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

Pset 1: C

a ser entregue até: 19:00, sex 02/03

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

Objetivos:

- Torná-lo mais confortável com Linux.
- Começar a pensar mais lógica e metodicamente.
- Resolver alguns problemas usando C.

Leitura recomendada:

- Seções 1 a 7, 9 e 10 de `http://informatica.hsw.uol.com.br/programacao-em-c.htm`

Honestidade Acadêmica.

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

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

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

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

Todas as formas de desonestidade acadêmica são tratadas com rigor.

Licença.

Notas:

Seu trabalho neste Pset será avaliado em três quesitos principais:

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

Introdução.

☑ Participe do CC50.

☐ Está na hora de instalar a sua cópia do Ubuntu Linux no qual você vai em breve escrever e compilar o seu próprio código (talvez pela primeira vez!).[1,] A partir de agora (quase) tudo o que você fizer para o CC50 deverá ser feito no seu Linux, então se acostume!

Agora, para que você fique um pouco perdido, lhe mostraremos duas (ou três) opções de instalação do Ubuntu Linux diferentes. Mas não se preocupe! Continue lendo.
Agora, para que você **fique** um pouco perdido, lhe mostraremos três opções de instalação do Ubuntu diferentes. Mas não se preocupe! Continue lendo.

☐ Primeiro você pode escolher baixar o Linux e instalá-lo com o Instalador do Windows (Wubi), assim o Ubuntu ficará instalado ao lado do seu Sistema Operacional padrão e sempre que o seu computador for iniciado você pode escolher qual dos Sistemas Operacionais você vai querer utilizar. Essa é a opção mais rápida e fácil, e, por acaso, é o método que eu uso.[2]

A segunda opção é instalar o Ubuntu normalmente, criando uma nova partição do seu HD, essa é a opção mais tradicional e o funcionamento é parecido com o da opção anterior, mas ela não é tão rápida.

Por último você pode rodar o Linux em uma Máquina Virtual, que é como se fosse uma janela dentro do seu Sistema Operacional padrão que, por acaso, roda Linux! Esse método pode ser muitas vezes mais prático mas, como você utiliza dois Sistemas Operacionais verdadeiramente ao mesmo tempo, o Linux não vai poder trabalhar utilizando 100% da capacidade do seu computador.

Bem, para as finalidades desse curso 90% da capacidade do seu computador é mais do que o suficiente e qualquer um dos três métodos vai funcionar muito bem, então a escolha é sua. De qualquer forma, se você tiver qualquer problema na instalação do Linux não hesite em mandar um email para `ajuda@cc50.com.br`.

Instalação através do Instalador do Windows – Wubi

☐ Se você quiser instalar o Ubuntu pelo método de particionamento tradicional ou em uma Máquina Virtual, rodando em uma janela do seu computador, pule essa seção.

☐ Siga para

`http://www.ubuntu.com/download/ubuntu/windows-installer`

e baixe o instalador para Windows. Essa é basicamente uma instalação de programa normal, o instalador vai baixar e instalar o Ubuntu, você só deve escolher em qual disco (C: ou D: ou algo assim), o idioma, o tamanho ocupado pela sua instalação, o seu nome de usuário e senha. Siga

[1] Se você já tiver uma cópia do Linux Ubuntu, tudo bem, pode pular essa parte!
[2] Usuários que não tem Windows tem que utilizar o método tradicional de particionamento.

todos os passos e no fim reinicie o computador, escolhendo Linux – Ubuntu na inicialização. Pronto, terminou! Muitos dizem que Linux é complicado. Foi difícil?!

Instalação normal

☐ Se você já instalou o Ubuntu através do Wubi ou quer instalá-lo em uma Máquina Virtual, pule essa seção.

☐ Dirija-se agora ao site

`http://www.ubuntu.com/download/ubuntu/download`

e siga as instruções indicadas para gravar um CD de instalação a partir do arquivo ISO. Após ter gravado o CD, insira-o no seu computador, e reinicialize-o, seguindo então as instruções da instalação. O próprio instalador vai criar uma nova partição do HD para você, só tome cuidado para não escolher a opção de "Substituir o seu Sistema Operacional atual pelo Ubuntu" para não perder todos os arquivos do seu Sistema Operacional, afinal de contas você quer ter os dois ao mesmo tempo.

Instalação através de uma Máquina Virtual

☐ Se você já instalou o Ubuntu, pule essa seção.

☐ Você precisará do programa Virtual Box para utilizar uma máquina virtual. Vá para a URL

`http://www.virtualbox.org/wiki/Downloads`

e baixe o Virtual Box. Você também precisará do arquivo ISO do Ubuntu que pode ser baixado em

`http://www.ubuntu.com/download/ubuntu/download`

Após baixar os dois vá ao seu Virtual Box e clique em **Machine → New**. Siga as instruções para criar uma nova máquina virtual utilizando o arquivo ISO que você baixou do Ubuntu. Se tiver dificuldades tente dar uma olhada no tutorial

`http://jmmwrite.files.wordpress.com/2008/11/virtual_box_install.pdf`

Está em inglês, mas tem imagens que podem ser úteis.

☐ Depois disso sempre que você quiser acessar o seu Linux, entre no Virtual Box (provavelmente através da sua área de trabalho) e clique duas vezes no ícone da máquina virtual que você criou.

Dentro do Linux!

☐ Tudo bem, pronto para brincar um pouco com a sua nova cópia do Ubuntu Linux? Vá em frente e dê uma olhada, mexa um pouco, comece a se acostumar. Nós sabemos que o Ubuntu tem uma GUI (*Graphical User Interface* - a possibilidade de usar o mouse e clicar para realizar a maioria das

funções) muito atraente, assim como o Windows ou o Mac, mas na maior parte desse curso você irá utilizar a Interface de Comando para fazer quase tudo. Isso mesmo, aquela tela preta entediante onde você só escreve e não acontece muita coisa além disso, mas se você quiser pode até mudar a cor da tela de preto para, digamos, azul ou vermelho![3]

Como chegar na tela preta então?! Após brincar um pouco com a GUI do Ubuntu, você deve clicar em **Acessórios → Terminal** e pronto! Você está na Interface de Comando, cujo "prompt", onde em breve você vai estar digitando comandos, se parece com:

```
username@computer (~):
```

O til `(~)` significa que você está atualmente dentro do seu diretório home. Essa linha de comando também é chamada de "shell". Uma shell é apenas uma intérprete de comandos. Você digita alguma coisa e ela faz alguma coisa. Por exemplo, vá em frente e digite

```
echo oi
```

seguido de Enter. Legal! A shell só disse oi de volta. Acontece que `o echo` é um programa (que alguém escreveu há muito tempo) que leva um "argumento da linha de comando" e simplesmente ecoa-o de volta. Em breve você vai aprender a escrever programas muito mais interessantes!

☐ Vamos agora criar um diretório para o CC50, onde todo o código que você escrever vai ficar em breve. Vá em frente e execute o comando abaixo (digitando-o e depois pressionando Enter):[4]

```
mkdir ~/cc50
```

mkdir significa "**m**a**k**e **dir**ectory" ou "criar diretório" em inglês, quase faz sentido, não?! Lembre-se que `~` denota o seu diretório home, assim o código acima significa "fazer um diretório chamado `cc50` dentro do meu diretório home."

Agora abra o diretório novo (ou, visto de outra forma, altere o "diretório de trabalho atual" para o novo diretório), executando este comando:

```
cd ~/cc50
```

cd significa "**c**hange **d**irectory" ou "mudar diretório". Parece que esses comandos são até intuitívos não é?!

Aviso! Como o seu prompt apenas mudou para

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

agora você está dentro do seu diretório `cc50`. Vale a pena mencionar que você poderia ter apenas executado

[3] Hmm... Na verdade nós recomendamos preto para não cansar a sua vista. Aquilo sobre azul ou vermelho foi meio que brincadeira.

[4] Não esqueça do espaço entre o *mkdir* e o ~.

```
cd cc50
```

um momento atrás, porque `cd` simplesmente assume que `cc50` deve estar dentro de qualquer diretório que você estiver. Quer tentar algo mais? Vá em frente e execute

```
cd
```

sem nenhum argumento, como é dito. Você deve encontrar-se novamente em seu diretório home, como indicado pelo ~ dentro dos parenteses:

```
username@computer (~):
```

Na verdade, se você alguma vez se perder dentro dos seus diretórios, basta executar `cd` sem nenhum argumento, e você será levado de volta ao "home, sweet home".[5] Em todo caso execute

```
cd cc50
```

sem til. Você deve encontrar-se dentro de `~/cc50`.

Faz sentido? Tudo o que você está fazendo, na verdade, é abrir e fechar pastas, ainda que através de uma interface de linha de comando (ao contrário de uma GUI - Gráfica).

Aliás, til é apenas uma notação abreviada para o caminho do seu diretório home. Vá em frente e crie um novo diretório chamado *pset1* para este Pset, dentro de *cc50.* Se você já não lembra mais como fazer isso, sinta-se a vontade para dar uma olhada na página anterior.

Quando você se encontrar dentro do novo diretório criado execute:

```
pwd
```

que significa "print working directory", ou "mostrar diretório atual":

```
/home/cc50/pset1
```

Se você não tiver visto essa linha, provavelmente fez algo errado nos últimos passos. Volte e revise.

Em um Linux, o primeiro `/` denota a "raiz" do disco rígido. E assim, a implicação aqui é que há um diretório chamado `home` na raiz do seu disco rígido, dentro do qual o seu diretório `pset1`, que você acabou de criar se encontra. Ok, vamos para casa. Você poderia simplesmente executar `cd`, mas vamos lhe ensinar outro caminho para home agora. Vá em frente e execute:

```
cd ..
```

Lembre-se da aula, que um único ponto `(.)` representa o seu diretório de trabalho atual. Acontece que "dot dot" `(..)` representa "diretório pai" do seu diretório de trabalho atual (o

[5] Muito sentimental? Pensei que poderia ser.

diretório anterior). E assim, se você está em `~/cc50/`pset1, mudando seu diretório de trabalho atual para `..` leva você de volta ao `~/cc50`. Em outras palavras, se você executar agora

```
pwd
```

você deve ver que você está realmente de volta a `/home/`*`cc50`* agora digite

cd ..

novamente e vá para casa.[6] Faz sentido?

Ok, que tal "dot dot dot" `(...)?`O que será que isso provavelmente significa?[7]

Tudo bem, verifique se você está realmente no seu diretório home. Vamos agora dar uma rápida olhada ao redor. Vá em frente e execute este comando:[8]

```
ls
```

Esse comando vai listar o conteúdo do (ou os arquivos e diretórios no) seu diretório de trabalho atual. Bem, a única coisa que você criou em home até agora é o diretório `cc50`, então, presumivelmente, você só deve ver `cc50`. No entanto, vá em frente e reexecute `ls`, desta vez com um "switch" (um argumento de linha de comando que influencia o comportamento de um programa):

```
ls -a
```

O -a implica "all", “todos”, e por isso esta invocação de `ls` listas absolutamente tudo em seu diretório home, incluindo "dotfiles", arquivos e diretórios que normalmente são ocultos. Não se importe com os dotfiles agora, mas saiba que eles existem, eles são geralmente arquivos de configuração.

Qualquer coisa que você pode fazer normalmente no computador pode ser feita através do prompt de comando. Por exemplo, se você gostaria de mudar a senha atual do seu computador, vá em frente e execute:[9]

```
passwd
```

A sua senha antiga será solicitada. E depois a nova duas vezes, não se esqueça de digitar duas vezes a mesma coisa.

Ok, precisa de uma pausa curta? Vá em frente e execute o comando abaixo:

```
exit
```

---

[6] Se algo deu errado simplesmente execute cd e você estará no seu diretório home.

[7] Na, nada. :-)

[8] Note que `l` é um L minúsculo.

[9] passwd não tem o “o” e o “r” de “password” (senha). Porquê? É mais fácil de digitar :-)

Uma outra forma é simplesmente pressionar Ctrl-D. Você acabou de sair do Terminal. É sempre melhor sair desta forma antes de desligar ou colocar o computador para dormir.

O hai, Nano!

☐ Voltou tão cedo? Ok, vá em frente e ligue o seu Terminal, se você ainda não está nele. (Lembra-se como?) Em seguida, navegue para `~/cc50/`pset1.(Lembra-se como?) Uma vez lá, execute o comando abaixo:

```
nano hello.c
```

Lembre-se que `nano` é um editor de texto (relativamente) simples com o qual você pode escrever código (ou qualquer texto, na verdade). Prossiga para escrever sua própria versão de "Hello World" Basta redigitar, quase caractere por caractere, o programa da semana 1 `hello.c`, mas, pelo menos, substitua "O hai, world!" pelo seu próprio argumento de `printf`.

```
#include <stdio.h>

int
main(void)
{
    printf("O hai, world!\n");
}
```

Uma vez pronto, pressione Ctrl-x para salvar, seguido de S - Sim (ou Y - Yes) e Enter, e você deve voltar ao seu prompt. Prossiga para executar o comando abaixo.

```
gcc hello.c
```

Se você não cometeu erros, você deve apenas ver outro prompt. Se você tiver cometido algum erro, você vai ver um ou mais alertas e / ou mensagens de erro. Mesmo que essas mensagens sejam enigmáticas, pense sobre o que elas podem significar, em seguida, vá encontrar o(s) seu(s) erro(s)! Para editar `hello.c`, `reexecute` `nano` como antes. Uma vez que seu código está correto e `hello.c` compila com sucesso, procure o seu programa em seu diretório de trabalho atual, digitando o seguinte comando.

```
ls
```

Você deverá ver uma resposta semelhante à abaixo.

```
a.out hello.c
```

Na verdade, alguns detalhes a mais seriam bons. Vá em frente e execute o comando abaixo

```
ls -l
```

Mais do que apenas listar o conteúdo de seu diretório de trabalho atual, este comando lista os tamanhos, datas e horários da criação, e muito mais. A resposta que você vê deve ser semelhante à abaixo.[10]

```
-rwx------ 1 username username 4647 2010-09-10 19:01 a.out
-rw------- 1 username username   66 2010-09-10 19:01 hello.c
```

Acontece `que -l` é um outro switch que controla o comportamento de `ls`. Para procurar mais switches para `ls` no manual do seu Linux, execute o comando abaixo.

```
man ls
```

Você pode andar para cima e para baixo neste manual usando as setas do teclado e a barra de espaço. Em geral, a qualquer momento que você quiser obter mais informações sobre algum comando, tente verificar sua "man page" ou “página do manual” pela execução de `man` seguida pelo nome do comando! Vamos agora confirmar que seu programa funciona. Execute o comando abaixo.

```
./a.out
```

Você deverá ver a sua saudação. Lembre-se que `.` denota o seu diretório de trabalho atual, e assim este comando significa "executar o programa chamado `a.out` no meu diretório de trabalho atual."

Antes de prosseguirmos, vamos dar ao seu programa um nome mais interessante do que `a.out`. Vá em frente e execute o seguinte comando.

```
gcc -o hello hello.c
```

Neste caso, `-o` é mais um switch para `gcc`. O efeito dessa opção é nomear o programa de saída compilado pelo gcc como `hello` em vez de `a.out`. Vamos agora nos livrar da primeira compilação. Para excluir `a.out`, execute o seguinte comando.

```
rm a.out
```

`rm` significa “remove” ou “remover”. Se solicitada a confirmação, aperte y (“yes”) seguido de Enter.

Bem-vindo a Linux e C!

Hora da historinha.

☐ Nós vimos nas aulas como os discos rígidos (e disquetes) funcionam, mas os computadores tem, na verdade, alguns outros tipos de memória *(*armazenamento), entre eles o cache nível 1, o cache nível 2, a RAM e a ROM. Dê uma olhada no artigo abaixo para saber um pouco sobre cada um:

`http://informatica.hsw.uol.com.br/memoria-do-computador.htm`

[10] Não se preocupe se os números e nomes que você vê são um pouco diferentes dos nossos.

É provável que você vai querer ler, pelo menos, as páginas de 1 a 5 desse artigo.

É isso por enquanto. Mas pode apostar que esse tópico vem de novo!

☐ Lembre-se que "estilo" geralmente se refere à estética do “source code” (código fonte), o quão legível por humanos é o código*(*comentado e indentado, com variáveis apropriadamente nomeadas). Provavelmente você não teve que pensar muito no estilo ao escrever `hello.c`, pois esse é um programa bem pequeno, mas você está prestes a começar a escrever programas nos quais você precisará tomar algumas decisões estilísticas.

Antes que você continue, leia o Guia de Estilo do CS50 (em Inglês):

`https://manual.cs50.net/Style_Guide`

Estilo é, de certa forma, uma questão de preferência pessoal, então o CC50 não obriga você a imitar os estilos que você vê nas aulas e no guia. Mas nós esperamos que você modele o seu próprio estilo, segundo as convenções comuns. Você verá que o Guia de Estilo do CS50 apresenta algumas dessas convenções. Mantenha-as em mente quando começar a escrever o seu código!

Makefile.

☐ Estamos quase chegando lá! Para compilar nosso código de forma mais simples utilizaremos as chamadas Makefiles. Bem, simplesmente copie o arquivo chamado Makefile que veio junto com esse documento para dentro do diretório pset1 que você criou.

Biblioteca do CC50.

☐ Ufa! Finalmente o último preparativo antes de realmente começarmos a programar!

Para realizar a maior parte dos Psets, você vai precisar da biblioteca do CC50. Após instalá-la em seu computador uma vez, você não precisa fazer de novo então vamos lá!

Simplesmente entre no site do CC50, na seção **Software** e baixe o I**nstalador da Biblioteca do CC50,** tenha certeza de baixar o instalador e não a biblioteca em si. Você também pode simplesmente ir para a URL abaixo:

`http://cc50.com.br/downloads/resources/install_cc50_c_library.zip`

Após descompactar o instalador, clique duas vezes no arquivo **install_cc50_library.command** e escolha a opção “executar no Terminal”. Se você fez tudo certo e não ocorreu nenhum erro, a biblioteca do CC50 foi instalada no seu computador!

A partir de agora, sempre que um programa seu precisar pedir um input do usuário você precisará incluir a biblioteca do CC50 no topo do programa com

```
#include <cc50.h>
```

Além disso, toda vez que você compilar um programa que usa essa biblioteca (usando o gcc), terá que adicionar a flag **-lcc50** para que o seu programa compile normalmente.

Abaixo está uma lista das funções que você encontra nessa biblioteca. Todas leem uma linha de input do usuário e retornam um valor como especificado no nome da função (Char, Double, Float, Int, LongLong ou String). Relaxe, por enquanto você só precisa saber o que alguns desses nomes significam.

GetChar();
GetDouble();
GetFloat();
GetInt();
GetLongLong();
GetString();

Lembramos ainda que nenhuma dessas funções aceita argumentos.

Doce grátis.

☐ Vamos começar a programar?!

Então, uma das melhores coisas sobre o Maxwell Dworkin (o edifício de Ciência da Computação de Harvard) é a máquina de doces grátis no salão[11,12]

Bem, isso e o gato no teto. De qualquer forma, há um monte de Skittles (balas) e M&Ms nessa máquina.[13] Quer adivinhar quantos Skittles tem lá?

Que bom que você disse sim! Implemente, em um arquivo chamado `skittles.c`, um programa que primeiro escolhe um número aleatório (pseudo) entre 0 e 1023 e então pede que você (o

[11] Pode ser que nós mesmos a tenhamos hackeado para que ela seja grátis.
[12] Foto por Dan Armendariz.
[13] Provavelmente deve ter menos agora que nós mencionamos a existência da máquina.

humano) adivinhe qual é esse valor.[14] O programa deve continuar pedindo para você adivinhar até que você adivinhe o valor certo, então ele deve lhe agradecer por jogar.

Por onde começar? Permita-nos entregar-lhe algumas peças do quebra-cabeça.

Para gerar um número aleatório, você pode usar uma função chamada `rand`. Dê uma olhada na página do manual, executando o comando abaixo:[15]

```
man 3 rand
```

O `3` instrui o `man` a consultar a seção 3 do manual do Linux: Considerando que os programas são geralmente documentado na seção 1 (o padrão se você não especificar nenhum número), as funções são muitas vezes documentadas na seção 3. Na parte **`SYNOPSIS`** do manual você pode ver que a função é declarada em `stdlib.h`. Então você vai querer colocar

```
#include <stdlib.h>
```

no topo de `skittles.c` juntamente com

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

como de costume. A ordem dos includes não tem importância, mas utilizar ordem alfabética é provavelmente uma boa decisão de estilo.

Note também que `rand` "retorna um valor entre 0 e `RAND_MAX`."Acontece que `RAND_MAX` é uma "constante" (um símbolo que representa algum valor) que é definido em `stdlib.h`. Seu valor pode variar de acordo com a versão do Linux, e por isso não consta no manual. Vamos supor que `RAND_MAX` é maior do que 1023. Como, porém, é que vamos mapear um número que está entre 0 e `RAND_MAX` para um número que está entre 0 e 1023?

Acontece que operador de módulo `(%)` que vimos nas aulas é útil para mais do que simples restos de divisão aritmética! Considere esta linha de código:

```
int skittles = rand() % 1024;
```

O efeito dessa linha é dividir o "valor de retorno" da função `rand` por 1024 e armazenar o resto na variável `skittles`. O que seria o restante, porém, ao dividir algum inteiro por 1024? Bem, pode não haver restante, caso em que a resposta é 0, ou pode haver um resto maior, caso em que a resposta é 1023. E, claro, um monte de outros restos são possíveis entre esses limites.

Bem, aí está uma forma de gerar um número pseudoaleatório entre 0 e 1023! Só tem um problema. Acontece que, por padrão, `rand` sempre retorna o mesmo número na primeira vez que ele é chamado em um programa (Hmm... eu diria, talvez, `1804289383?!)`. Isso quer dizer que sua máquina de balas sempre vai ser preenchida com o mesmo número de Skittles. Isso porque,

[14] Para ser mais claro o conjunto engloba 1024 números, de (e incluindo) 0 até (e incluindo) 1023.

[15] Lembre que para sair de man você pode simplesmente apertar q.

como consta na a página man do `rand`, "Se nenhum valor seed é fornecido, a função `rand()` é automaticamente semeada com o valor 1."

A "seed" é simplesmente um valor que influencia a sequência de valores retornados por um "gerador de números pseudoaleatórios" (PRNG – Pseudo Random Number Generator) como a função `rand`. Para ser claro, a "seed" não é o primeiro número retornado por um PRNG, mas, sim, uma influência nele. (Veja por que dizemos "pseudo" o tempo todo em vez de simplesmente "aleatório", os computadores não podem gerar números verdadeiramente aleatórios: eles têm de começar de algum ponto) Como você pode substituir essa "seed" padrão de valor 1? Antes de você chamar `rand`, chame `srand` com sua escolha de Seed *(*nesse caso escolhemos 2):

```
srand(2);
```

Melhor ainda, chame `srand` com uma semente que realmente muda com o tempo (literalmente), sem ter que recompilar o seu código cada vez que você quiser mudar:

```
srand(time(NULL));
```

O que `NULL` faz não importa agora, mas saiba que

```
time(NULL)
```

retorna a hora atual em segundos; Essa não é uma seed ruim. E não há necessidade de armazenar o valor de retorno de `time` em alguma variável primeiro, podemos passá-lo diretamente para `srand` entre os parênteses. É importante notar, porém, que a função `time` é declarada em `time.h`, então você precisa incluir essa biblioteca no seu programa também.

Tudo bem, quais outros pedaços de quebra-cabeça nós precisamos? Bem, o programa terá de informar ao usuário o que fazer, para o qual `printf` deve ser útil. E você vai querer permitir ao usuário um número infinito de tentativas, para isso a utilizacão de um loop é provavelmente necessária. E você também vai querer armazenar inteiros inseridos pelo usuário, para isso a função `GetInt`, declarada em `cc50.h`, é definitivamente útil.

Ok, por onde começar? Permita-nos sugerir que você comece preenchendo `skittles.c` com este código:

```
#include <cc50.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int
main(void)
{
    // seed do PRNG
    srand(time(NULL));

    // escolhe número pseudo-aleatório entre [0, 1023]
    int skittles = rand() % 1024;
```

```
    // TODO
}
```

Vamos deixar o `TODO` para você! Lembre-se, não tente implementar todo o programa de uma vez. Comece, talvez, por informar ao usuário (com `printf`) o valor pseudo-aleatório de `skittles`, só para ter certeza que você não cometeu nenhum erro de digitação. Em seguida, salve o seu código e saia do `nano` (com ctrl-x), prossiga agora para compilá-lo com:[16]

```
make skittles
```

Lembre-se que este comando poupa-lhe o trabalho de executar `gcc` diretamente com a flag `-o`. Para executar o seu programa (supondo que ele compile sem erros), digite

```
./skittles
```

para ver quantos Skittles a máquina tem. Espere um segundo e depois execute o programa novamente: (muito) provavelmente o número de Skittles será diferente. Agora reabra `skittles.c` com o `nano` e dê outro passo para resolver esse problema. Talvez a próxima implementação seja as instruções do seu programa, que explicam ao usuário como jogar este jogo de adivinhação.

Como deve se parecer o output do seu programa, uma vez totalmente implementado? Deixamos a personalidade da sua máquina de Skittles falante inteiramente para você, mas abaixo se encontra um projeto possível. Suponha que o texto em negrito é o que algum usuário digitou.

```
username@computer (~/cc50/pset1): ./skittles
Olá! Eu sou uma máquina de balas falante! Adivinhe quantos Skittles têm dentro de mim. Dica: Estou pensando em um número entre 0 e 1023. Qual é ele?
0
Haha! Tenho muito mais Skittles do que isso. Tente novamente.
1
Haha! Tenho muito mais Skittles do que isso. Tente novamente.
-1
Não tente ser difícil... Adivinhe novamente.
1024
Não tente ser difícil... Adivinhe novamente.
1023
Ok, eles não são tantos assim. Tente novamente.
42
Você está certo! Nom nom nom nom.
```

Seu programa deve terminar quando o usuário acertar. O design acima responde aos inputs do usuário de algumas maneiras diferentes, mas vamos deixar para você decidir o quanto você quer variar as respostas dadas pelo seu programa.

Aliás, saiba que você geralmente pode forçar a saída de um programa prematuramente pressionando ctrl-c. Em nome da eficiência, você pode achar útil abrir duas janelas do Terminal,

[16] Se fosse tão fácil assim fazer Skittles...

de modo que você possa manter o `nano` aberto em uma e usar a outra para compilar e testar tudo. E lembre da primeira aula, quando você viu que encontrar um valor entre 0 e 1023 na verdade não exige muitos palpites. Você provavelmente pode testar o programa de forma bem rápida. Você certamente pode usar temporariamente umas lista com menos de 1024 números para poupar ainda mais tempo. Basta ter certeza de que a sua versão final escolha um número entre [0, 1023].

Hora do troco.

☐ "Contar o troco é muito mais fácil com este trocador feito de molas que você usa no seu cinto para despejar moedas em sua mão." Ou assim está escrito no site no qual encontramos este simpático e engenhoso acessório de moda.[17]

Claro, a engenhosidade desta coisa rapidamente se esgota, especialmente quando algum idiota quiser pagar pelo seu jornal com uma nota de cinquenta reais. Felizmente, a ciência da computação deu aos caixas formas de minimizar o número de moedas gastas: algoritmos gulosos ou gananciosos (greedy algorithms).

De acordo com o National Institute of Standards and Technology (NIST), um algoritmo guloso é uma técnica que age "sempre realizando a escolha que parece ser a melhor no momento; fazendo uma escolha ótima local, na esperança de que esta escolha leve até a solução ótima global. Algoritmos gulosos encontram a solução global ideal para alguns problemas de otimização, mas eles podem encontrar soluções não-ideais para certos casos de alguns problemas." [18]

O que tudo isso significa? Bem, suponha que um caixa deve troco a um cliente e no seu cinto se encontram moedas de 25, 10, 5 e 1 centavos. Resolver este "problema" exige uma ou mais

[17] Descrição e imagens de `hearthsong.com`. Para 5 anos ou mais.

[18] `http://www.nist.gov/dads/HTML/greedyalgo.html`

retiradas de um ou mais tipos diferentes de moeda. Pense em um caixa "guloso", alguém que quer resolver, com cada retirada de uma única moeda, a maior parte possível deste problema. Por exemplo, se algum cliente precisa receber 41¢, o melhor primeiro passo *(*a melhor escolha imediata, ou local) é a retirada de uma moeda de 25¢ (é "melhor" pois essa retirada nos deixa mais próximo de 0¢ do que qualquer outra retirada de uma única moeda). Note que esse primeiro passo diminui o problema de 41¢ para um problema de 16¢, pois 41-25 = 16. Ou seja, o segundo é um problema semelhante, mas menor. Nem precisamos dizer que uma outra retirada de 25¢ seria muito grande (assumindo que o caixa prefere não perder dinheiro), e assim o nosso caixa guloso retiraria uma moeda de 10¢, ficando com um problema de 6¢. Nesse ponto, a ganância pede uma retirada de 5¢ seguida de uma retirada de 1¢, e agora o problema está resolvido. O cliente recebe uma moeda de 25¢, uma moeda de 10¢, uma de 5¢, e uma moeda de 1¢: quatro moedas no total.

Acontece que essa abordagem gulosa *(*algoritmo guloso) não só é localmente, mas também globalmente ideal para as moedas do Brasil (e também para as dos EUA ou da União Europeia). Ou seja, enquanto um caixa tem um número suficiente de cada moeda, esta abordagem irá retirar, no final, o menor número de moedas possível.[19]

Quão pequeno é esse número de moedas? Bem, conte-nos. Escreva, em `greedy.c`, um programa que primeiro pede ao usuário quanto de troco ele quer receber e, em seguida, reporta (via `printf`) o número mínimo de moedas com o qual esse troco pode ser dado. Use GetFloat da biblioteca do CC50 para obter o input do usuário e `printf` da biblioteca Standard I/O para dar a sua resposta.

Você deve usar GetFloat para que você possa lidar com reais e centavos. Além disso fique atento pois o padrão de formatação de números decimais em programação utiliza pontos (.) ao invés de vírgulas (,). Em outras palavras, se algum cliente deve receber R$9.75 (como no caso em que um jornal custa 25¢, mas o cliente paga com uma nota de R$10), assuma que o input do seu programa será **`9.75`** e não **`9.75 reais`** ou **`9,75`**. No entanto, se algum cliente deve receber exatamente R$9, assuma que o input do seu programa será **`9.00`** ou apenas **`9`**, mas, novamente, não R**`$9`** ou **`9,00`**. É claro que, pela natureza dos valores de ponto flutuante (floating point values), o seu programa provavelmente vai trabalhar com inputs como **`9.0`** e **`9.000`**, assim, você não precisa se preocupar sobre como verificar se o input do usuário é "formatado" como o dinheiro deveria ser. E você não precisa tentar verificar se o input de um usuário é muito grande para caber em um `float`. Mas você deve verificar se o input do usuário realmente pode ser visto como um número válido de centavos! Como assim? Usando `GetFloat` por si só, garante que o input do usuário é de fato um valor de ponto flutuante (ou inteiro), mas não que é positivo. Enquanto o usuário não fornecer um valor positivo, o programa deve solicitar novamente ao usuário uma quantidade válida até que o usuário coopere. Aliás, tome cuidado com a imprecisão inerente de valores de ponto flutuante.[20]

---

[19] Em contraste, suponha que um caixa não possua nenhuma moeda de 5¢ mas ainda assim precise dar 41¢ de troco. Quantas moedas serão utilizadas por um caixa guloso? E quantas serão utilizadas por um caixa "globalmente ótimo"?

[20] Por exemplo, 0.01 não pode ser exatamente representado por um `float`. Tente escrever esse número, digamos, mostrando dez casas decimais, utilizando código como o seguinte:

```
float f = 0.01;
printf("%.10f\n", f);
```

Antes de fazer qualquer conta, você provavelmente vai querer converter a entrada do usuário inteiramente para centavos (a partir de um `float` para um `int`) para evitar pequenos erros que poderiam se somar e tornar-se grandes![21] Tenha cuidado para não arrendondar demais os seus tostões!

Para que nós possamos automatizar alguns testes do seu código, pedimos que a última linha que o seu programa escreva seja apenas o número mínimo de moedas possíveis: um número inteiro seguido por `\n`. Considere a representação abaixo de como o seu programa deve se comportar; o input de algum usuário está destacado em negrito.

```
username@computer (~/cc50/pset1): ./greedy
Oi. Quanto troco você deve?
0.41
4
```

Pela natureza dos valores de ponto flutuante, o usuário também poderia ter digitado apenas **.41** representando o mesmo número.

Naturalmente, os usuários mais difíceis (vamos chamá-los de... Hmm... n00bs) verão algo mais parecido com o abaixo.

```
username@computer (~/cc50/pset1): ./greedy
Oi. Quanto troco você deve?
-0,41
Desculpe? Quanto você disse?
-0,41
Desculpe? Quanto você disse?
foo
Ah... Tente de novo.
0.41
4
```

Devido a essas exigências (e ao exemplo acima), o seu código provavelmente vai ter algum tipo de loop. Se, ao testar o seu programa, você se encontrar loopando para sempre, lembre-se que você sempre pode forçar o seu programa a morrer (ou seja curto-circuitar a sua execução), pressionando ctrl-c (talvez muitas vezes seguidas).

Deixamos para você a determinação de como compilar, executar e debugar este programa!

[21] Não transforme o input do usuário simplesmente de `float` para `int`! Pense um pouco, quantos centavos equivalem a um real?

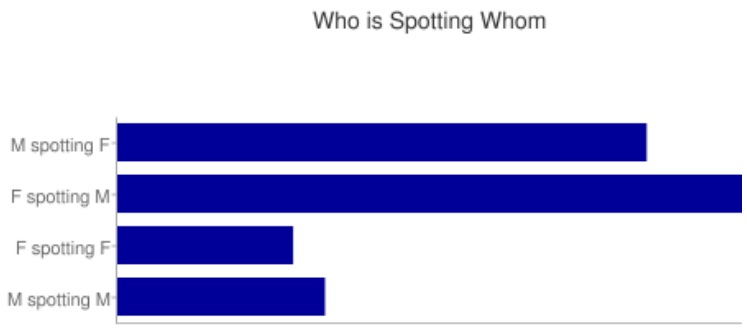
Eu te vi.

☐ Vá para

`http://isawyouharvard.com/`

onde você vai encontrar um site criado por um aluno do CS50 de Harvard, Tej Toor '10 como Projeto Final do ano passado, "sua fonte para postagem e busca de conexões perdidas." Quer deixar alguém especial saber que você viu ele ou ela algum dia? Aqui está a sua chance! Nós não saberemos que foi você.[22]

De qualquer forma, agora que nós temos a sua atenção de novo, siga o link **Statistics** na parte superior do site, onde você encontrará alguns simpáticos recursos visuais, entre eles um gráfico de barras. Que deve se parecer com o seguinte:



Acontece que é muito fácil integrar esse tipo de coisas em um site atualmente. Tej está usando o Google Chart API (uma biblioteca gratuita) para gerar os gráficos:

`http://code.google.com/apis/chart/`

Se você ficou curioso, a documentação sobre gráficos de barras, especificamente, se encontra em:

`http://code.google.com/apis/chart/docs/gallery/bar_charts.html`

Nós também utilizamos um serviço semelhante, o API de visualização do Google, em Harvard Energy, a App do CS50 com o qual você pode explorar o consumo de energia de Harvard e as suas influências no efeito estufa:

`http://energy.cs50.net/`

Selecione um dormitório ou casa através dos menus drop-down no canto superior esquerdo para ver vários tipos de dados interessantes. Aqui está o que você pode fazer além disso, com este API em particular:

`http://code.google.com/apis/visualization/documentation/gallery.html`

[22] Ou saberemos...?

Basta dizer então que no fim desse curso você vai ter todo o conhecimento necessário para implementar ISawYouHarvard e HarvardEnergy!
Por enquanto, porém, estamos confinados a um ambiente de linha de comando. Mas não se preocupe, ainda podemos fazer algumas coisas bem interessantes. Na verdade, nós certamente podemos gerar gráficos de barras com "arte ASCII". Faremos isso agora.

Implemente, em `chart.c`, um programa que solicita ao usuário quatro inteiros não negativos (um para cada um dos valores de **M procurando F**, **F procurando M**, **F procurando F** e **M procurando M**). O seu programa deve então gerar um gráfico de barras horizontais com esses valores, com a barra do primeiro valor no topo e a barra do quarto valor embaixo. Suponha que a janela do usuário é de pelo menos 80 caracteres de largura por 24 caracteres de altura. Cada barra deve ser representada como uma sequência horizontal de 0 ou mais sinais de jogos-da-velha (`#`), até um máximo de 80. O comprimento de cada barra deve ser proporcional ao valor correspondente e relativo à soma dos quatro valores . Por exemplo, se o usuário inserir 10, 0, 0 e 0, a primeira barra deve possuir 80 jogos-da-velha de comprimento, pois 10 é 100% de 10 + 0 + 0 + 0 = 10 e 100% de 80 é 80, e as três barras restantes devem possuir 0 jogos-da-velha de comprimento. Por outro lado, se o usuário insere 5, 5, 0 e 0, cada uma das duas barras principais deve ter 40 jogos-da-velha, pois 5 representa 50% dos 5 + 5 + 0 + 0 = 10 e 50% de 80 é 40, e as duas barras inferiores devem possuir 0 jogos-da-velha de comprimento. Assim, se o usuário insere 2, 2, 2, 2, cada uma das quatro barras deve ter 20 jogos-da-velha de comprimento, pois 2 é 25% de 2 + 2 + 2 + 2 = 8 e 25% de 80 é 20. E assim por diante. Quando for computar as proporções, vá em frente e arredonde para o `int` mais próximo (simplesmente transformando todos os valores de `float` para `int`).

Ao invés de rotular cada barra na esquerda como o Google faz, coloque cada rótulo imediatamente acima da barra correspondente; você pode, se quiser, deixar algumas linhas em branco para maior clareza, desde que as últimos 8 linhas do seu output sejam as barras e os seus rótulos. As suas barras não precisam ser, de forma alguma, animadas. Considere o exemplo de output abaixo; suponha que o texto em negrito é o que algum usuário digitou.

```
username@computer (~/cc50/pset1): ./chart

M procurando F: 3
F procurando M: 4
F procurando F: 1
M procurando M: 2

Quem procura quem?

M procurando F
########################
F procurando M
##############################
F procurando F
########
M procurando M
################
```

☐ Hmm... Acabou!