Isso é CS50

Introdução do CS50 à Ciência da Computação

OpenCourseWare

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

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.clubhouse.com/@davidjmalan) (https://www.facebook.com/dmalan)
(https://github.com/dmalan) (https://www.instagram.com/davidjmalan/)
(https://www.linkedin.com/in/malan/) (https://orcid.org/0000-0001-5338-2522)
(https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan)
(https://www.tiktok.com/@davidjmalan) (https://davidjmalan.t.me/)
(https://twitter.com/davidjmalan)

Aula 1

Bem-vindo!

- Em 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 enquanto você aprende a programar qualquer linguagem de programação.
- Lembre-se de que as máquinas só entendem binário. Onde os humanos escrevem *o código-fonte* , uma lista de instruções para o computador legível por humanos, as máquinas entendem apenas o que

podemos chamar *de código de máquina* . Este código de máquina é um padrão de uns e zeros que produz um efeito desejado.

- Acontece que podemos converter *o código-fonte* usando `machine code` um software muito especial chamado *compilador* . Hoje, apresentaremos a você um compilador que permitirá converter código-fonte na linguagem de programação *C* em código de máquina.
- Hoje, além de aprender como codificar, você aprenderá como escrever um bom código.
- O código pode ser avaliado em três eixos. Primeiro, *a correção* refere-se a "o código é executado como pretendido?" Em segundo lugar, *o design* refere-se a "quão bem o código foi projetado?" Finalmente, *estilo* refere-se a "quão esteticamente agradável e consistente é o código?"

Olá Mundo

- O compilador utilizado para este curso é o *Visual Studio Code* , carinhosamente conhecido como, que pode ser acessado pelo mesmo URL ou simplesmente como *VS Code.*
- Uma das razões mais importantes pela qual utilizamos o VS Code é que ele tem todo o software necessário para o curso já pré-carregado. Este curso e as instruções aqui contidas foram desenvolvidos com o VS Code em mente. Melhor sempre utilizar o VS Code para tarefas neste curso.
- Você pode abrir o VS Code em code.cs50.io (https://code.cs50.io/) .
- 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 região no meio chamada *editor de texto* onde você pode editar seu programa. Finalmente, existe um `command line interface` , conhecido como *CLI* , *linha de comando* ou *janela de terminal* onde podemos enviar comandos para o computador na nuvem.

- Podemos construir seu primeiro programa em C digitando `code hello.c` na janela do terminal. Observe que deliberadamente colocamos todo o nome do arquivo em minúsculas e incluímos a `.c` extensão. Em seguida, no editor de texto que aparece, escreva o código da seguinte forma:

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

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

Observe que cada caractere acima serve a um propósito. Se você digitá-lo incorretamente, o programa não será executado.

- Clicando novamente na janela do terminal, você pode compilar seu código executando `make hello`. Observe que estamos omitindo `.c`. `make` é um compilador que irá procurar nosso `hello.c` arquivo e transformá-lo em um programa chamado `hello`. Se a execução deste comando não resultar em erros, você poderá prosseguir. Caso contrário, verifique novamente seu código para garantir que ele corresponda ao acima.
- Agora, digite `./hello` e seu programa executará dizendo `hello, world`.
- Agora, abra o *explorador de arquivos* à esquerda. Você notará que agora há um arquivo chamado `hello.c` e outro arquivo chamado `hello`. `hello.c` pode ser lido pelo compilador: é onde seu código é armazenado. `hello` é um arquivo executável que você pode executar, mas não pode ser lido pelo compilador.
- Vejamos nosso código com mais atenção:

```c
#include <stdio.h>

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

Observe que nosso código é destacado usando realce de sintaxe.

Funções

- No Scratch, utilizamos 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 esta função:

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

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

- Um erro comum na programação C é a omissão de um ponto e vírgula. Modifique seu código da seguinte maneira:

```c
#include <stdio.h>

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

Observe que o ponto-e-vírgula desapareceu.

- Na janela do terminal, execute `make hello`. Agora você encontrará vários erros! Colocando o ponto e vírgula de volta na posição correta e executando `make hello` novamente, os erros desaparecem.

- Observe também o símbolo especial `\n` em seu código. Tente remover esses caracteres e *fazer* seu programa novamente executando `make hello`. Digitando `./hello` na janela do terminal, como seu programa mudou?
- Restaure seu programa para o seguinte:

```
#include <stdio.h>

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

  Observe o ponto e vírgula e `\n` foram restaurados.
- A instrução no início do código `#include <stdio.h>` é um comando muito especial que informa à compilação que você deseja usar os recursos da *biblioteca* chamada `stdio.h`. Isso permite que você, entre muitas outras coisas, utilize a `printf` função. Você pode ler sobre todos os recursos desta biblioteca nas páginas do manual (https://manual.cs50.io) .
- Acontece que o CS50 tem sua própria biblioteca chamada `cs50.h`. Vamos usar esta biblioteca em seu programa.

Variáveis

- Lembre-se que no Scratch tínhamos a possibilidade de perguntar ao usuário “Qual é o seu nome?” e diga "olá" com esse nome anexado a ele.
- Em C, podemos fazer o mesmo. Modifique seu código da seguinte maneira:

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

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

  Notice that `#include <cs50.h>` has been added to the top of your code. The `get_string` function is used to get a string from the user. Then, the variable `answer` is passed to the `printf` function. `%s` tells the `printf` function to prepare itself to receive a `string`.
- `answer` is a special holding place we call a *variable*. `answer` is of type `string` and can hold any string within it. There are many *data types*, such as `int`, `bool`, `char`, and many others.
- Running `make hello` again in the terminal window, you can run your program by typing `./hello`. The program now asks for your name and then says hello with your name attached.

Conditionals

- Another building block you utilized within Scratch was that of *conditionals*. For example, you might want to do one thing if x is greater than y. Further, you might want to do something else if that condition is not met.
- In the terminal window, type `code compare.c` and write code as follows:

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

int main(void)
{
    int x = get_int("What's x? ")
    int y = get_int("What's y? ")

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

Notice that we create two variables, an `int` or integer called `x` and another called `y`. The values of these are populated using the `get_int` function.

- You can run your code by executing `make compare` in the terminal window, followed by `./compare`. If you get any error messages, check your code for errors.
- We can improve your program by coding as follows:

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

int main(void)
{
    int x = get_int("What's x? ")
    int y = get_int("What's y? ")

    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");
    }
}
```

Notice that all potential outcomes are now accounted for.

- You can re-make and re-run your program and test it out.
- Considering another data type called a `char` we can start a new program by typing `code agree.c` into the terminal window. In the text editor, write code as follows:

```c
#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");
    }
}
```

Notice that single quotes are utilized for single characters. Further, notice that `==` ensure that something *is equal* to something else, where a single equal sign would have a very different function in C. Finally, notice that `||` effectively means *or*.

- You can test your code by typing `make agree` into the terminal window, followed by `./agree`.

Loops

- We can also utilize the loops building block from Scratch in our C programs.
- In your terminal window, type `code meow.c` and write code as follows:

```
#include <stdio.h>

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

Notice this does as intended but has an opportunity for better design.

- We can improve our program by modifying your code as follows:

```
#include <stdio.h>

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

Notice that we create an `int` called `i` and assign it the value `0`. Then, we create a `while` loop that will run as long as `i < 3`. Then, the loop runs. Every time `1` is added to `i` using the `i++` statement.

- Similarly, we can implement a count-down of sorts by modifying our code as follows:

```
#include <stdio.h>

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

Notice how our counter `i` is started at `3`. Each time the loop runs, it will reduce the counter by `1`. Once the counter is less than zero, it will stop the loop.

- We can further improve the design using a `for` loop. Modify your code as follows:

```
#include <stdio.h>

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

Notice that the `for` loop includes three arguments. The first argument `int i = 0` starts our counter at zero. The second argument `i < 3` is the condition that is being checked. Finally, the argument `i++` tells the loop to increment by one each time the loop runs.

- We can even loop forever using the following code:

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

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

Notice that `true` will always be the case. Therefore, the code will always run. You will lose control of your terminal window by running this code. You can break from an infinite by hitting `control-C` on your keyboard.

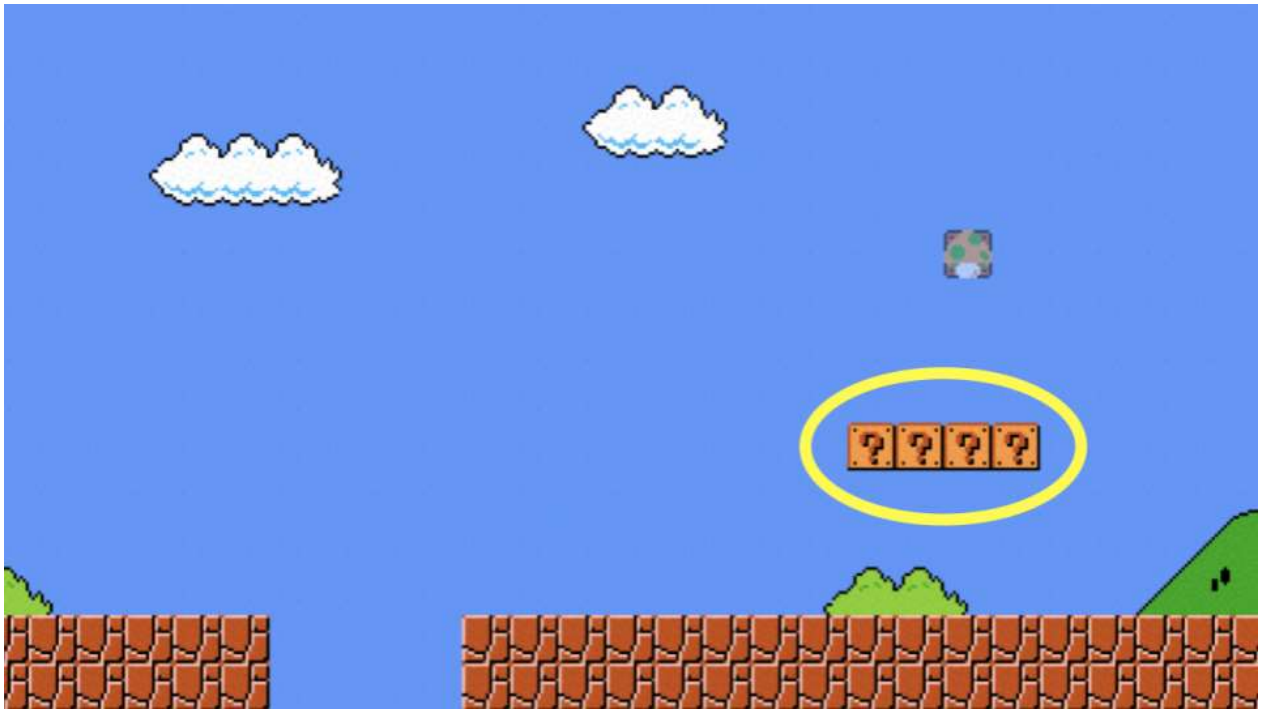
Linux and the Command Line

- *Linux* is an operating system that is accessible via the command line in the terminal window in VS Code.
- Some common command-line arguments we may use include:
  - `cd`, for changing our current directory (folder)
  - `cp`, for copying files and directories
  - `ls`, for listing files in a directory
  - `mkdir`, for making a directory
  - `mv`, for moving (renaming) files and directories
  - `rm`, for removing (deleting) files
  - `rmdir`, for removing (deleting) directories
- The most commonly used is `ls` which will list all the files in the current directory or directory. Go ahead and type `ls` into the terminal window and hit `enter`. You'll see all the files in the current folder.

- Another useful command is `mv`, where you can move a file from one file to another. For example, you could use this command to rename `Hello.c` (notice the uppercase `H`) to `hello.c` by typing `mv Hello.c hello.c`.
- You can also create folders. You can type `mkdir pset1` to create a directory called `pset1`.
- You can then use `cd pset1` to change your current directory to `pset1`.

Mario

- Everything we've discussed today has focused on various building-blocks of your work as a programmer.
- The following will help you orient toward working on a problem set for this class in general: How does one approach a computer science related problem?
- Imagine we wanted to emulate the visual of the game Super Mario Bros. Considering the four question-blocks pictured, how could we create code that roughly represents these four horizontal blocks?



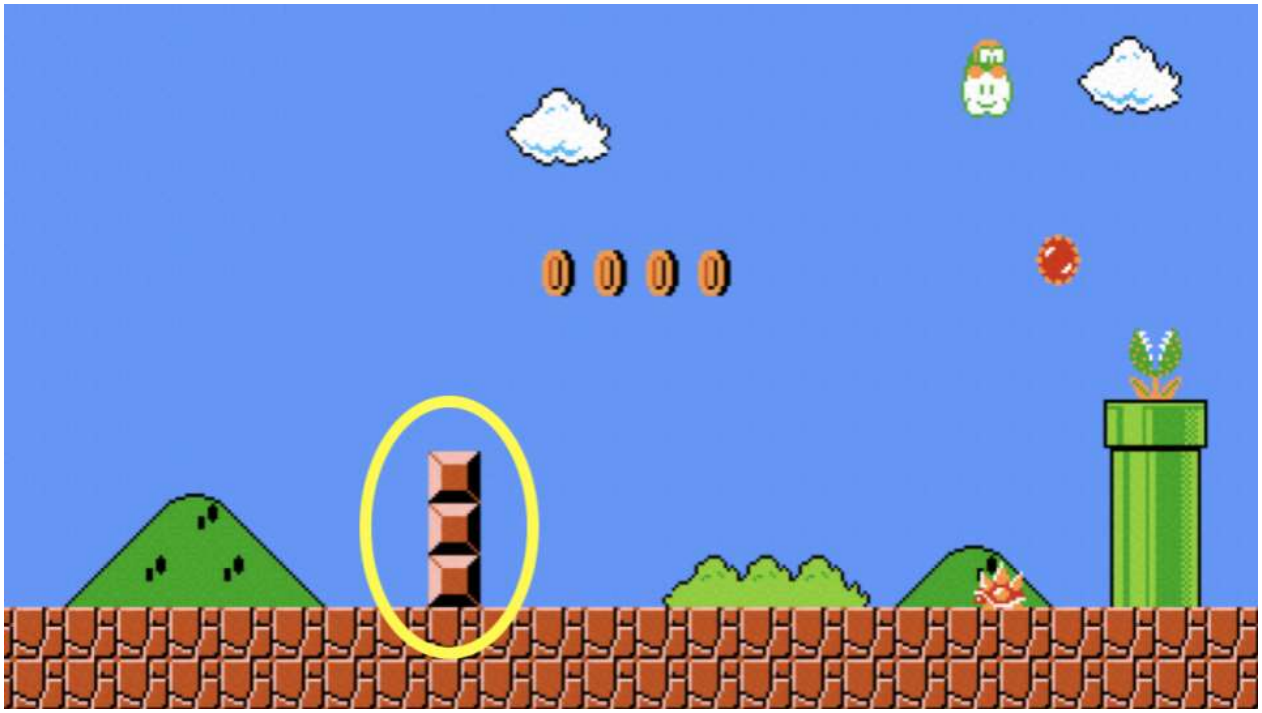
- In the terminal window, type `code mario.c` and code as follows:

```c
#include <stdio.h>

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

Notice how four question marks are printed here using a loop.

- Similarly, we can apply this same logic to be able to create three vertical blocks.



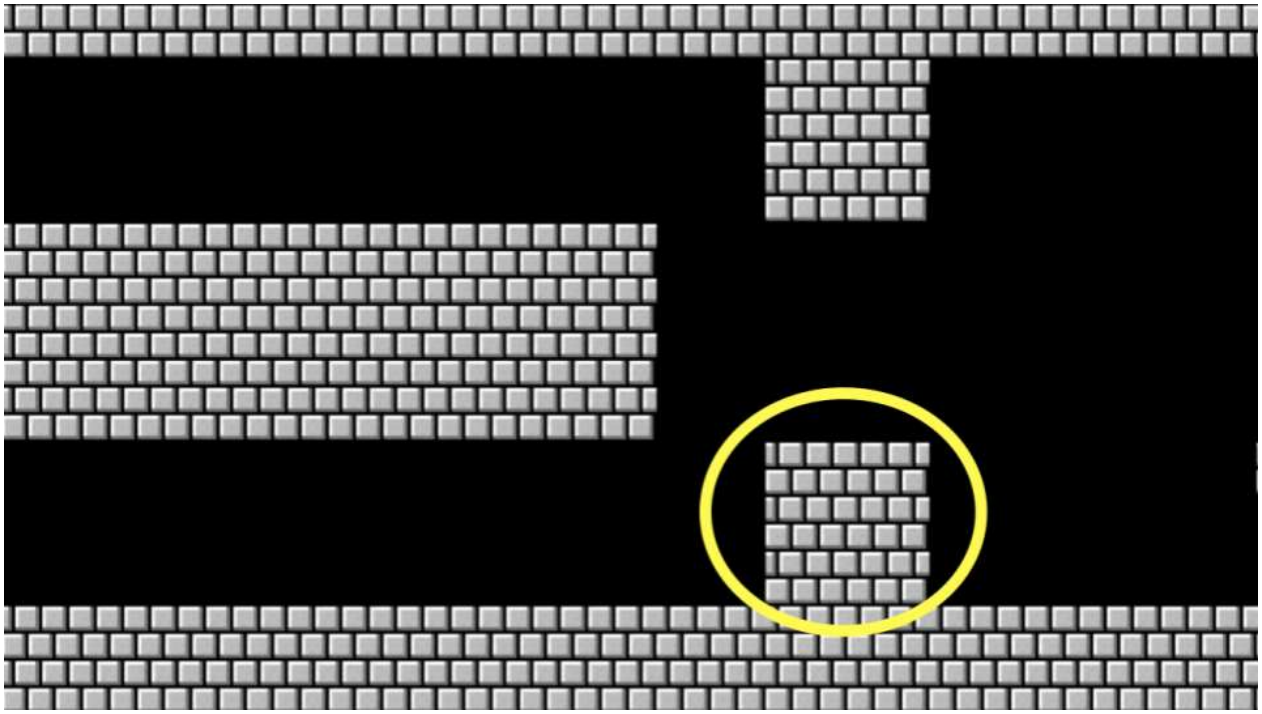
- To accomplish this, modify your code as follows:

```
#include <stdio.h>

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

Notice how three vertical bricks are printed using a loop.

- What if we wanted to combine these ideas to create a three-by-three group of blocks?



- We can follow the logic above, combining the same ideas. Modify your code as follows:

```c
#include <stdio.h>

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

Notice that one loop is inside another. The first loop defines what vertical row is being printed. For each row, three columns are printed. After each row, a new line is printed.

- What if we wanted to ensure that the number of blocks to be *constant*, that is, unchangeable? Modify your code as follows:

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

Notice how `n` is now a constant. It can never be changed.

- As illustrated earlier in this lecture, we can make our code prompt the user for the size of the grid. Modify your code as follows:

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

int main(void)
{
    int n = get_int("Size: ");

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

Notice that `get_int` is used to prompt the user.

- A general piece of advice within programming is that you should never fully trust your user. They will likely misbehave, typing incorrect values where they should not. We can protect our program from bad behavior by checking to make sure the user's input satisfies our needs. Modify your code as follows:

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

```c
int main(void)
{
    int n;
    do
    {
        n = get_int("Size: ");
    }
    while (n < 1);

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

Notice how the user is continuously prompted for the size until the user's input is 1 or greater.

Comments

- Comments are fundamental parts of a computer program, where you leave explanatory remarks to yourself and others that may be collaborating with you regarding your code.
- All code you create for this course must include robust comments.
- Typically each comment is a few words or more, providing the reader an opportunity to understand what is happening in a specific block of code. Further, such comments serve as a reminder for you later when you need to revise your code.
- Comments involve placing `//` into your code, followed by a comment. Modify your code as follows to integrate comments:

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

int main(void)
{
    // Get size of grid
    int n;
    do
    {
        n = get_int("Size: ");
    }
    while (n < 1);

    // Print grid of bricks
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("#");
        }
        printf("\n");
    }
}
```

Notice how each comment begins with a `//`.

Abstraction

- *Abstraction* is the art of simplifying our code such that it deals with smaller and smaller problems.
- Looking at your code, you can see how two essential problems in our code are *get size of grid* and *print grid of bricks*.
- We can abstract away these two problems into separate functions. Modify your code as follows:

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

int get_size(void);
void print_grid(int n);

int main(void)
{
    int n = get_size();
    print_grid(n);
}

int get_size(void)
{
    int n;
    do
    {
        n = get_int("Size: ");
    }
    while (n < 1);
    return n;
}

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

Notice that we have three functions now. First, we have the `main` function that calls two other functions called `get_size` and `print_grid`. Second, we have a second function called `get_size` which includes the exact code we had to accomplish this task prior. Third, we have another function called `print_grid` that prints the grid. Because we abstracted away the essential problems within our program, our `main` function is very short.

Operators and Types

- *Operators* refer to the mathematical operations that are supported by your compiler. In C, these mathematical operators include:
  - `+` for addition
  - `-` for subtraction

    - `*` for multiplication
    - `/` for division
    - `%` for remainder
- Types refer to the possible data that can be stored within a variable. For example, a `char` is designed to accommodate a single character like `a` or `2`.
- Types are very important because each type has specific limits. For example, because of the limits in memory, the highest value of an `int` can be `4294967296`.
- Types with which you might interact during this course include:
    - `bool`, a Boolean expression of either true or false
    - `char`, a single character like a or 2
    - `double`, a floating-point value with more digits than a float
    - `float`, a floating-point value, or real number with a decimal value
    - `int`, integers up to a certain size, or number of bits
    - `long`, integers with more bits, so they can count higher than an int
    - `string`, a string of characters
- You can implement a calculator in C. In your terminal, type `code calculator.c` and write code as follows:

```c
#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: ");

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

  Notice how the `get_int` function is utilized to obtain an integer from the user twice. One integer is stored in the `int` variable called `x`. Another is stored in the `int` variable called `y`. Then, the `printf` function prints the value of `x + y`, designated by the `%i` symbol.
- As you are coding, pay special attention to the types of variables you are using to avoid problems within your code.

Summing Up

In this lesson, you learned how to apply the building blocks you learned in Scratch to the C programming language. You learned...

- How to create your first program in C.
- Predefined functions that come natively with C and how to implement your own functions.
- Como usar variáveis, condicionais e loops.
- Como usar a linha de comando do Linux.

- Como abordar a solução de problemas para um problema de ciência da computação.
- Como integrar comentários em seu código.
- Como abordar a abstração para simplificar e melhorar seu código.
- Como utilizar tipos e operadores.

Vejo você na próxima vez!