Este é o CS50

Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

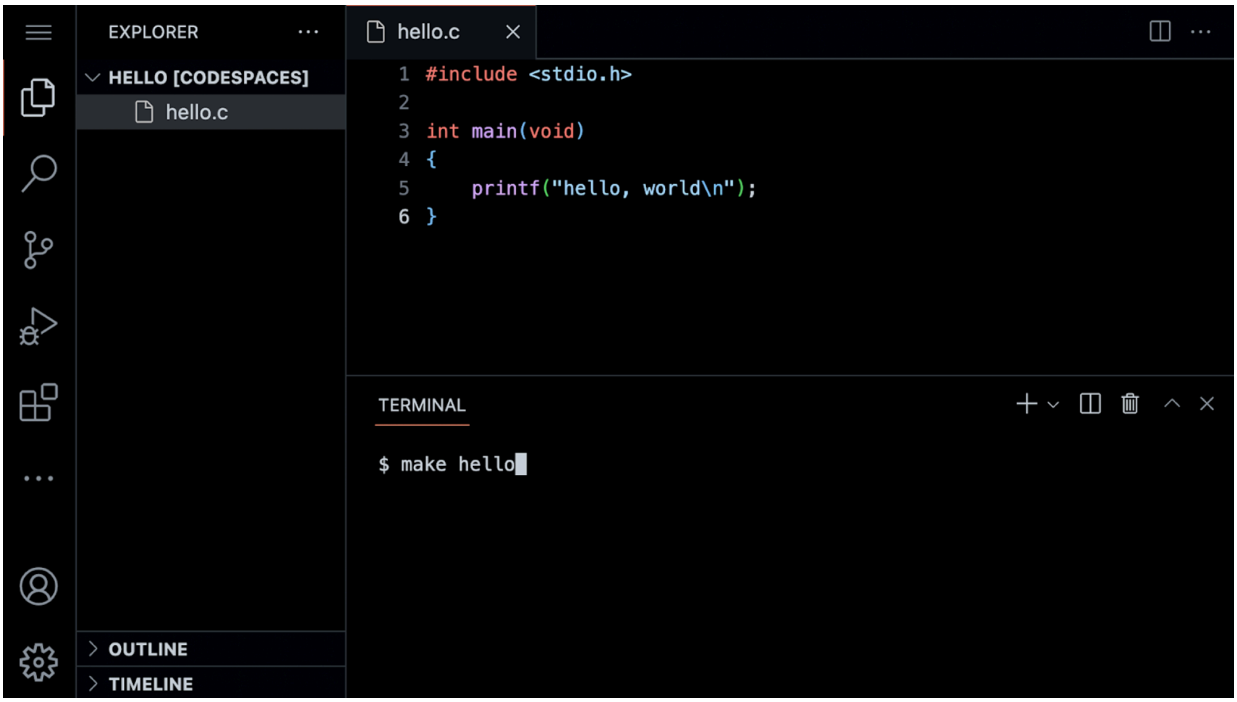
Aula 1

Bem-vindo!

- Na nossa sessão anterior, aprendemos sobre o Scratch, uma linguagem de programação visual.
- De fato, todos os conceitos essenciais de programação apresentados no Scratch serão utilizados à medida que você aprende a programar em qualquer linguagem de programação. Funções, condicionais, loops e variáveis encontrados no Scratch são blocos de construção fundamentais que você encontrará em qualquer linguagem de programação.
- Lembre-se de que as máquinas só entendem código binário. Enquanto os humanos escrevem *código-fonte* , uma lista de instruções para o computador que é legível para humanos, as máquinas só entendem o que hoje chamamos de *código de máquina* . Esse código de máquina é um padrão de uns e zeros que produz o efeito desejado.
- Descobrimos que podemos converter *código-fonte* em código de máquina usando um software muito especial chamado *compilador* . Hoje, apresentaremos um compilador que permite converter código-fonte da linguagem de programação *C* em código de máquina.
- Hoje, além de aprender a programar, você aprenderá a escrever um bom código.

Visual Studio Code para CS50

- O editor de texto utilizado neste curso é *o Visual Studio Code* , também conhecido como *VS Code* , carinhosamente chamado de cs50.dev (https://cs50.dev) , que pode ser acessado através do mesmo URL.
- Um dos principais motivos para utilizarmos o VS Code é que ele já vem com todos os softwares necessários para o curso pré-instalados. Este curso e as instruções aqui contidas foram desenvolvidos pensando especificamente no VS Code.
- Instalar manualmente o software necessário para o curso no seu próprio computador é uma tarefa árdua e complicada. O ideal é sempre utilizar o VS Code para as tarefas deste curso.
- Você pode abrir o VS Code em cs50.dev (https://cs50.dev/) .
- O compilador pode ser dividido em várias regiões:



Observe que há um *explorador de arquivos* no lado esquerdo, onde você pode encontrar seus arquivos. Além disso, observe que há uma área no meio chamada *editor de texto* , onde você pode editar seu programa. Finalmente, há uma `command line interface` interface de *linha de comando (CLI*) ou *janela de terminal* , onde podemos enviar comandos para o computador na nuvem.

- Na janela do terminal, alguns argumentos comuns da linha de comando que podemos usar incluem:
  - `cd` , para alterar nosso diretório (pasta) atual
  - `cp` , para copiar arquivos e diretórios
  - `ls` , para listar arquivos em um diretório
  - `mkdir` , para criar um diretório
  - `mv` , para mover (renomear) arquivos e diretórios
  - `rm` , para remover (excluir) arquivos
  - `rmdir` , para remover (excluir) diretórios
- O comando mais comum é `ls` o `ls`, que lista todos os arquivos no diretório atual. Digite `ls` `ls` na janela do terminal e pressione Enter `enter` . Você verá todos os arquivos na pasta atual.
- Como este ambiente de desenvolvimento integrado (IDE) já vem pré-configurado com todo o software necessário, você deve usá-lo para concluir todas as tarefas deste curso.

Olá, mundo!

- Usaremos três comandos para escrever, compilar e executar nosso primeiro programa:

```
code hello.c

make hello
```

```
./hello
```

O primeiro comando `code hello.c` cria um arquivo e nos permite digitar instruções para este programa. O segundo comando `make hello` compila o arquivo a partir de nossas instruções em C e cria um arquivo executável chamado . O último comando executa o programa *chamado* . `hello` `./hello` `hello`

- Podemos criar seu primeiro programa em C digitando `code hello.c` o seguinte no terminal. Observe que escrevemos o nome do arquivo inteiro em minúsculas e incluímos a `.c` extensão. Em seguida, no editor de texto que aparecer, escreva o código da seguinte forma:

```c
// A program that says hello to the world

#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Observe que cada caractere acima tem uma função. Se você digitar incorretamente, o programa não será executado. `print` `printf` é uma função que pode gerar uma linha de texto. Observe a posição das aspas e do ponto e vírgula. Além disso, observe que `print` `\n` cria uma nova linha após as palavras `hello, world` .

- Clicando novamente na janela do terminal, você pode compilar seu código executando o comando ` `make hello` . Observe que estamos omitindo o ` `.c` .`. O `.` `make` é um compilador que procurará nosso `hello.c` arquivo e o transformará em um programa chamado ` `hello` . Se a execução deste comando não resultar em erros, você pode prosseguir. Caso contrário, verifique seu código novamente para garantir que ele corresponda ao comando acima.
- Agora, digite `./hello` e seu programa será executado dizendo `hello, world` .
- Agora, abra o explorador de arquivos à esquerda. Você notará que agora existem um arquivo chamado `.txt` `hello.c` e outro chamado `.txt` `hello` . ` `hello.c` .txt` pode ser lido pelo compilador: é onde seu código está armazenado. ` .txt` `hello` é um arquivo executável que você pode executar, mas não pode ser lido pelo compilador.

Do zero ao C

- No Scratch, usávamos o `say` bloco para exibir qualquer texto na tela. De fato, em C, temos uma função chamada `printf` que faz exatamente isso.
- Observe que nosso código já invoca essa função:

```c
printf("hello, world\n");
```

Observe que a função printf é chamada. O argumento passado para printf é `hello, world\n`. A instrução de código é fechada com um `;`.

- Erros no código são comuns. Modifique seu código da seguinte forma:

```
// \n is missing

#include <stdio.h>

int main(void)
{
    printf("hello, world");
}
```

Note que o " `\n`i" desapareceu.

- Na janela do terminal, execute o comando `make hello`. Ao digitar `./hello` no terminal, como o seu programa mudou? Esse `\` caractere é chamado de *caractere de escape* e indica ao compilador que `\n` se trata de uma instrução especial para criar uma quebra de linha.
- Existem outros caracteres de escape que você pode usar:

```
\n  create a new line
\r  return to the start of a line
\"  print a double quote
\'  print a single quote
\\  print a backslash
```

- Restaure seu programa para o seguinte estado:

```
// A program that says hello to the world

#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Observe que o ponto e vírgula e `\n` foram restaurados.

Arquivos de cabeçalho e páginas do manual do CS50

- A declaração no início do código `#include <stdio.h>` é um comando muito especial que informa ao compilador que você deseja usar os recursos de uma *biblioteca* chamada `stdio.h`, um *arquivo de cabeçalho* . Isso permite, entre muitas outras coisas, utilizar a `printf` função.
- Uma *biblioteca* é uma coleção de código criada por alguém. Bibliotecas são coleções de código e funções pré-escritas que outros escreveram no passado e que podemos utilizar em nosso código.
- Você pode ler sobre todas as funcionalidades desta biblioteca nas Páginas de Manual (https://manual.cs50.io) . As Páginas de Manual fornecem um meio de compreender

melhor o que os vários comandos fazem e como funcionam.

- Acontece que o CS50 possui sua própria biblioteca chamada `cs50.h`. Ela inclui diversas funções que servem como *apoio* para quem está começando a programar em C:

```
get_char
get_double
get_float
get_int
get_long
get_string
```

- Vamos usar essa biblioteca em seu programa.

Olá, você!

- Lembre-se de que no Scratch tínhamos a possibilidade de perguntar ao usuário: "Qual é o seu nome?" e dizer "olá" acrescentando esse nome à mensagem.
- Em C, podemos fazer o mesmo. Modifique seu código da seguinte forma:

```
// get_string and printf with incorrect placeholder

#include <stdio.h>

int main(void)
{
    string answer = get_string("What's your name? ");
    printf("hello, answer\n");
}
```

A `get_string` função é usada para obter uma string do usuário. Em seguida, a variável `answer` é passada para a `printf` função.

- Ao executar `make hello` novamente o comando na janela do terminal, observe que vários erros aparecem.
- Analisando os erros, percebemos que `__init__` `string` e ` `get_string` __outit__` não são reconhecidos pelo compilador. Precisamos ensinar o compilador a usar esses recursos adicionando uma biblioteca chamada `__init__` `cs50.h`. Além disso, notamos que `__init__` `answer` não está sendo fornecido como pretendido. Modifique seu código da seguinte forma:

```
// get_string and printf with %s

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

int main(void)
{
    string answer = get_string("What's your name? ");
    printf("hello, %s\n", answer);
}
```

A `get_string` função é usada para obter uma string do usuário. Em seguida, a variável `answer` é passada para a `printf` função. O parâmetro `prepare` `%s` indica à `printf` função que ela deve se preparar para receber um valor `string`.

- Agora, executando `make hello` novamente na janela do terminal, você pode executar seu programa digitando `./hello`. O programa agora pede seu nome e, em seguida, diz "olá" com seu nome anexado, como esperado.
- `answer` é um espaço de armazenamento especial que chamamos de *variável* . `answer` É do tipo `string` e pode armazenar qualquer string. Existem muitos *tipos de dados* , como `int` , `bool` , `char` e muitos outros.
- `%s` é um marcador de posição chamado *código de formato* que indica à `printf` função que ela deve se preparar para receber um `string` . `answer` é o `string` que está sendo passado para `%s` .

Tipos

- `printf` Permite diversos códigos de formato. Aqui está uma lista não exaustiva dos que você pode utilizar neste curso:

```
%c
%f
%i
%li
%s
```

  `%s` é usado para `string` variáveis. `%i` é usado para `int` variáveis inteiras. Você pode encontrar mais informações sobre isso nas Páginas do Manual. (https://manual.cs50.io)

- Esses códigos de formato correspondem aos diversos tipos de dados disponíveis em C:

```
bool
char
float
int
long
string
...
```

- Ao longo deste curso, utilizaremos muitos dos tipos de dados disponíveis em C.

Condicionais

- Outro elemento fundamental que você utilizou no Scratch foram *as condicionais* . Por exemplo, você pode querer fazer uma coisa se x for maior que y. Além disso, você pode querer fazer outra coisa se essa condição não for atendida.
- Vamos analisar alguns exemplos do Scratch.
- Em C, você pode comparar dois valores da seguinte forma:

```c
// Conditionals that are mutually exclusive

if (x < y)
{
    printf("x is less than y\n");
}
else
{
    printf("x is not less than y\n");
}
```

Observe que, se `x < y`, ocorre um resultado. Se `x` não for menor que `y`, então ocorre outro resultado.

- Da mesma forma, podemos planejar três resultados possíveis:

```c
// Conditional that isn't necessary

if (x < y)
{
    printf("x is less than y\n");
}
else if (x > y)
{
    printf("x is greater than y\n");
}
else if (x == y)
{
    printf("x is equal to y\n");
}
```

Observe que nem todas essas linhas de código são necessárias. Como poderíamos eliminar o cálculo desnecessário acima?

- Você provavelmente já imaginou que podemos melhorar este código da seguinte forma:

```c
// Compare integers

if (x < y)
{
    printf("x is less than y\n");
}
else if (x > y)
{
    printf("x is greater than y\n");
}
else
{
    printf("x is equal to y\n");
}
```

Observe como a declaração final é substituída por `else`.

Operadores

- *Os operadores* referem-se às operações matemáticas suportadas pelo seu compilador. Em C, esses operadores matemáticos incluem:
  - `+` para adição
  - `-` para subtração
  - `*` para multiplicação
  - `/` para divisão
  - `%` pelo resto
- Usaremos todos esses operadores neste curso.

Variáveis

- Em C, você pode atribuir um valor a um `int` número inteiro da seguinte forma:

  ```
  int counter = 0;
  ```

  Observe como uma variável chamada `counter` do tipo `int` recebe o valor `0`.
- A linguagem C também pode ser programada para adicionar um, da `counter` seguinte forma:

  ```
  counter = counter + 1;
  ```

  Observe como `1` é adicionado ao valor de `counter`.
- Isso também pode ser representado como:

  ```
  counter += 1;
  ```
- Isso pode ser ainda mais simplificado para:

  ```
  counter++;
  ```

  Observe como o símbolo `++` é usado para adicionar 1.
- Você também pode subtrair um da `counter` seguinte forma:

  ```
  counter--;
  ```

  Observe como `1` é removido do valor de `counter`.

compare.c

- Com esse novo conhecimento sobre como atribuir valores a variáveis, você poderá programar sua primeira instrução condicional.
- Na janela do terminal, digite `code compare.c` e insira o código da seguinte forma:

```c
// Conditional, Boolean expression, relational operator

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

int main(void)
{
    // Prompt user for integers
    int x = get_int("What's x? ");
    int y = get_int("What's y? ");

    // Compare integers
    if (x < y)
    {
        printf("x is less than y\n");
    }
}
```

Observe que criamos duas variáveis, uma `int` do tipo inteiro chamada `x` `x` e outra chamada `y` `y`. Os valores dessas variáveis são preenchidos usando a `get_int` função `x`.

- Você pode executar seu código digitando `run` `make compare` na janela do terminal, seguido de `run` `./compare`. Se você receber alguma mensagem de erro, verifique seu código em busca de erros.
- *Os fluxogramas* são uma forma de examinar o funcionamento de um programa de computador. Esses fluxogramas podem ser usados para analisar a eficiência do nosso código.
- Ao analisar o fluxograma do código acima, podemos notar diversas deficiências.
- Podemos melhorar seu programa codificando-o da seguinte forma:

```c
// Conditionals

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

int main(void)
{
    // Prompt user for integers
    int x = get_int("What's x? ");
    int y = get_int("What's y? ");

    // Compare integers
    if (x < y)
    {
        printf("x is less than y\n");
    }
    else if (x > y)
    {
        printf("x is greater than y\n");
    }
    else
    {
        printf("x is equal to y\n");
    }
}
```

Observe que agora todos os resultados possíveis foram considerados.

- Você pode refazer e executar seu programa novamente para testá-lo.
- Ao analisar este programa em um fluxograma, você poderá observar a eficiência das nossas decisões de design de código.

Concordo.c

- Considerando outro tipo de dado chamado `char`, podemos iniciar um novo programa digitando `code agree.c` na janela do terminal.
- Onde a `string` representa uma série de caracteres, a `char` representa um único caractere.
- No editor de texto, escreva o código da seguinte forma:

```
// Comparing against lowercase char

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

int main(void)
{
    // Prompt user to agree
    char c = get_char("Do you agree? ");

    // Check whether agreed
    if (c == 'y')
    {
        printf("Agreed.\n");
    }
    else if (c == 'n')
    {
        printf("Not agreed.\n");
    }
}
```

Observe que as aspas simples são utilizadas para caracteres individuais. Além disso, note que isso `==` garante que algo *seja igual* a outra coisa, enquanto um sinal de igual simples teria uma função muito diferente em C.

- Você pode testar seu código digitando `make agree` no terminal, seguido de `./agree`.
- Também podemos permitir a entrada de caracteres maiúsculos e minúsculos:

```
// Comparing against lowercase and uppercase char

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

int main(void)
{
    // Prompt user to agree
    char c = get_char("Do you agree? ");

    // Check whether agreed
    if (c == 'y')
```

```
    {
        printf("Agreed.\n");
    }
    else if (c == 'Y')
    {
        printf("Agreed.\n");
    }
    else if (c == 'n')
    {
        printf("Not agreed.\n");
    }
    else if (c == 'N')
    {
        printf("Not agreed.\n");
    }
}
```

Note que opções adicionais são oferecidas. No entanto, este não é um código eficiente.

- Podemos melhorar este código da seguinte forma:

```
// Logical operators

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

int main(void)
{
    // Prompt user to agree
    char c = get_char("Do you agree? ");

    // Check whether agreed
    if (c == 'Y' || c == 'y')
    {
        printf("Agreed.\n");
    }
    else if (c == 'N' || c == 'n')
    {
        printf("Not agreed.\n");
    }
}
```

Note que " `||` efetivamente" significa " *ou* ".

Loops e miau.c

- Também podemos utilizar o bloco de construção de loop do Scratch em nossos programas em C.
- Na janela do terminal, digite `code meow.c` e escreva o código da seguinte forma:

```
// Opportunity for better design

#include <stdio.h>

int main(void)
```

```
{
    printf("meow\n");
    printf("meow\n");
    printf("meow\n");
}
```

Note que isso funciona como esperado, mas há espaço para melhorias no design. O código se repete diversas vezes.

- Podemos melhorar nosso programa modificando seu código da seguinte forma:

```
// Better design

#include <stdio.h>

int main(void)
{
    int i = 3;
    while (i > 0)
    {
        printf("meow\n");
        i--;
    }
}
```

Observe que criamos uma `int` variável chamada `x` `i` e atribuímos a ela o valor `y` `3`. Em seguida, criamos um `while` loop que será executado enquanto `y` for verdadeiro `i > 0`. Então, o loop é executado. A cada iteração, `y` `1` é subtraído de `x` `i` usando a `i--` instrução `while`.

- Da mesma forma, podemos implementar uma espécie de contagem crescente modificando nosso código da seguinte maneira:

```
// Print values of i

#include <stdio.h>

int main(void)
{
    int i = 1;
    while (i <= 3)
    {
        printf("meow\n");
        i++;
    }
}
```

Observe como nosso contador `i` é iniciado em `1`. A cada execução do loop, o contador será incrementado em `1`. Quando o contador for maior que `3`, o loop será interrompido.

- Em geral, na ciência da computação, contamos a partir do zero. O melhor é revisar seu código da seguinte forma:

```
// Better design

#include <stdio.h>
```

```c
int main(void)
{
    int i = 0;
    while (i < 3)
    {
        printf("meow\n");
        i++;
    }
}
```

Observe que agora contamos a partir do zero.

- Outra ferramenta em nosso conjunto de ferramentas para criar loops é um `for` próprio loop.
- Você pode aprimorar ainda mais o design do nosso `meow.c` programa usando um `for` laço de repetição. Modifique seu código da seguinte forma:

```c
// Better design

#include <stdio.h>

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

Observe que o `for` laço inclui três argumentos. O primeiro argumento `int i = 0` inicia o contador em zero. O segundo argumento `i < 3` é a condição que está sendo verificada. Finalmente, o argumento ` `i++` increment` instrui o laço a incrementar em um a cada iteração.

- Podemos até mesmo criar um loop infinito usando o seguinte código:

```c
// Infinite loop

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

int main(void)
{
    while (true)
    {
        printf("meow\n");
    }
}
```

Note que isso `true` sempre acontecerá. Portanto, o código sempre será executado. Você perderá o controle da janela do terminal ao executar este código. Você pode interromper um loop infinito pressionando a tecla Ctrl `control-C` no teclado.

Funções

- Embora forneçamos orientações mais detalhadas posteriormente, você pode criar sua própria função em C da seguinte maneira:

```
void meow(void)
{
    printf("meow\n");
}
```

O valor inicial `void` significa que a função não retorna nenhum valor. O `(void)` valor "0" significa que nenhum valor está sendo fornecido à função.

- Essa função pode ser usada na função principal da seguinte forma:

```
// Abstraction

#include <stdio.h>

void meow(void);

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

// Meow once
void meow(void)
{
    printf("meow\n");
}
```

Observe como a `meow` função é chamada com a `meow()` instrução. Isso é possível porque a `meow` função é definida na parte inferior do código e o *protótipo* da função é fornecido na parte superior do código como `void meow(void)` .

- Sua `meow` função pode ser ainda mais modificada para aceitar entradas:

```
// Abstraction with parameterization

#include <stdio.h>

void meow(int n);

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

// Meow some number of times
void meow(int n)
{
```

```
    for (int i = 0; i < n; i++)
    {
        printf("meow\n");
    }
}
```

Observe que o protótipo foi alterado para `void meow(int n)` mostrar que `meow` aceita um `int` como entrada.

- Além disso, podemos obter a entrada do usuário:

```
// User input

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

void meow(int n);

int main(void)
{
    int n;
    do
    {
        n = get_int("Number: ");
    }
    while (n < 1);
    meow(n);
}

// Meow some number of times
void meow(int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("meow\n");
    }
}
```

Observe que `get_int` é usado para obter um número do usuário. `n` é passado para `meow`.

- Podemos até testar para garantir que a entrada fornecida pelo usuário esteja correta:

```
// Return value

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

int get_positive_int(void);
void meow(int n);

int main(void)
{
    int n = get_positive_int();
    meow(n);
}

// Get number of meows
int get_positive_int(void)
{
```

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

// Meow some number of times
void meow(int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("meow\n");
    }
}
```

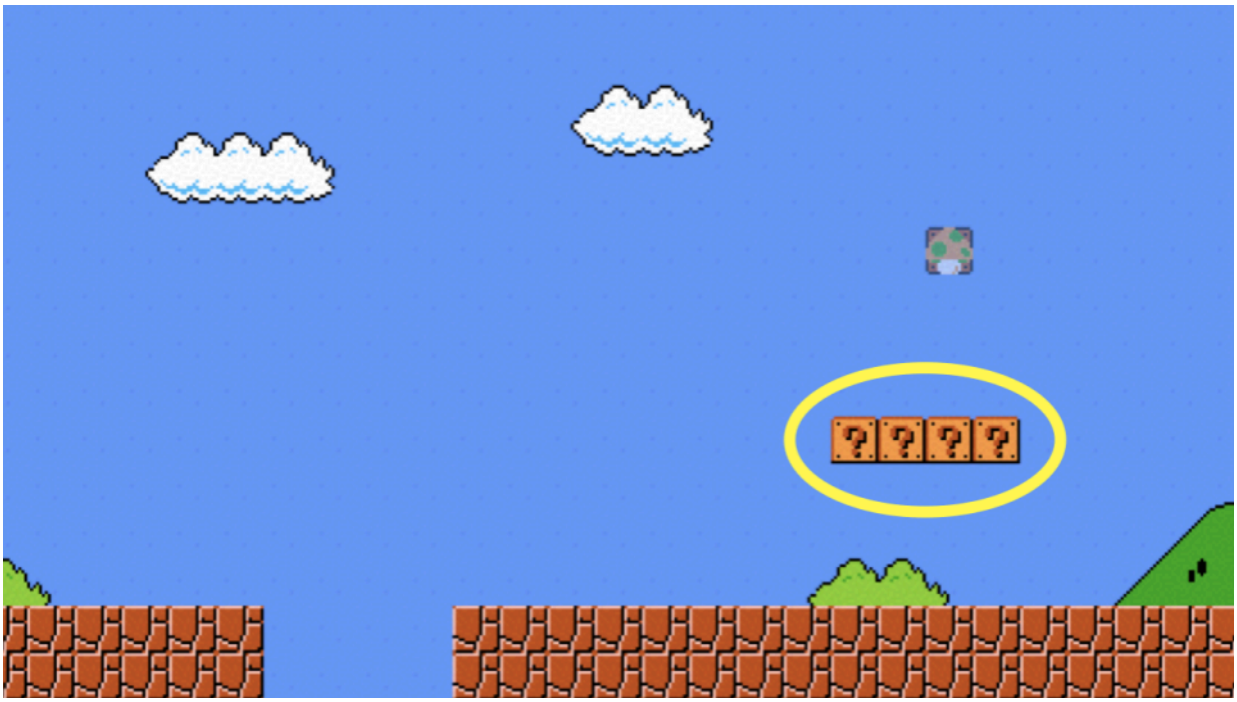
Observe que uma nova função chamada `is` `get_positive_int` solicita ao usuário um número inteiro enquanto `while` `n < 1`. Após obter um número inteiro positivo, essa função retornará `return n` à `main` função `is`.

Correção, Design, Estilo

- O código pode ser avaliado em três eixos.
- Primeiro, *a correção* se refere a "O código funciona como esperado?" Você pode verificar a correção do seu código com `check50`.
- Em segundo lugar, *o design* refere-se a "Quão bem o código foi projetado?" Você pode avaliar o design do seu código usando `design50`.
- Por fim, *estilo* se refere a "Quão esteticamente agradável e consistente é o código?" Você pode avaliar o estilo do seu código com `style50`.

Mario

- Tudo o que discutimos hoje focou em vários elementos fundamentais do seu trabalho como um cientista da computação em ascensão.
- As informações a seguir ajudarão você a se orientar na resolução de problemas para esta disciplina em geral: Como abordar um problema relacionado à ciência da computação?
- Imagine que quiséssemos emular o visual do jogo Super Mario Bros. Considerando os quatro blocos de interrogação na imagem, como poderíamos criar um código que representasse aproximadamente esses quatro blocos horizontais?



- Na janela do terminal, digite `code mario.c` o código da seguinte forma:

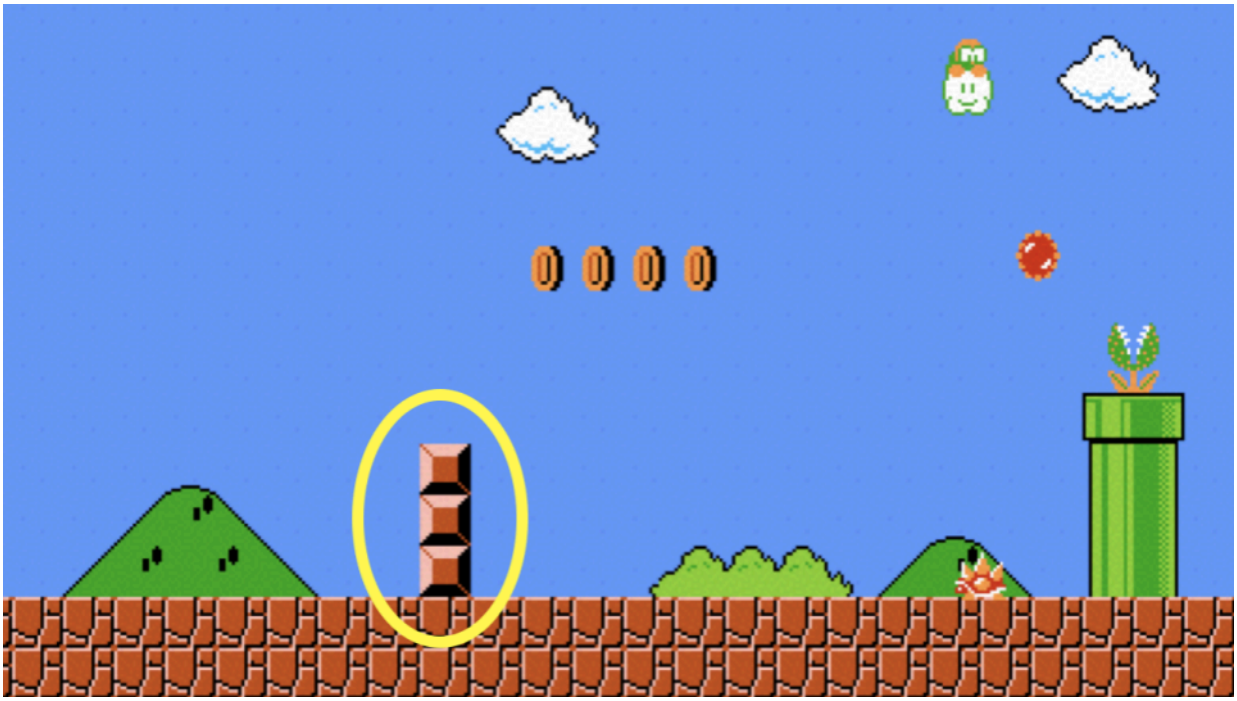
```c
// Prints a row of 4 question marks with a loop

#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 4; i++)
    {
        printf("?");
    }
    printf("\n");
}
```

Observe como quatro pontos de interrogação são impressos aqui usando um laço.

- Da mesma forma, podemos aplicar essa mesma lógica para criar três blocos verticais.



- Para fazer isso, modifique seu código da seguinte forma:

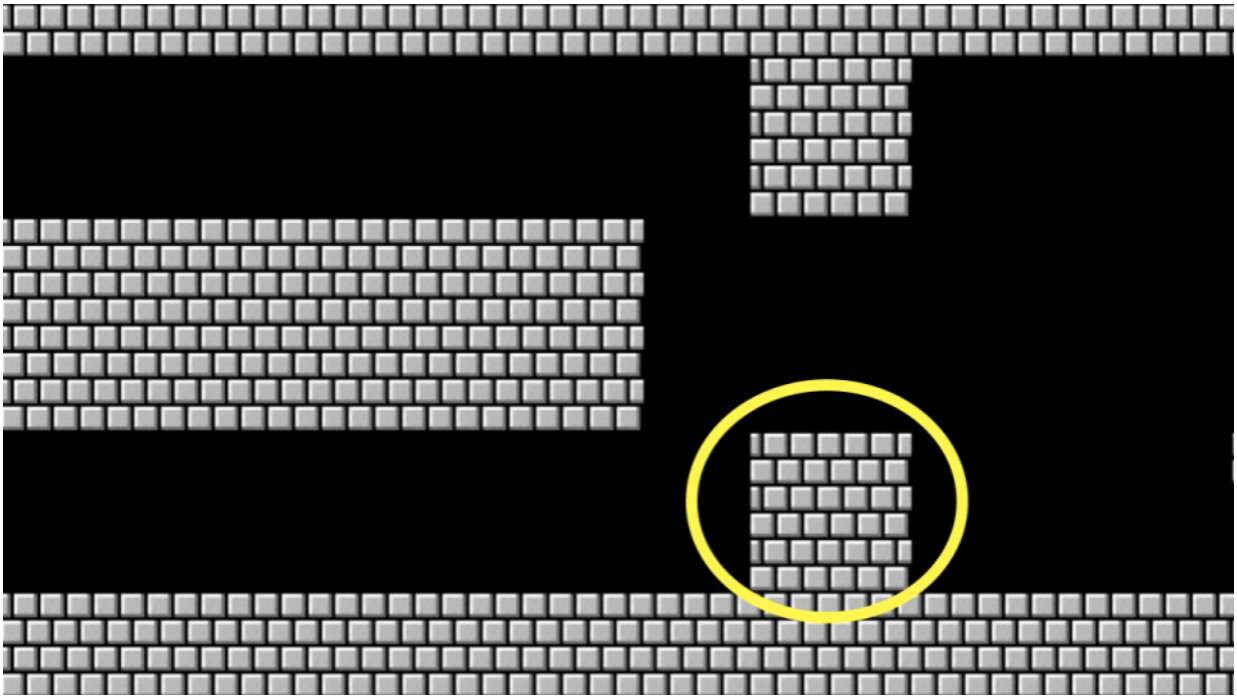
```c
// Prints a column of 3 bricks with a loop

#include <stdio.h>

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

Observe como três blocos verticais são impressos usando um laço.

- E se quiséssemos combinar essas ideias para criar um grupo de blocos de três por três?



- Podemos seguir a lógica acima, combinando as mesmas ideias. Modifique seu código da seguinte forma:

```c
// Prints a 3-by-3 grid of bricks with nested loops

#include <stdio.h>

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

Observe que um laço está dentro de outro. O primeiro laço define qual linha vertical está sendo impressa. Para cada linha, três colunas são impressas. Após cada linha, uma nova linha é impressa.

- E se quiséssemos garantir que o número de blocos fosse *constante* , ou seja, imutável? Modifique seu código da seguinte forma:

```
// Prints a 3-by-3 grid of bricks with nested loops using a constant

#include <stdio.h>

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

Observe como `n` agora é uma constante. Nunca poderá ser alterado.

- Como ilustrado anteriormente nesta aula, podemos *abstrair* funcionalidades em funções. Considere o seguinte código:

```
// Helper function

#include <stdio.h>

void print_row(int width);

int main(void)
{
    const int n = 3;
    for (int i = 0; i < n; i++)
    {
        print_row(n);
    }
}

void print_row(int width)
{
    for (int i = 0; i < width; i++)
    {
        printf("#");
    }
    printf("\n");
}
```

Observe como a impressão de uma linha é realizada por meio de uma nova função.

Comentários

- Os comentários são partes fundamentais de um programa de computador, onde você deixa observações explicativas para si mesmo e para outras pessoas que possam estar colaborando com você em relação ao seu código.
- Todo o código que você criar para este curso deve incluir comentários detalhados.
- Normalmente, cada comentário consiste em algumas palavras ou mais, proporcionando ao leitor a oportunidade de entender o que está acontecendo em um bloco de código específico. Além disso, esses comentários servem como um lembrete para você posteriormente, quando precisar revisar seu código.
- Os comentários consistem em inserir `//` um trecho de código seguido de um comentário. Modifique seu código da seguinte forma para integrar os comentários:

```
// Helper function

#include <stdio.h>

void print_row(int width);

int main(void)
{
    const int n = 3;

    // Print n rows
    for (int i = 0; i < n; i++)
    {
        print_row(n);
    }
}

void print_row(int width)
{
    for (int i = 0; i < width; i++)
    {
        printf("#");
    }
    printf("\n");
}
```

Observe como cada comentário começa com um ponto final `//`.

Mais sobre operadores

- Você pode implementar uma calculadora em C. No seu terminal, digite `code calculator.c` e escreva o código da seguinte forma:

```
// Addition with int

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

```c
int main(void)
{
    // Prompt user for x
    int x = get_int("x: ");

    // Prompt user for y
    int y = get_int("y: ");

    // Add numbers
    int z = x + y;

    // Perform addition
    printf("%i\n", z);
}
```

Observe como a `get_int` função é utilizada para obter um número inteiro do usuário duas vezes. Um número inteiro é armazenado na `int` variável chamada `i` `x`. Outro número inteiro é armazenado na `int` variável chamada `soma` `y`. A soma é armazenada em `soma` `z`. Em seguida, a `printf` função imprime o valor de `soma` `z`, representado pelo `%i` símbolo `\`.

- Também podemos dobrar um número:

```c
// int

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

int main(void)
{
    int dollars = 1;
    while (true)
    {
        char c = get_char("Here's $%i. Double it and give to next person? ", d
        if (c == 'y')
        {
            dollars *= 2;
        }
        else
        {
            break;
        }
    }
    printf("Here's $%i.\n", dollars);
}
```

Ao executar este programa, alguns erros aparentes aparecem `dollars`. Por quê?

- Uma das desvantagens da linguagem C é a facilidade com que ela gerencia a memória. Embora C ofereça um controle imenso sobre como a memória é utilizada, os programadores precisam estar muito atentos às possíveis armadilhas do gerenciamento de memória.
- Os tipos referem-se aos possíveis dados que podem ser armazenados em uma variável. Por exemplo, um tipo `char` é projetado para acomodar um único caractere, como `a` ou `2`.

- Os tipos são muito importantes porque cada tipo tem limites específicos. Por exemplo, devido aos limites de memória, o maior valor `int` possível para um tipo é `4294967295` 1. Se você tentar contar um `int` valor maior que 1, ocorrerá um *estouro de inteiro* , onde um valor incorreto será armazenado nessa variável.
- O número de bits limita o quão alto e baixo podemos contar.
- Isso pode ter impactos catastróficos no mundo real.
- Podemos corrigir isso usando um tipo de dados chamado `long` .

```
// long

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

int main(void)
{
    long dollars = 1;
    while (true)
    {
        char c = get_char("Here's $%li. Double it and give to next person? ",
        if (c == 'y')
        {
            dollars *= 2;
        }
        else
        {
            break;
        }
    }
    printf("Here's $%li.\n", dollars);
}
```

Observe como a execução deste código permitirá o uso de quantias em dólares muito elevadas.

- Os tipos de pessoas com os quais você poderá interagir durante este curso incluem:
  - `bool` , uma expressão booleana que pode ser verdadeira ou falsa
  - `char` , um único caractere como um ou 2
  - `double` , um valor de ponto flutuante com mais dígitos do que um float
  - `float` , um valor de ponto flutuante ou um número real com um valor decimal
  - `int` , números inteiros até um determinado tamanho, ou número de bits
  - `long` , números inteiros com mais bits, portanto podem ter uma contagem maior que um int
  - `string` , uma sequência de caracteres

Truncamento

- Outro problema que pode surgir ao usar tipos de dados é o truncamento.

```c
// Division with ints, demonstrating truncation

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

int main(void)
{
    // Prompt user for x
    int x = get_int("x: ");

    // Prompt user for y
    int y = get_int("y: ");

    // Divide x by y
    printf("%i\n", x / y);
}
```

Em C, a divisão de um número inteiro por outro número inteiro sempre resultará em um número inteiro. Consequentemente, o código acima frequentemente fará com que os dígitos após a vírgula sejam descartados.

- Isso pode ser resolvido empregando um(a) `float`:

```c
// Floats

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

int main(void)
{
    // Prompt user for x
    float x = get_float("x: ");

    // Prompt user for y
    float y = get_float("y: ");

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

Note que isso resolve alguns dos nossos problemas. No entanto, podemos perceber alguma imprecisão na resposta fornecida pelo programa.

- *A imprecisão em ponto flutuante* ilustra que existem limites para a precisão com que os computadores podem calcular números.
- Ao programar, preste atenção especial aos tipos de variáveis que você está usando para evitar problemas no seu código.
- Analisamos alguns exemplos de desastres que podem ocorrer devido a erros relacionados à digitação.

Resumindo

Nesta lição, você aprendeu como aplicar os conceitos básicos que aprendeu no Scratch à linguagem de programação C. Você aprendeu…

- Como criar seu primeiro programa em C.
- Como usar a linha de comando.
- Sobre funções predefinidas que vêm nativamente com a linguagem C.
- Como usar variáveis, condicionais e laços de repetição.
- Como criar suas próprias funções para simplificar e melhorar seu código.
- Como avaliar seu código em três eixos: correção, design e estilo.
- Como integrar comentários ao seu código.
- Como utilizar tipos e operadores e as implicações de suas escolhas.

Até a próxima!