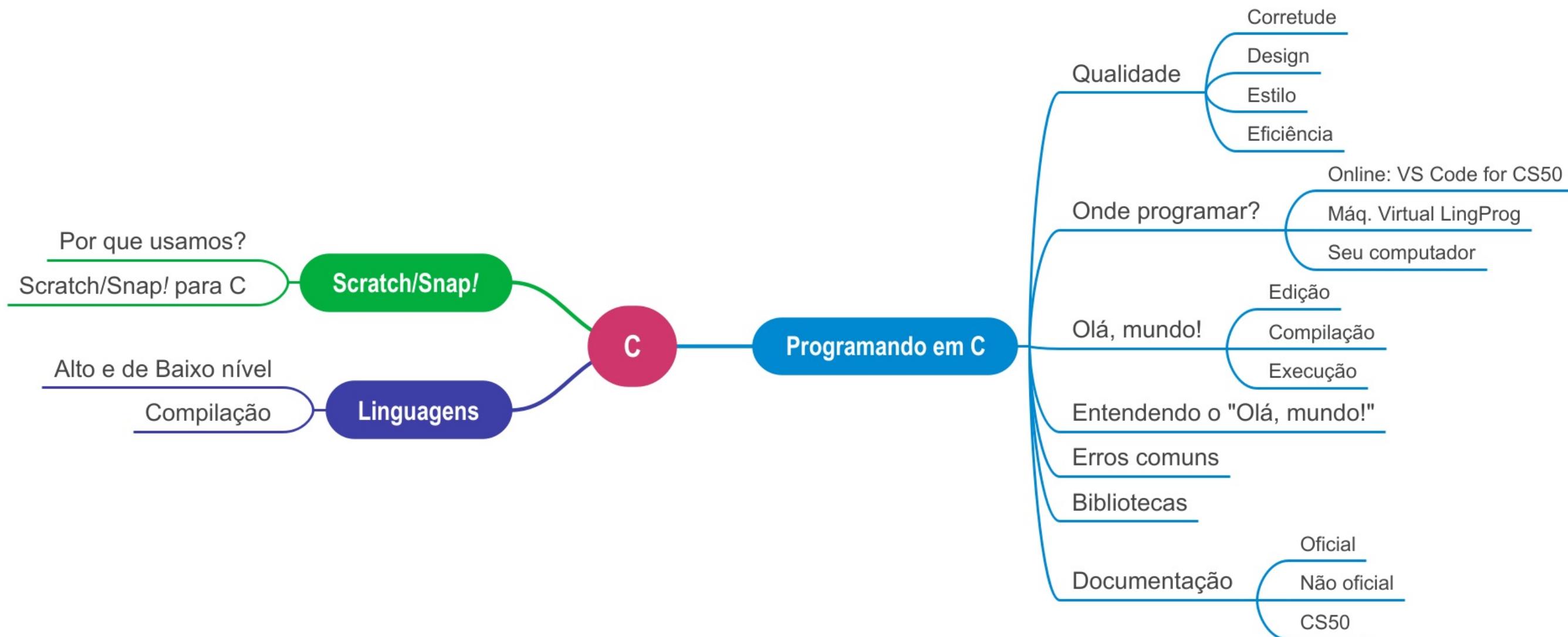


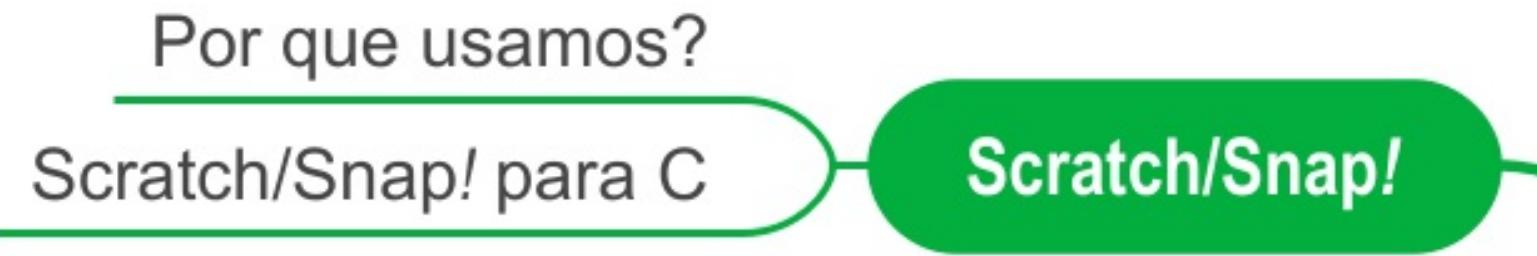
LINGUAGEM C



Linguagem C



Lembrete: por que usamos Scratch e Snap! ?



Lembrete: por que usamos Scratch e Snap! ?

Aprendemos a programar!

- Operações sequenciais
- Operações de repetição
- Operações de decisão e exp. booleanas
- Estruturas de dados (variáveis, arrays)

Aprendemos pensamento computacional!

- Decomposição, padrões, abstração, algoritmos, estruturas de dados

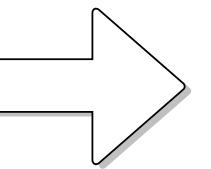
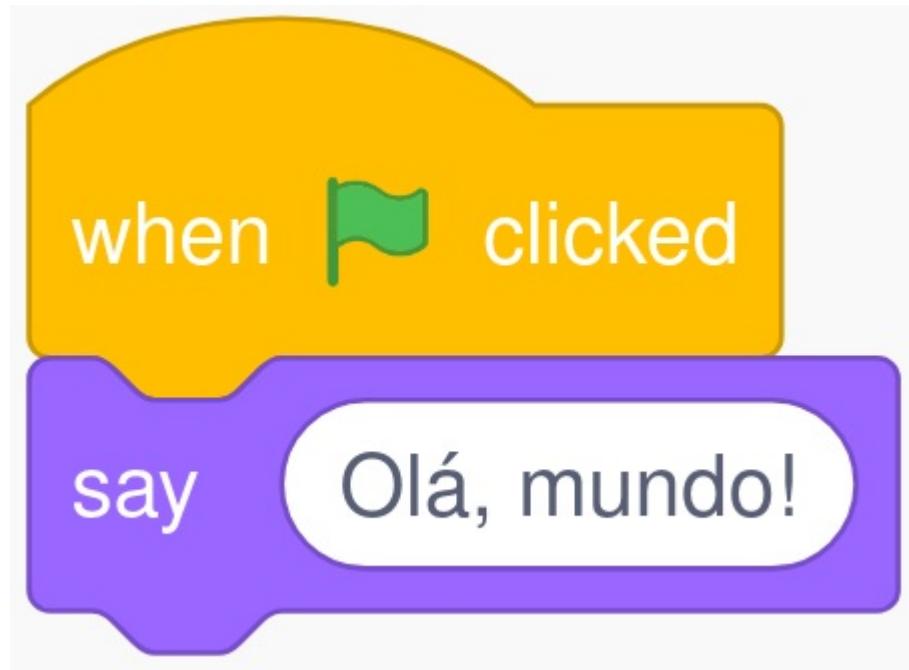
Aprendemos sobre subprogramas:

- Funções, predicados e procedimentos
- Parâmetros e argumentos
- Retorno e efeito colateral
- Funções puras

Aprendemos conceitos avançados diversos.

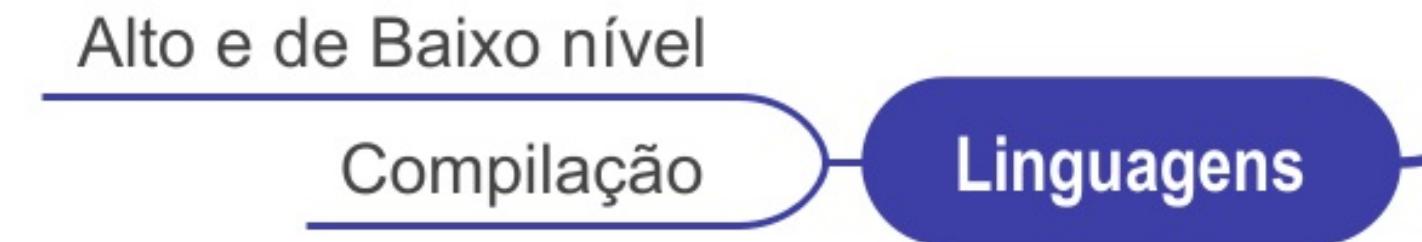


Transição: Scratch/Snap! para C



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

Linguagens de Alto Nível x Linguagens de Baixo Nível



Linguagens de Alto Nível x Linguagens de Baixo Nível

Linguagens de Alto Nível

O programador escreve os algoritmos em uma **linguagem de fácil entendimento, semelhante a um texto** (inglês).

O programador salva o programa em um arquivo chamado de **código fonte**.

Não importa a linguagem (C, Scratch, Python, etc), todo programa deve ser escrito e salvo em um arquivo de código fonte.

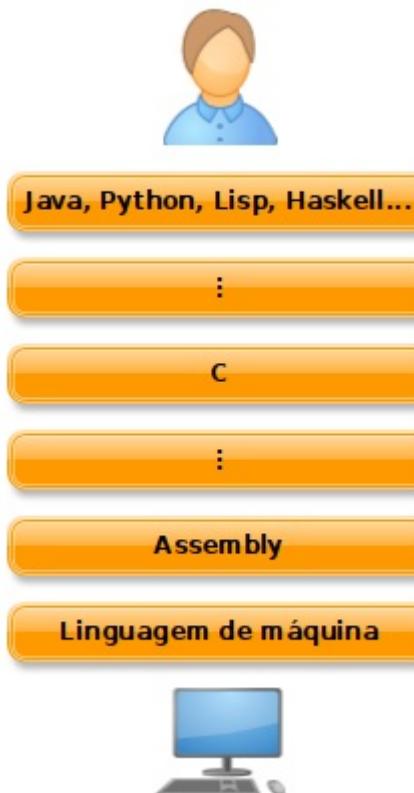
```
#include <stdio.h>

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

Linguagens de Baixo Nível

O programador escreve os algoritmos em uma **linguagem semelhante ao que, de fato, o computador entende**. A linguagem mais baixo nível que existe é a **linguagem de máquina** (também chamada de **código de máquina**).

A linguagem de máquina corresponde aos bits (0 e 1) do seu programa. Contém **dados e instruções**.



```
01111111 01000101 01001100 01000110 00000010
00000000 00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000 00000001
10110000 00000101 01000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000
11010000 00010011 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01000000
00001001 00000000 01000000 00000000 00100100
```

Como transformar o código fonte em código de máquina?

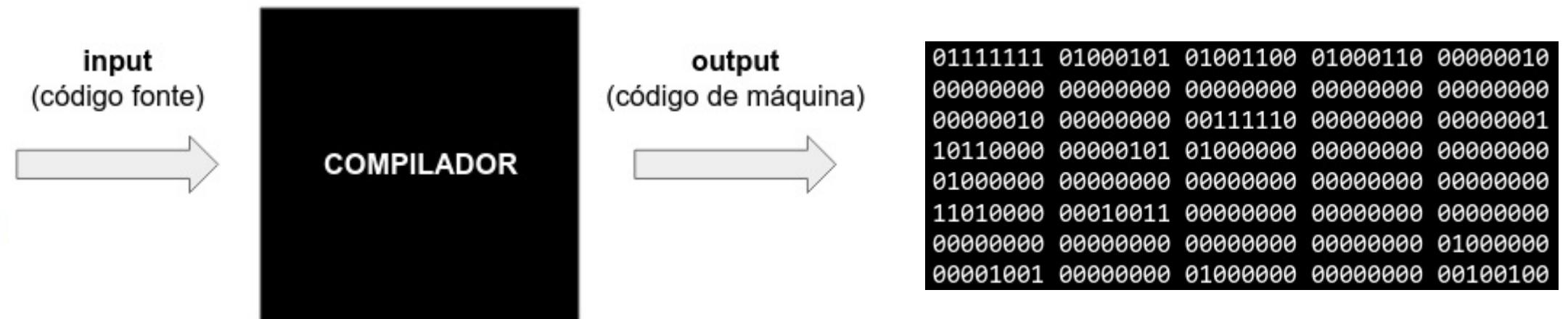
Se o computador só entende linguagem de máquina (bits 0 e 1) mas o programador escreve em uma linguagem de alto nível, **como fazer a transformação entre a linguagem de alto nível para a linguagem de máquina?**

Podemos encarar essa transformação com o mesmo modelo geral da computação:



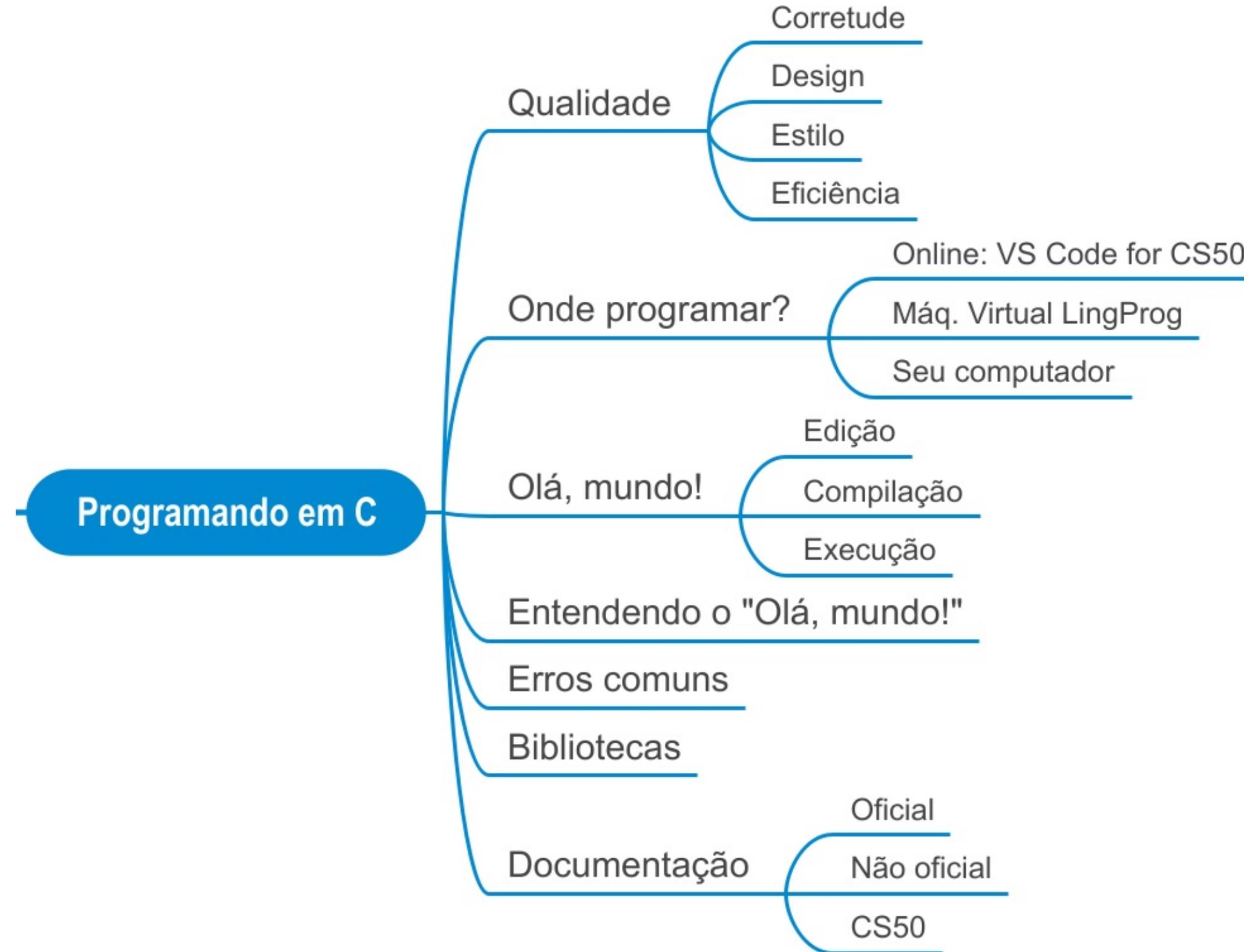
Como transformar o código fonte em código de máquina?

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



Compiladores para C:
GCC (GNU Compiler Collection)
Clang (Clang/LLVM)

Programando em C



Programando em C: avaliação de seus programas

Você não deve somente escrever códigos, deve escrever **bons códigos!**

- Difícil, não se aprende do dia para a noite
- Requer muita prática (de **programação** e de **leitura de programas**)

Qualidade do seu código:

Corretude:

Seu programa faz o que deveria fazer?

Design:

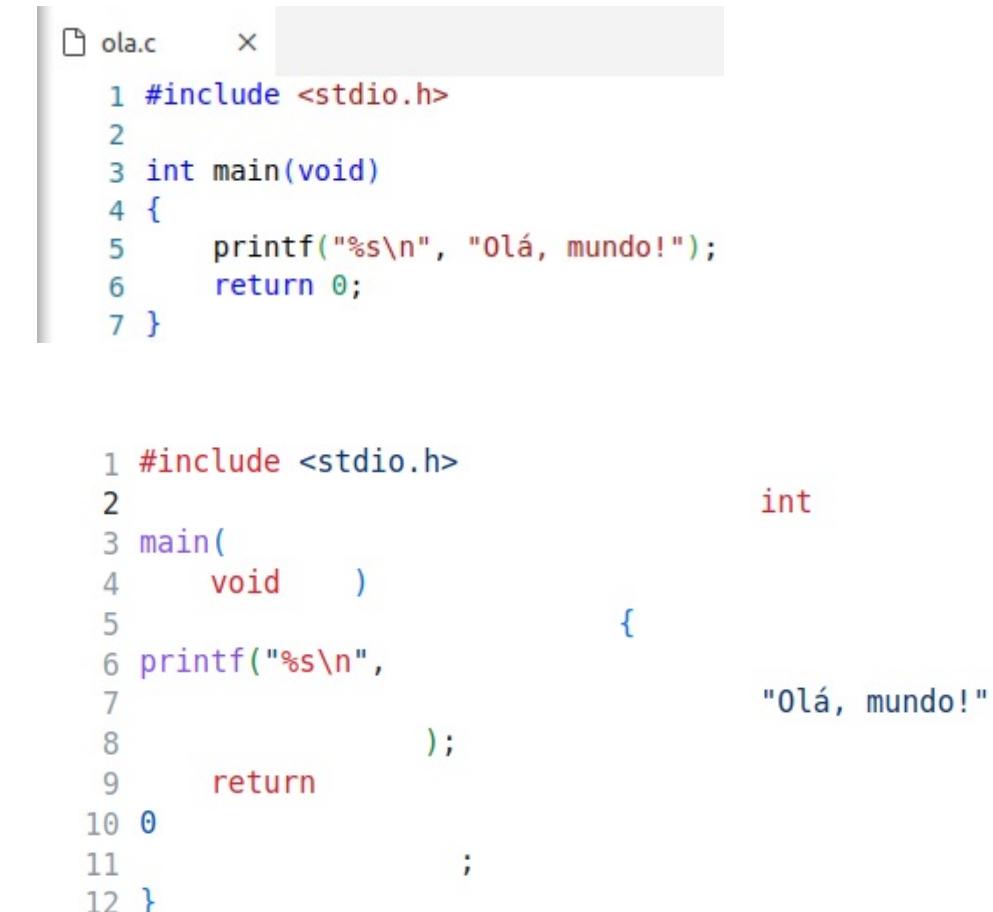
Elegância, sofisticação
(subjetivo, aprende-se com o tempo)

Estilo:

Estética, organização, do seu código
(<https://cs50.readthedocs.io/style/c/>)

Eficiência:

Seu código é rápido o suficiente?



```
ola.c      x
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

```
1 #include <stdio.h>
2
3 main(
4     void    )
5 {
6     printf("%s\n",
7         "Olá, mundo!");
8 }
9     return
10    0
11
12 }
```

Programando em C: onde programar?

Editor de texto puro +
Git/GitHub

Caderno, lápis, borracha e apontador!

Ferramentas online:

Visual Studio Code for CS50 (<https://cs50.dev>)

Máquina virtual:

Computação Raiz: LingProg(<https://we.tl/t-4dwWMffWod>)

Seu computador Linux/Mac:

Visual Studio Code (<https://code.visualstudio.com>)

VSCodium (<https://vscodium.org>)

GCC ou Clang

Seu computador Windows:

Visual Studio Code (<https://code.visualstudio.com>)

MinGW-w64 (<https://www.mingw-w64.org>)

Ubuntu no WSL (<https://ubuntu.com/wsl>)



Recomendado:
Ambiente de Desenvolvimento Integrado
(Integrated Development Environment - IDE)

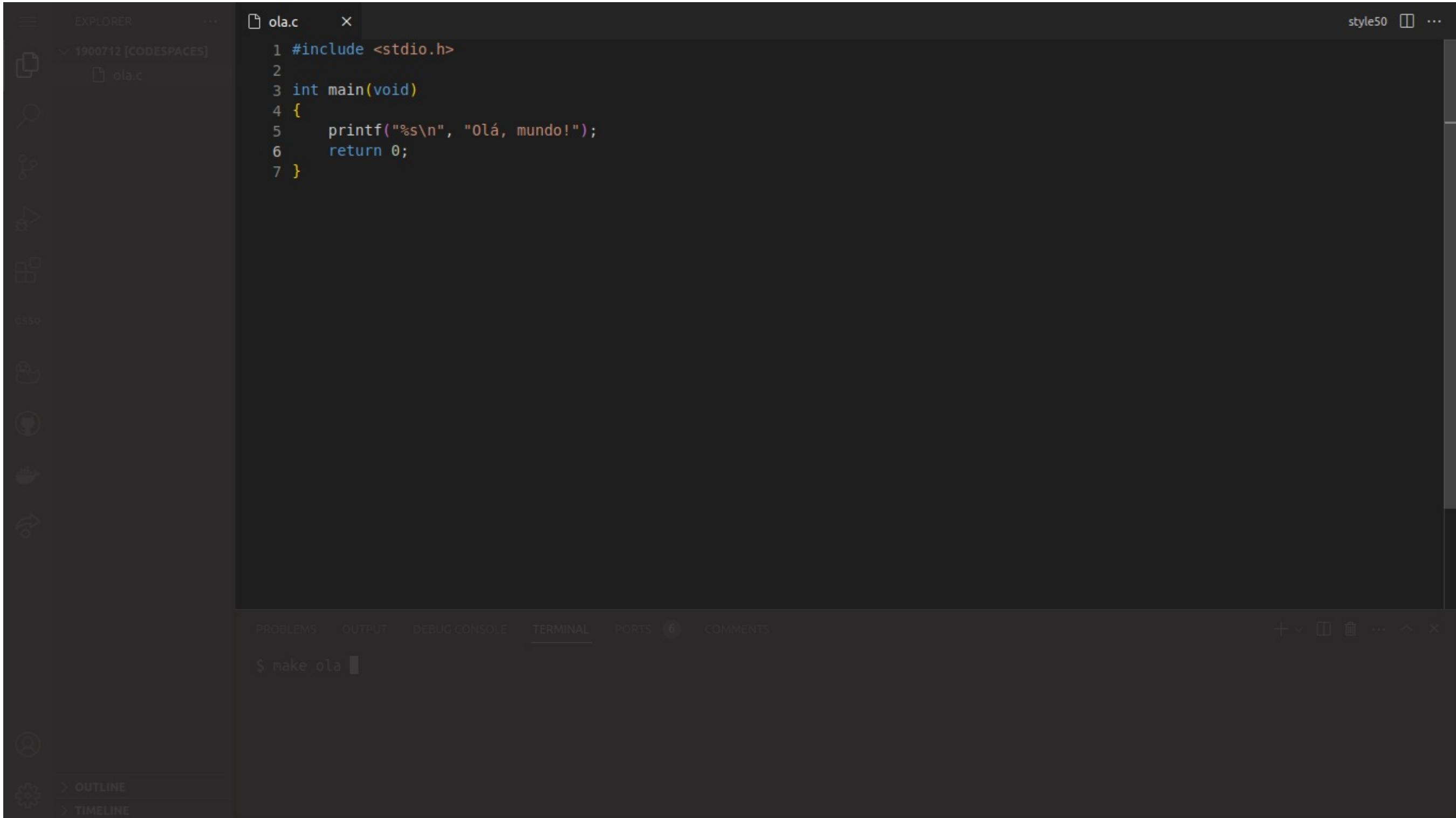
Programando em C: onde programar?

The screenshot shows the Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar, which displays a project named "1900712 [CODESPACES]" containing a file "ola.c". The main editor area shows the following C code:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

Below the editor is a terminal window. The tabs at the top of the terminal are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and COMMENTS. The TERMINAL tab is selected, showing the command "\$ make ola" entered by the user. The terminal output area is currently empty.

Programando em C: onde programar?



A screenshot of a dark-themed code editor, likely Visual Studio Code, showing a C program named `ola.c`. The code contains a single function `main` that prints "Olá, mundo!" to the console.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

The code editor interface includes a sidebar with icons for file operations like Open, Save, Find, and Share. The main area shows the file structure under `1900712 [CODESPACES]`. Below the code editor are several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS (with a count of 6), and COMMENTS. The TERMINAL tab is active, showing the command `$ make ola`. The bottom navigation bar includes icons for Help, Outline, and Timeline.

Programando em C: onde programar?



The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar with icons for files, folders, and other code spaces. The main area displays a file named 'ola.c' with the following code:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

To the right of the code, the text 'GUI x CLI' is displayed in yellow. Below it, definitions for GUI and CLI are given:

GUI = Graphical User Interface
CLI = Command Line Interface

A bold yellow text 'CLI é extremamente poderosa!' is centered below the definitions.

On the right side of the slide, there is explanatory text in yellow:

Ao dominar o uso da CLI você se tornará muito mais produtivo e conseguirá fazer coisas que não são possíveis em GUI:

- mais ferramentas
- digitação mais rápida
- combinação de comandos
- etc.

At the bottom of the slide, a terminal window is shown with the command '\$ make ola' entered, indicating the compilation of the C program.

Programando em C: onde programar?

The screenshot shows a dark-themed code editor interface, likely Visual Studio Code, with the following details:

- EXPLORER**: Shows a folder structure for '1900712 [CODESPACES]' containing a file named 'ola.c'. The 'ola.c' file is selected.
- Code Editor**: Displays the following C code:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```
- TERMINAL**: Shows a terminal window with the command '\$ make ola' entered.
- Bottom Bar**: Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS (with a notification count of 6), and COMMENTS. It also features a set of small icons for file operations like new, save, and close.
- Sidebar**: Features icons for file navigation, search, and other development tools.
- Bottom Navigation**: Buttons for OUTLINE and TIMELINE.

Programando em C: onde programar?

The screenshot shows a dark-themed code editor interface. On the left is a vertical sidebar with various icons: a file icon, a magnifying glass, a folder, a gear, a person, a gear with a person, a ship, and a refresh symbol. The main area displays a file named 'ola.c' with the following code:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("%s\n", "Olá, mundo!");
6     return 0;
7 }
```

Below the code editor, there is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS (with a notification count of 6), and COMMENTS. The TERMINAL tab is currently selected. In the terminal pane, the command '\$ make ola' is entered. The status bar at the bottom shows the path '1900712 [CODESPACES]' and other UI elements.

Programando em C: seu primeiro programa

Use o **VSCode for CS50** e crie o "Olá, mundo!"

\$ **code ola.c**

Cria um arquivo de **código fonte** chamado "ola.c".

Não use caracteres especiais!

Use letras minúsculas sem acentos, e underscore.

Extensão padrão: ".c"

\$ **make ola**

Compila o código fonte em **código de máquina** e cria o executável "ola".

Não coloque o ".c" no final!

\$ **./ola**

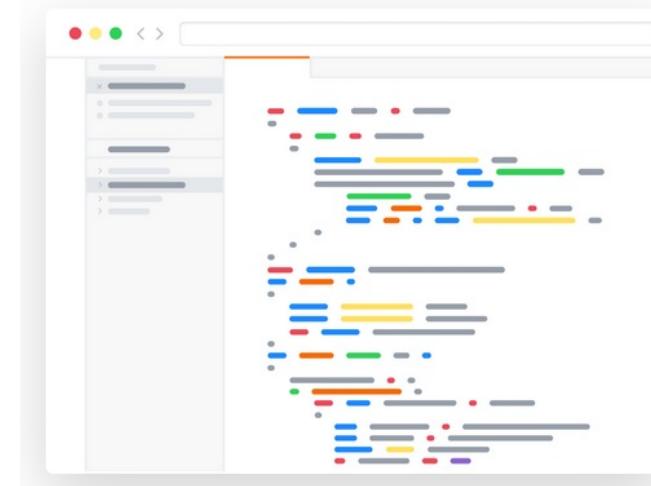
Executa o programa "ola" no diretório atual.

Visual Studio Code for CS50 

CS50's adaptation of [Codespaces](#) for students and teachers

with these features

[Log In](#) or browse documentation



<https://cs50.dev>

Programando em C: seu primeiro programa

Digite, compile e execute:

```
#include <stdio.h>

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

Note o **realce de sintaxe** (syntax highlighting) e a **indentação**.

Se você fizer qualquer alteração no código fonte, precisará
compilar novamente!

Programando em C: entendendo o programa

Como seu programa funciona?

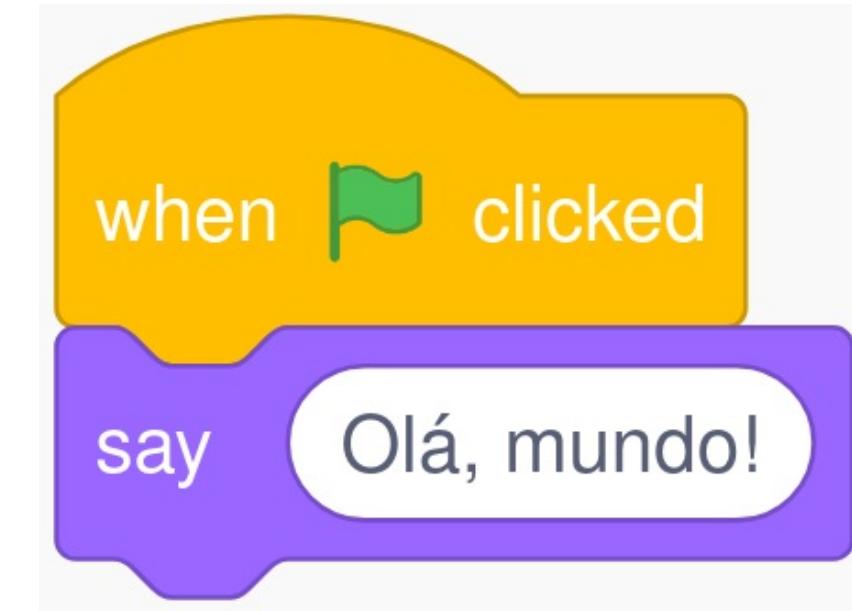
```
#include <stdio.h>

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

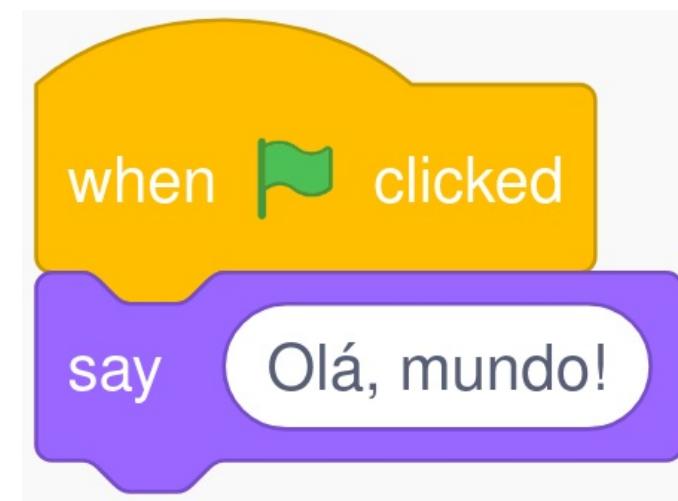
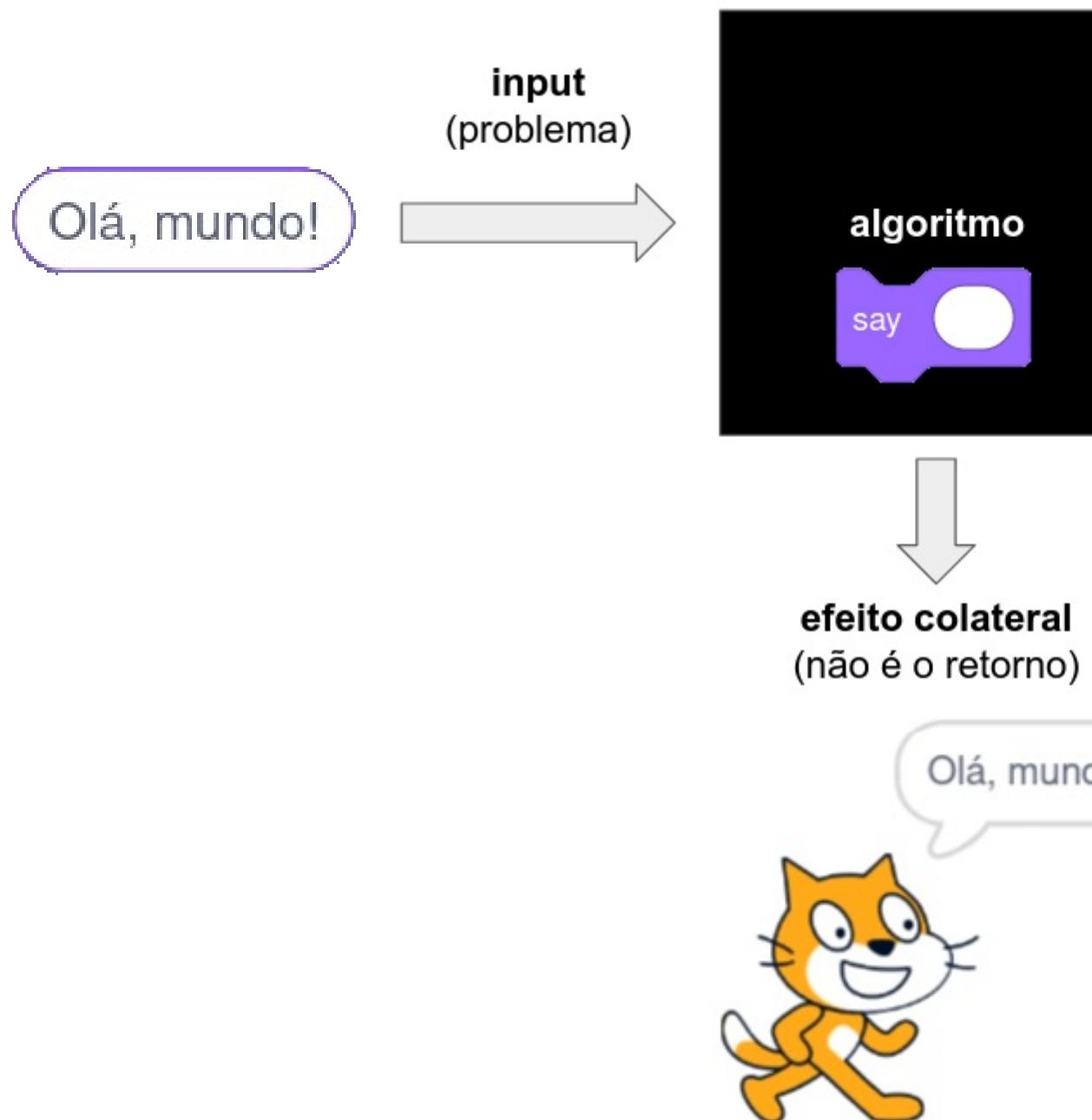
Programando em C: entendendo o programa

```
#include <stdio.h>

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



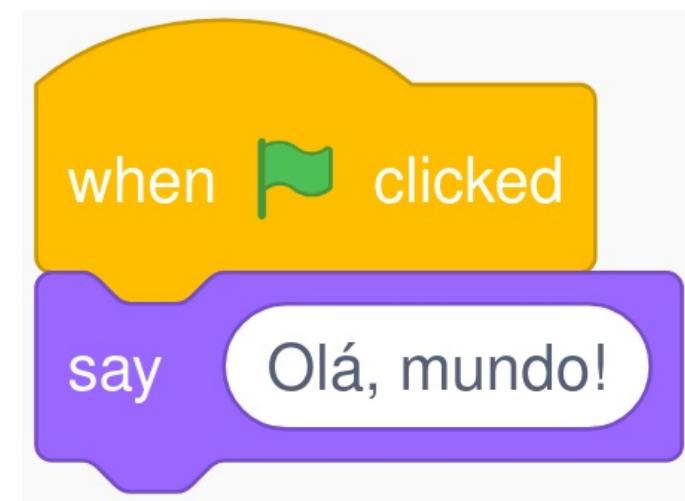
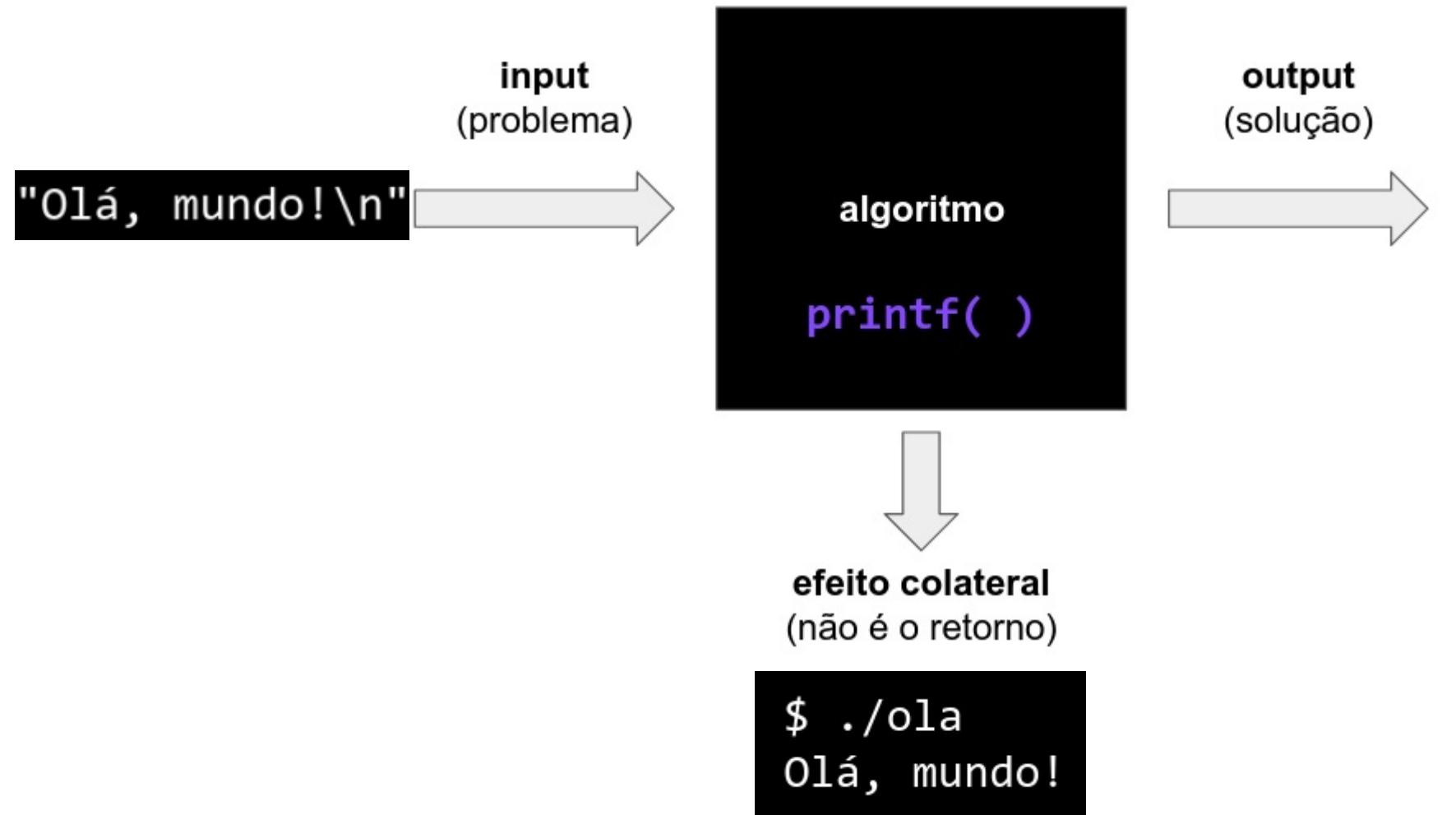
Programando em C: entendendo o programa



```
#include <stdio.h>

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

Programando em C: entendendo o programa



```
#include <stdio.h>

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

Programando em C: entendendo o programa

main: função especial que marca o início de seu programa (veremos posteriormente nessa unidade). Seu programa deve ficar "dentro" da função main, ou seja, dentro das chaves (há exceções, veremos posteriormente).

```
int main (void)  
{  
}  
}
```

when  clicked

Programando em C: entendendo o programa

printf:

função para imprimir alguma coisa, de acordo com um formato

string:

seqüência de 0 ou mais caracteres, entre aspas duplas

escape sequence: começa com uma \ e serve para imprimir caracteres não imprimíveis



Função

```
printf("Olá, mundo!\\n");
```

Argumento: "string"

\n escape sequence:
imprime uma quebra de linha

Finaliza a sentença

Programando em C: erros comuns

- Esquecer de **recompilar** o programa ao fazer qualquer alteração no código fonte.
- Esquecer o **;** no final de uma sentença.
- Colocar **;** no final de linhas que não são uma sentença.
- Esquecer o **\n** quando quiser uma quebra de linha.
- Tentar **quebrar a linha dentro de uma string, sem usar \n.**
- Escrever **studio.h**.
- Escrever **print**.
- Esquecer de **fechar as aspas duplas** nas strings.
- Usar **aspas simples para strings**.
- Usar **make ola.c**.
- **Desbalancear delimitadores** (chaves, colchetes e parênteses).
- ...

Não se desespere! Leia as mensagens de erro do compilador e veja se alguma coisa te ajuda a corrigir o erro.

Bibliotecas

O que a linha abaixo faz?

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

Por que o sinal de #?

Por que stdio.h?

Por que está entre <>?

Por que não tem ; no final?

O que ocorre se não colocar essa linha?

Por que essa complicação toda?

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

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

Bibliotecas

Tentativa de compilar o arquivo "ola.c" sem escrever a linha:

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



The screenshot shows a terminal window with the following output:

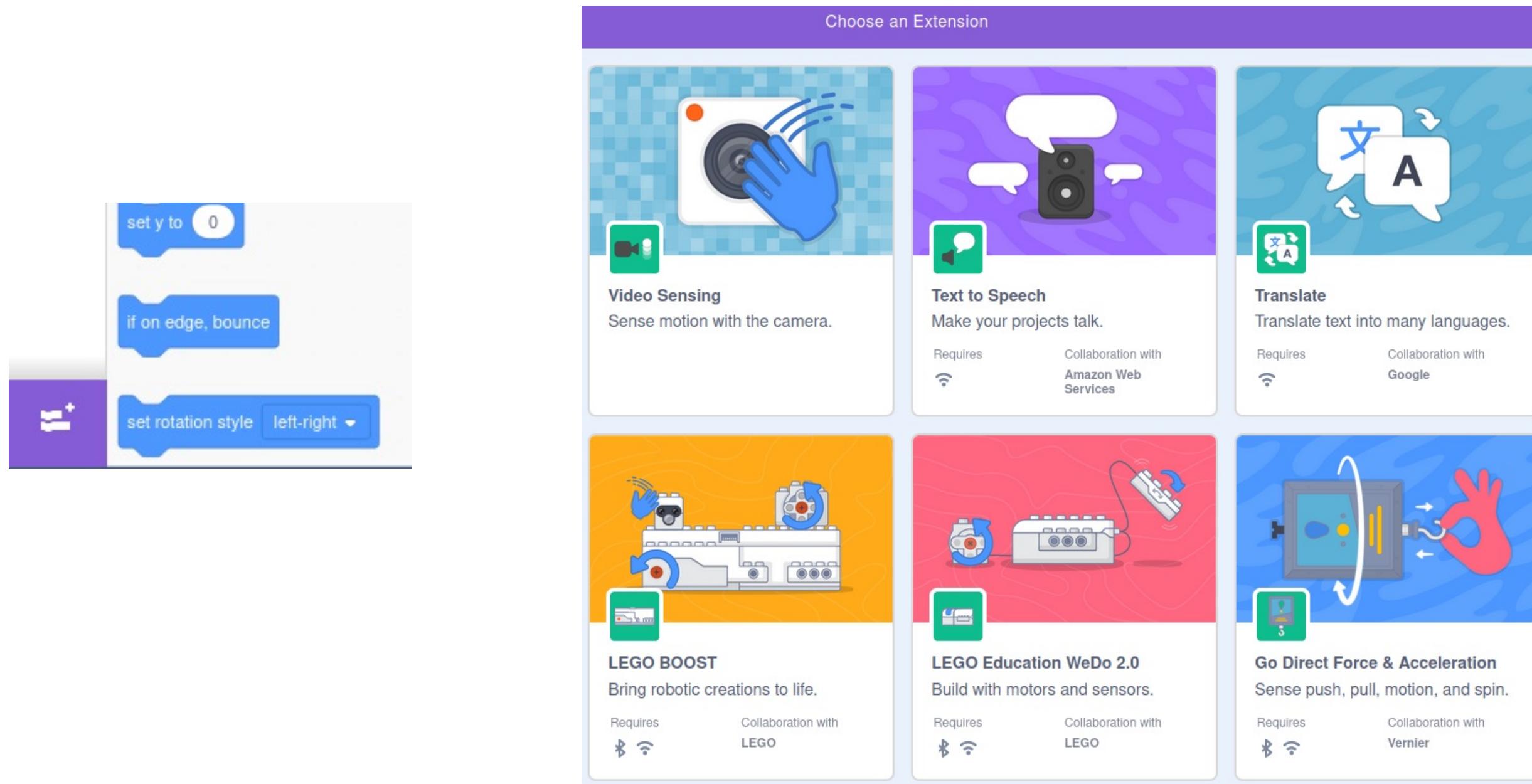
```
$ make ola
ola.c:5:5: error: call to undeclared library function 'printf' with type 'int (const char *, ...)';
ISO C99 and later do not support implicit function declarations [-Wimplicit-function-declaration]
    5 |     printf("Olá, mundo!\n");
      |
ola.c:5:5: note: include the header <stdio.h> or explicitly provide a declaration for 'printf'
1 error generated.
make: *** [<>builtin>: ola] Error 1
$
```



O que é esse negócio aí de "printf"?

Bibliotecas

No Scratch, para termos **acesso à algumas funções especiais**, era necessário clicar no botão "Add Extension" e escolher quais funções especiais iríamos utilizar:



Bibliotecas

Em C é a mesma coisa: muitas funções especiais não fazem parte da linguagem por si mesma, mas estão em **bibliotecas que podemos incluir em nossos programas**.

Bibliotecas de código: são conjuntos de **funções, variáveis e estruturas de dados** que outros programadores disponibilizam para usarmos em nossos próprios programas.

Essas **bibliotecas (libraries)** ficam armazenadas em arquivos chamados de **arquivos objeto (.o, .a, .so)**, em algum lugar do nosso computador. Nós podemos acessar esses arquivos através de uma **header file (.h)** em nosso programa. Por exemplo:

A library **stdio.o** é acessada através da header file **stdio.h**.

Usamos uma **header file** que "aponta" para a **library** que tem as funções que estamos querendo usar em nossa programa. A função printf está dentro da library stdio.o. Podemos usar várias headers files ao mesmo tempo.

Tecnicamente a coisa toda é mais complicada... veremos posteriormente.

Bibliotecas

```
#include <stdio.h>

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

(em algum lugar no seu computador)

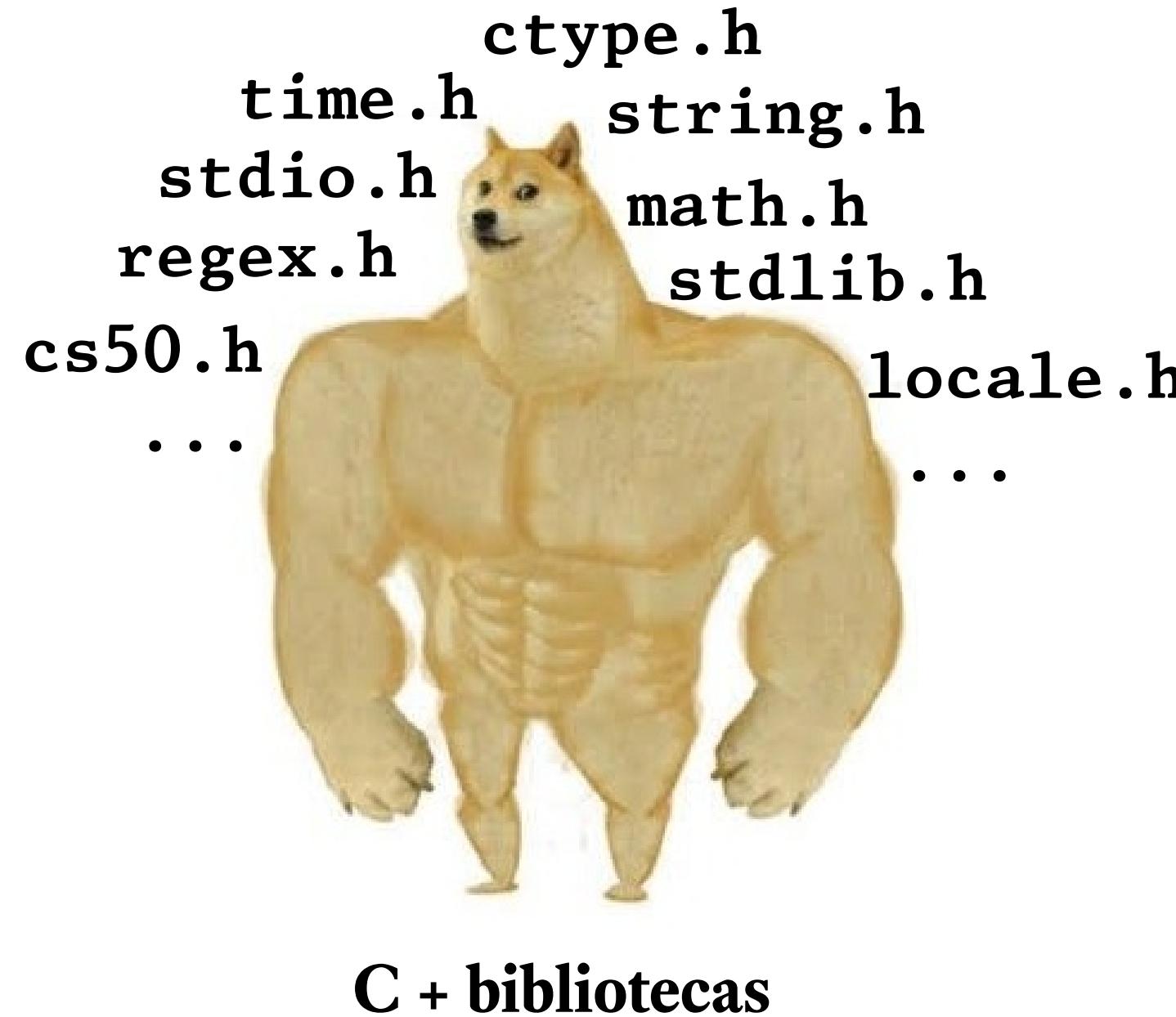
stdio.o

Tecnicamente a coisa toda é mais complicada... veremos posteriormente.

Bibliotecas: por que usar?



C



C + bibliotecas

Bibliotecas: OK, mas onde estão esses arquivos?

Depende do sistema operacional, da instalação das ferramentas de desenvolvimento, etc. O mais importante é o seguinte: se tudo está instalado e configurado corretamente, **você não precisa se preocupar** (por enquanto), o compilador faz a coisa certa e inclui a biblioteca no seu programa.

Somente como curiosidade:

```
[abrantesASF@cosmos ~]$ ls -lh /usr/include/stdio.*  
-rw-r--r-- 1 root root 31K jul 6 2022 /usr/include/stdio.h
```

```
[abrantesASF@cosmos ~]$ ls -lh /usr/lib/x86_64-linux-gnu/libc.*  
-rw-r--r-- 1 root root 5,8M jul 6 2022 /usr/lib/x86_64-linux-gnu/libc.a  
-rw-r--r-- 1 root root 283 jul 6 2022 /usr/lib/x86_64-linux-gnu/libc.so  
-rw-r--r-- 1 root root 2,2M jul 6 2022 /usr/lib/x86_64-linux-gnu/libc.so.6
```

```
[abrantesASF@cosmos ~]$ cp /usr/lib/x86_64-linux-gnu/libc.a .
```

```
[abrantesASF@cosmos ~]$ ar x libc.a stdio.o
```

```
[abrantesASF@cosmos ~]$ ls -lh stdio.o  
-rw-r--r-- 1 abrantesASF abrantesASF 1,3K ago 12 23:16 stdio.o
```

libc é a biblioteca padrão (standard library) da linguagem C. Contém várias outras bibliotecas em seu interior.

Bibliotecas: agora é possível responder

O que a linha abaixo faz?

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

Por que o sinal de #?

Por que stdio.h?

Por que está entre <>?

Por que não tem ; no final?

O que ocorre se não colocar essa linha?

Por que essa complicação toda?

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

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

Documentação: como saber quais bibliotecas existem?

A linguagem C é padronizada e tem um documento oficial:



Licensed to Abrantes Araújo Silva Filho
ISO Store Order: [redacted] license #1/ Downloaded: 2023-04-20
Single user licence only, copying and networking prohibited.

O padrão oficial atual da linguagem C, chamado de **C17**, é vendido no site da ISO. Em 2023, o mais recente é o padrão ISO/IEC 9899:2018, disponível em:

www.iso.org/standard/74528.html

The image is a screenshot of a website page titled "Buy this standard". It shows two options: "PDF" (selected) and "Paper", both in "English". The price is listed as "CHF 208" with a "Buy" button below it.

Documentação: como saber quais bibliotecas existem?

É possível acessar gratuitamente os PDFs dos rascunhos dos padrões na página do grupo de trabalho da ISO para a linguagem C: www.open-std.org/JTC1/SC22/WG14

ISO/IEC JTC1/SC22/WG14 - Mozilla Firefox

ISO/IEC JTC1/SC22/WG14 × +

https://www.open-std.org/JTC1/SC22/WG14/ 100% ★

Welcome to the official home of

ISO IEC JTC1/SC22/WG14 - C

2021-11-25: [projects](#) | [documents](#) | [contributing](#) | [internals](#) | [meetings](#) | [contacts](#)

ISO/IEC JTC1/SC22/WG14 is the international standardization working group for the programming language C.

The current C programming language standard (C17) [ISO/IEC 9899](#) was adopted by ISO and IEC in 2018.

To obtain the international standard, please contact your national [member body](#).

Work on [projects](#) and their [milestones](#) include:

- [9899: Programming Language C](#) 
- [Defect Reports and Record of Response](#)
- [TR 18037: Embedded C](#)
- [TR 19769: Extensions for the programming language C to support new character data types](#)
- [TR 24731-1: Extensions to the C Library Part I: Bounds-checking interfaces](#)
- [TR 24731-2: Extensions to the C Library Part II: Dynamic allocation functions](#)
- [TR 24732: Extensions for the programming language C to support decimal floating point arithmetic](#)
- WG14 has finished revising the C standard, under the name C11. A [charter](#) for the revision of the standard describes the rules for what has been done.
- [TR 24747: Programming language C - Extensions for the C language library to support mathematical special functions](#)
- The [rationale for the C99 standard](#) is available.

Other information available is:

- The [WG document register](#) including the documents
- [New WG wiki](#) (protected, only for members)
- [WG internal information](#) (protected, only for members)
- Information on [WG meetings](#)
- [WG14 Business Plan and Convener's Report 2012](#)
- Working Group Standing Document 1, [mailings and meetings information](#)
- Working Group Standing Document 2, [Informal Study Group Organization Information](#)
- Working Group Standing Document 3, [Partial list of proposals that did not fit into the former DR process for C11](#)
- [ISO/IEC 2382:2015 Information technology — Vocabulary](#) (freely available standard)
- [Contacts](#)

If you want further information, or want to participate, please contact your national [member body](#) or one of the [contact addresses](#) of the WG.

2021-11-25: [projects](#) | [documents](#) | [contributing](#) | [internals](#) | [meetings](#) | [contacts](#)

This page is sponsored by [DTU. HTML design](#) by Keld Simonsen. [Comments](#) welcome!

ISO IEC C - Project status and milestones

2023-04-05: [home](#) | [projects](#) | [documents](#) | [contributing](#) | [internals](#) | [meetings](#) | [contacts](#)

ISO/IEC 9899 - Revision of the C standard

The primary output of WG14 is ISO/IEC 9899, the C Standard. The following is a list of revisions to ISO/IEC 9899 that the committee has produced:

Revision	ISO publication	Similar draft
C2x	Not available	N3096 [2023-04-02]
C17	ISO/IEC 9899:2018	N2310 [2018-11-11] (early C2x draft)
C11	ISO/IEC 9899:2011	N120 [2011-04-04]
C99	ISO/IEC 9899:1999	N1256 [2007-09-07]
C89	ISO/IEC 9899:1990	Not available

Documentação: como saber quais bibliotecas existem?

No "The Open Group" é possível acessar a documentação das header files:
<https://pubs.opengroup.org/onlinepubs/9699919799/>

The Open Group Base Specifications Issue 7, 2018 edition — Mozilla Firefox

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1™-2017 (Revision of IEEE Std 1003.1-2008)
[Copyright](#) © 2001-2018 IEEE and The Open Group

INDEX

Search..

[Alphabetic | Topic | Word Search]

Select a Volume:

[Base Definitions | System Interfaces | Shell & Utilities | Rationale]

[Frontmatter]

[Main Index]

IEEE

THE Open GROUP

POSIX.1-2017 is simultaneously IEEE Std 1003.1™-2017 and The Open Group Technical Standard Base Specifications, Issue 7. POSIX.1-2017 defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level. POSIX.1-2017 is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a "shell") and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-2017 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of POSIX.1-2017:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2017 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX™), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Frontmatter (Informative)

[Preface | Typographical Conventions | Notice to Users | Participants | Trademarks | Acknowledgements | Referenced Documents]

Tables of Contents by volume: [XBD | XSH | XCU | XRAT]

Links: [Alphabetic Index | Topical Index | About the HTML version | Downloads | Report a defect]

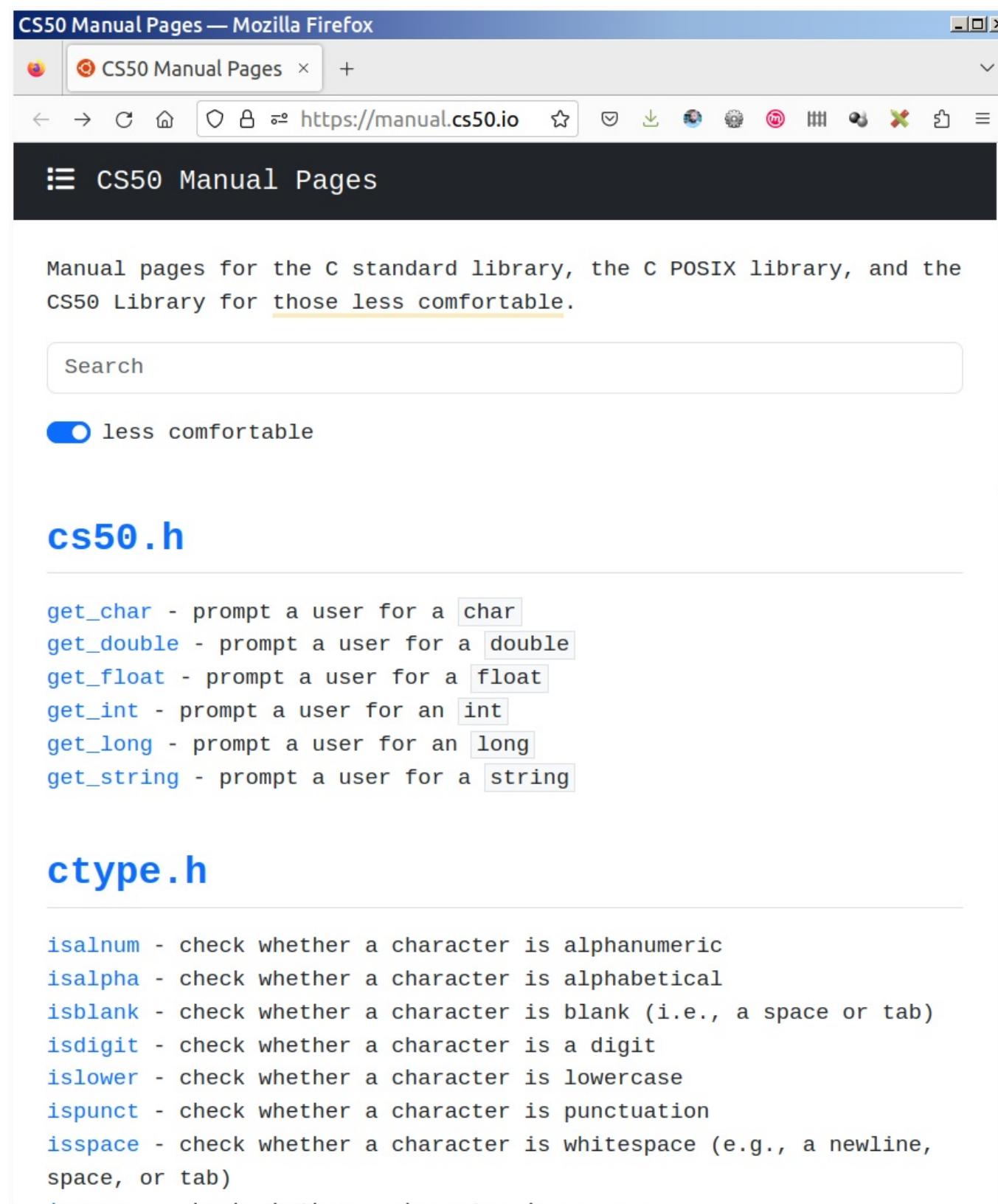
Previous Editions

Links: [2008 | 2013 | 2016]

UNIX ® is a registered Trademark of The Open Group.
POSIX™ is a Trademark of The IEEE.
Copyright © 2001-2018 IEEE and The Open Group, All Rights Reserved



Documentação: como saber quais bibliotecas existem?



The screenshot shows a Mozilla Firefox browser window with the title "CS50 Manual Pages — Mozilla Firefox". The address bar displays "https://manual.cs50.io". The page content is titled "CS50 Manual Pages" and describes manual pages for the C standard library, the C POSIX library, and the cs50 Library for those less comfortable. It includes a search bar and a toggle switch for "less comfortable" mode. Below this, there are two sections: "cs50.h" and "ctype.h", each listing functions from their respective header files.

cs50.h

```
get_char - prompt a user for a char
get_double - prompt a user for a double
get_float - prompt a user for a float
get_int - prompt a user for an int
get_long - prompt a user for an long
get_string - prompt a user for a string
```

ctype.h

```
isalnum - check whether a character is alphanumeric
isalpha - check whether a character is alphabetical
isblank - check whether a character is blank (i.e., a space or tab)
isdigit - check whether a character is a digit
islower - check whether a character is lowercase
ispunct - check whether a character is punctuation
isspace - check whether a character is whitespace (e.g., a newline,
space, or tab)
```

A Universidade de Harvard pegou as **header files** disponíveis no site do The Open Group e simplificou a formatação e o texto para alunos iniciantes na CS50 (a versão completa também está disponível). É essa documentação que você deve consultar periodicamente!

<https://manual.cs50.io>

Obrigado CS50!

Documentação: como saber quais bibliotecas existem?

Na documentação preparada pela CS50 você tem 2 opções:

less confortable

[stdio.h](#)

```
fclose - close a file
fopen - open a file
fprintf - print to a file
fread - read bytes from a file
fscanf - get input from a file
fwrite - write bytes to a file
printf - print to the screen
scanf - get input from a user
sprintf - print to a string
```

more confortable

[stdio.h](#)

```
_fbuflen - interfaces to stdio FILE structure
_fbf - interfaces to stdio FILE structure
_fpending - interfaces to stdio FILE structure
_fpurge - purge a stream
_freadable - interfaces to stdio FILE structure
_freading - interfaces to stdio FILE structure
_fsetlocking - interfaces to stdio FILE structure
_fwritable - interfaces to stdio FILE structure
_fwriting - interfaces to stdio FILE structure
_flushlbf - interfaces to stdio FILE structure
addmntent - get filesystem descriptor file entry
asprintf - print to allocated string
clearerr - check and reset stream status
clearerr_unlocked - nonlocking stdio functions
ctermid - get controlling terminal name
dprintf - formatted output conversion
endmntent - get filesystem descriptor file entry
fclose - close a stream
fcloseall - close all open streams
fdopen - stream open functions
feof - check and reset stream status
feof_unlocked - nonlocking stdio functions
ferror - check and reset stream status
ferror_unlocked - nonlocking stdio functions
fflush - flush a stream
fflush_unlocked - nonlocking stdio functions
fgetc - input of characters and strings
fgetc_unlocked - nonlocking stdio functions
fgetgrent - get group file entry
```

Documentação: como saber quais bibliotecas existem?

A documentação de cada função também tem opções com mais e menos detalhes:

The screenshot shows a Mozilla Firefox window displaying the `printf` manual page. The title bar reads "printf - CS50 Manual Pages — Mozilla Firefox". The address bar shows the URL <https://manual.cs50.io/3/printf>. The page content includes sections for NAME, SYNOPSIS, DESCRIPTION, and EXAMPLES. It also features a "less comfortable" toggle button.

NAME
printf - print to the screen

SYNOPSIS
Header File:

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

Prototype:

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

Note that `...` represents zero or more additional arguments.

DESCRIPTION
This function prints a “formatted string” to the screen. It expects as input a “format string” that specifies what to print and zero or more subsequent arguments. The format string can optionally contain “conversion specifications,” placeholders that begin with `%` that specify how to format the function’s subsequent arguments, if any. For instance, if `c` is a `char`, this function can print it as follows using `%c`:

```
printf("%c\n", c);
```

Alternatively, this function could format that same value as an `int` as well using `%i`, as in an ASCII chart:

```
printf("%c %i\n", c, c);
```

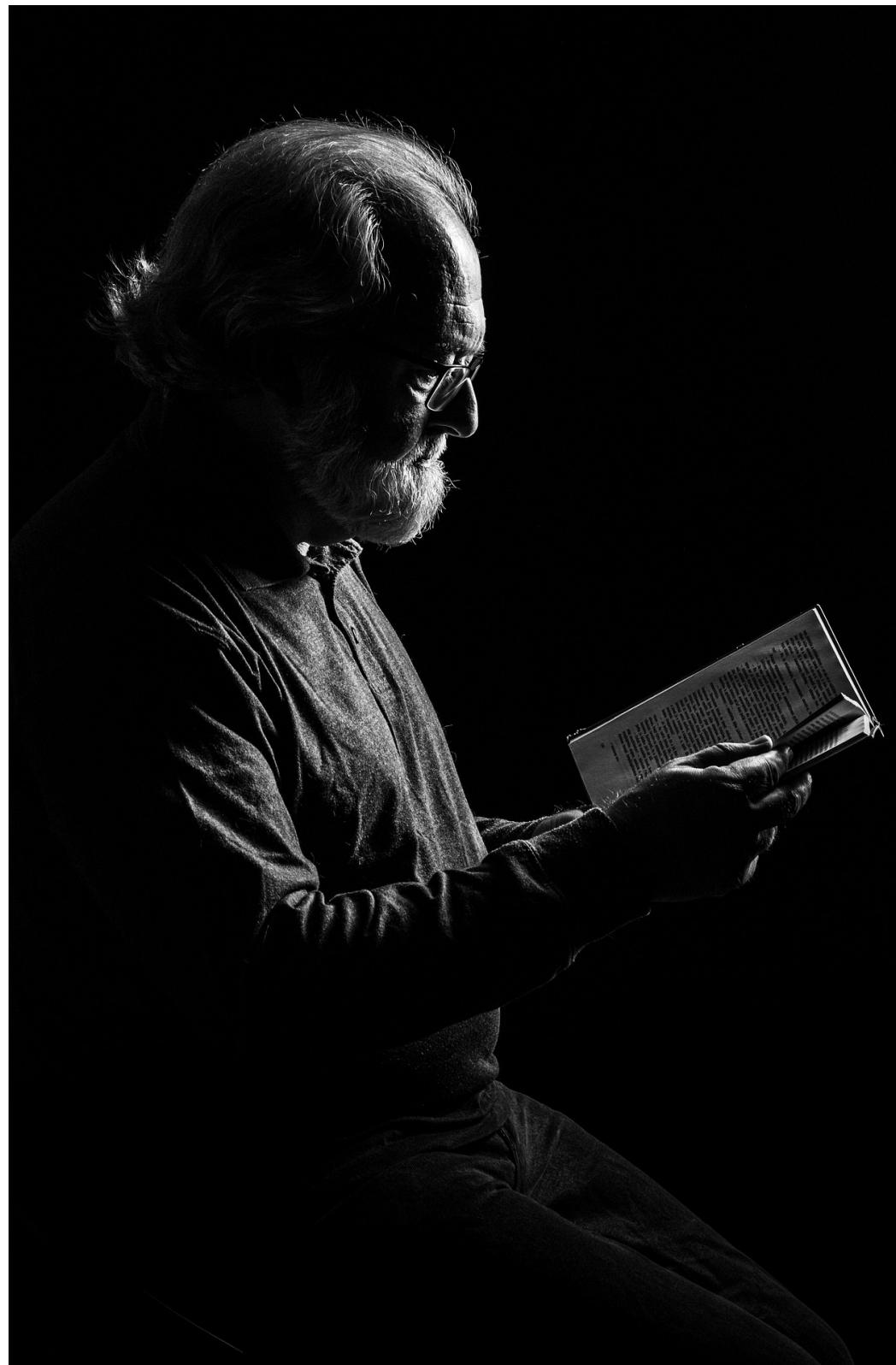
And this function can print strings without any conversion specifications as well:

```
printf("Hello world\n");
```

<https://manual.cs50.io/3/printf#synopsis>

Name
Library
Synopsis
Description
Return values
Examples
Attributes
Standards
History
Caveats/Notes
Bugs
See also
Colophon

Documentação: como saber quais bibliotecas existem?



Em resumo:
RTFM
(Read The Fucking Manual)

List of similar initialisms [edit]

- RTBM – "read the bloody manual"^[6]
- RTFA – "read the fucking/featured article" – common on news forums such as [Fark](#)^[7] and [Slashdot](#), where using "TFA" instead of "the article" has become a [meme](#)^[citation needed]
- RTDA – "read the damn article"
- RTDM – "read the damn menu"
- RTDM – "read the damn manual"
- WABM – "write a better manual" – an answer complaining that the manual is not written well^[8]
- RTFD – "read the fucking documentation"
- RTFE – "read the fucking error"
- RTFM41 – "read the fucking manual for once"
- RTFW – "read the fucking wiki"
- RTFC – "read the fucking chart" – a response that physicians give to [coders](#) who submit inappropriate queries
- RTFS – "read the fucking source" or "read the fucking standard" or "read the fucking syllabus"^[9]
- RTFB – "read the fucking binary"^[10]
- RTFI – "read the fussy instructions"
- RTFQ – "read the fucking question"
- RTMS – "read the manual, stupid"
- STFW – "search the fucking web"
- GIYF – "[Google](#) is your friend"
- JFGI – "just fucking [Google](#) it"
- DYOR – "do your own research"
- LMGTFY, LMGT4U – "let me Google that for you"

RTFM na Wikipedia: <https://en.wikipedia.org/wiki/RTFM>

Documentação: outras

Existem muitas outras fontes de documentação para a linguagem C, tais como:

- Livros**
- Tutoriais na Internet**
- Canais no YouTube**
- Cursos online**
- ...**

Vou citar algumas das melhores quando for apropriado.

Resumo

