez faça a cifra parecer mais complicada do que 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) como 0, B (ou b) como 1, ..., H (ou h) como 7, I (ou i) como 8, ... e Z (ou z) como 25. Suponha que César apenas queira dizer Oi para alguém confidencialmente usando, desta vez, uma chave, k , de 3. E então seu texto simples, p , é H_i ,

em cujo caso o primeiro caractere de seu texto simples, p_0 , é H (também conhecido como 7), e o segundo caractere de seu texto simples, p_1 , é i (também conhecido como 8). O primeiro caractere de seu texto cifrado, c_0 , é portanto K, e o segundo caractere de seu texto cifrado, c_1 , é assim L. Você pode ver por quê?

Vamos escrever um programa chamado **caesar** 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 que fornecerá no tempo de execução. Não devemos necessariamente presumir que a chave do usuário será um número; embora você possa assumir que, se for um número, será um inteiro positivo.

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

```
$ ./caesar 1
plaintext:  HELLO
ciphertext: IFMMP
```

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

```
$ ./caesar 13
plaintext:  hello, world
ciphertext: uryyb, jbeyq
```

Observe que nem a vírgula nem o espaço foram "deslocados" pela cifra. Gire apenas caracteres alfabéticos!

Que tal mais um? Veja como o programa pode funcionar se o usuário fornecer uma chave **13** novamente, com um texto simples mais complexo:

```
$ ./caesar 13
plaintext:  be sure to drink your Ovaltine
ciphertext: or fher gb qevax lbhe Binygvar
```

Observe que a caixa da mensagem original foi preservada. As letras minúsculas permanecem minúsculas e as maiúsculas permanecem maiúsculas.

E se um usuário não cooperar?

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

Ou realmente não coopera?

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

Ou mesmo ...

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

Especificação

Projete e implemente um programa, o **caesar**, que criptografa mensagens usando a cifra de César.

- Implemente seu programa em um arquivo denominado **caesar.c** em um diretório denominado **caesar**.
-
- Seu programa deve aceitar um único argumento de linha de comando, um inteiro não negativo. Vamos chamá-lo de **k** para 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, seu programa deve imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar de **main** um valor de **1** (o que tende a significar um erro) imediatamente.
-
- Se algum dos caracteres do argumento da linha de comando não for um dígito decimal, seu programa deve imprimir a mensagem Use: ./caesar key e retornar de **main** o valor **1**.
-
- Não suponha que **k** será menor ou igual a 26. Seu programa deve funcionar para todos os valores integrais não negativos de **k** menores que $2^{31} - 26$. Em outras palavras, você não precisa se preocupar se seu programa eventualmente quebra se o usuário escolher um valor para **k** que é muito grande ou quase grande para caber em um **int**. (Lembre-se de que um **int** pode estourar.) Mas, mesmo se **k** for maior que 26, os caracteres alfabéticos na entrada do programa devem permanecer caracteres alfabéticos na saída do programa. Por exemplo, se **k** é 27, **A** não deve se tornar [embora [esteja a 27 posições de **A** em ASCII, por [http://www.asciichart.com/\[asciichart.com\]](http://www.asciichart.com/[asciichart.com]); **A** deve tornar-se **B**, já que **B** esta a 27 posições de **A**, desde que você revolva em torno de **Z** a **A**.
-
- Seu programa deve produzir **plaintext**: (sem uma nova linha) [texto simples] e então solicitar ao usuário uma **string** de texto simples (usando **get_string**).
-
- Seu programa deve produzir **ciphertext**: (sem uma nova linha) [texto cifrado] seguido pelo texto cifrado correspondente do texto simples, com cada caractere alfabético no texto simples “girado” por **k** posições; os caracteres não alfabéticos devem ser reproduzidos inalterados.
-
- Seu programa deve preservar as letras maiúsculas e minúsculas: as letras maiúsculas, embora giradas, devem permanecer letras maiúsculas; as letras minúsculas, embora giradas, devem permanecer em minúsculas.
-
- Após a saída do texto cifrado, você deve imprimir uma nova linha. Seu programa deve então sair retornando **0** de **main**.

Como começar? Vamos abordar esse problema um passo de cada vez.

Pseudocódigo

Spoiler ;)

Existe mais de uma forma para resolver esse exercício, esse spoiler aqui é apenas uma delas!

- 1- Verifique se o programa foi executado com um argumento de linha de comando;
- 2- Repita o argumento fornecido para garantir que todos os caracteres sejam dígitos;

3- Converta o argumento da linha de comando de uma string para int

4- Solicite `plaintext`: (texto simples) ao usuário

5- Repita/Itere para cada caractere do `plaintext`: (texto simples)

1- Se é uma letra maiúscula, rotacione-a, preservando capitalização, e então imprima o caractere rotacionado.

2- Se é uma letra minúscula, rotacione-a, preservando capitalização, e então imprima o caractere rotacionado.

3- Se não for nenhum, imprima o caractere como está

6- Imprimir uma nova linha

Não há problema em editar o seu depois de ver este pseudocódigo aqui, mas não simplesmente copie/cole o nosso no seu. Ok?

Como testar seu código no IDE do CS50?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/caesar
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 caesar.c
```

Exercício 2: Substituição (desafio)

Implemente um programa que implemente uma cifra de substituição, conforme a seguir.

```
$ ./substitution JTREKYAVOGDXPSNCUIZLFBMWHQ
texto simples: HELLO
texto cifrado: VKXXN
```

Background

Em uma cifra de substituição, “criptografamos” (ou seja, ocultamos de forma reversível) uma mensagem substituindo cada letra por outra. Para isso, usamos uma chave : neste caso, um mapeamento de cada uma das letras do alfabeto à letra correspondente quando criptografada. Para “decifrar” a mensagem, o receptor da mensagem precisaria saber a chave, para que pudesse reverter o processo: traduzir o texto criptografado (geralmente chamado de texto cifrado) de volta na mensagem original (geralmente chamado de texto simples).

Uma chave, por exemplo, pode ser a sequência NQXPOMAFTRHLZGECYJIUWSKDVB. Esta chave de 26 caracteres significa que **A** (a primeira letra do alfabeto) deve ser convertido em **N** (o primeiro caractere da chave), **B** (a segunda letra do alfabeto) deve ser convertido em **Q** (o segundo caractere do chave), e assim por diante.

Uma mensagem como **HELLO** , então, seria criptografada como **FOLLE** , substituindo cada uma das letras de acordo com o mapeamento determinado pela chave.

Vamos escrever um programa chamado **substitution** que permite criptografar mensagens usando uma cifra de substituição. 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 que fornecerá durante a execução.

Aqui estão alguns exemplos de como o programa pode funcionar. Por exemplo, se o usuário inserir uma chave de **YTNSHKVEFXRBAUQZCLWDMIPGJO** e um texto simples de **HELLO** :

```
$ ./substitution YTNSHKVEFXRBAUQZCLWDMIPGJO
texto simples: HELLO
texto cifrado: EHBBQ
```

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

```
$ ./substitution VCHPRZGJNTLSKFBDQWAXEUYSOI
texto simples: hello, world
texto cifrado: jrssb, ybwsp
```

Observe que nem a vírgula nem o espaço foram substituídos pela cifra. Substitua apenas caracteres alfabéticos! Observe também que o case da mensagem original foi preservado. As letras minúsculas permanecem minúsculas e as maiúsculas permanecem maiúsculas.

Não importa se os caracteres da própria chave são maiúsculas ou minúsculas. Uma chave de **VCHPRZGJNTLSKFBDQWAXEUYSOI** é funcionalmente idêntica a uma chave de **vchprzgjntlskfbdqwaxeuysoi** (como é, nesse caso, **VcHpRzGjNtLsKfBdQwAxEuYmOi**).

E se um usuário não fornecer uma chave válida?

```
$ ./substitution ABC
A chave deve conter 26 caracteres.
```

Ou realmente não coopera?

```
$ ./substitution
Uso: ./ chave de substituição
```

Ou mesmo ...

```
$ ./substituição 1 2 3
Uso: ./ chave de substituição
```

Especificação

Projete e implemente um programa, **substitution**, que criptografa mensagens usando uma cifra de substituição.

- Implemente seu programa em um arquivo denominado **substitution.c** em um diretório denominado **substitution** .
-
- Seu programa deve aceitar um único argumento de linha de comando, a chave a ser usada para a substituição. A chave em si não deve fazer distinção entre maiúsculas e minúsculas, portanto, se algum caractere na chave estiver em maiúscula ou minúscula, isso não deve afetar o comportamento do seu programa.

-
- Se o seu programa for executado sem nenhum argumento de linha de comando ou com mais de um argumento de linha de comando, seu programa deve imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar de **main** um valor de **1** (o que tende a significar um erro) imediatamente.
-
- Se a chave for inválida (por não conter 26 caracteres, conter qualquer caractere que não seja um caractere alfabético, ou não conter cada letra exatamente uma vez), seu programa deverá imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar do **main** um valor de **1** imediatamente.
-
- Seu programa deve produzir na saída **plaintext**: - que significa texto simples: (sem uma nova linha) e então solicitar ao usuário uma string de texto simples (usando `get_string`).
-
- Seu programa ter na saída o seguinte texto **ciphertext**: - que significa texto cifrado - (sem uma nova linha) seguido pelo texto cifrado correspondente do texto simples, com cada caractere alfabético no texto simples substituído pelo caractere correspondente no texto cifrado; os caracteres não alfabéticos devem ser reproduzidos inalterados.
-
- Seu programa deve preservar maiúsculas e minúsculas: as letras maiúsculas devem permanecer letras maiúsculas; as letras minúsculas devem permanecer em minúsculas.
-
- Após a saída do texto cifrado, você deve imprimir uma nova linha. Seu programa deve então sair retornando **0** da **main**.

Como testar seu código no IDE do CS50?

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/substitution
```

Execute o seguinte para avaliar o style do seu código usando **style50**.

```
style50 substitution.c
```

Lab 2: Scrabble

Determine qual das duas palavras do Scrabble vale mais.

```
$ ./scrabble
Player 1: COMPUTER
Player 2: science
Player 1 wins!
```

Background

No jogo [Scrabble](#), os jogadores criam palavras para marcar pontos, e o número de pontos é a soma dos valores dos pontos de cada letra da palavra.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	3	3	2	1	4	2	4	1	8	5	1	3	1	1	3	10	1	1	1	1	4	4	8	4	10
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	----

Por exemplo, se quiséssemos pontuar a palavra **Code**, observaríamos que, nas regras gerais do Scrabble, o **C** vale 3 pontos, o **o** vale 1 ponto, o **d** vale 2 pontos e **e** vale 1 ponto. Somando isso, obtemos que **Code** vale $3 + 1 + 2 + 1 = 7$ pontos.

Começando

Copie o “código de distribuição” (ou seja, código inicial) a seguir em um novo arquivo em seu IDE chamado **scrabble.c**.

```
#include <ctype.h>
#include <cs50.h>
#include <stdio.h>
#include <string.h>

// Points assigned to each letter of the alphabet
int POINTS[] = {1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10};

int compute_score(string word);

int main(void)
{
    // Get input words from both players
    string word1 = get_string("Player 1: ");
    string word2 = get_string("Player 2: ");

    // Score both words
    int score1 = compute_score(word1);
    int score2 = compute_score(word2);

    // TODO: Print the winner
}

int compute_score(string word)
{
    // TODO: Compute and return score for string
}
```

- Você também pode fazer download do código de distribuição executando o comando **wget** <https://cdn.cs50.net/2020/fall/labs/2/scrabble.c> no CS50 IDE.

Detalhes de Implementação

Conclua a implementação de **scrabble.c**, de modo que determine o vencedor de um jogo curto do tipo scrabble, em que dois jogadores digitam suas palavras, e o jogador com maior pontuação vence.

Observe que armazenamos os valores dos pontos de cada letra do alfabeto em uma matriz de inteiros chamada **POINTS**.

- Por exemplo, **A** ou **a** vale **1** ponto (representado por **POINTS[0]**), **B** ou **b** vale **3** pontos (representado por **POINTS[1]**), etc.

Observe que criamos um protótipo para uma função auxiliar chamada **compute_score()** que recebe uma string como input e retorna um **int**. Sempre que quisermos atribuir valores de pontos a uma palavra específica, podemos chamar essa função. Observe que este protótipo é necessário para o C saber que **compute_score()** existe posteriormente no programa.

Em **main()**, o programa pede aos dois jogadores suas palavras usando a função **get_string()**. Esses valores são armazenados dentro de variáveis chamadas **word1** e **word2**.

Em **compute_score()**, seu programa deve calcular, usando o array **POINTS**, e retornar a pontuação para o argumento string. Os caracteres que não são letras devem receber zero pontos, e as letras maiúsculas e minúsculas devem receber os mesmos valores de pontos.

- Por exemplo, **!** vale **0** pontos, enquanto **A** e **a** valem **1** ponto.
-
- Embora as regras do Scrabble normalmente exijam que uma palavra esteja no dicionário, não há necessidade de verificar isso neste problema!

Em **main()**, seu programa deve imprimir, dependendo da pontuação dos jogadores, **Player 1 wins!**, **Player 2 wins!**, ou **Tie!**.

Dicas

Você pode achar as funções **isupper()** e **islower()** úteis para você. Essas funções usam um caractere como argumento e retornam um valor diferente de zero se o caractere for maiúsculo (para **isupper**) ou minúsculo (para **islower**).

Para encontrar o valor no *n*-ésimo índice de um array chamado **arr**, podemos escrever **arr[n]**. Podemos aplicar isso a strings também, já que strings são arrays de caracteres.

Lembre-se de que os computadores representam caracteres usando [ASCII](#), um padrão que representa cada caractere como um número.

Como testar seu código

Seu código deve resolver os seguintes casos de teste:

- ```
$./scrabble
Player 1: Question?
Player 2: Question!
Tie!
```
- 
- ```
$ ./scrabble
Player 1: Oh,
Player 2: hai!
Player 2 wins!
```
-

- `$./scrabble`
Player 1: COMPUTER
Player 2: science
Player 1 wins!
-
- `$./scrabble`
Player 1: Scrabble
Player 2: wiNNeR
Player 1 wins!

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/scrabble
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 scrabble.c
```

ANOTAÇÕES MÓDULO 3

Módulo Anterior

Aprendemos sobre ferramentas para resolver problemas, ou bugs, em nosso código. Em particular, descobrimos como usar um depurador, uma ferramenta que nos permite percorrer lentamente nosso código e examinar os valores na memória enquanto nosso programa está em execução.

Outra ferramenta poderosa, embora menos técnica, é a duck debugging (“depuração de pato de borracha”), onde tentamos explicar o que estamos tentando fazer com um pato de borracha (ou algum outro objeto) e, no processo, encontramos o problema (e esperamos, a solução!) sozinhos.

Olhamos para a memória, visualizando bytes em uma grade e armazenando valores em cada caixa, ou byte, com variáveis e matrizes.

Busca/Searching

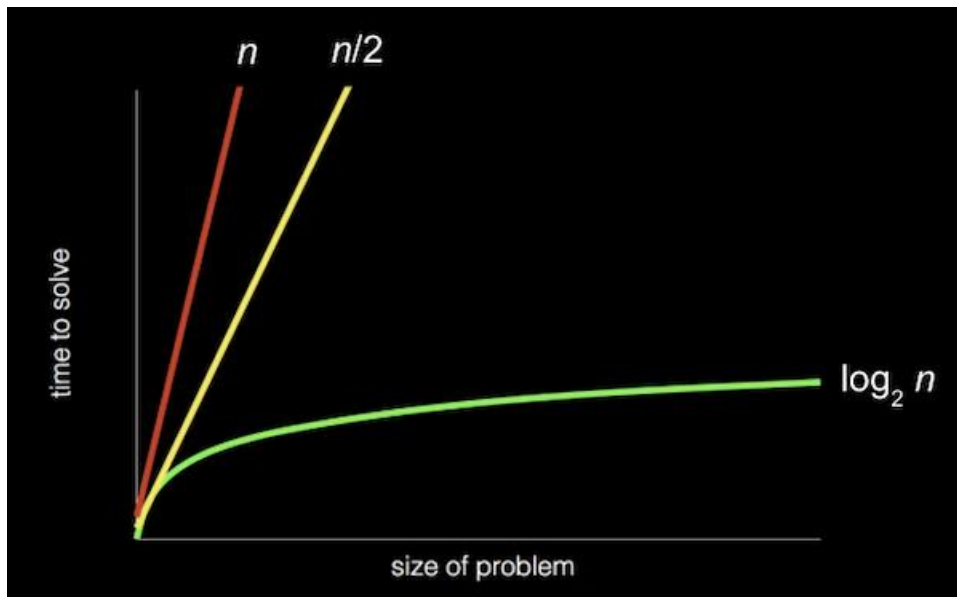
Acontece que, com matrizes, um computador não pode olhar para todos os elementos de uma vez. Em vez disso, um computador só pode olhar para eles um de cada vez, embora a ordem possa ser arbitrária. (Lembre-se de que, na semana 0, David só conseguia olhar uma página de cada vez na lista telefônica, quer folheasse em ordem ou de maneira mais sofisticada.)

Searching (“busca”) é como resolvemos o problema de encontrar um valor específico. Um caso simples pode ter como input algum array de valores, e a saída pode ser simplesmente um bool, esteja ou não um determinado valor no array.

Hoje veremos algoritmos de pesquisa. Para discuti-los, consideraremos o **tempo de execução**, ou quanto tempo um algoritmo leva para ser executado dado algum tamanho de input.

Big O

Na semana 0, vimos diferentes tipos de algoritmos e seus tempos de execução:



Lembre-se de que a linha vermelha está pesquisando linearmente, uma página por vez; a linha amarela está pesquisando duas páginas por vez; e a linha verde está pesquisando logaritmicamente, dividindo o problema pela metade a cada vez.

E esses tempos de execução são para o pior caso, ou o caso em que o valor leva mais tempo para ser encontrado (na última página, ao contrário da primeira página).

A maneira mais formal de descrever cada um desses tempos de execução é com a notação **Big O** (“grande O”), que podemos pensar como “na ordem de”. Por exemplo, se nosso algoritmo for uma pesquisa linear, ele levará aproximadamente $O(n)$ etapas, lidas como “grande O de n ” ou “na ordem de n ”. Na verdade, mesmo um algoritmo que analisa dois itens por vez e executa $n/2$ etapas tem $O(n)$. Isso ocorre porque, à medida que n fica cada vez maior, apenas o fator dominante, ou o maior termo, n , importa. No gráfico acima, se afastássemos o zoom e mudássemos as unidades em nossos eixos, veríamos as linhas vermelha e amarela ficando muito próximas.

Um tempo de execução logarítmico é $O(\log n)$, não importa qual seja a base, pois isso é apenas uma aproximação do que acontece fundamentalmente com o tempo de execução se n for muito grande.

Existem alguns tempos de execução comuns:

- $O(n^2)$
-
- $O(n \log n)$
-
- $O(n)$
 -
 - (pesquisando uma página por vez, em ordem)
 -
-
-
- $O(\log n)$

-
- (dividindo a lista telefônica pela metade a cada vez)
-
-
-
- $O(1)$
 -
 - Um algoritmo que executa um número **constante** de etapas, independentemente do tamanho do problema.
 -
-

Os cientistas da computação também podem usar a notação *big Ω* , *grande Omega*, que é o limite inferior do número de etapas de nosso algoritmo. *Big O* é o limite superior do número de etapas ou o pior caso.

E temos um conjunto semelhante de tempos de execução mais comuns para big Ω :

- $\Omega(n^2)$
-
- $\Omega(n \log n)$
-
- $\Omega(n)$
-
- $\Omega(\log n)$
-
- $\Omega(1)$
 -
 - (pesquisar em uma lista telefônica, pois podemos encontrar nosso nome na primeira página que verificarmos)
 -
-

Pesquisa linear, pesquisa binária

No palco, temos algumas portas de mentira, com números escondidos atrás delas. Como um computador só pode olhar para um elemento de cada vez em um array, só podemos abrir uma porta de cada vez.

Se quisermos procurar o número zero, por exemplo, teríamos que abrir uma porta por vez, e se não soubéssemos nada sobre os números atrás das portas, o algoritmo mais simples seria ir da esquerda para a direita.

Então, podemos escrever pseudocódigo para **busca linear** com:

```
For i from 0 to n-1
  If number behind i'th door
```

```
        Return true
Return false
```

- Rotulamos cada uma das n portas de 0 a $n - 1$ e verificamos cada uma delas em ordem.
-
- “Return false” está fora do for loop, já que só queremos fazer isso depois de olharmos por trás de todas as portas.
-
- O **big O** para este algoritmo seria $O(n)$, e o limite inferior, **big Ω** , seria $\Omega(1)$.

Se soubermos que os números atrás das portas estão classificados, podemos começar pelo meio e encontrar nosso valor com mais eficiência.

Para busca binária, nosso algoritmo pode ser semelhante a:

```
If no doors
    Return false
If number behind middle door
    Return true
Else if number < middle door
    Search left half
Else if number > middle door
    Search right half
```

- O limite superior da pesquisa binária é $O(\log n)$, e o limite inferior também $\Omega(1)$, se o número que procuramos estiver no meio, onde começamos.

Com 64 lâmpadas, notamos que a pesquisa linear leva muito mais tempo do que a pesquisa binária, que leva apenas alguns passos.

Desligamos as lâmpadas na frequência de um **hertz**, ou ciclo por segundo, e a velocidade de um processador pode ser medida em gigahertz, ou bilhões de operações por segundo.

Realizando a busca em código

Vamos dar uma olhada em numbers.c :

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    int numbers[] = {4, 6, 8, 2, 7, 5, 0};
    for (int i = 0; i < 7; i++)
    {
        if (numbers[i] == 0)
        {
            printf("Found\n");
            return 0;
        }
    }
}
```

```

    }
}
printf("Not found\n");
return 1;
}

```

- Aqui, inicializamos uma matriz com alguns valores entre chaves e verificamos os itens na matriz, um de cada vez, para ver se eles são iguais a zero (o que estávamos originalmente procurando atrás das portas no palco).
-
- Se encontrarmos o valor zero, retornamos um código de saída 0 (para indicar sucesso). Caso contrário, após nosso loop for, retornamos 1 (para indicar falha).

Podemos fazer o mesmo com os nomes:

```

• #include • #include • #include • • int main(void) • { • string names[] =
{"Bill", "Charlie", "Fred", "George", "Ginny", "Percy", "Ron"}; • • for
(int i = 0; i < 7; i++) • { • if (strcmp(names[i], "Ron") == 0) • { •
printf("Found\n"); • return 0; • } • } • printf("Not found\n"); • return
1; • }

```

- Observe que os **names** é uma matriz ordenada de strings.
-
- Não podemos comparar strings diretamente em C, uma vez que não são um tipo de dados simples, mas sim um array de muitos caracteres. Felizmente, a biblioteca de **string** tem uma função **strcmp** ("string compare") que compara strings para nós, um caractere por vez, e retorna 0 se forem iguais.
-
- Se checarmos apenas `strcmp (nomes [i], "Ron")` e não `strcmp (nomes [i], "Ron") == 0`, então imprimiremos Encontrado mesmo se o nome não for encontrado. Isso ocorre porque `strcmp` retorna um valor que não é 0 se duas strings não corresponderem, e qualquer valor diferente de zero é equivalente a verdadeiro em uma condição.

Structs

Se quisermos implementar um programa que pesquisa uma lista telefônica, podemos querer um tipo de dado para uma "pessoa", com seu nome e número de telefone.

Acontece que em C podemos definir nosso próprio tipo de dados, ou *data structure* ("estrutura de dados"), com um **struct** na seguinte sintaxe:

```

typedef struct
{
    string name;
    string number;
}
person;

```

- Usamos **string** para o **number**, pois queremos incluir símbolos e formatação, como sinais de mais ou hifens.
-
- Nossa estrutura contém outros tipos de dados dentro dela.

Vamos tentar implementar nossa lista telefônica sem structs primeiro:

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string names[] = {"Brian", "David"};
    string numbers[] = {"+1-617-495-1000", "+1-949-468-2750"};
    for (int i = 0; i < 2; i++)
    {
        if (strcmp(names[i], "David") == 0)
        {
            printf("Encontrado %s\n", numbers[i]);
            return 0;
        }
    }
    printf("Nao encontrado\n");
    return 1;
}
```

- Precisamos ter cuidado para garantir que **firstname** nos **names** corresponda ao primeiro número em **numbers** e assim por diante.
-
- Se o nome em um determinado índice **i** na matriz de **names** corresponder ao que estamos procurando, podemos retornar o número de telefone em **numbers** no mesmo índice.

Com structs, podemos ter um pouco mais de certeza de que não teremos erros humanos em nosso programa:

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
```

```
typedef struct
{
    string name;
    string number;
}
person;
```

```
int main(void)
```

```

{
    person people[2];
    people[0].name = "Brian";
    people[0].number = "+1-617-495-1000";
    people[1].name = "David";
    people[1].number = "+1-949-468-2750";

    for (int i = 0; i < 2; i++)
    {
        if (strcmp(people[i].name, "David") == 0)
        {
            printf("Encontrado %s\n", people[i].number); return 0;
        }
    }
    printf("Não encontrado\n");
    return 1;
}

```

- Criamos uma matriz do tipo struct **person** e a damos o nome **people** (como em `int numbers[]`, embora pudéssemos nomeá-la arbitrariamente, como qualquer outra variável). Definimos os valores para cada campo, ou variável, dentro de cada estrutura de **person**, usando o operador ponto, ..
-
- Em nosso loop, agora podemos ter mais certeza de que o **number(número)** corresponde ao **name(nome)**, pois eles são da mesma estrutura de **person**.
-
- Também podemos melhorar o design do nosso programa com uma constante, como `const int NUMBER = 10;`, e armazenar nossos valores não em nosso código, mas em um arquivo separado ou mesmo em um banco de dados, que veremos em breve.

Em breve, também escreveremos nossos próprios arquivos de cabeçalho com definições para structs, para que possam ser compartilhados entre diferentes arquivos para nosso programa.

Ordenação

Se nossa entrada for uma lista não ordenada de números, existem muitos algoritmos que podemos usar para produzir um output de uma lista classificada, onde todos os elementos estão em ordem.

Com uma lista classificada, podemos usar a pesquisa binária para eficiência, mas pode levar mais tempo para escrever um algoritmo de classificação para essa eficiência, então às vezes encontraremos a compensação de tempo que leva para um ser humano escrever um programa em comparação com o tempo é preciso um computador para executar algum algoritmo. Outras compensações que veremos podem ser tempo e complexidade ou uso de tempo e memória.

Ordenação de seleção

Brian está nos bastidores com um conjunto de números em uma prateleira, em ordem não classificada:

6 3 8 5 2 7 4 1

Pegando alguns números e colocando-os no lugar certo, Brian os classifica rapidamente.

Indo passo a passo, Brian olha cada número da lista, lembrando-se do menor que vimos até agora. Ele chega ao final e vê que 1 é o menor, e ele sabe que deve ir no início, então ele vai apenas trocá-lo pelo número do início, 6:

```
6 3 8 5 2 7 4 1
-             -
1 3 8 5 2 7 4 6
```

Agora, Brian sabe que pelo menos o primeiro número está no lugar certo, então ele pode procurar o menor número entre os demais e trocá-lo pelo próximo número não ordenado (agora o segundo número):

```
1 3 8 5 2 7 4 6
-   -
1 2 8 5 3 7 4 6
```

E ele repete isso, trocando o próximo menor, 3, pelo 8:

```
1 2 8 5 3 7 4 6
-   -
1 2 3 5 8 7 4 6
```

Depois de mais algumas trocas, terminamos com uma lista ordenada.

Esse algoritmo é chamado de **selection sort**, e podemos ser um pouco mais específicos com alguns pseudocódigos:

```
Para i de 0 a n-1
  Encontre o menor item entre i-ésimo item e último
  Troque o menor item com i-ésimo item
```

- A primeira etapa do loop é procurar o menor item na parte não ordenada da lista, que estará entre o i-ésimo item e o último item, pois sabemos que classificamos até o "i-1º" item.
-
- Em seguida, trocamos o menor item pelo i-ésimo item, o que compõe tudo até o item que eu classifiquei.

Vemos uma [visualização online](#) com animações de como os elementos se movem durante um selection sort.

Para esse algoritmo, examinamos quase todos os n elementos para encontrar o menor e fizemos n passos para classificar todos os elementos.

Mais formalmente, podemos usar algumas fórmulas matemáticas para mostrar que o maior fator é de fato n^2 . Começamos tendo que olhar para todos os n elementos, então apenas $n - 1$, então $n - 2$:

$$n + (n - 1) + (n - 2) + \dots + 1$$

$$\begin{aligned}
 &n(n+1)/2 \\
 &(n^2 + n)/2 \\
 &n^2/2 + n/2 \\
 &O(n^2)
 \end{aligned}$$

- Como n^2 é o maior, ou dominante, fator, podemos dizer que o algoritmo tem um tempo de execução de $O(n^2)$.

Bubble sort

Podemos tentar um algoritmo diferente, em que trocamos pares de números repetidamente, chamado de **bubble sort**.

Brian vai olhar para os dois primeiros números e trocá-los para que fiquem em ordem:

```

6 3 8 5 2 7 4 1
- -
3 6 8 5 2 7 4 1

```

O próximo par, **6** e **8**, está em ordem, então não precisamos trocá-los.

O próximo par, **8** e **5**, precisa ser trocado:

```

3 6 8 5 2 7 4 1
- -
3 6 5 8 2 7 4 1

```

Brian continua até chegar ao final da lista:

```

3 6 5 2 8 7 4 1
- -
3 6 5 2 7 8 4 1
- -
3 6 5 2 7 4 8 1
- -
3 6 5 2 7 4 1 8
-

```

Nossa lista ainda não foi ordenada, mas estamos um pouco mais próximos da solução porque o maior valor, **8**, foi deslocado totalmente para a direita. E outros números maiores também se moveram para a direita, ou “bubbled up”.

Brian fará outra passagem pela lista:

```

3 6 5 2 7 4 1 8
- -
3 6 5 2 7 4 1 8
- -
3 5 6 2 7 4 1 8
- -
3 5 2 6 7 4 1 8
- -

```



```

3 5 2 6 7 4 1 8
      - -
3 5 2 6 4 7 1 8
      - -
3 5 2 6 4 1 7 8
      - -

```

- Observe que não precisamos trocar o 3 e o 6, ou o 6 e o 7.
-
- Mas agora, o próximo maior valor, **7**, mudou totalmente para a direita.

Brian repetirá esse processo mais algumas vezes e cada vez mais a lista será ordenada, até que tenhamos uma lista totalmente ordenada.

Com selection sort, o melhor caso com uma lista ordenada ainda levaria tantos passos quanto o pior caso, uma vez que verificamos apenas o menor número em cada passagem.

O pseudocódigo para bubble sort pode ser parecido com:

Repita até estar ordenado

Para i de 0 a $n-2$

Se $i^{\text{º}}$ ésimo and $i+1^{\text{º}}$ ésimo elemento fora de ordem

Troque eles

- Uma vez que estamos comparando o $i^{\text{º}}$ ésimo e $i + 1^{\text{º}}$ ésimo elemento, só precisamos ir até $n - 2$ para i . Em seguida, trocamos os dois elementos se eles estiverem fora de ordem.
-
- E podemos parar assim que a lista for ordenada, já que podemos apenas lembrar se fizemos alguma troca. Caso não tenhamos feito, a lista já deve estar ordenada.

Para determinar o tempo de execução para classificação por bolha, temos $n - 1$ comparações no loop e no máximo $n - 1$ loops, portanto, obtemos $n^2 - 2n + 2$ etapas no total. Mas o maior fator, ou termo dominante, é novamente n^2 à medida que n fica cada vez maior, então podemos dizer que a classificação por bolha tem $O(n^2)$. Portanto, basicamente, a classificação por seleção e a classificação por bolha têm o mesmo limite superior para o tempo de execução.

O limite inferior para o tempo de execução aqui seria $\Omega(n)$, uma vez que examinamos todos os elementos uma vez.

Portanto, nossos limites superiores para o tempo de execução que vimos são:

- $O(n^2)$
 -
 - selection sort, bubble sort
 -
-
-
- $O(n \log n)$

-
- $O(n)$
 -
 - busca linear
 -
-
-
- $O(\log_{10} n)$
 -
 - busca binária
 -
-
-
- $O(1)$

E para limites inferiores:

- $\Omega(n^2)$
 -
 - selection sort
 -
-
-
- $\Omega(n \log_{10} n)$
-
- $\Omega(n)$
 -
 - bubble sort
 -
-
-
- $\Omega(\log_{10} n)$
-
- $\Omega(1)$
 -
 - pesquisa linear, pesquisa binária
 -
-

Recursão

Recursão é a capacidade de uma função chamar a si mesma. Ainda não vimos isso no código, mas vimos algo em pseudocódigo na semana 0 que podemos converter:

- 1 Pegue a lista telefônica
- 2 Abra no meio da lista telefônica
- 3 Olhe para a página

```
4 Se Smith estiver na página
5   Ligue para Mike
6 Caso contrário, se Smith estiver no início do livro
7   Aberto no meio da metade esquerda do livro
8   Volte para a linha 3
9 Caso contrário, se Smith estiver mais tarde no livro
10  Aberto no meio da metade direita do livro
11  Volte para a linha 3
12 Senão
13  Desistir
```

- Aqui, estamos usando uma instrução semelhante a um loop para voltar a uma linha específica.

Em vez disso, poderíamos apenas repetir todo o nosso algoritmo na metade do livro que restou:

```
1 Pegue a lista telefônica
2 Abra no meio da lista telefônica
3 Olhe para a página
4 Se Smith estiver na página
5   Ligue para Mike
6 Caso contrário, se Smith estiver no início do livro
7   Pesquise na metade esquerda do livro
8
9 Caso contrário, se Smith estiver mais tarde no livro
10  Pesquise a metade direita do livro
11
12 Senão
13  Desistir
```

- Parece um processo cíclico que nunca termina, mas na verdade estamos mudando a entrada para a função e dividindo o problema pela metade a cada vez, parando assim que não houver mais livro sobrando.

Na semana 1, também implementamos uma "pirâmide" de blocos na seguinte forma:

```
#
##
###
####
```

Mas observe que uma pirâmide de altura 4 é na verdade uma pirâmide de altura 3, com uma linha extra de 4 blocos adicionados. E uma pirâmide de altura 3 é uma pirâmide de altura 2, com uma linha extra de 3 blocos. Uma pirâmide de altura 2 é uma pirâmide de altura 1, com uma linha extra de 2 blocos. E, finalmente, uma pirâmide de altura 1 é apenas um único bloco.

Com essa ideia em mente, podemos escrever uma função recursiva para desenhar uma pirâmide, uma função que se chama para desenhar uma pirâmide menor antes de adicionar outra linha.

Merge sort

Podemos levar a ideia de recursão para classificação, com outro algoritmo chamado merge sort . O pseudocódigo pode ser semelhante a:

```
Se apenas um número
  Retornar
Senão
  Ordenar a metade esquerda do número
  Ordenar a metade direita do número
  Mesclar metades classificadas
```

Veremos isso melhor na prática com duas listas classificadas:

3 5 6 8 | 1 2 4 7

Vamos mesclar as duas listas para uma lista final classificada, pegando o menor elemento na frente de cada lista, um de cada vez:

3 5 6 8 | _ 2 4 7
1

O 1 no lado direito é o menor entre 1 e 3, então podemos começar nossa lista ordenada com ele.

3 5 6 8 | _ _ 4 7
1 2

O próximo menor número, entre 2 e 3, é 2, então usamos o 2.

_ 5 6 8 | _ _ 4 7
1 2 3

_ 5 6 8 | _ _ _ 7
1 2 3 4

_ _ 6 8 | _ _ _ 7
1 2 3 4 5

_ _ _ 8 | _ _ _ 7
1 2 3 4 5 6

_ _ _ 8 | _ _ _ _
1 2 3 4 5 6 7

_ _ _ _ | _ _ _ _
1 2 3 4 5 6 7 8

- Agora temos uma lista completamente ordenada.

Vimos como a linha final em nosso pseudocódigo pode ser implementada e agora veremos como todo o algoritmo funciona:

```
Se apenas um número
  Retornar
Senão
  Ordenar a metade esquerda do número
```

Ordenar a metade direita do número
Mesclar metades classificadas

Começamos com outra lista não classificada:

6 3 8 5 2 7 4 1

Para começar, precisamos classificar a metade esquerda primeiro:

6 3 8 5

Bem, para classificar isso, precisamos classificar a metade esquerda da metade esquerda primeiro:

6 3

Agora, ambas as metades têm apenas um item cada, portanto, estão classificadas. Nós mesclamos essas duas listas, para obter uma lista classificada:

— — 8 5 2 7 4 1
3 6

Estamos de volta à classificação da metade direita da metade esquerda, mesclando-as:

— — — — 2 7 4 1
3 6 5 8

As duas metades da *metade esquerda* foram classificadas individualmente, então agora precisamos mesclá-las:

— — — — 2 7 4 1
3 5 6 8

Faremos o que acabamos de fazer, com a metade certa:

— — — — — — — —
3 5 6 8 2 7 4 1

- Primeiro, classificamos as duas metades da metade direita.

— — — — — — — —
3 5 6 8 2 7 1 4

- Em seguida, nós os mesclamos para uma metade direita classificada.

Por fim, temos duas metades classificadas novamente e podemos mesclá-las para obter uma lista totalmente classificada:

— — — — — — — —
1 2 3 4 5 6 7 8

Cada número foi movido de uma prateleira para outra três vezes (uma vez que a lista foi dividida de 8 para 4, para 2 e para 1 antes de ser mesclada novamente em listas ordenadas de 2, 4 e, finalmente, 8 novamente). E cada prateleira exigia que todos os 8 números fossem mesclados, um de cada vez.

Cada prateleira exigia n passos, e havia apenas $\log_{\frac{n}{2}} n$ prateleiras necessárias, então multiplicamos esses fatores juntos. Nosso tempo total de execução para pesquisa binária é $O(\log_{\frac{n}{2}} n)$:

- $O(n^2)$
 -
 - selection sort, bubble sort
 -
-
-
- $O(n \log_{\frac{n}{2}} n)$
 -
 - merge sort
 -
-
-
- $O(n)$
 -
 - busca linear
 -
-
-
- $O(\log_{\frac{n}{2}} n)$
 -
 - busca binária
 -
-
-
- $O(1)$

(Uma vez que $\log_{\frac{n}{2}} n$ é maior que 1, mas menor que n , $n \log_{\frac{n}{2}} n$ está entre n (vezes 1) e n^2 .)

O melhor caso, Ω , ainda é $n \log_{\frac{n}{2}} n$, uma vez que ainda temos que classificar cada metade primeiro e, em seguida, mesclá-los juntos:

- $\Omega(n^2)$
 -
 - selection sort
 -
-
-
- $\Omega(n \log_{\frac{n}{2}} n)$

-
- merge sort
-
-
-
- $\Omega(n)$
 -
 - bubble sort
 -
-
-
- $\Omega(\log_{10} n)$
-
- $\Omega(1)$
 -
 - pesquisa linear, pesquisa binária
 -
-

Embora a merge sort provavelmente seja mais rápida do que a selection sort ou bubble sort, precisamos de outra prateleira, ou mais memória, para armazenar temporariamente nossas listas mescladas em cada estágio. Enfrentamos a desvantagem de incorrer em um custo mais alto, outro array na memória, pelo benefício de uma classificação mais rápida.

Finalmente, há outra notação, Θ , Theta, que usamos para descrever os tempos de execução de algoritmos se o limite superior e o limite inferior forem iguais. Por exemplo, merge sort tem **$\Theta(n \log_{10} n)$** , uma vez que o melhor e o pior caso requerem o mesmo número de passos. E a classificação de seleção tem **$\Theta(n^2)$** :

- $\Theta(n^2)$
 -
 - selection sort
 -
-
-
- $\Theta(n \log_{10} n)$
 -
 - merge sort
 -
-
-
- $\Theta(n)$
-
- $\Theta(\log_{10} n)$
-
- $\Theta(1)$

Vemos uma [visualização final](#) dos algoritmos de classificação com um número maior de inputs, em execução ao mesmo tempo.

EXERCÍCIOS MÓDULO 3

Exercício 1: Plurality

Implemente um programa que execute uma eleição de pluralidade, conforme a seguir.

```
$ ./plurality Alice Bob Charlie
Número de eleitores: 4
Voto: Alice
Voto: Bob
Voto: Charlie
Voto: Alice
Alice
```

Background

As eleições acontecem em todas as formas e tamanhos. No Reino Unido, o [primeiro-ministro](#) é oficialmente nomeado pelo monarca, que geralmente escolhe o líder do partido político que obtém mais cadeiras na Câmara dos Comuns. Os Estados Unidos usam um processo de [Colégio Eleitoral](#) de várias etapas, no qual os cidadãos votam em como cada estado deve alocar os Eleitores que então elegem o Presidente.

Talvez a maneira mais simples de realizar uma eleição, no entanto, seja por meio de um método comumente conhecido como “voto plural” (também conhecido como “primeiro a chegar” ou “o vencedor leva tudo”). Na votação por pluralidade, cada eleitor pode votar em um candidato. No final da eleição, o candidato que obtiver o maior número de votos é declarado vencedor da eleição.

Vamos começar!

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute `cd ~` (ou simplesmente `cd` sem argumentos) para garantir que você está no seu diretório pessoal).
-
- Execute `mkdir pset3` para fazer (ou seja, criar) um diretório chamado **pset3**.
-
- Execute `cd pset3` para mudar para (ou seja, abrir) esse diretório.
-
- Execute `mkdir plurality` para fazer (ou seja, criar) um diretório chamado **plurality** em seu diretório **pset3**.
-
- Execute `cd plurality` para mudar para (isto é, abrir) esse diretório.
-
- Execute `wget https://cdn.cs50.net/2020/fall/psets/3/plurality/plurality.c` para baixar o código de distribuição deste problema.
-

- Execute ls . Você deve ver o código de distribuição deste problema, em um arquivo chamado **plurality.c** .

Entendendo...

Vamos agora dar uma olhada em **plurality.c** e ler o código de distribuição que foi fornecido a você.

A linha **#define MAX 9** é alguma sintaxe usada aqui para significar que **MAX** é uma constante (igual a **9**) que pode ser usada em todo o programa. Aqui, ele representa o número máximo de candidatos que uma eleição pode ter.

O arquivo então define uma **struct** chamada **candidate** . Cada **candidate** tem dois campos: uma **string** chamada **name** representando o nome do candidato e um **int** chamado **votes** representando o número de votos que o candidato possui. Em seguida, o arquivo define uma matriz global de **candidates** , em que cada elemento é ele próprio um **candidate** .

Agora, dê uma olhada na própria função **main** . Veja se você consegue descobrir onde o programa define uma variável global **candidate_count** representando o número de candidatos na eleição, copia os argumentos da linha de comando para a vetor **candidates** e pede ao usuário para digitar o número de eleitores. Em seguida, o programa permite que cada eleitor digite um voto (entendeu como?), acionando a função de **vote** em cada candidato votado. Finalmente, main faz uma chamada para a função **print_winner** para imprimir o vencedor (ou vencedores) da eleição.

Se você olhar mais abaixo no arquivo, no entanto, notará que as funções **vote** e **print_winner** foram deixadas em branco. Esta parte depende de você concluir!

Especificações

Complete a implementação de **plurality.c** de tal forma que o programa simule uma eleição de voto plural.

- Complete a função de **vote**.
 -
 - **vote** leva um único argumento, uma **string** chamada **name** , que representa o nome do candidato que foi votado.
 -
 - Se o **name** corresponder a um dos nomes dos candidatos na eleição, atualize o total de votos desse candidato para contabilizar a nova votação. A função de votação(**vote**) neste caso deve retornar **true** para indicar uma votação bem-sucedida.
 -
 - Se o **name** não corresponder ao nome de nenhum dos candidatos na eleição, nenhum total de votos deve mudar e a função de voto(**vote**) deve retornar **false** para indicar uma cédula inválida.
 -
 - Você pode presumir que dois candidatos não terão o mesmo nome.
 -
-
-
- Conclua a função **print_winner** .
 -

- A função deve imprimir o nome do candidato que recebeu mais votos na eleição e, em seguida, imprimir uma nova linha.
-
- É possível que a eleição termine em empate se vários candidatos tiverem, cada um, o número máximo de votos. Nesse caso, você deve imprimir os nomes de cada um dos candidatos vencedores, cada um em uma linha separada.
-
-

Você não deve modificar nada mais em `plurality.c` além das implementações das funções **vote** e **print_winner** (e a inclusão de arquivos de cabeçalho adicionais, se desejar).

Uso

Seu programa deve se comportar de acordo com os exemplos abaixo.

- ```
$./plurality Alice Bob
 Number of voters: 3
 Vote: Alice
 Vote: Bob
 Vote: Alice
 Alice
```
- 
- ```
$ ./plurality Alice Bob
    Number of voters: 3
    Vote: Alice
    Vote: Charlie
    Invalid vote.
    Vote: Alice
    Alice
```
-
- ```
$./plurality Alice Bob Charlie
 Number of voters: 5
 Vote: Alice
 Vote: Charlie
 Vote: Bob
 Vote: Bob
 Vote: Alice
 Alice
 Bob
```

### Como testar o seu código?

Certifique-se de testar seu código para certificar-se de que é capaz de lidar com as seguintes situações:

- Uma eleição com qualquer número de candidatos (até o **MAX** de **9**);
- 
- Votar em um candidato pelo nome;

- 
- Votos inválidos para candidatos que não estão na cédula;
- 
- Imprimir o vencedor da eleição se houver apenas um;
- 
- Imprimir o vencedor da eleição se houver vários vencedores;

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/plurality
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 plurality.c
```

### Exercício 2: Runoff

Implemente um programa que execute uma eleição de segundo turno, conforme a seguir.

```
./runoff Alice Bob Charlie
Número de eleitores: 5
Rank 1: Alice
Rank 2: Bob
Rank 3: Charlie
```

```
Rank 1: Alice
Rank 2: Charlie
Rank 3: Bob
```

```
Rank 1: Bob
Rank 2: Charlie
Rank 3: Alice
```

```
Rank 1: Bob
Rank 2: Alice
Rank 3: Charlie
```

```
Rank 1: Charlie
Rank 2: Alice
Rank 3: Bob
```

Alice

### Background

Você já conhece as eleições pluralistas, que seguem um algoritmo muito simples para determinar o vencedor de uma eleição: cada eleitor ganha um voto e o candidato com mais votos vence.

Mas o voto de pluralidade tem algumas desvantagens. O que acontece, por exemplo, em uma eleição com três candidatos, e as cédulas abaixo são lançadas?

| Ballot | Ballot | Ballot | Ballot | Ballot  |
|--------|--------|--------|--------|---------|
| Alice  | Alice  | Bob    | Bob    | Charlie |

Uma votação de pluralidade declararia aqui um empate entre Alice e Bob, uma vez que cada um tem dois votos. Mas esse é o resultado certo?

Existe outro tipo de sistema de votação conhecido como sistema de votação por escolha ranqueada. Em um sistema de escolha ranqueada, os eleitores podem votar em mais de um candidato. Em vez de apenas votar na primeira escolha, eles podem classificar os candidatos em ordem de preferência. As cédulas resultantes podem, portanto, ser semelhantes às apresentadas a seguir.

| Ballot                           | Ballot                           | Ballot                           | Ballot                           | Ballot                           |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 1. Alice<br>2. Bob<br>3. Charlie | 1. Alice<br>2. Charlie<br>3. Bob | 1. Bob<br>2. Alice<br>3. Charlie | 1. Bob<br>2. Alice<br>3. Charlie | 1. Charlie<br>2. Alice<br>3. Bob |

Aqui, cada eleitor, além de especificar seu primeiro candidato preferencial, também indicou sua segunda e terceira opções. E agora, o que antes era uma eleição empatada, agora pode ter um vencedor. A corrida foi originalmente empatada entre Alice e Bob, então Charlie estava fora da corrida. Mas o eleitor que escolheu Charlie preferiu Alice a Bob, então Alice poderia ser declarada vencedora.

A votação por escolha ranqueada também pode resolver outra desvantagem potencial da votação por pluralidade. Dê uma olhada nas seguintes cédulas.

| Ballot                           | Ballot                           | Ballot                           | Ballot                           | Ballot                           |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 1. Alice<br>2. Bob<br>3. Charlie | 1. Alice<br>2. Bob<br>3. Charlie | 1. Bob<br>2. Alice<br>3. Charlie | 1. Bob<br>2. Alice<br>3. Charlie | 1. Bob<br>2. Alice<br>3. Charlie |

| Ballot                           | Ballot                           | Ballot                           | Ballot                           |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 1. Charlie<br>2. Alice<br>3. Bob | 1. Charlie<br>2. Alice<br>3. Bob | 1. Charlie<br>2. Bob<br>3. Alice | 1. Charlie<br>2. Bob<br>3. Alice |

Quem deve ganhar esta eleição? Em uma votação de pluralidade em que cada eleitor escolhe apenas sua primeira preferência, Charlie vence esta eleição com quatro votos, em comparação com apenas três para Bob e dois para Alice. Mas a maioria dos eleitores (5 de 9) ficaria mais feliz com Alice ou Bob em vez de Charlie. Ao considerar as preferências ranqueadas, um sistema de votação pode ser capaz de escolher um vencedor que reflita melhor as preferências dos eleitores.

Uma dessas opções de sistema de votação por classificação é o sistema de runoff instantâneo (ou uma eleição com turnos). Em uma eleição de runoff instantâneo, os eleitores podem ranquear quantos candidatos quiserem. Se algum candidato tiver a maioria (mais de 50%) dos votos da primeira preferência, esse candidato é declarado vencedor da eleição.

Se nenhum candidato tiver mais de 50% dos votos, ocorre um “runoff instantâneo”. O candidato que recebeu o menor número de votos é eliminado da eleição, e quem originalmente escolheu esse candidato como sua primeira preferência agora tem sua segunda preferência considerada. Por que fazer assim? Efetivamente, isso simula o que teria acontecido se o candidato menos popular não tivesse estado na eleição para começar.

O processo repete-se: se nenhum candidato obtiver a maioria dos votos, o último candidato colocado é eliminado e quem votou nele votará na sua próxima preferência (os que ainda não foram eliminados). Uma vez que um candidato tenha a maioria, ele é declarado o vencedor.

Vamos considerar as nove cédulas acima e examinar como ocorreria uma eleição com turnos.

Alice tem dois votos, Bob tem três votos e Charlie tem quatro votos. Para ganhar uma eleição com nove pessoas, é necessária uma maioria (cinco votos). Como ninguém tem maioria, é necessário realizar um segundo turno. Alice tem o menor número de votos (com apenas dois), então Alice é eliminada. Os eleitores que votaram originalmente em Alice listaram Bob como segunda preferência, então Bob obtém os dois votos extras. Bob agora tem cinco votos e Charlie ainda tem quatro votos. Bob agora tem a maioria e Bob é declarado o vencedor.

Que casos extremos precisamos considerar aqui?

Uma possibilidade é que haja um empate para quem deve ser eliminado. Podemos lidar com esse cenário dizendo que todos os candidatos que estão empatados no último lugar serão eliminados. Se todos os candidatos restantes tiverem exatamente o mesmo número de votos, porém, eliminar os candidatos empatados em último lugar significa eliminar todos! Então, nesse caso, teremos que ter cuidado para não eliminar todos, e apenas declarar a eleição um empate entre todos os candidatos restantes.

Algumas eleições de runoff instantâneo não exigem que os eleitores classifiquem todas as suas preferências - portanto, pode haver cinco candidatos em uma eleição, mas um eleitor pode escolher apenas dois. Para os fins deste problema, entretanto, vamos ignorar esse caso particular e presumir que todos os eleitores classificarão todos os candidatos em sua ordem preferida.

Parece um pouco mais complicado do que uma votação plural, não é? Mas pode-se argumentar que tem a vantagem de ser um sistema eleitoral em que o vencedor da eleição representa com mais precisão as preferências dos eleitores.

### Começando

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#), em uma janela de terminal, execute cada um dos itens abaixo.

- Navegue até o diretório pset3 que já deve existir.
- 
- Execute `mkdir runoff` de `mkdir` para criar (ou seja, criar) um diretório chamado `runoff` em seu diretório `pset3`.
- 
- Execute `cd runoff` para mudar para (ou seja, abrir) esse diretório.
- 
- Execute `wget https://cdn.cs50.net/2020/fall/psets/3/runoff/runoff.c` para baixar o código de distribuição deste problema.
- 
- Execute `ls`. Você deve ver o código de distribuição deste problema, em um arquivo chamado `runoff.c`.

### Entendendo...

Vamos abrir `runoff.c` para dar uma olhada no que já está lá. Estamos definindo duas constantes: `MAX_CANDIDATES` para o número máximo de candidatos na eleição e `MAX_VOTERS` para o número máximo de eleitores na eleição.

Em seguida, estão as preferências de matriz bidimensional. A matriz de preferências `[i]` representará todas as preferências para o eleitor número `i`, e as preferências de inteiros `[i][j]` aqui armazenarão o índice do candidato que é a `j` ésima preferência do eleitor `i`.

O próximo é um struct chamado `candidate`. Cada `candidate` tem um campo de string para seu `name` e `int` que representa o número de votes que eles têm atualmente, e um valor `bool` chamado `eliminated` que indica se o candidato foi eliminado da eleição. Os `candidates` da matriz acompanharão todos os candidatos na eleição.

O programa também possui duas variáveis globais: `voter_count` e `candidate_count`.

Agora em `main`. Observe que, após determinar o número de candidatos e eleitores, o ciclo de votação principal começa, dando a cada eleitor a chance de votar. Conforme o eleitor insere suas preferências, a função de voto é chamada para controlar todas as preferências. Se a qualquer momento a cédula for considerada inválida, o programa é encerrado.

Uma vez que todos os votos foram alcançados, outro ciclo começa: este vai continuar repetindo o processo de segundo turno para verificar se há um vencedor e eliminar o último candidato a lugar até que haja um vencedor.

A primeira chamada aqui é para uma função chamada `tabulate`, que deve olhar para todas as preferências dos eleitores e computar os totais de votos atuais, observando cada candidato eleito pela primeira vez que ainda não foi eliminado. Em seguida, a função `print_winner` deve imprimir o vencedor, caso aplicável; se houver, o programa acabou. Mas, caso contrário, o programa precisa determinar o menor número de votos que alguém ainda na eleição recebeu (por meio de uma chamada para `find_min`). Se ficar claro que todos na eleição estão empatados com o mesmo número de votos (conforme determinado pela função `is_tie`), a eleição é declarada empatada; caso contrário, o candidato (ou candidatos) em último lugar é eliminado da eleição por meio de uma chamada para a função de eliminação.

Se você olhar um pouco mais abaixo no arquivo, verá que essas funções - **`vote`**, **`tabulate`**, **`print_winner`**, **`find_min`**, **`is_tie`** e **`eliminate`** - são todas deixadas para você concluir!

## Especificações

Conclua a implementação de runoff.c de forma que simule uma eleição de turnos. Você deve completar as implementações das funções `vote`, `tabulate`, `print_winner`, `find_min`, `is_tie` e `eliminate`, e não deve modificar mais nada em runoff.c (e a inclusão de arquivos de cabeçalho adicionais, se desejar).

- `vote`
  - 
  - A função leva os argumentos `voter`, `rank` e `name`. Se o `name` corresponder ao nome de um candidato válido, você deve atualizar a matriz de preferências globais para indicar que o eleitor `voter` tem aquele candidato como sua preferência de `rank` (onde 0 é a primeira preferência, 1 é a segunda preferência, etc.)
  - 
  - Se a preferência for registrada com sucesso, a função deve retornar `true`; caso contrário, a função deve retornar `false` (se, por exemplo, nome não for o nome de um dos candidatos).
  - 
  - Você pode presumir que dois candidatos não terão o mesmo nome.
  - 
  - Lembre-se de que `candidate_count` armazena o número de candidatos na eleição.
  - 
  - Lembre-se de que você pode usar `strcmp` para comparar duas strings.
  - 
  - Lembre-se de que as `preferences[i][j]` armazenam o índice do candidato que é a `j`ésima preferência classificada para o `i`ésimo eleitor.
  -
- 
- 
- `tabulate`
  - 
  - A função deve atualizar o número de votes que cada candidato possui nesta fase do segundo turno.
  - 
  - Lembre-se de que em cada estágio do segundo turno, cada eleitor vota efetivamente em seu candidato preferido que ainda não foi eliminado.
  - 
  - Lembre-se de que `voter_count` armazena o número de eleitores na eleição.
  - 
  - Lembre-se de que, para um eleitor `i`, seu candidato de primeira escolha é representado por `preferences[i][0]`, seu candidato de segunda escolha por `preferences[i][1]`, etc.
  - 
  - Lembre-se de que a estrutura do candidato possui um campo denominado `eliminado`, o que será `true` caso o candidato tenha sido eliminado da eleição.
  -

- Lembre-se de que a struct do candidate tem um campo chamado votes , que você provavelmente desejará atualizar para o candidato preferido de cada eleitor.
- 
- 
- 
- print\_winner
  - 
  - Se algum candidato tiver mais da metade dos votos, seu nome deve ser impresso em stdout e a função deve retornar true .
  - 
  - Se ninguém ganhou a eleição ainda, a função deve retornar falso .
  - 
  - Lembre-se de que voter\_count armazena o número de eleitores na eleição. Diante disso, como você expressaria o número de votos necessários para vencer a eleição?
  -
- 
- 
- find\_min
  - 
  - A função deve retornar o total mínimo de votos para qualquer candidato que ainda esteja na eleição.
  - 
  - Você provavelmente vai querer percorrer os candidatos para encontrar aquele que ainda está na eleição e tem o menor número de votos. Que informações você deve acompanhar enquanto analisa os candidatos?
  -
- 
- 
- is\_tie
  - 
  - A função leva um argumento min , que será o número mínimo de votos que alguém na eleição tem atualmente.
  - 
  - A função deve retornar verdadeiro se todos os candidatos restantes na eleição tiverem o mesmo número de votos, e deve retornar falso caso contrário.
  - 
  - Lembre-se de que o empate ocorre se todos os candidatos ainda na eleição tiverem o mesmo número de votos. Observe também que a função is\_tie leva um argumento min , que é o menor número de votos que qualquer candidato possui atualmente. Como você pode usar essas informações para determinar se a eleição é um empate (ou, inversamente, não um empate)?
  -
- 
- 
- eliminate
  -



- A função leva um argumento min , que será o número mínimo de votos que alguém na eleição tem atualmente.
- 
- A função deve eliminar o candidato (ou candidatos) com número mín de votos.
- 

•

### Uso

Seu programa deve se comportar conforme o exemplo abaixo:

```
./runoff Alice Bob Charlie
```

```
Número de eleitores: 5
```

```
Rank 1: Alice
```

```
Rank 2: Charlie
```

```
Rank 3: Bob
```

```
Rank 1: Alice
```

```
Rank 2: Charlie
```

```
Rank 3: Bob
```

```
Rank 1: Bob
```

```
Rank 2: Charlie
```

```
Rank 3: Alice
```

```
Rank 1: Bob
```

```
Rank 2: Charlie
```

```
Rank 3: Alice
```

```
Rank 1: Charlie
```

```
Rank 2: Alice
```

```
Rank 3: Bob
```

```
Alice
```

### Como testar o código?

Certifique-se de testar seu código para ver como ele se comporta no caso de:

- Uma eleição com qualquer número de candidatos (até o **MAX 9**)
- 
- Votar em um candidato pelo nome
- 
- Votos inválidos para candidatos que não estão na cédula
- 
- Imprimir o vencedor da eleição se houver apenas um
- 
- Não eliminando ninguém em caso de empate entre todos os candidatos restantes

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50** . Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/runoff

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

style50 runoff.c

### Exercício 3: Tideman

Implemente um programa que execute uma eleição Tideman (ou eleição de pares ranqueados), conforme a seguir.

```
./tideman Alice Bob Charlie
Número de eleitores: 5
Rank 1: Alice
Rank 2: Charlie
Rank 3: Bob
```

```
Rank 1: Alice
Rank 2: Charlie
Rank 3: Bob
```

```
Rank 1: Bob
Rank 2: Charlie
Rank 3: Alice
```

```
Rank 1: Bob
Rank 2: Charlie
Rank 3: Alice
```

```
Rank 1: Charlie
Rank 2: Alice
Rank 3: Bob
```

Charlie

### Background

Você já conhece as eleições pluralistas, que seguem um algoritmo muito simples para determinar o vencedor de uma eleição: cada eleitor ganha um voto e o candidato com mais votos vence.

Mas o voto de pluralidade tem algumas desvantagens. O que acontece, por exemplo, em uma eleição com três candidatos, e as cédulas abaixo são lançadas?

•

Uma votação de pluralidade declararia aqui um empate entre Alice e Bob, uma vez que cada um tem dois votos. Mas esse é o resultado certo?

Existe outro tipo de sistema de votação conhecido como sistema de votação por escolha ranqueada. Em um sistema de escolha ranqueada, os eleitores podem votar em mais de um candidato. Em vez de apenas votar na primeira escolha, eles podem classificar os candidatos em ordem de preferência. As cédulas resultantes podem, portanto, ser semelhantes às apresentadas a seguir.

•

Aqui, cada eleitor, além de especificar seu primeiro candidato preferencial, também indicou sua segunda e terceira opções. E agora, o que antes era uma eleição empatada, agora pode ter um vencedor. A corrida foi originalmente empatada entre Alice e Bob, então Charlie estava fora da corrida. Mas o eleitor que escolheu Charlie preferiu Alice a Bob, então Alice poderia ser declarada vencedora.

A votação por escolha ranqueada também pode resolver outra desvantagem potencial da votação por pluralidade. Dê uma olhada nas seguintes cédulas.

•

Quem deve ganhar esta eleição? Em uma votação de pluralidade em que cada eleitor escolhe apenas sua primeira preferência, Charlie vence a eleição com quatro votos, em comparação com apenas três para Bob e dois para Alice. (Observe que, se você estiver familiarizado com o sistema de runoff instantâneo, ou eleição em turnos, Charlie também vence nesse sistema). Alice, no entanto, pode razoavelmente argumentar que ela deveria ser a vencedora da eleição em vez de Charlie: afinal, dos nove eleitores, a maioria (cinco deles) preferia Alice a Charlie, então a maioria das pessoas ficaria mais feliz com Alice como o vencedor em vez de Charlie.

Alice é, nesta eleição, a chamada “vencedora Condorcet” da eleição: a pessoa que teria vencido qualquer confronto direto com outro candidato. Se a eleição tivesse sido apenas Alice e Bob, ou apenas Alice e Charlie, Alice teria vencido.

O método de votação Tideman (também conhecido como “pares ranqueados”) é um método de votação de escolha ranqueada que garante o vencedor da eleição de Condorcet, se houver.

De modo geral, o método Tideman funciona construindo um “gráfico” de candidatos, onde uma seta (isto é, aresta) do candidato A ao candidato B indica que o candidato A vence o candidato B em um confronto direto. O gráfico para a eleição acima, então, seria parecido com o abaixo.

•

A seta de Alice para Bob significa que mais eleitores preferem Alice a Bob (5 preferem Alice, 4 preferem Bob). Da mesma forma, as outras setas significam que mais eleitores preferem Alice a Charlie e mais eleitores preferem Charlie a Bob.

Olhando para este gráfico, o método Tideman diz que o vencedor da eleição deve ser a “fonte” do gráfico (ou seja, o candidato que não tem uma seta apontando para ele). Nesse caso, a fonte é Alice - Alice é a única que não tem uma seta apontando para ela, o que significa que ninguém é preferido frente a frente com a Alice. Alice é, portanto, declarada a vencedora da eleição.

É possível, entretanto, que quando as flechas forem puxadas, não haja um vencedor Condorcet. Considere as cédulas abaixo.



Entre Alice e Bob, Alice tem preferência sobre Bob por uma margem de 7-2. Entre Bob e Charlie, Bob é o preferido em relação a Charlie por uma margem de 5-4. Mas entre Charlie e Alice, Charlie é o preferido a Alice por uma margem de 6-3. Se desenharmos o gráfico, não haverá fonte! Temos um ciclo de candidatos, onde Alice vence Bob que vence Charlie que vence Alice (muito parecido com um jogo de pedra-papel-tesoura). Nesse caso, parece que não há como escolher um vencedor.

Para lidar com isso, o algoritmo Tideman deve ser cuidadoso para evitar a criação de ciclos no gráfico candidato. Como é feito isso? O algoritmo “bloqueia” as setas mais “fortes” primeiro, uma vez que essas são, sem dúvida, as mais significativas. Em particular, o algoritmo do Tideman especifica que as setas de cada “duelo” devem ser “travadas” no gráfico, uma de cada vez, com base na “força” da vitória (quanto mais pessoas preferirem um candidato ao adversário, mais forte será a vitória) . Desde que a aresta possa ser travada no gráfico sem criar um ciclo, a aresta é adicionada; caso contrário, é ignorada.

Como isso funcionaria no caso dos votos acima? Bem, a maior margem de vitória para um par é Alice derrotando Bob, já que 7 eleitores preferem Alice a Bob (nenhuma outra disputa direta tem um vencedor preferido por mais de 7 eleitores). Portanto, a seta Alice-Bob é travada no gráfico primeiro. A próxima maior margem de vitória é a vitória de Charlie por 6-3 sobre Alice, de modo que a flecha é a próxima.

A seguir vem a vitória de Bob por 5-4 sobre Charlie. Mas observe: se adicionássemos uma flecha de Bob a Charlie agora, criaríamos um ciclo! Uma vez que o gráfico não permite ciclos, devemos pular esta borda e não adicioná-la ao gráfico. Se houvesse mais setas a serem consideradas, olharíamos para as próximas, mas essa foi a última seta, então o gráfico está completo.

Este processo passo a passo é mostrado abaixo, com o gráfico final à direita.

•

Com base no gráfico resultante, Charlie é a fonte (não há nenhuma seta apontando para Charlie), então Charlie é declarado o vencedor desta eleição.

Colocado de forma mais formal, o método de votação do Tideman consiste em três partes:

- **Contagem/Tally:** Uma vez que todos os eleitores indicaram todas as suas preferências, determine, para cada par de candidatos, quem é o candidato preferido e com que margem ele é preferido.
- 
- **Classificação/Ordenação/Sort :** Classifique os pares de candidatos em ordem decrescente de força de vitória, onde a força de vitória é definida pelo número de eleitores que preferem o candidato preferido.
- 
- **Decisão/Lock:** começando com o par mais forte, passe pelos pares de candidatos em ordem e “trave” cada par no gráfico candidato, contanto que travar nesse par não crie um ciclo no gráfico.

Assim que o gráfico estiver completo, a fonte do gráfico (aquela sem arestas apontando para ele) é a vencedora!

### Começando

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Navegue até o diretório pset3 que já deve existir.
-

- Execute `mkdir tideman` para fazer (ou seja, criar) um diretório chamado `tideman` em seu diretório `pset3`.
- 
- Execute `cd tideman` para mudar para (ou seja, abrir) esse diretório.
- 
- Execute `wget https://cdn.cs50.net/2020/fall/psets/3/tideman/tideman.c` para baixar o código de distribuição deste problema.
- 
- Execute `ls`. Você deve ver o código de distribuição deste problema, em um arquivo chamado `tideman.c`.

### Entendendo...

Vamos abrir o `tideman.c` para dar uma olhada no que já está lá.

Primeiro, observe as preferências de matriz bidimensional. O `int preferences[i][j]` representa o número de eleitores que preferem o candidato `i` ao candidato `j`.

O arquivo também define outra matriz bidimensional, chamada `locked`, que representará o gráfico candidato. `locked` é um array booleano, portanto, `locked[i][j]` sendo que `true` representa a existência de uma aresta apontando do candidato `i` para o candidato `j`; `false` significa que não há vantagem. (Se estiver curioso, esta representação de um gráfico é conhecida como uma “matriz de adjacência”).

Em seguida, vem uma struct chamada `pair`, usada para representar um par de candidatos: cada par inclui o índice de candidato vencedor `winner` e do candidato perdedor `loser`.

Os próprios candidatos são armazenados na matriz `candidates`, que é uma matriz de strings que representa os nomes de cada um dos candidatos. Há também uma matriz de `pairs`, que representará todos os pares de candidatos (para os quais um é preferido em relação ao outro) na eleição.

O programa também possui duas variáveis globais: `pair_count` e `candidate_count`, representando o número de pares e o número de candidatos nos `pairs` e `candidates` das matrizes, respectivamente.

Agora em `main`. Observe que, após determinar o número de candidatos, o programa percorre o gráfico `locked` e inicialmente define todos os valores como `false`, o que significa que nosso gráfico inicial não terá arestas.

Em seguida, o programa faz um loop sobre todos os eleitores e recolhe as suas preferências em um array chamado `ranks` (via uma chamada para `vote`), onde `ranks[i]` é o índice do candidato que é a `i`ésima preferência para o eleitor. Essas classificações são passadas para a função `record_preference`, cujo trabalho é pegar essas classificações e atualizar a variável de preferências globais.

Uma vez que todos os votos estão registrados, os pares de candidatos são adicionados ao array de pares por meio de um chamado para `add_pairs`, classificados por meio de uma chamada para `sort_pairs` e “fixos” no gráfico por meio de uma chamada para `lock_pairs`. Finalmente, `print_winner` é chamado para imprimir o nome do vencedor da eleição!

Mais abaixo no arquivo, você verá que as funções `vote`, `record_preference`, `add_pairs`, `sort_pairs`, `lock_pairs` e `print_winner` são deixadas em branco. Isso é contigo!

## Especificação

Complete a implementação de `tideman.c` de forma que simule uma eleição Tideman. Observe as especificações das funções:

- `vote .`
  - 
  - A função leva os argumentos `rank` , `name` e `rank`s . Se o nome corresponder ao nome de um candidato válido, você deve atualizar a matriz de classificação para indicar que o eleitor tem o candidato como sua preferência de classificação (onde 0 é a primeira preferência, 1 é a segunda preferência, etc.)
  - 
  - Lembre-se que fileiras `[i]` aqui representa a `i` ésima preferência do usuário.
  - 
  - A função deve retornar `true` se a classificação foi registrada com sucesso e `false` caso contrário (se, por exemplo, `name` não for o nome de um dos candidatos).
  - 
  - Você pode presumir que dois candidatos não terão o mesmo nome.
  -
- 
- 
- `record_preferences .`
  - 
  - A função é chamada uma vez para cada eleitor, e toma como argumento a matriz `rank`s, (lembrar que `rank`s `[i]` é a `i` ésima preferência do eleitor, onde `rank`s `[0]` é a primeira preferência).
  - 
  - A função deve atualizar a matriz de `rank`s globais para adicionar as preferências do eleitor atual. Lembre-se de que `rank`s `[i] [j]` deve representar o número de eleitores que preferem o candidato `i` ao candidato `j` .
  - 
  - Você pode presumir que cada eleitor classificará cada um dos candidatos.
  -
- 
- 
- `add_pairs .`
  - 
  - A função deve adicionar todos os pares de candidatos onde um candidato é preferido à matriz de `pair`s . Um par de candidatos empatados (um não tem preferência sobre o outro) não deve ser adicionado à matriz.
  - 
  - A função deve atualizar a variável global `pair_count` para ser o número de pares de candidatos. (Os pares devem, portanto, ser armazenados entre os pares `[0]` e os pares `[pair_count - 1]` , inclusive).
  -
- 
-

- `sort_pairs` .
  - 
  - A função deve classificar a matriz de pairs em ordem decrescente de força de vitória, onde força de vitória é definida como o número de eleitores que preferem o candidato preferido. Se vários pares tiverem a mesma força de vitória, você pode assumir que a ordem não importa.
  -
- 
- 
- `lock_pairs` .
  - 
  - A função deve criar o gráfico `locked` , adicionando todas as arestas em ordem decrescente de força de vitória, desde que a aresta não crie um ciclo.
  -
- 
- 
- `print_winner` .
  - 
  - A função deve imprimir o nome do candidato que é a fonte do gráfico. Você pode presumir que não haverá mais de uma fonte.
  -
- 

Você não deve modificar nada mais em `tideman.c` além das implementações das funções `vote` , `record_preferences` , `add_pairs` , `sort_pairs` , `lock_pairs` e `print_winner` (e a inclusão de arquivos de cabeçalho adicionais, se desejar). Você tem permissão para adicionar funções adicionais a `tideman.c` , desde que não altere as declarações de nenhuma das funções existentes.

### Mão na massa

Seu programa deve se comportar conforme o exemplo abaixo:

```
./tideman Alice Bob Charlie
Número de eleitores: 5
Rank 1: Alice
Rank 2: Charlie
Rank 3: Bob
```

```
Rank 1: Alice
Rank 2: Charlie
Rank 3: Bob
```

```
Rank 1: Bob
Rank 2: Charlie
Rank 3: Alice
```

```
Rank 1: Bob
Rank 2: Charlie
Rank 3: Alice
```

Rank 1: Charlie

Rank 2: Alice

Rank 3: Bob

Charlie

### Testando seu código

Certifique-se de testar seu código para ver como ele se comporta no caso de:

- Uma eleição com qualquer número de candidatos (até o **MAX 9**)
- 
- Votar em um candidato pelo nome
- 
- Votos inválidos para candidatos que não estão na cédula
- 
- Imprimindo o vencedor da eleição

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/tideman
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 tideman.c
```

## Lab 3: Ordenação

### Laboratório 3: Ordenar

Analise três programas de classificação para determinar quais algoritmos eles usam.

### Background

Lembre-se da aula em que vimos alguns algoritmos para classificar uma sequência de números. Eram eles: selection sort, bubble sort e merge sort.

- A selection sort itera através das partes não classificadas de uma lista, selecionando o menor elemento a cada vez e movendo-o para seu local correto.
- 
- A bubble sort compara pares de valores adjacentes, um de cada vez, e os troca se estiverem na ordem incorreta. Isso continua até que a lista seja classificada.
- 
- A merge sort divide recursivamente a lista em duas repetidamente e, em seguida, mescla as listas menores de volta em uma maior na ordem correta.

### Começando

Faça login em [CS50 IDE](#) usando sua conta GitHub.



- Na janela do seu terminal, execute wget <https://cdn.cs50.net/2020/fall/labs/3/lab3.zip> para baixar um arquivo Zip do código de distribuição do laboratório.
- 
- Na janela do seu terminal, execute unzip lab3.zip para descompactar (ou seja, descompactar) esse arquivo Zip.
- 
- Na janela do terminal, execute cd lab3 para alterar os diretórios para o diretório lab3.

### Instruções

São fornecidos três programas C já compilados, **sort1**, **sort2** e **sort3**. Cada um desses programas implementa um algoritmo de classificação diferente: selection sort, bubble sort ou merge sort (embora não necessariamente nessa ordem!). Sua tarefa é determinar qual algoritmo de classificação é usado por cada arquivo.

- sort1, sort2 e sort3 são arquivos binários, então você não poderá ver o código-fonte C de cada um. Para avaliar qual classificação implementa qual algoritmo, execute as classificações em diferentes listas de valores.
- 
- Vários arquivos **.txt** são fornecidos a você. Esses arquivos contêm n linhas de valores, revertidos, embaralhados ou classificados.
  - 
  - Por exemplo, reversed10000.txt contém 10.000 linhas de números que são revertidas de 10.000, enquanto random100000.txt contém 100.000 linhas de números que estão em ordem aleatória.
  -
- 
- 
- Para executar as classificações nos arquivos de texto, no terminal, execute **./[nome\_do\_programa] [arquivo\_texto.txt]**.
  - 
  - Por exemplo, para classificar reversed10000.txt com SORT1, execute reversed10000.txt ./sort1.
  -
- 
- 
- Você pode achar útil cronometrar os tempos de execucao. Para fazer isso, execute o **time./[sort\_file] [text\_file.txt]**.
  - 
  - Por exemplo, você pode executar time ./sort1 reversed10000.txt para executar sort1 em 10.000 números invertidos. No final da saída do seu terminal, você pode olhar o tempo real para ver quanto tempo realmente passou durante a execução do programa.
  -
- 
-

- Registre suas respostas em `answers.txt` , juntamente com uma explicação para cada programa, preenchendo os espaços marcados com **TODO** .

#### Dica:

Os diferentes tipos de arquivos `.txt` podem ajudá-lo a determinar qual é o tipo de classificação utilizada . Considere o desempenho de cada algoritmo com uma lista já classificada. Que tal uma lista invertida? Ou lista embaralhada? Pode ajudar trabalhar com uma lista menor de cada tipo e percorrer cada processo de classificação.

#### Como verificar suas respostas

Execute o seguinte comando para avaliar a correção de suas respostas usando **check50** . Mas não deixe de preencher também seus comentários, pois o **check50** não marcará aqui!

```
check50 cs50/labs/2021/x/sort
```

## ANOTAÇÕES MÓDULO 4

### Hexadecimal

Na semana 2, falamos sobre memória e como cada byte tem um endereço, ou identificador, para que possamos nos referir a onde nossos dados estão realmente armazenados.

Acontece que, por convenção, os endereços de memória usam o sistema de contagem **hexadecimal**, ou base-16, onde existem 16 dígitos: 0-9, e AF como equivalentes a 10-15.

Vamos considerar um número hexadecimal de dois dígitos:

$$\begin{array}{cc} 16^1 & 16^0 \\ 0 & A \end{array}$$

- Aqui, o A na casa das unidades (uma vez que  $16^0 = 1$ ) tem um valor decimal de 10. Podemos continuar contando até **0F**, que é equivalente a 15 em decimal.

Depois de **0F** , precisamos carregar o um, pois iríamos de 09 para 10 em decimal:

$$\begin{array}{cc} 16^1 & 16^0 \\ 1 & 0 \end{array}$$

- Aqui, o **1** tem um valor de  $16^1 * 1 = 16$ , então **10** em hexadecimal é 16 em decimal.

Com dois dígitos, podemos ter um valor máximo de **FF** , ou  $16^1 * 15 + 16^0 * 15 = 240 + 15 = 255$ , que é o mesmo valor máximo com 8 bits de binário. Portanto, dois dígitos em hexadecimal podem representar convenientemente o valor de um byte em binário. ( Cada dígito em hexadecimal, com 16 valores, mapeia para quatro bits em binário.)

Por escrito, indicamos que um valor está em hexadecimal prefixando-o com **0x** , como em **0x10** , onde o valor é igual a 16 em decimal, em oposição a 10.

O sistema de cores RGB convencionalmente usa hexadecimal para descrever a quantidade de cada cor. Por exemplo, 000000 em hexadecimal representa 0 para cada um de vermelho, verde e azul, para uma cor combinada de preto. E FF0000 seria 255, ou a maior quantidade possível de vermelho. FFFFFFF indicaria o valor mais alto de cada cor, combinando para ser o branco mais brilhante. Com valores diferentes para cada cor, podemos representar milhões de cores diferentes.

Para a memória do nosso computador, também usaremos hexadecimal para cada endereço ou localização.

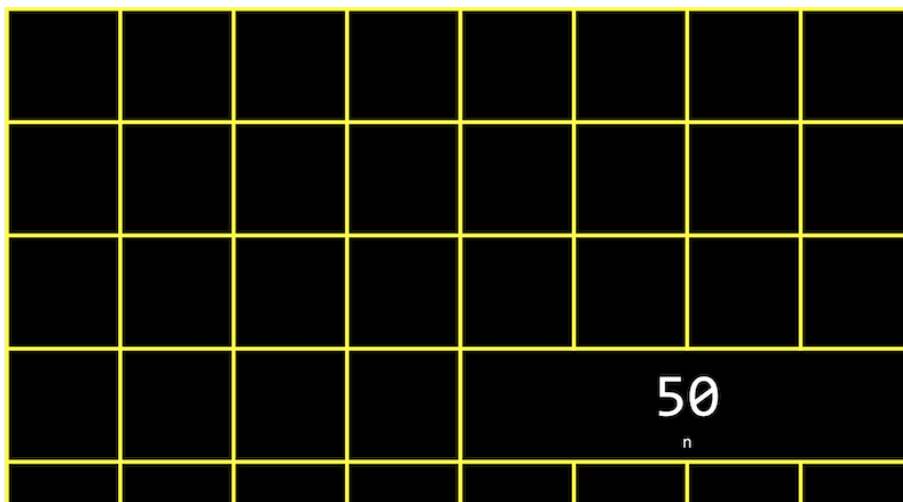
### Endereços de Memória

Podemos criar um valor `n` e imprimi-lo:

```
#include <stdio.h>

int main(void)
{
 int n = 50;
 printf("%i\n", n);
}
```

Na memória do nosso computador, agora existem 4 bytes em algum lugar que têm o valor binário de 50, rotulados `n`:



Acontece que, com os bilhões de bytes na memória, esses bytes para a variável `n` começam em algum local, que pode ser algo como 0x12345678.

Em C, podemos realmente ver o endereço com o operador `&`, que significa “obter o endereço desta variável”:

```
#include <stdio.h>

int main(void)
{
 int n = 50;
 printf("%p\n", &n);
}
```

- `%p` é o código de formato de um endereço.
- 
- No IDE CS50, vemos um endereço como `0x7ffd80792f7c`. O valor do endereço em si não é útil, pois é apenas algum local na memória onde a variável está armazenada; em vez disso, a ideia importante é que podemos usar esse endereço mais tarde.

O operador `*`, ou operador de desreferência, nos permite “ir para” o local para o qual um ponteiro está apontando.

Por exemplo, podemos imprimir `*&n`, onde “vamos para” o endereço de `n`, e isso imprimirá o valor de `n`, `50`, já que esse é o valor no endereço de `n`:

```
#include <stdio.h>
```

```
int main(void)
{
 int n = 50;
 printf("%i\n", *&n);
}
```

## Ponteiros

Uma variável que armazena um endereço é chamada de **pointer** (“**ponteiro**”), que podemos pensar como um valor que “aponta” para um local na memória. Em C, os ponteiros podem se referir a tipos específicos de valores.

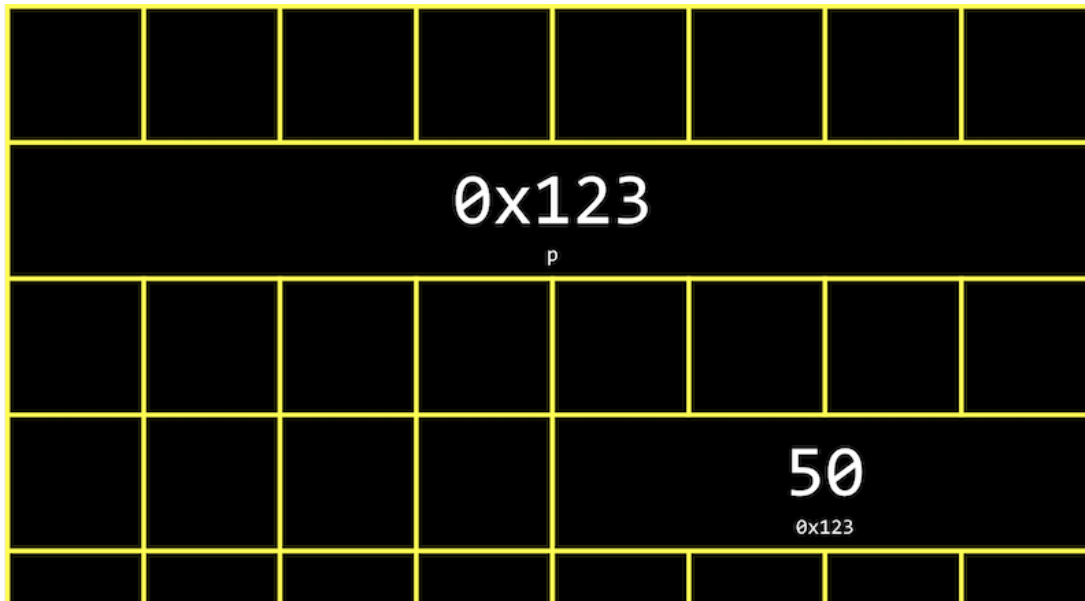
Podemos usar o operador `*` (de uma forma infelizmente confusa) para declarar uma variável que queremos que seja um pointer:

```
#include <stdio.h>
```

```
int main(void)
{
 int n = 50;
 int *p = &n;
 printf("%p\n", p);
}
```

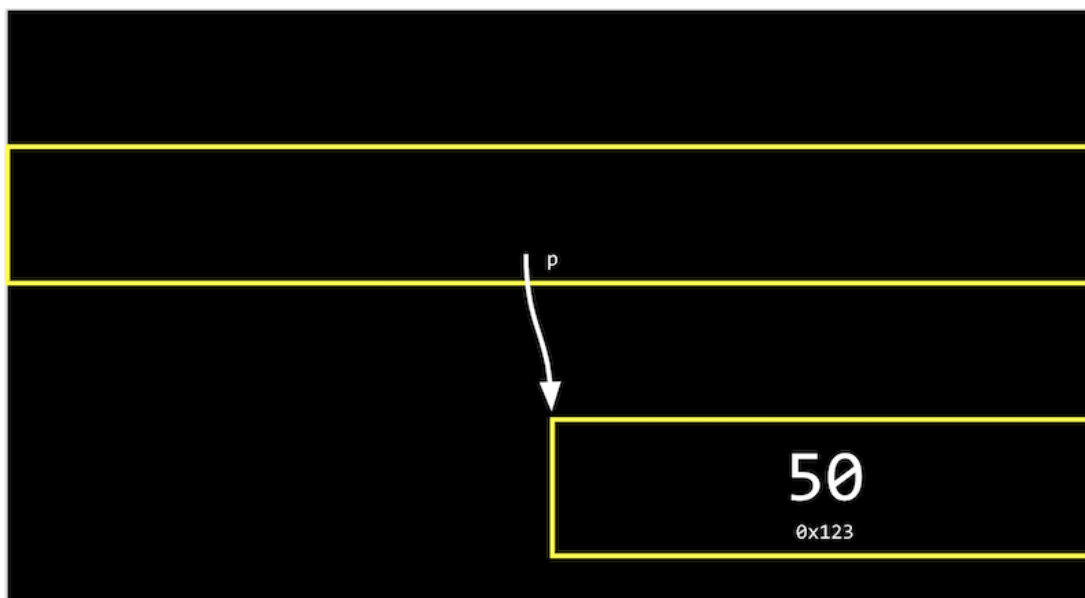
- Aqui, usamos `int * p` para declarar uma variável, `p`, que tem o tipo `*`, um ponteiro, para um valor do tipo `int`, um inteiro. Então, podemos imprimir seu valor (um endereço, algo como `0x12345678`), ou imprimir o valor em sua localização com `printf ("%i\n", *p);`.

Na memória do nosso computador, as variáveis serão assim:



- Como **p** é uma variável em si, está em algum lugar na memória e o valor armazenado lá é o endereço de **n**.
- 
- Os sistemas de computador modernos são de “64 bits”, o que significa que eles usam 64 bits para endereçar a memória, então um ponteiro terá na realidade 8 bytes, duas vezes o tamanho de um inteiro de 4 bytes.

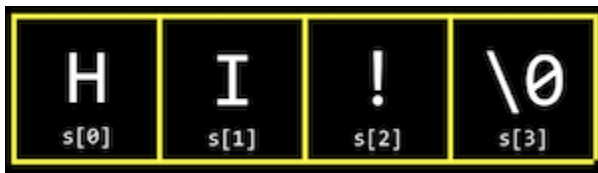
Podemos abstrair o valor real dos endereços, uma vez que eles serão diferentes conforme declaramos variáveis em nossos programas e não muito úteis, e simplesmente pensar em **p** como "apontando para" algum valor:



No mundo real, podemos ter uma caixa de correio identificada como “p”, entre muitas caixas de correio com endereços. Dentro de nossa caixa de correio, podemos colocar um valor como **0x123**, que é o endereço de alguma outra caixa de correio **n**, com o endereço **0x123**.

## Strings

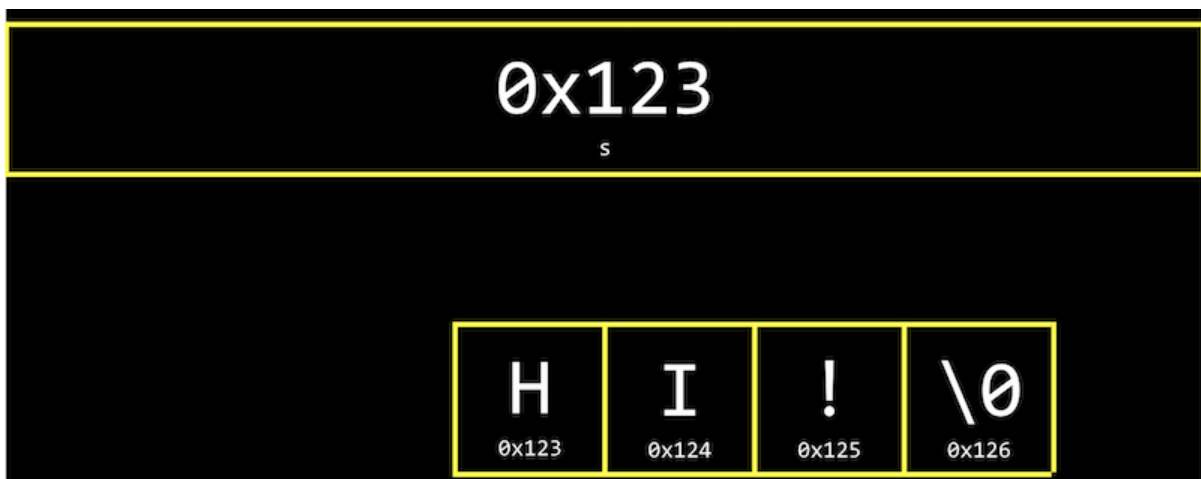
Uma variável declarada com **string s = "HI!";** será armazenado um caractere por vez na memória. E podemos acessar cada caractere com **s[0]**, **s[1]**, **s[2]** e **s[3]**:



Mas acontece que cada caractere, por estar armazenado na memória, também possui algum endereço exclusivo, e **s** é na verdade apenas um ponteiro com o endereço do primeiro caractere:



E a variável **s** armazena o endereço do primeiro caractere da string. O valor **\0** é o único indicador do final da string:



- Já que o resto dos caracteres estão em um array, um apos o outro, podemos começar no endereço indicado no **s** e continuar lendo um caractere de cada vez a partir da memória até chegarmos no **\0**.

Vamos imprimir uma string:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
 string s = "HI!";
 printf("%s\n", s);
}
```

Podemos ver o valor armazenado em **s** com **printf("%p \n", s);**, e vemos algo como **0x4006a4**, pois estamos imprimindo o endereço na memória do primeiro caractere da string.

Se adicionarmos outra linha, `printf("%p \n", &s[1]);`, de fato vemos o próximo endereço na memória: `0x4006a5`.

Acontece que a **string** `s` é apenas um ponteiro, um endereço para algum caractere na memória.

Na verdade, a biblioteca CS50 define um tipo que não existe em C, **string**, como `char *`, com `typedef char *string;`. O tipo personalizado, **string**, é definido apenas como um `char *` com `typedef`. Então `string s = "HI!";` é o mesmo que `char *s = "HI!";`. E podemos usar strings em C exatamente da mesma maneira sem a biblioteca CS50, usando `char *`.

### *Pointer arithmetic/Aritmética de ponteiros*

**Pointer arithmetic** (“aritmética de ponteiros”) são operações matemáticas em endereços com ponteiros.

Podemos imprimir cada caractere em uma string (usando `char *` diretamente):

```
#include <stdio.h>
```

```
int main(void)
{
 char *s = "HI!";
 printf("%c\n", s[0]);
 printf("%c\n", s[1]);
 printf("%c\n", s[2]);
}
```

Mas podemos ir diretamente para os endereços:

```
#include <stdio.h>
```

```
int main(void)
{
 char *s = "HI!";
 printf("%c\n", *s);
 printf("%c\n", *(s+1));
 printf("%c\n", *(s+2));
}
```

- `*s` vai para o endereço armazenado em `s`, e `*(s+1)` vai para o local na memória com um endereço um byte acima, ou o próximo caractere. `s[1]` é açúcar sintático para `*(s+1)`, equivalente em função, mas mais amigável para ler e escrever.

Podemos até tentar ir para endereços na memória que não deveríamos, como com `*(s+10000)`, e quando executarmos nosso programa, teremos uma **falha de segmentação**(**segmentation fault**) ou travar como resultado de nosso programa tocar na memória em um segmento que não deveria.

### **Compare e copie**

Vamos tentar comparar dois inteiros que o usuário forneceu:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
 int i = get_int("i: ");
 int j = get_int("j: ");
 if (i == j)
 {
 printf("Igual\n");
 }
 else
 {
 printf("Diferente\n");
 }
}
```

- Compilamos e executamos nosso programa, e ele funciona como esperávamos, com os mesmos valores dos dois inteiros nos dando “igual” e valores diferentes “diferente”.

Quando tentamos comparar duas strings, vemos que os mesmo inputs estão fazendo com que nosso programa imprima "Diferente":

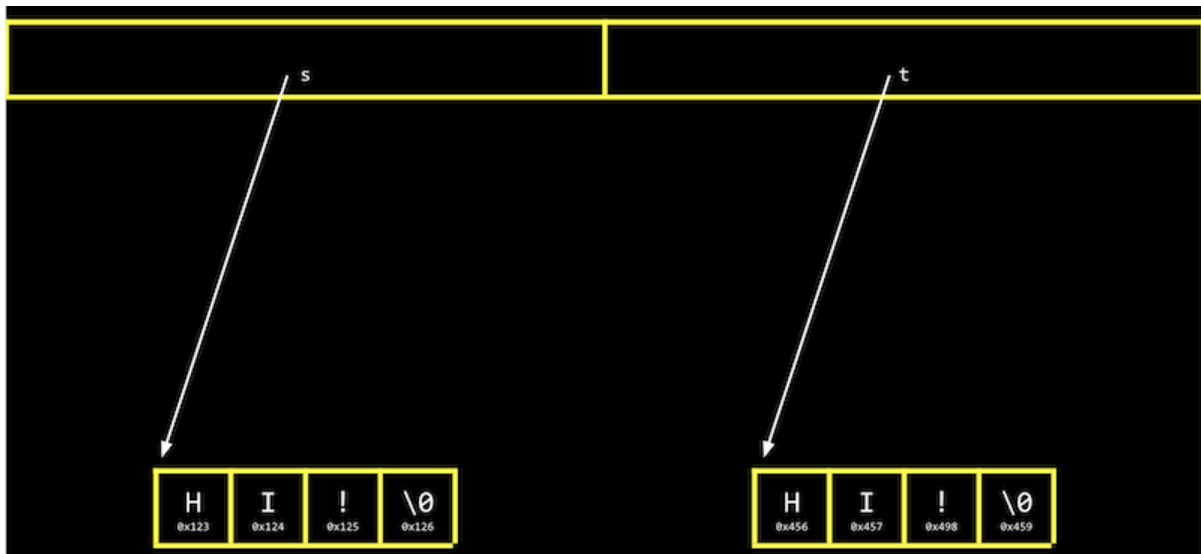
```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
 char *s = get_int("s: ");
 char *t = get_int("t: ");
 if (s == t)
 {
 printf("Igual\n");
 }
 else
 {
 printf("Diferente\n");
 }
}
```

- Mesmo quando nossos inputs são os mesmas, vemos "diferente" impresso.
- 
- Cada “string” é um ponteiro, **char \***, para um local diferente na memória, onde o primeiro caractere de cada string é armazenado. Portanto, mesmo se os caracteres na string forem iguais, isso sempre imprimirá “Diferente”.



Por exemplo, nossa primeira string pode estar no endereço 0x123, nossa segunda pode estar em 0x456 e **s** terá o valor de **0x123**, apontando para aquele local e **t** terá o valor de **0x456**, apontando para outro local:



E **get\_string**, esse tempo todo, retornou apenas um **char \***, ou um ponteiro para o primeiro caractere de uma string do usuário. Como chamamos **get\_string** duas vezes, recebemos dois ponteiros diferentes de volta.

Vamos tentar copiar uma string:

```
#include <cs50.h>
#include <ctype.h>
#include <stdio.h>

int main(void)
{
 char *s = get_string("s: ");
 char *t = s;
 t[0] = toupper(t[0]);
 printf("s: %s\n", s);
 printf("t: %s\n", t);
}
```

- Pegamos uma string **s** e copiamos o valor de **s** em **t**. Em seguida, colocamos a primeira letra em **t** em maiúscula.
- 
- Mas quando executamos nosso programa, vemos que tanto **s** quanto **t** agora estão em letras maiúsculas.
- 
- Como definimos **s** e **t** com o mesmo valor ou o mesmo endereço, eles apontam para o mesmo caractere e, portanto, colocamos o mesmo caractere em maiúscula na memória!

Para realmente fazer uma cópia de uma string, temos que trabalhar um pouco mais e copiar cada caractere em **s** para outro lugar na memória:

```

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
 char *s = get_string("s: ");
 char *t = malloc(strlen(s) + 1);
 for (int i = 0, n = strlen(s); i < n + 1; i++)
 {
 t[i] = s[i];
 }
 t[0] = toupper(t[0]);
 printf("s: %s\n", s);
 printf("t: %s\n", t);
}

```

- Nós criamos uma nova variável, **t**, do tipo **char \***, com **char \*t**. Agora, queremos apontá-lo para um novo pedaço de memória que é grande o suficiente para armazenar a cópia do string. Com **malloc**, nós alocamos algum número de bytes de memória (que já não são utilizados para armazenar outros valores), e nós passamos no número de bytes que gostaríamos de marcar para uso. Nós já sabemos o comprimento de **s**, e nós adicionamos 1 ao do caractere nulo de terminação. Assim, a última linha de código é **char \*t = malloc(strlen(s) + 1);**.
- 
- Em seguida, copiamos cada caractere, um de cada vez, com um **for-loop**. Usamos **i < n + 1**, uma vez que realmente queremos ir até **n**, o comprimento da string, para garantir que copiaremos o caractere de terminação na string. No loop, definimos **t[i] = s[i]**, copiando os caracteres. Embora possamos usar **\*(t+1) = \*(s+1)** para o mesmo efeito, é indiscutivelmente menos legível.
- 
- Agora, podemos colocar apenas a primeira letra de **t** em maiúscula.

Podemos adicionar verificação de erros ao nosso programa:

```

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
 char *s = get_string("s: ");
 char *t = malloc(strlen(s) + 1);
 if (t == NULL)

```

```

{
 return 1;
}
for (int i = 0, n = strlen(s); i < n + 1; i++)
{
 t[i] = s[i];
}
if (strlen(t) > 0)
{
 t[0] = toupper(t[0]);
}
printf("s: %s\n", s);
printf("t: %s\n", t);
free(t);
}

```

- Se nosso computador estiver sem memória, **malloc** retornará **NULL**, o ponteiro nulo ou um valor especial que indica que não há um endereço para o qual apontar. Portanto, devemos verificar esse caso e sair se **t** for **NULL**.
- 
- Também poderíamos verificar se **t** tem um comprimento, antes de tentar colocar o primeiro caractere em maiúscula.
- 
- Finalmente, devemos **liberar(free)** a memória que alocamos anteriormente, o que a marca como utilizável novamente por algum outro programa. Chamamos a função **free** e passamos o ponteiro **t**, já que terminamos com aquele pedaço de memória. (**get\_string**, também, chama **malloc** para alocar memória para strings e chama **free** antes do retorno da função principal(**main**)).

Na verdade, também podemos usar a função **strcpy**, da biblioteca de strings do C, com **strcpy(t,s);** em vez de nosso loop, para copiar a string **s** em **t**.

## Valgrind

**valgrind** é uma ferramenta de linha de comando que podemos usar para executar nosso programa e ver se há algum **memory leak** (“vazamento de memória”) ou memória que alocamos sem liberar, o que pode eventualmente fazer com que o computador fique sem memória.

Vamos construir uma string, mas alocar menos do que precisamos na memória.c :

```

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 char *s = malloc(3);
 s[0] = 'H';
 s[1] = 'I';
 s[2] = '!';
}

```

```

 s[3] = '\\0';
 printf("%s\\n", s);
}

```

- Também não liberamos a memória que alocamos.
- 
- Executaremos **valgrind ./memory** após a compilação e veremos muitos resultados, mas podemos executar **help valgrind ./memory** para ajudar a explicar algumas dessas mensagens. Para este programa, vemos trechos como “Gravação inválida de tamanho 1”, “Leitura inválida de tamanho 1” e, finalmente, “3 bytes em 1 bloco são definitivamente perdidos”, com números de linha próximos. Na verdade, estamos gravando na memória, **s[3]**, que não faz parte do que alocamos originalmente para **s**. E quando imprimimos **s**, estamos lendo até **s[3]** também. E, finalmente, **s** não é liberado no final do nosso programa.

Podemos nos certificar que alocamos o número certo de bytes e liberar memória no final:

```

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 char *s = malloc(4);
 s[0] = 'H';
 s[1] = 'I';
 s[2] = '!';
 s[3] = '\\0';
 printf("%s\\n", s);
 free(s);
}

```

- Agora, **valgrind** não mostra nenhuma mensagem de aviso.

### Garbage Values(valores de lixo)

Vamos dar uma olhada no seguinte:

```

int main(void)
{
 int *x;
 int *y;
 x = malloc(sizeof(int));
 *x = 42;
 *y = 13;
 y = x;
 *y = 13;
}

```

- Declaramos dois ponteiros para inteiros, **x** e **y**, mas não atribuímos valores a eles. Usamos **malloc** para alocar memória suficiente para um inteiro com **sizeof(int)** e armazená-lo em **x**. **\*x = 42** vai para o endereço **x** aponta para e define esse local na memória para o valor 42.
- 
- Com **\*y = 13**, estamos tentando colocar o valor 13 no endereço que **y** aponta. Mas, como nunca atribuímos um valor a **y**, ele tem um **garbage value** (“valor lixo”), ou qualquer valor desconhecido que estava na memória, de qualquer programa que estava em execução em nosso computador antes. Então, quando tentamos ir para o valor lixo em **y** como um endereço, estamos indo para algum endereço desconhecido, que provavelmente causará uma falha de segmentação ou segfault.

Assistimos [Pointer Fun with Binky](#), um vídeo animado que demonstra os conceitos do código acima.

Podemos imprimir valores inúteis, declarando uma matriz, mas não definindo nenhum de seus valores:

```
#include <stdio.h>

int main(void)
{
 int scores[3];
 for (int i = 0; i < 3; i++)
 {
 printf("%i\n", scores[i]);
 }
}
```

- Quando compilamos e executamos este programa, vemos vários valores impressos.

### Swap(Troca)

Vamos tentar trocar os valores de dois inteiros.

```
include <stdio.h>

void swap(int a, int b);

int main(void)
{
 int x = 1;
 int y = 2;
 printf("x is %i, y is %i\n", x, y);
 swap(x, y);
 printf("x is %i, y is %i\n", x, y);
}
```

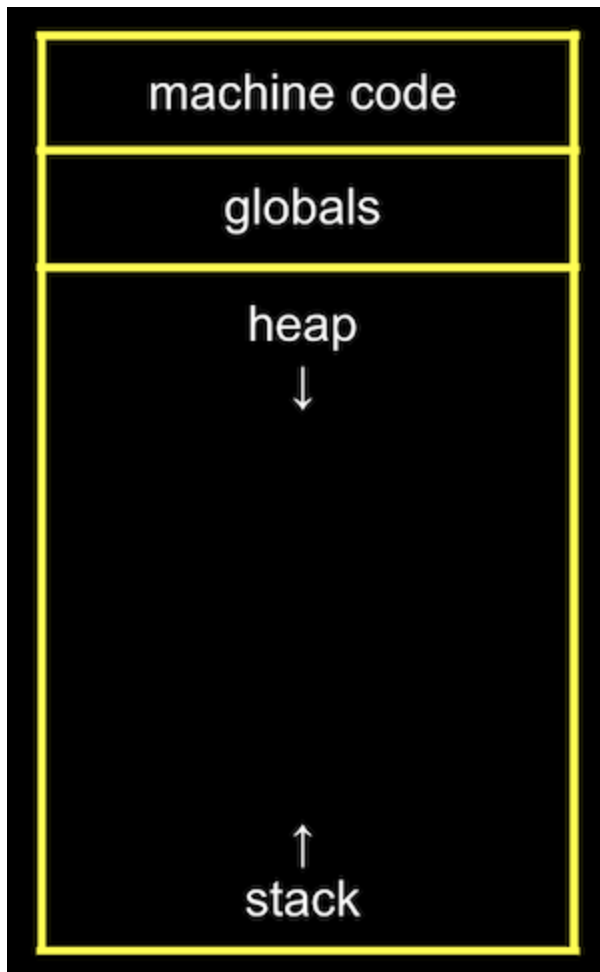
```
void swap(int a, int b)
{
 int tmp = a;
 a = b;
 b = tmp;
}
```

- No mundo real, se tivéssemos um líquido vermelho em um copo e um líquido azul em outro e quiséssemos trocá-los, precisaríamos de um terceiro copo para conter temporariamente um dos líquidos, talvez o vidro vermelho. Então, podemos derramar o líquido azul no primeiro copo e, finalmente, o líquido vermelho do copo temporário no segundo.
- 
- Em nossa função **swap** (“troca”), temos uma terceira variável para usar também como espaço de armazenamento temporário. Colocamos **a** em **tmp** e, em seguida, definimos **a** com o valor de **b** e, finalmente, **b** pode ser alterado para o valor original de **a**, agora em **tmp**.

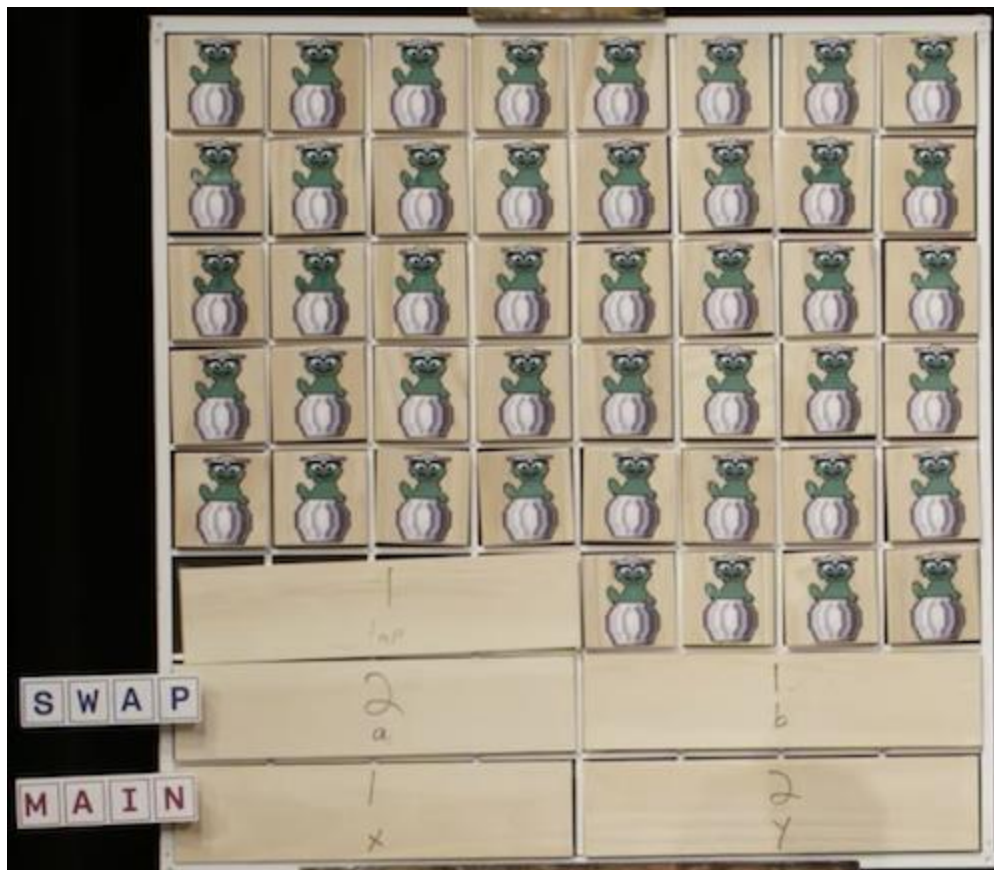
Mas, se tentamos usar essa função em um programa, não vemos nenhuma mudança. Acontece que a função **swap** obtém suas próprias variáveis, **a** e **b** quando eles são passados como argumentos, que são cópias de **x** e **y**, e assim mudar esses valores não mudam **x** e **y** na função **main**.

### Layout de memória

Na memória do nosso computador, os diferentes tipos de dados que precisam ser armazenados para o nosso programa são organizados em diferentes seções:



- A seção de **machine code** (“código de máquina”) é o código binário do nosso programa compilado. Quando executamos nosso programa, esse código é carregado no “topo” da memória.
- 
- Logo abaixo, ou na próxima parte da memória, estão as **variáveis globais** que declaramos em nosso programa.
- 
- A seção de **heap** é uma área vazia de onde **malloc** pode obter memória livre para nosso programa usar. Como chamamos **malloc**, começamos a alocar memória de cima para baixo.
- 
- A seção de **stack** é usada por funções em nosso programa conforme são chamadas e cresce para cima. Por exemplo, nossa função principal(**main**) está na parte inferior da pilha e tem as variáveis locais **x** e **y**. A função **swap**, quando chamada, tem sua própria área de memória que fica no topo da **main**, com as variáveis locais **a**, **b** e **tmp**:



Assim que a função **swap** retorna algo, a memória que estava usando é liberada para a próxima vez que a função for chamada. **x** e **y** são argumentos, por isso eles são copiados como **a** e **b** para **swap**, por isso não vemos nossas alterações de volta na **main**.

Ao passar o endereço de **x** e **y**, nossa função **swap** pode realmente funcionar:

```
#include <stdio.h>
```

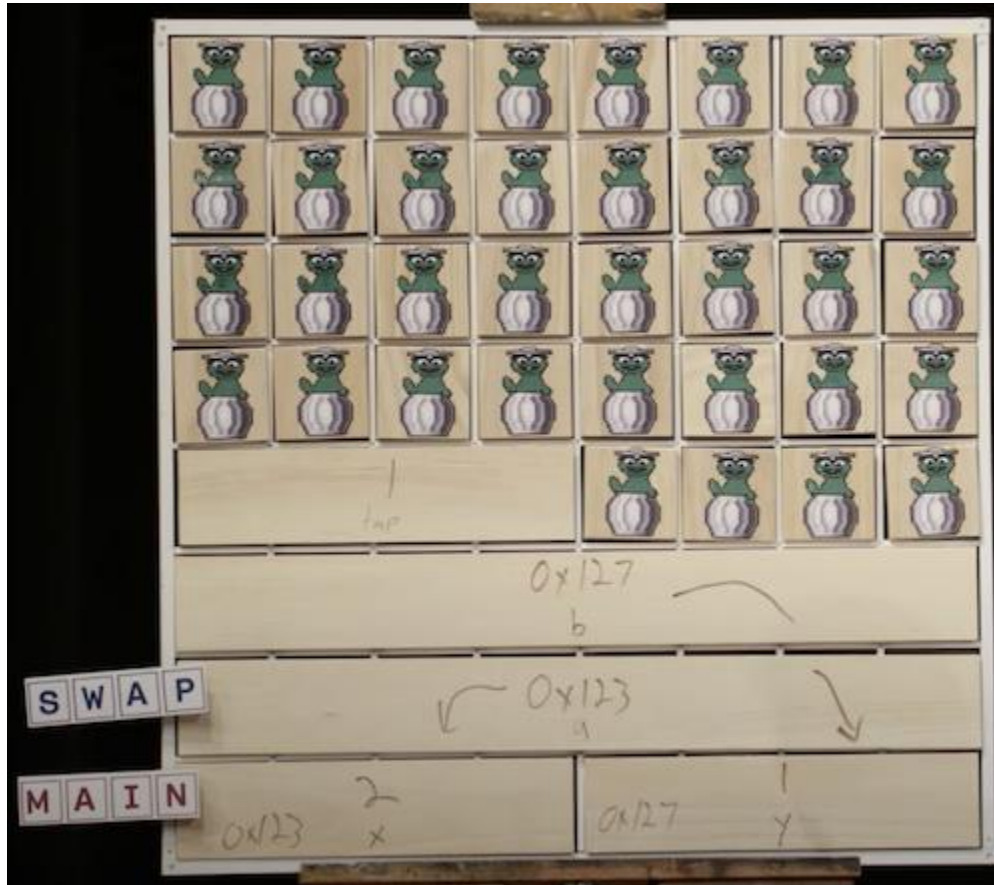
```
void swap(int *a, int *b);
```

```
int main(void)
{
 int x = 1;
 int y = 2;
 printf("x é %i, y é %i\n", x, y);
 swap(&x, &y);
 printf("x é %i, y é %i\n", x, y);
}
```

```
void swap(int *a, int *b)
{
 int tmp = *a;
 *a = *b;
 *b = tmp;
}
```



- Os endereços de **x** e **y** são passados na **main** para a **swap** com **&x** e **&y**, e usamos a sintaxe **int \*a** para declarar que nossa função **swap** recebe ponteiros. Salvamos o valor de **x** para **tmp** seguindo o ponteiro **a**, e então pegamos o valor de **y** seguindo o ponteiro **b**, e armazenamos isso no local que **a** está apontando para (**x**). Por fim, armazenamos o valor de **tmp** no local apontado por **b** (**y**) e pronto:



Se chamarmos **malloc** para muita memória, teremos um **heap overflow**, uma vez que acabamos ultrapassando nosso heap. Ou, se chamarmos muitas funções sem retornar delas, teremos um **stack overflow**, onde nossa pilha também tem muita memória alocada.

Vamos implementar o desenho da pirâmide de Mario, chamando uma função:

```
#include <cs50.h>
#include <stdio.h>

void draw(int h);

int main(void)
{
 int height = get_int("Altura: ");
 draw(height);
}

void draw(int h)
{
 for (int i = 1; i <= h; i++)
```

```

 {
 for (int j = 1; j <= i; j++)
 {
 printf("#");
 }
 printf("\n");
 }
}

```

Podemos alterar **draw** para ser recursiva:

```

void draw(int h)
{
 draw(h - 1);
 for (int i = 0; i < h; i++)
 {
 printf("#");
 }
 printf("\n");
}

```

- Quando tentamos compilar isso com **make**, vemos um aviso de que a função **draw** se chamará recursivamente sem parar. Portanto, usaremos o **clang** sem as verificações extras e, quando executamos este programa, obtemos uma falha de segmentação imediatamente. **draw** está chamando a si mesmo indefinidamente e ficamos sem memória na pilha.

Ao adicionar um base case (“caso base”), a função **draw** irá parar de chamar a si mesma em algum ponto:

```

void draw(int h)
{
 if (h == 0)
 {
 return;
 }
 draw(h - 1);
 for (int i = 0; i < h; i++)
 {
 printf("#");
 }
 printf("\n");
}

```

- Mas se inserirmos um valor grande o suficiente para a altura, como **2000000000**, ainda ficaremos sem memória, já que chamaremos **draw** muitas vezes sem retornar.

Um **buffer overflow** ocorre quando passamos do final de um buffer, algum pedaço de memória que alocamos como um array, e acessamos partes da memória que não deveríamos.

## scanf

Podemos implementar `get_int` nós mesmos com uma função de biblioteca C, `scanf`:

```
#include <stdio.h>

int main(void)
{
 int x;
 printf("x: ");
 scanf("%i", &x);
 printf("x: %i\n", x);
}
```

- `scanf` assume um formato, `%i`, então a entrada é “escaneada” para esse formato. Também passamos na memória o endereço para onde queremos que essa entrada vá. Mas `scanf` não tem muita verificação de erros, então podemos não obter um número inteiro.

Podemos tentar obter uma string da mesma maneira:

```
#include <stdio.h>

int main(void)
{
 char *s;
 printf("s: ");
 scanf("%s", s);
 printf("s: %s\n", s);
}
```

- Mas, na verdade, não alocamos nenhuma memória para `s`, então precisamos chamar `malloc` para alocar memória para caracteres de nossa string. Também poderíamos usar `char s[4];` para declarar uma matriz de quatro caracteres. Então, `s` será tratado como um ponteiro para o primeiro caractere em `scanf` e `printf`.
- 
- Agora, se o usuário digitar uma string de comprimento 3 ou menos, nosso programa funcionará com segurança. Mas se o usuário digitar uma string mais longa, `scanf` pode estar tentando escrever além do final de nosso array na memória desconhecida, fazendo com que nosso programa trave.
- 
- `get_string` da biblioteca CS50 aloca continuamente mais memória conforme o `scanf` lê mais caracteres, portanto, ele não tem esse problema.

## Arquivos

Com a capacidade de usar ponteiros, também podemos abrir arquivos, como uma lista telefônica digital:

```

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
 FILE *file = fopen("phonebook.csv", "a");
 if (file == NULL)
 {
 return 1;
 }
 char *nome = get_string("Nome: ");
 char *numero = get_string("Número: ");
 fprintf(file, "%s,%s\n", nome, numero);
 fclose(file);
}

```

- **fopen** é uma nova função que podemos usar para abrir um arquivo. Ele retornará um ponteiro para um novo tipo, **FILE**, de onde podemos ler e escrever. O primeiro argumento é o nome do arquivo, e o segundo argumento é o modo em que queremos abrir o arquivo (**r** para ler, **w** para escrever e **a** para acrescentar ou adicionar).
- 
- Adicionaremos um “checkpoint” para sair, caso não possamos abrir o arquivo por algum motivo.
- 
- Depois de obter algumas strings, podemos usar **fprintf** para imprimir em um arquivo.
- 
- Finalmente, fechamos o arquivo com **fclose**.

Agora podemos criar nossos próprios arquivos CSV, um arquivo de valores separados por vírgulas (como uma mini planilha), programaticamente.

## Gráficos

Podemos ler em binário e mapeá-los em pixels e cores, para exibir imagens e vídeos. Com um número finito de bits em um arquivo de imagem, porém, só podemos ampliar até certo ponto antes de começarmos a ver pixels individuais.

- Com inteligência artificial e machine learning, no entanto, podemos usar algoritmos que podem gerar detalhes adicionais que não existiam antes, por adivinhação com base em outros dados.

Vejamos um programa que abre um arquivo e nos diz se é um arquivo JPEG, um arquivo de imagem em um formato específico:

```

#include <stdint.h>
#include <stdio.h>

```

```

typedef uint8_t BYTE;

int main(int argc, char *argv[])
{
 // Verificar o uso
 if (argc != 2)
 {
 return 1;
 }
 // Abrir o arquivo
 FILE *file = fopen(argv[1], "r");
 if (!file)
 {
 return 1;
 }
 // Ler os primeiros 3 bytes
 BYTE bytes[3];
 fread(bytes, sizeof(BYTE), 3, file);
 // Verificar os três primeiros bytes
 if (bytes[0] == 0xff && bytes[1] == 0xd8 && bytes[2] == 0xff)
 {
 printf("Talvez\n");
 }
 else
 {
 printf("Não\n");
 }
 // Fechar o arquivo
 fclose(file);
}

```

- Primeiro, definimos um **BYTE** como 8 bits, para que possamos nos referir a um byte como um tipo mais facilmente em C.
- 
- Em seguida, tentamos abrir um arquivo (verificando se realmente obtemos um arquivo não NULL de volta) e lemos os primeiros três bytes do arquivo com **fread** , em um buffer chamado **bytes**.
- 
- Podemos comparar os primeiros três bytes (em hexadecimal) aos três bytes necessários para iniciar um arquivo JPEG. Se forem iguais, é provável que nosso arquivo seja um arquivo JPEG (embora outros tipos de arquivos ainda possam começar com esses bytes). Mas se eles não forem iguais, sabemos que definitivamente não é um arquivo JPEG.

Podemos até copiar arquivos nós mesmos, um byte de cada vez agora:

```

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

```

```

typedef uint8_t BYTE;

int main(int argc, char *argv[])
{
 // Garanta o uso adequado
 if(argc != 3)
 {
 fprintf(stderr, "Use: copy SOURCE DESTINATION\n");
 return 1;
 }
 // Abrir o arquivo de entrada
 FILE *source = fopen(argv[1], "r");
 if(source == NULL)
 {
 printf("Não foi possível abrir %s.\n", argv[1]);
 return 1;
 }
 // Abrir o arquivo de saída
 FILE *destination = fopen(argv[2], "w");
 if (destination == NULL)
 {
 fclose(source);
 printf("Não foi possível criar %s.\n", argv[2]);
 return 1;
 }
 // Copiar um byte de cada vez do arquivo origem(source) para o
 arquivo destino(destination)
 BYTE buffer;
 while(fread(&buffer, sizeof(BYTE), 1, source))
 {
 fwrite(&buffer, sizeof(BYTE), 1, destination);
 }
 // Fechar os arquivos
 fclose(source);
 fclose(destination);
 return 0;
}

```

- Usamos **argv** para obter argumentos, usando-os como nomes de arquivos para abrir arquivos para ler e escrever.
- 
- Em seguida, lemos um byte do arquivo de **origem(source)** em um buffer e gravamos esse byte no arquivo de **destino(destination)**. Podemos usar um **while** loop para chamar **fread** , que vai parar uma vez que não há mais bytes para ler.

Podemos usar essas habilidades para ler e gravar arquivos, recuperando imagens de um arquivo e adicionando filtros às imagens, alterando os bytes nelas, no conjunto de problemas desta semana!

## Lab 4: Volume

### Lab: Volume

Escreva um programa para modificar o volume de um arquivo de áudio.

```
$./volume input.wav output.wav 2.0
```

### Arquivos WAV

Os [arquivos WAV](#) são um formato de arquivo comum para representar áudio. Os arquivos WAV armazenam áudio como uma sequência de “amostras”: números que representam o valor de algum sinal de áudio em um determinado momento. Os arquivos WAV começam com um “cabeçalho” de 44 bytes que contém informações sobre o próprio arquivo, incluindo o tamanho do arquivo, o número de amostras por segundo e o tamanho de cada amostra. Após o cabeçalho, o arquivo WAV contém uma sequência de amostras, cada uma com um único inteiro de 2 bytes (16 bits) representando o sinal de áudio em um determinado momento.

Escalar cada valor de amostra por um determinado fator tem o efeito de alterar o volume do áudio. Multiplicar cada valor de amostra por 2,0, por exemplo, terá o efeito de dobrar o volume do áudio de origem. Multiplicar cada amostra por 0,5, entretanto, terá o efeito de cortar o volume pela metade.

### Tipos

Até agora, vimos vários tipos diferentes em C, incluindo **int**, **bool**, **char**, **double**, **float** e **long**. Dentro de um arquivo de cabeçalho chamado **stdint.h** estão as declarações de vários outros tipos que nos permitem definir com muita precisão o tamanho (em bits) e o sinal (com ou sem sinal) de um inteiro. Dois tipos em particular serão úteis para nós neste laboratório.

- **uint8\_t** é um tipo que armazena um inteiro sem sinal de 8 bits (ou seja, não negativo). Podemos tratar cada byte do cabeçalho de um arquivo WAV como um valor **uint8\_t**.
- 
- **int16\_t** é um tipo que armazena um inteiro com sinal de 16 bits (ou seja, positivo ou negativo). Podemos tratar cada amostra de áudio em um arquivo WAV como um valor **int16\_t**.

### Vamos começar...

- Faça login em [ide.cs50.io](https://ide.cs50.io) usando sua conta GitHub.
- 
- Na janela do seu terminal, execute **wget** <https://cdn.cs50.net/2020/fall/labs/4/lab4.zip> para baixar um arquivo Zip do código de distribuição do laboratório.
- 
- Na janela do seu terminal, execute **unzip lab4.zip** para descompactar esse arquivo Zip.
- 
- Na janela do terminal, execute **cd lab4** para alterar os diretórios para o diretório **lab4**.

### Detalhes de implementação

Conclua a implementação de `volume.c`, de forma que mude o volume de um arquivo de som por um determinado fator (**factor**).

- Por exemplo, se o **factor** for **2.0**, então seu programa deve dobrar o volume do arquivo de áudio em **input** e salvar o arquivo de áudio recém-gerado em **output**.

O programa aceita três argumentos de linha de comando: **input** representa o nome do arquivo de áudio original, **output** representa o nome do novo arquivo de áudio que deve ser gerado e **factor** é a quantidade pela qual o volume do arquivo de áudio original deve ser escalado.

- 
- Observe que `volume.c` já define uma variável para você chamada `HEADER_SIZE`, igual ao número de bytes no cabeçalho.
- 

Seu programa deve primeiro ler o cabeçalho do arquivo de entrada e gravar o cabeçalho no arquivo de saída. Lembre-se de que esse cabeçalho tem sempre exatamente 44 bytes. Seu programa deve então ler o restante dos dados do arquivo WAV, uma amostra de 16 bits (2 bytes) por vez. Seu programa deve multiplicar cada amostra pelo fator e gravar a nova amostra no arquivo de saída.

- Você pode presumir que o arquivo WAV usará valores com sinal de 16 bits como amostras. Na prática, os arquivos WAV podem ter vários números de bits por amostra, mas assumiremos amostras de 16 bits para este laboratório.

Seu programa, se usar `malloc`, não deve permitir nenhum vazamento de memória.

### Dicas

Provavelmente, você desejará criar uma matriz de bytes para armazenar os dados do cabeçalho do arquivo WAV que lerá do arquivo de entrada. Usando o tipo `uint8_t` para representar um byte, você pode criar uma matriz de `n` bytes para o seu cabeçalho com sintaxe como

```
uint8_t header[n];
```

substituindo `n` pelo número de bytes. Você pode então usar o **header** como um argumento para **fread** ou **fwrite** para ler ou escrever a partir do cabeçalho.

Provavelmente, você desejará criar um “buffer” para armazenar amostras de áudio lidas do arquivo WAV. Usando o tipo `int16_t` para armazenar uma amostra de áudio, você pode criar uma variável de buffer com sintaxe como

```
int16_t buffer;
```

Você pode então usar o **&buffer** como um argumento para **fread** ou **fwrite** para ler ou escrever no buffer. (Lembre-se de que o operador **&** é usado para obter o endereço da variável.)

Você pode achar a documentação para [fread](#) e [fwrite](#) útil aqui.



- Em particular, observe que ambas as funções aceitam os seguintes argumentos:
  - 
  - **ptr** : um ponteiro para o local na memória para armazenar dados (ao ler de um arquivo) ou de onde gravar dados (ao gravar dados em um arquivo)
  - 
  - **size**: o número de bytes em um item de dados
  - 
  - **nmemb**: o número de itens de dados (cada um de bytes de size ) para ler ou escrever
  - 
  - **stream**: o ponteiro do arquivo a ser lido ou escrito
  -
- 
- 
- De acordo com sua documentação, o **fread** retornará o número de itens de dados lidos com sucesso. Pode ser útil verificar quando chegar ao final do arquivo!

#### Como testar seu código

Seu programa deve se comportar de acordo com os exemplos abaixo.

```
$./volume input.wav output.wav 2.0
```

Quando você escuta **output.wav**(clcando duas vezes em **output.wav** no navegador de arquivos ou executando **open output.wav** no terminal), deve ser duas vezes mais alto que **input.wav**!

```
$./volume input.wav output.wav 0.5
```

Quando você ouve **output.wav** , deve ter metade do volume de **input.wav** !

#### Não sabe como resolver?

Execute o seguinte para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/volume
```

Execute o seguinte para avaliar o estilo do seu código usando **style50**

```
style50 volume.c
```

## EXERCÍCIOS MÓDULO 4

### Exercício 1 - Filtro(versão fácil)

Implemente um programa que aplique filtros a BMPs, conforme a seguir.

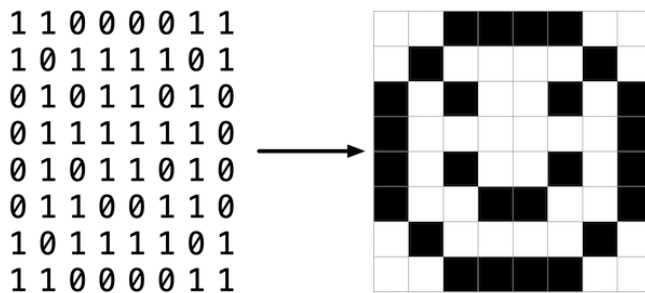
```
$./filter -r image.bmp reflected.bmp
```

#### Background

##### Bitmaps

Talvez a maneira mais simples de representar uma imagem seja com uma grade de pixels (ou seja, pontos), cada um dos quais pode ser de uma cor diferente. Para imagens em preto e branco,

precisamos, portanto, de 1 bit por pixel, já que 0 pode representar preto e 1 pode representar branco, como mostrado a seguir.



Nesse sentido, então, uma imagem é apenas um bitmap (ou seja, um mapa de bits). Para imagens mais coloridas, você simplesmente precisa de mais bits por pixel. Um formato de arquivo (como BMP, JPEG ou PNG) que suporta “cores de 24 bits” usa 24 bits por pixel. (BMP, na verdade, suporta cores de 1, 4, 8, 16, 24 e 32 bits.)

Um BMP de 24 bits usa 8 bits para significar a quantidade de vermelho na cor de um pixel, 8 bits para significar a quantidade de verde na cor de um pixel e 8 bits para significar a quantidade de azul na cor de um pixel. Se você já ouviu falar em cores RGB, bem, aí está: vermelho, verde, azul.

Se os valores R, G e B de algum pixel em um BMP são, digamos, 0xff, 0x00 e 0x00 em hexadecimal, esse pixel é puramente vermelho, pois 0xff (também conhecido como 255 em decimal) implica “muito vermelho”, “Enquanto 0x00 e 0x00 implicam” sem verde “e” sem azul”, respectivamente.

#### Um Bit(map) Mais Técnico

Lembre-se de que um arquivo é apenas uma sequência de bits, organizados de alguma forma. Um arquivo BMP de 24 bits, então, é essencialmente apenas uma sequência de bits, (quase) cada 24 dos quais representam a cor de algum pixel. Mas um arquivo BMP também contém alguns “metadados”, informações como a altura e largura de uma imagem. Esses metadados são armazenados no início do arquivo na forma de duas estruturas de dados geralmente chamadas de “cabeçalhos”, não devem ser confundidos com os arquivos de cabeçalho de C. (Aliás, esses cabeçalhos evoluíram com o tempo. Esse problema usa a versão mais recente do formato BMP da Microsoft, 4.0, que estreou com o Windows 95.)

O primeiro desses cabeçalhos, chamado BITMAPFILEHEADER, tem 14 bytes de comprimento. (Lembre-se de que 1 byte é igual a 8 bits.) O segundo desses cabeçalhos, chamado BITMAPINFOHEADER, tem 40 bytes de comprimento. Imediatamente após esses cabeçalhos está o bitmap real: uma matriz de bytes, triplos dos quais representam a cor de um pixel. No entanto, o BMP armazena esses triplos ao contrário (ou seja, como BGR), com 8 bits para o azul, seguidos por 8 bits para o verde, seguidos por 8 bits para o vermelho. (Alguns BMPs também armazenam todo o bitmap de trás para frente, com a linha superior de uma imagem no final do arquivo BMP. Mas armazenamos os BMPs desse conjunto de problemas conforme descrito aqui, com cada linha superior do bitmap primeiro e a linha inferior por último.) Em outras palavras, se convertêssemos o smiley de 1 bit acima em um smiley de 24 bits, substituindo vermelho por preto, um BMP de 24 bits armazenaria esse bitmap da seguinte maneira, onde 0000ff significa vermelho e ffffff significa branco; destacamos em vermelho todas as ocorrências de 0000ff.

```

ffffff fffffff 0000ff 0000ff 0000ff 0000ff fffffff fffffff
ffffff 0000ff fffffff fffffff fffffff fffffff 0000ff fffffff
0000ff fffffff 0000ff fffffff fffffff 0000ff fffffff 0000ff
0000ff fffffff fffffff fffffff fffffff fffffff fffffff 0000ff
0000ff fffffff 0000ff fffffff fffffff 0000ff fffffff 0000ff
0000ff fffffff fffffff 0000ff 0000ff fffffff fffffff 0000ff
ffffff 0000ff fffffff fffffff fffffff fffffff 0000ff fffffff
ffffff fffffff 0000ff 0000ff 0000ff 0000ff fffffff fffffff

```

Como apresentamos esses bits da esquerda para a direita, de cima para baixo, em 8 colunas, você pode realmente ver o emoticon vermelho se der um passo para trás.

Para ser claro, lembre-se de que um dígito hexadecimal representa 4 bits. Consequentemente, fffffff em hexadecimal realmente significa 11111111111111111111111111111111 em binário.

Observe que você pode representar um bitmap como uma matriz bidimensional de pixels: onde a imagem é uma matriz de linhas, cada linha é uma matriz de pixels. Na verdade, é assim que escolhemos representar as imagens de bitmap neste problema.

### Filtragem de Imagens

O que significa filtrar uma imagem? Você pode pensar em filtrar uma imagem como pegar os pixels de alguma imagem original e modificar cada pixel de forma que um efeito específico seja aparente na imagem resultante.

### Escala de cinza

Um filtro comum é o filtro “escala de cinza”, onde pegamos uma imagem e queremos convertê-la em preto e branco. Como isso funciona?

Lembre-se de que se os valores de vermelho, verde e azul estiverem todos configurados para 0x00 (hexadecimal para 0), o pixel é preto. E se todos os valores forem configurados para 0xff (hexadecimal para 255), o pixel é branco. Contanto que os valores de vermelho, verde e azul sejam todos iguais, o resultado será tons de cinza variados ao longo do espectro preto e branco, com valores mais altos significando tons mais claros (mais perto de branco) e valores mais baixos significando tons mais escuros (mais perto de Preto).

Portanto, para converter um pixel em tons de cinza, só precisamos ter certeza de que os valores de vermelho, verde e azul são todos iguais. Mas como sabemos que valor devemos criá-los? Bem, provavelmente é razoável esperar que, se os valores originais de vermelho, verde e azul fossem todos muito altos, o novo valor também deveria ser muito alto. E se os valores originais fossem todos baixos, o novo valor também deveria ser baixo.

Na verdade, para garantir que cada pixel da nova imagem ainda tenha o mesmo brilho ou escuridão geral da imagem antiga, podemos obter a média dos valores de vermelho, verde e azul para determinar qual tom de cinza deve ser feito no novo pixel.

Se você aplicar isso a cada pixel da imagem, o resultado será uma imagem convertida em tons de cinza.

## Sépia

A maioria dos programas de edição de imagem oferece suporte a um filtro “sépia”, que dá às imagens uma aparência antiga, fazendo com que toda a imagem pareça um pouco marrom-avermelhada.

Uma imagem pode ser convertida em sépia tomando cada pixel e computando novos valores de vermelho, verde e azul com base nos valores originais dos três.

Existem vários algoritmos para converter uma imagem em sépia, mas, para esse problema, pediremos que você use o seguinte algoritmo. Para cada pixel, os valores da cor sépia devem ser calculados com base nos valores da cor original conforme a seguir.

```
sepiaRed = .393 * originalRed + .769 * originalGreen + .189 * originalBlue
sepiaGreen = .349 * originalRed + .686 * originalGreen + .168 *
originalBlue
sepiaBlue = 0,272 * originalRed + 0,534 * originalGreen + 0,131 *
originalBlue
```

Obviamente, o resultado de cada uma dessas fórmulas pode não ser um número inteiro, mas cada valor pode ser arredondado para o número inteiro mais próximo. Também é possível que o resultado da fórmula seja um número maior que 255, o valor máximo para um valor de cor de 8 bits. Nesse caso, os valores de vermelho, verde e azul devem ser limitados a 255. Como resultado, podemos garantir que os valores de vermelho, verde e azul resultantes serão números inteiros entre 0 e 255, inclusive.

## Refletir

Alguns filtros também podem mover pixels. Refletir uma imagem, por exemplo, é um filtro em que a imagem resultante é o que você obterá colocando a imagem original na frente de um espelho. Portanto, quaisquer pixels no lado esquerdo da imagem devem terminar no lado direito e vice-versa.

Observe que todos os pixels originais da imagem original ainda estarão presentes na imagem refletida, mas esses pixels podem ter sido reorganizados para estar em um local diferente na imagem.

## Blur

Existem várias maneiras de criar o efeito de desfocar ou suavizar uma imagem. Para este problema, usaremos o “box blur”, que funciona pegando cada pixel e, para cada valor de cor, dando a ele um novo valor calculando a média dos valores de cor dos pixels vizinhos.

Considere a seguinte grade de pixels, onde numeramos cada pixel.

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

O novo valor de cada pixel seria a média dos valores de todos os pixels que estão dentro de 1 linha e coluna do pixel original (formando uma caixa 3x3). Por exemplo, cada um dos valores de cor para o pixel 6 seria obtido pela média dos valores de cor originais dos pixels 1, 2, 3, 5, 6, 7, 9, 10 e 11

(observe que o próprio pixel 6 está incluído na média). Da mesma forma, os valores de cor para o pixel 11 seriam obtidos pela média dos valores de cor dos pixels 6, 7, 8, 10, 11, 12, 14, 15 e 16.

Para um pixel ao longo da borda ou canto, como o pixel 15, ainda procuraríamos todos os pixels em 1 linha e coluna: neste caso, os pixels 10, 11, 12, 14, 15 e 16.

### Arestas

Em algoritmos de inteligência artificial para processamento de imagens, muitas vezes é útil detectar bordas em uma imagem: linhas na imagem que criam um limite entre um objeto e outro. Uma maneira de obter esse efeito é aplicando o operador Sobel à imagem.

Assim como o desfoque da imagem, a detecção de bordas também funciona pegando cada pixel e modificando-o com base na grade 3x3 de pixels que circunda esse pixel. Mas em vez de apenas tirar a média dos nove pixels, o operador Sobel calcula o novo valor de cada pixel tomando uma soma ponderada dos valores para os pixels circundantes. E como as bordas entre os objetos podem ocorrer nas direções vertical e horizontal, você realmente calculará duas somas ponderadas: uma para detectar bordas na direção x e outra para detectar bordas na direção y. Em particular, você usará os seguintes dois “kernels”:

| Gx |   |   | Gy |    |    |
|----|---|---|----|----|----|
| -1 | 0 | 1 | -1 | -2 | -1 |
| -2 | 0 | 2 | 0  | 0  | 0  |
| -1 | 0 | 1 | 1  | 2  | 1  |

Como interpretar esses kernels? Resumindo, para cada um dos três valores de cor de cada pixel, calcularemos dois valores Gx e Gy. Para calcular Gx para o valor do canal vermelho de um pixel, por exemplo, pegaremos os valores vermelhos originais para os nove pixels que formam uma caixa 3x3 ao redor do pixel, multiplicaremos cada um pelo valor correspondente no kernel Gx e tomaremos a soma dos valores resultantes.

Por que esses valores particulares para o kernel? Na direção Gx, por exemplo, estamos multiplicando os pixels à direita do pixel alvo por um número positivo e multiplicando os pixels à esquerda do pixel alvo por um número negativo. Quando fazemos a soma, se os pixels da direita forem de cor semelhante aos pixels da esquerda, o resultado será próximo de 0 (os números se cancelam). Mas se os pixels da direita forem muito diferentes dos pixels da esquerda, o valor resultante será muito positivo ou muito negativo, indicando uma mudança na cor que provavelmente é o resultado de um limite entre os objetos. E um argumento semelhante é válido para calcular arestas na direção y.

Usando esses kernels, podemos gerar um valor Gx e Gy para cada um dos canais vermelho, verde e azul de um pixel. Mas cada canal pode assumir apenas um valor, não dois: portanto, precisamos de alguma forma para combinar Gx e Gy em um único valor. O algoritmo de filtro de Sobel combina Gx e Gy em um valor final calculando a raiz quadrada de  $Gx^2 + Gy^2$ . E uma vez que os valores do canal só podem assumir valores inteiros de 0 a 255, certifique-se de que o valor resultante seja arredondado para o inteiro mais próximo e limitado a 255!

E quanto ao manuseio de pixels na borda ou no canto da imagem? Existem muitas maneiras de lidar com pixels na borda, mas para os fins deste problema, pediremos que você trate a imagem como se houvesse uma borda preta sólida de 1 pixel ao redor da borda da imagem: portanto, tentando acessar um pixel além da borda da imagem deve ser tratado como um pixel preto sólido (valores de 0 para cada vermelho, verde e azul). Isso irá efetivamente ignorar esses pixels de nossos cálculos de  $G_x$  e  $G_y$ .

### Vamos começar

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute **cd ~** (ou simplesmente **cd** sem argumentos) para garantir que você está em seu diretório pessoal.
- 
- Execute **mkdir pset4** para fazer (ou seja, criar) um diretório chamado **pset4**.
- 
- Execute **cd pset4** para mudar para (ou seja, abrir) esse diretório.
- 
- Execute **wget**  
<https://cdn.cs50.net/2020/fall/psets/4/filter/more/filter.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- 
- Execute **unzip filter.zip** para descompactar esse arquivo.
- 
- Execute **rm filter.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- 
- Execute **ls**. Você deverá ver um diretório chamado **filter**, que estava dentro desse arquivo ZIP.
- 
- Execute **cd filter** para mudar para esse diretório.
- 
- Execute **ls**. Você deve ver a distribuição deste problema, incluindo **bmp.h**, **filter.c**, **helpers.h**, **helpers.c** e **Makefile**. Você também verá um diretório chamado **images**, com algumas imagens de bitmap de amostra.

### Entendendo o problema

Vamos agora dar uma olhada em alguns dos arquivos fornecidos a você como código de distribuição para entender o que há dentro deles.

- 

### **bmp.h**

Abra o **bmp.h** (clcando duas vezes nele no navegador de arquivos) e dê uma olhada.

Você verá as definições dos cabeçalhos que mencionamos (**BITMAPFILEHEADER** e **BITMAPINFOHEADER**). Além disso, esse arquivo define **BYTE**, **DWORD**, **LONG** e **WORD**, tipos de dados normalmente encontrados no mundo da programação do Windows. Observe como eles são apenas apelidos para primitivos com os quais você (com sorte) já está familiarizado. Parece que **BITMAPFILEHEADER** e **BITMAPINFOHEADER** fazem uso desses tipos.

Talvez o mais importante para você, este arquivo também define uma **struct** chamada **RGBTRIPLE** que, simplesmente, "encapsula" três bytes: um azul, um verde e um vermelho (a ordem, lembre-se, em que esperamos encontrar triplos RGB realmente em disco).

Por que essas structs são úteis? Bem, lembre-se de que um arquivo é apenas uma sequência de bytes (ou, no final das contas, bits) no disco. Mas esses bytes são geralmente ordenados de forma que os primeiros representem algo, os próximos representem outra coisa e assim por diante. Os "formatos de arquivo" existem porque o mundo padronizou o significado de bytes. Agora, poderíamos simplesmente ler um arquivo do disco para a RAM como um grande array de bytes. E poderíamos apenas lembrar que o byte em array [i] representa uma coisa, enquanto o byte em array [j] representa outra. Mas por que não dar nomes a alguns desses bytes para que possamos recuperá-los da memória com mais facilidade? Isso é precisamente o que as estruturas em `bmp.h` nos permitem fazer. Em vez de pensar em algum arquivo como uma longa sequência de bytes, podemos pensar nele como uma sequência de structs.

- 

## filter.c

Agora, vamos abrir `filter.c`. Este arquivo já foi escrito para você, mas há alguns pontos importantes que devem ser observados aqui.

Primeiro, observe a definição de `filters` na linha 11. Essa string informa ao programa quais são os argumentos de linha de comando permitidos para o programa: **b**, **g**, **r** e **s**. Cada um deles especifica um filtro diferente que podemos aplicar às nossas imagens: `blur`, `grayscale`, `reflexo` e `sépia`.

As próximas linhas abrem um arquivo de imagem, certifique-se de que é realmente um arquivo BMP e leia todas as informações de pixel em um array 2D chamado **image**.

Role para baixo até a instrução `switch` que começa na linha 102. Observe que, dependendo do *filtro* que escolhemos, uma função diferente é chamada: se o usuário escolher o filtro **b**, o programa chama a função de **blur**; se **e**, a **edge** é chamada; se **g**, a **grayscale** é chamada; se **r**, então **reflect** é chamado. Observe também que cada uma dessas funções toma como argumentos a altura da imagem, a largura da imagem e a matriz 2D de pixels.

Estas são as funções que você (em breve!) implementará. Como você pode imaginar, o objetivo é que cada uma dessas funções edite o array 2D de pixels de forma que o filtro desejado seja aplicado à imagem.

As linhas restantes do programa pegam a `image` resultante e as gravam em um novo arquivo de imagem.

- 

## helpers.h

A seguir, dê uma olhada em **helpers.h**. Este arquivo é bastante curto e fornece apenas os protótipos de funções para as funções que você viu anteriormente.

Aqui, observe o fato de que cada função recebe uma matriz 2D chamada `image` como argumento, onde `image` é uma matriz de altura (`height`) com várias linhas, e cada linha é e la própria outra matriz de `width` (largura) de muitos **RGBTRIPLES**. Portanto, se a `image` representa a imagem inteira, a `image[0]` representa a primeira linha e a `image[0][0]` representa o pixel no canto superior esquerdo da imagem.

- 

## helpers.c

Agora, abra **helpers.c**. É aqui que pertence a implementação das funções declaradas em **helpers.h**. Mas note que, agora, as implementações estão faltando! Esta parte é com você.

- 

## Makefile

Finalmente, vamos dar uma olhada no **Makefile**. Este arquivo especifica o que deve acontecer quando executamos um comando de terminal como **make filter**. Enquanto os programas que você pode ter escrito antes estavam confinados a apenas um arquivo, o `filter` parece usar vários arquivos: **filter.c**, **bmp.h**, **helpers.h** e **helpers.c**. Então, vamos precisar dizer ao `make` como compilar este arquivo.

Tente compilar o `filter` para você mesmo indo para o seu terminal e executando

```
$ make filter
```

Em seguida, você pode executar o programa executando:

```
$./filter -g images/yard.bmp out.bmp
```

que obtém a imagem em `images/yard.bmp` e gera uma nova imagem chamada **out.bmp** após executar os pixels por meio da função de **grayscale**. A **grayscale** ainda não faz nada, portanto, a imagem de saída deve ter a mesma aparência do original.

### Especificação

Implemente as funções em `helpers.c` de forma que um usuário possa aplicar filtros de `grayscale`, `sépie`, `refletir` ou `blur` às suas imagens.



- A função **grayscale** deve pegar uma imagem e transformá-la em uma versão em preto e branco da mesma imagem.
- 
- A função de **reflect** deve pegar uma imagem e refleti-la horizontalmente.
- 
- A função de **blur** deve pegar uma imagem e transformá-la em uma versão desfocada da mesma imagem.
- 
- A função de **edges** deve pegar uma imagem e destacar as bordas entre os objetos, de acordo com o operador Sobel.

Você não deve modificar nenhuma das assinaturas de função, nem deve modificar nenhum outro arquivo além de helpers.c.

### Uso

Seu programa deve se comportar de acordo com os exemplos abaixo.

```
$./filter -g infile.bmp outfile.bmp
```

```
$./filter -r infile.bmp outfile.bmp
```

```
$./filter -b infile.bmp outfile.bmp
```

```
$./filter -e infile.bmp outfile.bmp
```

### Dicas

Os valores dos componentes `rgbRed`, `rgbGreen` e `rgbBlue` de um pixel são todos inteiros, então certifique-se de arredondar quaisquer números de ponto flutuante para o inteiro mais próximo ao atribuí-los a um valor de pixel!

### Testando seu código

Certifique-se de testar todos os seus filtros nos arquivos de bitmap de amostra fornecidos!

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/filter/more
```

Execute o seguinte para avaliar o estilo do seu código usando **style50**.

```
style50 helpers.c
```

### Exercício 2 - Filtro(versão desafiadora)

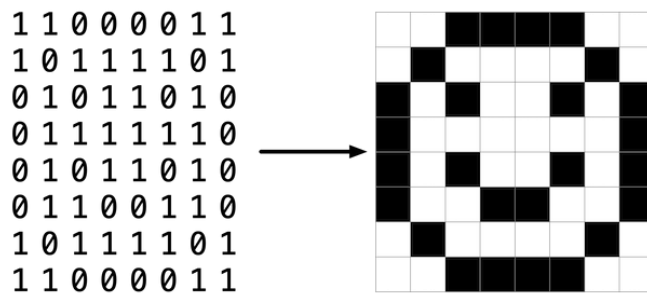
Implemente um programa que aplique filtros a BMPs, conforme a seguir.

```
$./filter -r image.bmp reflected.bmp
```

### Background

#### Bitmaps

Talvez a maneira mais simples de representar uma imagem seja com uma grade de pixels (ou seja, pontos), cada um dos quais pode ser de uma cor diferente. Para imagens em preto e branco, precisamos, portanto, de 1 bit por pixel, já que 0 pode representar preto e 1 pode representar branco, como mostrado a seguir.



Nesse sentido, então, uma imagem é apenas um bitmap (ou seja, um mapa de bits). Para imagens mais coloridas, você simplesmente precisa de mais bits por pixel. Um formato de arquivo (como BMP, JPEG ou PNG) que suporta “cores de 24 bits” usa 24 bits por pixel. (BMP, na verdade, suporta cores de 1, 4, 8, 16, 24 e 32 bits.)

Um BMP de 24 bits usa 8 bits para significar a quantidade de vermelho na cor de um pixel, 8 bits para significar a quantidade de verde na cor de um pixel e 8 bits para significar a quantidade de azul na cor de um pixel. Se você já ouviu falar em cores RGB, bem, aí está: vermelho, verde, azul.

Se os valores R, G e B de algum pixel em um BMP são, digamos, 0xff, 0x00 e 0x00 em hexadecimal, esse pixel é puramente vermelho, pois 0xff (também conhecido como 255 em decimal) implica “muito vermelho”, enquanto 0x00 e 0x00 implicam “sem verde” e “sem azul”, respectivamente.

#### Um Bit(map) Mais Técnico

Lembre-se de que um arquivo é apenas uma sequência de bits, organizados de alguma forma. Um arquivo BMP de 24 bits, então, é essencialmente apenas uma sequência de bits, (quase) cada 24 dos quais representam a cor de algum pixel. Mas um arquivo BMP também contém alguns “metadados”, informações como a altura e largura de uma imagem. Esses metadados são armazenados no início do arquivo na forma de duas estruturas de dados geralmente chamadas de “cabeçalhos”, não devem ser confundidos com os arquivos de cabeçalho de C. (Aliás, esses cabeçalhos evoluíram com o tempo. Esse problema usa a versão mais recente do formato BMP da Microsoft, 4.0, que estreou com o Windows 95.)

O primeiro desses cabeçalhos, chamado BITMAPFILEHEADER, tem 14 bytes de comprimento. (Lembre-se de que 1 byte é igual a 8 bits.) O segundo desses cabeçalhos, chamado BITMAPINFOHEADER, tem 40 bytes de comprimento. Imediatamente após esses cabeçalhos está o bitmap real: uma matriz de bytes, triplos dos quais representam a cor de um pixel. No entanto, o BMP armazena esses triplos ao contrário (ou seja, como BGR), com 8 bits para o azul, seguidos por 8 bits para o verde, seguidos por 8 bits para o vermelho. (Alguns BMPs também armazenam todo o bitmap de trás para frente, com a linha superior de uma imagem no final do arquivo BMP. Mas armazenamos os BMPs desse conjunto de problemas conforme descrito aqui, com cada linha superior do bitmap primeiro e a linha inferior por último.) Em outras palavras, se convertêssemos o smiley de 1 bit acima em um smiley de 24 bits, substituindo vermelho por preto, um BMP de 24 bits armazenaria esse bitmap da seguinte maneira, onde 0000ff significa vermelho e ffffff significa branco; destacamos em vermelho todas as ocorrências de 0000ff.

```

ffffff fffffff 0000ff 0000ff 0000ff 0000ff fffffff fffffff
ffffff 0000ff fffffff fffffff fffffff fffffff 0000ff fffffff
0000ff fffffff 0000ff fffffff fffffff 0000ff fffffff 0000ff
0000ff fffffff fffffff fffffff fffffff fffffff fffffff 0000ff
0000ff fffffff 0000ff fffffff fffffff 0000ff fffffff 0000ff
0000ff fffffff fffffff 0000ff 0000ff fffffff fffffff 0000ff
ffffff 0000ff fffffff fffffff fffffff fffffff 0000ff fffffff
ffffff fffffff 0000ff 0000ff 0000ff 0000ff fffffff fffffff

```

Como apresentamos esses bits da esquerda para a direita, de cima para baixo, em 8 colunas, você pode realmente ver o emoticon vermelho se der um passo para trás.

Para ser claro, lembre-se de que um dígito hexadecimal representa 4 bits. Consequentemente, fffffff em hexadecimal realmente significa 11111111111111111111111111111111 em binário.

Observe que você pode representar um bitmap como uma matriz bidimensional de pixels: onde a imagem é uma matriz de linhas, cada linha é uma matriz de pixels. Na verdade, é assim que escolhemos representar as imagens de bitmap neste problema.

### Filtragem de Imagens

O que significa filtrar uma imagem? Você pode pensar em filtrar uma imagem como pegar os pixels de alguma imagem original e modificar cada pixel de forma que um efeito específico seja aparente na imagem resultante.

### Escala de cinza

Um filtro comum é o filtro “escala de cinza”, onde pegamos uma imagem e queremos convertê-la em preto e branco. Como isso funciona?

Lembre-se de que se os valores de vermelho, verde e azul estiverem todos configurados para 0x00 (hexadecimal para 0), o pixel é preto. E se todos os valores forem configurados para 0xff (hexadecimal para 255), o pixel é branco. Contanto que os valores de vermelho, verde e azul sejam todos iguais, o resultado será tons de cinza variados ao longo do espectro preto e branco, com valores mais altos significando tons mais claros (mais perto de branco) e valores mais baixos significando tons mais escuros (mais perto de Preto).

Portanto, para converter um pixel em tons de cinza, só precisamos ter certeza de que os valores de vermelho, verde e azul são todos iguais. Mas como sabemos que valor devemos criá-los? Bem, provavelmente é razoável esperar que, se os valores originais de vermelho, verde e azul fossem todos muito altos, o novo valor também deveria ser muito alto. E se os valores originais fossem todos baixos, o novo valor também deveria ser baixo.

Na verdade, para garantir que cada pixel da nova imagem ainda tenha o mesmo brilho ou escuridão geral da imagem antiga, podemos obter a média dos valores de vermelho, verde e azul para determinar qual tom de cinza deve ser feito no novo pixel.

Se você aplicar isso a cada pixel da imagem, o resultado será uma imagem convertida em tons de cinza.

## Sépia

A maioria dos programas de edição de imagem oferece suporte a um filtro “sépia”, que dá às imagens uma aparência antiga, fazendo com que toda a imagem pareça um pouco marrom-avermelhada.

Uma imagem pode ser convertida em sépia tomando cada pixel e computando novos valores de vermelho, verde e azul com base nos valores originais dos três.

Existem vários algoritmos para converter uma imagem em sépia, mas, para esse problema, pediremos que você use o seguinte algoritmo. Para cada pixel, os valores da cor sépia devem ser calculados com base nos valores da cor original conforme a seguir.

```
sepiaRed = .393 * originalRed + .769 * originalGreen + .189 * originalBlue
sepiaGreen = .349 * originalRed + .686 * originalGreen + .168 *
originalBlue
sepiaBlue = 0,272 * originalRed + 0,534 * originalGreen + 0,131 *
originalBlue
```

Obviamente, o resultado de cada uma dessas fórmulas pode não ser um número inteiro, mas cada valor pode ser arredondado para o número inteiro mais próximo. Também é possível que o resultado da fórmula seja um número maior que 255, o valor máximo para um valor de cor de 8 bits. Nesse caso, os valores de vermelho, verde e azul devem ser limitados a 255. Como resultado, podemos garantir que os valores de vermelho, verde e azul resultantes serão números inteiros entre 0 e 255, inclusive.

## Refletir

Alguns filtros também podem mover pixels. Refletir uma imagem, por exemplo, é um filtro em que a imagem resultante é o que você obterá colocando a imagem original na frente de um espelho. Portanto, quaisquer pixels no lado esquerdo da imagem devem terminar no lado direito e vice-versa.

Observe que todos os pixels originais da imagem original ainda estarão presentes na imagem refletida, mas esses pixels podem ter sido reorganizados para estar em um local diferente na imagem.

## Blur

Existem várias maneiras de criar o efeito de desfocar ou suavizar uma imagem. Para este problema, usaremos o “box blur”, que funciona pegando cada pixel e, para cada valor de cor, dando a ele um novo valor calculando a média dos valores de cor dos pixels vizinhos.

Considere a seguinte grade de pixels, onde numeramos cada pixel.

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

O novo valor de cada pixel seria a média dos valores de todos os pixels que estão dentro de 1 linha e coluna do pixel original (formando uma caixa 3x3). Por exemplo, cada um dos valores de cor para o pixel 6 seria obtido pela média dos valores de cor originais dos pixels 1, 2, 3, 5, 6, 7, 9, 10 e 11 (observe que o próprio pixel 6 está incluído no média). Da mesma forma, os valores de cor para o pixel 11 seriam obtidos pela média dos valores de cor dos pixels 6, 7, 8, 10, 11, 12, 14, 15 e 16.

Para um pixel ao longo da borda ou canto, como o pixel 15, ainda procuraríamos todos os pixels em 1 linha e coluna: neste caso, os pixels 10, 11, 12, 14, 15 e 16.

#### Começando

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute **cd ~** (ou simplesmente **cd** sem argumentos) para garantir que você está em seu diretório pessoal.
- 
- Execute **mkdir pset4** para fazer (ou seja, criar) um diretório chamado pset4 .
- 
- Execute **cd pset4** para mudar para (ou seja, abrir) esse diretório.
- 
- Execute **wget**  
<https://cdn.cs50.net/2020/fall/psets/4/filter/less/filter.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- 
- Execute **unzip filter.zip** para descompactar esse arquivo.
- 
- Execute **rm filter.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- 
- Execute **ls**. Você deverá ver um diretório chamado **filter** , que estava dentro desse arquivo ZIP.
- 
- Execute **cd filter** para mudar para esse diretório.
-

- Execute **ls**. Você deve ver a distribuição deste problema, incluindo `bmp.h` , `filter.c` , `helpers.h` , `helpers.c` e `Makefile` . Você também verá um diretório chamado `images` , com algumas imagens de bitmap de amostra.

### Entendendo o problema :)

Vamos agora dar uma olhada em alguns dos arquivos fornecidos a você como código de distribuição para entender o que há dentro deles.

- 

### `bmp.h`

Abra o `bmp.h` (clcando duas vezes nele no navegador de arquivos) e dê uma olhada.

Você verá as definições dos cabeçalhos que mencionamos (**`BITMAPFILEHEADER`** e **`BITMAPINFOHEADER`**). Além disso, esse arquivo define `BYTE` , `DWORD` , `LONG` e `WORD` , tipos de dados normalmente encontrados no mundo da programação do Windows. Observe como eles são apenas apelidos para primitivos com os quais você (com sorte) já está familiarizado. Parece que **`BITMAPFILEHEADER`** e **`BITMAPINFOHEADER`** fazem uso desses tipos.

Talvez o mais importante para você, este arquivo também define uma **`struct`** chamada **`RGBTRIPLE`** que, simplesmente, "encapsula" três bytes: um azul, um verde e um vermelho (a ordem, lembre-se, em que esperamos encontrar triplos RGB realmente em disco).

Por que essas structs são úteis? Bem, lembre-se de que um arquivo é apenas uma sequência de bytes (ou, no final das contas, bits) no disco. Mas esses bytes são geralmente ordenados de forma que os primeiros representem algo, os próximos representem outra coisa e assim por diante. Os “formatos de arquivo” existem porque o mundo padronizou o significado de bytes. Agora, poderíamos simplesmente ler um arquivo do disco para a RAM como um grande array de bytes. E poderíamos apenas lembrar que o byte em array `[i]` representa uma coisa, enquanto o byte em array `[j]` representa outra. Mas por que não dar nomes a alguns desses bytes para que possamos recuperá-los da memória com mais facilidade? Isso é precisamente o que as estruturas em `bmp.h` nos permitem fazer. Em vez de pensar em algum arquivo como uma longa sequência de bytes, podemos pensar nele como uma sequência de structs.

- 

### `filter.c`

Agora, vamos abrir `filter.c` . Este arquivo já foi escrito para você, mas há alguns pontos importantes que devem ser observados aqui.

Primeiro, observe a definição de `filters` na linha 11. Essa string informa ao programa quais são os argumentos de linha de comando permitidos para o programa: **`b`** , **`g`** , **`r`** e **`s`**. Cada um deles especifica um filtro diferente que podemos aplicar às nossas imagens: `blur` , `grayscale` , `reflexo` e `sépia`.

As próximas linhas abrem um arquivo de imagem, certifique-se de que é realmente um arquivo BMP e leia todas as informações de pixel em um array 2D chamado **`image`**.

Role para baixo até a instrução switch que começa na linha 102. Observe que, dependendo do *filtro* que escolhermos, uma função diferente é chamada: se o usuário escolher o filtro *b*, o programa chama a função de **blur**; se *g*, a **grayscale** é chamada; se *r*, então **reflect** é chamado; e se *s*, **sepia** é chamado. Observe também que cada uma dessas funções toma como argumentos a altura da imagem, a largura da imagem e a matriz 2D de pixels.

Estas são as funções que você (em breve!) implementará. Como você pode imaginar, o objetivo é que cada uma dessas funções edite o array 2D de pixels de forma que o filtro desejado seja aplicado à imagem.

As linhas restantes do programa pegam a image resultante e as gravam em um novo arquivo de imagem.

- 

### helpers.h

A seguir, dê uma olhada em **helpers.h**. Este arquivo é bastante curto e fornece apenas os protótipos de funções para as funções que você viu anteriormente.

Aqui, observe o fato de que cada função recebe uma matriz 2D chamada *image* como argumento, onde *image* é uma matriz de altura (*height*) com várias linhas, e cada linha é ela própria outra matriz de *width* (*largura*) de muitos **RGBTRIPLES**. Portanto, se a *image* representa a imagem inteira, a *image[0]* representa a primeira linha e a *image[0][0]* representa o pixel no canto superior esquerdo da imagem.

- 

### helpers.c

Agora, abra *helpers.c*. É aqui que pertence a implementação das funções declaradas em *helpers.h*. Mas note que, agora, as implementações estão faltando! Esta parte é com você.

- 

### Makefile

Finalmente, vamos dar uma olhada no *Makefile*. Este arquivo especifica o que deve acontecer quando executamos um comando de terminal como **make filter**. Enquanto os programas que você pode ter escrito antes estavam confinados a apenas um arquivo, o *filter* parece usar vários arquivos: **filter.c**, **bmp.h**, **helpers.h** e **helpers.c**. Então, vamos precisar dizer ao *make* como compilar este arquivo.

Tente compilar o *filter* para você mesmo indo para o seu terminal e executando

```
$ make filter
```

Em seguida, você pode executar o programa executando:

```
$./filter -g images/yard.bmp out.bmp
```

que obtém a imagem em `images/yard.bmp` e gera uma nova imagem chamada `out.bmp` após executar os pixels por meio da função de **grayscale**. A grayscale ainda não faz nada, portanto, a imagem de saída deve ter a mesma aparência do original.

### Especificação

Implemente as funções em `helpers.c` de forma que um usuário possa aplicar filtros de grayscale, sépia, refletir ou blur às suas imagens.

- A função **grayscale** deve pegar uma imagem e transformá-la em uma versão em preto e branco da mesma imagem.
- 
- A função **sepia** deve pegar uma imagem e transformá-la em uma versão sépia da mesma imagem.
- 
- A função de **reflect** deve pegar uma imagem e refleti-la horizontalmente.
- 
- Finalmente, a função de **blur** deve pegar uma imagem e transformá-la em uma versão desfocada da mesma imagem.

Você não deve modificar nenhuma das assinaturas de função, nem deve modificar nenhum outro arquivo além de `helpers.c`.

### Uso

Seu programa deve se comportar de acordo com os exemplos abaixo.

```
$./filter -g infile.bmp outfile.bmp
```

```
$./filter -s infile.bmp outfile.bmp
```

```
$./filter -r infile.bmp outfile.bmp
```

```
$./filter -b infile.bmp outfile.bmp
```

### Dicas

Os valores dos componentes **rgbtRed**, **rgbtGreen** e **rgbtBlue** de um pixel são todos inteiros, então certifique-se de arredondar quaisquer números de ponto flutuante para o inteiro mais próximo ao atribuí-los a um valor de pixel!

### Testando

Certifique-se de testar todos os seus filtros nos arquivos de bitmap de amostra fornecidos!

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/filter/less
```

Execute o seguinte para avaliar o estilo do seu código usando **style50**.

```
style50 helpers.c
```



### Exercício 3 - Recover

Implemente um programa que recupere JPEGs de uma imagem forense, conforme a seguir.

```
$./recover card.raw
```

#### Background

Prevendo esse problema, passamos os últimos dias tirando fotos de pessoas que conhecemos, todas as quais foram salvas em uma câmera digital como JPEG em um cartão de memória. (Ok, é possível que, em vez disso, tenhamos passado os últimos dias no Facebook.) Infelizmente, de alguma forma excluímos todos eles! Felizmente, no mundo da informática, "excluído" tende a não significar "excluído", mas sim "esquecido". Mesmo que a câmera insista que o cartão agora está em branco, temos certeza de que isso não é bem verdade. Na verdade, estamos torcendo (er, esperando!) Que você possa escrever um programa que recupere as fotos para nós!

Mesmo que JPEGs sejam mais complicados do que BMPs, JPEGs têm “assinaturas”, padrões de bytes que podem distingui-los de outros formatos de arquivo. Especificamente, os primeiros três bytes de JPEGs são

`0xff 0xd8 0xff`

do primeiro para o terceiro byte, da esquerda para a direita. O quarto byte, entretanto, é `0xe0` , `0xe1` , `0xe2` , `0xe3` , `0xe4` , `0xe5` , `0xe6` , `0xe7` , `0xe8` , `0xe9` , `0xea` , `0xeb` , `0xec` , `0xed` , `0xee` ou `0xef` . Dito de outra forma, os primeiros quatro bits do quarto byte são 1110 .

As probabilidades são de que, se você encontrar esse padrão de quatro bytes na mídia conhecida por armazenar fotos (por exemplo, meu cartão de memória), eles demarcam o início de um JPEG. Para ser justo, você pode encontrar esses padrões em algum disco puramente por acaso, então a recuperação de dados não é uma ciência exata.

Felizmente, as câmeras digitais tendem a armazenar fotos de forma contígua em cartões de memória, sendo que cada foto é armazenada imediatamente após a foto anterior. Da mesma forma, o início de um JPEG geralmente marca o final de outro. No entanto, as câmeras digitais geralmente inicializam os cartões com um sistema de arquivos FAT cujo “tamanho de bloco” é de 512 bytes (B). A implicação é que essas câmeras só gravam nesses cartões em unidades de 512 B. Uma foto com 1 MB (ou seja, 1.048.576 B) ocupa  $1048576 \div 512 = 2048$  “blocos” em um cartão de memória. Mas o mesmo acontece com uma foto que é, digamos, um byte menor (ou seja, 1.048.575 B)! O espaço desperdiçado em disco é chamado de “espaço livre”. Os investigadores forenses costumam olhar para a folga em busca de resquícios de dados suspeitos.

A implicação de todos esses detalhes é que você, o investigador, provavelmente pode escrever um programa que itera sobre uma cópia do meu cartão de memória, procurando assinaturas de JPEGs. Cada vez que encontrar uma assinatura, você pode abrir um novo arquivo para escrever e começar a preencher esse arquivo com bytes do meu cartão de memória, fechando esse arquivo apenas quando encontrar outra assinatura. Além disso, em vez de ler os bytes do meu cartão de memória um de cada vez, você pode ler 512 deles por vez em um buffer para fins de eficiência. Graças ao FAT, você pode confiar que as assinaturas de JPEGs serão “alinhadas em bloco”. Ou seja, você só precisa procurar essas assinaturas nos primeiros quatro bytes de um bloco.

Perceba, é claro, que JPEGs podem abranger blocos contíguos. Caso contrário, nenhum JPEG pode ser maior que 512 B. Mas o último byte de um JPEG pode não cair no final de um bloco. Lembre-se da possibilidade de espaço livre. Mas não se preocupe. Como este cartão de memória era novo quando comecei a tirar fotos, é provável que tenha sido “zerado” (ou seja, preenchido com 0s) pelo fabricante, caso em que qualquer espaço livre será preenchido com 0s. Tudo bem se esses 0s finais acabarem nos JPEGs que você recuperar; eles ainda devem estar visíveis.

Agora, só tenho um cartão de memória, mas são muitos de vocês! E então fui em frente e criei uma “imagem forense” do cartão, armazenando seu conteúdo, byte após byte, em um arquivo chamado **card.raw**. Para que você não perca tempo repetindo milhões de zeros desnecessariamente, imaginei apenas os primeiros megabytes do cartão de memória. Mas você deve descobrir que a imagem contém 50 JPEGs.

### Vamos começar

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Navegue até o diretório **pset4** que já deve existir.
- 
- Execute **wget** <http://cdn.cs50.net/2020/fall/psets/4/recover/recover.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- 
- Execute **unzip recover.zip** para descompactar esse arquivo.
- 
- Execute **rm recover.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- 
- Execute **ls**. Você deve ver um diretório chamado **recover**, que estava dentro desse arquivo ZIP.
- 
- Execute **cd recover** para mudar para esse diretório.
- 
- Execute **ls**. Você deve ver a distribuição deste problema, incluindo **card.raw** e **recover.c**.

### Especificação

Implemente um programa denominado **recover** que recupera JPEGs de uma imagem forense.

- Implemente seu programa em um arquivo chamado **recover.c** em um diretório chamado **recover**.
- 
- Seu programa deve aceitar exatamente um argumento de linha de comando, o nome de uma imagem forense da qual recuperar JPEGs.
- 
- Se seu programa não for executado com exatamente um argumento de linha de comando, ele deve lembrar o usuário do uso correto e **main** deve retornar **1**.
-

- Se a imagem forense não puder ser aberta para leitura, seu programa deve informar isso ao usuário, e **main** deve retornar **1**.
- 
- Cada um dos arquivos gerados deve ser nomeado como **###.jpg**, onde **###** é um número decimal de três dígitos, começando com 000 para a primeira imagem e aumentando.
- 
- Seu programa, se usar **malloc**, não deve perder memória.

### Uso

Seu programa deve se comportar de acordo com os exemplos abaixo.

```
$./recover
```

Uso: ./recuperar imagem

```
$./recover card.raw
```

### Dicas

Lembre-se de que você pode abrir **card.raw** programaticamente com **fopen**, como abaixo, desde que **argv[1]** exista.

```
FILE *file = fopen(argv[1], "r");
```

Quando executado, seu programa deve recuperar cada um dos JPEGs de **card.raw**, armazenando cada um como um arquivo separado em seu diretório de trabalho atual. Seu programa deve numerar os arquivos de saída nomeando cada **###.jpg**, onde **###** é um número decimal de três dígitos de **000** em diante. (Vire amigo do [sprintf](#).) Você não precisa tentar recuperar os nomes originais dos JPEGs. Para verificar se os JPEGs que seu programa emitiu estão corretos, basta clicar duas vezes e dar uma olhada! Se cada foto aparecer intacta, sua operação provavelmente foi um sucesso!

As probabilidades são, no entanto, os JPEGs que o primeiro rascunho de seu código divulga não estarão corretos. (Se você abri-los e não ver nada, provavelmente eles não estão corretos!) Execute o comando abaixo para excluir todos os JPEGs em seu diretório de trabalho atual.

```
$ rm * .jpg
```

Se você preferir não ser solicitado a confirmar cada exclusão, execute o comando abaixo.

```
$ rm -f * .jpg
```

Apenas tome cuidado com a opção **-f**, pois ela “força” a exclusão sem avisar você.

Se você gostaria de criar um novo tipo para armazenar um byte de dados, você pode fazer isso por meio do comando abaixo, que define um novo tipo chamado **BYTE** como **uint8\_t** (um tipo definido em **stdint.h**, representando um inteiro sem sinal).

```
typedef uint8_t BYTE;
```

Lembre-se também de que você pode ler dados de um arquivo usando o [fread](#), que lerá os dados de um arquivo em um local da memória e retornará o número de itens lidos com êxito do arquivo.

### Testando

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/recover

Execute o seguinte para avaliar o estilo do seu código usando **style50**.

















style50 recover.c

## ANOTAÇÕES MÓDULO 5

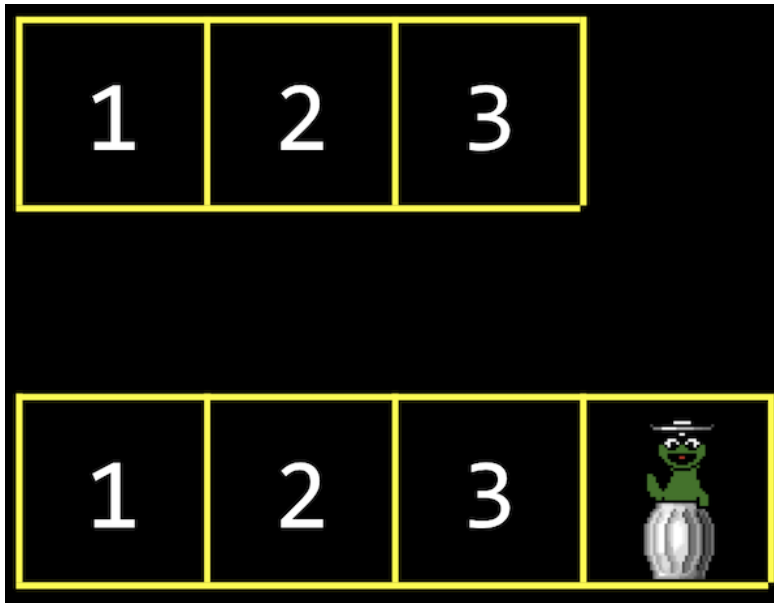
### Redimensionamento de matrizes

Da última vez, aprendemos sobre ponteiros, **malloc** e outras ferramentas úteis para trabalhar com memória.

Na semana 2, aprendemos sobre matrizes, onde poderíamos armazenar o mesmo tipo de valor em uma lista, consecutivamente na memória. Quando precisamos inserir um elemento, precisamos aumentar o tamanho do array também. Mas, o espaço da memória ao lado desse array em nosso computador já pode estar em uso para alguns outros dados, como uma string:

|                                                                                    |                                                                                     |                                                                                     |                                                                                     |                                                                                     |                                                                                      |                                                                                       |                                                                                       |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
|   |    |    |    |    |    |    |    |
|  | 1                                                                                   | 2                                                                                   | 3                                                                                   | h                                                                                   | e                                                                                    | l                                                                                     | l                                                                                     |
| o                                                                                  | ,                                                                                   |                                                                                     | w                                                                                   | o                                                                                   | r                                                                                    | l                                                                                     | d                                                                                     |
| \0                                                                                 |  |  |  |  |  |  |  |

Uma solução pode ser alocar mais memória onde houver espaço suficiente e mover nosso array para lá. Mas precisaremos copiar nosso array lá, o que se torna uma operação com tempo de execução de  $O(n)$ , já que precisamos copiar cada um dos  $n$  elementos originais primeiro:



- 

O limite inferior da inserção de um elemento em um array seria  $O(1)$ , pois já podemos ter espaço para ele no array.

### Estruturas de dados

As **estruturas de dados** são formas mais complexas de organizar os dados na memória, permitindo-nos armazenar informações em diferentes layouts.

Para construir uma estrutura de dados, vamos precisar de algumas ferramentas:

- **struct** para criar tipos de dados personalizados
- 
- . para acessar propriedades em uma estrutura
- 
- \* para ir para um endereço na memória apontado por um ponteiro
- 
- -> para acessar propriedades em uma estrutura apontada por um ponteiro

### Linked lists

Com uma linked list (ou lista encadeada), podemos armazenar uma lista de valores que pode ser facilmente aumentada, armazenando valores em diferentes partes da memória:

- 

- Temos os valores **1**, **2** e **3**, cada um em algum endereço na memória como **0x123**, **0x456** e **0x789**.

- 
- Isso é diferente de uma matriz, pois nossos valores não estão mais próximos um do outro na memória. Podemos usar quaisquer locais da memória que estejam livres.

Para rastrear todos esses valores, precisamos vincular nossa lista, alocando, para cada elemento, memória suficiente para o valor que queremos armazenar e o endereço do próximo elemento:

- 
- Próximo ao nosso valor de **1**, por exemplo, também armazenamos um ponteiro, **0x456**, para o próximo valor. Chamaremos isso de **node (ou nó)**, um componente de nossa estrutura de dados que armazena um valor e um ponteiro. Em C, implementaremos nossos nós com um struct.
- 
- Para nosso último nó com valor **3**, temos o ponteiro nulo, já que não há próximo elemento. Quando precisamos inserir outro nó, podemos apenas alterar aquele único ponteiro nulo para apontar para nosso novo valor.

Temos a desvantagem de precisar alocar o dobro de memória para cada elemento, a fim de gastar menos tempo adicionando valores. E não podemos mais usar a pesquisa binária, uma vez que nossos nós podem estar em qualquer lugar da memória. Só podemos acessá-los seguindo os ponteiros, um de cada vez.

No código, podemos criar nossa própria estrutura chamada **node** e precisamos armazenar tanto nosso valor, um **int** chamado **number**, quanto um ponteiro para o próximo **node**, chamado **next**:

```
typedef struct node
{
 int number;
 struct node *next;
}
node;
```

Iniciamos este com **typedef struct node** para que possamos nos referir a um **node** dentro de nosso struct.

Podemos construir uma lista vinculada no código começando com nosso struct. Primeiro, queremos lembrar de uma lista vazia, para que possamos usar o ponteiro nulo: **node \*list = NULL;**

Para adicionar um elemento, primeiro precisamos alocar um pouco de memória para um nó e definir seus valores:

```
// Usamos sizeof (node) para obter a quantidade certa de memória para
alocar, e
// malloc retorna um ponteiro que salvamos como
node *n = malloc(sizeof(node));

// Queremos ter certeza de que malloc conseguiu obter memória para nós
if(n != NULL)
{
 // Isso é equivalente a (*n).number, onde primeiro vamos para o nó
```

apontado

```
// para por n e, em seguida, defina a propriedade number. Em C,
também podemos usar este
// notação de seta
n->number = 1
```

```
// Então precisamos ter certeza de que o ponteiro para o próximo nó
em nossa lista
// não é um valor lixo, mas o novo nó não apontará para nada (por
enquanto)
n->next = NULL;
}
```

Agora nossa lista precisa apontar para este nó: **list = n;**

•

Para adicionar à lista, criaremos um novo nó da mesma maneira, alocando mais memória:

```
n = malloc(sizeof(node));
```

```
if(n != NULL)
{
 n->number = 2;
 n->next = NULL;
}
```

Mas agora precisamos atualizar o ponteiro em nosso primeiro nó para apontar para nosso novo **n**:

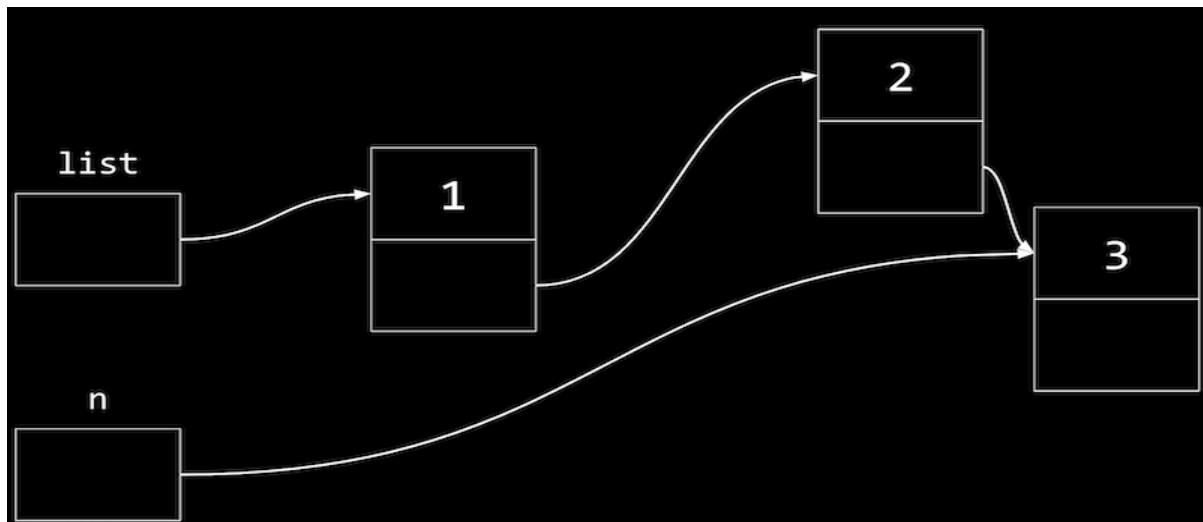
```
list->next = n;
```

Para adicionar um terceiro nó, nós vamos fazer o mesmo, seguindo o ponteiro **next** na nossa lista em primeiro lugar, em seguida, definir o ponteiro **next** para lá para apontar para o novo nó:

```
n = malloc(sizeof(node));
```

```
if(n != NULL)
{
 n->number = 3;
 n->next = NULL;
}
list->next->next = n;
```

Graficamente, nossos nós na memória se parecem com isto:



- **n** é uma variável temporária, apontando para nosso novo nó com valor 3.
- 
- Queremos que o ponteiro em nosso nó com valor 2 aponte para o novo nó também, então começamos da **list**(que aponta para o nó com valor 1), seguimos o ponteiro **next** para chegar ao nosso nó com valor 2 e atualizamos o ponteiro **next** para apontar para **n**.

Como resultado, pesquisar uma lista encadeada também terá um tempo de execução de  $O(n)$ , uma vez que precisamos olhar todos os elementos em ordem seguindo cada ponteiro, mesmo se a lista estiver classificada. A inserção em uma lista encadeada pode ter um tempo de execução de  $O(1)$ , se inserirmos novos nós no início da lista.

### Implementando Vetores

Vamos ver como podemos implementar o redimensionamento de uma matriz:

```

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 // Use malloc para alocar espaço suficiente para uma matriz com 3
 inteiros
 int *list = malloc(3 * sizeof(int));
 if (list == NULL)
 {
 return 1;
 }

 // Defina os valores em nosso array
 list[0] = 1;
 list[1] = 2;
 list[2] = 3;

 // Agora, se quisermos armazenar outro valor, podemos alocar mais

```



memória

```
int *tmp = malloc(4 * sizeof(int));
if (tmp == NULL)
{
 free(list);
 return 1;
}

// Copie a lista de tamanho 3 para a lista de tamanho 4
for(int i = 0; i < 3; i++)
{
 tmp[i] = list[i];
}

// Adicionar novo número à lista de tamanho 4
tmp[3] = 4;

// Lista original grátis de tamanho 3
free(list);

// Lembre-se da nova lista de tamanho 4
list = tmp;

// Imprimir lista
for (int i = 0; i < 4; i++)
{
 printf("%i\n", list[i]);
}

// Free nova lista
free(list);
}
```

Lembre-se de que o **malloc** aloca e libera memória da área de heap. Acontece que podemos chamar outra função de biblioteca, **realloc**, para realocar alguma memória que alocamos anteriormente:

```
int * tmp = realloc (list, 4 * sizeof (int));
```

- E **realloc** copia nosso antigo array, **list**, para nós em um pedaço maior de memória do tamanho que passamos. Se acontecer de haver espaço depois de nosso pedaço de memória existente, vamos obter o mesmo endereço de volta, mas com a memória depois de alocado à nossa variável também.

### Implementando Listas Encadeadas

Vamos combinar nossos trechos de código anteriores em um programa que implementa uma lista vinculada:

```

#include <stdio.h>
#include <stdlib.h>

// Representando um nó
typedef struct node
{
 int number; struct node *next;
}
node;

int main(void)
{
 // Lista de tamanho 0. Inicializamos o valor para NULL
 explicitamente, então há
 // nenhum valor de lixo para nossa variável de lista
 node *list = NULL;

 // Alocar memória para um node, n
 node *n = malloc(sizeof(node));
 if(n == NULL)
 {
 return 1;
 }

 // Definir o valor e ponteiro no nosso node
 n->number = 1;
 n->next = NULL;

 // Adicione o nó n apontando lista para ele, uma vez que só temos um
 nó até agora
 list = n;

 // Aloque memória para outro nó, e podemos reutilizar nossa variável n
 para
 // aponte para ele, uma vez que a lista já aponta para o primeiro nó
 n = malloc(sizeof(node));
 if(n == NULL)
 {
 free(list);
 return 1;
 }

 // Defina os valores em nosso novo nó
 n->number = 2;
 n->next = NULL;

 // Atualize o ponteiro em nosso primeiro nó para apontar para o
 segundo nó
 list->next = n;

```

```

// Alocar memória para um terceiro nó
n = malloc(sizeof(node));

if(n == NULL){
 // Libere nossos outros nós
 free(list->next);
 free(list);
 return 1;
}
n->number = 3;
n->next = NULL;

// Siga o próximo ponteiro da lista para o segundo nó e atualize
// o próximo ponteiro para apontar para n
list->next->next = n;

// Imprime a lista usando um loop, usando uma variável temporária,
tmp, para apontar
// para listar, o primeiro nó. Então, toda vez que examinamos o loop,
usamos
// tmp = tmp-> next para atualizar nosso ponteiro temporário para o
próximo nó.
// continue enquanto tmp aponta para algum lugar, parando quando
chegarmos a
// o último nó e tmp-> next é nulo.
for(node *tmp = list; tmp != NULL; tmp = tmp->next)
{
 printf("%i\n", tmp->number);
}

// Libere a lista, usando um loop while e uma variável temporária
para apontar
// para o próximo nó antes de liberar o atua
while(list != NULL)
{
 // Nós apontamos para o próximo nó primeiro
 node *tmp = list->next;

 // Então, podemos liberar o primeiro nó free(list);
 // Agora podemos definir a lista para apontar para o próximo nó
 list = tmp;

 // Se a lista for nula, quando não houver mais nós restantes,
 nosso loop while irá parar
}
}

```

Se quisermos inserir um nó à frente de nossa lista vinculada, precisaremos atualizar cuidadosamente nosso nó para apontar para o seguinte, antes de atualizar a variável de lista. Caso contrário, perderemos o resto da nossa lista:

```
// Aqui, estamos inserindo um nó na frente da lista, então queremos que seu
// próximo ponteiro para apontar para a lista original. Então podemos
mudar a lista para
// aponta para n.
n-> próximo = lista; lista = n;
```

A princípio, teremos um nó com valor **1** apontando para o início de nossa lista, um nó com valor **2**:

•

- Agora podemos atualizar nossa variável **list** para apontar para o nó com valor **1**, e não perder o resto de nossa lista.

Da mesma forma, para inserir um nó no meio de nossa lista, mudamos o ponteiro **next** do novo nó primeiro para apontar para o resto da lista e, em seguida, atualizamos o nó anterior para apontar para o novo nó.

Uma lista vinculada demonstra como podemos usar ponteiros para construir estruturas de dados flexíveis na memória, embora estejamos apenas visualizando em uma dimensão.

## Árvores

Com um array ordenado, podemos usar a pesquisa binária para encontrar um elemento, começando no meio (amarelo), depois no meio da metade (vermelho) e, finalmente, à esquerda ou à direita (verde), conforme necessário:

•

- Com um array, podemos acessar elementos aleatoriamente no tempo  $O(1)$ , uma vez que podemos usar a aritmética para ir para um elemento em qualquer índice.

Uma **árvore(tree)** é outra estrutura de dados onde cada nó aponta para dois outros nós, um à esquerda (com um valor menor) e um à direita (com um valor maior):



- Observe que agora visualizamos essa estrutura de dados em duas dimensões (mesmo que os nós na memória possam estar em qualquer local).
- 
- E podemos implementar isso com uma versão mais complexa de um nó em uma lista vinculada, onde cada nó tem não um, mas dois ponteiros para outros nós. Todos os valores à esquerda de um nó são menores e todos os valores dos nós à direita são maiores, o que

permite que isso seja usado como uma **árvore de busca binária**. E a própria estrutura de dados é definida recursivamente, então podemos usar funções recursivas para trabalhar com ela.

- 
- Cada nó tem no máximo dois **filhos(children)**, ou nós para os quais está apontando.
- 
- E como uma lista vinculada, queremos manter um ponteiro apenas para o início da lista, mas neste caso queremos apontar para a **raíz(root)**, ou nó central superior da árvore (o 4).

Podemos definir um nó não com um, mas com dois ponteiros:

```
typedef struct node
{
 int number;
 struct node *left;
 struct node *right;
}
node;
```

E escreva uma função para pesquisar recursivamente em uma árvore:

```
// árvore é um ponteiro para um nó que é a raiz da árvore que estamos
pesquisando.
// número é o valor que estamos tentando encontrar na árvore.
bool search(node *tree, int number)
{
 // Primeiro, nos certificamos de que a árvore não é NULL, se
 alcançamos um nó
 // na parte inferior, ou se nossa árvore estiver totalmente vazia
 if(tree == NULL){
 return false;
 }
 // Se estivermos procurando por um número menor que o número da
 árvore,
 // pesquisa do lado esquerdo, usando o nó à esquerda como a nova raiz
 else if(number < tree->number)
 {
 return search(tree->left, number);
 }
 // Caso contrário, pesquise no lado direito, usando o nó à direita
 como a nova raiz
 else if (number > tree->number)
 {
 return search(tree->right, number);
 }
 // Finalmente, encontramos o número que procuramos, portanto, podemos
 retornar true
 // Podemos simplificar isso para apenas "outro", uma vez que não há
 outro caso possível
 else if(number == tree->number)
 {

```

```
 return true;
 }
}
```

Com uma árvore de pesquisa binária, incorremos no custo de ainda mais memória, já que cada nó agora precisa de espaço para um valor e dois ponteiros. A inserção de um novo valor levaria tempo  $O(\log n)$ , uma vez que precisamos encontrar os nós entre os quais ele deve ficar.

No entanto, se adicionarmos nós suficientes, nossa árvore de pesquisa pode começar a se parecer com uma lista vinculada:

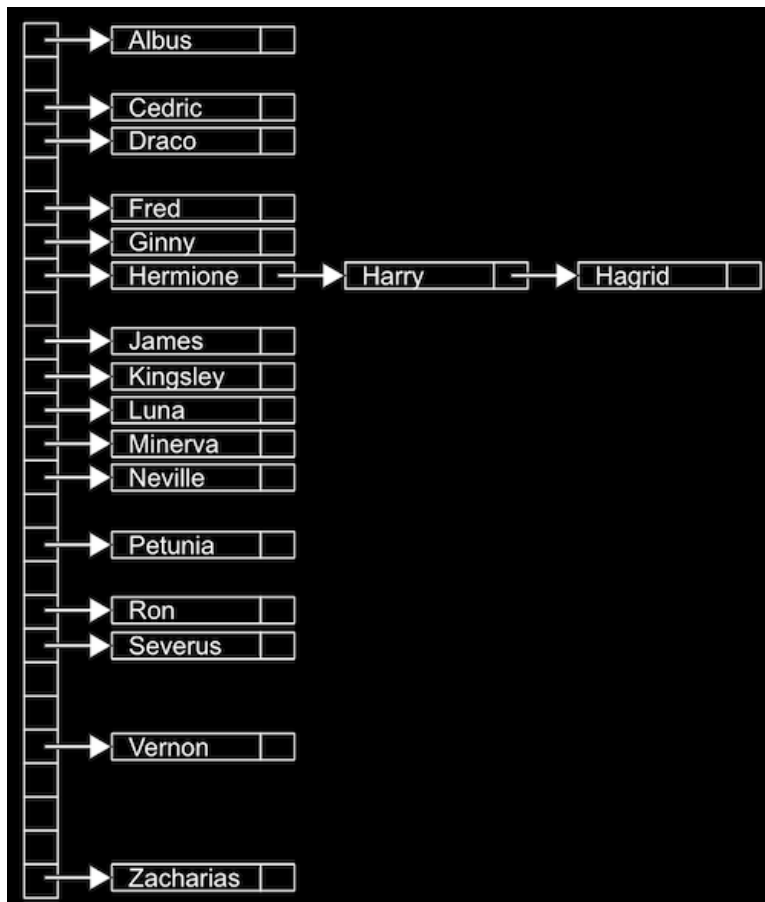


- Começamos nossa árvore com um nó com valor **1**, depois adicionamos o nó com valor **2** e, finalmente, adicionamos o nó com valor **3**. Mesmo que essa árvore siga as restrições de uma árvore de pesquisa binária, não é tão eficiente quanto poderia ser.
- 
- Podemos tornar a árvore balanceada, ou ótima, tornando o nó com valor **2** o novo nó raiz. Cursos mais avançados cobrirão estruturas de dados e algoritmos que nos ajudam a manter as árvores equilibradas conforme os nós são adicionados.

### Mais estruturas de dados

Uma estrutura de dados com tempo quase constante,  $O(1)$  search é uma **tabela hash**, que é essencialmente um *array* de listas encadeadas. Cada lista encadeada na matriz possui elementos de uma determinada categoria.

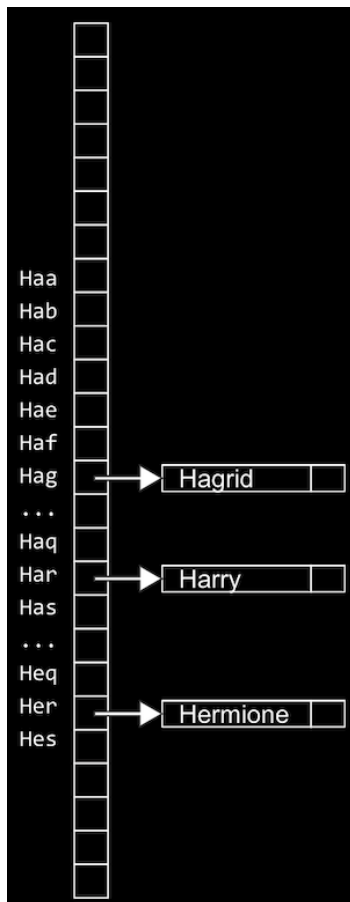
Por exemplo, podemos ter muitos nomes e podemos classificá-los em uma matriz com 26 posições, uma para cada letra do alfabeto:



- Como temos acesso aleatório com matrizes, podemos definir elementos e índices em um local, ou intervalo, na matriz rapidamente.
- 
- Um local pode ter vários valores correspondentes, mas podemos adicionar um valor a outro valor, já que eles são nós em uma lista vinculada, como vemos com Hermione, Harry e Hagrid. Não precisamos aumentar o tamanho de nossa matriz ou mover qualquer um de nossos outros valores.

Isso é chamado de tabela hash porque usamos uma **função hash**, que pega alguma entrada e mapeia de forma determinística para o local em que deveria ir. Em nosso exemplo, a função hash apenas retorna um índice correspondente à primeira letra do nome, como como **0** para “Alvo” e **25** para “Zacarias”.

Mas, na pior das hipóteses, todos os nomes podem começar com a mesma letra, então podemos terminar com o equivalente a uma única lista vinculada novamente. Podemos olhar para as duas primeiras letras e alocar baldes suficientes para  $26 * 26$  valores de hash possíveis, ou mesmo as três primeiras letras, exigindo  $26 * 26 * 26$  baldes:



- Agora, estamos usando mais espaço na memória, já que alguns desses depósitos estarão vazios, mas é mais provável que necessitemos apenas uma etapa para procurar um valor, reduzindo nosso tempo de execução para pesquisa.

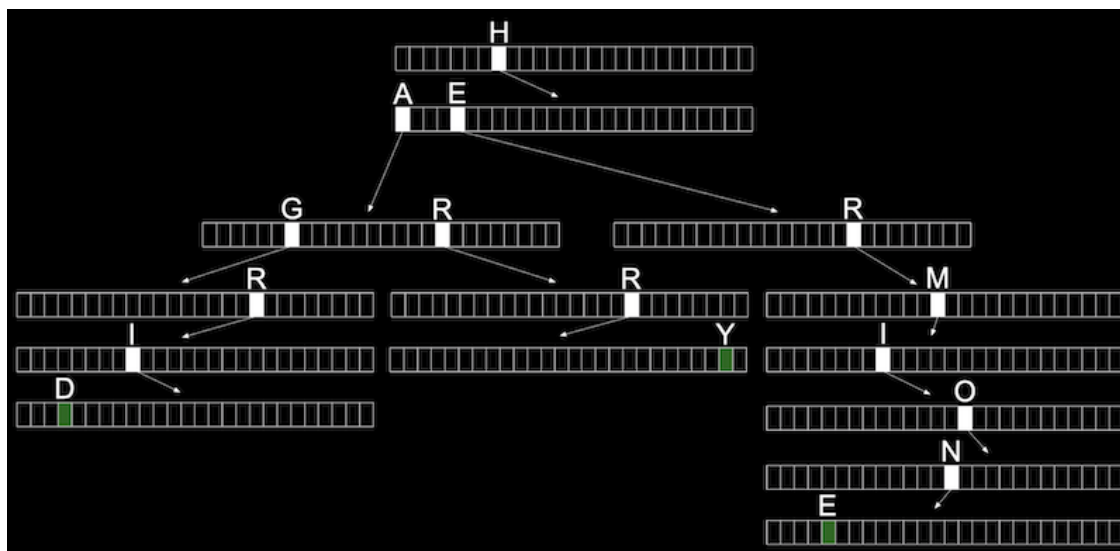
Para classificar algumas cartas de jogo padrão, também podemos começar colocando-as em pilhas por naipe, de espadas, ouros, copas e paus. Então, podemos classificar cada pilha um pouco mais rapidamente.

Acontece que o pior caso de tempo de execução para uma tabela hash é  $O(n)$ , uma vez que, à medida que  $n$  fica muito grande, cada depósito terá valores da ordem de  $n$ , mesmo que tenhamos centenas ou milhares de depósitos. Na prática, porém, nosso tempo de execução será mais rápido, pois estamos dividindo nossos valores em vários depósitos.

No conjunto de problemas 5, seremos desafiados a melhorar o tempo de execução do mundo real de pesquisa de valores em nossas estruturas de dados, ao mesmo tempo em que equilibramos nosso uso de memória.

Podemos usar outra estrutura de dados chamada **trie** (pronuncia-se "try" e é uma abreviação de "recuperação"). Um trie é uma árvore com matrizes como nós:





- Cada array terá cada letra, AZ, armazenada. Para cada palavra, a primeira letra apontará para um array, onde a próxima letra válida apontará para outro array, e assim por diante, até chegarmos a um valor booleano indicando o final de uma palavra válida, marcada em verde acima. Se nossa palavra não estiver no teste, então uma das matrizes não terá um ponteiro ou caractere de terminação para nossa palavra.
- 
- No trie acima, temos as palavras Hagrid, Harry e Hermione.
- 
- Agora, mesmo que nossa estrutura de dados tenha muitas palavras, o tempo máximo de pesquisa será apenas o comprimento da palavra que estamos procurando. Isso pode ser um máximo fixo, então podemos ter  $O(1)$  para pesquisa e inserção.
- 
- O custo disso, porém, é que precisamos de muita memória para armazenar ponteiros e valores booleanos como indicadores de palavras válidas, embora muitos deles não sejam usados.

Existem construções de nível ainda mais alto, **estruturas de dados abstratas**, onde usamos nossos blocos de construção de arrays, listas vinculadas, tabelas de hash e tentamos implementar uma solução para algum problema.

Por exemplo, uma estrutura de dados abstrata é uma **queue (ou fila)**, como uma fila de pessoas esperando, onde o primeiro valor que colocamos são os primeiros valores que são removidos, ou first-in-first-out (FIFO). Para adicionar um valor que **enfileira-lo**, e para remover um valor que **desenfileira-lo**. Essa estrutura de dados é abstrata porque é uma ideia que podemos implementar de diferentes maneiras: com um array que redimensionamos à medida que adicionamos e removemos itens, ou com uma lista vinculada onde acrescentamos valores ao final.

Uma estrutura de dados “oposta” seria um **stack (ou pilha)**, onde os itens adicionados mais recentemente são removidos primeiro: último a entrar, primeiro a sair (UEPS). Em uma loja de roupas, podemos pegar, ou **abrir**, o suéter de cima de uma pilha, e novos suéteres são adicionados, ou **empurrados**, para cima também.

Outro exemplo de estrutura de dados abstrata é um **dicionário**, onde podemos mapear chaves para valores, como palavras para suas definições. Podemos implementar um com uma tabela hash ou um array, levando em consideração a relação entre tempo e espaço.

Dê uma olhada em [“Jack Learns the Facts About Queues and Stacks”](#) (em inglês), uma animação sobre essas estruturas de dados.

## EXERCÍCIOS MÓDULO 5

### Exercício 1: Speller

Certifique-se de ler esta especificação por completo antes de começar, para saber o que fazer e como fazer!

Implemente um programa que verifica a ortografia de um arquivo, como mostrado a seguir, usando uma tabela de hash.

```
$./speller texts/lalaland.txt
MISSPELLED WORDS
```

```
[...]
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHT
[...]
Shangri
[...]
fianc
[...]
Sebastian's
[...]
```

```
WORDS MISSPELLED:
WORDS IN DICTIONARY:
WORDS IN TEXT:
TIME IN load:
TIME IN check:
TIME IN size:
TIME IN unload:
TIME IN TOTAL:
```

## Vamos começar

Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute **cd ~/** (ou simplesmente **cd** sem argumentos) para garantir que você está no seu diretório pessoal.
- Execute **mkdir pset5** para fazer (ou seja, criar) um diretório chamado **pset5**.
- Execute **cd pset5** para mudar para (ou seja, abrir) esse diretório.
- Execute <http://cdn.cs50.net/2021/spring/psets/5/speller/speller.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.

- Execute **unzip speller.zip** para descompactar esse arquivo.
- Execute **rm speller.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- Execute **ls**. Você deve ver um diretório chamado **speller**, que estava dentro desse arquivo ZIP.
- Execute **cd speller** para mudar para esse diretório.
- Execute **ls**. Você deve ver a distribuição deste problema:

```

dictionaries/ dictionary.c dictionary.h keys/ Makefile speller.c
texts/

```

## Entendimento

Teoricamente, na entrada de tamanho  $n$ , um algoritmo com um tempo de execução de  $n$  é “assintoticamente equivalente,” em termos de  $O$ , a um algoritmo com um tempo de execução de  $2n$ . Na verdade, ao descrever o tempo de execução de um algoritmo, normalmente nos concentramos no termo dominante (ou seja, mais impactante) (ou seja,  $n$  neste caso, uma vez que  $n$  pode ser muito maior do que 2). No mundo real, porém, o fato é que  $2n$  parece duas vezes mais lento que  $n$ .

O desafio à sua frente é implementar o corretor ortográfico mais rápido possível! Por "mais rápido", porém, estamos falando de um "relógio de parede" real, não assintótico, tempo.

Em `speller.c`, criamos um programa projetado para verificar a ortografia de um arquivo após carregar um dicionário de palavras do disco para a memória. Esse dicionário, entretanto, é implementado em um arquivo chamado `dictionary.c`. (Ele poderia apenas ser implementado em `speller.c`, mas à medida que os programas ficam mais complexos, geralmente é conveniente dividi-los em vários arquivos.) Os protótipos para as funções nele, entretanto, são definidos não no próprio `dictionary.c`, mas em `dictionary.h` ao invés disso. Dessa forma, tanto `speller.c` quanto `dictionary.c` podem **#include** o arquivo. Infelizmente, não conseguimos implementar a parte de carregamento. Ou a parte de verificação. Ambos (e um pouco mais) deixamos para você! Mas primeiro, um tour.

## dictionary.h

Abra **dictionary.h**, e você verá alguma nova sintaxe, incluindo algumas linhas que mencionam **DICTIONARY\_H**. Não há necessidade de se preocupar com essas, mas, se curioso, essas linhas apenas garantem que, mesmo que **dictionary.c** e **speller.c** (que você verá em um momento) **#include** este arquivo, **clang** só irá compilá-lo uma vez.

A seguir, observe como **#include** um arquivo chamado **stdbool.h**. Esse é o arquivo no qual o próprio `bool` é definido. Você não precisou disso antes, uma vez que a Biblioteca CS50 usava `#include` isso para você.

Observe também nosso uso de **#define**, uma “diretiva de pré-processador” que define uma “constante” chamada **LENGTH** que tem um valor de **45**. É uma constante no sentido de que você não pode (acidentalmente) alterá-la em seu próprio código. Na verdade, o **clang** substituirá qualquer menção de **LENGTH** em seu próprio código por, literalmente, **45**. Em outras palavras, não é uma variável, apenas um truque de localizar e substituir.

Finalmente, observe os protótipos para cinco funções: **check**, **hash**, **load**, **size**, e **unload**. Observe como três deles tomam um ponteiro como um argumento, de acordo com **\***:

```
bool check(const char *word);
unsigned int hash(const char *word);
bool load(const char *dictionary);
```

Lembre-se de que **char \*** é o que costumávamos chamar de **string** . Portanto, esses três protótipos são essencialmente apenas:

```
bool check(const string word);
unsigned int hash(const string word);
bool load(const string dictionary);
```

E **const** , entretanto, apenas diz que essas strings, quando passadas como argumentos, devem permanecer constantes; você não poderá alterá-los, acidentalmente ou não!

## dicionário.c

Agora abra o dictionary.c . Observe como, no topo do arquivo, definimos uma **struct** chamada **node** que representa um nó em uma tabela hash. E declaramos uma matriz de ponteiro global, **table** , que (em breve) representará a tabela hash que você usará para controlar as palavras no dicionário. A matriz contém **N** ponteiros de nó, e definimos **N** igual a **1** por enquanto, o que significa que esta tabela hash tem apenas 1 depósito no momento. Você provavelmente vai querer aumentar o número de depósitos, alterando **N** , para algo maior!

Em seguida, observe que implementamos **load** , **hash** , **check** , **size** e **unload** , mas apenas um pouco, apenas o suficiente para que o código seja compilado. Seu trabalho, em última análise, é reimplementar essas funções da forma mais inteligente possível para que este corretor ortográfico funcione como anunciado. E rápido!

## speller.c

Ok, em seguida abra speller.c e passe algum tempo examinando o código e os comentários nele. Você não precisará alterar nada neste arquivo e não precisa entender sua totalidade, mas tente ter uma noção de sua funcionalidade mesmo assim. Observe como, por meio de uma função chamada **getrusage** , estaremos fazendo um “benchmarking” (isto é, cronometrando a execução de) suas implementações de **load** , **hash** , **check** , **size** e **unload** . Observe também como fazemos para passar na **check** , palavra por palavra, o conteúdo de algum arquivo a ser verificado. Por fim, relatamos cada erro ortográfico nesse arquivo junto com um monte de estatísticas.

Observe, aliás, que definimos o uso do *speller* como

```
Usage: speller [dictionary] text
```

onde *dictionary* é considerado um arquivo contendo uma lista de palavras minúsculas, uma por linha, e **text** é um arquivo a ser verificado. Como os colchetes sugerem, o fornecimento de *dictionary* é opcional; se este argumento for omitido, o **speller** usará dictionaries/large como padrão. Em outras palavras, executar

```
$./speller text
```

será equivalente a executar

```
$./speller dictionaries/large text
```

onde **text** é o arquivo que você deseja verificar a ortografia. Basta dizer que o primeiro é mais fácil de digitar! (Obviamente, o speller não será capaz de carregar nenhum dicionário até que você implemente `load in dictionary.c` ! Até então, você verá `Could not load` .)

No dicionário padrão, veja bem, existem 143.091 palavras, todas as quais devem ser carregadas na memória! Na verdade, dê uma olhada nesse arquivo para ter uma noção de sua estrutura e tamanho. Observe que todas as palavras desse arquivo aparecem em minúsculas (até mesmo, para simplificar, nomes próprios e acrônimos). De cima para baixo, o arquivo é classificado lexicograficamente, com apenas uma palavra por linha (cada uma das quais termina com `\n` ). Nenhuma palavra tem mais de 45 caracteres e nenhuma palavra aparece mais de uma vez. Durante o desenvolvimento, você pode achar útil fornecer ao speller um dicionário próprio que contenha muito menos palavras, para não ter que lutar para depurar uma estrutura enorme na memória. Em `dictionaries/small` é um desses dicionários. Para usá-lo, execute

```
$./speller dictionaries/small text
```

onde **text** é o arquivo que você deseja verificar a ortografia. Não siga em frente até ter certeza de que entendeu como o speller funciona!

Provavelmente, você não gastou tempo suficiente examinando `speller.c` . Volte um quadrado e percorra-o novamente!

## texts/

Para que você possa testar sua implementação do **speller** , fornecemos também um monte de textos, entre eles o roteiro de *La La Land* , o texto do *Affordable Care Act*, três milhões de bytes de Tolstoy, alguns trechos de *The Federalist Papers* e Shakespeare e muito mais. Para que você saiba o que esperar, abra e folheie cada um desses arquivos, todos os quais estão em um diretório chamado de **texts** dentro do seu diretório **pset5** .

Agora, como você deve saber por ter lido `speller.c` cuidadosamente, o output de **speller**, se executado com, digamos,

```
$./speller text/lalaland.txt
```

eventualmente se parecerá com o abaixo.

Abaixo estão alguns dos outputs que você verá. Para fins de informação, extraímos alguns exemplos de “erros ortográficos”. E, para não estragar a diversão, omitimos nossas próprias estatísticas por enquanto.

MISSPELLED WORDS

```
[...]
AHHHHHHHHHHHHHHHHHHHHHHHHHHHHHT
[...]
Shangri
[...]
fianc
[...]
Sebastian's
```

[...]

WORDS MISPELLED:  
WORDS IN DICTIONARY:  
WORDS IN TEXT:  
TIME IN load:  
TIME IN check:  
TIME IN size:  
TIME IN unload:  
TIME IN TOTAL:

**TIME IN load** representa o número de segundos que o **speller** gasta executando sua implementação de **load** . **TIME IN check** representa o número de segundos que o **speller** gasta, no total, executando sua implementação de **check** . **TIME IN size** representa o número de segundos que o **speller** gasta executando sua implementação de **size** . **TIME IN unload** representa o número de segundos que o **speller** gasta executando sua implementação de **unload** . **TIME IN TOTAL** é a soma dessas quatro medições.

Observe que esses tempos podem variar um pouco entre as execuções do **speller** , dependendo do que mais o CS50 IDE está fazendo, mesmo se você não alterar seu código.

A propósito, para ser claro, por “erro ortográfico” queremos dizer simplesmente que alguma palavra não está no *dictionary* fornecido.

## Makefile

E, por último, lembre-se de que o **make** automatiza a compilação do seu código para que você não precise executar o **clang** manualmente junto com um monte de opções. No entanto, conforme seus programas crescem em tamanho, o **make** não será mais capaz de inferir a partir do contexto como compilar seu código; você precisará começar a dizer ao **make** como compilar seu programa, particularmente quando eles envolvem vários arquivos de origem (ou seja, **.c**), como no caso deste problema. E então vamos utilizar um **Makefile** , um arquivo de configuração que diz ao **make** exatamente o que fazer. Abra o **Makefile** e você verá quatro linhas:

- A primeira linha diz ao **make** para executar as linhas subsequentes sempre que você executar **make speller**(ou apenas **make** ).
- A segunda linha informa ao **make** como compilar **speller.c** em código de máquina (ou seja, **speller.o** ).
- A terceira linha diz ao **make** como compilar o **dictionary.c** em código de máquina (isto é, **dictionary.o** ).
- A quarta linha informa ao **make** para vincular **speller.o** e **dictionary.o** em um arquivo chamado **speller** .

Certifique-se de compilar o **speller** executando **make speller** (ou apenas **make** ). Executar o **make dictionary** não funcionará!

# Especificação

Tudo bem, o desafio agora é implementar, em ordem, **load** , **hash** , **size** , **check** e **unload** da forma mais eficiente possível usando uma tabela de hash de tal forma que **TIME IN load** , **TIME IN check** , **TIME IN size** e **TIME IN unload** são todos minimizados. Para ter certeza, não é óbvio o que significa ser minimizado, visto que esses benchmarks certamente irão variar à medida que você fornece valores diferentes para o **dictionary** e para o **text** ao **speller** . Mas aí está o desafio, senão a diversão, desse problema. Este problema é sua chance de projetar. Embora o convidemos a minimizar o espaço, seu maior inimigo é o tempo. Mas antes de você mergulhar, algumas especificações nossas.

- Você não pode alterar `speller.c` , `dictionary.h` ou `Makefile` .
- Você pode alterar o `dictionary.c` (e, de fato, deve, a fim de completar as implementações de **load** , **hash** , **size** , **check** e **unload** ), mas você não pode alterar as declarações (ou seja, protótipos) de **load** , **hash** , **size** , **check** e **unload** . Você pode, entretanto, adicionar novas funções e variáveis (locais ou globais) ao `dictionary.c` .
- Você pode alterar o valor de **N** no `dictionary.c` , para que sua tabela hash possa ter mais depósitos.
- Sua implementação de **check** deve não fazer distinção entre maiúsculas e minúsculas. Em outras palavras, se `foo` estiver no dicionário, então `check` deve retornar `true` considerando qualquer capitalização; `foo` , `foO` , `fOo` , `fOO` , `fOO` , `Foo` , `FoO` , `FOO` e `FOO` deve ser considerados corretos.
- Deixando as letras maiúsculas de lado, sua implementação de **check** só deve retornar **true** para palavras realmente no `dictionary` . Cuidado com as palavras comuns embutidas no código (por exemplo, `o` ), para não passarmos à sua implementação um `dictionary` sem essas mesmas palavras. Além disso, os únicos possessivos permitidos são aqueles que constam do `dictionary` . Em outras palavras, mesmo se `foo` estiver no `dictionary` , `check` deve retornar `false` dado `foo's` se `foo's` também não estiver no `dictionary` .
- Você pode presumir que qualquer `dictionary` passado para o seu programa será estruturado exatamente como o nosso, classificado em ordem alfabética de cima para baixo com uma palavra por linha, cada uma delas terminando com `\n` . Você também pode presumir que o `dictionary` conterá pelo menos uma palavra, que nenhuma palavra terá mais do que `LENGTH` (uma constante definida no `dictionary.h` ) caracteres, que nenhuma palavra aparecerá mais de uma vez, que cada palavra conterá apenas caracteres alfabéticos minúsculos e possivelmente apóstrofes, e nenhuma palavra começará com um apóstrofo.
- Você pode presumir que **check** será aprovada apenas em palavras que contenham caracteres alfabéticos (maiúsculos ou minúsculos) e possivelmente apóstrofes.
- Seu corretor ortográfico pode aceitar apenas **text** e, opcionalmente, `dictionary` como entrada. Embora você possa estar inclinado (principalmente se estiver entre os mais confortáveis) a "pré-processar" nosso dicionário padrão para derivar uma "função de hash ideal" para ele, você não pode salvar a saída de qualquer pré-processamento em disco em para carregá-lo de volta na memória em execuções subsequentes de seu corretor ortográfico para obter uma vantagem.
- Seu verificador ortográfico não deve vaziar nenhuma memória. Certifique-se de verificar se há vazamentos com **valgrind** .

- Você pode pesquisar funções hash (boas) online, desde que cite a origem de qualquer função hash integrada em seu próprio código.

Tudo bem, pronto para ir?

- Implemente **load**.
- Implemente **hash**.
- Implemente **size**.
- Implemente **check**.
- Implemente **unload**.

## Dicas

Para comparar duas strings sem distinção entre maiúsculas e minúsculas, você pode achar [strcasecmp](#) (declarado em **strings.h**) útil! Você provavelmente também desejará garantir que sua função hash não faça distinção entre maiúsculas e minúsculas, de forma que **foo** e **FOO** tenham o mesmo valor de hash.

Por fim, certifique-se de free ao unload qualquer memória que você alocou no **load** ! Lembre-se de que **valgrind** é seu mais novo melhor amigo. Saiba que o valgrind observa vazamentos enquanto seu programa está realmente em execução, portanto, certifique-se de fornecer argumentos de linha de comando se quiser que o valgrind analise o speller enquanto você usa um dictionary e / ou texto específico, como mostrado a seguir. No entanto, é melhor usar um texto pequeno, senão o valgrind pode demorar um pouco para ser executado.

```
$ valgrind ./speller text/cat.txt
```

Se você executar o valgrind sem especificar um texto para o speller , suas implementações de load e unload não serão realmente chamadas (e, portanto, analisadas).

Se não tiver certeza de como interpretar a saída de valgrind, use help50 para obter ajuda:

```
$ help50 valgrind ./speller text/cat.txt
```

## Testando

Como verificar se seu programa está apresentando as palavras corretas com erros ortográficos? Bem, você pode consultar as respostas que estão dentro do diretório de keys que está dentro do seu diretório de speller . Por exemplo, dentro de keys / lalaland.txt estão todas as palavras que seu programa deve achar que foram escritas incorretamente.

Você pode, portanto, executar seu programa em algum texto em uma janela, como a seguir.

```
$./speller text/lalaland.txt
```

E você poderia então executar a solução do Staff no mesmo texto em outra janela, como a seguir.

```
$ ~ cs50/2019/fall/pset5/speller text/lalaland.txt
```

E você pode comparar as janelas visualmente lado a lado. Isso pode se tornar tedioso rapidamente, no entanto. Portanto, você pode querer “redirecionar” a saída do seu programa para um arquivo, como mostrado a seguir.



```
$./speller text/lalaland.txt > student.txt
$ ~ cs50/2019/fall/pset5/speller texts/lalaland.txt > staff.txt
```

Você pode então comparar os dois arquivos lado a lado na mesma janela com um programa como o diff , como o mostrado abaixo.

```
$ diff -y student.txt staff.txt
```

Como alternativa, para economizar tempo, você pode apenas comparar a saída do seu programa (supondo que você o redirecionou para, por exemplo, student.txt ) com uma das respostas sem executar a solução do Staff, como mostrado a seguir.

```
$ diff -y student.txt keys/lalaland.txt
```

Se a saída do seu programa coincidir com a do Staff, diff produzirá duas colunas que devem ser idênticas, exceto, talvez, pelos tempos de execução na parte inferior. Se as colunas forem diferentes, porém, você verá um > ou | onde eles diferem. Por exemplo, se você ver

```
MISSPELLED WORDS MISSPELLED WORDS
```

```
TECHNO TECHNO
L L
> Thelonious
Prius Prius
> MIA
L L
```

isso significa que seu programa (cujo output está à esquerda) não pensa que Thelonious ou MIA está incorreto, embora o output do Staff (à direita) ache , como está implícito pela ausência de, digamos, Thelonious na coluna da esquerda e a presença de Thelonious na coluna da direita.

### check50

Para testar seu código menos manualmente (embora ainda não exaustivamente), você também pode executar o seguinte.

```
$ check50 cs50/problems/2021/x/speller
```

Observe que **check50** também verificará vazamentos de memória, portanto, certifique-se de executar o valgrind também.

### style50

Execute o seguinte para avaliar o estilo do seu código usando **style50** .

```
style50 dictionary.c
```

## Solução da equipe

Como avaliar o quão rápido (e correto) é o seu código? Bem, como sempre, fique à vontade para brincar com a solução do staff, como abaixo, e compare seus números com os seus.

```
$ ~ cs50/2019/fall/pset5/speller text/lalaland.txt
```

# Conselhos

- Assista a aula do módulo 5 novamente :)
- Se você ver algum erro de compilação, use **make**. Se não tiver certeza do que isso significa, escreva **help50** para ajuda.

make speller

Se os erros continuarem...

help50 make speller

## Lab 5: Herança

# Lab: Herança

Simule a herança de tipos sanguíneos para cada membro de uma família.

```
$./inheritance
Generation 0, blood type OO
 Generation 1, blood type AO
 Generation 2, blood type OA
 Generation 2, blood type BO
 Generation 1, blood type OB
 Generation 2, blood type AO
 Generation 2, blood type BO
```

## Background

O tipo sanguíneo de uma pessoa é determinado por dois alelos (isto é, diferentes formas de um gene). Os três alelos possíveis são A, B e O, dos quais cada pessoa tem dois (possivelmente iguais, possivelmente diferentes). Cada um dos pais de uma criança passa aleatoriamente um de seus dois alelos de tipo sanguíneo para o filho. As combinações possíveis de tipo sanguíneo, então, são: OO, OA, OB, AO, AA, AB, BO, BA e BB.

Por exemplo, se um dos pais tem o tipo sanguíneo AO e o outro tem o tipo sanguíneo BB, os possíveis tipos sanguíneos da criança seriam AB e OB, dependendo de qual alelo é recebido de cada pai. Da mesma forma, se um dos pais tem tipo de sangue AO e o outro OB, os possíveis tipos de sangue da criança seriam AO, OB, AB e OO.

## Começando

Crie um novo diretório em seu IDE chamado **lab5**. Nesse diretório, execute <https://cdn.cs50.net/2020/fall/labs/5/inheritance.c> para baixar o código de distribuição para este projeto.

## Entendendo o problema

Dê uma olhada no código de distribuição em inheritance.c .

Observe a definição de um tipo chamado **person** . Cada pessoa tem um vetor(array) de dois pais(**parents**), cada um dos quais é um ponteiro para a estrutura de outra **person** . Cada pessoa também tem uma matriz de dois alelos(**alleles**), cada um dos quais é um **char** ( 'A' , 'B' ou 'O' ).

Agora, dê uma olhada na função **main** . A função começa “semeando” (ou seja, fornecendo alguma entrada inicial para) um gerador de números aleatórios, que usaremos mais tarde para gerar alelos aleatórios. A função **main**, em seguida, chama o **create\_family** função para simular a criação de estruturas do tipo **person** para uma família de 3 gerações (ou seja, uma pessoa, seus pais e seus avós). Em seguida, chamamos **print\_family** para imprimir cada um desses membros da família e seus tipos de sangue. Finalmente, a função chama **free\_family** para “liberar” (**free**) qualquer memória alocada anteriormente com **malloc** .

As funções **create\_family** e **free\_family** são deixados para que você escreva!

## Detalhes de implementação

Conclua a implementação de **inheritance.c** , de modo que crie uma família com um tamanho de geração especificado e atribua alelos de tipo sanguíneo a cada membro da família. A geração mais velha terá alelos atribuídos aleatoriamente a eles.

A função **create\_family** leva um inteiro (**generations**) como input e deve alocar (como via **malloc**) uma **person** para cada membro da família daquele número de gerações, retornando um ponteiro para a **person** na geração mais jovem.

- Por exemplo, **create\_family(3)** deve retornar um ponteiro para uma pessoa com dois pais, onde cada pai também tem dois pais.
- Cada **person** deve ter **alleles** atribuídos a ela. A geração mais velha deve ter alelos escolhidos aleatoriamente (como chamando a função **random\_allele**), e as gerações mais jovens devem herdar um alelo (escolhido aleatoriamente) de cada pai.
- Cada **person** deve ter **parents** designados para ela. A geração mais velha deve ter ambos os **parents** definidos como **NULL**, e as gerações mais novas devem ter os **parents** como uma matriz de dois ponteiros, cada um apontando para um pai diferente.

Dividimos a função **create\_family** em alguns **TO DO's** para você completar.

Primeiro, você deve alocar memória para uma nova pessoa. Lembre-se de que você pode usar **malloc** para alocar memória e **sizeof(person)** para obter o número de bytes a serem alocados.

Em seguida, incluímos uma condição para verificar se as **generations > 1** .

- Se **generations > 1**, então há mais gerações que ainda precisam ser alocadas. Sua função deve definir ambos os **parents** chamando recursivamente **create\_family**. (Quantas **generations** devem ser passadas como entrada para cada pai?) A função deve então definir ambos os **alleles** escolhendo aleatoriamente um alelo de cada pai.
- Caso contrário (**if generations == 1**), então não haverá dados dos pais para esta pessoa. Ambos os **parents** devem ser definidos como **NULL**, e cada **allele** deve ser gerado aleatoriamente.

Finalmente, sua função deve retornar um ponteiro para a **person** que foi alocada.

A função **free\_family** deve aceitar como entrada um ponteiro para uma **person**, liberar memória para essa pessoa e, em seguida, liberar memória recursivamente para todos os seus ancestrais.

- Como esta é uma função recursiva, você deve primeiro lidar com o caso base. Se a entrada para a função for **NULL**, não há nada para liberar, então sua função pode retornar imediatamente.
- Caso contrário, você deve liberar (**free**) recursivamente os pais da pessoa antes de liberar a criança.

## Dicas

Você pode achar a função **rand()** útil para atribuir alelos aleatoriamente. Esta função retorna um número inteiro entre **0** e **RAND\_MAX**, ou **32767**.

Em particular, para gerar um número pseudoaleatório que seja **0** ou **1**, você pode usar a expressão **rand()%2**.

Lembre-se, para alocar memória para uma pessoa em particular, podemos usar **malloc(n)**, que toma um tamanho como argumento e alocará n bytes de memória.

Lembre-se, para acessar uma variável por meio de um ponteiro, podemos usar a notação de seta.

- Por exemplo, se p é um ponteiro para uma pessoa, então um ponteiro para o primeiro pai dessa pessoa pode ser acessado por **p-> parents[0]**.

## Como testar seu código

Ao executar **./inheritance**, seu programa deve aderir às regras descritas em segundo plano. A criança deve ter dois alelos, um de cada pai. Cada um dos pais deve ter dois alelos, um de cada um de seus pais.

Por exemplo, no exemplo abaixo, a criança na Geração 0 recebeu um alelo O de ambos os pais da Geração 1. O primeiro pai recebeu um A do primeiro avô e um O do segundo avô. Da mesma forma, o segundo pai recebeu um O e um B de seus avós.

```
$./inheritance
Generation 0, blood type OO
 Generation 1, blood type AO
 Generation 2, blood type OA
 Generation 2, blood type BO
 Generation 1, blood type OB
 Generation 2, blood type AO
 Generation 2, blood type BO
```

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/inheritance
```

Execute o seguinte para avaliar o estilo do seu código usando **style50** .

```
style50 inheritance.c
```

## ANOTAÇÕES MÓDULO 6

# Conceitos Básicos de Python

Hoje vamos aprender uma nova linguagem de programação chamada Python. Uma linguagem mais recente que C, ela possui recursos adicionais e também simplicidade, o que leva à sua popularidade.

O código-fonte em Python parece muito mais simples do que C. Na verdade, para imprimir “hello, world”, tudo o que precisamos escrever é:

```
print("hello, world")
```

- Observe que, ao contrário de C, não precisamos especificar uma nova linha na função de **print** ou usar um ponto e vírgula para terminar nossa linha.
- Para escrever e executar este programa, usaremos o IDE CS50, salvaremos um novo arquivo como `hello.py` apenas com a linha acima e executaremos o comando `python hello.py`.

Podemos obter strings de um usuário:

```
answer = get_string("Qual o seu nome? ")
print("ola, " + answer)
```

- Também precisamos importar a versão Python da biblioteca CS50, **cs50** , apenas para a função **get\_string**, então nosso código ficará assim:

```
from cs50 import get_string
```

```
answer = get_string("Qual o seu nome? ")
print("ola, " + answer)
```

- Criamos uma variável chamada **answer** sem especificar o tipo e podemos combinar, ou concatenar, duas strings com o operador `+` antes de passá-lo para **print**.

Podemos usar a sintaxe para **strings de formato** , `f "..."` , para inserir variáveis. Por exemplo, poderíamos ter escrito `print(f "hello, {answer}")` para inserir o valor de **answer** em nossa string colocando-o entre chaves.

Podemos criar variáveis apenas com **counter = 0**. Ao atribuir o valor de **0**, estamos definindo implicitamente o tipo como um inteiro, portanto, não precisamos especificar o tipo. Para incrementar uma variável, podemos usar **counter = counter + 1** ou **counter + = 1**.

As condições são semelhantes a:

```
if x < y:
 print("X é menor que Y")
elif x > y:
 print("X é maior que Y")
else:
 print("X é igual a Y")
```

- Ao contrário de C, onde as chaves são usadas para indicar blocos de código, o recuo exato de cada linha é o que determina o nível de aninhamento em Python.
- E em vez de **else if**, apenas dizemos **elif**.

As expressões booleanas também são ligeiramente diferentes:

```
while True:
 print("hello, world")
```

- Ambos **True** e **False** são capitalizados em Python.

Podemos escrever um loop com uma variável:

```
i = 0
while i < 3:
 print("hello, world")
 i += 1
```

Também podemos usar um **for** loop , onde podemos fazer algo para cada valor em uma lista:

```
for i in [0, 1, 2]:
 print("cough")
```

- Listas em Python, **[0, 1, 2]**, são como matrizes em C.
- Este **for** loop irá definir a variável **i** para o primeiro elemento, **0** , executar, em seguida, para o segundo elemento, **1**, executar, e assim por diante.
- E podemos usar uma função especial, **range**, para obter algum número de valores, como em **for i in range(3):**. **range(3)** nos dará uma lista de até, mas não incluindo **3**, com os valores **0**, **1** e **2**, que podemos usar. **range()** tem outras opções também, então podemos ter listas que começam com valores diferentes e têm incrementos diferentes entre os valores. Olhando a documentação , por exemplo, podemos usar **range(0, 101, 2)** para obter um intervalo de **0** a **100** (já que o segundo valor é exclusivo), incrementando em **2** de cada vez.
- Para imprimir **i** , também podemos escrever **print(i)**.
- Como geralmente há várias maneiras de escrever o mesmo código em Python, as formas mais comumente usadas e aceitas são chamadas de **Pythonic**.

Em Python, existem muitos tipos de dados integrados:

- **bool** , **True** ou **False**
- **float** , números reais
- **int** , inteiros
- **str** , strings

Enquanto C é uma **linguagem fortemente tipada**, onde precisamos especificar tipos, Python é **fracamente tipada**, onde o tipo está implícito nos valores.

Outros tipos em Python incluem:

- **range**, sequência de números
- **list**, sequência de valores mutáveis ou valores que podemos alterar
  - E as listas, mesmo que sejam como arrays em C, podem aumentar e diminuir automaticamente em Python
- **tuple**, coleção de valores ordenados como coordenadas xey ou longitude e latitude
- **dict**, dicionários, coleção de pares chave / valor, como uma tabela hash
- **set**, coleção de valores únicos ou valores sem duplicatas

A biblioteca CS50 para Python inclui:

- `get_float`
- `get_int`
- `get_string`

E podemos importar funções uma de cada vez ou todas juntas:

```
from cs50 import get_float
from cs50 import get_int
from cs50 import get_string

import cs50

from cs50 import get_float, get_int, get_string
```

## Exemplos

Como o Python inclui muitos recursos, bem como bibliotecas de código escrito por outros, podemos resolver problemas em um nível mais alto de abstração, em vez de implementar todos os detalhes nós mesmos.

Podemos desfocar uma imagem com:

```
from PIL import Image, ImageFilter

before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BoxBlur(1))
after.save("out.bmp")
```

- No Python, incluímos outras bibliotecas com **import** e aqui vamos importar os nomes de **Image** e **ImageFilter** da biblioteca PIL. (Outras pessoas escreveram esta biblioteca, entre outras, e disponibilizaram para todos nós baixarmos e usarmos.)
- **Image** é uma estrutura que não apenas possui dados, mas funções que podemos acessar com o “.”, como com **Image.open**.
- Abrimos uma imagem chamada **bridge.bmp**, chamamos uma função de filtro de desfoque e salvamos em um arquivo chamado **out.bmp**.

- E nós podemos executar isso com **python blur.py** depois de salvar para um arquivo chamado **blur.py**.

Podemos implementar um dicionário com:

```
words = set()
```

```
def load(dictionary):
 file = open(dictionary, "r")
 for line in file:
 words.add(line.rstrip())
 file.close()
 return True
```

```
def check(word):
 if word.lower() in words:
 return True
 else:
 return False
```

```
def size():
 return len(words)
```

```
def unload():
 return True
```

- Primeiro, criamos um novo conjunto chamado **words**.
- Observe que não precisamos de uma função main. Nosso programa Python será executado de cima para baixo. Aqui, queremos definir uma função, então usamos `def load()`. `load` terá um parâmetro, `dictionary`, e seu valor de retorno está implícito. Nós abrimos o arquivo com `open`, e iteramos sobre as linhas no arquivo com apenas `for line in file:`. Em seguida, removemos a nova linha no final da linha e a adicionamos à `words`. Observe que `line` é uma string, mas tem uma função `.rstrip` que podemos chamar.
- Então, para `check`, podemos apenas perguntar `if word.lower() in words`. Para o tamanho, podemos usar `len` para contar o número de elementos em nosso conjunto e, finalmente, para `unload`, não precisamos fazer nada, já que o Python gerencia a memória para nós.

Acontece que, embora implementar um programa em Python seja mais simples para nós, o tempo de execução de nosso programa em Python é mais lento do que nosso programa em C, pois a linguagem tem que trabalhar mais para nós com soluções de uso geral, como para gestão de memória.

Além disso, Python também é o nome de um programa chamado **interpretador**, que lê nosso código-fonte e o traduz para um código que nossa CPU pode entender, linha por linha.



Por exemplo, se nosso pseudocódigo da semana 0 estava em espanhol e não entendíamos espanhol, teríamos que traduzi-lo lentamente, linha por linha, para o inglês antes de podermos pesquisar um nome em uma lista telefônica:

```
1 Recoge guía telefónica
2 Abre a la mitad de guía telefónica
3 Ve la página
4 Si la persona está en la página
5 Llama a la persona
6 Si no, si la persona está antes de mitad de guía telefónica
7 Abre a la mitad de la mitad izquierda de la guía telefónica
8 Regresa a la línea 3
9 Si no, si la persona está después de mitad de guía telefónica
10 Abre a la mitad de la mitad derecha de la guía telefónica
11 Regresa a la línea 3
12 De lo contrario
13 Abandona
```

Portanto, dependendo de nossos objetivos, também teremos que considerar a troca de tempo humano ao escrever um programa que é mais eficiente, versus o tempo de execução do programa.

## Input, condições

Podemos obter a entrada do usuário com a função de input:

```
answer = input("Qual é seu nome? ")
print(f"olá, {answer}")
```

Podemos pedir ao usuário dois inteiros e adicioná-los:

```
from cs50 import get_int

Solicitar x ao usuário
x = get_int("x:")

Solicitar ao usuário y
y = get_int("y:")

Executar adição
print(x + y)
```

- Os comentários começam com # em vez de //.

Se chamarmos **input** nós mesmos, obtemos strings de volta para nossos valores:

```
Solicitar x ao usuário
x = input("x:")

Solicitar ao usuário y
y = input("y:")
```

```
Executar adição
print(x + y)
```

Portanto, precisamos fazer o **cast**, ou converter, cada valor de **input** em um **int** antes de armazená-lo:

```
Solicitar x ao usuário
x = int(input("x:"))
```

```
Solicitar ao usuário y
y = int(input ("y:"))
```

```
Executar adição
print(x + y)
```

- Mas se o usuário não digitou um número, precisaremos fazer ainda mais verificações de erro ou nosso programa travará. Portanto, geralmente queremos usar uma biblioteca comumente usada para resolver problemas como este.

Vamos dividir os valores:

```
Solicitar x ao usuário
x = int(input("x:"))
```

```
Solicitar ao usuário y
y = int(input("y:"))
```

```
Executar divisão
print(x/y)
```

- Observe que obtemos valores decimais de ponto flutuante de volta, mesmo se dividirmos dois inteiros.

E podemos demonstrar as condições:

```
from cs50 import get_int
```

```
x = get_int("x:")
y = get_int("y:")
```

```
if x < y
 print("x é menor que y")
elif x > y:
 print("x é maior que y")
else:
 print("x é igual a y")
```

Podemos importar bibliotecas inteiras e usar funções dentro delas como se fossem uma estrutura:

```
import cs50
```

```
x = cs50.get_int("x:")
y = cs50.get_int("y:")
```

- Se nosso programa precisar importar duas bibliotecas diferentes, cada uma com uma função **get\_int**, por exemplo, precisaríamos usar este método para funções de **namespace**, mantendo seus nomes em espaços diferentes para evitar que colidam.

Para comparar strings, podemos dizer:

```
from cs50 import get_string
```

```
s = get_string("Você concorda?")
```

```
if s == "Y" or s == "y":
 print("Concordo")
elif s == "N" or s == "n" :
 print("Não concordo.")
```

- Python não tem caracteres, então verificamos **Y** e outras letras como strings. Também podemos comparar strings diretamente com **==**. Finalmente, em nossas expressões booleanas, usamos **or** e **and** em vez de símbolos.
- Também podemos dizer **if s.lower() in ["y", "yes"]:** para verificar se nossa string está em uma lista, depois de convertê-la para minúsculas primeiro.

## Miau

Podemos melhorar as versões do **miau** também:

```
print("miau")
print("miau")
print("miau")
```

- Não precisamos declarar uma função principal, então apenas escrevemos a mesma linha de código três vezes.

Podemos definir uma função que podemos reutilizar:

```
for i in range(3):
 miau()
```

```
def miau():
 print("miau")
```

Mas isso causa um erro quando tentamos executá-lo: `NameError: name 'meow' is not defined`. Acontece que precisamos definir nossa função antes de usá-la, então podemos mover nossa definição de **miau** para o topo ou definir uma função principal primeiro:

```
def main():
 for i in range(3):
 miau()
```

```
def miau():
 print("miau")
```

```
main()
```

- Agora, quando realmente chamarmos nossa função **main**, a função **miau** já terá sido definida.

Nossas funções também podem receber inputs:

```
def main():
 miau(3)
def miau(n):
 for i in range(n):
 print("miau")
```

```
main()
```

- Nossa função **miau** recebe um parâmetro, **n**, e o passa para **range**.

## get\_positive\_int

Podemos definir uma função para obter um número inteiro positivo:

```
from cs50 import get_int
```

```
def main():
 i = get_positive_int()
 print(i)
```

```
def get_positive_int():
 while True:
 n = get_int("Inteiro Positivo: ")
 if n > 0:
 break
 return n
```

```
main()
```

- Uma vez que não há um do-while loop em Python como existe em C, temos um **while** loop que vai continuar infinitamente, e usar **break** para terminar o loop, assim que **n > 0**. Finalmente, a nossa função vai **return n**, no nosso nível de recuo original, fora do **while** loop.

- Observe que as variáveis em Python têm funções como **variável de escopo** padrão, o que significa que **n** pode ser inicializado em um loop, mas ainda pode ser acessado posteriormente na função.

## Mario

Podemos imprimir uma linha de pontos de interrogação na tela:

```
for i in range(4):
 print("?", end="")
 print()
```

- Quando imprimimos cada bloco, não queremos a nova linha automática, então podemos passar um argumento nomeado, também conhecido como argumento de palavra-chave, para a função de impressão, que especifica o valor para um parâmetro específico. Até agora, vimos apenas argumentos posicionais, onde os parâmetros são definidos com base em sua posição na chamada de função.
- Aqui, dizemos `end = ""` para especificar que nada deve ser impresso no final de nossa string. `end` também é um argumento opcional, que não precisamos passar, com um valor padrão de `\n`, que é o motivo pelo qual `print` geralmente adiciona uma nova linha para nós.
- Finalmente, depois de imprimir nossa linha com o loop, podemos chamar **print** sem nenhum outro argumento para obter uma nova linha.

Também podemos “multiplicar” uma string e imprimi-la diretamente com: **print("?" \* 4)**.

Podemos implementar loops aninhados:

```
for i in range(3):
 for j in range(3):
 print("#", end="")
 print()
```

## Overflow, imprecisão

Em Python, tentar causar um overflow de inteiros não funcionará:

```
i = 1
while True:
 print(i)
 i *= 2
```

- Vemos números cada vez maiores sendo impressos, já que o Python usa automaticamente cada vez mais memória para armazenar números para nós, ao contrário de C, em que os inteiros são fixados em um determinado número de bytes.

A imprecisão de ponto flutuante também ainda existe, mas pode ser evitada por bibliotecas que podem representar números decimais com quantos bits forem necessários.

# Listas, strings

Podemos fazer uma lista:

```
scores = [72, 73, 33]
```

```
print("Average: " + str(sum(scores) / len(scores)))
```

- Podemos usar **sum**, uma função incorporada ao Python, para somar os valores em nossa lista e dividi-la pelo número de pontuações, usando a função **len** para obter o comprimento da lista. Em seguida, convertemos o float em uma string antes de podermos concatená-lo e imprimi-lo.
- Podemos até adicionar a expressão inteira em uma string formatada para o mesmo efeito:

```
print(f"Average: {sum(scores) / len(scores)}")
```

Podemos adicionar itens a uma lista com:

```
from cs50 import get_int
```

```
scores = []
for i in range(3):
 scores.append(get_int("Score: "))
...
```

Podemos iterar sobre cada caractere em uma string:

```
from cs50 import get_string
```

```
s = get_string("Antes: ")
print("Depois: ", end="")
for c in s:
 print(c.upper(), end="")
print()
```

- Python irá iterar sobre cada caractere na string para nós com apenas **for c in s**.

Para tornar uma string maiúscula, também podemos simplesmente chamar `s.upper()`, sem ter que iterar nós mesmos sobre cada caractere.

## Argumentos de linha de comando, códigos de saída

Podemos aceitar argumentos de linha de comando com:

```
from sys import argv
```

```
if len(argv) == 2:
```

```
print(f"hello, {argv[1]}")
else:
 print("hello, world")
```

- Importamos **argv** do **sys**, ou módulo do sistema, integrado ao Python.
- Como **argv** é uma lista, podemos obter o segundo item com **argv[1]**, portanto, adicionar um argumento com o comando **python argv.py David** resultará em **olá, David** impresso.
- Como em C, **argv[0]** seria o nome do nosso programa, como **argv.py**.

Também podemos permitir que o Python itere sobre a lista para nós:

```
from sys import argv

for arg in argv:
 print(arg)
```

Também podemos retornar códigos de saída quando nosso programa for encerrado:

```
import sys

if len(sys.argv) != 2:
 print("missing command-line argument")
 sys.exit(1)
print(f"ola, {sys.argv[1]}")
sys.exit(0)
```

- Importamos todo o módulo **sys** agora, já que estamos usando vários componentes dele. Agora podemos usar **sys.arg** e **sys.exit()** para sair de nosso programa com um código específico.

## Algoritmos

Podemos implementar a pesquisa linear apenas verificando cada elemento em uma lista:

```
import sys

numbers = [4, 6, 8, 2, 7, 5, 0]

if 0 in numbers:
 print("Encontrado")
 sys.exit(0)

print("Nao Encontrado")
sys.exit(1)
```

- Com **if 0 in numbers:**, estamos pedindo ao Python para verificar a lista para nós.

Uma lista de strings também pode ser pesquisada com:

```
names = ["Bill", "Charlie", "Fred", "George", "Ginny", "Percy",
"Ron"]
```

```
if "Ron" in names:
 print("Found")
else:
 print("Not found")
```

Se tivermos um dicionário, um conjunto de pares de chaves-valores, também podemos verificar se há uma chave específica e examinar o valor armazenado para ela:

```
from cs50 import get_string
```

```
people = {
 "Brian": "+1-617-495-1000",
 "David": "+1-949-468-2750"
}
```

```
name = get_string("Nome: ")
if name in people:
 print(f"Numero: {people[name]}")
```

- Primeiro declaramos um dicionário, **people**, onde as chaves são strings de cada nome que queremos armazenar, e o valor que queremos associar a cada chave é uma string de um número de telefone correspondente.
- Então, usamos **if name in people:** para pesquisar as chaves do nosso dicionário por um **name**. Se a chave existe, então podemos obter o valor com a notação de colchetes, **people[name]**, bem como indexar em um array com C, exceto que aqui usamos uma string em vez de um inteiro.

Os dicionários, assim como os conjuntos, são normalmente implementados em Python com uma estrutura de dados como uma tabela hash, para que possamos ter uma pesquisa de tempo quase constante. Novamente, temos a desvantagem de ter menos controle sobre exatamente o que acontece nos bastidores, como poder escolher uma função hash, com a vantagem de ter que fazer menos trabalho nós mesmos.

A troca de duas variáveis também pode ser feita simplesmente atribuindo os dois valores ao mesmo tempo:

```
x = 1
y = 2

print(f"x is {x}, y is {y}")
x, y = y, x
print(f"x is {x}, y is {y}")
```

Em Python, não temos acesso a ponteiros, o que nos protege de cometer erros com a memória.



# Arquivos

Vamos abrir um arquivo CSV:

```
import csv

from cs50 import get_string

file = open("phonebook.csv", "a")

name = get_string("Nome: ")
number = get_string("Numero: ")

writer = csv.writer(file)
writer.writerow([name, number])

file.close()
```

- Acontece que Python também tem uma biblioteca **csv** que nos ajuda a trabalhar com arquivos CSV, então, depois de abrir o arquivo para anexar, podemos chamar **csv.writer** para criar um **writer** do arquivo, o que oferece funcionalidade adicional, como **writer.writerow** para escrever uma lista como uma linha.

Podemos usar a palavra-chave **with**, que fechará o arquivo para nós quando terminarmos:

```
...
with open("phonebook.csv", "a") as file:
 writer = csv.writer(file)
 writer.writerow((name, number))
```

Podemos abrir outro arquivo CSV...

```
import csv

houses = {
 "Gryffindor": 0,
 "Hufflepuff": 0,
 "Ravenclaw": 0,
 "Slytherin": 0
}

with open("Sorting Hat (Responses) - Form Responses 1.csv", "r") as file:
 reader = csv.reader(file)
 next(reader)
 for row in reader:
 house = row[1]
 houses[house] += 1
```

```
for house in houses:
 print(f"{house}: {houses[house]}")
```

- Usamos a função de **reader** da biblioteca **csv**, pulamos a linha do cabeçalho com **next(reader)** e, em seguida, iteramos sobre cada uma das linhas restantes.
- O segundo item em cada linha, **row[1]**, é a string de uma casa, então podemos usá-la para acessar o valor armazenado em **houses** para aquela chave e adicionar um a ela.
- Por fim, imprimiremos a contagem de cada casa.

## Mais bibliotecas

Em nosso próprio Mac ou PC, podemos abrir um terminal após instalar o Python e usar outra biblioteca para converter texto em fala:

```
importar pyttsx3

engine = pyttsx3.init ()
engine.say("olá, mundo")
engine.runAndWait()
```

- Lendo a documentação, podemos descobrir como inicializar a biblioteca e dizer uma string.
- Podemos até passar uma string de formato com **engine.say (f "olá, {nome}")** para dizer alguma entrada.

Podemos usar outra biblioteca, **face\_recognition**, para encontrar rostos nas imagens:

```
Encontre rostos na imagem
#
https://github.com/ageitgey/face_recognition/blob/master/examples/find_faces_in_picture.py
from PIL import Image
import face_recognition

Carregue o arquivo jpg em uma matriz numpy
image = face_recognition.load_image_file("office.jpg")

Encontre todos os rostos na imagem usando o modelo baseado em HOG padrão.
Este método é bastante preciso, mas não tão preciso quanto o modelo CNN e não é acelerado por GPU.
Veja também: find_faces_in_picture_cnn.py
face_locations = face_recognition.face_locations(image)

for face_location in face_locations:
 # Imprima a localização de cada rosto nesta imagem: cima, direita, baixo, esquerda
```

```

top, right, bottom, left = face_location

Você pode acessar o próprio rosto desta forma:
face_image = image [top: bottom, left: right]
pil_image = Image.fromarray(face_image)
pil_image.show()

```

Com [recognize.py](#) podemos escrever um programa que encontra uma correspondência para um rosto específico.

Podemos criar um código QR, ou código de barras bidimensional, com outra biblioteca:

```

import os
import qrcode

img = qrcode.make("https://youtu.be/oHg5SJYRHA0")
img.save("qr.png", "PNG")
os.system("open qr.png")

```

Podemos reconhecer a entrada de áudio de um microfone:

```

import speech_recognition

Obtenha áudio do microfone
recognizer = speech_recognition.Recognizer()
with speech_recognition.Microphone() como fonte:
 print("Say something:")
 audio = reconhecerizer.listen(fonte)

Reconhecer fala usando o Google Speech Recognition
print("Você disse:")
print(recognizer.recognize_google(audio))

```

Estamos seguindo a documentação da biblioteca para ouvir nosso microfone e convertê-lo em texto.

Podemos até adicionar lógica adicional para respostas básicas:

```

...
words = recognizer.recognize_google(audio)

Responda ao discurso
if "hello" in words:
 print("Olá para você também :)")
elif "how are you" em palavras:
 print("Estou bem, obrigado!")
elif "goodbye" em palavras:
 print("Tchau pra você também")
else:
 print("Hm?")

```

- Finalmente, usamos outro programa mais sofisticado para gerar deepfakes, ou vídeos que parecem realistas, mas gerados por computador, de várias personalidades.
- Tirando proveito de todas essas bibliotecas disponíveis gratuitamente online, podemos facilmente adicionar funcionalidades avançadas aos nossos próprios aplicativos.

## Hello World usando Python

Implemente um código que imprima da seguinte forma, capturando um valor do usuário:

```
$ python hello.py
What is your name?
David
hello, David
```

## Especificação

Escreva em um arquivo chamado `hello.py` em `~/pset6/hello`, um programa que imprima o nome que o usuário escrever e imprima "hello". Vale lembrar que é igual aquele exercício de Introdução da IDE que você fez em C, lembra?

## Colocando para funcionar

Seu programa deve comportar da seguinte forma:

```
$ python hello.py
What is your name?
Emma
hello, Emma
```

## Testando

Enquanto **check50** pode ser utilizado neste problema, te aconselho primeiro a testar seu código próprio.

- Execute seu programa com o comando **python hello.py** e aperte Enter. Escreva David e aperte Enter. A saída do seu programa deve ser **hello, David**.
- Execute seu programa com o comando **python hello.py** e aperte Enter. Escreva Brian e aperte Enter. A saída do seu programa deve ser **hello, Brian**.

Execute o comando abaixo para verificar se o seu código está correto, utilizando **check50**.

```
check50 cs50/problems/2021/x/sentimental/hello
```

Execute o comando abaixo para verificar o estilo, utilizando **style50**.

```
style50 hello.py
```

O código será analisado pela indentação.

## EXERCÍCIOS MÓDULO 6

### Exercício 1: Mario



Implemente um programa que imprima uma meia-pirâmide de uma altura(height) especificada, conforme a seguir.

```
$ python mario.py
Height: 4
#
##
###
####
```

## Especificação

- Escreva, em um arquivo chamado `mario.py` em `~/pset6/mario/less/`, um programa que recria a meia-pirâmide usando hashes ( `#` ) para blocos, exatamente como você fez em **um dos exercícios do Módulo 1**, exceto que seu programa desta vez deve ser escrito em Python.
- Para tornar as coisas mais interessantes, primeiro solicite ao usuário `get_int` para a altura da meia-pirâmide, um número inteiro positivo entre **1** e **8**, inclusive.
- Se o usuário não fornecer um número inteiro positivo não superior a **8**, você deve solicitar o mesmo novamente.
- Em seguida, gere (com a ajuda de `print` e um ou mais loops) a meia-pirâmide desejada.
- Tome cuidado para alinhar o canto esquerdo inferior da sua meia pirâmide com a borda esquerda da janela do terminal.

## Uso

Seu programa deve se comportar conforme o exemplo abaixo.

```
$ python mario.py
Height: 4
```

```
#
##
###
####
```

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python mario.pye** e aguarde uma solicitação de entrada. Digite -1 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 0 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 1 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto inferior esquerdo do seu terminal e que não haja espaços extras no final de cada linha.

```
#
```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 2 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```
#
##
```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 8 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```
#
##
###
####
#####
#####
#####
#####
```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 9 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número. Em seguida, digite 2 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a

pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```

##
```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite foo e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Não digite nada e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.

Execute o seguinte para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/sentimental/mario/less
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**

```
style50 mario.py
```

Este problema será classificado apenas de acordo com os eixos de correção e estilo.

## Exercício 2: Mario(versão desafiadora)



Implemente um programa que imprima uma meia-pirâmide dupla de uma altura(height) especificada, conforme a seguir.

```
$ python mario.py
```

```
Height: 4
```

```
 # #
 ## ##

####
```

## Especificação

- Escreva, em um arquivo chamado **mario.py** em **~/pset6/mario/more/**, um programa que recria essas meias-pirâmides usando hashes ( **#** ) para blocos, exatamente como você fez **em um dos exercícios do Módulo 1**, exceto que seu programa desta vez deve ser escrito em Python.

- Para tornar as coisas mais interessantes, primeiro solicite ao usuário **get\_int** para a altura da meia pirâmide, um número inteiro positivo entre **1** e **8**, inclusive. (A altura das meias-pirâmides ilustradas acima é **4**, a largura de cada meia-pirâmide **4**, com uma lacuna de tamanho **2** separando-as).
- Se o usuário não fornecer um número inteiro positivo não superior a **8**, você deve solicitar o mesmo novamente.
- Em seguida, gere (com a ajuda de print e um ou mais loops) as meias-pirâmides desejadas.
- Tome cuidado para alinhar o canto inferior esquerdo da pirâmide com a borda esquerda da janela do terminal e certifique-se de que haja dois espaços entre as duas pirâmides e que não haja espaços adicionais após o último conjunto de hashes em cada fileira.

## Uso

Seu programa deve se comportar conforme o exemplo abaixo.

```
$ python mario.py
Height: 4
 # #
 ## ##
###
####
```

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite -1 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 0 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 1 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```
#
```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 2 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```
#
##
```



- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 8 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```

#
##
###
####
#####
#####
#####
#####
#####

```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite 9 e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número. Em seguida, digite 2 e pressione enter. Seu programa deve gerar a saída abaixo. Certifique-se de que a pirâmide esteja alinhada com o canto esquerdo inferior do seu terminal e que não haja espaços extras no final de cada linha.

```

#
##

```

- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Digite foo e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python mario.py** e aguarde uma solicitação de entrada. Não digite nada e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/sentimental/mario/more
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 mario.py
```

Este problema será classificado apenas de acordo com os eixos de correção e estilo.

### Exercício 3: Dinheiro

Implemente um programa que calcule o número mínimo de moedas necessárias para dar o troco a um usuário.

```

$ python cash.py
Change owed: 0.41

```

# Especificação

Escreva, em um arquivo chamado **cash.py** em **~/pset6/cash/**, um programa que primeiro pergunte ao usuário quanto dinheiro é devido e, em seguida, diga o número mínimo de moedas com as quais esse troco pode ser feito, exatamente como você fez no **Exercício do Módulo 1(C)** exceto que seu programa, desta vez, deve ser escrito em Python.

Use **get\_float** da biblioteca CS50 para obter a entrada do usuário e **print** para produzir sua resposta. Suponha que as únicas moedas disponíveis sejam de 25¢, 10¢, 5¢ e centavos 1¢.

- Pedimos que você use **get\_float** para que possa lidar com dólares e centavos, embora sem o cifrão. Em outras palavras, se algum cliente deve \$9.75 (como no caso em que um jornal custa 25 centavos, mas o cliente paga com uma nota de \$10), suponha que a entrada de seu programa será de **9.75** e não de \$9.75 ou 975. No entanto, se algum cliente deve exatamente \$ 9, suponha que o input de seu programa será **9.00** ou apenas **9**, mas, novamente, não \$9 ou 900. Claro, pela natureza dos valores de ponto flutuante, seu programa provavelmente funcionará com inputs como 9.0 e 9.000 também; você não precisa se preocupar em verificar se o input do usuário está “formatada” como o dinheiro deveria estar.

Se o usuário não fornecer um valor não negativo, seu programa deve solicitar novamente ao usuário um valor válido até que o usuário concorde.

A propósito, para que possamos automatizar alguns testes do seu código, pedimos que a última linha de saída do seu programa seja apenas o número mínimo de moedas possível: um inteiro seguido por uma nova linha.

## Uso

Seu programa deve se comportar conforme o exemplo abaixo.

```
$ python cash.py
Change owed: 0.41
4
```

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **0.41** e pressione Enter. Seu programa deve gerar 4.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **0.01** e pressione Enter. Seu programa deve gerar 1.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **0.15** e pressione Enter. Seu programa deve gerar 2.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **1.60** e pressione enter. Seu programa deve gerar **7**.

- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **23** pressione enter. Seu programa deve gerar 92.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **4.2** e pressione enter. Seu programa deve gerar 18.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **-1** e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Digite **foo** pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.
- Execute seu programa como **python cash.py** e aguarde um prompt de entrada. Não digite nada e pressione enter. Seu programa deve rejeitar esta entrada como inválida, solicitando novamente que o usuário digite outro número.

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/sentimental/cash
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 cash.py
```

Este problema será classificado apenas de acordo com os eixos de correção e estilo.

## Exercício 4: Crédito(versão desafiadora)

Implemente um programa que determine se um número de cartão de crédito fornecido é válido de acordo com o algoritmo de Luhn.

```
$ python credit.py
Number: 378282246310005
AMEX
```

## Especificação

Em **credit.py** em **~/pset6/credit**, escreva um programa que solicite ao usuário um número de cartão de crédito e, em seguida, informe (via print ) se é um número de cartão American Express, MasterCard ou Visa válido, exatamente como você fez no **Exercício do Módulo 1(C)** , exceto que seu programa, desta vez, deve ser escrito em Python.

Para que possamos automatizar alguns testes do seu código, pedimos que a última linha de saída do seu programa seja **AMEX\n ou MASTERCARD\n ou VISA\n ou INVALID\n** , nada mais, nada menos.

Para simplificar, você pode assumir que a entrada do usuário será inteiramente numérica (ou seja, sem hífen, como pode ser impresso em um cartão real).

Melhor usar **get\_int** ou **get\_string** da biblioteca do CS50 para obter a entrada dos usuários, dependendo de como você decidir implementar este.

# Uso

Seu programa deve se comportar conforme o exemplo abaixo.

```
$ python credit.py
Number: 378282246310005
AMEX
```

## Dicas

É possível usar expressões regulares para validar a entrada do usuário. Você pode usar o módulo `re` do Python, por exemplo, para verificar se a entrada do usuário é de fato uma sequência de dígitos com o comprimento correto.

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **378282246310005** e pressione Enter. Seu programa deve gerar AMEX.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **371449635398431** e pressione Enter. Seu programa deve gerar AMEX.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **5555555555554444** e pressione Enter. Seu programa deve gerar MASTERCARD.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **5105105105105100** e pressione Enter. Seu programa deve gerar MASTERCARD.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **4111111111111111** e pressione Enter. Seu programa deve gerar VISA.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **4012888888881881** e pressione Enter. Seu programa deve gerar VISA.
- Execute seu programa como **python credit.py** e aguarde uma solicitação de entrada. Digite **1234567890** e pressione Enter. Seu programa deve gerar INVALID.

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/sentimental/credit
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 credit.py
```

Este problema será classificado apenas de acordo com os eixos de correção e estilo.

## Exercício 5: Legibilidade

Implemente um programa que calcule o nível escolar aproximado necessário para compreender algum texto, conforme a seguir.

```
$ python readability.py
```

```
Text: Congratulations! Today is your day. You're off to Great
Places! You're off and away!
```

```
Grade 3
```

## Especificação

Escreva, em um arquivo chamado `readability.py` em `~/pset6/legibilidade/`, um programa que primeiro pede ao usuário para digitar algum texto e, em seguida, emite o nível escolar necessário para compreender o texto, de acordo com a fórmula Coleman-Liau, exatamente como você fez em um dos exercícios dos módulos anteriores, exceto que seu programa desta vez deve ser escrito em Python.

Lembre-se de que o índice Coleman-Liau é calculado com  $0.0588 * L - 0.296 * S - 15.8$ , onde  $L$  é o número médio de letras por 100 palavras no texto e  $S$  é o número médio de sentenças por 100 palavras no texto.

Use `get_string` da Biblioteca CS50 para obter a entrada do usuário e `print` para produzir sua resposta.

Seu programa deve contar o número de letras, palavras e frases do texto. Você pode assumir que uma letra é qualquer caractere minúsculo de `a` a `z` ou qualquer caractere maiúsculo de `A` a `Z`, qualquer sequência de caracteres separados por espaços deve contar como uma palavra e que qualquer ocorrência de um ponto final, ponto de exclamação ou ponto de interrogação indica o final de uma frase.

Seu programa deve imprimir como saída "**Grade X**", onde **X** é o nível escolar calculado pela fórmula de Coleman-Liau, arredondado para o número inteiro mais próximo.

Se o número do índice resultante for 16 ou superior (equivalente ou superior ao nível de leitura de graduação sênior), seu programa deve produzir "**Grade 16+**" em vez de fornecer o número do índice exato. Se o número do índice for menor que 1, seu programa deve imprimir "**Before Grade 1**".

## Uso

Seu programa deve se comportar conforme o exemplo abaixo.

```
$ python readability.py
```

```
Text: Congratulations! Today is your day. You're off to Great
Places! You're off and away!
```

```
Grade 3
```

# Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **One fish. Two fish. Red fish. Blue fish.** e pressione Enter. Seu programa deve produzir Before Grade 1.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **Would you like them here or there? I would not like them here or there. I would not like them anywhere.** e pressione Enter. Seu programa deve produzir Grade 2.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **Congratulations! Today is your day. You're off to Great Places! You're off and away!** e pressione Enter. Seu programa deve produzir Grade 3.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Tipo em **Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard.** e pressione Enter. Seu programa deve produzir Grade 5.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.** e pressione Enter. Seu programa deve produzir Grade 7.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digitar **Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice "without pictures or conversation?"**. e pressione Enter. Seu programa deve produzir Grade 8.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow. When it healed, and Jem's fears of never being able to play football were assuaged, he was seldom self-conscious about his injury. His left arm was somewhat shorter than his right; when he stood or walked, the back of his hand was at right angles to his body, his thumb parallel to his thigh.** e pressione Enter. Seu programa deve produzir Grade 8.

- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **There are more things in Heaven and Earth, Horatio, than are dreamt of in your philosophy.** e pressione Enter. Seu programa deve produzir Grade 9.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **It was a bright cold day in April, and the clocks were striking thirteen. Winston Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped quickly through the glass doors of Victory Mansions, though not quickly enough to prevent a swirl of gritty dust from entering along with him.** e pressione Enter. Seu programa deve produzir Grade 10.
- Execute seu programa como **python readability.py** e aguarde um prompt de entrada. Digite **A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains.** e pressione Enter. Seu programa deve produzir Grade 16+.

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/problems/2021/x/sentimental/legibilidade
```

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

```
style50 readability.py
```

Este problema será classificado apenas de acordo com os eixos de correção e estilo.

## Exercício 6: DNA

Implemente um programa que identifique uma pessoa com base em seu DNA, conforme a seguir.

```
$ python dna.py databases/large.csv sequences/5.txt
Lavende
```

## Começando

Veja como baixar esse problema em seu próprio [IDE CS50](#). Faça login no CS50 IDE e, em uma janela de terminal, execute cada um dos itens abaixo.

- Navegue até o diretório **pset6** que já deve existir.
- Execute <https://cdn.cs50.net/2020/fall/psets/6/dna/dna.zip> para baixar um arquivo ZIP (compactado) com a distribuição deste problema.
- Execute **unzip dna.zip** para descompactar esse arquivo.
- Execute **rm dna.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.

- Execute ls. Você deve ver um diretório chamado **dna** , que estava dentro desse arquivo ZIP.
- Execute cd dna para mudar para esse diretório.
- Execute ls. Você deve ver um diretório de **databases** de amostra e um diretório de chamado **sequences**.

## Background

O DNA, o portador de informações genéticas em seres vivos, tem sido usado na justiça criminal há décadas. Mas como, exatamente, o perfil de DNA funciona? Dada uma sequência de DNA, como os investigadores forenses podem identificar a quem ela pertence?

Bem, o DNA é na verdade apenas uma sequência de moléculas chamadas nucleotídeos, arranjadas em uma forma particular (uma dupla hélice). Cada nucleotídeo do DNA contém uma de quatro bases diferentes: adenina (A), citosina (C), guanina (G) ou timina (T). Cada célula humana possui bilhões desses nucleotídeos organizados em sequência. Algumas partes dessa sequência (ou seja, genoma) são iguais, ou pelo menos muito semelhantes, em quase todos os humanos, mas outras partes da sequência têm uma diversidade genética maior e, portanto, variam mais na população.

Um lugar onde o DNA tende a ter alta diversidade genética é em Short Tandem Repeats (STRs). Um STR é uma sequência curta de bases de DNA que tende a se repetir consecutivamente inúmeras vezes em locais específicos dentro do DNA de uma pessoa. O número de vezes que um determinado STR se repete varia muito entre os indivíduos. Nas amostras de DNA abaixo, por exemplo, Alice tem o STR AGAT repetido quatro vezes em seu DNA, enquanto Bob tem o mesmo STR repetido cinco vezes.

**Alice:** CTAGATAGATAGATAGATGACTA

**Bob:** CTAGATAGATAGATAGATAGATT

O uso de vários STRs, em vez de apenas um, pode melhorar a precisão do perfil de DNA. Se a probabilidade de duas pessoas terem o mesmo número de repetições para um único STR for 5% e o analista olhar para 10 STRs diferentes, a probabilidade de duas amostras de DNA corresponderem puramente ao acaso é de cerca de 1 em 1 quatrilhão (assumindo todos os STRs são independentes entre si). Portanto, se duas amostras de DNA corresponderem ao número de repetições para cada um dos STRs, o analista pode ter certeza de que vieram da mesma pessoa. CODIS, o banco de dados de DNA do FBI , usa 20 STRs diferentes como parte de seu processo de criação de perfil de DNA.

Qual seria a aparência desse banco de dados de DNA? Bem, em sua forma mais simples, você poderia imaginar a formatação de um [banco de dados de DNA](#) como um arquivo CSV,



em que cada linha corresponde a um indivíduo e cada coluna corresponde a um determinado STR.

```
name,AGAT,AATG,TATC
Alice,28,42,14
Bob,17,22,19
Charlie,36,18,25
```

Os dados no arquivo acima sugerem que Alice tem a sequência **AGAT** repetida 28 vezes consecutivamente em algum lugar de seu DNA, a sequência **AATG** repetida 42 vezes e **TATC** repetida 14 vezes. Enquanto isso, Bob tem esses mesmos três STRs repetidos 17 vezes, 22 vezes e 19 vezes, respectivamente. E Charlie tem esses mesmos três STRs repetidos 36, 18 e 25 vezes, respectivamente.

Portanto, dada uma sequência de DNA, como você pode identificar a quem ela pertence? Bem, imagine que você buscou na sequência de DNA a sequência consecutiva mais longa de **AGATs** repetidos e descobriu que a sequência mais longa tinha 17 repetições. Se você descobrir que a sequência mais longa de **AATG** tem 22 repetições, e a sequência mais longa de **TATC** tem 19 repetições, isso forneceria uma boa evidência de que o DNA era de Bob. Claro, também é possível que, depois de fazer as contagens para cada um dos STRs, elas não correspondam a ninguém em seu banco de dados de DNA, nesse caso, você não tem uma correspondência.

Na prática, como os analistas sabem em qual cromossomo e em qual local no DNA um STR será encontrado, eles podem localizar sua pesquisa em apenas uma seção estreita do DNA. Mas vamos ignorar esse detalhe para este problema.

Sua tarefa é escrever um programa que pegará uma sequência de DNA e um arquivo CSV contendo contagens de STR para uma lista de indivíduos e, em seguida, enviará a quem o DNA (mais provavelmente) pertence.

## Especificação

Em um arquivo chamado **dna.py** em `~/pset6/dna/`, implemente um programa que identifique a quem pertence uma sequência de DNA.

- O programa deve exigir como seu primeiro argumento de linha de comando o nome de um arquivo CSV contendo as contagens STR para uma lista de indivíduos e deve exigir como seu segundo argumento de linha de comando o nome de um arquivo de texto contendo a sequência de DNA a ser identificada.
  - Se o seu programa for executado com o número incorreto de argumentos da linha de comando, ele deverá imprimir uma mensagem de erro de sua escolha (com `print`). Se o número correto de argumentos for fornecido, você pode assumir que o primeiro argumento é realmente o nome do arquivo de um arquivo CSV válido e que o segundo argumento é o nome do arquivo de um arquivo de texto válido.
- Seu programa deve abrir o arquivo CSV e ler seu conteúdo na memória.

- Você pode presumir que a primeira linha do arquivo CSV serão os nomes das colunas. A primeira coluna será a palavra name e as colunas restantes serão as próprias sequências STR.
- Seu programa deve abrir a sequência de DNA e ler seu conteúdo na memória.
- Para cada um dos STRs (da primeira linha do arquivo CSV), seu programa deve calcular a execução mais longa de repetições consecutivas do STR na sequência de DNA para identificar.
- Se a contagem de STR corresponder exatamente a qualquer um dos indivíduos no arquivo CSV, seu programa deve imprimir o nome do indivíduo correspondente.
  - Você pode presumir que as contagens de STR não corresponderão a mais de um indivíduo.
  - Se as contagens de STR não corresponderem exatamente a qualquer um dos indivíduos no arquivo CSV, seu programa deve imprimir "No match".

## Uso

Seu programa deve se comportar conforme o exemplo abaixo:

```
$ python dna.py databases/large.csv sequences/5.txt
Lavender
```

```
$ python dna.py
Usage: python dna.py data.csv sequence.txt
```

```
$ python dna.py data.csv
Usage: python dna.py data.csv sequence.txt
```

## Dicas

O [módulo csv do Python](#) é útil para ler arquivos CSV na memória. Você pode aproveitar as vantagens de [csv.reader](#) ou [csv.DictReader](#).

As funções [open](#) e [read](#) podem ser úteis para ler arquivos de texto na memória.

Considere quais estruturas de dados podem ser úteis para manter o rastreamento das informações em seu programa. Uma [list](#) ou um [dictionary](#) pode ser útil.

Strings em Python podem ser “fatiadas” (acessar uma substring particular dentro de uma string). Se `s` for uma string, então `s[i: j]` retornará uma nova string com apenas os caracteres de `s` começando do caractere `i` até (mas não incluindo) o caractere `j`.

Pode ser útil começar escrevendo uma função que, dada uma sequência de DNA e um STR como entradas, retorne o número máximo de vezes que o STR se repete. Você pode então usar essa função em outras partes do seu programa!

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a primeiro testar seu código por conta própria para cada um dos itens a seguir.

- Execute seu programa como **python dna.py databases/small.csv sequences/1.txt**. Seu programa deve gerar Bob.
- Execute seu programa como **python dna.py databases/small.csv sequences/2.txt**. Seu programa deve imprimir No match.
- Execute seu programa como **python dna.py databases/small.csv sequences/3.txt**. Seu programa deve exibir No match.
- Execute seu programa como **python dna.py databases/small.csv sequences/4.txt**. Seu programa deve gerar Alice.
- Execute seu programa como **python dna.py databases/large.csv sequences/5.txt**. Seu programa deve produzir Lavender.
- Execute seu programa como **python dna.py databases/large.csv sequences/6.txt**. Seu programa deve gerar Luna.
- Execute seu programa como **python dna.py databases/large.csv sequences/7.txt**. Seu programa deve gerar Ron.
- Execute seu programa como **python dna.py databases/large.csv sequences/8.txt**. Seu programa deve gerar Ginny.
- Execute seu programa como **python dna.py databases/large.csv sequences/9.txt**. Seu programa deve gerar Draco.
- Execute seu programa como **python dna.py databases/large.csv sequences/10.txt**. Seu programa deve imprimir Albus.
- Execute seu programa como **python dna.py databases/large.csv sequences/11.txt**. Seu programa deve gerar Hermione.
- Execute seu programa como **python dna.py databases/large.csv sequences/12.txt**. Seu programa deve gerar Lily.
- Execute seu programa como **python dna.py databases/large.csv sequences/13.txt**. Seu programa deve exibir No match.
- Execute seu programa como **python dna.py databases/large.csv sequences/14.txt**. Seu programa deve imprimir Severus.
- Execute seu programa como **python dna.py databases/large.csv sequences/15.txt**. Seu programa deve gerar o Sirius.
- Execute seu programa como **python dna.py databases/large.csv sequences/16.txt**. Seu programa deve exibir No match.
- Execute seu programa como **python dna.py databases/large.csv sequences/17.txt**. Seu programa deve gerar Harry.
- Execute seu programa como **python dna.py databases/large.csv sequences/18.txt**. Seu programa deve imprimir No match.
- Execute seu programa como **python dna.py databases/large.csv sequences/19.txt**. Seu programa deve gerar Fred.
- Execute seu programa como **python dna.py databases/large.csv sequences/20.txt**. Seu programa deve exibir No match.

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/dna

Execute o seguinte para avaliar o estilo do seu código usando **style50**.

style50 dna.py

## Lab 6: Copa do Mundo

# LAB 6: Copa Do Mundo

Escreva um programa para fazer simulações da Copa do Mundo FIFA.

```
$ python tournament.py 2018m.csv
Belgium: 20.9% chance of winning
Brazil: 20.3% chance of winning
Portugal: 14.5% chance of winning
Spain: 13.6% chance of winning
Switzerland: 10.5% chance of winning
Argentina: 6.5% chance of winning
England: 3.7% chance of winning
France: 3.3% chance of winning
Denmark: 2.2% chance of winning
Croatia: 2.0% chance of winning
Colombia: 1.8% chance of winning
Sweden: 0.5% chance of winning
Uruguay: 0.1% chance of winning
Mexico: 0.1% chance of winning
```

## Background

Na Copa do Mundo de futebol, a fase de mata-mata é formada por 16 times. A cada rodada, cada equipe joga com outra equipe e as perdedoras são eliminadas. Quando restarem apenas duas equipes, o vencedor da partida final é o campeão.

No futebol, os times recebem [classificações da FIFA](#), que são valores numéricos que representam o nível de habilidade relativa de cada time. As classificações mais altas da FIFA indicam melhores resultados em jogos anteriores e, considerando as classificações da FIFA de duas equipes, é possível estimar a probabilidade de que uma das equipes ganhe um jogo com base em suas classificações atuais. As classificações da FIFA anteriores às duas Copas do Mundo anteriores estão disponíveis como [classificações masculinas da FIFA de maio de 2018](#) e [classificações femininas da FIFA de março de 2019](#).

Usando essas informações, podemos simular todo o torneio simulando rodadas repetidamente até ficarmos com apenas uma equipe. E se quisermos estimar a probabilidade de uma determinada equipe ganhar o torneio, podemos simular o torneio várias vezes (por exemplo, 1000 simulações) e contar quantas vezes cada equipe ganha um torneio simulado.

Sua tarefa neste laboratório é fazer exatamente isso usando Python!

# Começando

- Faça login em [IDE CS50](#) usando sua conta GitHub.
- Na janela do seu terminal, execute <https://cdn.cs50.net/2020/fall/labs/6/lab6.zip> para baixar um arquivo Zip do código de distribuição do laboratório.
- Na janela do seu terminal, execute **unzip lab6.zip** para descompactar esse arquivo Zip.
- Na janela do terminal, execute **cd lab6** para mudar os diretórios para o diretório **lab6**.

## Entendendo o problema

Comece dando uma olhada no arquivo **2018m.csv**. Este arquivo contém as 16 equipes nas eliminatórias da Copa do Mundo Masculina 2018 e as classificações de cada equipe.

Observe que o arquivo CSV tem duas colunas, uma chamada **team** (representando o nome do país da equipe) e outra chamada **rating** (representando a classificação da equipe).

A ordem em que as equipes são listadas determina quais equipes se enfrentarão em cada rodada (na primeira rodada, por exemplo, o Uruguai jogará contra Portugal e a França jogará contra a Argentina; na próxima rodada, o vencedor da partida Uruguai-Portugal vai jogar o vencedor da partida França-Argentina). Portanto, certifique-se de não editar a ordem em que as equipes aparecem neste arquivo!

Em última análise, em Python, podemos representar cada equipe como um dicionário que contém dois valores: o nome da equipe e a classificação. Uruguai, por exemplo, gostaríamos de representar em Python como **{"team": "Uruguay", "rating": 976}**.

A seguir, dê uma olhada em **2019w.csv**, que contém dados formatados da mesma forma para a Copa do Mundo Feminina 2019.

Agora, abra **tournament.py** e veja se já escrevemos algum código para você. A variável **N** no topo representa quantas simulações da Copa do Mundo executar: neste caso, 1000.

A função **simulate\_game** aceita duas equipes como entradas (lembre-se de que cada equipe é um dicionário contendo o nome da equipe e a classificação da equipe) e simula um jogo entre elas. Se a primeira equipe vencer, a função retorna **True**; caso contrário, a função retorna **False**.

A função **simulate\_round** aceita uma lista de times (em uma variável chamada **teams**) como entrada e simula jogos entre cada par de times. A função então retorna uma lista de todas as equipes que ganharam a rodada.

Na função **main**, observe que primeiro garantimos que **len(sys.argv)** (o número de argumentos da linha de comando) é 2. Usaremos argumentos da linha de comando para informar ao Python qual arquivo CSV de equipe usar para executar o torneio simulação. Em

seguida, definimos uma lista chamada `teams` (que eventualmente será uma lista de times) e um dicionário chamado `counts` (que irá associar os nomes dos times ao número de vezes que aquele time ganhou um torneio simulado). No momento, eles estão vazios, então preenchê-los depende de você!

Por fim, no final do `main`, classificamos os times em ordem decrescente de quantas vezes eles ganharam as simulações (de acordo com `counts`) e imprimimos a probabilidade estimada de que cada time ganhe a Copa do Mundo.

Preencher `teams` e `counts` e escrever a função `simulate_tournament` são deixados para você!

## Detalhes de implementação

Complete a implementação de **`tournament.py`**, de forma que simule uma série de torneios e gere a probabilidade de vitória de cada equipe.

Em primeiro lugar, em **`main`**, ler os dados da equipe a partir do arquivo CSV na memória do seu programa, e adicionar cada equipe para a lista de equipes.

- O arquivo a ser usado será fornecido como um argumento de linha de comando. Você pode acessar o nome do arquivo, então, com `sys.argv[1]`.
- Lembre-se de que você pode abrir um arquivo com `open(filename)`, onde `filename` é uma variável que armazena o nome do arquivo.
- Depois de ter um arquivo `f`, você pode usar `csv.DictReader(f)` para dar a você um “leitor”: um objeto em Python que você pode percorrer para ler o arquivo uma linha por vez, tratando cada linha como um dicionário.
- Por padrão, todos os valores lidos do arquivo serão strings. Portanto, certifique-se primeiro de converter `rating` da equipe em um `int` (você pode usar a função `int` em Python para fazer isso).
- Por fim, anexe o dicionário de cada equipe à `teams`. A chamada de função `teams.append(x)` acrescentará `x` à lista `teams`.
- Lembre-se de que cada equipe deve ser um dicionário com um nome em `team` e uma `rating`.

Em seguida, implemente a função `simulate_tournament`. Esta função deve aceitar como entrada uma lista de equipes e simular rodadas repetidamente até que você fique com uma equipe. A função deve retornar o nome dessa equipe.

- Você pode chamar a função `simulate_round`, que simula uma única rodada, aceitando uma lista de equipes como entrada e retornando uma lista de todos os vencedores.
- Lembre-se de que se `x` for uma lista, você pode usar `len(x)` para determinar o comprimento da lista.

- Você não deve presumir o número de equipes no torneio, mas pode presumir que será uma potência de 2.

Por fim, de volta à função `main`, execute `N` simulações de torneios e controle quantas vezes cada equipe vence no dicionário de `counts`.

- Por exemplo, se o Uruguai ganhou 2 torneios e Portugal ganhou 3 torneios, então seu dicionário de contagens deve ser `{"Uruguay": 2, "Portugal": 3}`.
- Você deve usar seu `simulate_tournament` para simular cada torneio e determinar o vencedor.
- Lembre-se de que se `counts` for um dicionário, então a sintaxe como `counts[team_name] = x` associará a chave armazenada em `team_name` com o valor armazenado em `x`.
- Você pode usar o `in` em palavras-chave em Python para verificar se um dicionário tem uma chave especial já. Por exemplo, `if "Portugal" in counts:` irá verificar se "Portugal" já tem um valor existente no dicionário de `counts`.

## Dicas

Ao ler o arquivo, você pode achar esta sintaxe útil, com `filename` como o nome do seu arquivo e `file` como uma variável.

```
with open(filename) as file:
 reader = csv.DictReader(file)
```

Em Python, para acrescentar ao final de uma lista, use a função `.append()`.

## Testando

Seu programa deve se comportar de acordo com os exemplos abaixo. Como as simulações têm aleatoriedade em cada uma, sua saída provavelmente não corresponderá perfeitamente aos exemplos abaixo.

```
$ python tournament.py 2018m.csv
Belgium: 20.9% chance of winning
Brazil: 20.3% chance of winning
Portugal: 14.5% chance of winning
Spain: 13.6% chance of winning
Switzerland: 10.5% chance of winning
Argentina: 6.5% chance of winning
England: 3.7% chance of winning
France: 3.3% chance of winning
Denmark: 2.2% chance of winning
Croatia: 2.0% chance of winning
Colombia: 1.8% chance of winning
Sweden: 0.5% chance of winning
Uruguay: 0.1% chance of winning
Mexico: 0.1% chance of winning
```

```
$ python tournament.py 2019w.csv
Germany: 17.1% chance of winning
United States: 14.8% chance of winning
England: 14.0% chance of winning
France: 9.2% chance of winning
Canada: 8.5% chance of winning
Japan: 7.1% chance of winning
Australia: 6.8% chance of winning
Netherlands: 5.4% chance of winning
Sweden: 3.9% chance of winning
Italy: 3.0% chance of winning
Norway: 2.9% chance of winning
Brazil: 2.9% chance of winning
Spain: 2.2% chance of winning
China PR: 2.1% chance of winning
Nigeria: 0.1% chance of winning
```

- Você deve estar se perguntando o que realmente aconteceu nas Copas do Mundo de 2018 e 2019! Para os homens, a França venceu, derrotando a Croácia na final. A Bélgica derrotou a Inglaterra pelo terceiro lugar. Para as mulheres, os Estados Unidos venceram, derrotando a Holanda na final. A Inglaterra derrotou a Suécia pelo terceiro lugar.

## Como testar seu código

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/copa_mundial
```

Execute o seguinte para avaliar o estilo do seu código usando **style50**.

```
style50 tournament.py
```

## ANOTAÇÕES MÓDULO 7

# Processamento de dados

Na semana passada, coletamos uma pesquisa sobre as preferências de [Casas de Hogwarts](#) e registramos os dados de um arquivo CSV com Python.

Nesta semana, coletaremos mais alguns dados sobre seus programas de TV favoritos e seus gêneros. [Clique aqui para realizar o download do arquivo .CSV abaixo!](#)

Com centenas de respostas, podemos começar a olhar as respostas no Planilhas Google, um aplicativo de planilha baseado na web, mostrando nossos dados em linhas e colunas:



|    | A                   | B                         | C                                                      |
|----|---------------------|---------------------------|--------------------------------------------------------|
|    | Timestamp           | title                     | genres                                                 |
| 2  | 10/19/2020 1:41:38  | Punisher                  | Action, Adventure, Comedy, Crime, Romance, Sci-Fi, War |
| 3  | 10/19/2020 13:34:57 | The Office                | Comedy                                                 |
| 4  | 10/19/2020 13:35:01 | Breaking Bad              | Crime, Drama, Thriller                                 |
| 5  | 10/19/2020 13:35:01 | new girl                  | Comedy, Romance                                        |
| 6  | 10/19/2020 13:35:02 | archer                    | Action, Adventure, Animation, Comedy                   |
| 7  | 10/19/2020 13:35:03 | The Office                | Comedy                                                 |
| 8  | 10/19/2020 13:35:03 | Brooklyn99                | Comedy                                                 |
| 9  | 10/19/2020 13:35:03 | Fleabag                   | Comedy, Drama                                          |
| 10 | 10/19/2020 13:35:03 | community                 | Comedy                                                 |
| 11 | 10/19/2020 13:35:03 | Suits                     | Drama                                                  |
| 12 | 10/19/2020 13:35:04 | Gilmore Girls             | Comedy, Family                                         |
| 13 | 10/19/2020 13:35:04 | Parks and Recreation      | Comedy                                                 |
| 14 | 10/19/2020 13:35:04 | Umbrella Academy          | Comedy, Drama, Mystery, Sci-Fi                         |
| 15 | 10/19/2020 13:35:04 | avatar the last airbender | Action, Adventure, Animation                           |
| 16 | 10/19/2020 13:35:06 | New Girl                  | Comedy                                                 |
| 17 | 10/19/2020 13:35:06 | the office                | Comedy                                                 |
| 18 | 10/19/2020 13:35:06 | The Office                | Comedy, Documentary, Reality-TV                        |
| 19 | 10/19/2020 13:35:08 | Game of Thrones           | Drama                                                  |
| 20 | 10/19/2020 13:35:08 | Breaking Bad              | Action, Adventure, Crime                               |
| 21 | 10/19/2020 13:35:08 | Sherlock                  | Action, Crime, Mystery, Thriller                       |
| 22 | 10/19/2020 13:35:08 | Breaking Bad              | Crime, Drama, Thriller                                 |
| 23 | 10/19/2020 13:35:08 | Dragon Ball Z             | Action, Adventure, Animation                           |
| 24 | 10/19/2020 13:35:09 | Tiny Words                | Documentary                                            |

- Algumas respostas mostram um único gênero selecionado, como “Comédia”, enquanto outras, com vários gêneros, os mostram em uma célula ainda mas separados por uma vírgula, como “Crime, Drama”.

Com um aplicativo de planilhas como o Google Sheets, Apple's Numbers, Microsoft Excel ou outros, podemos:

- classificar nossos dados
- armazenar dados em linhas e colunas, onde cada entrada adicional é uma linha e as propriedades de cada entrada, como título ou gênero, é uma coluna
- decidir sobre o **esquema (schema)**, ou formato, de nossos dados com antecedência, escolhendo as colunas

Um **banco de dados (database)** é um arquivo ou programa que armazena dados para nós.

Um arquivo CSV é um **banco de dados de arquivo simples**, em que os dados de cada coluna são separados por vírgulas e cada linha está em uma nova linha, salva simplesmente como um arquivo.

- Se alguns dados em um CSV contiverem uma vírgula, geralmente estarão entre aspas como uma string para evitar confusão.
- Fórmulas e cálculos em programas de planilhas são integrados aos próprios programas; um arquivo CSV só pode armazenar valores brutos e estáticos.

Vamos baixar um arquivo CSV com os dados da planilha com “Arquivo> Download”, carregá-lo em nosso IDE arrastando e soltando-o em nossa árvore de arquivos e ver se é realmente um arquivo de texto com valores separados por vírgula correspondendo ao dados da planilha.

# Limpeza

Começaremos escrevendo `favorites.py`, escolhendo Python em vez de C como nossa ferramenta de escolha para suas bibliotecas e abstração:

```
import csv

with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.reader(file)
 next(reader)
 for row in reader:
 print(row[1])
```

- Vamos abrir o arquivo e ter certeza de que podemos imprimir o título de cada linha, usando a palavra - chave **with** em Python, que fechará nosso arquivo para nós depois de deixarmos seu escopo, com base no recuo.
- **open** usa o modo de leitura por padrão, mas para ser claro em nosso código, adicionaremos **r** explicitamente.
- A biblioteca **csv** tem uma função de **reader** que criará uma variável de **reader** que podemos usar.
- Chamaremos **next** para pular a primeira linha, já que é a linha do cabeçalho, e então usaremos um loop para imprimir a segunda coluna em cada linha, que é o título.

Para melhorar isso, usaremos um DictReader, leitor de dicionário, que cria um dicionário a partir de cada linha, permitindo acessar cada coluna pelo seu nome. Também não precisamos pular a linha do cabeçalho neste caso, pois o DictReader a usará automaticamente.

```
import csv

with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.DictReader(file)

 for row in reader:
 print(row["title"])
```

- Como a primeira linha em nosso CSV tem os nomes das colunas, ela também pode ser usada para rotular cada coluna em nossos dados.

Agora, vamos dar uma olhada em todos os títulos exclusivos em nossas respostas:

```
import csv

titles = set()

with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.DictReader(file)

 for row in reader:
```

```
titles.add(row["title"])
```

```
for title in titles:
 print(title)
```

- Vamos criar um conjunto chamado **titles** e adicionar o valor do título de cada linha a ele. Chamar **add** em um conjunto verificará automaticamente se há duplicatas e garantirá que haja apenas valores exclusivos.
- Então, podemos iterar sobre os elementos no conjunto com um loop **for** , imprimindo cada um deles.

Para classificar os títulos, podemos apenas alterar nosso loop **for title in sorted(titles)**, que classificará nosso conjunto antes de iterarmos sobre ele.

Veremos que nossos títulos são considerados diferentes se a capitalização ou a pontuação forem diferentes, portanto, limparemos a capitalização adicionando o título todo em maiúsculas com **titles.add(row["title"].upper())**

Também teremos que remover os espaços antes ou depois, para que possamos adicionar **titles.add(row["title"].strip().upper())** que remove os espaços em branco do título e depois os converte em maiúsculas.

Agora, **padronizamos** nossos dados e nossa lista de títulos está muito mais limpa.

## Contagem

Podemos usar um dicionário, em vez de um conjunto (set), para contar o número de vezes que vimos cada título, com as chaves sendo os títulos e os valores sendo um número inteiro contando o número de vezes que vemos cada um deles:

```
import csv
```

```
titles = {}
```

```
with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.DictReader(file)
```

```
 for row in reader:
 title = row["title"].strip().upper()
 if title not in titles:
 titles[title] = 0
 titles[title] += 1
```

```
for title in sorted(titles):
 print(title, titles[title])
```

- Aqui, primeiro verificamos se não vimos o título antes (se estiver em **not in titles** ). Definimos o valor inicial como 0 se for esse o caso e, então, podemos incrementar com segurança o valor em 1 todas as vezes.

- Finalmente, podemos imprimir as chaves e valores de nosso dicionário, passando-os como argumentos para **print**, que os separará com um espaço para nós.

Podemos classificar pelos valores no dicionário mudando nosso loop para:

```
...
def f(title):
 return titles[title]

for title in sorted(titles, key=f, reverse=True):
 ...
```

- Definimos uma função, **f**, que apenas retorna a contagem de um título no dicionário com **titles[title]**. A função **sorted**, por sua vez, usará essa função como **chave** para ordenar os elementos do dicionário. E também passaremos em **reverse=True** para classificar do maior para o menor, em vez do menor para o maior.

Portanto, agora veremos os programas mais populares impressos.

Na verdade, podemos definir nossa função na mesma linha, com esta sintaxe:

```
for title in sorted(titles, key=lambda title: titles[title],
reverse=True):
```

- Passamos um **lambda**, ou função anônima, que não tem nome, mas recebe algum argumento ou argumentos e retorna um valor imediatamente.

## Busca

Podemos escrever um programa para pesquisar um título e relatar sua popularidade:

```
import csv

title = input("Title: ").strip().upper()

with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.DictReader(file)

 counter = 0
 for row in reader:
 if row["title"].strip().upper() == title:
 counter += 1

print(counter)
```

- Solicitamos o input do usuário e, em seguida, abrimos nosso arquivo CSV. Como estamos procurando apenas um título, podemos ter uma variável de **counter** que incrementamos.

- Verificamos as correspondências depois de padronizar a entrada e os dados à medida que verificamos cada linha.

O tempo de execução disso é  $O(n)$ , uma vez que precisamos olhar para cada linha.

## Bancos de dados relacionais

**Bancos de dados relacionais** são programas que armazenam dados, basicamente em arquivos, mas com estruturas de dados adicionais que nos permitem pesquisar e armazenar dados com mais eficiência.

Com outra linguagem de programação, **SQL** (pronuncia-se como “sequel”), Structured Query Language, podemos interagir com muitos bancos de dados relacionais e suas **tabelas**, como planilhas, que armazenam dados.

Usaremos um programa de banco de dados comum chamado **SQLite**, um dos muitos programas disponíveis que oferecem suporte a SQL. Outros programas de banco de dados incluem Oracle Database, MySQL, PostgreSQL e Microsoft Access.

O SQLite armazena nossos dados em um arquivo binário, com 0s e 1s que representam os dados de forma eficiente. Vamos interagir com nossas tabelas de dados por meio de um programa de linha de comando, **sqlite3**.

Vamos executar alguns comandos no IDE CS50 para importar nosso arquivo CSV para uma tabela chamada “shows”:

```
~/ $ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .mode csv
sqlite> .import 'Favorite TV Shows (Responses) - Form Responses
1.csv' shows
```

- Com base nas linhas do arquivo CSV, o SQLite criará uma tabela em nosso banco de dados com os dados e colunas.
- Vamos definir o SQLite para o modo CSV e usar o comando **.import** para criar uma tabela a partir de nosso arquivo.

Acontece que, ao trabalhar com dados, geralmente precisamos de quatro tipos de operações suportadas por bancos de dados relacionais:

- **CREATE**
- **READ**
- **UPDATE**
- **DELETE**

## SQL

No SQL, os comandos para realizar cada uma dessas operações são:

- **CREATE, INSERT**
  - Por exemplo, para criar uma nova tabela, podemos usar: **CREATE TABLE table (column type, ...);** onde **table** é o nome de nossa nova tabela e **column** é o nome de uma coluna, seguido por seu tipo.
- **SELECT**
  - **SELECT column FROM table;**
- **UPDATE**
- **DELETE**

Podemos verificar o esquema de nossa nova tabela com **.schema**:

```
• sqlite> .schema
CREATE TABLE shows(
 "Timestamp" TEXT,
 "title" TEXT,
 "genres" TEXT
);
```

- Vemos que **.import** usou o comando **CREATE TABLE ...** listado para criar uma tabela chamada **shows**, com os nomes das colunas copiados automaticamente da linha do cabeçalho do CSV e os tipos interpretados como texto.

Podemos selecionar uma coluna com:

```
sqlite> SELECT title FROM shows;
title
...
"Madam Secretary"
"Game of Thrones"
"Still Game"
```

- Observe que capitalizamos as palavras-chave SQL por convenção e veremos os títulos de nossas linhas impressos na ordem do CSV.
- Também podemos selecionar várias colunas com **SELECT Timestamp, title FROM shows;** (O Timestamp estava em maiúscula no CSV), ou todas as colunas com **SELECT \* FROM shows;**

O SQL oferece suporte a muitas funções que podemos usar para contar e resumir dados:

- **AVG**
- **COUNT**
- **DISTINCT**, para obter valores distintos sem duplicatas
- **LOWER**
- **MAX**
- **MIN**
- **UPPER**
- ...

Podemos limpar nossos títulos como antes, convertendo-os em maiúsculas e imprimindo apenas os valores exclusivos:

```
sqlite> SELECT DISTINCT(UPPER(title)) FROM shows;
title
...
"GREY'S ANATOMY"
"SCOOBY DOO"
"MADAM SECRETARY"
```

Também podemos adicionar mais **cláusulas** ou frases que modificam nossa consulta:

- **WHERE**, resultados correspondentes em uma condição estrita
- **LIKE**, resultados correspondentes em uma condição menos estrita
- **ORDER BY**, resultados de ordenação de alguma forma
- **LIMIT**, limitando o número de resultados
- **GROUP BY**, agrupando resultados de alguma forma
- ...

Vamos filtrar as linhas por títulos:

```
sqlite> SELECT title FROM shows WHERE title = "The Office";
title
...
"The Office"
"The Office"
"The Office"
```

Mas existem outras entradas que gostaríamos de capturar, então podemos usar:

```
sqlite> SELECT title FROM shows WHERE title LIKE "%Office%";
title
...
office
"The Office"
"the office "
"The Office"
```

- O caractere % é um espaço reservado para zero ou mais outros caracteres.

Podemos ordenar nossos títulos:

```
sqlite> SELECT DISTINCT(UPPER(title)) FROM shows ORDER BY
UPPER(title);
...
X-FILES
"ZETA GUNDAM"
"ZONDAG MET LUBACH"
```

Podemos até agrupar os mesmos títulos e contar quantas vezes eles aparecem:

```
sqlite> SELECT UPPER(title), COUNT(title) FROM shows GROUP BY
UPPER(title);
```

```
...
```

```
"THE OFFICE",23
```

```
...
```

```
"TOP GEAR",1
```

```
...
```

```
"TWIN PEAKS",4
```

```
...
```

Podemos ordenar pela contagem:

```
sqlite> SELECT UPPER(title), COUNT(title) FROM shows GROUP BY
UPPER(title) ORDER BY COUNT(title);
```

```
...
```

```
"THE OFFICE",23
```

```
FRIENDS,26
```

```
"GAME OF THRONES",33
```

- E se adicionarmos **DESC** ao final, poderemos ver os resultados em ordem decrescente.

Com **LIMIT 10** adicionado também, vemos as 10 linhas principais:

```
sqlite> SELECT UPPER(title), COUNT(title) FROM shows GROUP BY
UPPER(title) ORDER BY COUNT(title) DESC LIMIT 10;
```

```
UPPER(title),COUNT(title)
```

```
"GAME OF THRONES",33
```

```
FRIENDS,26
```

```
"THE OFFICE",23
```

```
...
```

Por fim, também cortaremos os espaços em branco de cada título, aninhando essa função:

```
sqlite> SELECT UPPER(TRIM(title)), COUNT(title) FROM shows GROUP BY
UPPER(TRIM(title)) ORDER BY COUNT(title) DESC LIMIT 10;
```

```
UPPER(title),COUNT(title)
```

```
"GAME OF THRONES",33
```

```
FRIENDS,26
```

```
"THE OFFICE",23
```

```
...
```

Antes de terminar, queremos salvar nossos dados em um arquivo com `.save shows.db`, que veremos em nosso IDE depois de executar esse comando.

Observe que nosso programa para encontrar os programas mais populares de antes, que ocupava dezenas de linhas de código em Python, agora requer apenas uma (longa) linha de SQL.



Usamos a interface de linha de comando do SQLite, mas também existem programas gráficos que suportam o trabalho com consultas SQL e a visualização dos resultados de forma mais visual.

## Tabelas

Nossa coluna de **genres** tem vários gêneros no mesmo campo, então usaremos **LIKE** para obter todos os títulos que contêm algum gênero:

```
sqlite> SELECT title FROM shows WHERE genres LIKE "%Comedy%";
...
```

- Mas os gêneros ainda são armazenados como uma lista separada por vírgulas, o que não é tão claro. Por exemplo, se nossos gêneros incluíssem “Music” e “Musical”, seria difícil selecionar títulos apenas com o gênero “Music”.

Podemos inserir dados em uma tabela manualmente com **INSERT INTO table (column, ...) VALUES(value, ...);** .

- Por exemplo, podemos dizer:

```
sqlite> INSERT INTO shows (Timestamp, title, genres) VALUES("now",
"The Muppet Show", "Comedy, Musical");
```

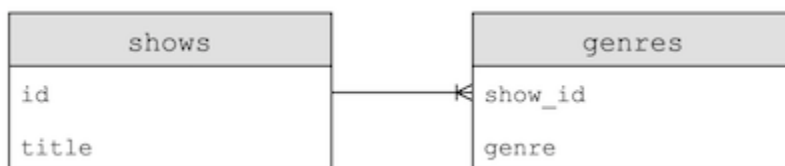
Podemos atualizar uma linha com a tabela **UPDATE table SET column = value WHERE condition;** , por exemplo:

```
sqlite> INSERT INTO shows (Timestamp, title, genres) VALUES("now",
"The Muppet Show", "Comedy, Musical");
```

Podemos até remover todas as linhas que combinam com: **DELETE FROM table WHERE condition;**

```
sqlite> DELETE FROM shows WHERE title LIKE "Friends";
```

Agora vamos escrever nosso próprio programa Python que usará SQL para importar nossos dados CSV para tabelas com o seguinte design:



- Este design vai começar a **normalizar** nossos dados, ou reduzir a redundância e garantir uma única fonte de verdade.
- Aqui, por exemplo, temos uma tabela chamada **shows** , com cada show tendo um **id** e **title**, e outra tabela, **genre**, que usa o **id** de cada show para associar um gênero a um show. Observe que o **title** do programa não precisa ser armazenado várias vezes.
- Agora também podemos adicionar várias linhas na tabela de **genre** , para associar um show a mais de um gênero.

Acontece que o SQL também tem seus próprios tipos de dados para otimizar a quantidade de espaço usado para armazenar dados, que precisaremos especificar ao criar uma tabela manualmente:

- BLOB, para "Objeto binário grande", dados binários brutos que podem representar arquivos
- INTEGER
- NUMERIC, como um número, mas não exatamente um número, como uma data ou hora
- REAL, para valores de ponto flutuante
- TEXT, como strings

As colunas também podem ter atributos adicionais:

- NOT NULL, que especifica que deve haver algum valor
- UNIQUE, o que significa que o valor dessa coluna deve ser único para cada linha da tabela
- PRIMARY KEY, como a coluna id acima, que será usada para identificar de forma única cada linha
- FOREIGN KEY, como a coluna show\_id acima que se refere a uma coluna em alguma outra tabela

Usaremos o recurso SQL da biblioteca CS50 para fazer consultas facilmente e também existem outras bibliotecas para Python:

```
import csv
```

```
from cs50 import SQL
```

```
open("shows.db", "w").close()
db = SQL("sqlite:///shows.db")
```

```
db.execute("CREATE TABLE shows (id INTEGER, title TEXT, PRIMARY KEY(id))")
db.execute("CREATE TABLE genres (show_id INTEGER, genre TEXT, FOREIGN KEY(show_id) REFERENCES shows(id))")
```

```
with open("Favorite TV Shows - Form Responses 1.csv", "r") as file:
 reader = csv.DictReader(file)
```

```
 for row in reader:
```

```
 title = row["title"].strip().upper()
```

```
 id = db.execute("INSERT INTO shows (title) VALUES(?)", title)
```

```
 for genre in row["genres"].split(", "):
```

```
 db.execute("INSERT INTO genres (show_id, genre) VALUES(?, ?)", id, genre)
```

- Primeiro, vamos abrir um arquivo shows.db e fechá-lo, para ter certeza de que o arquivo foi criado.

- Em seguida, criaremos uma variável **db** para armazenar nosso banco de dados criado pelo **SQL**, que pega o arquivo de banco de dados que acabamos de criar.
- A seguir, executaremos comandos SQL escrevendo-o como uma string e chamando **db.execute** com ele. Aqui, criaremos duas tabelas conforme projetamos acima, indicando os nomes, tipos e propriedades de cada coluna que queremos em cada tabela.
- Agora, podemos ler nosso arquivo CSV original linha por linha, obtendo o título e usando **db.execute** para executar um comando **INSERT** para cada linha. Acontece que podemos usar o **?** espaço reservado em um comando SQL e passar uma variável a ser substituída. Então, vamos obter de volta um **id** que é criado automaticamente para nós para cada linha, uma vez que declaramos que é uma chave primária.
- Por fim, dividiremos a string de gênero em cada linha pela vírgula e inseriremos cada um deles na tabela de **genres**, usando o **id** para o show como **show\_id**.

Depois de executar este programa, podemos ver os IDs e títulos de cada programa, bem como os gêneros onde a primeira coluna é o ID de um programa:

```
sqlite> SELECT * FROM shows;
...
511 | MADAM SECRETARY
512 | GAME OF THRONES
513 | STILL GAME
sqlite> SELECT * FROM genres;
...
511 | Drama
512 | Action
512 | Adventure
512 | History
512 | Thriller
512 | War
513 | Comedy
```

- Observe que o programa com id 512, “GAME OF THRONES”, agora possui cinco gêneros associados a ele.

Para encontrar todos os musicais, por exemplo, podemos executar:

```
sqlite> SELECT show_id FROM genres WHERE genre = "Musical";
...
422
435
468
```

E podemos aninhar essa consulta para obter títulos da lista de IDs de programa:

```
sqlite> SELECT title FROM shows WHERE id IN (SELECT show_id FROM
genres WHERE genre = "Musical");
title
BREAKING BAD
```

...

THE LAWYER

MY BROTHER, MY BROTHER, AND ME

- Nossa primeira consulta, entre parênteses, será executada primeiro e, em seguida, usada na consulta externa.

Podemos encontrar todas as linhas em programas(**shows**) com o título de “THE OFFICE”, e encontrar todos os gêneros associados:

```
sqlite> SELECT DISTINCT(genre) FROM genres WHERE show_id IN (SELECT
id FROM shows WHERE title = "THE OFFICE") ORDER BY genre;
```

genre

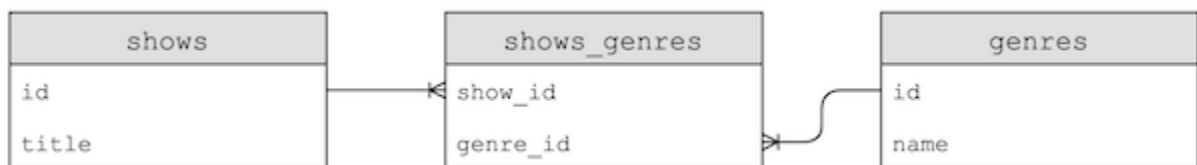
...

Comedy

Documentary

...

Poderíamos melhorar ainda mais o design de nossas tabelas, com uma terceira tabela:



- Agora, o nome de cada gênero será armazenado apenas uma vez, com uma nova tabela, uma tabela de junção(**JOIN TABLE**), chamada **shows\_genres**, que contém chaves estrangeiras que vinculam shows aos gêneros. Essa é uma relação de **muitos para muitos**, em que um programa pode ter vários gêneros e um gênero pode pertencer a muitos programas.
- Se precisássemos mudar o nome de um gênero, teríamos apenas que mudar uma linha agora, em vez de muitas.

Acontece que também existem subtipos para uma coluna, que podemos ser ainda mais específicos com:

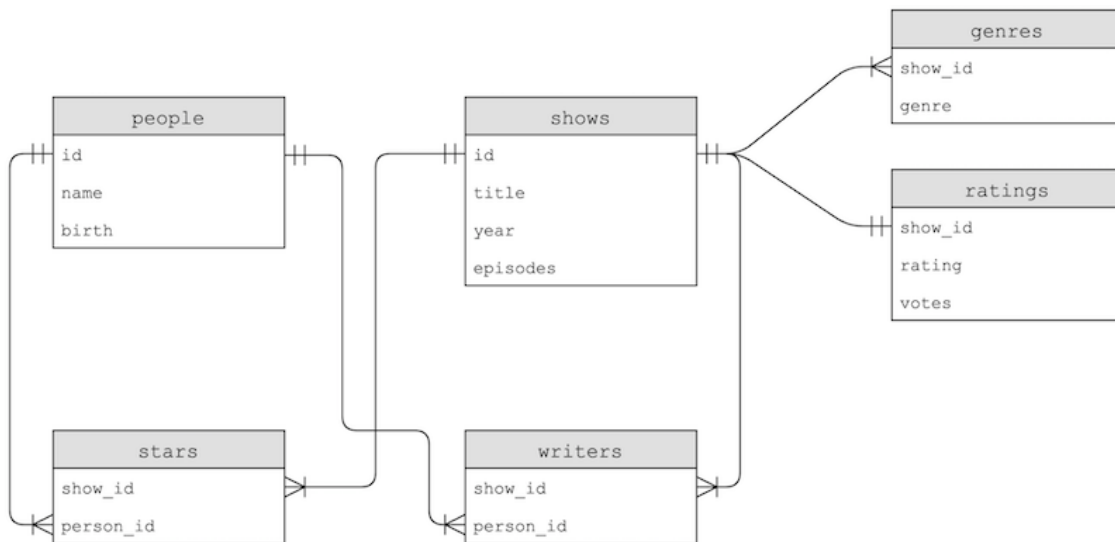
- **INTEGER**
  - smallint , com menos bits
  - integer
  - bigint , com mais bits
- **NUMERIC**
  - boolean
  - date
  - datetime
  - numeric(scale, precision) , com um número fixo de dígitos
  - time
  - timestamp
- **REAL**
  - real
  - double precision , com o dobro de bits

- **TEXTO**
  - char(n) , um número fixo de caracteres
  - varchar(n) , um número variável de caracteres, até algum limite n
  - text, uma string sem limite

## IMDb

IMDb, ou Internet Movie Database, tem conjuntos de dados disponíveis para download como arquivos TSV (valores separados por tabulação).

Depois de importar um desses conjuntos de dados, veremos tabelas com o seguinte esquema:



- A tabela de **genres** tem alguma duplicação, já que a coluna de **genre** é repetida, mas a tabela de **stars** e **writers** junta linhas nas tabelas **people** e **shows** com base em seu relacionamento.

Com **SELECT COUNT (\*) FROM shows;** podemos ver que há mais de 150.000 shows em nossa tabela, portanto, com uma grande quantidade de dados, podemos usar **índices(indexes)**, o que diz ao nosso programa de banco de dados para criar estruturas de dados adicionais para que possamos pesquisar e classificar com tempo logarítmico:

```
sqlite> CREATE INDEX title_index ON shows (title);
```

- Acontece que essas estruturas de dados são geralmente **árvores B**, como as árvores binárias que vimos em C, com nós organizados de forma que possamos pesquisar mais rápido do que linearmente.
- A criação de um índice leva algum tempo, mas depois podemos executar consultas muito mais rapidamente.

Com nossos dados espalhados entre diferentes tabelas, podemos usar comandos **JOIN** para combiná-los em nossas consultas:

```
sqlite3> SELECT title FROM people
...> JOIN stars ON people.id = stars.person_id
...> JOIN shows ON stars.show_id = shows.id
```

```
...> WHERE name = "Steve Carell";
...
The Morning Show
LA Times: the Envelope
```

- Com a sintaxe **JOIN**, podemos combinar tabelas virtualmente com base em suas chaves estrangeiras e usar suas colunas como se fossem uma única tabela.

Depois de criar mais alguns índices, nosso comando **JOIN** também é executado com muito mais rapidez.

## Problemas

Um problema em SQL é chamado de **ataque de injeção de SQL (SQL injection attack)**, em que alguém pode injetar ou colocar seus próprios comandos em entradas que executamos em nosso banco de dados.

Nossa consulta para fazer o login de um usuário pode ser `rows = db.execute("SELECT * FROM users WHERE username =? AND password =?", Username, password)`. Usando o `?` espaços reservados, nossa biblioteca SQL escapará do input ou evitará que caracteres perigosos sejam interpretados como parte do comando.

Em contraste, podemos ter uma consulta SQL que é uma string formatada, como:

```
f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
```

Se um usuário digitar [malan@harvard.edu](mailto:malan@harvard.edu)-- , a consulta será:

```
f "SELECT * FROM users WHERE username = 'malan@harvard.edu' --' AND password = '{ password }'"
```

Esta consulta irá realmente selecionar a linha onde username = '[malan@harvard.edu](mailto:malan@harvard.edu)', sem verificar a senha, uma vez que -- transforma o resto da linha em um comentário no SQL.

Outro problema com bancos de dados são as condições de corrida (race conditions), em que o código em um ambiente multithread pode ser combinado, ou misturado, em cada thread.

Um exemplo é um post popular que recebe muitos likes. Um servidor pode tentar incrementar o número de curtidas, solicitando ao banco de dados o número atual de curtidas, adicionando um e atualizando o valor no banco de dados:

```
rows = db.execute("SELECT likes FROM posts WHERE id = ?", id);
likes = rows[0]["likes"]
db.execute("UPDATE posts SET likes = ? WHERE id = ?", likes + 1,
id);
```

- Mas, para aplicativos com vários servidores, cada um deles pode tentar adicionar curtidas ao mesmo tempo. Dois servidores, respondendo a dois usuários diferentes, podem obter o mesmo número inicial de curtidas, já que a primeira linha do código é

executada ao mesmo tempo em cada servidor. Em seguida, ambos definirão o mesmo novo número de curtidas, embora deva haver dois incrementos separados.

Para resolver esse problema, o SQL oferece suporte a transações, nas quais podemos bloquear linhas em um banco de dados, de forma que um determinado conjunto de ações aconteçam juntas, com sintaxe como:

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK

Por exemplo, podemos resolver nosso problema acima com:

```
db.execute("BEGIN TRANSACTION")
rows = db.execute("SELECT likes FROM posts WHERE id = ?", id);
likes = rows[0]["likes"]
db.execute("UPDATE posts SET likes = ? WHERE id = ?", likes + 1,
id);
db.execute("COMMIT")
```

- O banco de dados garantirá que todas as consultas intermediárias sejam executadas juntas.

Outro exemplo pode ser de dois colegas de quarto e uma geladeira compartilhada em seu dormitório. O primeiro colega de quarto chega em casa e vê que não há leite na geladeira. Assim, o primeiro colega de quarto sai na loja para comprar leite e, enquanto eles estão na loja, o segundo colega de quarto chega em casa, vê que não há leite e sai para outra loja para pegar leite também. Mais tarde, haverá duas jarras de leite na geladeira.

Podemos resolver esse problema trancando a geladeira para que nosso colega de quarto não verifique se há leite antes de voltarmos.

## EXERCÍCIOS MÓDULO 7

### Exercício 1: Movies

Escreva consultas SQL para responder a perguntas sobre um banco de dados de filmes.

## Começando

Veja como baixar esse problema em seu próprio IDE CS50. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute `cd ~` (ou simplesmente `cd` sem argumentos) para garantir que você está em seu diretório pessoal.
- Execute `mkdir pset7` para fazer (ou seja, criar) um diretório chamado **pset7**.
- Execute `cd pset7` para mudar para (ou seja, abrir) esse diretório.
- Execute <https://cdn.cs50.net/2020/fall/psets/7/movies/movies.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- Execute `unzip movies.zip` para descompactar esse arquivo.
- Execute `rm movies.zip` seguido por **yes** ou **y** para excluir o arquivo ZIP.

- Execute `ls`. Você deverá ver um diretório chamado **movies** , que estava dentro desse arquivo ZIP.
- Execute `cd movies` para mudar para esse diretório.
- Execute `ls`. Você deve ver um arquivo **movies.db** e também alguns arquivos **.sql** vazios .

## Entendendo o problema

É fornecido a você um arquivo chamado `movies.db` , um banco de dados SQLite que armazena dados do [IMDb](#) sobre filmes, as pessoas que os dirigiram e estrelaram e suas classificações. Em uma janela de terminal, execute **sqlite3 movies.db** para que você possa começar a executar consultas no banco de dados.

Primeiro, quando **sqlite3** solicitar que você forneça uma consulta, digite `.schema` e pressione Enter. Isso produzirá as instruções `CREATE TABLE` que foram usadas para gerar cada uma das tabelas no banco de dados. Ao examinar essas declarações, você pode identificar as colunas presentes em cada tabela.

Observe que a tabela de **movies** possui uma coluna de `id` que identifica exclusivamente cada filme, bem como colunas para o **title** de um filme e o `year` em que o filme foi lançado. A tabela de **people** também tem uma coluna de `id` e também colunas para o **name** de cada pessoa e ano de nascimento (**birth**).

As classificações dos filmes, por sua vez, são armazenadas na tabela de **ratings**. A primeira coluna da tabela é **movie\_id**: uma chave estrangeira que faz referência ao `id` da tabela `movies` . O resto da linha contém dados sobre a `ratings` de cada filme e o número de votos que o filme recebeu no IMDb.

Por fim, as tabelas de `stars` e `directors` relacionam as pessoas aos filmes em que atuaram ou dirigiram. ([Apenas](#) estrelas e diretores principais estão incluídos.) Cada tabela tem apenas duas colunas: `movie_id` e `person_id` , que fazem referência a um filme e pessoa específicos, respectivamente.

O desafio à sua frente é escrever consultas SQL para responder a uma variedade de perguntas diferentes, selecionando dados de uma ou mais dessas tabelas.

## Especificação

Para cada um dos problemas a seguir, você deve escrever uma única consulta SQL que produza os resultados especificados por cada problema. Sua resposta deve ter a forma de uma única consulta SQL, embora você possa aninhar outras consultas dentro de sua consulta.

Você **não deve** presumir nada sobre os `ids` de nenhum filme ou pessoa em particular: suas perguntas devem ser precisas, mesmo que o `id` de qualquer filme ou pessoa em particular seja diferente. Por fim, cada consulta deve retornar apenas os dados necessários para responder à pergunta: se o problema pede apenas que você envie os nomes dos filmes, por exemplo, sua consulta não deve também gerar o ano de lançamento de cada filme.

Fique a vontade para verificar os resultados das suas consultas com a própria IMDb , mas observe que as classificações no site podem ser diferentes daquelas em **movies.db**, pois mais votos podem ter sido incluídos desde que baixamos os dados!



- Em **1.sql**, escreva uma consulta SQL para listar os títulos de todos os filmes lançados em 2008.
  - Sua consulta deve gerar uma tabela com uma única coluna para o título de cada filme.
- Em **2.sql**, escreva uma consulta SQL para determinar o ano de nascimento de Emma Stone.
  - Sua consulta deve gerar uma tabela com uma única coluna e uma única linha (sem incluir o cabeçalho) contendo o ano de nascimento de Emma Stone.
  - Você pode presumir que há apenas uma pessoa no banco de dados com o nome Emma Stone.
- Em **3.sql**, escreva uma consulta SQL para listar os títulos de todos os filmes com data de lançamento igual ou posterior a 2018, em ordem alfabética.
  - Sua consulta deve gerar uma tabela com uma única coluna para o título de cada filme.
  - Os filmes lançados em 2018 devem ser incluídos, assim como os filmes com datas de lançamento no futuro.
- Em **4.sql**, escreva uma consulta SQL para determinar o número de filmes com uma classificação IMDb de 10,0.
  - Sua consulta deve gerar uma tabela com uma única coluna e uma única linha (sem incluir o cabeçalho) contendo o número de filmes com uma classificação de 10,0.
- Em **5.sql**, escreva uma consulta SQL para listar os títulos e anos de lançamento de todos os filmes de Harry Potter, em ordem cronológica.
  - Sua consulta deve gerar uma tabela com duas colunas, uma para o título de cada filme e outra para o ano de lançamento de cada filme.
  - Você pode presumir que o título de todos os filmes de Harry Potter começará com as palavras “Harry Potter” e que se o título de um filme começar com as palavras “Harry Potter”, é um filme de Harry Potter.
- Em **6.sql**, escreva uma consulta SQL para determinar a avaliação média de todos os filmes lançados em 2012.
  - Sua consulta deve gerar uma tabela com uma única coluna e uma única linha (sem incluir o cabeçalho) contendo a classificação média.
- Em **7.sql**, escreva uma consulta SQL para listar todos os filmes lançados em 2010 e suas classificações, em ordem decrescente por classificação. Para filmes com a mesma classificação, ordene-os em ordem alfabética por título.
  - Sua consulta deve gerar uma tabela com duas colunas, uma para o título de cada filme e outra para a classificação de cada filme.
  - Filmes sem classificação não devem ser incluídos no resultado.
- Em **8.sql**, escreva uma consulta SQL para listar os nomes de todas as pessoas que estrelaram Toy Story.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada pessoa.
  - Você pode presumir que há apenas um filme no banco de dados com o título Toy Story.
- Em **9.sql**, escreva uma consulta SQL para listar os nomes de todas as pessoas que estrelaram um filme lançado em 2004, ordenado por ano de nascimento.

- Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada pessoa.
- Pessoas com o mesmo ano de nascimento podem ser listadas em qualquer ordem.
- Não precisa se preocupar com pessoas que não têm ano de nascimento listado, desde que aqueles que têm ano de nascimento estejam listados em ordem.
- Se uma pessoa apareceu em mais de um filme em 2004, ela só deve aparecer uma vez nos resultados.
- Em **10.sql**, escreva uma consulta SQL para listar os nomes de todas as pessoas que dirigiram um filme que recebeu uma classificação de pelo menos 9,0.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada pessoa.
  - Se uma pessoa dirigiu mais de um filme que recebeu uma classificação de pelo menos 9,0, eles só devem aparecer em seus resultados uma vez.
- Em **11.sql**, escreva uma consulta SQL para listar os títulos dos cinco filmes com melhor classificação (em ordem) que Chadwick Boseman estrelou, começando com os de maior classificação.
  - Sua consulta deve gerar uma tabela com uma única coluna para o título de cada filme.
  - Você pode presumir que há apenas uma pessoa no banco de dados com o nome Chadwick Boseman.
- Em **12.sql**, escreva uma consulta SQL para listar os títulos de todos os filmes em que Johnny Depp e Helena Bonham Carter estrelaram juntos.
  - Sua consulta deve gerar uma tabela com uma única coluna para o título de cada filme.
  - Você pode presumir que há apenas uma pessoa no banco de dados com o nome Johnny Depp.
  - Você pode presumir que há apenas uma pessoa no banco de dados com o nome Helena Bonham Carter.
- Em **13.sql**, escreva uma consulta SQL para listar os nomes de todas as pessoas que estrelaram um filme no qual Kevin Bacon também estrelou.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada pessoa.
  - Pode haver várias pessoas chamadas Kevin Bacon no banco de dados. Certifique-se de selecionar apenas Kevin Bacon nascido em 1958.
  - O próprio Kevin Bacon não deve ser incluído na lista resultante.

## Uso

Para testar suas consultas no CS50 IDE, você pode consultar o banco de dados executando

```
$ cat filename.sql | sqlite3 movies.db
```

onde **filename.sql** é o arquivo que contém sua consulta SQL.

Você também pode correr

```
$ cat filename.sql | sqlite3 movies.db > output.txt
```

para redirecionar o resultado da consulta para um arquivo de texto denominado **output.txt**. (Isso pode ser útil para verificar quantas linhas são retornadas por consulta!)

## Dicas

Consulte esta referência de [palavras-chave SQL](#) para alguma sintaxe SQL que pode ser útil!

## Testando

Embora **check50** esteja disponível para este problema, você é encorajado a testar seu código por conta própria para cada um dos itens a seguir. Você pode executar `sqlite3 movies.db` para executar consultas adicionais no banco de dados para garantir que seu resultado esteja correto.

Se você estiver usando o banco de dados `movies.db` fornecido na distribuição deste conjunto de problemas, você deve descobrir que

- A execução de **1.sql** resulta em uma tabela com 1 coluna e 9.545 linhas.
- A execução de **2.sql** resulta em uma tabela com 1 coluna e 1 linha.
- A execução de **3.sql** resulta em uma tabela com 1 coluna e 50.863 linhas.
- A execução de **4.sql** resulta em uma tabela com 1 coluna e 1 linha.
- A execução de **5.sql** resulta em uma tabela com 2 colunas e 10 linhas.
- A execução de **6.sql** resulta em uma tabela com 1 coluna e 1 linha.
- A execução de **7.sql** resulta em uma tabela com 2 colunas e 6.864 linhas.
- A execução de **8.sql** resulta em uma tabela com 1 coluna e 4 linhas.
- A execução de **9.sql** resulta em uma tabela com 1 coluna e 18.237 linhas.
- A execução de **10.sql** resulta em uma tabela com 1 coluna e 1.887 linhas.
- A execução de **11.sql** resulta em uma tabela com 1 coluna e 5 linhas.
- A execução de **12.sql** resulta em uma tabela com 1 coluna e 6 linhas.
- A execução de **13.sql** resulta em uma tabela com 1 coluna e 176 linhas.

Observe que as contagens de linha não incluem linhas de cabeçalho que mostram apenas os nomes das colunas.

Se a sua consulta retornar um número de linhas ligeiramente diferente da saída esperada, certifique-se de que está lidando com as duplicatas corretamente! Para consultas que pedem uma lista de nomes, nenhuma pessoa deve ser listada duas vezes, mas duas pessoas diferentes com o mesmo nome devem ser listadas cada uma.

Execute o seguinte para avaliar a exatidão do seu código usando **check50**.

```
check50 cs50/problems/2021/x/movies
```

## Exercício 2: Fiftyville

Escreva consultas SQL para resolver um mistério.

## Um mistério em Fiftyville

O pato de estimação do CC50 foi roubado! A cidade de Fiftyville o convocou para resolver o mistério do pato roubado. Autoridades acreditam que o ladrão roubou o pato e, pouco depois, voou para fora da cidade com a ajuda de um cúmplice. Seu objetivo é identificar:

- Quem é o ladrão,
- Para qual cidade o ladrão fugiu, e
- Quem é o cúmplice do ladrão que os ajudou a escapar

Tudo o que você sabe é que o roubo **ocorreu em 28 de julho de 2020 na Chamberlin Street**.

Como você resolverá esse mistério? As autoridades de Fiftyville pegaram alguns dos registros da cidade na época do roubo e prepararam um banco de dados SQLite para você, **fiftyville.db**, que contém tabelas de dados de toda a cidade. Você pode consultar essa tabela usando consultas SQL **SELECT** para acessar os dados de seu interesse. Usando apenas as informações do banco de dados, sua tarefa é resolver o mistério.

## Começando

Veja como baixar esse problema em seu próprio IDE CS50. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Navegue até o diretório **pset7** que já deve existir.
- Execute <https://cdn.cs50.net/2020/fall/psets/7/fiftyville/fiftyville.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- Execute `unzip fiftyville.zip` para descompactar esse arquivo.
- Execute `rm fiftyville.zip` seguido por **yes** ou **y** para excluir o arquivo ZIP.
- Execute `ls`. Você deve ver um diretório chamado **fiftyville**, que estava dentro desse arquivo ZIP.
- Execute `cd fiftyville` para mudar para esse diretório.
- Execute `ls`. Você deve ver um arquivo **fiftyville.db**, um arquivo **log.sql** e um arquivo **answers.txt**.

## Especificação

Para este problema, tão importante quanto resolver o mistério em si é o processo que você usa para resolver o mistério. Em **log.sql**, mantenha um registro de todas as consultas SQL executadas no banco de dados. Acima de cada consulta, rotule cada um com um comentário (em SQL, comentários são quaisquer linhas que começam com `--`) descrevendo por que você está executando a consulta e / ou quais informações você espera obter dessa consulta específica. Você pode usar comentários no arquivo de log para adicionar notas adicionais sobre seu processo de pensamento à medida que resolve o mistério: em última análise, este arquivo deve servir como evidência do processo que você usou para identificar o ladrão!

Depois de resolver o mistério, complete cada uma das linhas em **answers.txt** preenchendo o nome do ladrão, a cidade para onde o ladrão escapou e o nome do cúmplice do ladrão que os ajudou a escapar da cidade. (Certifique-se de não alterar nenhum texto existente no arquivo ou de adicionar quaisquer outras linhas ao arquivo!)

Por fim, você deve enviar seus arquivos **log.sql** e **answers.txt**.

## Dicas

Execute `sqlite3 thirtyville.db` para começar a executar consultas no banco de dados.

- Durante a execução do **sqlite3**, a execução de **.tables** listará todas as tabelas no banco de dados.
- Durante a execução de **sqlite3**, a execução de **.schema TABLE\_NAME**, onde **TABLE\_NAME** é o nome de uma tabela no banco de dados, mostrará o comando **CREATE TABLE** usado para criar a tabela. Isso pode ser útil para saber quais colunas consultar!
- Você pode achar útil começar com a tabela **crime\_scene\_reports**. Comece procurando um relatório da cena do crime que corresponda à data e ao local do crime.

Consulte esta referência de [palavras-chave SQL](#) para alguma sintaxe SQL que pode ser útil!

## Testando

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**

```
check50 cs50/problems/2021/x/fiftyville
```

## Lab 7: Songs

# Laboratório 7: Songs

## Começando

Veja como baixar este laboratório em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute **cd** para garantir que você está em ~/ (ou seja, seu diretório pessoal, também conhecido como ~).
- Execute <https://cdn.cs50.net/2020/fall/labs/7/lab7.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- Execute **unzip lab7.zip** para descompactar esse arquivo.
- Execute **rm lab7.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- Execute **ls**. Você deve ver um diretório chamado **lab7**, que estava dentro desse arquivo ZIP.
- Execute **cd lab7** para mudar para esse diretório.
- Execute **ls**. Você deve ver um arquivo **songs.db**, e alguns arquivos **.sql** vazios também.

## Entendendo o problema

Fornecido a você é um arquivo chamado **songs.db**, um banco de dados SQLite que armazena dados do Spotify sobre músicas e seus artistas. Este conjunto de dados contém as 100 principais músicas transmitidas no Spotify em 2018. Em uma janela de terminal, execute **sqlite3 songs.db** para que você possa começar a executar consultas no banco de dados.

Primeiro, quando **sqlite3** solicitar que você forneça uma consulta, digite **.schema** e pressione Enter. Isso produzirá as declarações de output **CREATE TABLE** que foram usadas para gerar cada uma das tabelas no banco de dados. Ao examinar essas declarações, você pode identificar as colunas presentes em cada tabela.

Observe que todo **artist** tem um **id** e um **name** . Observe, também, que cada música tem um **name** , um **artist\_id** (correspondendo ao id do artista da música), bem como valores para dançabilidade, energia, tonalidade, volume, prolixidade (presença de palavras faladas em uma música), valência, ritmo e duração da música (medido em milissegundos).

O desafio à sua frente é escrever consultas SQL para responder a uma variedade de perguntas diferentes, selecionando dados de uma ou mais dessas tabelas.

## Detalhes de implementação

Para cada um dos problemas a seguir, você deve escrever uma única consulta SQL que produza os resultados especificados por cada problema. Sua resposta deve ter a forma de uma única consulta SQL, embora você possa aninhar outras consultas dentro de sua consulta.

Você **não deve** presumir nada sobre os ids de nenhuma música ou artista em particular: suas perguntas devem ser precisas mesmo se o id de qualquer música ou pessoa em particular for diferente. Finalmente, cada consulta deve retornar apenas os dados necessários para responder à pergunta: se o problema pede apenas que você envie os nomes das músicas, por exemplo, sua consulta não deve também produzir o tempo de cada música.

- No **1.sql**, escreva uma consulta SQL para listar os nomes de todas as músicas no banco de dados.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.
- No **2.sql**, escreva uma consulta SQL para listar os nomes de todas as músicas em ordem crescente de ritmo.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.
- No **3.sql**, escreva uma consulta SQL para listar os nomes das 5 músicas mais longas, em ordem decrescente de duração.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.
- No **4.sql**, escreva uma consulta SQL que liste os nomes de quaisquer músicas que tenham dançabilidade, energia e valência maior que 0,75.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.
- No **5.sql**, escreva uma consulta SQL que retorne a energia média de todas as músicas.
  - Sua consulta deve gerar uma tabela com uma única coluna e uma única linha contendo a energia média.
- No **6.sql**, escreva uma consulta SQL que lista os nomes das músicas de Post Malone.
  - Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.
  - Você não deve fazer suposições sobre qual é o **artist\_id** de Post Malone.
- No **7.sql**, escreva uma consulta SQL que retorne a energia média das músicas de Drake.
  - Sua consulta deve gerar uma tabela com uma única coluna e uma única linha contendo a energia média.
  - Você não deve fazer suposições sobre qual é o **artist\_id** de Drake.
- No **8.sql**, escreva uma consulta SQL que lista os nomes das músicas que apresentam “feat.” (participação) de outros artistas.

- Músicas que apresentam outros artistas incluirão “feat.” no nome da música.
- Sua consulta deve gerar uma tabela com uma única coluna para o nome de cada música.

## Dicas

Consulte [esta referência de palavras-chave SQL](#) para alguma sintaxe SQL que pode ser útil!

## Testando

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**.

```
check50 cs50/labs/2021/x/songs
```

## ANOTAÇÕES MÓDULO 8

# A Internet

Hoje daremos uma olhada na programação web, utilizando um conjunto de novas linguagens e tecnologias para criar aplicações gráficas e visuais para a internet.

A **Internet** é a rede de redes de computadores que se comunicam entre si, fornecendo a infraestrutura para enviar zeros e uns. No topo dessa base, podemos construir aplicativos que enviam e recebem dados.

**Roteadores** são computadores especializados, com CPUs e memória, cujo objetivo é retransmitir dados através de cabos ou tecnologias wireless, entre outros dispositivos na internet.

Os **protocolos** são um conjunto de convenções padrão, como um handshake físico, com o qual o mundo concordou para que os computadores se comuniquem. Por exemplo, existem certos padrões de zeros e uns, ou mensagens, que um computador deve usar para informar a um roteador para onde deseja enviar os dados.

**TCP/IP** são dois protocolos para enviar dados entre dois computadores. No mundo real, podemos escrever um endereço em um envelope para enviar uma carta a alguém, junto com nosso próprio endereço para receber uma carta. A versão digital de um envelope, ou uma mensagem com endereços de e para, é chamada de **pacote (packet)**.

**IP** significa protocolo de internet, um protocolo suportado por softwares de computadores modernos, que inclui uma forma padrão para os computadores se endereçarem. Os **endereços IP** são **endereços** exclusivos de computadores conectados à Internet, de forma que um pacote enviado de um computador a outro será repassado aos roteadores até chegar ao seu destino.

- Os roteadores têm, em sua memória, uma tabela que mapeia os endereços IP para os cabos, cada um conectado a outros roteadores, para que saibam para onde encaminhar os pacotes. Acontece que existem protocolos para os roteadores se comunicarem e descobrirem esses caminhos também.

**DNS**, sistema de nome de domínio, é outra tecnologia que traduz nomes de domínio como cs50.harvard.edu em endereços IP. O DNS é geralmente fornecido como um serviço pelo provedor de serviços de Internet mais próximo, ou ISP.

Por fim, o **TCP**, protocolo de controle de transmissão, é um protocolo final que permite a um único servidor, no mesmo endereço IP, fornecer vários serviços por meio do uso de um **número de porta**, um pequeno número inteiro adicionado ao endereço IP. Por exemplo, HTTP, HTTPS, e-mail e até mesmo Zoom têm seus próprios números de porta para esses programas usarem para se comunicar na rede.

O TCP também fornece um mecanismo para reenviar pacotes se um pacote for perdido e não recebido. Acontece que, na internet, existem vários caminhos para um pacote a ser enviado, uma vez que muitos roteadores estão interconectados. Portanto, um navegador da web, fazendo uma solicitação para um gato, pode ver seu pacote enviado por um caminho de roteadores e o servidor de resposta pode ver seus pacotes de resposta enviados por outro.

- Uma grande quantidade de dados, como uma imagem, será dividida em pedaços menores para que os pacotes tenham todos tamanhos semelhantes. Dessa forma, os roteadores ao longo da Internet podem enviar os pacotes de todos de forma mais justa e fácil. Neutralidade da rede (net neutrality) refere-se à ideia de que esses roteadores públicos tratam os pacotes igualmente, em oposição a permitir que pacotes de certas empresas ou de certos tipos sejam priorizados.
- Quando houver vários pacotes para uma única resposta, o TCP também especificará que cada um deles seja rotulado, como “1 de 2” ou “2 de 2”, para que possam ser combinados ou reenviados conforme necessário.

Com todas essas tecnologias e protocolos, somos capazes de enviar dados de um computador para outro e abstrair a Internet para construir aplicativos no topo.

## Desenvolvimento WEB

A web é um aplicativo executado sobre a Internet, o que nos permite obter páginas da web. Outros aplicativos, como o Zoom, fornecem videoconferência, e o e-mail também é outro aplicativo.

**HTTP**, ou protocolo de transferência de hipertexto, rege como os navegadores da web e os servidores da web se comunicam nos pacotes TCP/IP.

Dois comandos suportados por HTTP incluem **GET** e **POST**. GET permite que um navegador solicite uma página ou arquivo, e POST permite que um navegador envie dados para o servidor.

Um **URL** ou endereço da web pode ser semelhante a <https://www.example.com/>.

- **https** é o protocolo que está sendo usado e, neste caso, HTTPS é a versão segura do HTTP, garantindo que o conteúdo dos pacotes entre o navegador e o servidor sejam criptografados.
- **example.com** é o nome de domínio, onde **.com** é o domínio de nível superior, convencionalmente indicando o “tipo” de site da Web, como um site comercial para .com ou uma organização para .org. Agora, existem centenas de domínios de nível superior, e suas restrições variam quanto a quem pode usá-los, mas muitos deles permitem que qualquer pessoa se registre para um domínio.
- **www** é o nome do host que, por convenção, nos indica que se trata de um serviço “world wide web”. Não é obrigatório, então hoje muitos sites não estão configurados para incluí-lo.



- Finalmente, o / no final é uma solicitação para o arquivo padrão, como index.html , com o qual o servidor da web responderá.

Uma solicitação HTTP começará com:

```
GET / HTTP / 1.1
Host: www.example.com
```

...

- O **GET** indica que a solicitação é para algum arquivo e / indica o arquivo padrão. Uma solicitação pode ser mais específica e começar com **GET /index.html**.
- Existem diferentes versões do protocolo HTTP, portanto, **HTTP / 1.1** indica que o navegador está usando a versão 1.1.
- **Host:** [www.example.com](http://www.example.com) indica que a solicitação é para [www.example.com](http://www.example.com) , pois o mesmo servidor da web pode hospedar vários sites e domínios.

Uma resposta começará com:

```
HTTP / 1.1 200 OK
Content-Type: text/html
```

...

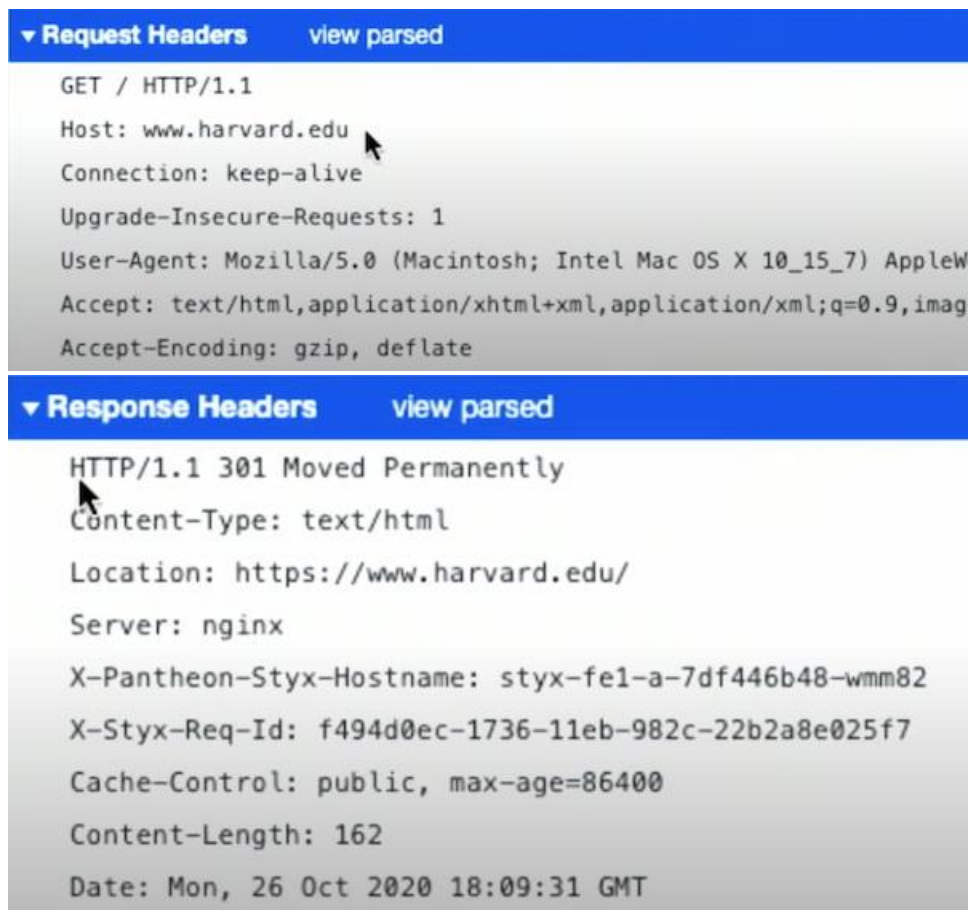
- O servidor web responderá com a versão do HTTP, seguido por um código de status, que é **200 OK** aqui, indicando que a solicitação era válida.
- Em seguida, o servidor da web indica o tipo de conteúdo em sua resposta, que pode ser texto, imagem ou outro formato.
- Finalmente, o resto do pacote ou pacotes incluirão o conteúdo.

Podemos ver um redirecionamento em um navegador digitando uma URL, como <http://www.harvard.edu>, e olhando para a barra de endereço após o carregamento da página, que mostrará <https://www.harvard.edu>. Os navegadores incluem ferramentas de desenvolvedor, que nos permitem ver o que está acontecendo. No menu do Chrome, por exemplo, podemos ir em Exibir> Desenvolvedor> Ferramentas do desenvolvedor, que abrirá um painel na tela. Na guia Rede, podemos ver que houve muitos pedidos de texto, imagens e outros dados que foram baixados separadamente para as páginas da web individuais.

A primeira solicitação, na verdade, retornou um código de status **301 Moved Permanently**, redirecionando nosso navegador de **http: // ...** para **https: // ...** :

| Name                                                | Status | Type                  | Initi |
|-----------------------------------------------------|--------|-----------------------|-------|
| <input type="checkbox"/> www.harvard.edu            | 301    | document / Redirect   | Oth   |
| <input type="checkbox"/> www.harvard.edu            | 200    | 301 Moved Permanently | ww    |
| <input type="checkbox"/> harvard.min.css?v=20180820 | 200    | stylesheet            | (ind  |

A solicitação e a resposta também incluem vários cabeçalhos ou dados adicionais:



- Observe que a resposta inclui um cabeçalho Location: para qual o navegador nos redireciona.

Outros códigos de status HTTP incluem:

- 200 OK
- 301 Moved Permanently
- 304 Not Modified
  - Isso permite que o navegador use seu cache, ou cópia local, de algum recurso como uma imagem, em vez de fazer com que o servidor o envie de volta.
- 307 Temporary Redirect
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 418 I'm a Teapot
- 500 Internal Server Error
  - O código com erros em um servidor pode resultar neste código de status.
- 503 Service Unavailable
- ...

Podemos usar uma ferramenta de linha de comando, **curl**, para conectar a um URL. Podemos executar:

```
curl -I http://safetyschool.org
HTTP/1.1 301 Moved Permanently
Server: Sun-ONE-Web-Server/6.1
Date: Wed, 26 Oct 2020 18:17:05 GMT
Content-length: 122
Content-type: text/html
Location: http://www.yale.edu
Connection: close
```

- Acontece que *safetyschool.org* redireciona para *yale.edu*!
- E *harvardsucks.org* é um site com outra pegadinha em Harvard!

Por fim, uma solicitação HTTP pode incluir entradas para servidores, como a string **q = cats** após o **?**:

```
GET / search?q=cats HTTP / 1.1
Host: www.google.com
```

...

- Isso usa um formato padrão para passar input, como argumentos de linha de comando, para servidores da web.

## HTML

Agora que podemos usar a Internet e o HTTP para enviar e receber mensagens, é hora de ver o que há no conteúdo das páginas da web. **HTML**, Hypertext Markup Language, não é uma linguagem de programação, mas sim usada para formatar páginas da web e dizer ao navegador como exibir as páginas, usando tags e atributos.

Uma página simples em HTML pode ter a seguinte aparência:

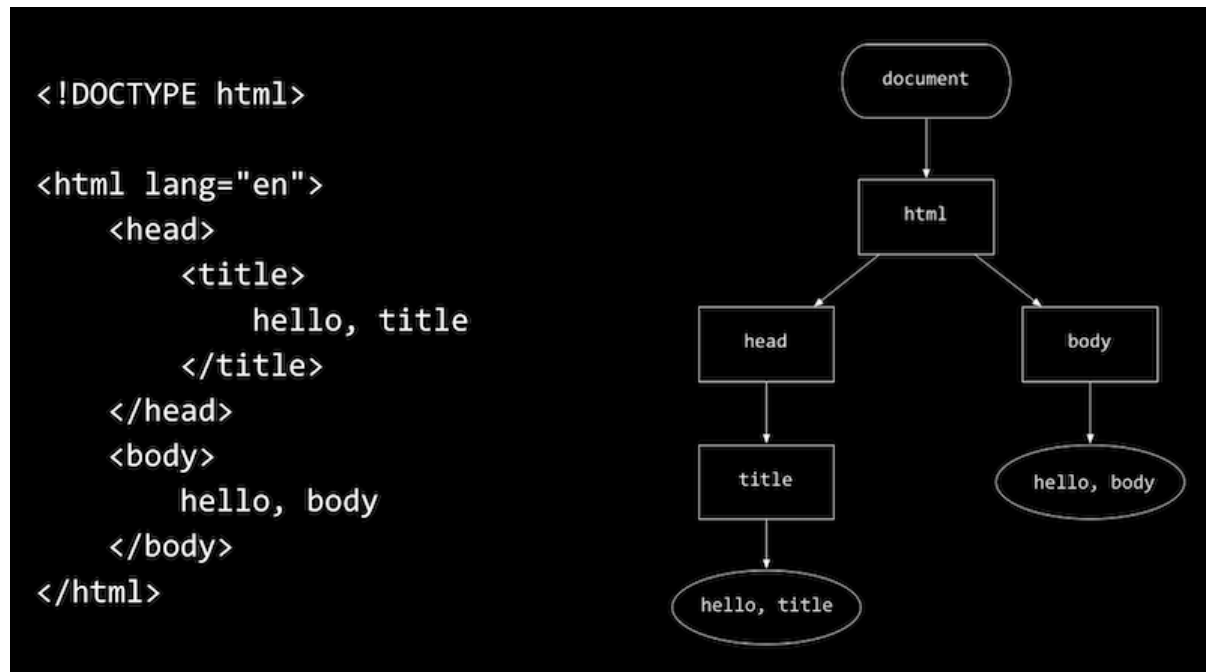
```
<!DOCTYPE html>
<html lang = "en" >
 <head>
 <title>
 hello, title
 </title>
 </head>
 <body>
 hello, body
 </body>
</html>
```

- A primeira linha é uma declaração de que a página segue o padrão HTML.
- A seguir vem uma **tag**, uma palavra entre colchetes, como **<html>** e **</html>**. A primeira é uma tag de início ou de abertura e a segunda é uma tag de fechamento. Nesse caso, as tags indicam o início e o fim da página HTML. A tag de início aqui

também tem um atributo , **lang = "en"** que especifica que o idioma da página será em inglês, para ajudar o navegador a traduzir a página, se necessário. </p>

- Dentro da tag <html> há mais duas tags, <head> e <body>, que são como nós filhos em uma árvore. E dentro de <head> está a tag <title> , cujo conteúdo vemos em uma guia ou título de janela em um navegador. Em <body> está o conteúdo da própria página, que veremos na visualização principal de um navegador também.

A página acima será carregada no navegador como uma estrutura de dados, como esta árvore:



- Observe que há uma hierarquia mapeando cada tag e seus filhos. Nós retangulares são tags, enquanto os ovais são texto.

Podemos salvar o código acima como um HTML em nossos computadores locais, o que funcionaria em um navegador, mas apenas para nós. Com o IDE CS50, podemos criar um arquivo HTML e realmente disponibilizá-lo na Internet.

Vamos criar hello.html com o código acima, e iniciar um servidor web instalado no CS50 IDE com **http-server**, um programa que irá ouvir HTTP solicitações e responder com páginas ou outro conteúdo.

O próprio CS50 IDE já está sendo executado em algum servidor web, usando as portas 80 e 443, portanto, nosso próprio servidor web dentro do IDE terá que usar uma porta diferente, **8080** por padrão. Veremos uma URL longa, terminando em **cs50.ws** e, se abrirmos essa URL, veremos uma lista de arquivos, incluindo hello.html.

De volta ao terminal de nosso IDE, veremos novas linhas de texto impressas por nosso servidor web, um log de solicitações que ele está recebendo.

Vamos dar uma olhada em [paragraphs.html](#) .

- Com a tag <p>, podemos indicar que cada seção do texto deve ser um parágrafo.
- Depois de salvar este arquivo, precisaremos atualizar o índice no navegador da web e, em seguida, abrir paragraphs.html .

Podemos adicionar cabeçalhos com tags que começam com h e ter níveis de **1** a **6** em [headings.html](#).

Também examinamos [list.html](#), [table.html](#) e [image.html](#) para adicionar listas, tabelas e imagens.

- Podemos usar a tag **<ul>** para criar uma lista não ordenada, como uma lista em tópicos, e **<ol>** para uma lista ordenada com números.
- As tabelas começam com uma marca **<table>** e têm marcas **<tr>** como linhas e marcas **<td>** para células individuais.
- Para image.html, podemos fazer upload de uma imagem para o IDE CS50, para incluí-la em nossa página, bem como usar o atributo **alt** para adicionar texto alternativo para acessibilidade.

Procurando documentação ou outros recursos online, podemos aprender as tags que existem em HTML e como usá-las.

Podemos criar links em [link.html](#) com a tag **<a>** ou âncora. O atributo **href** é para uma referência de hipertexto, ou simplesmente para onde o link deve nos levar, e dentro da tag está o texto que deve aparecer como o link.

- Poderíamos definir o **href** como <https://www.yale.edu> mas deixar *Harvard* dentro da tag, o que pode enganar os usuários ou até mesmo induzi-los a visitar uma versão falsa de algum site. **Phishing** é um ato de enganar os usuários, uma forma de engenharia social que inclui links enganosos.

Em search.html, podemos criar um formulário mais complexo que recebe a entrada do usuário e a envia para o mecanismo de pesquisa do Google:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <title>search</title>
 </head>
 <body>
 <form action="https://www.google.com/search" method="get">
 <input name="q" type="search">
 <input type="submit" value="Search">
 </form>
 </body>
</html>
```

- Primeiro, temos uma tag **<form>** que possui uma ação(**action**) do URL de pesquisa do Google, com um método de GET.
- Dentro do formulário, temos um **<input>**, com o nome **q**, e outro **<input>** com o tipo de envio(**submit**). Quando a segunda entrada, um botão, é clicado, o formulário irá anexar o texto na primeira entrada para o URL de ação, terminando com **search?q=...**

- Portanto, quando abrimos **search.html** em nosso navegador, podemos usar o formulário para pesquisar através do Google.
- Um formulário também pode usar um método POST, que não inclui os dados do formulário no URL, mas em outro lugar na solicitação.

## CSS

Podemos melhorar a estética de nossas páginas com CSS , Cascading Style Sheets, outra linguagem que informa ao nosso navegador como exibir tags em uma página. CSS usa **propriedades** ou pares de valores-chave, como **color: red;** para tags com seletores.

Em HTML, temos algumas opções para incluir CSS. Podemos adicionar uma tag <style> dentro da tag <head> , com estilos diretamente dentro, ou podemos vincular a um arquivo styles.css com uma tag <link> dentro da tag <head> .

Também podemos incluir CSS diretamente em cada tag:

```
<!DOCTYPE html>

<html lang = "en" >
 <head>
 <title> css </title>
 </head>
 <body>
 <header style = "font-size: large; text-align: center;" >
 John Harvard
 </header>
 <main style = "font-size: medium; text-align: center;" >
 Welcome to my homepage!
 </main>
 <footer style = "font-size: small; text-align: center;" >
 Copyright © John Harvard
 </footer>
 </body>
</html>
```

- As tags <header> , <main> e <footer> são como as tags <p> , indicando as seções em que o texto de nossa página está.
- Para cada tag, podemos adicionar um atributo de estilo(**style**), com o valor sendo uma lista de propriedades de valores-chave CSS, separadas por ponto e vírgula. Aqui, estamos definindo o tamanho da fonte(**font-size**) para cada tag e alinhando o texto no centro.
- Observe que podemos usar **&#169;**, uma **entidade HTML**, como um código para incluir algum símbolo em nossa página da web.

Podemos alinhar o texto de uma só vez:

```
<!DOCTYPE html>
```

```

<html lang="en">
 <head>
 <title>css</title>
 </head>
 <body style="text-align: center;">
 <header style="font-size: large;">
 John Harvard
 </header>
 <main style="font-size: medium;">
 Welcome to my home page!
 </main>
 <footer style="font-size: small;">
 Copyright © John Harvard
 </footer>
 </body>
</html>

```

- Aqui, o estilo aplicado à tag **<body>** é aplicado em cascata, ou se aplica a seus filhos, de forma que todas as seções internas também terão texto centralizado.

Para fatorar ou separar nosso CSS do HTML, podemos incluir estilos na tag **<head>** :

```

<!DOCTYPE html>

<html lang="en">
 <head>
 <style>
 header
 {
 font-size: large;
 text-align: center;
 }

 main
 {
 font-size: medium;
 text-align: center;
 }

 footer
 {
 font-size: small;
 text-align: center;
 }
 </style>
 <title>css</title>
 </head>

```

```

<body>
 <header>
 John Harvard
 </header>
 <main>
 Welcome to my home page!
 </main>
 <footer>
 Copyright © John Harvard
 </footer>
</body>
</html>

```

- Para cada tipo de tag, usamos um **seletor de tipo(type selector)** CSS para **definir** o estilo.

Também podemos usar um **seletor de class(class selector)** mais específico:

```

<!DOCTYPE html>
<html lang="en">
 <head>
 <style>
 .centered
 {
 text-align: center;
 }
 .large
 {
 font-size: large;
 }
 .medium
 {
 font-size: medium;
 }
 .small
 {
 font-size: small;
 }
 </style>
 <title>css</title>
 </head>
 <body>
 <header class="centered large">
 John Harvard

```



```

 </header>
 <main class="centered medium">
 Welcome to my home page!
 </main>
 <footer class="centered small">
 Copyright © John Harvard
 </footer>
</body>
</html>

```

- Podemos definir nossa própria classe CSS com um `.` seguido por uma palavra-chave que escolher, então aqui nós criamos `.large`, `.medium` e `.small`, cada um com alguma propriedade para o tamanho da fonte.
- Então, em qualquer número de tags no HTML de nossa página, podemos adicionar uma ou mais dessas classes com `class="centered large"`, reutilizando esses estilos.
- Podemos remover a redundância de `centered` e aplicá-la apenas à tag `<body>` também.

Finalmente, podemos pegar todo o CSS das propriedades e movê-los para outro arquivo com a tag `<link>`:

```

<!DOCTYPE html>

<html lang="en">
 <head>
 <link href="styles.css" rel="stylesheet">
 <title>css</title>
 </head>
 <body>
 <header class="centered large">
 John Harvard
 </header>
 <main class="centered medium">
 Welcome to my home page!
 </main>
 <footer class="centered small">
 Copyright © John Harvard
 </footer>
 </body>
</html>

```

- Agora, uma pessoa pode trabalhar no HTML e outra pode trabalhar no CSS, de forma mais independente.

Com CSS, também contaremos com referências e outros recursos para descobrir como usar as propriedades conforme necessário.

Podemos usar **pseudoseletores (pseudoselectors)**, que selecionam certos estados:

```

<!DOCTYPE html>
<html lang="en">
 <head>
 <style>
 #harvard
 {
 color: #ff0000;
 }
 #yale
 {
 color: #0000ff;
 }
 a
 {
 text-decoration: none;
 }
 a:hover
 {
 text-decoration: underline;
 }
 </style>
 <title>link</title>
 </head>
 <body>
 Visit <a href="https://www.harvard.edu/" id="harvard"
>Harvard or Yale.
 </body>
</html>

```

- Aqui, estamos usando **a:hover** para definir propriedades em tags **<a>** quando o usuário passa o mouse sobre elas.
- Também temos um atributo **id** em cada tag **<a>**, para definir cores diferentes em cada um com **seletores de ID (ID selectors)** que começam com um **#** no CSS.

## JavaScript

Para escrever um código que pode ser executado nos navegadores dos usuários ou no cliente, usaremos uma nova linguagem, **JavaScript**.

A sintaxe do JavaScript é semelhante à do C e do Python para construções básicas:

```
let counter = 0;
```

```
counter = counter + 1;
counter += 1;
counter++;

if(x < y)
{

}

if(x < y)
{

}
else
{

}

if(x < y)
{

}
else if(x > y)
{

}
else
{

}

while(true)
{

}

for(let i = 0; i < 3; i++)
{

}
```

- Observe que o JavaScript também é digitado livremente, com **let** sendo a palavra-chave para declarar variáveis de qualquer tipo.

Com o JavaScript, podemos alterar o HTML no navegador em tempo real. Podemos usar tags **<script>** para incluir nosso código diretamente ou de um arquivo **.js**.

Vamos criar outro formulário:

```
<!DOCTYPE html>
```

```

<html lang="en">
 <head>
 <script>
 function greet()
 {
 alert('hello, body');
 }
 </script>
 <title>hello</title>
 </head>
 <body>
 <form onsubmit="greet(); return false;">
 <input id="name" type="text">
 <input type="submit">
 </form>
 </body>
</html>

```

- Aqui, não adicionaremos uma ação ao nosso formulário, pois ele permanecerá na mesma página. Em vez disso, teremos um atributo onsubmit que chamará uma função que definimos em JavaScript e usará return false; para evitar que o formulário seja realmente enviado em qualquer lugar.
- Agora, se carregarmos essa página, veremos hello, body sendo mostrado quando enviarmos o formulário.

Como nossa tag de entrada, ou **elemento**, tem um ID de name , podemos usá-lo em nosso script:

```

<script>
 function greet()
 {
 let name = document.querySelector('#name').value;
 alert('hello, ' + name);
 }
</script>

```

- document é uma variável global que vem com JavaScript no navegador, e querySelector é outra função que podemos usar para selecionar um nó no DOM , Document Object Model ou a estrutura em árvore da página HTML. Depois de selecionar o elemento com o nome do ID , obtemos o value dentro do input e o adicionamos ao nosso alerta.
- Observe que JavaScript usa aspas simples para strings por convenção, embora aspas duplas também possam ser usadas, desde que correspondam a cada string.

Podemos adicionar mais atributos ao nosso formulário, para alterar o texto do espaço reservado, alterar o texto do botão, desativar o preenchimento automático ou focar o input automaticamente:

```
<form>
 <input autocomplete="off" autofocus id="name" placeholder="Name"
type="text">
 <input type="submit">
</form>
```

Também podemos ouvir **eventos(events)** em JavaScript, que ocorrem quando algo acontece na página. Por exemplo, podemos ouvir o evento de **submit** em nosso formulário e chamar a função **greet**:

```
<script>

function greet()
{
 let name = document.querySelector('#name').value;
 alert('hello, ' + name);
}

function listen() {
 document.querySelector('form').addEventListener('submit', greet);
}

document.addEventListener('DOMContentLoaded', listen);

</script>
```

- Aqui, em **listen**, passamos a função **greet** por nome, e não a chamamos ainda. O ouvinte do evento irá chamá-lo para nós quando o evento acontecer.
- Precisamos primeiro ouvir o evento **DOMContentLoaded**, já que o navegador lê nosso arquivo HTML de cima para baixo, e o **form** não existiria até que ele lesse todo o arquivo e carregasse o conteúdo. Portanto, ao ouvir esse evento e chamar nossa função de **listen**, sabemos que form existirá.

Também podemos usar **funções anônimas** em JavaScript:

```
<script>

document.addEventListener('DOMContentLoaded', function() {
 document.querySelector('form').addEventListener('submit',
function() {
 let name = document.querySelector('#name').value;
 alert('hello, ' + name);
});
});

</script>
```

- Podemos passar uma função lambda com a sintaxe **function()**, então aqui passamos os dois ouvintes diretamente para **addEventListener**.

Além de **submit**, existem muitos outros eventos que podemos ouvir:

- **blur**
- **change**
- **click**
- **drag**
- **focus**
- **keyup**
- **load**
- **mousedown**
- **mouseover**
- **mouseup**
- **submit**
- **touchmove**
- **unload**
- ...

Por exemplo, podemos ouvir o evento **keyup** e alterar o DOM assim que liberarmos uma tecla:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <script>
 document.addEventListener('DOMContentLoaded', function()
{
 let input = document.querySelector('input');
 input.addEventListener('keyup', function(event) {
 let name = document.querySelector('#name');
 if (input.value) {
 name.innerHTML = `hello, ${input.value}`;
 }
 else {
 name.innerHTML = 'hello, whoever you are';
 }
 });
 });
 </script>
 <title>hello</title>
 </head>
 <body>
 <form>
```

```

 <input autocomplete="off" autofocus placeholder="Name"
type="text">
 </form>
 <p id="name"></p>
</body>
</html>

```

- Observe que também podemos substituir strings em JavaScript, com **`${input.value}`** dentro de uma string cercada por crases, ``` .

Também podemos alterar o estilo de maneira programática:

```

<!DOCTYPE html>
<html lang="en">
 <head>
 <title>background</title>
 </head>
 <body>
 <button id="red">R</button>
 <button id="green">G</button>
 <button id="blue">B</button>
 <script>

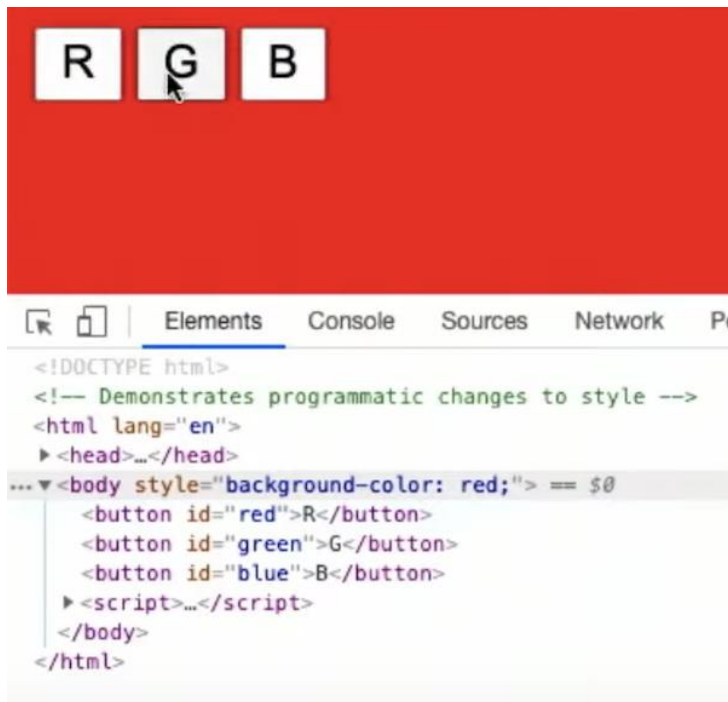
 let body = document.querySelector('body');
 document.querySelector('#red').onclick = function() {
 body.style.backgroundColor = 'red';
 };
 document.querySelector('#green').onclick = function() {
 body.style.backgroundColor = 'green';
 };
 document.querySelector('#blue').onclick = function() {
 body.style.backgroundColor = 'blue';
 };

 </script>
 </body>
</html>

```

- Depois de selecionar um elemento, podemos usar a propriedade **style** para definir os valores das propriedades CSS também. Aqui, temos três botões, cada um dos quais com um ouvinte **onclick** que altera a cor de fundo do elemento **<body>**
- Observe aqui que nossa tag **<script>** está no final de nosso arquivo HTML, então não precisamos ouvir o evento **DOMContentLoaded** , uma vez que o resto do DOM já terá sido lido pelo navegador.

Também nas ferramentas de desenvolvedor de um navegador, podemos ver o DOM e todos os estilos aplicados por meio da guia Elements :



- Podemos até usar isso para alterar uma página em nosso navegador depois de carregada, clicando em algum elemento e editando o HTML. Mas essas alterações serão feitas apenas em nosso navegador, não em nosso arquivo HTML original ou em alguma página da web em outro lugar.

Em [size.html](#), podemos definir o tamanho da fonte com um menu suspenso via JavaScript, e em [blink.html](#) podemos fazer um elemento “piscar”, alternando entre visível e oculto.

Com [geolocation.html](#), podemos pedir ao navegador as coordenadas GPS de um usuário e, com [autocomplete.html](#), podemos preencher automaticamente algo que digitamos, com palavras de um arquivo de dicionário.

Finalmente, podemos usar Python para escrever código que se conecta a outros dispositivos em uma rede local, como uma lâmpada, por meio de uma API , interface de programação de aplicativo. A **API** da nossa lâmpada, em particular, aceita solicitações em determinados URLs:

```
import os
import requests

USERNAME = os.getenv("USERNAME")
IP = os.getenv("IP")

URL = f"http://{IP}/api/{USERNAME}/lights/1/state"

requests.put(URL, json={"on": False})
```

- Com este código, podemos usar o método PUT para enviar uma mensagem para a nossa lâmpada, desligando-a.
- Usamos variáveis de ambiente, valores armazenados em outro lugar em nosso computador, para nosso nome de usuário e endereço IP.

Agora, com um pouco mais de lógica, podemos fazer nossa lâmpada piscar:



```

import os
import requests
import time

USERNAME = os.getenv("USERNAME")
IP = os.getenv("IP")
URL = f"http://{IP}/api/{USERNAME}/lights/1/state"

while True:
 requests.put(URL, json={"bri": 254, "on": True})
 time.sleep(1)
 requests.put(URL, json={"on": False})
 time.sleep(1)

```

- Vamos montar HTML, CSS, JavaScript, Python e SQL na próxima vez!

## EXERCÍCIOS MÓDULO 8

### Exercício 1: Homepage

Construa uma página inicial simples usando HTML, CSS e JavaScript.

## Background

A internet possibilitou coisas incríveis: podemos usar um mecanismo de busca para pesquisar qualquer coisa imaginável, nos comunicar com amigos e familiares ao redor do mundo, jogar, fazer cursos e muito mais. Mas acontece que quase todas as páginas que podemos visitar são construídas em três idiomas principais, cada um dos quais serve a uma finalidade ligeiramente diferente:

- *HTML, ou HyperText Markup Language*, que é usado para descrever o conteúdo de sites;
- *CSS, Cascading Style Sheets*, que é usado para descrever a estética de sites; e
- *JavaScript*, que é usado para tornar os sites interativos e dinâmicos.

Crie uma página inicial simples que apresente você, seu hobby ou atividade extracurricular favorita, ou qualquer outra coisa do seu interesse.

## Começando

Veja como baixar o “código de distribuição” desse problema (ou seja, código inicial) em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute `cd ~` (ou simplesmente `cd` sem argumentos) para garantir que você está em seu diretório pessoal.
- Execute `mkdir pset8` para fazer (ou seja, criar) um diretório chamado **pset8**.
- Execute `cd pset8` para mudar para (ou seja, abrir) esse diretório.
- Execute <http://cdn.cs50.net/2020/fall/psets/8/homepage/homepage.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.

- Execute `unzip homepage.zip` para descompactar esse arquivo.
- Execute `rm homepage.zip` seguido por `yes` ou `y` para excluir o arquivo ZIP.
- Execute `ls`. Você deverá ver um diretório chamado **homepage**, que está dentro desse arquivo ZIP.
- Execute `cd homepage` para mudar para esse diretório.
- Execute `ls`. Você deve ver a distribuição desse problema, incluindo **index.html** e **styles.css**.
- Você pode iniciar imediatamente um servidor para visualizar o site executando

\$ http-server

na janela do terminal e clicando no link que aparece.

## Especificação

Implemente no diretório da sua *homepage* um site que deve:

- Contém pelo menos quatro páginas **.html** diferentes, pelo menos uma delas é `index.html` (a página principal do seu site), e deve ser possível ir de qualquer página do seu site para qualquer outra página seguindo um ou mais hiperlinks.
- Use pelo menos dez (10) tags HTML distintas além de `<html>`, `<head>`, `<body>` e `<title>`. Usar alguma tag (por exemplo, `<p>`) várias vezes ainda conta como apenas uma (1) dessas dez!
- Integre um ou mais recursos do Bootstrap em seu site. Bootstrap é uma biblioteca popular (que vem com muitas classes CSS e mais) por meio da qual você pode embelezar seu site. Consulte a [documentação do Bootstrap](#) para começar. Em particular, você pode encontrar alguns dos [componentes interessantes do Bootstrap](#). Para adicionar Bootstrap ao seu site, é suficiente incluir

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+0GpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx"
crossorigin="anonymous"></script>
```

- no `<head>` de suas páginas, abaixo do qual você também pode incluir

```
<link href = "styles.css" rel = "stylesheet" >
```

- para vincular seu próprio CSS.

- Tenha pelo menos um stylesheet (arquivo que contém diretrizes de estilo/design) de sua própria criação, **styles.css**, que usa pelo menos cinco (5) seletores CSS diferentes (por exemplo, tag (**example**), class (**.example**) ou ID (**#example**)), e dentro dos quais você usa um total de pelo menos cinco (5) propriedades CSS diferentes, como font-size ou margin ; e
- Integre um ou mais recursos de JavaScript em seu site para torná-lo mais interativo. Por exemplo, você pode usar JavaScript para adicionar alertas, para ter um efeito em um intervalo recorrente ou para adicionar interatividade a botões, menus suspensos ou formulários. Sinta-se à vontade para ser criativo!
- Certifique-se de que seu site tenha uma boa aparência em navegadores, tanto em dispositivos móveis quanto em laptops e desktops.

## Testando

Se quiser ver a aparência do seu site enquanto você trabalha nele, há duas opções:

- No CS50 IDE, navegue até o diretório da sua **homepage** (lembra como?) e execute
- `$ http-server`
- No CS50 IDE, clique com o botão direito (ou Ctrl + clique, em um Mac) no diretório da homepage na árvore de arquivos à esquerda. A partir das opções que aparecem, selecione Servir , que deve abrir uma nova guia em seu navegador (pode demorar um ou dois segundos) com seu site nela.

Lembre-se também de que, ao abrir as Ferramentas do desenvolvedor no Google Chrome, você pode simular uma visita à sua página em um dispositivo móvel clicando no ícone em forma de telefone à esquerda de **Elementos** na janela de ferramentas do desenvolvedor ou, uma vez que a guia Ferramentas do desenvolvedor já tenha sido aberta , digitando Ctrl + Shift + M em um PC ou Cmd + Shift + M em um Mac, em vez de precisar visitar seu site em um dispositivo móvel separadamente!

## Avaliação

Sem **check50** para esta tarefa! Em vez disso, a exatidão do seu site será avaliada com base em se você atende aos requisitos da especificação, conforme descrito acima, e se o seu HTML é bem formado e válido. Para garantir que suas páginas atendam esses requisitos, você pode usar este serviço de validação de marcação , copiando e colando seu HTML diretamente na caixa de texto fornecida. Tome cuidado para eliminar quaisquer avisos ou erros sugeridos pelo validador antes de enviar!

Considere também:

- se a estética de seu site é intuitiva e direta para o usuário navegar;
- se o seu CSS foi dividido em um (s) arquivo (s) CSS separado (s); e
- se você evitou a repetição e a redundância ao “cascatear” propriedades de estilo de tags “mãe”: - ou seja, se tags que dividem muitas propriedades estão agrupadas de forma a evitar repetição.

O **style50** não oferece suporte a arquivos HTML e, portanto, é sua responsabilidade recuar e alinhar suas tags HTML de forma limpa. Saiba também que você pode criar um comentário HTML com:

```
<!-- O comentário vai aqui -->
```

mas comentar seu código HTML não é tão imperativo quanto ao comentar o código em, digamos, C ou Python. Você também pode comentar seu CSS, em arquivos CSS, com:

```
/ * O comentário vai aqui * /
```

## Dicas

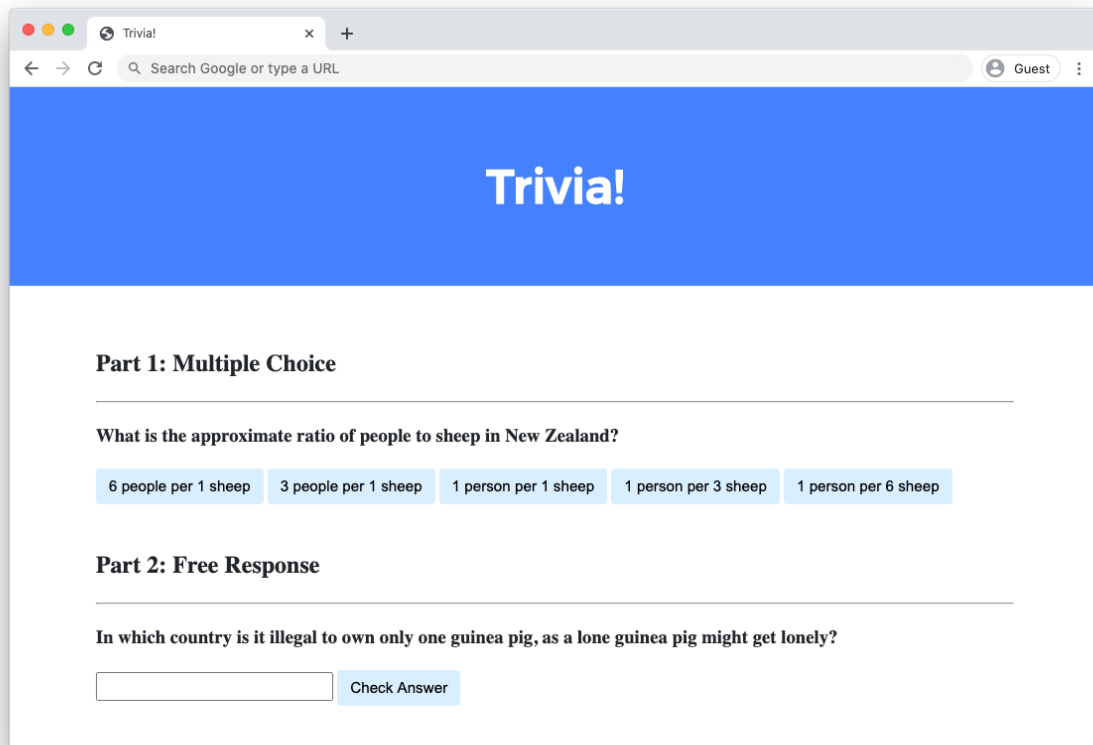
Para guias bastante abrangentes sobre os idiomas introduzidos neste problema, verifique estes tutoriais:

- [HTML](#)
- [CSS](#)
- [JavaScript](#)

### Lab 8: Trivia

## Lab 8: Trivia

Escreva uma página da web que permita aos usuários responder a perguntas triviais.



## Vamos começar...

Veja como baixar este laboratório em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute **cd** para garantir que você está em ~/ (ou seja, seu diretório pessoal, também conhecido como ~ ).
- Execute <https://cdn.cs50.net/2020/fall/labs/8/lab8.zip> para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- Execute **unzip lab8.zip** para descompactar esse arquivo.
- Execute **rm lab8.zip** seguido por **yes** ou **y** para excluir o arquivo ZIP.
- Execute **ls**. Você deve ver um diretório chamado **lab8** , que estava dentro desse arquivo ZIP.
- Execute **unzip lab8** para mudar para esse diretório.
- Execute **ls**. Você deve ver um arquivo **index.html** e um **styles.css**.

## Detalhes de implementação

Projete uma página da web usando HTML, CSS e JavaScript para permitir que os usuários respondam a perguntas triviais.

- Em **index.html** , adicione abaixo da “Parte 1” uma pergunta trivial de múltipla escolha de sua preferência com HTML.
  - Você deve usar um cabeçalho **h3** para o texto de sua pergunta.

- Você deve ter um **button** para cada uma das opções de resposta possíveis. Deve haver pelo menos três opções de resposta, das quais exatamente uma deve estar correta.
- Usando JavaScript, adicione lógica para que os botões mudem de cor quando um usuário clicar neles.
  - Se um usuário clicar em um botão com uma resposta incorreta, o botão deve ficar vermelho e o texto deve aparecer abaixo da pergunta que diz “Incorreto”.
  - Se um usuário clicar em um botão com a resposta correta, o botão deve ficar verde e o texto deve aparecer abaixo da pergunta que diz “Correto!”.
- Em **index.html**, adicione abaixo da “Parte 2” uma pergunta de resposta livre baseada em texto de sua escolha com HTML.
  - Você deve usar um cabeçalho **h3** para o texto de sua pergunta.
  - Você deve usar um campo de **input** para permitir que o usuário digite uma resposta.
  - Você deve usar um **button** para permitir que o usuário confirme sua resposta.
- Usando JavaScript, adicione lógica para que o campo de texto mude de cor quando um usuário confirmar sua resposta.
  - Se o usuário digitar uma resposta incorreta e pressionar o botão de confirmação, o campo de texto ficará vermelho e o texto deverá aparecer abaixo da pergunta que diz “Incorreto!”.
  - Se o usuário digitar a resposta correta e pressionar o botão de confirmação, o campo de entrada deve ficar verde e o texto deve aparecer abaixo da pergunta que diz “Correto!”.

Opcionalmente, você também pode:

- Editar **styles.css** para alterar o CSS da sua página da web!
- Adicione outras perguntas a sua trivía, se desejar!

## Dicas

- Use [document.querySelector](#) para consultar um único elemento HTML.
- Use [document.querySelectorAll](#) para consultar vários elementos HTML que correspondem a uma consulta. A função retorna uma matriz de todos os elementos correspondentes.

## Testando

**Não há check50 para este laboratório**, pois as implementações variam de acordo com suas perguntas! Mas certifique-se de testar as respostas incorretas e corretas para cada uma de suas perguntas para garantir que sua página da web responda de forma adequada.

Execute **http-server** em seu terminal enquanto em seu diretório **lab8** para iniciar um servidor web que atende sua página web.

## ANOTAÇÕES MÓDULO 9

# Desenvolvimento WEB

Hoje vamos criar aplicativos da web mais avançados escrevendo código que é executado no servidor.

Na semana passada, usamos **http-server** no CS50 IDE como um **servidor web**, um programa que escuta conexões e solicitações e responde com páginas web ou outros recursos.

Uma solicitação HTTP possui cabeçalhos, como:

```
GET / HTTP / 1.1
```

...

Esses cabeçalhos podem solicitar algum arquivo ou página ou enviar dados do navegador de volta ao servidor.

Embora o servidor http responda apenas com páginas estáticas, podemos usar outros servidores da web que “leem” ou analisam cabeçalhos de solicitação, como **GET / search? Q = cats HTTP / 1.1**, para retornar as páginas dinamicamente.

## Flask

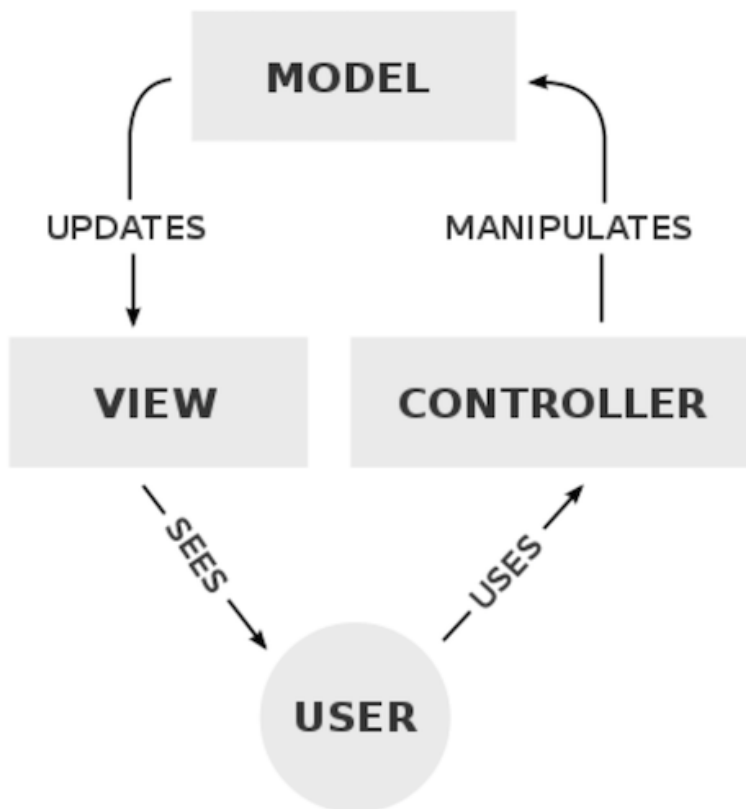
Usaremos Python e uma biblioteca chamada **Flask** para escrever nosso próprio servidor web, implementando recursos adicionais. O Flask também é um **framework**, onde a biblioteca de código também vem com um conjunto de convenções de como deve ser usada. Por exemplo, como outras bibliotecas, o Flask inclui funções que podemos usar para analisar solicitações individualmente, mas como uma estrutura, também requer que o código do nosso programa seja organizado de uma determinada maneira:

```
application.py
requirements.txt
static/
templates/
```

- **application.py** terá o código Python para nosso servidor web.
- **requirements.txt** inclui uma lista de bibliotecas necessárias para nosso aplicativo.
- **static/** é um diretório de arquivos estáticos, como arquivos CSS e JavaScript.
- **templates/** é um diretório para arquivos que serão usados para criar nosso HTML final.

Existem muitas estruturas de servidor web para cada uma das linguagens populares, e o Flask será um representante que usaremos hoje.

O Flask também implementa um **padrão de design** específico, ou a maneira como nosso programa e código são organizados. Para Flask, o padrão de design é geralmente MVC, ou Model – view – controller:



- O controlador é nossa lógica e código que gerencia nossa aplicação de maneira geral, com base na entrada do usuário. No Flask, este será o nosso código Python.
- A visualização é a interface do usuário, como o HTML e CSS que o usuário verá e com os quais irá interagir.
- O modelo são os dados de nosso aplicativo, como um banco de dados SQL ou arquivo CSV.

A aplicação de Flask mais simples pode ter a seguinte aparência:

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route("/")
```

```
def index():
```

```
 return "hello, world"
```

- Primeiro, importaremos o **Flask** da biblioteca **flask**, que usa uma letra maiúscula como nome principal.
- Em seguida, criaremos uma variável de **app** atribuindo o nome do nosso arquivo à variável **Flask**.
- A seguir, identificaremos uma função para a rota / ou URL com **@ app.route** . O símbolo **@** em Python é chamado de decorador, que aplica uma função a outra.
- Chamaremos o **index** da função , pois ele deve responder a uma solicitação de / , a página padrão. E nossa função apenas responderá com uma string por enquanto.



No IDE CS50, podemos ir para o diretório com o código do nosso aplicativo e digitar **flask run** para iniciá-lo. Veremos um URL e podemos abri-lo para ver **hello, world**.

Atualizaremos nosso código para realmente retornar HTML com a função **render\_template**, que encontra um arquivo fornecido e retorna seu conteúdo:

```
from flask import Flask, render_template
```

```
app = Flask(__name__)
```

```
@app.route("/")
```

```
def index():
```

```
 return render_template("index.html")
```

- Precisaremos criar um diretório **templates/** e criar um arquivo **index.html** com algum conteúdo dentro dele.
- Agora, digitar **flask run** retornará esse arquivo HTML quando visitarmos a URL do nosso servidor.

Vamos passar um argumento para **render\_template** em nosso código de controlador:

```
from flask import Flask, render_template, request
```

```
app = Flask(__name__)
```

```
@app.route("/")
```

```
def index():
```

```
 return render_template("index.html",
name=request.args.get("name", "world"))
```

- Acontece que podemos fornecer a **render\_template** qualquer argumento nomeado, como **name**, e ele irá substituí-lo em nosso modelo ou em nosso arquivo HTML com marcadores de posição.
  - Em **index.html**, substituiremos **hello, world** por **hello**, para dizer ao Flask onde substituir a variável de **name**:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
 <head>
```

```
 <title>hello</title>
```

```
 </head>
```

```
 <body>
```

```
 hello, {{ name }}
```

```
 </body>
```

```
</html>
```

- Podemos usar a variável de **request** da biblioteca Flask para obter um parâmetro da solicitação de HTTP, neste caso também **name**, e voltar para um padrão de **world** se nenhum foi fornecido.

- Agora, quando reiniciarmos nosso servidor após fazer essas alterações, e visitarmos a página padrão com uma URL como `/?name=David`, veremos essa mesma entrada retornada para nós no HTML gerado por nosso servidor.

Podemos presumir que a consulta de pesquisa do Google, em `/search?q=cats`, também é analisada por algum código para o parâmetro `q` e passada para algum banco de dados para obter todos os resultados relevantes. Esses resultados são então usados para gerar a página HTML final.

## Formulários

Vamos mover nosso modelo original para `greet.html`, de forma que ele cumprimente o usuário com seu nome. Em `index.html`, criaremos um formulário:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <title>hello</title>
 </head>
 <body>
 <form action="/greet" method="get">
 <input name="name" type="text">
 <input type="submit">
 </form>
 </body>
</html>
```

- Enviaremos o formulário para a rota `/greet`, e teremos uma entrada para o parâmetro de `name` e outra para o botão de envio.
- Em nosso controlador `applications.py`, também precisaremos adicionar uma função para a rota `/greet`, que é quase exatamente o que tínhamos para `/` antes:

```
@app.route("/")
def index():
 return render_template("index.html")
```

```
@app.route("/greet")
def greet():
 return render_template("greet.html",
 name=request.args.get("name", "world"))
```

- Nosso formulário em `index.html` será estático, pois pode ser o mesmo sempre.

Agora, podemos executar nosso servidor, ver nosso formulário na página padrão e usá-lo para gerar outra página.

# POST

Nosso formulário acima usou o método GET, que inclui os dados do nosso formulário na URL.

Vamos mudar o método em nosso HTML: `<\form action="/greet" method="post">`. Nosso controlador também precisará ser alterado para aceitar o método POST e procurar o parâmetro em outro lugar:

```
@app.route("/greet", methods=["POST"])
def greet():
 return render_template("greet.html",
name=request.form.get("name", "world"))
```

- Embora **request.args** seja para parâmetros em uma solicitação GET, temos que usar **request.form** no Flask para parâmetros em uma solicitação POST.

Agora, quando reiniciamos nosso aplicativo após fazer essas alterações, podemos ver que o formulário nos leva a `/greet`, mas o conteúdo não está mais incluído na URL.

## Layouts

Em **index.html** e **greet.html**, temos alguns códigos HTML repetidos. Com apenas HTML, não podemos compartilhar código entre arquivos, mas com modelos Flask (e outras estruturas da web), podemos fatorar esse conteúdo comum.

Criaremos outro modelo, **layout.html**:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <title>hello</title>
 </head>
 <body>
 {% block body %}{% endblock %}
 </body>
</html>
```

- O Flask suporta Jinja, uma linguagem de templates, que usa a sintaxe `{% %}` para incluir blocos de espaço reservado ou outros pedaços de código. Aqui, nomeamos nosso bloco **body**, pois ele contém o HTML que deve ir no elemento `<body>`.

Em **index.html**, usaremos o **layout.html** como modelo e apenas definiremos o bloco do **body** com:

```
{% extends "layout.html" %}

{% block body %}

 <form action="/greet" method="post">
 <input autocomplete="off" autofocus name="name"
placeholder="Name" type="text">
```

```
 <input type="submit">
 </form>

{% endblock %}
```

Da mesma forma, em `greet.html`, definimos o bloco do corpo apenas com a saudação:

```
{% extends "layout.html" %}
{% block body %}
hello, {{ name }}
{% endblock %}
```

Agora, se reiniciarmos nosso servidor e visualizarmos o código-fonte de nosso HTML após abrirmos a URL de nosso servidor, veremos uma página completa com nosso formulário dentro de nosso arquivo HTML, gerado pelo Flask.

Podemos até reutilizar a mesma rota para oferecer suporte aos métodos GET e POST:

```
@app.route("/", methods=["GET", "POST"])

def index():
 if request.method == "POST":
 return render_template("greet.html",
 name=request.form.get("name", "world"))
 return render_template("index.html")
```

- Primeiro, verificamos se o **method** da **request** é uma solicitação POST. Neste caso, procuraremos o parâmetro **name** e retornaremos o HTML do modelo **greet.html**. Caso contrário, retornaremos o HTML de **index.html**, que contém nosso formulário.
- Também precisaremos alterar a **action** do formulário para o padrão de rota `/`.

## Frosh IMs

Um dos primeiros aplicativos da web de David foi para que os alunos do campus se registrassem em “IMs frosh”, ou seja, jogos internos.

Usaremos um **layout.html** semelhante ao que tínhamos antes:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <meta name="viewport" content="initial-scale=1,
width=device-width">
 <title>froshims</title>
 </head>
 <body>
 {% block body %}{% endblock %}
 </body>
</html>
```

- Uma tag **<meta>** em **<head>** nos permite adicionar mais metadados à nossa página. Neste caso, estamos adicionando um atributo de **content** para os metadados da **viewport**, a fim de dizer ao navegador para dimensionar automaticamente o tamanho e as fontes de nossa página para o dispositivo.

Em nosso **application.py**, retornaremos nosso modelo **index.html** para o padrão de rota **/**:

```
from flask import Flask, render_template, request

app = Flask(__name__)
```

```
SPORTS = [
 "Dodgeball",
 "Flag Football",
 "Soccer",
 "Volleyball",
 "Ultimate Frisbee"
]
```

```
@app.route("/")
def index():
 return render_template("index.html")
```

Nosso modelo **index.html** terá a seguinte aparência:

```
{% extends "layout.html" %}

{% block body %}
 <h1>Register</h1>

 <form action="/register" method="post">
 <input autocomplete="off" autofocus name="name"
placeholder="Name" type="text">
 <select name="sport">
 <option disabled selected value="">Sport</option>
 <option value="Dodgeball">Dodgeball</option>
 <option value="Flag Football">Flag Football</option>
 <option value="Soccer">Soccer</option>
 <option value="Volleyball">Volleyball</option>
 <option value="Ultimate Frisbee">Ultimate
Frisbee</option>
 </select>
 <input type="submit" value="Register">

 </form>
{% endblock %}
```

- Teremos um formulário como antes, e teremos um menu **<select>** com opções para cada esporte.

Em nosso **application.py**, permitiremos POST para nossa rota **/register**:

```
@app.route("/register", methods=["POST"])
def register():

 if not request.form.get("name") or not request.form.get("sport"):
 return render_template("failure.html")

 return render_template("success.html")
```

- Verificaremos se os valores do nosso formulário são válidos e, em seguida, retornaremos um modelo dependendo dos resultados, embora não estejamos realmente fazendo nada com os dados ainda.

Mas um usuário pode alterar o HTML do formulário em seu navegador e enviar uma solicitação que contenha algum outro esporte como opção selecionada!

Verificaremos se o valor para sport é válido criando uma lista em **application.py**:

```
from flask import Flask, render_template, request

app = Flask(__name__)

SPORTS = [
 "Dodgeball",
 "Flag Football",
 "Soccer",
 "Volleyball",
 "Ultimate Frisbee"
]

@app.route("/")
def index():
 return render_template("index.html", sports=SPORTS)

...
```

Em seguida, passaremos essa lista para o modelo **index.html**.

Em nosso modelo, podemos até usar loops para gerar uma lista de opções da lista de cordas passadas como **sports**:

```
...
<select name="sport">
 <option disabled selected value="">Sport</option>
 {% for sport in sports %}
 <option value="{{ sport }}">{{ sport }}</option>
 {% endfor %}
</select>
...
```

Por fim, podemos verificar se o **sport** enviado na solicitação POST está na lista **SPORTS** em **application.py**:

```
...
@app.route("/register", methods=["POST"])
def register():

 if not request.form.get("name") or request.form.get("sport") not
in SPORTS:
 return render_template("failure.html")

 return render_template("success.html")
```

Podemos alterar o menu de seleção em nosso formulário para caixas de seleção, para permitir vários esportes:

```
{% extends "layout.html" %}

{% block body %}
 <h1>Register</h1>

 <form action="/register" method="post">

 <input autocomplete="off" autofocus name="name"
placeholder="Name" type="text">
 {% for sport in sports %}
 <input name="sport" type="checkbox" value="{{ sport }}">
{{ sport }}
 {% endfor %}
 <input type="submit" value="Register">

 </form>
{% endblock %}
```

- Em nossa função de **register**, podemos chamar **request.form.getlist** para obter a lista de opções marcadas.

Também podemos usar “radio buttons”, o que permitirá que apenas uma opção seja escolhida por vez.

## Armazenamento de dados

Vamos armazenar nossos alunos registrados, ou inscritos, em um dicionário na memória de nosso servidor web:

```
from flask import Flask, redirect, render_template, request

app = Flask(__name__)

REGISTRANTS = {}
```

...

```
@app.route("/register", methods=["POST"])
def register():

 name = request.form.get("name")
 if not name:
 return render_template("error.html", message="Missing name")

 sport = request.form.get("sport")
 if not sport:
 return render_template("error.html", message="Missing sport")
 if sport not in SPORTS:
 return render_template("error.html", message="Invalid sport")

 REGISTRANTS[name] = sport

 return redirect("/registrants")
```

- Criaremos um dicionário chamado **REGISTRANTS**, e no **register** verificaremos primeiro o **name** e o **sport**, retornando uma mensagem de erro diferente em cada caso. Assim, podemos armazenar com segurança o nome e o esporte em nosso dicionário **REGISTRANTS** e redirecionar para outra rota que exibirá os alunos registrados.
- O modelo de mensagem de erro, por sua vez, exibirá apenas a mensagem:

```
{% extends "layout.html"%}

{% block body%}
 {{ mensagem }}
{% endblock%}
```

Vamos adicionar a rota /registrants e o modelo para mostrar aos alunos registrados:

```
@app.route("/registrants")
def registrants():
 return render_template("registrants.html",
 registrants=REGISTRANTS)
```

- Em nossa rota, passaremos do dicionário **REGISTRANTS** ao template como um parâmetro chamado **registrants**:

```
{% extends "layout.html" %}

{% block body %}
 <h1>Registrants</h1>
 <table>
 <thead>
```



```

 <tr>
 <th>Name</th>
 <th>Sport</th>
 </tr>
 </thead>
 <tbody>
 {% for name in registrants %}
 <tr>
 <td>{{ name }}</td>
 <td>{{ registrants[name] }}</td>
 </tr>
 {% endfor %}
 </tbody>
</table>
{% endblock %}

```

- Nosso modelo terá uma tabela, com uma linha de título e uma linha para cada chave e valor armazenado nos **registrants**.

Se nosso servidor web parar de funcionar, perderemos os dados armazenados, então usaremos um banco de dados SQLite com a biblioteca SQL de **cs50**:

```

from cs50 import SQL
from flask import Flask, redirect, render_template, request

app = Flask(__name__)
db = SQL("sqlite:///froshims.db")
...

```

- No terminal do IDE, podemos executar **sqlite3 froshims.db** para abrir o banco de dados e usar o comando **.schema** para ver a tabela com colunas de **id**, **nome** e **sport**, que foi criada antecipadamente.

Agora, em nossas rotas, podemos inserir e selecionar linhas com SQL:

```

@app.route("/register", methods=["POST"])

def register():
 name = request.form.get("name")
 if not name:
 return render_template("error.html", message="Missing name")
 sport = request.form.get("sport")
 if not sport:
 return render_template("error.html", message="Missing sport")
 if sport not in SPORTS:
 return render_template("error.html", message="Invalid sport")

 db.execute("INSERT INTO registrants (name, sport) VALUES(?, ?)",
name, sport)

```

```

 return redirect("/registrants")

@app.route("/registrants")
def registrants():
 registrants = db.execute("SELECT * FROM registrants")
 return render_template("registrants.html",
registrants=registrants)

```

- Depois de validar a solicitação, podemos usar **INSERT INTO** para adicionar uma linha e, da mesma forma, em **registrants()**, podemos **SELECT** todas as linhas e passá-las para o modelo como uma lista de linhas.

Nosso modelo **registrants.html** também precisará ser ajustado, pois cada linha retornada de **db.execute** é um dicionário. Portanto, podemos usar **registrant.name** e **registrant.sport** para acessar o valor de cada chave em cada linha:

```

<tbody>
 {% for registrant in registrants %}
 <tr>
 <td>{{ registrant.name }}</td>
 <td>{{ registrant.sport }}</td>
 <td>
 <form action="/deregister" method="post">
 <input name="id" type="hidden" value="{{
registrant.id }}">
 <input type="submit" value="Deregister">
 </form>
 </td>
 </tr>
 {% endfor %}
</tbody>

```

Podemos até enviar e-mail aos usuários com outra biblioteca, **flask\_mail**:

```

import os
import re

from flask import Flask, render_template, request
from flask_mail import Mail, Message

app = Flask(__name__)
app.config["MAIL_DEFAULT_SENDER"] = os.getenv("MAIL_DEFAULT_SENDER")
app.config["MAIL_PASSWORD"] = os.getenv("MAIL_PASSWORD")
app.config["MAIL_PORT"] = 587
app.config["MAIL_SERVER"] = "smtp.gmail.com"
app.config["MAIL_USE_TLS"] = True

```

```
app.config["MAIL_USERNAME"] = os.getenv("MAIL_USERNAME")
mail = Mail(app)
```

- Definimos algumas variáveis sensíveis fora de nosso código, no ambiente do IDE, para que possamos evitar incluí-las em nosso código.
- Acontece que podemos fornecer detalhes de configuração como nome de usuário e senha e servidor de e-mail, neste caso o do Gmail, para a variável **Mail**, que enviará e-mail por nós.

Finalmente, em nossa rota de **register**, podemos enviar um e-mail para o usuário:

```
@app.route("/register", methods=["POST"])
```

```
def register():
 email = request.form.get("email")
 if not email:
 return render_template("error.html", message="Missing email")
 sport = request.form.get("sport")
 if not sport:
 return render_template("error.html", message="Missing sport")
 if sport not in SPORTS:
 return render_template("error.html", message="Invalid sport")

 message = Message("You are registered!", recipients=[email])
 mail.send(message)

 return render_template("success.html")
```

- Em nosso formulário, também precisaremos solicitar um e-mail em vez de um nome:

```
<input autocomplete="off" name="email" placeholder="Email"
type="email">
```

Agora, se reiniciarmos nosso servidor e usarmos o formulário para fornecer um e-mail, veremos que de fato recebemos um!

## Sessões

As **sessões** são como os servidores da web lembram as informações sobre cada usuário, o que ativa recursos como permitir que os usuários permaneçam logados.

Acontece que os servidores podem enviar outro cabeçalho em uma resposta, chamado **Set-Cookie**:

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=value
```

...

- **Cookies** são pequenos pedaços de dados de um servidor web que o navegador salva para nós. Em muitos casos, eles são grandes números aleatórios ou sequências usadas para identificar e rastrear um usuário de maneira exclusiva entre as visitas.
- Nesse caso, o servidor está pedindo ao nosso navegador para definir um cookie para esse servidor, chamado de **session** para um valor de **value**.

Então, quando o navegador fizer outra solicitação ao mesmo servidor, ele enviará de volta os cookies que o mesmo servidor configurou antes:

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=value
...
```

No mundo real, os parques de diversões podem dar a você um carimbo manual para que você possa voltar depois de sair. Da mesma forma, nosso navegador está apresentando nossos cookies de volta ao servidor da web, para que ele possa se lembrar de quem somos.

As empresas de publicidade podem definir cookies de vários sites, a fim de rastrear os usuários em todos eles. No modo de navegação anônima, por outro lado, o navegador não envia cookies definidos anteriormente.

No Flask, podemos usar a biblioteca **flask\_session** para gerenciar isso para nós:

```
from flask import Flask, redirect, render_template, request, session
from flask_session import Session
```

```
app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)
```

```
@app.route("/")
def index():
 if not session.get("name"):
 return redirect("/login")
 return render_template("index.html")
```

```
@app.route("/login", methods=["GET", "POST"])
def login():
 if request.method == "POST":
 session["name"] = request.form.get("name")
 return redirect("/")
 return render_template("login.html")
```

```
@app.route("/logout")
def logout():
 session["name"] = None
 return redirect("/")
```

Em nosso **login.html**, teremos um formulário com apenas um nome:

- **Cookies** são pequenos pedaços de dados de um servidor web que o navegador salva para nós. Em muitos casos, eles são grandes números aleatórios ou sequências usadas para identificar e rastrear um usuário de maneira exclusiva entre as visitas.
- Nesse caso, o servidor está pedindo ao nosso navegador para definir um cookie para esse servidor, chamado de **session** para um valor de **value**.
  - Configuraremos a biblioteca de sessão para usar o sistema de arquivos do IDE e usaremos a **session** como um dicionário para armazenar o nome de um usuário. Acontece que o Flask usará cookies HTTP para nós, para manter esta variável de **session** para cada usuário visitando nosso servidor web. Cada visitante obterá sua própria variável de **session**, embora pareça ser global em nosso código.
  - Para nossa rota **/** padrão, iremos redirecionar para **/login** se não houver um nome definido na **session** para o usuário ainda, e caso contrário mostraremos um modelo **index.html** padrão.
  - Para nossa rota **/login**, definiremos o nome na session para o valor do formulário enviado via POST e, em seguida, redirecionaremos para a rota padrão. Se visitamos a rota via GET, renderizaremos o formulário de login em **login.html**.
  - Para a rota **/logout**, podemos limpar o valor de **name** na **session** definindo-o como **None** e redirecionar para **/** novamente.
  - Geralmente, também precisaremos de um **requirements.txt** que inclua os nomes das bibliotecas que desejamos usar, para que possam ser instaladas em nosso aplicativo, mas as que usamos aqui foram pré-instaladas no IDE.

Em nosso **login.html**, teremos um formulário com apenas um nome:

```
{% extends "layout.html" %}

{% block body %}

 <form action="/login" method="post">
 <input autocomplete="off" autofocus name="name"
placeholder="Name" type="text">
 <input type="submit" value="Log In">
 </form>

{% endblock %}
```

E em nosso **index.html**, podemos verificar se **session.name** existe e mostrar conteúdo diferente:

```
{% extends "layout.html" %}

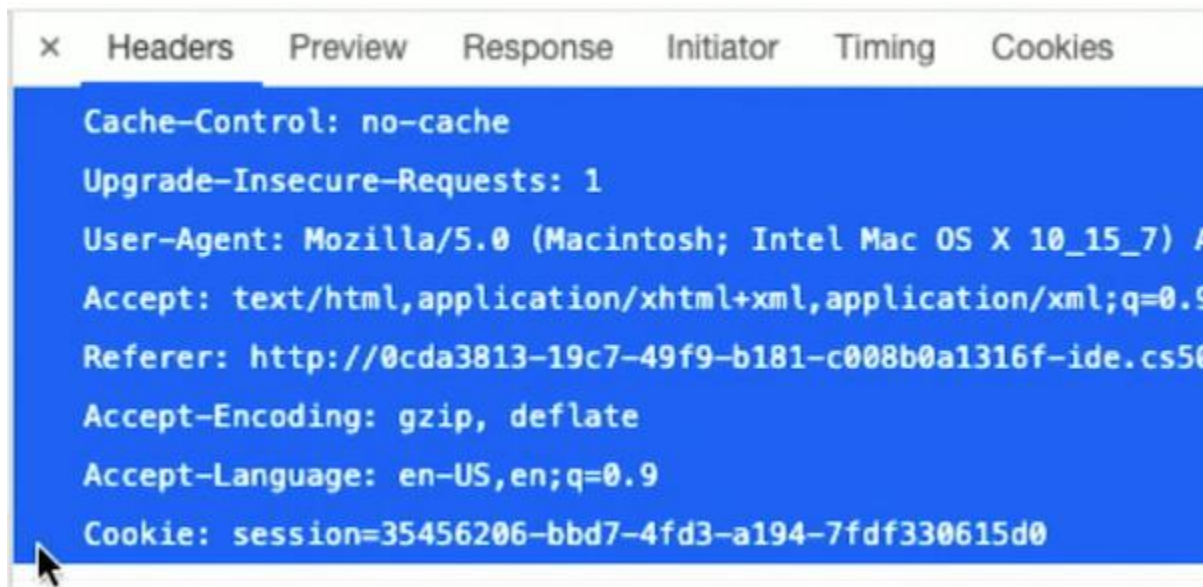
{% block body %}

 {% if session.name %}
 You are logged in as {{ session.name }}. Log out.
```

```
{% else %}
 You are not logged in. Log in.
{% endif %}

{% endblock %}
```

Quando reiniciamos nosso servidor, vamos ao seu URL e logamos, podemos ver na aba Rede que nosso navegador está de fato enviando um cabeçalho Cookie: na solicitação:



## Aplicação Loja(shows)

Veremos um exemplo, [store](#):

- **application.py** inicializa e configura nosso aplicativo para usar um banco de dados e sessões. Em **index()**, a rota padrão renderiza uma lista de livros armazenados no banco de dados.
- **templates/books.html** mostra a lista de books, bem como um formulário que permite clicar em “Adicionar ao carrinho” para cada um deles.
- A rota **cart**, por sua vez, armazena um id de uma solicitação POST na variável de **session** em uma lista. Se a solicitação usasse um método GET, entretanto, **/cart** mostraria uma lista de livros com **id** s correspondendo à lista de ids armazenada na **session**.

Assim, “carrinhos de compras” em sites podem ser implementados com cookies e variáveis de sessão armazenadas no servidor.

Quando visualizamos a fonte gerada por nossa rota padrão, vemos que cada livro tem seu próprio elemento **<form>**, cada um com uma entrada de **id** diferente que é oculta e gerada. Este **id** vem do banco de dados SQLite em nosso servidor e é enviado de volta para a rota **/cart**.

Veremos outro exemplo, **shows**, onde podemos usar JavaScript no **front-end**, ou lado que o usuário vê, e Python no **back-end**, ou lado do servidor.

Em **application.py** aqui, vamos abrir um banco de dados, **shows.db**:

```
from cs50 import SQL
from flask import Flask, render_template, request

app = Flask(__name__)

db = SQL("sqlite:///shows.db")

@app.route("/")
def index():
 return render_template("index.html")

@app.route("/search")
def search():
 shows = db.execute("SELECT * FROM shows WHERE title LIKE ?", "%"
+ request.args.get("q") + "%")
 return render_template("search.html", shows=shows)
```

- A rota padrão / mostrará um formulário, onde podemos digitar algum termo de pesquisa.
- O formulário usará o método GET para enviar a consulta de pesquisa para **/search**, que por sua vez usará o banco de dados para encontrar uma lista de programas correspondentes. Finalmente, um template **search.html** mostrará a lista de programas.

Com JavaScript, podemos mostrar uma lista parcial de resultados à medida que digitamos. Primeiro, usaremos uma função chamada jsonify para retornar nossos programas no formato JSON, um formato padrão que o JavaScript pode usar.

```
@app.route("/search")

def search():
 shows = db.execute("SELECT * FROM shows WHERE title LIKE ?", "%"
+ request.args.get("q") + "%")
 return jsonify(shows)
```

Agora podemos enviar uma consulta de pesquisa e ver se obtemos uma lista de dicionários:

```
[{"id":108878,"title":"Nice
Day at the Office"},
{"id":112108,"title":"The
Office"},
{"id":122441,"title":"Avocat
d'office"},
```

Então, nosso modelo index.html pode converter essa lista em elementos no DOM:

```
<!DOCTYPE html>

<html lang="en">
 <head>
 <meta name="viewport" content="initial-scale=1,
width=device-width">
 <title>shows</title>
 </head>
 <body>

 <input autocomplete="off" autofocus placeholder="Query"
type="search">

 <script crossorigin="anonymous" integrity="sha256-
9/aliU8dGd2tb60SsuzixeV4y/faTqgFtohetphbbj0="
src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
 <script>

 let input = document.querySelector('input');
 input.addEventListener('keyup', function() {
 $.get('/search?q=' + input.value, function(shows) {
 let html = '';
 for (let id in shows)
 {
 let title = shows[id].title;
 html += '' + title + '';
 }
 document.querySelector('ul').innerHTML = html;
 });
 });
 </script>
```



```
</body>
</html>
```

- Usaremos outra biblioteca, JQuery, para fazer solicitações com mais facilidade.
- “Ouviremos” as mudanças no elemento de **input** e usaremos **\$ .get**, que chama uma função de biblioteca JQuery para fazer uma solicitação GET com o valor de entrada. Em seguida, a resposta será passada para uma função anônima por meio da variável **shows**, que definirá o DOM com os elementos **<li>** gerados com base nos dados da resposta.
- **\$ .get** é uma chamada **AJAX**, que permite ao JavaScript fazer solicitações HTTP adicionais após o carregamento da página, para obter mais dados. Se abrirmos a guia Rede novamente, podemos ver de fato que cada tecla pressionada fez outra solicitação, com uma resposta:

Name	×	Headers	Preview	Response	Initiator	Timing	Cookies
<input type="checkbox"/> search?q=of	1			1978,"title":"The Office"),{"id":292829,"title":"Office			
<input type="checkbox"/> search?q=off	2						
<input type="checkbox"/> search?q=offic							
<input type="checkbox"/> search?q=office							

- Como a solicitação de rede pode ser lenta, a função anônima que passamos para **\$ .get** é uma função de **retorno de chamada**, que só é chamada depois de obtermos uma resposta do servidor. Nesse ínterim, o navegador pode executar outro código JavaScript.

## EXERCÍCIOS MÓDULO 9

### Exercício 1: C\$50 Finanças

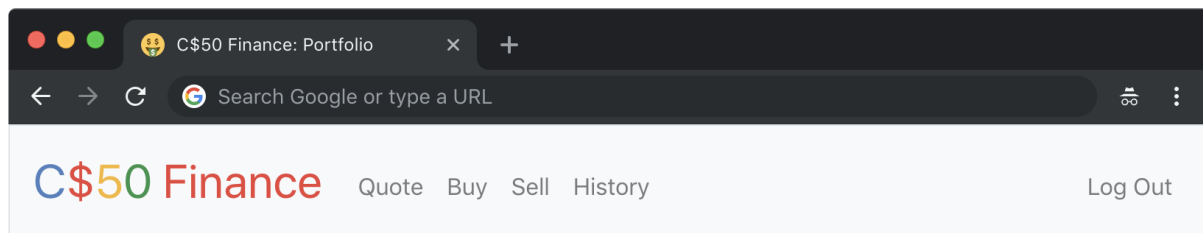
## C\$50 Finanças

```
$ rm helpers.py
```

```
$ wget
```

<https://cdn.cs50.net/2020/fall/psets/9/finance/finance/helpers.py>

Implemente um site por meio do qual os usuários possam “comprar” e “vender” ações, como abaixo:



Symbol	Name	Shares	Price	TOTAL
NFLX	Netflix Inc.	1	\$301.78	\$301.78
CASH				\$9,698.22
				<b>\$10,000.00</b>

Data provided for free by [IEX](#). View [IEX's Terms of Use](#).

## Background

Se você não tiver certeza do que significa comprar e vender ações (no caso, ações de uma empresa), clique [aqui](#) para um tutorial.

Você está prestes a implementar o C\$ 50 Finance, um aplicativo da web por meio do qual é possível gerenciar carteiras de ações. Essa ferramenta não apenas permitirá que você verifique os preços reais das ações e os valores das carteiras, mas também permitirá que você compre (ok, “compre”) e venda (ok, “venda”) ações consultando o IEX sobre os preços das ações.

Na verdade, o IEX permite que você baixe cotações de ações por meio de sua API (interface de programação de aplicativo) usando URLs como [https://cloud.iexapis.com/stable/stock/nflx/quote?token=API\\_KEY](https://cloud.iexapis.com/stable/stock/nflx/quote?token=API_KEY). Observe como o símbolo da Netflix (NFLX) está embutido neste URL; é assim que a IEX sabe quais dados retornar. Esse link não retornará realmente nenhum dado porque o IEX requer que você use uma chave de API (mais sobre isso daqui a pouco), mas se o fizesse, você veria uma resposta no formato JSON (JavaScript Object Notation) como este:

```
{
 "symbol": "NFLX",
 "companyName": "Netflix, Inc.",
 "primaryExchange": "NASDAQ",
 "calculationPrice": "close",
 "open": 317.49,
 "openTime": 1564752600327,
 "close": 318.83,
 "closeTime": 1564776000616,
 "high": 319.41,
```

```
"low": 311.8,
"latestPrice": 318.83,
"latestSource": "Close",
"latestTime": "August 2, 2019",
"latestUpdate": 1564776000616,
"latestVolume": 6232279, "iexRealtimePrice": null,
"iexRealtimeSize": null,
"iexLastUpdated": null,
"delayedPrice": 318.83,
"delayedPriceTime": 1564776000616,
"extendedPrice": 319.37,
"extendedChange": 0.54,
"extendedChangePercent": 0.00169,
"extendedPriceTime": 1564876784244,
"previousClose": 319.5,
"previousVolume": 6563156,
"change": -0.67,
"changePercent": -0.0021,
"volume": 6232279,
"iexMarketPercent": null,
"iexVolume": null,
"avgTotalVolume": 7998833,
"iexBidPrice": null,
"iexBidSize": null,
"iexAskPrice": null,
"iexAskSize": null,
"marketCap": 139594933050,
"peRatio": 120.77,
"week52High": 386.79,
"week52Low": 231.23,
"ytdChange": 0.18907500000000002,
"lastTradeTime": 1564776000616
}
```

Observe como, entre as chaves, há uma lista separada por vírgulas de pares de valores-chave, com dois pontos separando cada chave de seu valor.

## Distribuição

### Baixando

```
$ wget http://cdn.cs50.net/2020/fall/psets/9/finance/finance.zip
$ unzip finance.zip
$ rm finance.zip
$ cd finance
$ ls
```

```
application.py helpers.py static/
finance.db requirements.txt templates/
```

## Configurando

Antes de começar esta atribuição, precisaremos nos registrar para obter uma chave de API para poder consultar os dados do IEX. Para fazer isso, siga estas etapas:

- Visite [iexcloud.io/cloud-login#/register/](https://iexcloud.io/cloud-login#/register/).
- Selecione o tipo de conta “Individual”, digite seu endereço de e-mail e uma senha e clique em “Create account”.
- Depois de registrado, role para baixo até “Get started for free” e clique em “Select Start” para escolher o plano gratuito.
- Depois de confirmar sua conta por e-mail de confirmação, visite <https://iexcloud.io/console/tokens>.
- Copie a chave que aparece na coluna Token (deve começar com **pk\_**).
- Em uma janela de terminal no [CS50 IDE](#), execute:

```
$ export API_KEY=value
```

onde **value** é aquele valor (colado), sem nenhum espaço imediatamente antes ou depois de =. Você também deve colar esse valor em um documento de texto em algum lugar, caso precise dele novamente mais tarde.

## Executando

Inicie o servidor da web integrado do Flask (em `finance/`):

```
$ flask run
```

Visite o URL gerado pelo **flask** para ver o código de distribuição em ação. Porém, você não poderá fazer o login ou registrar-se ainda!

Por meio do navegador de arquivos do CC50, clique duas vezes **finance.db**, a fim de abri-lo com phpLiteAdmin. Observe como **finance.db** vem com uma tabela chamada **users**. Dê uma olhada em sua estrutura (ou seja, schema). Observe como, por padrão, os novos usuários receberão \$10.000 em dinheiro. Mas não há (ainda!) Nenhum usuário (ou seja, linhas) nele para navegar.

Daqui em diante, se você preferir usar a linha de comando, pode usar **sqlite3** em vez de phpLiteAdmin.

## Entendendo o problema...

- **application.py**

Abra `application.py`. No topo do arquivo estão várias importações, entre elas o módulo SQL do CC50 e algumas funções auxiliares. Mais sobre isso em breve.

Depois de configurar o [Flask](#), observe como este arquivo desativa o cache de respostas (desde que você esteja no modo de depuração, que você está por padrão no IDE CS50), para que não faça uma alteração em algum arquivo, mas seu navegador não perceba. Observe a seguir como ele configura o [Jinja](#) com um “filtro” personalizado, **usd**, uma função (definida em

helpers.py) que tornará mais fácil formatar valores como dólares americanos (USD). Em seguida, ele configura o Flask para armazenar [sessões](#) no sistema de arquivos local (ou seja, disco) em vez de armazená-las dentro de cookies (assinados digitalmente), que é o padrão do Flask. O arquivo então configura o módulo SQL do CC50 para usar **finance.db**, um banco de dados SQLite cujo conteúdo veremos em breve!

Depois disso, há um monte de rotas, das quais apenas duas estão totalmente implementadas: **login** e **logout**. Leia a implementação do login primeiro. Observe como ele usa db.execute (da biblioteca do CS50) para consultar finance.db . E observe como ele usa **check\_password\_hash** para comparar hashes de senhas de usuários. Finalmente, observe como o login “lembra” que um usuário está logado armazenando seu **user\_id**, um INTEGER, na **session**. Dessa forma, qualquer uma das rotas desse arquivo pode verificar qual usuário, se houver, está logado. Enquanto isso, observe como o logout simplesmente limpa a session , efetivamente desconectando um usuário.

Observe como a maioria das rotas são “decoradas” com **@login\_required** (uma função definida em helpers.py também). Esse decorador garante que, se um usuário tentar visitar qualquer uma dessas rotas, ele será redirecionado primeiro para fazer o **login**.

Observe também como a maioria das rotas oferece suporte a GET e POST. Mesmo assim, a maioria deles (por enquanto!) Simplesmente retorna um “pedido de desculpas”, uma vez que ainda não foram implementados.

- **helpers.py**

Em seguida, dê uma olhada em helpers.py. Ah, aí está a implementação de **apology**. Observe como, em última análise, ele renderiza um modelo, apology.html. Ele também define dentro de si mesmo outra função, **escape**, que ele simplesmente usa para substituir caracteres especiais em **apology**. Ao definir escape dentro de apology , limitamos o primeiro apenas ao último; nenhuma outra função será capaz (ou precisará) chamá-lo.

O próximo no arquivo é **login\_required**. Não se preocupe se este for um pouco enigmático, mas se você já se perguntou como uma função pode retornar outra função, aqui está um exemplo!

Em seguida, é **lookup**, uma função que, dado um **symbol** (por exemplo, NFLX), retorna uma cotação de ações para uma empresa na forma de um dictionary com três chaves: name , cujo valor é um **str**, o nome da empresa; price , cujo valor é um **float** ; e symbol , cujo valor é str , uma versão canônica (maiúscula) do símbolo de uma ação, independentemente de como esse símbolo foi capitalizado quando passou para o lookup .

O último no arquivo é **usd** , uma função curta que simplesmente formata um valor float como USD (por exemplo, 1234,56 é formatado como \$ 1.234,56 ).

- **requirements.txt**

A seguir, dê uma olhada rápida em requirements.txt. Esse arquivo simplesmente prescreve os pacotes dos quais este aplicativo dependerá.

- **static/**

Dê uma olhada também em static/, dentro do qual está **styles.css**. É aí que residem alguns CSS iniciais. Você pode alterá-lo como achar necessário.

- **templates/**

Agora olhe em templates/. Em **login.html** tem, essencialmente, apenas um formulário HTML, estilizado com [Bootstrap](#). Em **apology.html** , entretanto, esta um modelo para um pedido de desculpas. Lembre-se de que o **apology** em **helpers.py** recebeu dois argumentos: **message** , que foi passada para **render\_template** como o valor de **bottom** e, opcionalmente, **code** , que foi passado para **render\_template** como valor de **top** . Observe em **apology.html** como esses valores são finalmente usados! [E aqui está o porquê 0 :-\)](#)

O último é **layout.html** . É um pouco maior do que o normal, mas principalmente porque vem com uma “navbar” (barra de navegação) sofisticada e otimizada para dispositivos móveis, também baseada no Bootstrap. Observe como ele define um bloco, **main** , dentro do qual os modelos (incluindo **apology.html** e **login.html** ) devem ir. Também inclui suporte para [flash de mensagem](#) do Flask para que você possa retransmitir mensagens de uma rota para outra para o usuário ver.

## Especificação

### **register**

Concluir a implementação do **register** de forma a permitir ao utilizador o registo de uma conta através de um formulário.

- Exigir que um usuário insira um nome de usuário, implementado como um campo de texto cujo name é **username** . Peça desculpas se o input do usuário estiver em branco ou se o nome de usuário já existir.
- Exigir que um usuário insira uma senha, implementada como um campo de texto cujo name é **password** e, em seguida, essa mesma senha novamente, implementada como um campo de texto cujo name é **confirmação** . Peça desculpas se a entrada estiver em branco ou se as senhas não corresponderem.
- Envie a entrada do usuário via **POST** para **/register** .
- **INSERT** o novo usuário em **users** , armazenando um hash da senha do usuário, não a senha em si. Hash a senha do usuário com **generate\_password\_hash** As chances são de você querer criar um novo modelo (por exemplo, **register.html** ) que é bastante semelhante ao **login.html** .
- Depois que o usuário estiver registrado, você pode fazer login automaticamente no usuário ou levá-lo a uma página onde ele mesmo possa fazer o login.

Depois de implementar o **register** corretamente, você deve ser capaz de registrar uma conta e fazer o login (já que o **login** e o **logout** já funcionam)! E você deve ser capaz de ver suas linhas via **sqlite3** ou **phpLiteAdmin**.

### **quote**

Conclua a implementação de **quote** de forma que permita ao usuário consultar o preço atual de uma ação.

- Exigir que um usuário insira um símbolo de ação, implementado como um campo de texto cujo name é **symbol** .
- Envie a entrada do usuário via **POST** para **/quote** .

- Você provavelmente vai querer criar dois novos modelos (por exemplo, quote.html e quoted.html ). Quando um usuário visita /quote via GET, renderiza um desses modelos, dentro do qual deve estar um formulário HTML que é submetido a /quote via POST. Em resposta a um POST, o quote pode renderizar aquele segundo template, incorporando nele um ou mais valores de lookup .

## buy

Concluir a implementação de buy de forma que permita ao usuário comprar ações.

- Exigir que um usuário insira um símbolo de ação, implementado como um campo de texto cujo name é symbol . Peça desculpas se a entrada estiver em branco ou se o símbolo não existir (de acordo com o valor de retorno de lookup ).
- Exigir que um usuário insira um número de ações, implementado como um campo de texto cujo name é shares . Peça desculpas se a entrada não for um número inteiro positivo.
- Envie a entrada do usuário via POST para /buy .
- Provavelmente, você vai querer chamar lookup para consultar o preço atual de uma ação.
- Provavelmente, você vai querer SELECT quanto dinheiro o usuário tem atualmente em users .
- Adicione uma ou mais novas tabelas a finance.db por meio das quais você pode controlar a compra. Armazene informações suficientes para saber quem comprou o quê, a que preço e quando.
  - Use os tipos SQLite apropriados.
  - Defina índices UNIQUE (unicos) em quaisquer campos que devam ser exclusivos.
  - Defina índices (não-UNIQUE ) em quaisquer campos pelos quais você fará a pesquisa (como por meio de SELECT com WHERE ).
- Peça desculpas, sem concluir a compra, se o usuário não puder pagar o número de ações no preço atual.
- Quando a compra for concluída, redirecione o usuário de volta à página de index .
- Você não precisa se preocupar com as condições de corrida (ou usar transações).

Depois de implementar buy corretamente, você poderá ver as compras dos usuários em sua(s) nova(s) tabela(s) via **sqlite3** ou phpLiteAdmin.

## index

Conclua a implementação de index de forma que exiba uma tabela HTML resumindo, para o usuário atualmente conectado, quais ações o usuário possui, o número de ações que possui, o preço atual de cada ação e o valor total de cada ação (ou seja, ações vezes preço). Também exibe o saldo de caixa atual do usuário junto com um total geral (ou seja, o valor total das ações mais dinheiro).

- É provável que você queira executar vários SELECT s. Dependendo de como você implementar sua (s) tabela (s), você pode achar GROUP BY HAVING SUM e / ou WHERE de interesse.
- Provavelmente, você deseja chamar lookup para cada ação.

## sell

Conclua a implementação de sell de tal forma que permita a um usuário vender ações (que ele ou ela possui).

- Exigir que um usuário insira um símbolo de ação, implementado como um menu de select cujo name é symbol . Peça desculpas se o usuário deixar de selecionar uma ação ou se (de alguma forma, uma vez enviada) o usuário não possui nenhuma ação dessa empresa.
- Exigir que um usuário insira um número de ações, implementado como um campo de texto cujo name é shares . Peça desculpas se a entrada não for um número inteiro positivo ou se o usuário não possuir tantas ações do estoque.
- Envie a entrada do usuário via **POST** para **/sell**.
- Quando a venda for concluída, redirecione o usuário de volta à página de index .
- Você não precisa se preocupar com as condições de corrida (ou usar transações).

## history

Conclua a implementação do histórico de tal forma que exiba uma tabela HTML resumindo todas as transações de um usuário, listando linha por linha cada compra e cada venda.

- Para cada linha, deixe claro se uma ação foi comprada ou vendida e inclua o símbolo da ação, o preço (de compra ou venda), o número de ações compradas ou vendidas e a data e hora em que a transação ocorreu.
- Pode ser necessário alterar a tabela criada para buy ou complementá-la com uma tabela adicional. Tente minimizar redundâncias.

## toque pessoal

Implemente pelo menos um toque pessoal de sua escolha:

- Permitir que os usuários alterem suas senhas.
- Permitir que os usuários adicionem dinheiro em suas contas.
- Permite que os usuários comprem mais ações ou vendam ações que já possuem através do próprio index , sem ter que digitar os símbolos das ações manualmente.
- Exigir que as senhas dos usuários tenham um certo número de letras, números e/ou símbolos.
- Implemente algum outro recurso de escopo comparável.

# Testando

Para testar seu código com **check50**, execute o seguinte.

```
$ check50 cs50/problems/2021/x/finance
```

Esteja ciente de que check50 testará todo o seu programa como um todo. Se você executá-lo antes de concluir todas as funções necessárias, ele pode relatar erros em funções que estão realmente corretas, mas dependem de outras funções.

Certifique-se de testar seu aplicativo da web manualmente também, como

- inserir sequências alfabéticas em formulários quando apenas números são esperados,
- inserir zero ou números negativos em formulários quando apenas números positivos são esperados,



- inserir valores de ponto flutuante em formulários quando apenas inteiros são esperados,
- tentar gastar mais dinheiro do que o usuário tem,
- tentar vender mais ações do que o usuário possui,
- inserir um símbolo de ação inválido e
- incluir personagens potencialmente perigosos como ' e ; em consultas SQL.

Execute o seguinte para avaliar o estilo de seus arquivos Python usando **style50**.

```
style50 *.py
```

## Solução da equipe

Você pode estilizar seu próprio aplicativo de maneira diferente, mas aqui está a aparência visual da solução da equipe!

<https://finance.cs50.net/>

Sinta-se à vontade para registrar uma conta e brincar. Você **não deve** usar uma senha que você usa em outros sites.

É **razoável** olhar para o HTML e CSS da equipe.

## Dicas

- Para formatar um valor como um valor em dólares americanos (com centavos listados em duas casas decimais), você pode usar o filtro `usd` em seus modelos Jinja (imprimindo valores como `{{ value | usd }}` em vez de `{{ value }}`).
- Dentro de `cs50.SQL` está um método de `execute` cujo primeiro argumento deve ser um `str` de SQL. Se esse `str` contiver parâmetros de ponto de interrogação aos quais os valores devem ser associados, esses valores podem ser fornecidos como parâmetros nomeados adicionais para `execute`. Veja a implementação de login para um exemplo. O valor de retorno de `execute` é o seguinte:
  - Se **str** for um **SELECT**, `execute` retorna uma list de zero ou mais objetos dict, dentro dos quais estão as chaves e os valores que representam os campos e células de uma tabela, respectivamente.
  - Se **str** for um **INSERT**, e a tabela na qual os dados foram inseridos contiver uma **PRIMARY KEY** de autoincremento, `execute` retorna o valor da chave primária da linha recém-inserida.
  - Se **str** for **DELETE** ou **UPDATE**, `execute` retorna o número de linhas excluídas ou atualizadas por **str**.
- Lembre-se de que o `cs50.SQL` registrará em sua janela de terminal todas as consultas que você executar por meio de `execute` (para que você possa confirmar se estão conforme o pretendido).
- Certifique-se de usar parâmetros vinculados a pontos de interrogação (ou seja, um [estilo de parâmetro](#) de **named**) ao chamar o método `execute` do `CS50`, como por exemplo **WHERE?**. Você **não deve** usar f-strings, `format` ou `+` (ou seja, de concatenação), para que não corra o risco de um ataque de injeção SQL.
- Se (e somente se) já estiver familiarizado com SQL, você pode usar [SQLAlchemy Core](#) ou [Flask-SQLAlchemy](#) (ou seja, [SQLAlchemy ORM](#)) em vez de `cs50.SQL`.

- Você pode adicionar outros arquivos estáticos a `static/`.
- Provavelmente, você vai querer consultar a [documentação do Jinja](#) ao implementar seus modelos.
- É razoável pedir a outras pessoas que testem (e tentem achar erros) em seu site.
- Você pode alterar a estética dos sites, via
  - <https://bootswatch.com/>,
  - <https://getbootstrap.com/docs/4.1/content/>,
  - <https://getbootstrap.com/docs/4.1/components/> e/ou <https://memegen.link/>.
- Você pode achar a [documentação do Flask](#) e a [documentação do Jinja](#) úteis!

## FAQs

### ImportError: No module named ‘application’

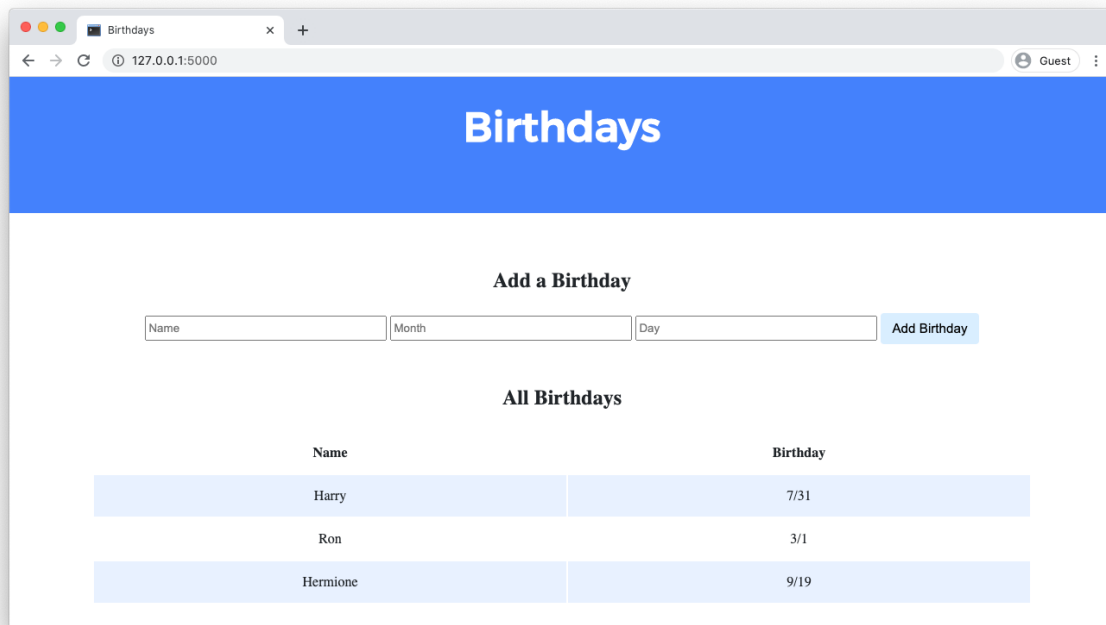
Por padrão, o **flask** procura um arquivo chamado **application.py** em seu diretório de trabalho atual (porque configuramos o valor de **FLASK\_APP**, uma variável de ambiente, como **application.py**). Se você se deparar com esse erro, provavelmente executou o **flask** no diretório errado!

### OSError: [Errno 98] Address already in use

Se, ao executar o **flask**, você se deparar com este erro, é provável que você (ainda) tenha o **flask** rodando em outra guia. Certifique-se de encerrar aquele outro processo, como com `ctrl-c`, antes de iniciar o **flask** novamente. Se você não tiver nenhuma outra guia, execute **fuser -k 8080 /tcp** para encerrar todos os processos que estão (ainda) escutando na porta TCP 8080.

## Lab 9: Birthdays

Crie um aplicativo da web para controlar os aniversários de amigos.



## Começando

Veja como baixar este laboratório em seu próprio CS50 IDE. Faça login no [CS50 IDE](#) e, em uma janela de terminal, execute cada um dos itens abaixo.

- Execute `cd` para garantir que você está em `~/` (ou seja, seu diretório pessoal, também conhecido como `~`).
- Execute `https://cdn.cs50.net/2020/fall/labs/9/lab9.zip` para baixar um arquivo ZIP (compactado) com a distribuição desse problema.
- Execute `unzip lab9.zip` para descompactar esse arquivo.
- Execute `rm lab9.zip` seguido por **yes** ou **y** para excluir o arquivo ZIP.
- Execute `ls`. Você deve ver um diretório chamado **lab9**, que estava dentro desse arquivo ZIP.
- Execute `cd lab9` para mudar para esse diretório.
- Execute `ls`. Você deve ver um arquivo **application.py**, um arquivo **birthdays.db**, um diretório **static** e um diretório de **templates**.

## Entendendo o problema

Em **application.py**, você encontrará o início de um aplicativo da web Flask. O aplicativo tem uma rota (`/`) que aceita solicitações **POST** (após o **if**) e solicitações **GET** (após o **else**). Atualmente, quando a rota `/` é solicitada via **GET**, o modelo **index.html** é renderizado. Quando a rota `/` é solicitada via **POST**, o usuário é redirecionado de volta para `/` via **GET**.

**birthdays.db** é um banco de dados SQLite com uma tabela, **birthdays**, que possui quatro colunas: **id**, **name**, **month** e **day**. Já existem algumas linhas nesta tabela, embora, em última análise, seu aplicativo da web suporte a capacidade de inserir linhas nesta tabela!

No diretório **static** está um arquivo **styles.css** contendo o código CSS para este aplicativo da web. Não há necessidade de editar este arquivo, embora seja bem-vindo, se desejar!

No diretório de **templates** está um arquivo **index.html** que será renderizado quando o usuário visualizar seu aplicativo da web.

## Detalhes de implementação

Conclua a implementação de um aplicativo da web para permitir que os usuários armazenem e acompanhem os aniversários.

- Quando a rota `/` for solicitada via **GET**, sua aplicação web deverá mostrar, em uma tabela, todas as pessoas em seu banco de dados junto com seus aniversários.
  - Primeiro, em `application.py`, adicione lógica em seu tratamento de solicitação GET para consultar o banco de dados **birthdays.db** para todos os aniversários. Passe todos esses dados para o seu modelo `index.html`.
  - Em seguida, em `index.html`, adicione lógica para processar cada aniversário como uma linha na tabela. Cada linha deve ter duas colunas: uma coluna para o nome da pessoa e outra coluna para o aniversário da pessoa.
- Quando a rota `/` é solicitada via **POST**, seu aplicativo da web deve adicionar um novo aniversário ao seu banco de dados e, em seguida, renderizar novamente a página de índice.
  - Primeiro, em `index.html`, adicione um formulário HTML. O formulário deve permitir que os usuários digitem um nome, um mês de aniversário e um dia de aniversário. Certifique-se de que o formulário seja submetido a `/` (sua “ação”) com um método de `post`.
  - Em seguida, em `application.py`, adicione lógica em seu tratamento de solicitação POST para **INSERT** uma nova linha na tabela de `birthdays` com base nos dados fornecidos pelo usuário.

Opcionalmente, você também pode:

- Adicione a capacidade de excluir e `/` ou editar entradas de aniversário.
- Adicione quaisquer recursos adicionais de sua escolha!

## Dicas

- Lembre-se de que você pode chamar **db.execute** para executar consultas SQL em `application.py`.
  - Se você chamar **db.execute** para executar uma consulta **SELECT**, lembre-se de que a função retornará para você uma lista de dicionários, onde cada dicionário representa uma linha retornada por sua consulta.
- Você provavelmente achará útil passar dados adicionais para `render_template()` em sua função de `index` para acessar os dados de aniversário dentro de seu modelo `index.html`.

- Lembre-se de que a tag **tr** pode ser usada para criar uma linha da tabela e a tag **td** pode ser usada para criar uma célula de dados da tabela.
- Lembre-se de que, com o Jinja, você pode criar um [loop for](#) dentro do seu arquivo `index.html`.
- Em `application.py`, você pode obter os dados POST ados pelo envio do formulário do usuário por meio de `request.form.get(field)`, onde `field` é uma string que representa o atributo de name de um input de seu formulário.
  - Por exemplo, se em `index.html`, você tinha um `<input name="foo" type="text">`, você poderia usar `request.form.get("foo")` em `application.py` para extrair a entrada do usuário.

Não sabe como resolver?

## Testando

Sem **check50** para este laboratório! Mas certifique-se de testar seu aplicativo da web adicionando alguns aniversários e garantindo que os dados apareçam em sua tabela conforme o esperado. Execute o `flask run` em seu terminal enquanto está no diretório `lab9` para iniciar um servidor da web que atende seu aplicativo Flask.

## ANOTAÇÕES MÓDULO 10

### O Fim

O CS50 agradece ao [American Repertory Theatre](#) por sediar as palestras neste semestre, fornecendo adereços, iluminação e sons incríveis para o palco. A equipe do CS50 também tem feito tudo o que é possível fora do palco, incluindo os professores e assistentes do curso. E até mesmo David comete erros e pode não ter certeza sobre as respostas para algumas perguntas, então saiba que o **aprendizado continuará muito além deste curso**.

Não se esqueça de que, em última análise:

O que importa neste curso não é tanto onde você termina em relação aos seus colegas, mas onde você termina em relação a si mesmo quando você começou.

Aprendemos alguns princípios básicos:

- pensamento computacional
- usar algoritmos para resolver problemas, dadas alguns inputs para produzir outputs
- eixos de correção, design e estilo para avaliar nosso código
- abstração, usando camadas para simplificar os problemas, como acontece com as funções no código
- precisão, considerando todos os casos extremos possíveis em nossas instruções

Com esses blocos de construção básicos, podemos aprender a usar ferramentas no futuro, além de C e Python, para resolver ainda mais problemas.

Pedimos a um voluntário para dar uma série de instruções sobre como desenhar um cubo e um boneco de neve e, como todos os outros interpretaram cada instrução de maneira um pouco diferente, os desenhos finais acabaram todos muito diferentes.

## Ética

Podemos pensar em ética em termos de se devemos fazer algo ou de que maneira, mesmo quando temos a capacidade de fazê-lo.

Por exemplo, agora podemos usar código para enviar muitos e-mails, criando mais spam. Podemos coletar senhas para nosso site com um formulário e, se um usuário usar a mesma senha para outro site, acabamos tendo acesso à sua conta, a menos que armazenamos as senhas de forma segura.

JavaScript também pode ser usado para registrar as ações dos usuários em nossos sites, como itens que eles adicionaram ao carrinho. Mas registrar todas as ações, ao longo do tempo, pode causar preocupação com a privacidade dos usuários.

Antes do Facebook, o site, havia outro site, o [Facemash](#), onde o código era usado para raspar, ou baixar, imagens de alunos de Harvard e usá-las sem permissão prévia.

Com alguns colegas do Departamento de Filosofia, Meica Magnani e Susan Kennedy, discutimos algumas estruturas para a tomada de decisões com mais rigor.

O programa [Embedded EthiCS](#) em Harvard integra ferramentas de raciocínio ético em cursos de ciência da computação, para ajudar a garantir que os futuros cientistas da computação criarão e usarão a tecnologia de forma ética.

A transcrição desta seção da aula, bem como as leituras do laboratório relacionado, foram publicadas separadamente.

## Próximos Passos

Mesmo sem cursos adicionais em ciência da computação, esperamos que agora você esteja preparado para usar a tecnologia para resolver problemas em seu próprio domínio.

Quando enfrentamos novos problemas, podemos contar com uma ou mais das seguintes habilidades:

- fazer perguntas
- encontrar respostas
- ler documentação
- aprender novas línguas para si mesmo