

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>) 

([@davidjmalan](https://www.threads.net/@davidjmalan))  (<https://twitter.com/davidjmalan>)

Aula 2

- [Bem-vindo!](#)
- [Níveis de leitura](#)
- [Compilando](#)
- [Depuração](#)
- [Matrizes](#)
- [Cordas](#)
- [Comprimento da corda](#)
- [Argumentos da linha de comando](#)
- [Status de saída](#)
- [Criptografia](#)
- [Resumindo](#)

Bem-vindo!

- Na nossa sessão anterior, aprendemos sobre C, uma linguagem de programação baseada em texto.
- Esta semana, vamos analisar mais detalhadamente os elementos fundamentais adicionais que darão suporte aos nossos objetivos de aprender mais sobre programação desde a base.

- Fundamentalmente, além dos fundamentos da programação, este curso aborda a resolução de problemas. Portanto, também nos concentraremos em como abordar problemas de ciência da computação.
- Ao final do curso, você aprenderá como usar esses elementos básicos mencionados anteriormente para resolver uma ampla gama de problemas de ciência da computação.

Níveis de leitura

- Um dos problemas do mundo real que abordaremos neste curso é a compreensão dos níveis de leitura.
- Com a ajuda de alguns de seus colegas, apresentamos textos em vários níveis de leitura.
- Nesta semana, iremos quantificar os níveis de leitura como um dos seus muitos desafios de programação.

Compilando

- *Criptografia* é o ato de ocultar um texto simples de olhares curiosos. *Descriptografia*, por sua vez, é o ato de pegar um texto criptografado e transformá-lo em um texto legível para humanos.
- Um texto criptografado pode ter a seguinte aparência:

U I J T J T D T 5 0
- Lembre-se que na semana passada você aprendeu sobre um *compilador*, um programa de computador especializado que converte o *código-fonte* em *código de máquina*, que pode ser entendido por um computador.
- Por exemplo, você pode ter um programa de computador com a seguinte aparência:

```
#include <stdio.h>

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

- Um compilador pegará o código acima e o transformará no seguinte código de máquina:

```

01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000 00000001 00000000 00000000 00000000
10110000 00000101 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11010000 00010011 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01000000 00000000 00111000 00000000
00001001 00000000 01000000 00000000 00100100 00000000 00100001 00000000
00000110 00000000 00000000 00000000 00000101 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
00001000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00000100 00000000 00000000 00000000
00111000 00000010 00000000 00000000 00000000 00000000 00000000 00000000
...

```

- O VS Code , o ambiente de programação fornecido a você como aluno do CS50, utiliza um compilador chamado c clang ou linguagem c .
- Você pode digitar o seguinte na janela do terminal para compilar seu código: clang -o hello hello.c .
- Os argumentos da linha de comando são fornecidos na linha de comando clang como -o hello hello.c .
- Ao executar ./hello o programa na janela do terminal, ele funciona conforme o esperado.
- Considere o seguinte código da semana passada:

```

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

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

```

- Para compilar este código, você pode digitar clang -o hello hello.c -lcs50 .
- Se você digitar make hello , será executado um comando que utiliza o clang para criar um arquivo de saída que você poderá executar como usuário.
- O VS Code já vem pré-programado para make executar diversos argumentos de linha de comando juntamente com o clang, para sua conveniência como usuário.
- Embora o exemplo acima seja apenas uma ilustração para que você possa compreender melhor o processo e o conceito de compilação de código, usá-lo make no CS50 é perfeitamente aceitável e até esperado!
- A compilação envolve quatro etapas principais, incluindo as seguintes:
- Primeiramente, o pré-processamento consiste em copiar e colar os arquivos de cabeçalho do seu código, indicados por um prefixo # (como `<header>`), no seu arquivo `<header>` .

Durante essa etapa, o código de `<header>` é copiado para o seu programa. Da mesma forma que o seu código contém `<header>` , o código contido em algum lugar do seu computador é copiado para o seu programa. Essa etapa pode ser visualizada da seguinte forma:

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

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

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

- Em segundo lugar, *a compilação* é a etapa em que seu programa é convertido em código assembly. Essa etapa pode ser visualizada da seguinte forma:

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

- Em terceiro lugar, *a montagem* envolve o compilador convertendo seu código assembly em código de máquina. Essa etapa pode ser visualizada da seguinte forma:

```
01111111010001010100110001000110  
0000001000000001000000010000000  
00000000000000000000000000000000  
00000000000000000000000000000000  
00000001000000000011111000000000  
0000000100000000000000000000000000  
0000000000000000000000000000000000
```

00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
10100000000000100000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
01000000000000000000000000000000
00000000000000000000000000000000
00001010000000000000000000000000
01010101010010001000100111100101
010010001000001111101100000010000
001100011100000010001001110000111
0100100010111110000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
1110100000000000000000000000000000
000000000100100010111111000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
...

• • •

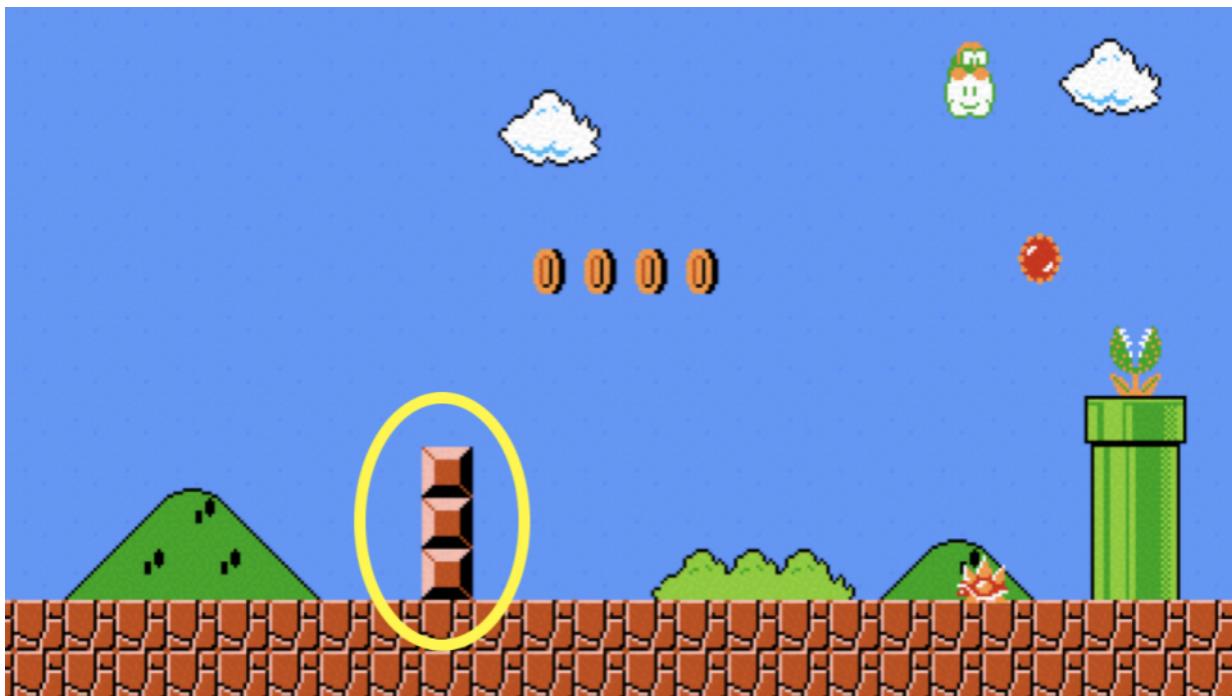
- Finalmente, durante a etapa *de vinculação*, o código das bibliotecas incluídas também é convertido em código de máquina e combinado com o seu código. O arquivo executável final é então gerado.

Depuração

- Todos cometem erros ao programar.
 - *Depurar* é o processo de localizar e remover erros do seu código.
 - Uma das técnicas de depuração que você usará neste curso para depurar seu código é chamada de *depuração com patinho de borracha*, onde você pode conversar com um objeto inanimado (ou consigo mesmo) para ajudar a entender o seu código e por que ele não está funcionando como esperado. Quando estiver com dificuldades com o seu código,

considere a possibilidade de falar em voz alta, literalmente, com um patinho de borracha sobre o problema. Se preferir não falar com um pequeno patinho de plástico, fique à vontade para falar com uma pessoa próxima!

- Criamos o CS50 Duck e [o CS50.ai \(<https://cs50.ai>\)](https://cs50.ai) como ferramentas que podem ajudá-lo a depurar seu código.
- Considere a seguinte imagem da semana passada:



- Considere o seguinte código que contém um bug inserido propositalmente:

```
// Buggy example for printf

#include <stdio.h>

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

Observe que este código imprime quatro blocos em vez de três.

- Digite `code buggy0.c` o código acima na janela do terminal.
- Ao executar este código, quatro tijolos aparecem em vez dos três pretendidos.
- `printf` É uma forma muito útil de depurar seu código. Você pode modificar seu código da seguinte maneira:

```
// Buggy example for printf

#include <stdio.h>

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

```

    {
        printf("i is %i\n", i);
        printf("#\n");
    }
}

```

Observe como este código exibe o valor de `x` `i` durante cada iteração do loop, permitindo-nos depurar o código.

- Ao executar este código, você verá diversas instruções, incluindo `i is 0` <stdio.h>`, `` i` `is 1 <stdout.h>` `i is 2` <stdin.h>` e `` <stdin.h i is 3``. Ao ver isso, você poderá perceber que outras partes do código precisam ser corrigidas da seguinte forma:

```

#include <stdio.h>

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

```

Observe que o `<=` foi substituído por `<`.

- Este código pode ser ainda mais aprimorado da seguinte forma:

```

// Buggy example for debug50

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

void print_column(int height);

int main(void)
{
    int h = get_int("Height: ");
    print_column(h);
}

void print_column(int height)
{
    for (int i = 0; i <= height; i++)
    {
        printf("#\n");
    }
}

```

Note que compilar e executar esse código ainda resulta em um erro.

- Para corrigir esse erro, usaremos uma nova ferramenta.
- Uma segunda ferramenta na depuração é chamada de *depurador*, uma ferramenta de software criada por programadores para ajudar a rastrear erros no código.
- No VS Code, um depurador pré-configurado já está disponível para você.

- Para utilizar este depurador, primeiro defina um *ponto de interrupção* clicando à esquerda de uma linha do seu código, logo à esquerda do número da linha. Ao clicar, você verá um ponto vermelho aparecer. Imagine isso como uma placa de pare, pedindo ao compilador para pausar para que você possa analisar o que está acontecendo nesta parte do seu código.

```

1 #include <cs50.h>
2 #include <stdio.h>
3
4 void print_column(int height);
5
6 int main(void)
7 {
8     int h = get_int("Height: ");
9     print_column(h);
10 }
11
12 void print_column(int height)
13 {
14     for (int i = 0; i <= height; i++)

```

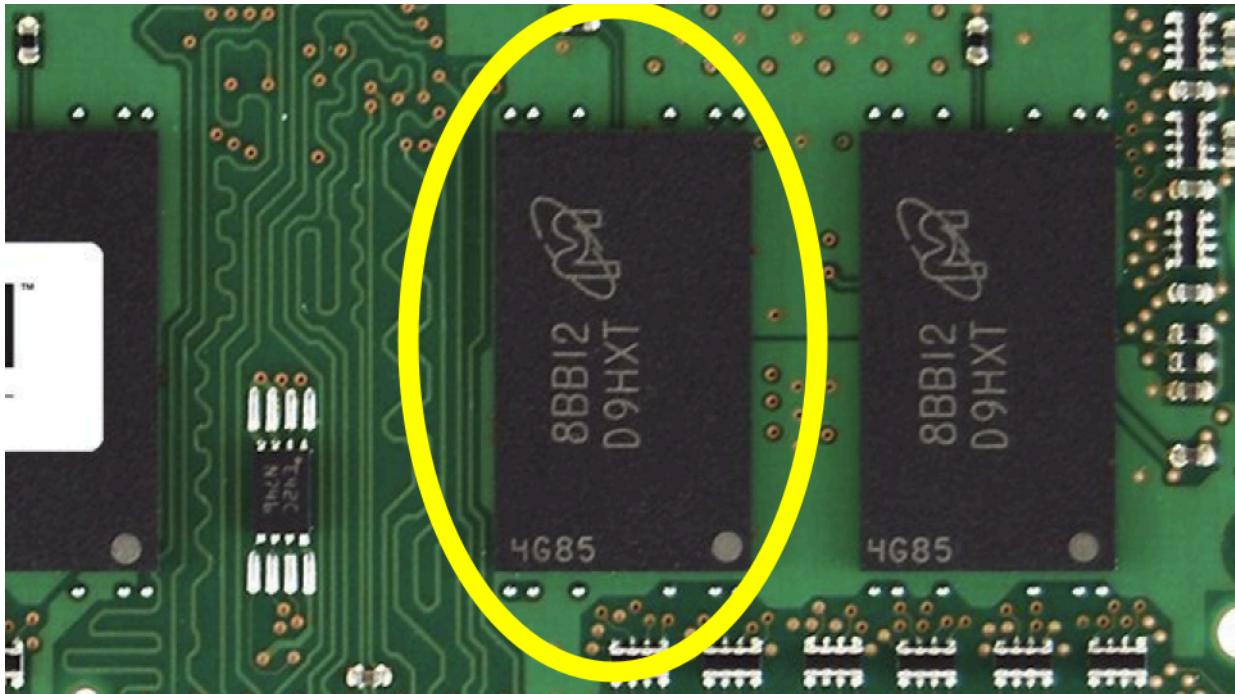
- Em seguida, execute o comando `debug50 ./buggy0`. Você notará que, após o depurador ser ativado, uma linha do seu código será destacada com uma cor dourada. Literalmente, o código pausou *nessa* linha. Observe no canto superior esquerdo como todas as variáveis locais são exibidas, incluindo `x` `h`, que atualmente não possui um valor. Na parte superior da janela, você pode clicar no `step over` botão e o depurador continuará percorrendo o código. Observe como o valor de `h` aumenta.
- Embora esta ferramenta não mostre onde está o erro, ela ajudará você a analisar o código em detalhes, passo a passo. Você pode usar `step into`-la para examinar mais a fundo o código com erros.

Matrizes

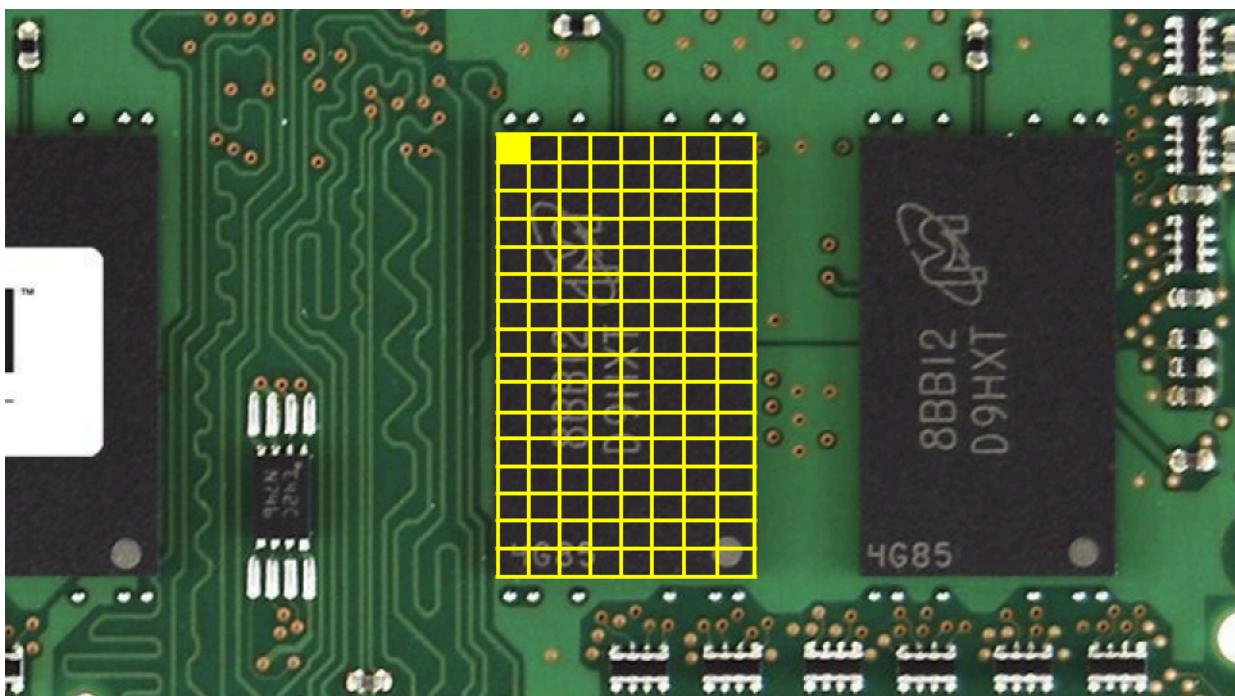
- Na Semana 0, falamos sobre *tipos de dados* como `bool`, `int`, `char`, `string`, etc.
- Cada tipo de dado requer uma certa quantidade de recursos do sistema:
 - `bool` 1 byte

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

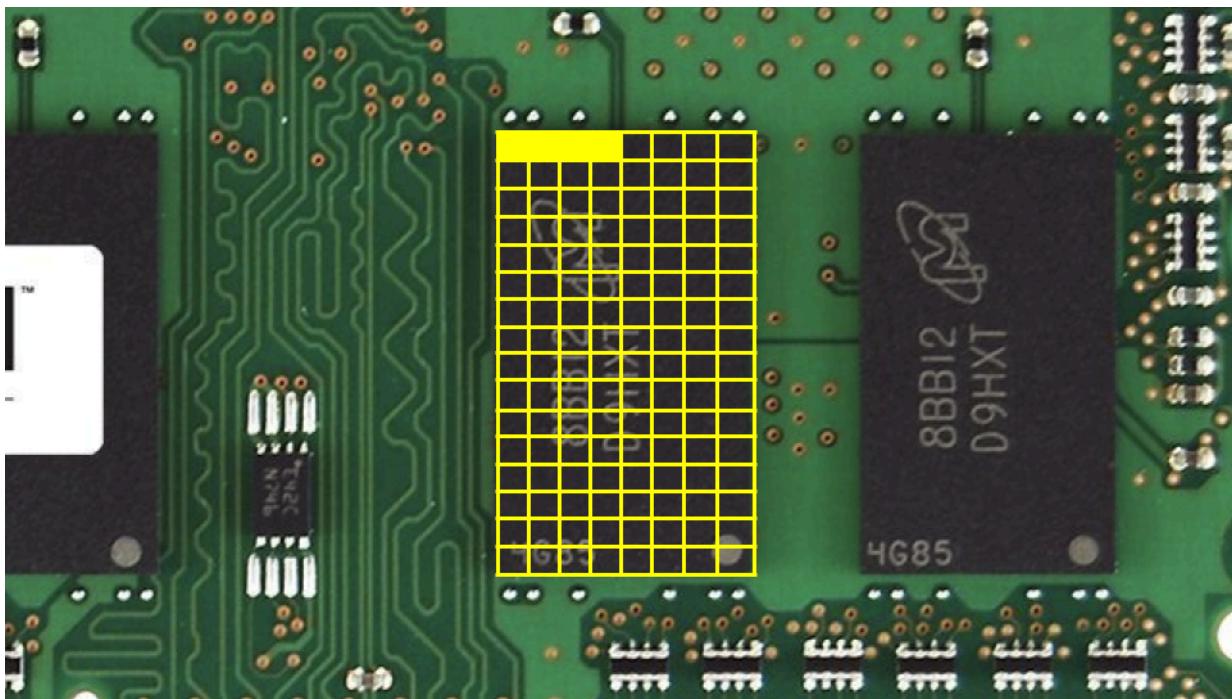
- Dentro do seu computador, você tem uma quantidade finita de memória disponível.



- Fisicamente, na memória do seu computador, você pode imaginar como tipos específicos de dados são armazenados. Você pode imaginar que um `string` `char`, que requer apenas 1 byte de memória, tenha a seguinte aparência:



- Da mesma forma, um `int`, que requer 4 bytes, pode ter a seguinte aparência:



- Podemos criar um programa que explore esses conceitos. No seu terminal, digite `code scores.c` e escreva o código da seguinte forma:

```
// Averages three (hardcoded) numbers

#include <stdio.h>

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

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

Observe que o número à direita é um valor de ponto flutuante de `3.0`, de modo que o cálculo é apresentado como um valor de ponto flutuante no final.

- Executando `make scores`, o programa é executado.
- Você pode imaginar como essas variáveis são armazenadas na memória:

| | | | | | |
|---------------------|---------------------|--|--|--|--|
| 72 score1 | 73 score2 | | | | |
| 33 score3 | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

- Arrays são uma sequência de valores armazenados sequencialmente na memória.
- `int scores[3]` é uma forma de instruir o compilador a fornecer três espaços de memória consecutivos com tamanho suficiente `int` para armazenar três valores `scores`. Considerando nosso programa, você pode revisar seu código da seguinte forma:

```
// Averages three (hardcoded) numbers using an array

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

int main(void)
{
    // Scores
    int scores[3];
    scores[0] = 72;
    scores[1] = 73;
    scores[2] = 33;

    // Print average
    printf("Average: %f\n", (scores[0] + scores[1] + scores[2]) / 3.0);
}
```

Observe que o `score[0]` programa examina o valor nesse local da memória através `indexing into` do array chamado `scores`local`` `0` para verificar qual valor está armazenado ali.

- Você pode ver que, embora o código acima funcione, ainda há espaço para melhorá-lo. Revise seu código da seguinte forma:

```
// Averages three numbers using an array and a loop

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

int main(void)
{
```

```

    // Get scores
    int scores[3];
    for (int i = 0; i < 3; i++)
    {
        scores[i] = get_int("Score: ");
    }

    // Print average
    printf("Average: %f\n", (scores[0] + scores[1] + scores[2]) / 3.0);
}

```

Observe como indexamos `scores` usando `scores[i]` `where`, que `i` é fornecido pelo `for` loop.

- Podemos simplificar ou *abstrair* o cálculo da média. Modifique seu código da seguinte forma:

```

// Averages three numbers using an array, a constant, and a helper function

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

// Constant
const int N = 3;

// Prototype
float average(int length, int array[]);

int main(void)
{
    // Get scores
    int scores[N];
    for (int i = 0; i < N; i++)
    {
        scores[i] = get_int("Score: ");
    }

    // Print average
    printf("Average: %f\n", average(N, scores));
}

float average(int length, int array[])
{
    // Calculate average
    int sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += array[i];
    }
    return sum / (float) length;
}

```

Observe que uma nova função chamada `f` `average` foi declarada. Além disso, observe que um `const` valor constante `f` ou `i` `N` foi declarado. Mais importante ainda, observe como a `average` função recebe um `int array[]` valor `i`, o que significa que o compilador passa um array para essa função.

- Os arrays não servem apenas como contêineres: eles podem ser passados entre funções.

Cordas

- A `string` é simplesmente uma matriz de variáveis do tipo `char`: uma matriz de caracteres.
- Para explorar `char` e `string`, digite `code hi.c` na janela do terminal e escreva o código da seguinte forma:

```
// Prints chars

#include <stdio.h>

int main(void)
{
    char c1 = 'H';
    char c2 = 'I';
    char c3 = '!';

    printf("%c%c%c\n", c1, c2, c3);
}
```

Observe que isso produzirá uma sequência de caracteres.

- Da mesma forma, faça a seguinte modificação em seu código:

```
#include <stdio.h>

int main(void)
{
    char c1 = 'H';
    char c2 = 'I';
    char c3 = '!';

    printf("%i %i %i\n", c1, c2, c3);
}
```

Observe que os códigos ASCII são impressos substituindo `%c` por `%i`.

- Considerando a imagem a seguir, você pode ver como uma string é uma matriz de caracteres que começa com o primeiro caractere e termina com um caractere especial chamado ponto e vírgula (:) `NUL character`.

| | | | | | | |
|-----------|-----------|-----------|------------|--|--|--|
| H s[0] | I s[1] | ! s[2] | \0 s[3] | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

- Imaginando isso em decimal, seu array teria a seguinte aparência:

| | | | | | | |
|------------|------------|------------|-----------|--|--|--|
| 72 s[0] | 73 s[1] | 33 s[2] | 0 s[3] | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

- Para entender melhor como `string` funciona, revise seu código da seguinte forma:

```
// Treats string as array

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

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

Observe como a `printf` declaração apresenta três valores do nosso array chamado `s`.

- Como antes, podemos substituir `%c` da `%i` seguinte forma:

```
// Prints string's ASCII codes, including NUL

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

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

Observe que isso imprime os códigos ASCII da string, incluindo NUL.

- Vamos imaginar que queremos dizer " HI!" e "BYE!". Modifique seu código da seguinte forma:

```
// Multiple strings

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

int main(void)
{
    string s = "HI!";
    string t = "BYE!";

    printf("%s\n", s);
    printf("%s\n", t);
}
```

Observe que duas strings são declaradas e utilizadas neste exemplo.

- Você pode visualizar isso da seguinte forma:

| | | | | | | | |
|------------|-----------|-----------|------------|-----------|-----------|-----------|-----------|
| H s[0] | I s[1] | ! s[2] | \0 s[3] | B t[0] | Y t[1] | E t[2] | ! t[3] |
| \0 t[4] | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

- Podemos aprimorar ainda mais esse código. Modifique seu código da seguinte forma:

```
// Array of strings
```

```

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

int main(void)
{
    string words[2];

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

    printf("%s\n", words[0]);
    printf("%s\n", words[1]);
}

```

Observe que ambas as strings estão armazenadas em um único array do tipo `string`.

- Podemos consolidar nossas duas strings em um array de strings.

```

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

int main(void)
{
    string words[2];

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

    printf("%c%c%c\n", words[0][0], words[0][1], words[0][2]);
    printf("%c%c%c%c\n", words[1][0], words[1][1], words[1][2], words[1][3]);
}

```

Observe que um array `words` de strings é criado. Cada palavra é armazenada em `words`.

Comprimento da corda

- Um problema comum em programação, e talvez mais especificamente em C, é descobrir o comprimento de um array. Como podemos implementar isso em código? Digite `code length.c` o seguinte código na janela do terminal:

```

// Determines the length of a string

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

int main(void)
{
    // Prompt for user's name
    string name = get_string("Name: ");

    // Count number of characters up until '\0' (aka NUL)
    int n = 0;
    while (name[n] != '\0')
    {
        n++;
    }
}

```

```

    }
    printf("%i\n", n);
}

```

Observe que este código entra em loop até que o caractere NUL seja encontrado.

- Este código pode ser melhorado abstraindo a contagem para uma função, da seguinte forma:

```

// Determines the length of a string using a function

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

int string_length(string s);

int main(void)
{
    // Prompt for user's name
    string name = get_string("Name: ");
    int length = string_length(name);
    printf("%i\n", length);
}

int string_length(string s)
{
    // Count number of characters up until '\0' (aka NUL)
    int n = 0;
    while (s[n] != '\0')
    {
        n++;
    }
    return n;
}

```

Observe que uma nova função chamada `string_length` `counts` conta os caracteres até encontrar o caractere nulo ('NUL').

- Como esse é um problema muito comum em programação, outros programadores criaram código na `string.h` biblioteca para encontrar o comprimento de uma string. Você pode encontrar o comprimento de uma string modificando seu código da seguinte forma:

```

// Determines the length of a string using a function

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

int main(void)
{
    // Prompt for user's name
    string name = get_string("Name: ");
    int length = strlen(name);
    printf("%i\n", length);
}

```

Observe que este código utiliza a `string.h` biblioteca, declarada no início do arquivo. Além disso, ele utiliza uma função dessa biblioteca chamada `strlen`, que calcula o comprimento da string passada para ela.

- Nossa código pode se apoiar no trabalho de programadores que vieram antes e usar as bibliotecas que eles criaram.
- `ctype.h` é outra biblioteca bastante útil. Imagine que queremos criar um programa que converta todos os caracteres minúsculos em maiúsculos. Na janela do terminal, digite `code uppercase.c` e escreva o código da seguinte forma:

```
// Uppercases a string

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

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

Observe que este código *itera* por cada valor na string. O programa examina cada caractere. Se o caractere for minúsculo, ele subtrai o valor 32 para convertê-lo em maiúsculo.

- Relembrando o trabalho da semana passada, você deve se lembrar desta tabela de valores ASCII:

| | | | | | | | | | | | | |
|---|------|----|-----|----|----|----|---|----|----|----|---|----|
| 0 | NULO | 16 | DLE | 32 | SP | 48 | 0 | 64 | @ | 80 | P | 96 |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | UM | 81 | Q | 97 |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 |
| 4 | EOT | 20 | DC4 | 36 | \$ | 52 | 4 | 68 | D | 84 | T | 10 |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 10 |

| | | | | | | | | | | | | |
|----|-------|----|----------|----|---|----|---|----|----|----|---|----|
| 6 | ACK | 22 | SINÔNIMO | 38 | & | 54 | 6 | 70 | F | 86 | V | 10 |
| 7 | BEL | 23 | ETB | 39 | ' | 55 | 7 | 71 | G | 87 | C | 10 |
| 8 | BS | 24 | PODE | 40 | (| 56 | 8 | 72 | H | 88 | X | 10 |
| 9 | HT | 25 | EM | 41 |) | 57 | 9 | 73 | EU | 89 | Y | 10 |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 10 |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [| 10 |
| 12 | FF | 28 | FS | 44 | , | 60 | < | 76 | L | 92 | \ | 10 |
| 13 | CR | 29 | GS | 45 | - | 61 | = | 77 | M | 93 |] | 10 |
| 14 | ENTÃO | 30 | RS | 46 | . | 62 | > | 78 | N | 94 | ^ | 11 |
| 15 | SI | 31 | NÓS | 47 | / | 63 | ? | 79 | O | 95 | _ | 11 |

- Quando um caractere minúsculo é `32` subtraído de outro, o resultado é uma versão maiúscula desse mesmo caractere.
- Embora o programa faça o que queremos, existe uma maneira mais fácil usando a `ctype.h` biblioteca. Modifique seu programa da seguinte forma:

```
// Uppercases string using ctype library (and an unnecessary condition)

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

int main(void)
{
    string s = get_string("Before: ");
    printf("After: ");
    for (int i = 0, n = strlen(s); i < n; i++)
    {
        if (islower(s[i]))
        {
            printf("%c", toupper(s[i]));
        }
        else
        {
            printf("%c", s[i]);
        }
    }
    printf("\n");
}
```

Observe que o programa itera por cada caractere da string. A `toupper` função recebe um argumento `s[i]`. Cada caractere (se minúsculo) é convertido para maiúsculo.

- Vale mencionar que o `toupper` programa reconhece automaticamente apenas caracteres minúsculos como maiúsculas. Portanto, seu código pode ser simplificado da seguinte forma:

```
// Uppercases string using ctype library

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

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

Observe que este código converte uma string para maiúsculas usando a `ctype` biblioteca.

- Você pode ler sobre todas as funcionalidades da `ctype` biblioteca nas [Páginas do Manual](https://manual.cs50.io/#ctype.h) (<https://manual.cs50.io/#ctype.h>) .

Argumentos da linha de comando

- `Command-line arguments` São os argumentos passados para o seu programa na linha de comando. Por exemplo, todas as instruções digitadas após ` `clang` são consideradas argumentos de linha de comando. Você pode usar esses argumentos em seus próprios programas!

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

```
// Uses get_string

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

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

Note que isso se refere `hello` ao usuário.

- Ainda assim, não seria bom poder receber argumentos antes mesmo do programa ser executado? Modifique seu código da seguinte forma:

```
// Prints a command-line argument

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

int main(int argc, string argv[])
{
    if (argc == 2)
    {
        printf("hello, %s\n", argv[1]);
    }
    else
    {
        printf("hello, world\n");
    }
}
```

Observe que este programa conhece tanto `argc`, o número de argumentos da linha de comando, quanto `argv`, que é uma matriz dos caracteres passados como argumentos na linha de comando.

- Portanto, usando a sintaxe deste programa, a execução `./greet David` resultaria na exibição da seguinte mensagem `hello, David`:
- Você pode imprimir cada um dos argumentos da linha de comando com o seguinte comando:

```
// Prints command-line arguments

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

int main(int argc, string argv[])
{
    for (int i = 0; i < argc; i++)
    {
        printf("%s\n", argv[i]);
    }
}
```

Status de saída

- Quando um programa termina, um código de saída especial é fornecido ao computador.
- Quando um programa termina sem erros, o computador recebe um código de status de 0. Frequentemente, quando ocorre um erro que resulta no encerramento do programa, o computador retorna um código de status de 0.
- Você pode escrever um programa que ilustre isso digitando `code status.c` e escrevendo o código da seguinte forma:

```
// Returns explicit value from main

#include <cs50.h>
```

```

#include <stdio.h>

int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("Missing command-line argument\n");
        return 1;
    }
    printf("hello, %s\n", argv[1]);
    return 0;
}

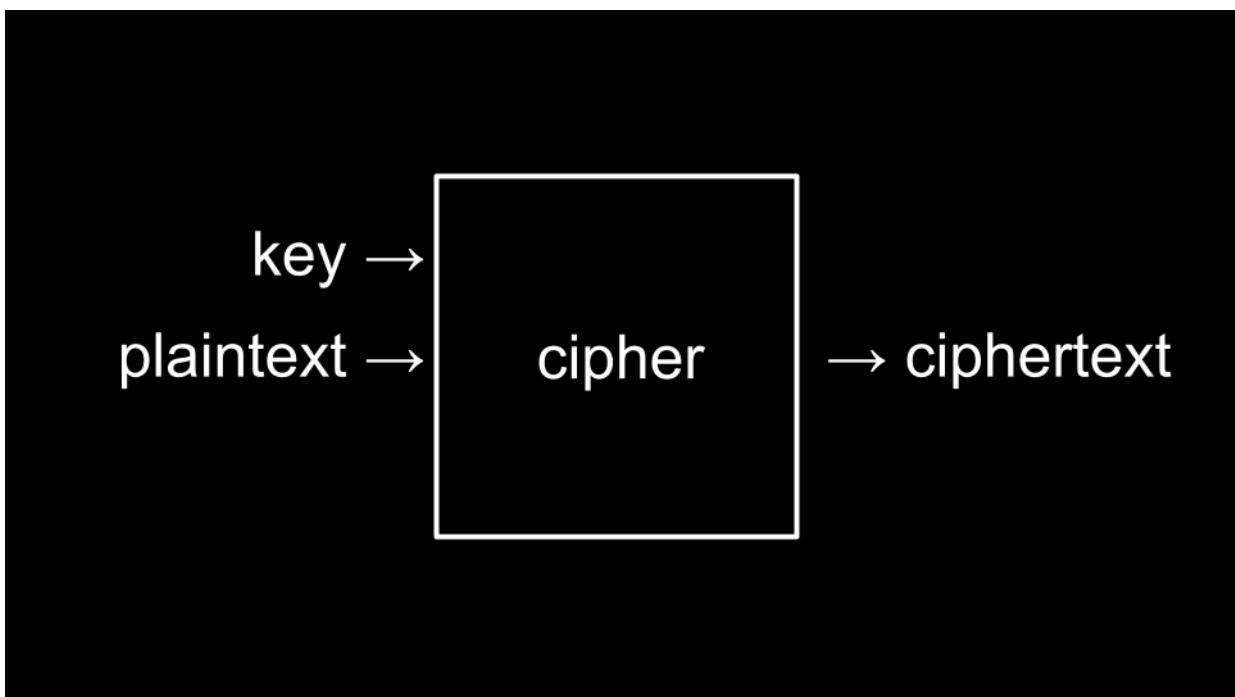
```

Observe que, se você não fornecer `./status David`, receberá um código de saída `1`. No entanto, se você fornecer `./status David`, receberá um código de saída `0`.

- Você pode digitar `echo $?` no terminal para ver o status de saída do último comando executado.
- Você pode imaginar como partes do programa acima poderiam ser usadas para verificar se um usuário forneceu o número correto de argumentos de linha de comando.

Criptografia

- Criptografia é a arte de cifrar e decifrar uma mensagem.
- Agora, com os elementos básicos de matrizes, caracteres e strings, você pode criptografar e decifrar uma mensagem.
- `plaintext` e um `key` são fornecidos a um `cipher`, resultando em texto cifrado.



- A chave é um argumento especial passado para a cifra juntamente com o texto plano. A cifra usa a chave para tomar decisões sobre como implementar seu algoritmo de criptografia.
- Esta semana você enfrentará desafios de programação semelhantes aos descritos acima.

Resumindo

Nesta lição, você aprendeu mais detalhes sobre compilação e como os dados são armazenados em um computador. Especificamente, você aprendeu...

- Em linhas gerais, é assim que um compilador funciona.
- Como depurar seu código usando quatro métodos.
- Como utilizar arrays em seu código.
- Como os arrays armazenam dados em porções de memória consecutivas.
- Como as strings são simplesmente matrizes de caracteres.
- Como interagir com arrays no seu código.
- Como passar argumentos de linha de comando para seus programas.
- Os elementos básicos da criptografia.

Até a próxima!