C


complexidade
de algoritmos
arrays
compilação
SQL
ponteiros
estrutura
de dados
internet
algoritmos
fundamentos da programação
web
linux
pensamento
computacional
memória
C
fundamentos da computação

C

Lembrando: por que Scratch?

Lembrando: por que Scratch?

Aprender os fundamentos da programação:

- **Funções**
  - (**parâmetros** e **argumentos**)
  - (**retorno** e **efeito colateral**)
- **Condições**
- **Expressões booleanas**
- **Loops**
- **Estruturas de dados**
  - (**variáveis**, **arrays**, **outras**)
- **Conceitos avançados**
  - (**eventos**, **ouvintes**)
  - (**threads**, **outros**)
- **Boas práticas**

Transição: Scratch para C


```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

Linguagens de Alto Nível x Linguagens de Baixo Nível

Linguagens de Alto Nível x Linguagens de Baixo Nível

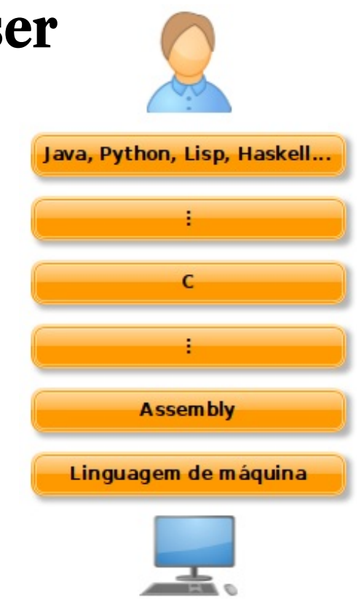
Linguagens de Alto Nível:

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

```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```



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.

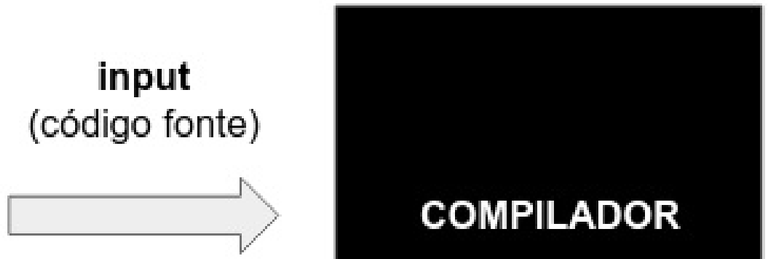
```
01111111 01000101 01001100 01000110 00000010
00000000 00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000 00000001
10110000 00000101 01000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000
11010000 00010011 00000000 00000000 00000000
00000000 00000000 00000000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

Do código fonte em C à linguagem de máquina

Como transformar um programa escrito em linguagem de alto nível para um programa em linguagem de máquina?

Podemos encarar essa transformação com o mesmo modelo de computação que vimos!

Do código fonte em C à linguagem de máquina




```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

```
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
```

Compiladores para C:

GCC (GNU Compiler Collection)

Clang (Clang/LLVM)

Prática da programação

Prática da programação: qualidade do código

Você não deve escrever códigos:

Você deve escrever bons códigos!

- Não se aprende da noite para o dia

- Requer muito tempo e prática

Qualidade de seu código:

Corretude:

seu código faz o que era para fazer?

Design:

elegância, é subjetivo, aprende-se com o tempo

Estilo:

estética do seu código; seguiremos:

https://cs50.readthedocs.io/style/c/

Eficiência:

seu código roda rápido o suficiente?

```c
#include <stdio.h>

int
main(
    void    )
{
printf("%s\n",
                "Olá, mundo!"
              );
    return
0
                ;
}
```

Prática da programação: onde programar?

Editor de Text Puro +
Git/GitHub +
Ambiente de Desenvolvimento Integrado
(integrated development environment - IDE)

Caderno, lápis e borracha

Ferramentas online

- **Visual Studio Code for CS50** (**https://cs50.dev**)
- **Replit** (**https://replit.com**)

Máquina virtual

- **Computação Raiz: LingProg** (**https://we.tl/t-bK9ZIRkT4G**)

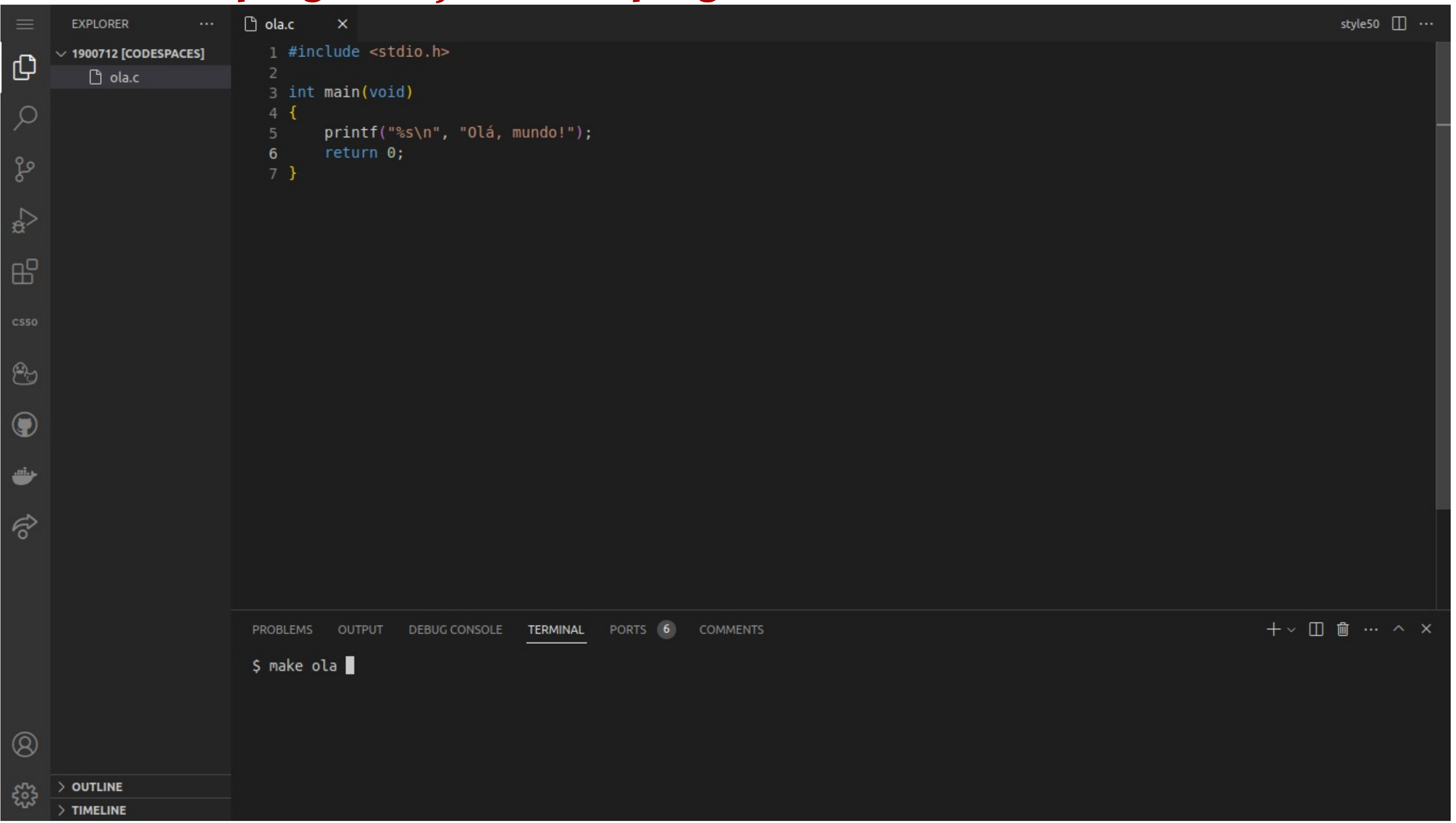
Seu computador Linux/Mac

- **Visual Studio Code** (**https://code.visualstudio.com**)
- **VSCodium** (**https://vscodium.org**)
- **GCC**

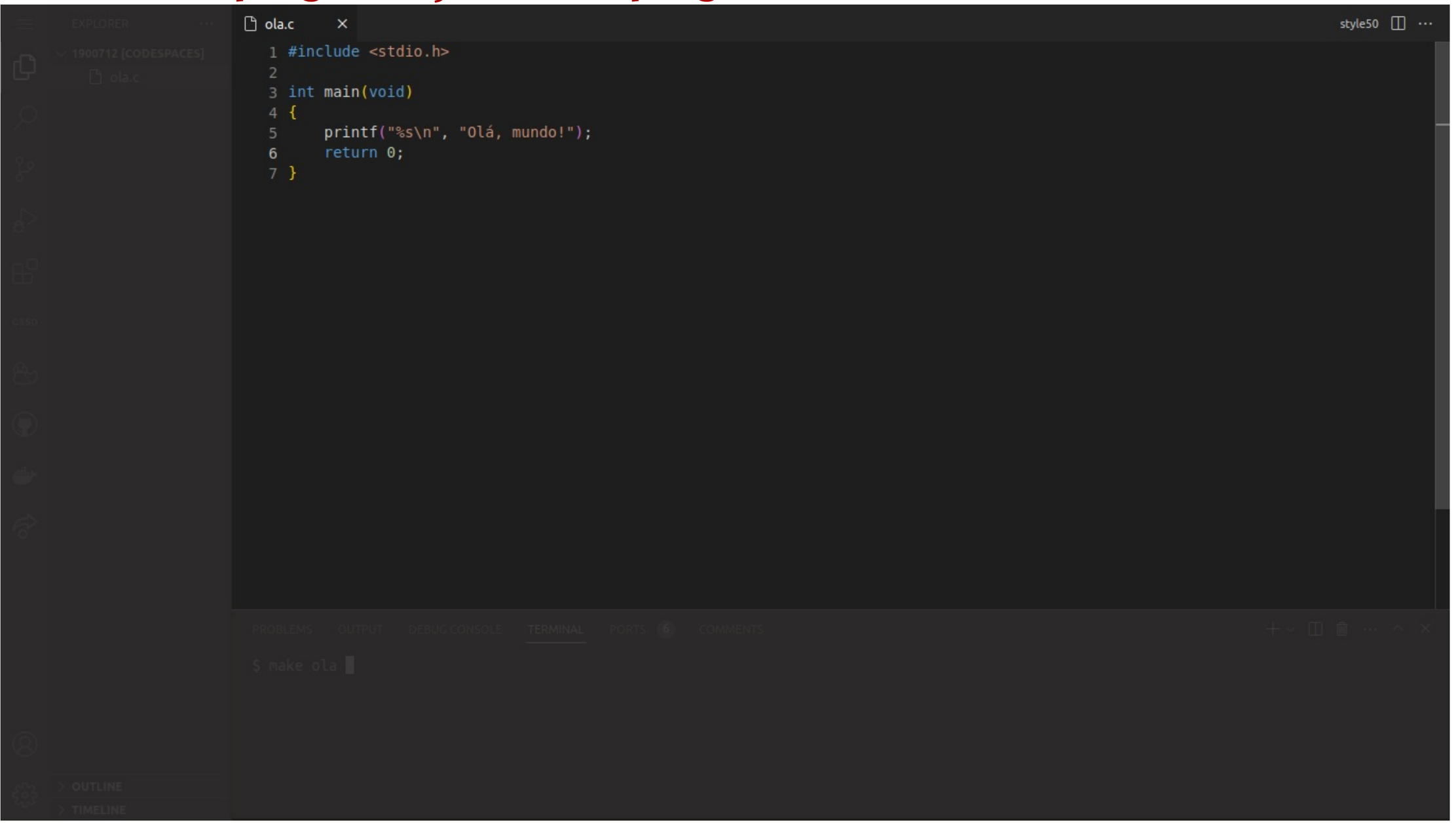
Seu computador Windows (????)

- **Visual Studio Code** (**https://code.visualstudio.com**)
- **MinGW-w64** (**https://mingw-w64.org**)
- **Ubuntu on WSL** (**https://ubuntu.com/wsl**)

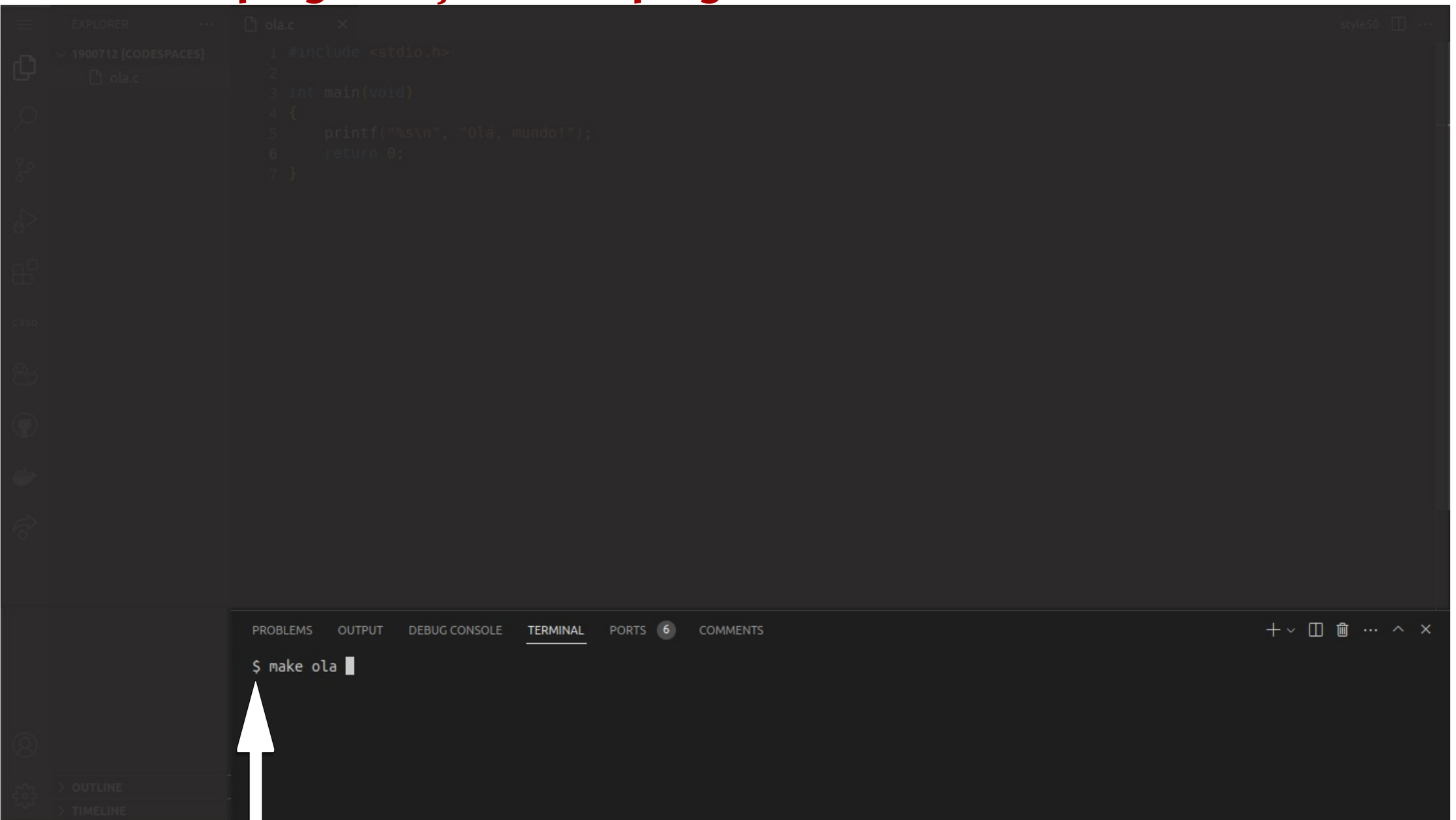
Prática da programação: onde programar?



Prática da programação: onde programar?



Prática da programação: onde programar?



Prompt

Prática da programação: onde programar?

Prática da programação: onde programar?

Prática da programação: demonstração 1

Visual Studio Code for CS50 (cs50.dev)
- necessário conta no GitHub

```
$ code ola.c
```

Cria um arquivo fonte chamado "ola.c"

Não use caracteres especiais (só underscore e letras sem acentos)

Tudo em minúscula

Extensão .c

ola.c

```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

Atenção ao destaque de sintaxe (syntax highlighting)

```
$ make ola
```

Compila o código fonte em código de máquina e cria o programa "ola"

Não coloque o .c no final

```
$ ./ola
```

Executa o programa "ola" no diretório atual

Atenção aos arquivos criados

Prática da programação: entendendo o programa

```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

Prática da programação: entendendo o programa

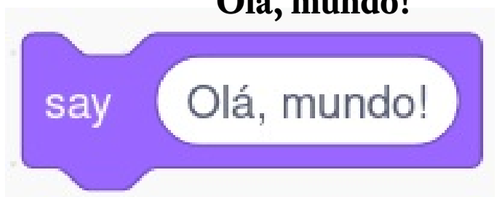
```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

Lembre: o "Olá, mundo!" é um efeito colateral da função, não é o retorno.

Prática da programação: entendendo o programa

String: conjunto de caracteres. Em C devem sempre estar entre aspas duplas: "string"

1 argumento: Olá, mundo!



Atenção: a vírgula do "Olá, mundo!" não é um separador de argumentos, faz parte da string.

2 argumentos: %s\n e Olá, mundo!

```
printf("%s\n", "Olá, mundo!");
```

imprimir formatado

vírgula separa os argumentos

aspas duplas para strings

ponto e vírgula (;) finaliza sentenças

parênteses recebem o(s) argumento(s)

Atenção: a vírgula dentro do "Olá, mundo!" não é um separador de argumentos, é apenas um caractere que faz parte da string.

Prática da programação: erros comuns

Verifique o que ocorre com seu programa quando:

- esquecer o ; no final do printf

- esquecer o \n

- não colocar o \n, mas tentar quebrar a linha antes da última aspas simples

- errar o nome do stdio.h

- errar o nome da printf

Bibliotecas de código e documentação

Bibliotecas de código e documentação

Nosso código não compila sem o "stdio.h".
Por quê?

```c
int main(void)
{
    printf("%s\n", "Olá, mundo!");
    return 0;
}
```

```
$ make ola
ola.c:3:5: error: implicitly declaring library function 'printf' with type 'int
 (const char *, ...)' [-Werror,-Wimplicit-function-declaration]
    printf("%s\n", "Olá, mundo!");
    ^
ola.c:3:5: note: include the header <stdio.h> or explicitly provide a declarati
on for 'printf'
1 error generated.
make: *** [<builtin>: ola] Error 1
$
```

Bibliotecas de código e documentação

Bibliotecas de código: são conjuntos de funções adicionais que outros programadores disponibilizam para usarmos em nossos próprios programas.

Funções relacionadas ficam agrupadas em arquivos chamados de libraries (.o), que podemos acessar através das headers files (.h). Exemplo:

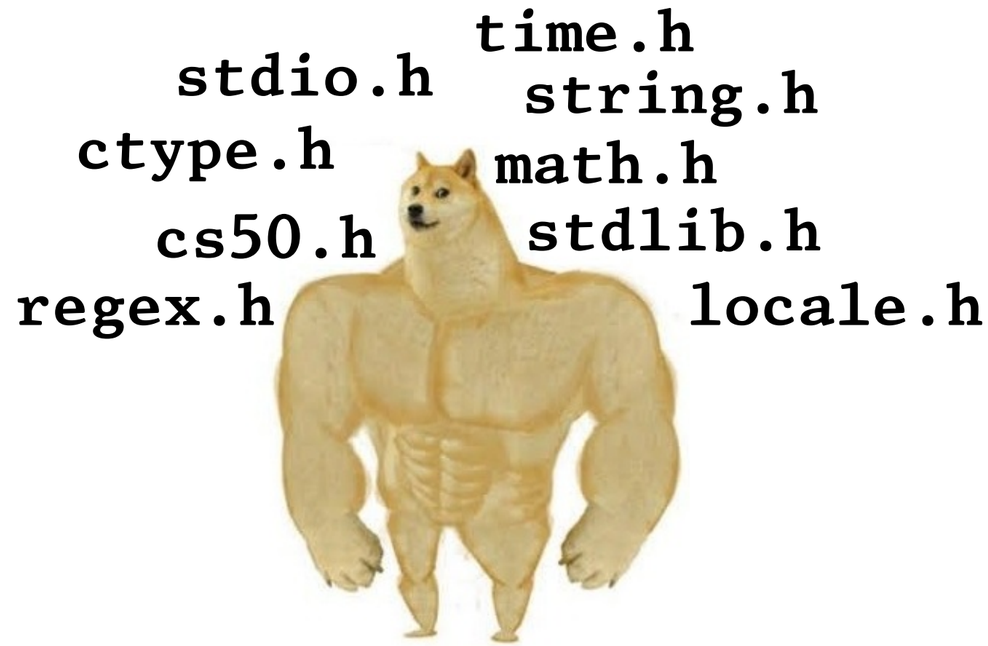
- A library stdio.o é acessada pela header s

Usamos a header file que aponta para a library que tem as funções que precisamos em nosso programa. Podemos usar várias ao mesmo tempo, se necessário.

C

Tecnicamente a coisa toda é um pouco mais complicada, mas veremos isso posteriormente.

```
[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
```



C + bibliotecas

Bibliotecas de código e documentação

INTERNATIONAL STANDARD

ISO/IEC 9899

Fourth edition
2018-07

Information technology — Programming languages — C

Technologies de l'information — Langages de programmation — C

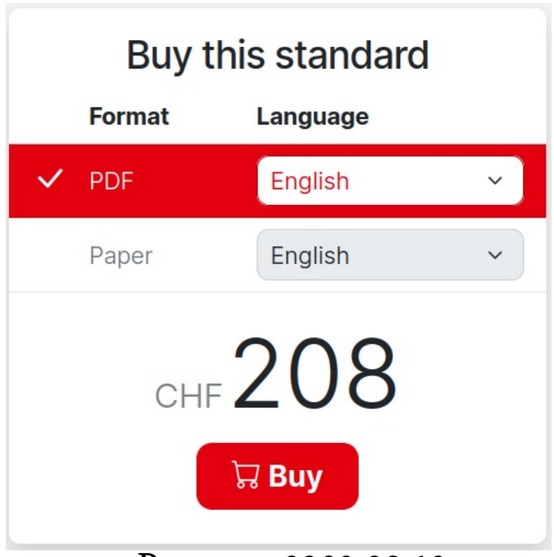
Reference number
ISO/IEC 9899:2018(E)

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

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

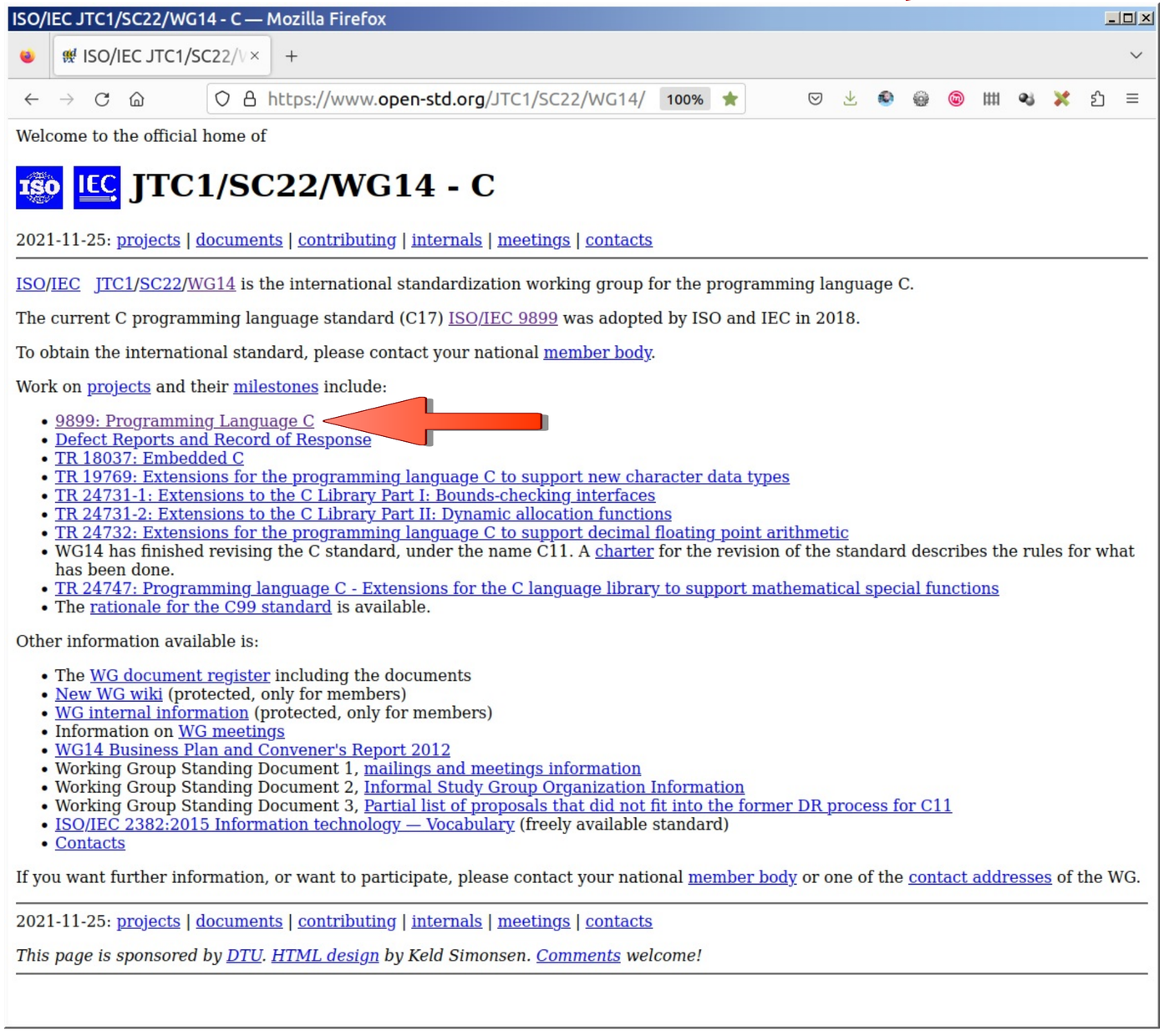
www.iso.org/standard/74528.html

É caro!



Preço em 2023-08-12

Bibliotecas de código e documentação



Na página do grupo de trabalho da ISO é possível acessar gratuitamente PDFs dos rascunhos dos padrões:
www.open-std.org/JTC1/SC22/WG14

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	N1570 [2011-04-04]
C99	ISO/IEC 9899:1999	N1256 [2007-09-07]
C89	ISO/IEC 9899:1990	Not available

Bibliotecas de código e documentação

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

The Open Group Bas ×

https://pubs.opengroup.org/onlinepubs/9699919799/ 110%

INDEX

Search..

[Alphabetic | Topic | Word Search]

Select a Volume:
[*Base Definitions* | *System Interfaces* | *Shell & Utilities* | *Rationale*]

[Frontmatter]

[Main Index]

Base Definitions

1. Introduction
2. Conformance
3. Definitions
4. General Concepts
5. File Format Notation
6. Character Set
7. Locale
8. Environment Variables
9. Regular Expressions
10. Directory Structure and Devices
11. General Terminal Interface
12. Utility Conventions
13. Headers

IEEE

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

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]

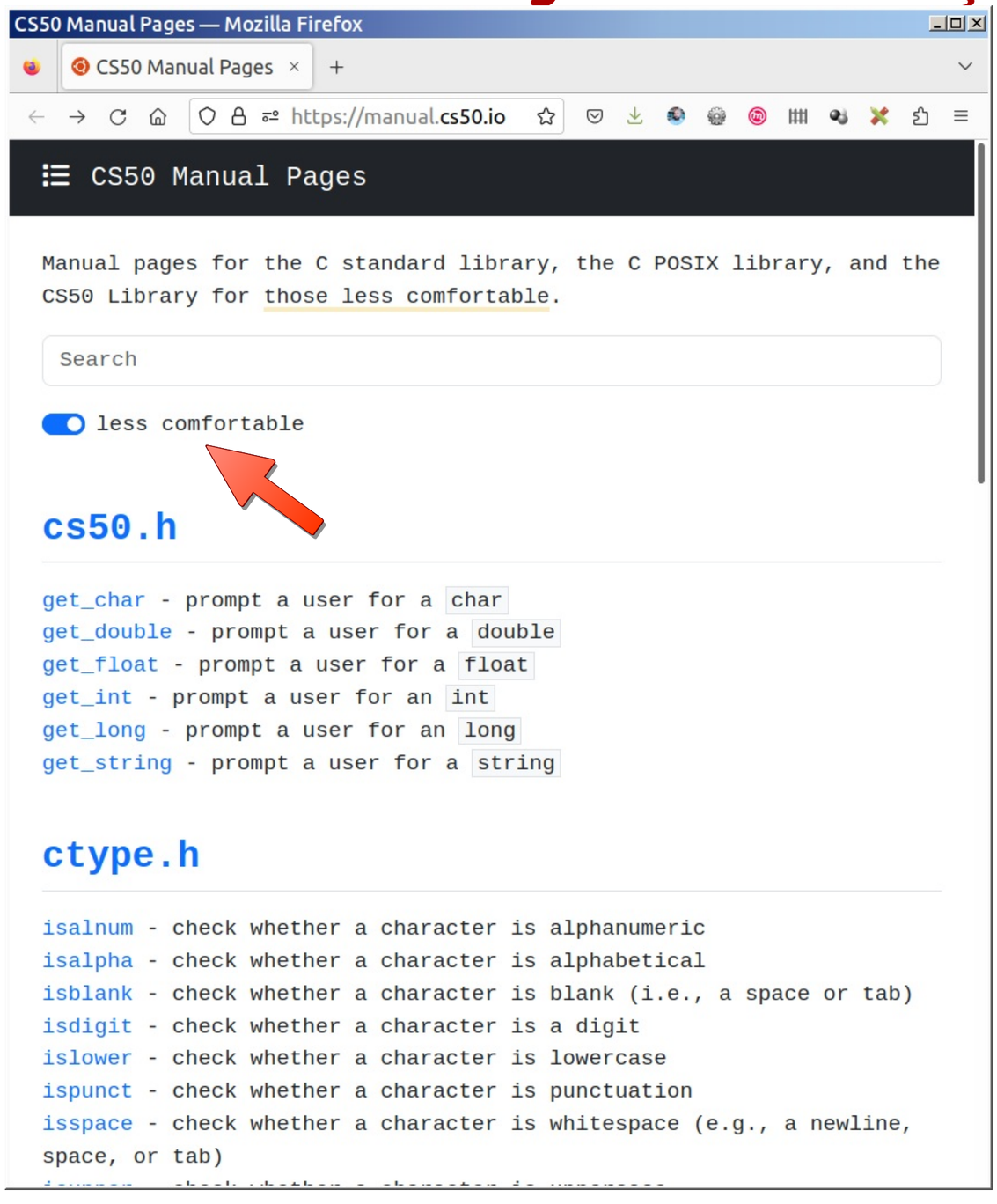
Previous Editions

Links: [2008 | 2013 | 2016]

No "The Open Group" é possível acessar a documentação das header files:

https://pubs.opengroup.org/onlinepubs/9699919799/

Bibliotecas de código e documentação



A Universidade de Harvard pegou as header files disponíveis no site do The Open Group e "simplificou" a formatação para as mais utilizadas pelos alunos da CS50, a disciplina de Introdução à Ciência da Computação.

Essa documentação está disponível em:

https://manual.cs50.io

É essa documentação que você deve consultar para aprender a usar as funções e bibliotecas em seus programas!

Obrigado CS50!

Bibliotecas de código e documentação

"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
```

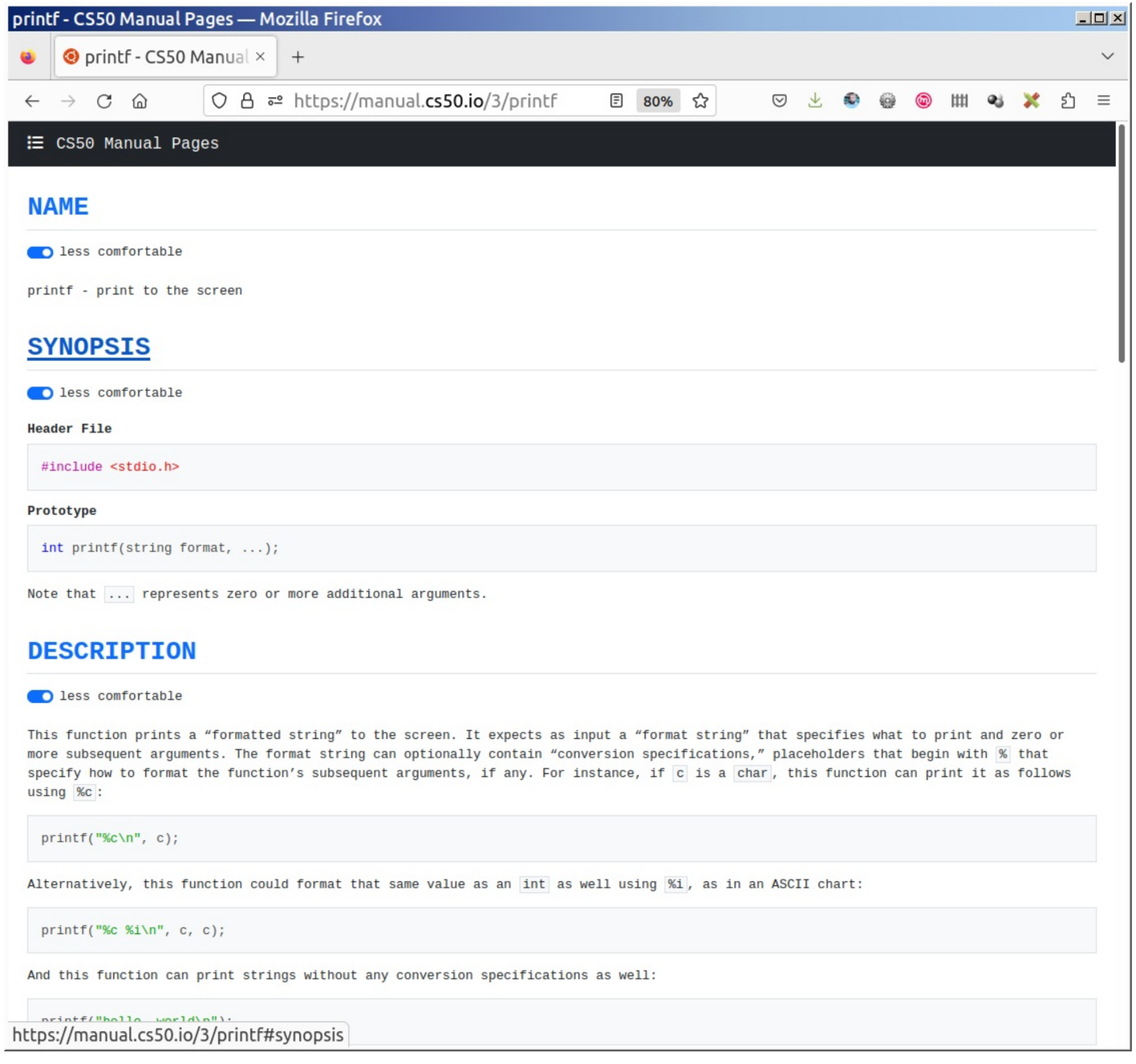
Documentação disponível em:
https://manual.cs50.io

"more confortable"

stdio.h

```
__fbufsize - interfaces to stdio FILE structure
__flbf - 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
```

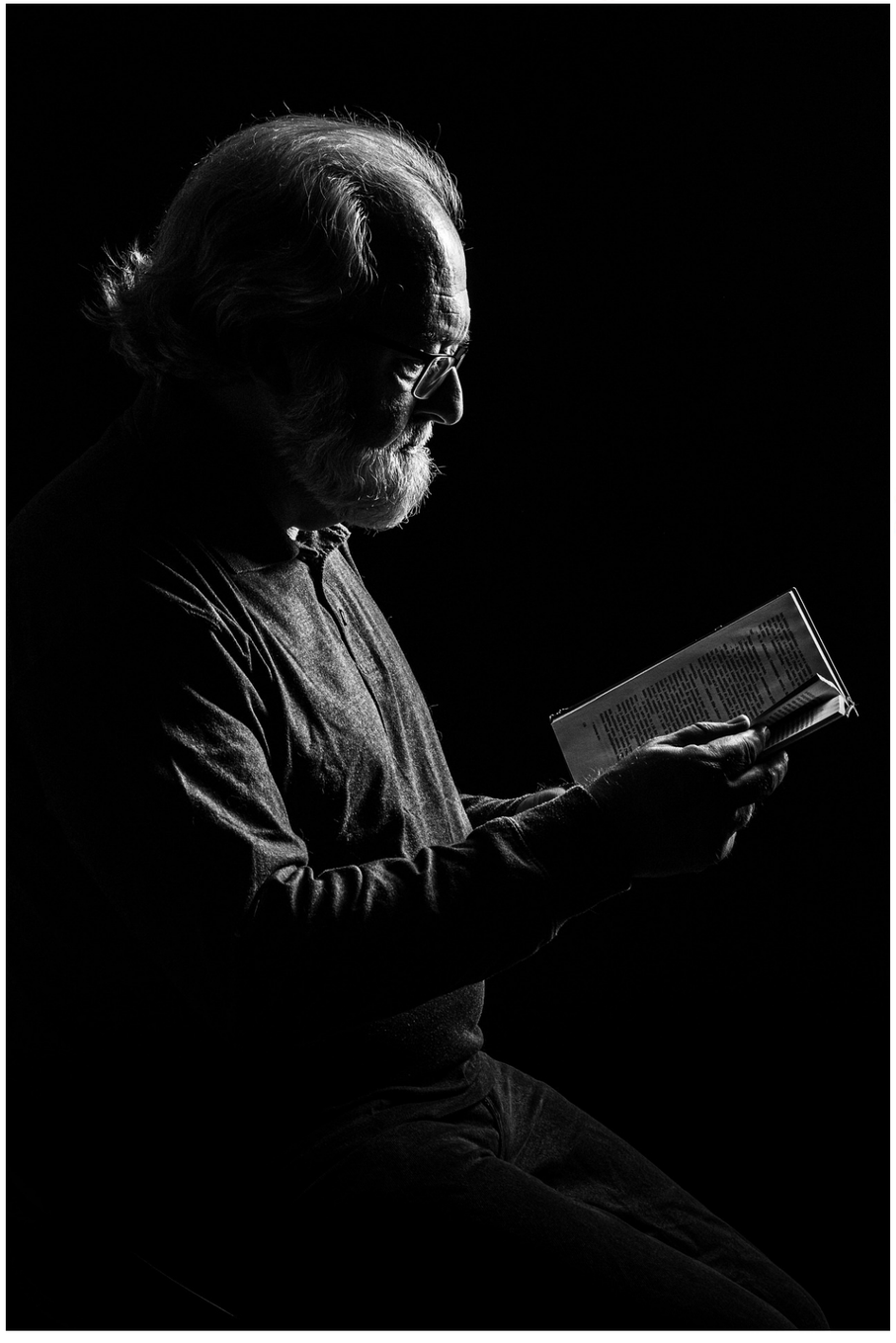
Bibliotecas de código e documentação



Documentação da função printf:

- Name
- Synopsis
- Description
- Return values
- Examples
- Attributes
- Conforming to
- Notes
- Bugs
- See also
- Colophon

Bibliotecas de código e documentação



Em resumo:

RTFM

(Read The Fucking Manual)

RTFM, Wikipedia (https://en.wikipedia.org/wiki/RTFM)

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"

Obtendo e mostrando dados: input e output revisitados

Obtendo e mostrando dados: input e output revisitados

Em C:

- Output é relativamente fácil:

```
printf("%s\n", "Olá, mundo!");
```

- Input é difícil!

Algumas funções da `cs50.h`:

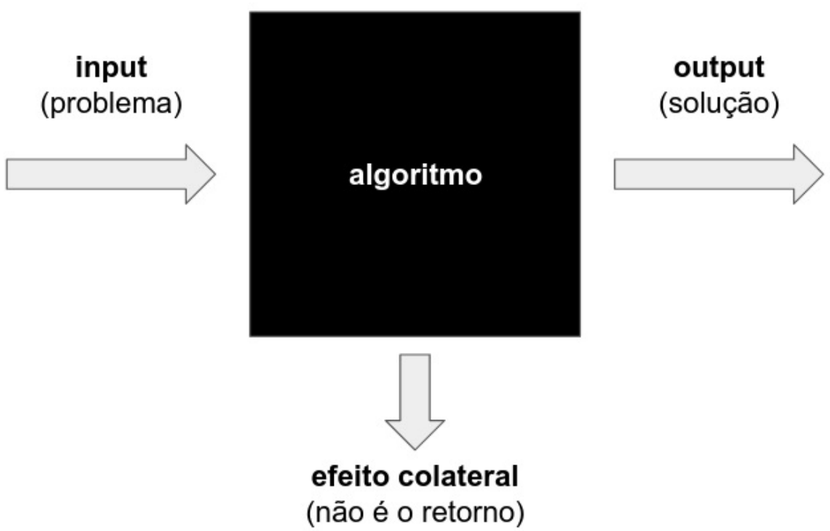
- get_char
- get_float
- get_double
- get_int
- get_long
- get_string
- ...

`cs50.h`

Obtendo e mostrando dados: input e output revisitados

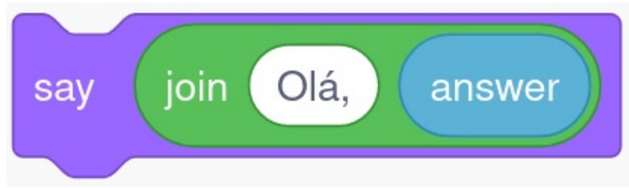
O que de fato está ocorrendo aqui?

1. **A função "ask" é chamada e o usuário passa para ela o argumento "Qual o seu nome";**
2. **A função "ask" imprime o argumento na tela (e isso é um efeito colateral) e aguarda o input do usuário**
3. **O usuário digita um nome**
4. **A função "ask" retorna o valor em uma variável chamada de "answer" (o valor retornado é o nome digitado)**
5. **O retorno da função pode ser utilizado posteriormente, por exemplo, na função "say"**

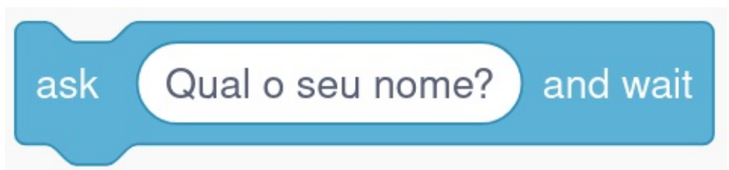


Obtendo e mostrando dados: input e output revisitados

O valor de retorno da função "`ask`" (ou simplesmente retorno) foi colocado dentro da variável "`answer`". Esse retorno pode ser utilizado em outras partes do programa.



Obtendo e mostrando dados: input e output revisitados



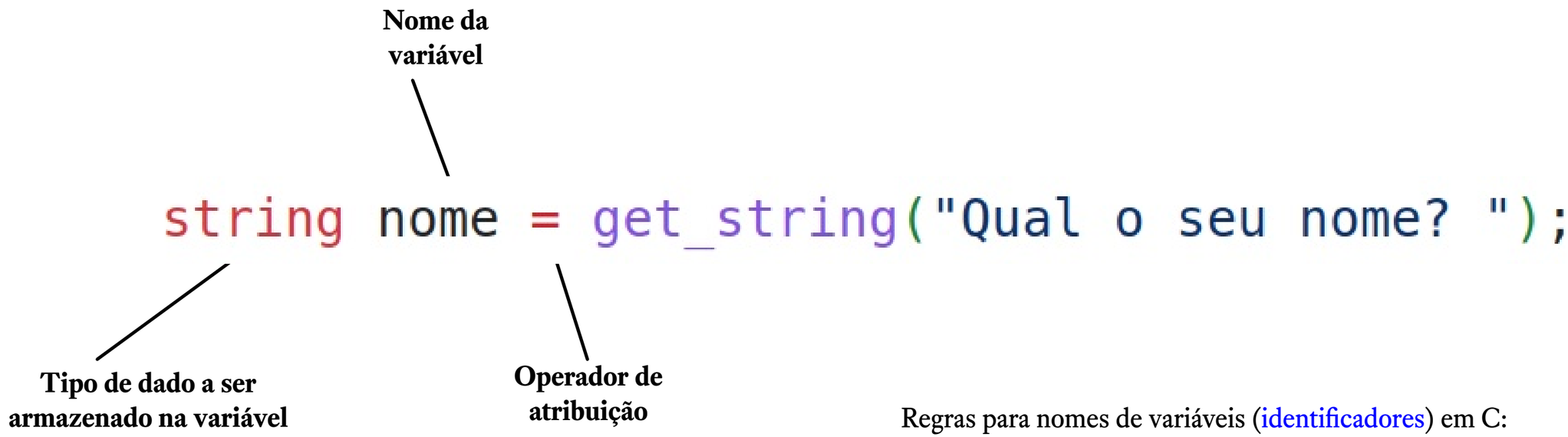

```c
#include <cs50.h>

string nome = get_string("Qual o seu nome? ");
```

Crie a variável nome para armazenar uma string, recebendo o retorno da função get_string chamada com o argumento "Qual o seu nome? ".

A variável nome passa a conter o retorno da função e esse retorno pode ser utilizado posteriormente em outras partes do seu programa.

Uso da função get_string( ):
- Argumento: string que será exibida para o usuário (efeito colateral), entre aspas duplas
- Retorno: a string digitada pelo usuário

Obtendo e mostrando dados: input e output revisitados



Regras para nomes de variáveis (identificadores) em C:
- Não pode ser nenhuma palavra chave ou reservada da linguagem
- O nome deve ser único
- Não pode começar com um número
- Deve começar com uma letra ou um underscore e, depois, pode ter letras, números ou outros underscores
- Caracteres especiais, letras com acentos, espaços, etc., não são permitidos
- Letras minúsculas são diferentes de maiúsculas
- O tamanho não deve exceder 31 caracteres

Conceitos importantes:

- **Declarar** uma variável (definir nome e tipo):

```
string nome;
```

- **Atribuir** uma variável (colocar um valor):

```
nome = get_string("Qual o seu nome? ");
```

- **Inicializar** uma variável (definir nome e tipo, e colocar valor):

```
string nome = get_string("Qual o seu nome? ");
```

Obtendo e mostrando dados: input e output revisitados

Qual o bug neste código?

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

int main(void)
{
    string nome = get_string("Qual o seu nome? ");
    printf("%s\n", "Olá, nome!");
    return 0;
}
```

Obtendo e mostrando dados: input e output revisitados

Placeholders com format specifiers e escape sequences: entendendo o "%s\n" !

```
printf("%s\n", "Olá, mundo!");
```

1º argumento:
Formato do output com placeholders para valores, criados com os format specifiers. Pode ter escape sequences.

2º argumento:
Valor a ser colocado no placeholder criado por um format code.

Obs.: para cada format specifier deve haver um argumento para preenchê-lo. O preenchimento é feito da esquerda para direita, na ordem em que aparecem no 1º argumento.

Placeholders: são definidos pelos format specifiers criados com "% + letra", por exemplo: %i, %d, %f, %s, %p, %g. Indicam o local dos valores na string que será impressa. São específicos para o tipo de dado.

Escape sequences: são criados com "\ + letra", por exemplo: \n, \t, \a, \b, \r, \v. Servem para representar caracteres especiais.

Obtendo e mostrando dados: input e output revisitados

Bug corrigido!

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

int main(void)
{
    string nome = get_string("Qual o seu nome? ");
    printf("Olá, %s!\n", nome);
    return 0;
}
```

```
$ make ola
$ ./ola
Qual o seu nome? Abrantes
Olá, Abrantes!
```

Obtendo e mostrando dados: input e output revisitados

Exemplo mais complexo:

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

int main(void)
{
    string nome = get_string("Qual o seu nome? ");
    string sobrenome = get_string("Qual seu sobrenome? ");
    int idade = get_int("Qual sua idade? ");

    printf("Olá, %s %s! Você tem %i anos, parabéns!\n", nome, sobrenome, idade);

    return 0;
}
```

```
$ make ola
$ ./ola
Qual o seu nome? Abrantes
Qual seu sobrenome? A. Silva Filho
Qual sua idade? 49
Olá, Abrantes A. Silva Filho! Você tem 49 anos, parabéns!
```

Obtendo e mostrando dados: input e output revisitados

Existem inúmeros format specifiers e escape sequences. Não é necessário decorar, consulte a documentação quando for necessário. Boa referência para format specifiers:

https://cplusplus.com/reference/cstdio/printf/

A ***format specifier*** follows this prototype: [see compatibility note below]

`%[flags][width][.precision][length]specifier`

Where the ***specifier character*** at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	**Output**	**Example**
d ***or*** i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

The ***format specifier*** can also contain sub-specifiers: ***flags***, ***width***, ***.precision*** and ***modifiers*** (in that order), which are optional and follow these specifications:

flags	**description**
-	Left-justify within the given field width; Right justification is the default (see ***width*** sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceeded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see ***width*** sub-specifier).

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The ***width*** is not specified in the ***format*** string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	**description**
.number	For integer specifiers (d, i, o, u, x, X): ***precision*** specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A ***precision*** of 0 means that no character is written for the value 0. For a, A, e, E, f and F specifiers: this is the number of digits to be printed **after** the decimal point (by default, this is 6). For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for ***precision***, 0 is assumed.
.*	The ***precision*** is not specified in the ***format*** string, but as an additional integer value argument preceding the argument that has to be formatted.

Obtendo e mostrando dados: input e output revisitados

Existem inúmeros format specifiers e escape sequences. Não é necessário decorar, consulte a documentação quando for necessário. Boa referência para escape sequences:

https://en.cppreference.com/w/c/language/escape

Escape sequence	Description	Representation
Simple escape sequences		
\'	single quote	byte 0x27 in ASCII encoding
\"	double quote	byte 0x22 in ASCII encoding
\?	question mark	byte 0x3f in ASCII encoding
****	backslash	byte 0x5c in ASCII encoding
\a	audible bell	byte 0x07 in ASCII encoding
\b	backspace	byte 0x08 in ASCII encoding
\f	form feed - new page	byte 0x0c in ASCII encoding
\n	line feed - new line	byte 0x0a in ASCII encoding
\r	carriage return	byte 0x0d in ASCII encoding
\t	horizontal tab	byte 0x09 in ASCII encoding
\v	vertical tab	byte 0x0b in ASCII encoding
Numeric escape sequences		
*****nnn*	arbitrary octal value	code unit *nnn*
\x*n...*	arbitrary hexadecimal value	code unit *n...* (arbitrary number of hexadecimal digits)
Universal character names		
\u*nnnn* (since C99)	Unicode value in allowed range; may result in several code units	code point U+*nnnn*
\U*nnnnnnnn* (since C99)	Unicode value in allowed range; may result in several code units	code point U+*nnnnnnnn*

Tipos de dados

Tipos de dados

Um tipo de dado é a definição de um conjunto de valores e das operações que são possíveis de serem realizadas com esse conjunto de valores:

tipo de dado = valores + operações

Exemplo: tipo de dado int

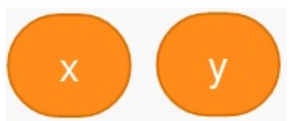
- **Conjunto de valores: ..., -3, -2, -1, 0, 1, 2, 3, ...**
- **Operações: soma, subtração, multiplicação, divisão, módulo, ...**

Existem certos limites aos valores que podem ser armazenados em um tipo de dado, e alguns tipos não existem no C padrão (mas a cs50.h e outras header files fornecem). Veremos com o tempo. O importante é que variáveis em C precisam de um tipo de dado específico.

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

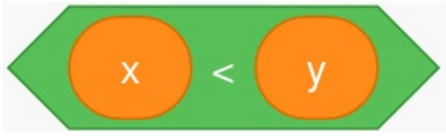
Condicionais e expressões booleanas

Condicionais e expressões booleanas

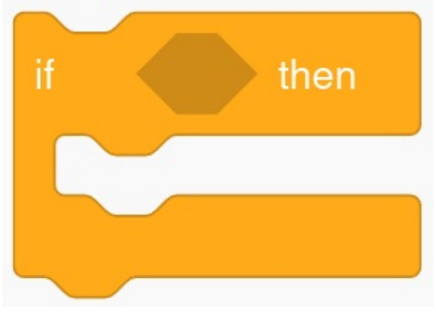


variáveis

```
int x = 3;
int y = 4;
```



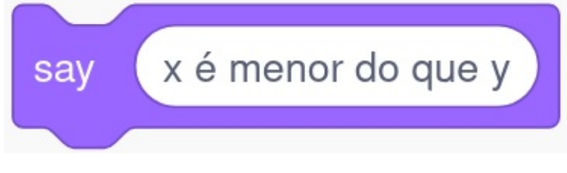
expressão booleana `x < y`



estrutura condicional

```
if ( )
{

}
```

Se há somente 1 sentença "dentro" do IF, as chaves não são necessárias. Mas, de qualquer forma, é uma boa norma de estilo usá-las.



mostrar o resultado `printf("%s\n", "x é menor do que y");`

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

int main(void)
{
    int x = 3;
    int y = 4;

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }

    return 0;
}
```

Condicionais e expressões booleanas


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

int main(void)
{
    int x = 5;
    int y = 4;

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }
    else
    {
        printf("%s\n", "x não é menor do que y");
    }

    return 0;
}
```

```c
if ( )
{

}
else
{

}
```

Note que o "else" é opcional, só utilizamos quando necessário.

Condicionais e expressões booleanas


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

int main(void)
{
    int x = 4;
    int y = 4;

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }
    else if (x > y)
    {
        printf("%s\n", "x é maior do que y");
    }
    else if (x == y)
    {
        printf("%s\n", "x é igual a y");
    }

    return 0;
}
```

```c
if ( )
{

}
else if ( )
{

}
else if ( )
{

}
```

Note que:
- O "else if" é opcional, só utilizamos quando necessário;
- Podemos usar quantos "else if" quisermos;
- A comparação de igualdade em C é com "==", atenção!

Condicionais e expressões booleanas


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

int main(void)
{
    int x = 4;
    int y = 4;

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }
    else if (x > y)
    {
        printf("%s\n", "x é maior do que y");
    }
    else
    {
        printf("%s\n", "x é igual a y");
    }

    return 0;
}
```

```c
if ( )
{

}
else if ( )
{

}
else
{

}
```

Note que:
- O "else if" é opcional, só utilizamos quando necessário;
- Podemos usar quantos "else if" quisermos;
- O "else" final é opcional, só usamos quando necessário; e
- Só pode existir 1 único "else" no final;

Condicionais e expressões booleanas

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

int main(void)
{
    int x = get_int("Qual o valor de x? ");
    int y = get_int("Qual o valor de y? ");

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }

    return 0;
}
```

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

int main(void)
{
    int x = get_int("Qual o valor de x? ");
    int y = get_int("Qual o valor de y? ");

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }
    else if (x > y)
    {
        printf("%s\n", "x é maior do que y");
    }

    return 0;
}
```

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

int main(void)
{
    int x = get_int("Qual o valor de x? ");
    int y = get_int("Qual o valor de y? ");

    if (x < y)
    {
        printf("%s\n", "x é menor do que y");
    }
    else if (x > y)
    {
        printf("%s\n", "x é maior do que y");
    }
    else
    {
        printf("%s\n", "x é igual a y");
    }

    return 0;
}
```

Condicionais e expressões booleanas

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

int main(void)
{
    char c = get_char("Você concorda (s/n)? ");

    if (c == 's')
    {
        printf("%s\n", "Concordou.");
    }
    else if (c == 'n')
    {
        printf("%s\n", "Não concordou.");
    }

    return 0;
}
```

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

int main(void)
{
    char c = get_char("Você concorda (s/n)? ");

    if (c == 's')
    {
        printf("%s\n", "Concordou.");
    }
    else if (c == 'S')
    {
        printf("%s\n", "Concordou.");
    }
    else if (c == 'n')
    {
        printf("%s\n", "Não concordou.");
    }
    else if (c == 'N')
    {
        printf("%s\n", "Não concordou.");
    }

    return 0;
}
```

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

int main(void)
{
    char c = get_char("Você concorda (s/n)? ");

    if (c == 's' || c == 'S')
    {
        printf("%s\n", "Concordou.");
    }
    else if (c == 'n' || c == 'N')
    {
        printf("%s\n", "Não concordou.");
    }

    return 0;
}
```

Tipo `char`:

- **Armazena 1 único caractere!**

Atenção ao uso das aspas!

- **Para strings, usa-se aspas duplas;**
- **Para char, usa-se aspas simples.**

Condicionais e expressões booleanas

É importante dominar o uso dos operadores aritméticos, relacionais e lógicos que podem ser utilizados nas expressões booleanas. Boa referência para isso:

https://www.tutorialspoint.com/cprogramming/c_operators.htm

Operator	Description
+	Adds two operands.
–	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and remainder of after an integer division.

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

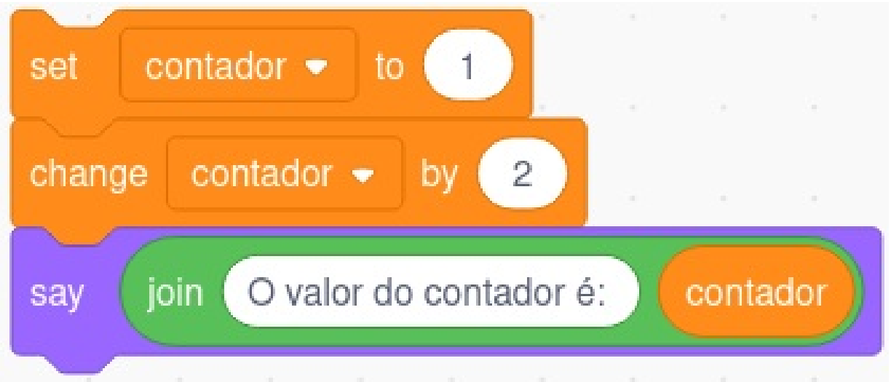
Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Atribuição de valor à variáveis: alguns detalhes importantes

Atribuição de valor à variáveis: alguns detalhes importantes

Alterar o valor de uma variável fazendo operações com ela mesma é uma operação extremamente comum!

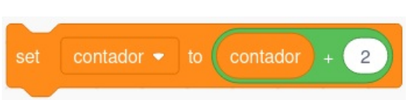
```c
#include <stdio.h>

int main(void)
{
    int contador = 1;

    contador = contador + 2;

    printf("O valor do contador é: %i\n", contador);

    return 0;
}
```



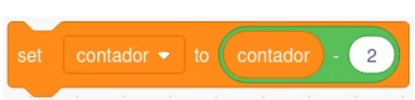
Atribuição de valor à variáveis: alguns detalhes importantes

Syntactic sugar: açúcar sintático para operações de atribuição

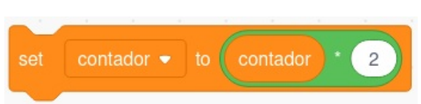
```
contador = contador + 2;
contador += 2;
```



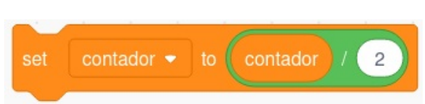

```
contador = contador - 2;
contador -= 2;
```



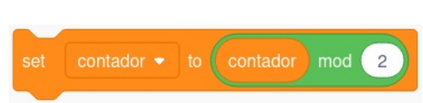

```
contador = contador * 2;
contador *= 2;
```




```
contador = contador / 2;
contador /= 2;
```

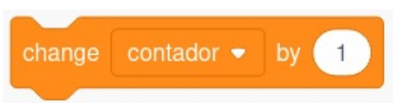



```
contador = contador % 2;
contador %= 2;
```

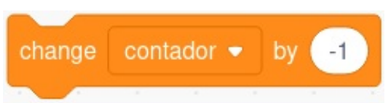


Se o aumento ou a diminuição envolver a unidade (1), existem outras formas de açúcar sintático:

- **++ (pós ou pré-fixada)**
- **-- (pós ou pré-fixada)**




```
contador = contador + 1;
contador += 1;
contador++;
++contador;
```




```
contador = contador - 1;
contador -= 1;
contador--;
--contador;
```

Loops

Loops

Problema: fazer o gato miar 3 vezes.

- O código abaixo está correto?

- O código abaixo está bem projetado (bom design)?

```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", "miau");
    printf("%s\n", "miau");
    printf("%s\n", "miau");

    return 0;
}
```

Loops

Existem 3 grandes formas de loop na linguagem C (algumas formas têm variações):

- **while** loop
- **for** loop
- **do while** loop

```
while ( )
{

}
```

Expressão Booleana

```
for (   ;   ;   )
{

}
```

Contador — Expressão Booleana — Mudança no contador

```
do
{

}
while ( );
```

Expressão Booleana

Loops

While loop:

- **continua executando os comandos dentro de seu corpo, enquanto uma expressão booleana for verdadeira;**
- **bom para quando não sabemos exatamente quantas vezes o loop será executado (o controle é com a exp. booleana);**
- **pode não executar nenhuma vez;**
- **temos que garantir que a condição se torne falsa, ou teremos uma condição chamada de loop infinito (geralmente é erro).**

```c
#include <stdio.h>

int main(void)
{
    int c = 3;

    while (c > 0)
    {
        printf("%s\n", "miau");
        c--;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int contador = 3;

    while (contador > 0)
    {
        printf("%s\n", "miau");
        contador = contador - 1;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int c = 3;

    while (c > 0)
    {
        printf("%s\n", "miau");
        c = c - 1;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int c = 3;

    while (c > 0)
    {
        printf("%s\n", "miau");
        c -= 1;
    }

    return 0;
}
```

Loops

Cientistas da computação começam a contagem sempre do zero! Acostume-se com isso o mais rápido possível!

```c
#include <stdio.h>

int main(void)
{
    int c = 1;

    while (c <= 3)
    {
        printf("%s\n", "miau");
        c++;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int c = 0;

    while (c < 3)
    {
        printf("%s\n", "miau");
        c++;
    }

    return 0;
}
```

Edsger Wybe Dijkstra: "Why numbering should start at zero"
https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

Loops

For loop:

- **executa os comandos dentro de seu corpo um determinado número pré-definido de vezes;**
- **bom para quando sabemos exatamente a quantidade de vezes que o loop tem que executar;**
- **o controle de quantas vezes foi executado é automático.**

```c
#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 3; i++)
    {
        printf("%s\n", "miau");
    }

    return 0;
}
```

Loops

Do While loop:

- **praticamente igual ao while loop, com uma única exceção: a expressão booleana é verificada após o corpo e, portanto, ele sempre executa uma vez (a primeira vez). Lembre-se de que o while loop pode não executar nenhuma vez.;**
- **bom para solicitar dados ao usuário e mostrar menus.**

```c
#include <stdio.h>

int main(void)
{
    int c = 0;

    do
    {
        printf("%s\n", "miau");
        c++;
    }
    while (c < 3);

    return 0;
}
```

Loops

Loop infinito: em geral é erro na programação, mas pode ser correto (escutar por um evento).


```c
#include <stdio.h>

int main(void)
{
    while (1)
    {

    }
}
```

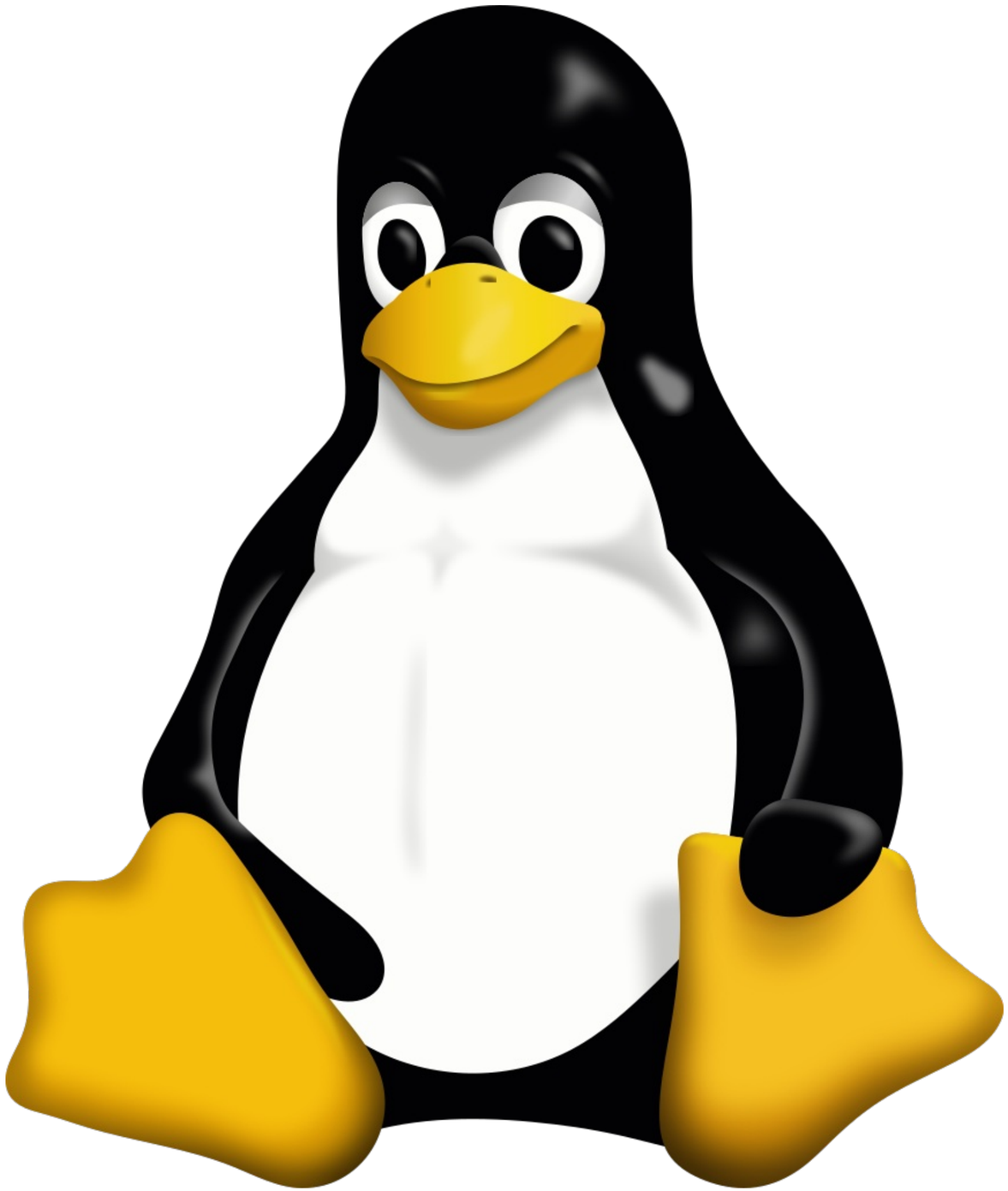
Qualquer coisa que não seja 0 (zero) é considerada como verdade.

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

int main(void)
{
    while (true)
    {

    }
}
```

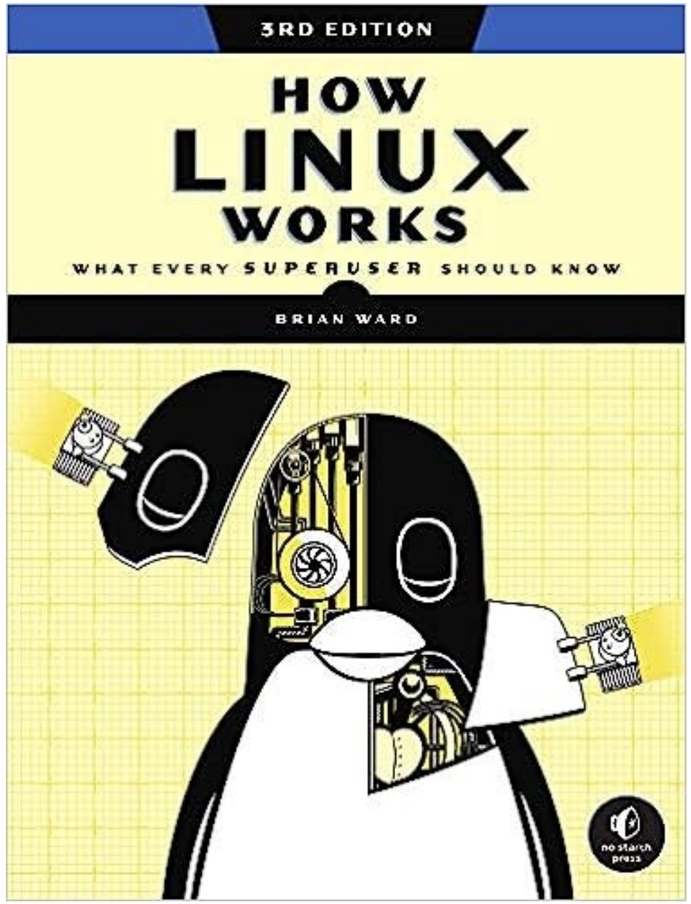
`cs50.h` já inclui `stdbool.h`.

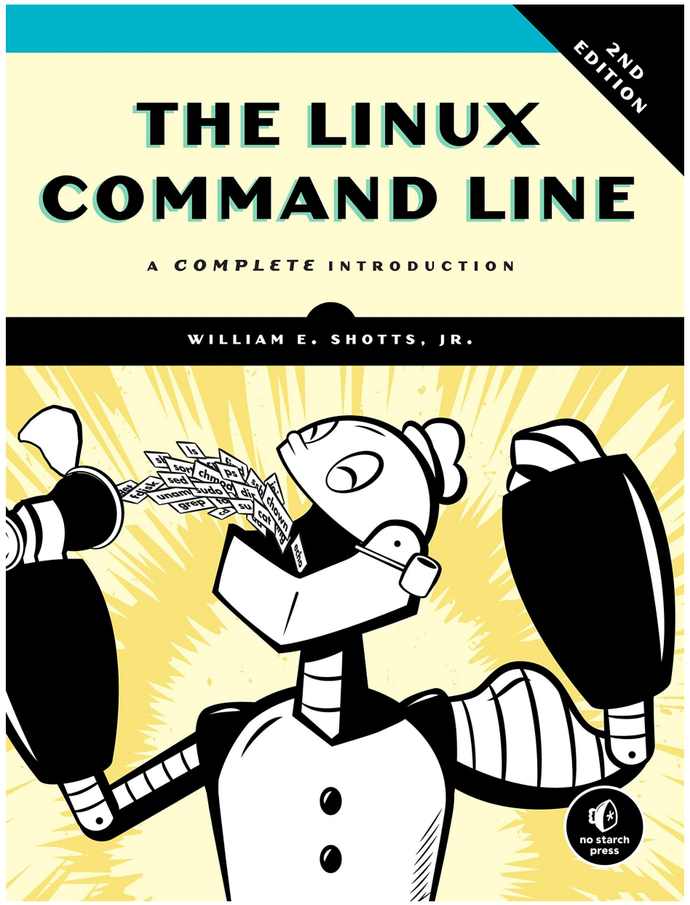
`Ctrl-c` interrompe um loop infinito!

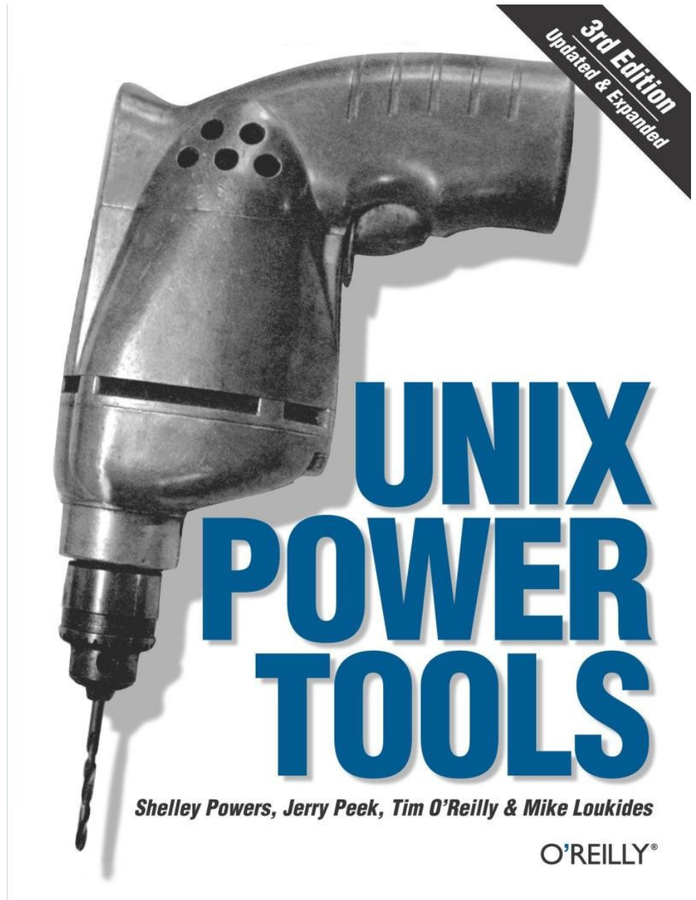
Linux



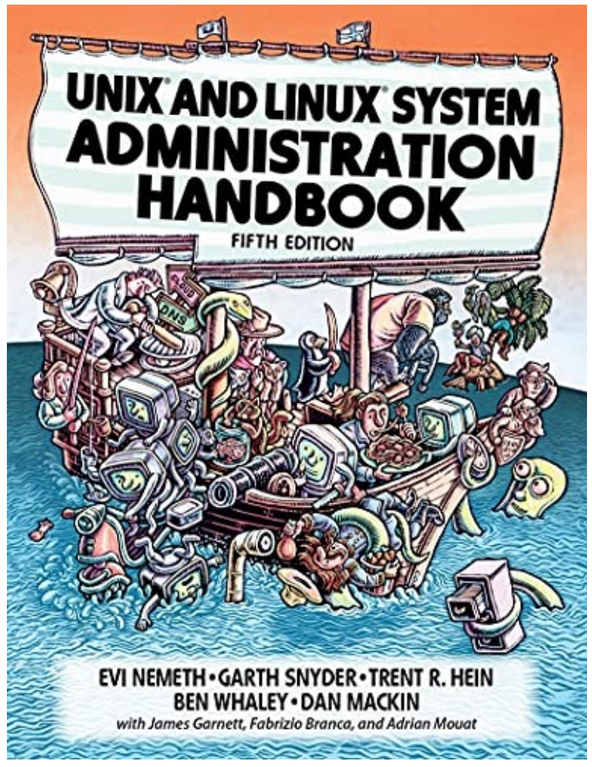
Linux

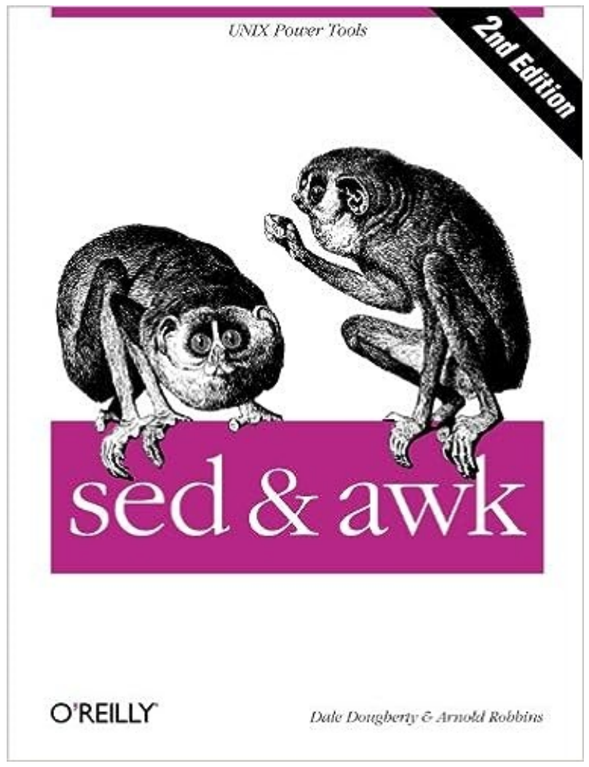

3RD EDITION
HOW LINUX WORKS
WHAT EVERY SUPERUSER SHOULD KNOW
BRIAN WARD
no starch press

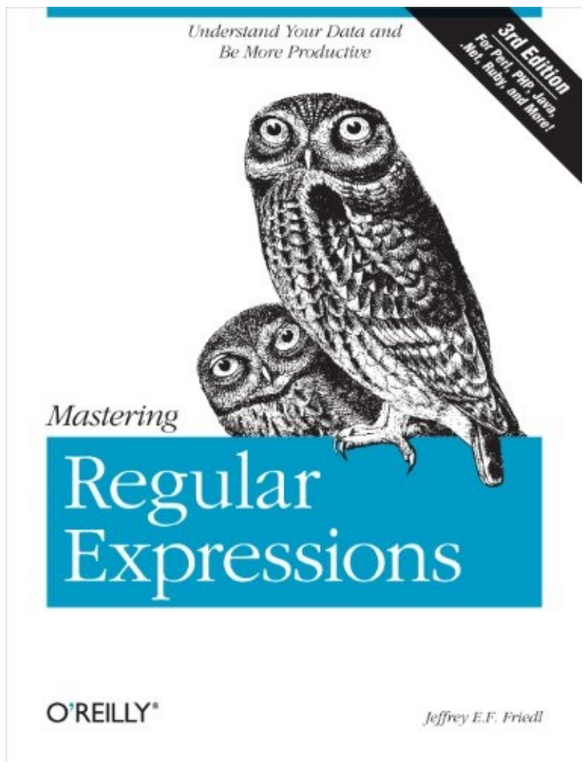

2ND EDITION
THE LINUX COMMAND LINE
A COMPLETE INTRODUCTION
WILLIAM E. SHOTTS, JR.
no starch press


3rd Edition
Updated & Expanded
UNIX POWER TOOLS
Shelley Powers, Jerry Peek, Tim O'Reilly & Mike Loukides
O'REILLY®

Linux


UNIX AND LINUX SYSTEM ADMINISTRATION HANDBOOK
FIFTH EDITION
EVI NEMETH • GARTH SNYDER • TRENT R. HEIN
BEN WHALEY • DAN MACKIN
with James Garnett, Fabrizio Branca, and Adrian Mouat


UNIX Power Tools
2nd Edition
sed & awk
O'REILLY
Dale Dougherty & Arnold Robbins


Understand Your Data and Be More Productive
3rd Edition
For Perl, PHP, Java, .Net, Ruby, and More!
Mastering
Regular Expressions
O'REILLY
Jeffrey E.F. Friedl


5ª EDIÇÃO
Revisada e ampliada
EXPRESSÕES REGULARES
UMA ABORDAGEM DIVERTIDA
(bb|[^b]{2})
[ER]
novatec
Aurelio Marinho Jargas
www.aurelio.net

Linux

Command Line Interface (CLI):
- Separa os meninos dos homens.

Alguns comandos mais comuns:

cd
cp
ls
mkdir
mv
rm
rmdir

$$\mathrm{CLI} = (\mathrm{produtividade})^{\infty}$$

$$\mathrm{Linux} = [(\mathrm{produtividade})^{\infty}]^{(\mathrm{produtividade})^{\infty}}$$

Comece a aprender Linux ainda HOJE, pelo menos com uma máquina virtual.

UFA!

Hora de esfriar a cabeça!

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)

Ferramenta poderosa ou magia sem controle?

- texto grande padrão

"Texto longo para leitura"
citação

Mitologia Grega: Orion, o gigante cego, carregando seu servo Cedalion para atuar como seus olhos. Circa 1410. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Library_of_Congress,_Rosenwald_4,_Bl._5r.jpg)

John of Salisbury teaching philosofy. Circa 1520. Public domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Policraticon_de_Jean_de_Salisbury_-_BSG_Ms1145_f3r_(Jean_de_Salisbury_enseignant).jpeg)

Retrado de Isaac Newton, por Godfrey Kneller, em 1689. Public Domain.
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689.jpg)