FEELT31201
Programação Procedimental

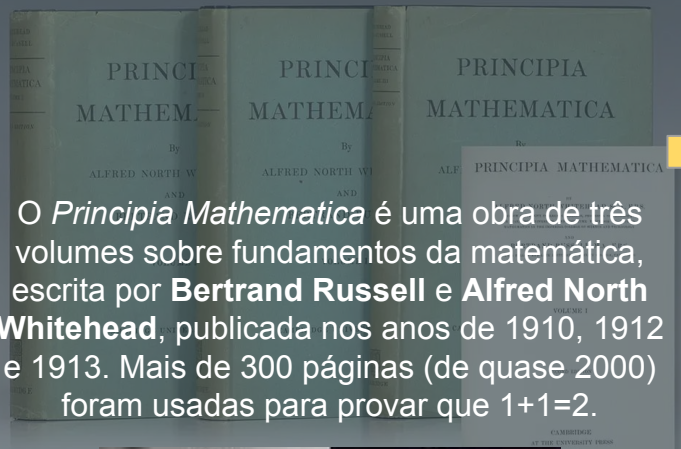
...

Aula 01:
Introdução

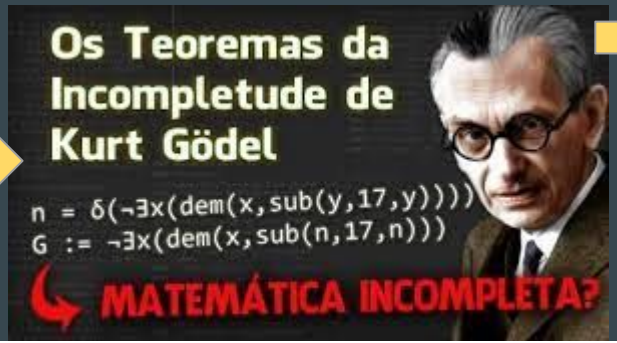
Prof. Igor Peretta

Breve histórico e fundamentos da computação

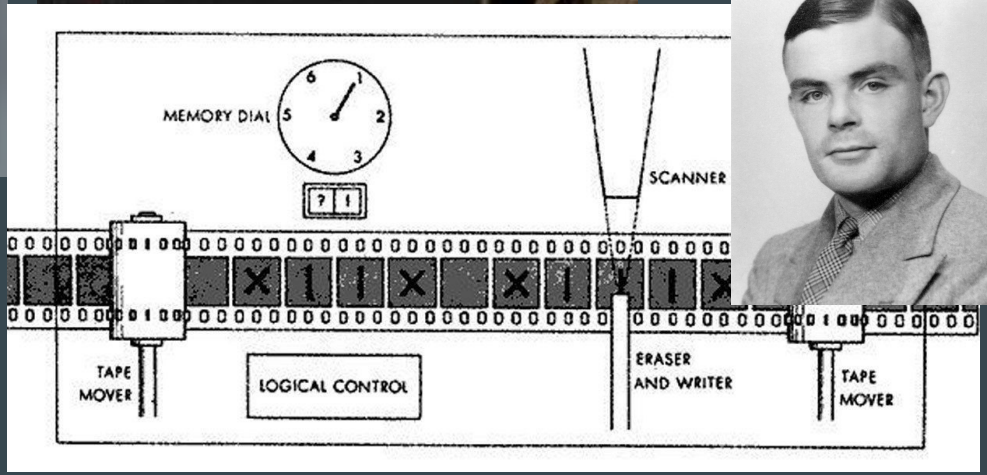
Fundamentos na Lógica Matemática



O *Principia Mathematica* é uma obra de três volumes sobre fundamentos da matemática, escrita por **Bertrand Russell** e **Alfred North Whitehead**, publicada nos anos de 1910, 1912 e 1913. Mais de 300 páginas (de quase 2000) foram usadas para provar que 1+1=2.

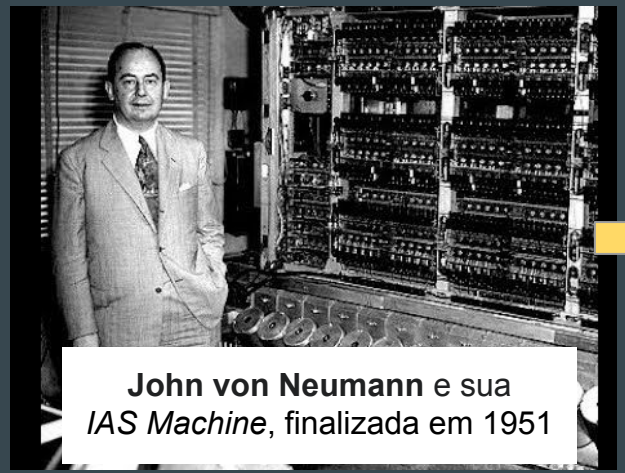


Em 1936-1937, **Alan Turing** apresentou a ideia da *Máquina de Turing*. "Tudo que é computável é computável por uma Máquina de Turing", *Tese de Church-Turing*.

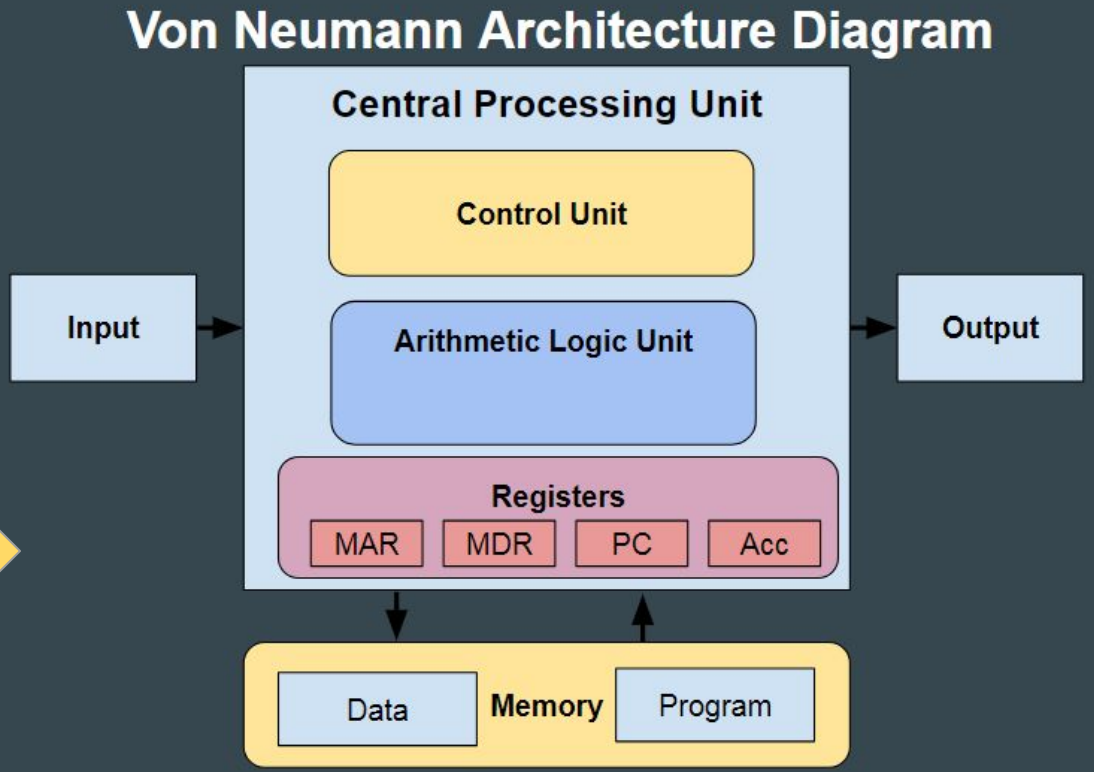


Da Teoria para a Prática

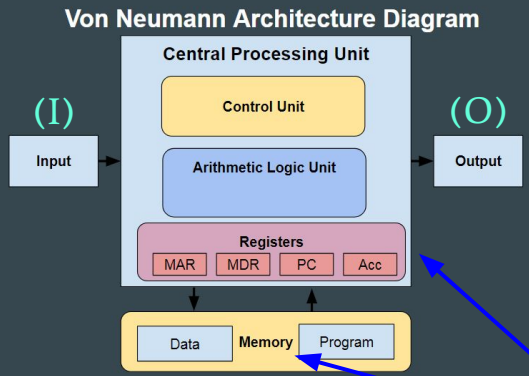
Gottfried W. Leibniz
Explication de l'Arithmétique Binaire, 1703

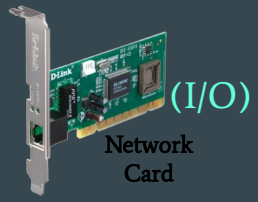


John von Neumann e sua *IAS Machine*, finalizada em 1951



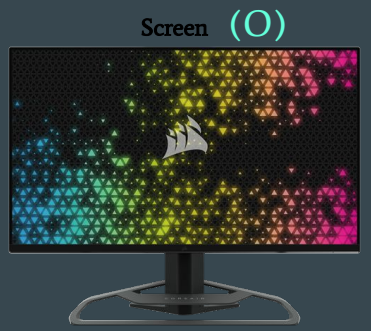
Partes de Hardware (Atual)




(I/O)
Network Card

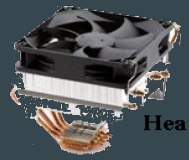

Scanner (I)

Printer (O)


Screen (O)


System Fan


Floppy
(I/O)


Heat Sink


Power Supply

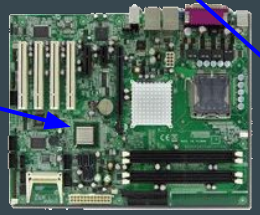

(I ou I/O)


Hard Drive
(I/O)


Keyboard
(I)


Optical Drive


Mouse
(I)


Motherboard

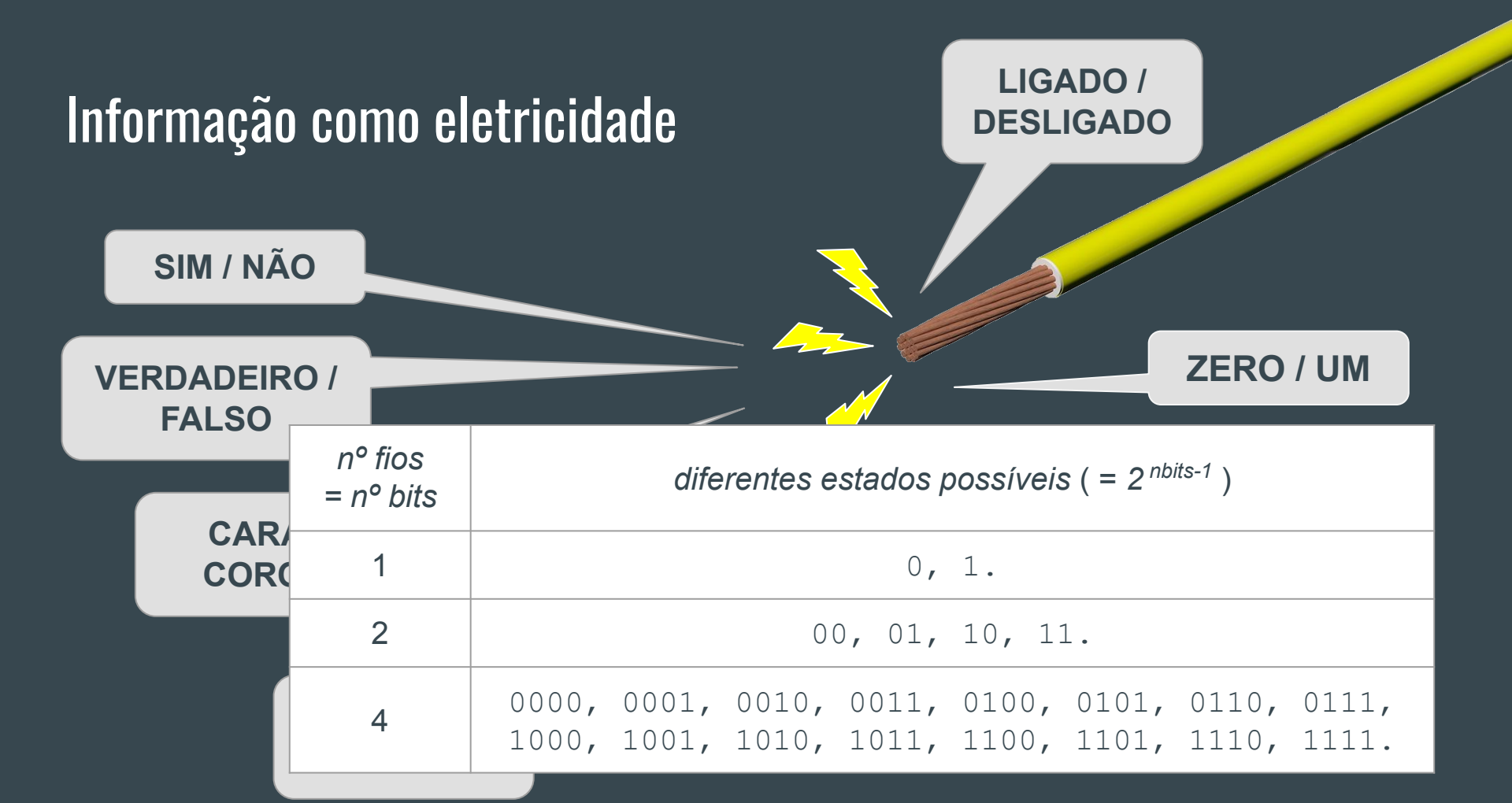

Processors (CPU)


RAM Moduels


(I/O)
Pendrive

Solid State Drive
(I/O)

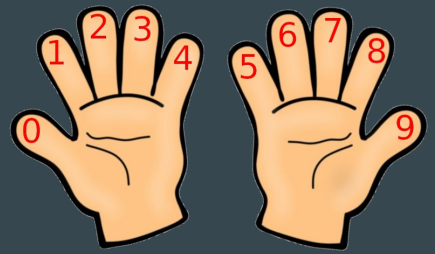
Informação como eletricidade



nº fios = nº bits	*diferentes estados possíveis (= $2^{nbits-1}$)*
1	0, 1.
2	00, 01, 10, 11.
4	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

Sistema binário

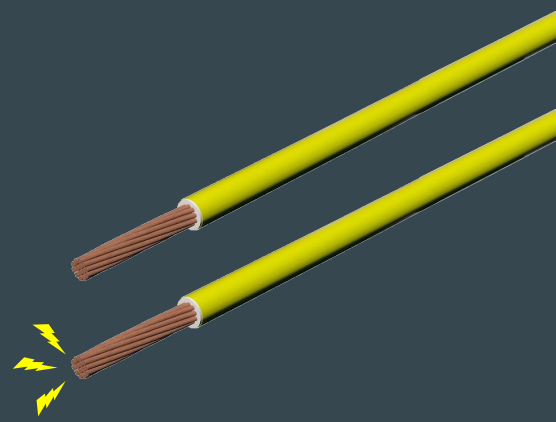
Sistema decimal



Sistema binário

0

1



1	9	7	4
10^3	10^2	10^1	10^0

$9 \times 10^2 = 900$

1	0	0	1
2^3	2^2	2^1	2^0

$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$

Representação binária

Qualquer número pode ser representado com 0's e 1's
ou por um conjunto de fios onde cada qual está ou não energizado.

- Com **8 fios** (ou bits), você pode manipular números de 0 a 255
- Com **32 fios** (ou bits), de 0 a 4.294.967.295, ou seja, > 4 bilhões

8 bits = 1 Byte

Podemos representar qualquer número quisermos, mas e outros tipos de informação?

Acontece que textos, sons, imagens e vídeos também podem ser representados com números!

c/ 3 "fios"	
0	**000**
1	**001**
2	**010**
3	**011**
4	**100**
5	**101**
6	**110**
7	**111**

Texto em binário

= 1

= 2

= 3

= 4

...

= 26

18 1 20 15

00010010 00000001 00010100 00001111

Todos os textos que você lê em um computador – por exemplo, as mensagens que você recebe dos amigos no seu celular –, são representados por um **sistema numérico** similar a esse.

Sons em binário

01100000

01010000

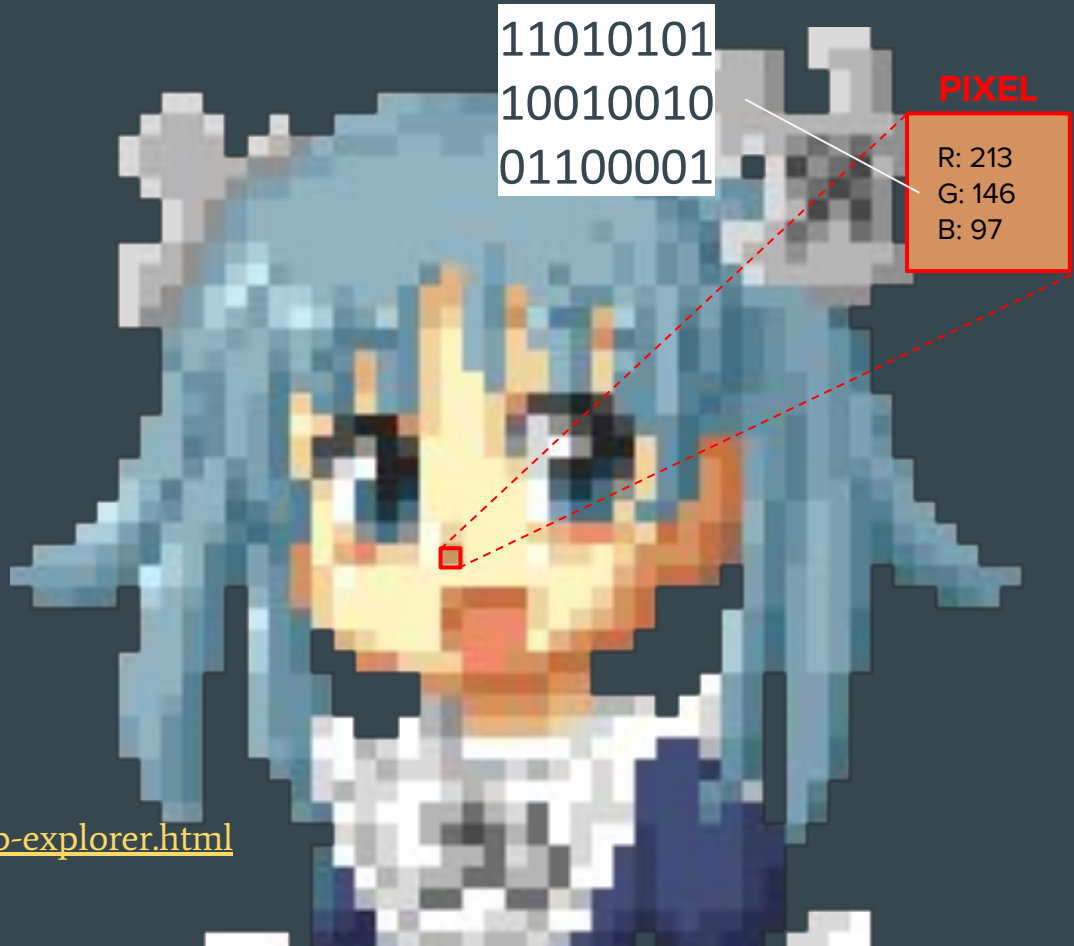
96

80

13

00001101

Imagem em binário



Sugestão para gerar cores com RGB:
http://web.stanford.edu/class/cs101/image-rgb-explorer.html

Vídeo em binário

Vídeos são coleções de imagens que passam a mais de 12 quadros por segundo.

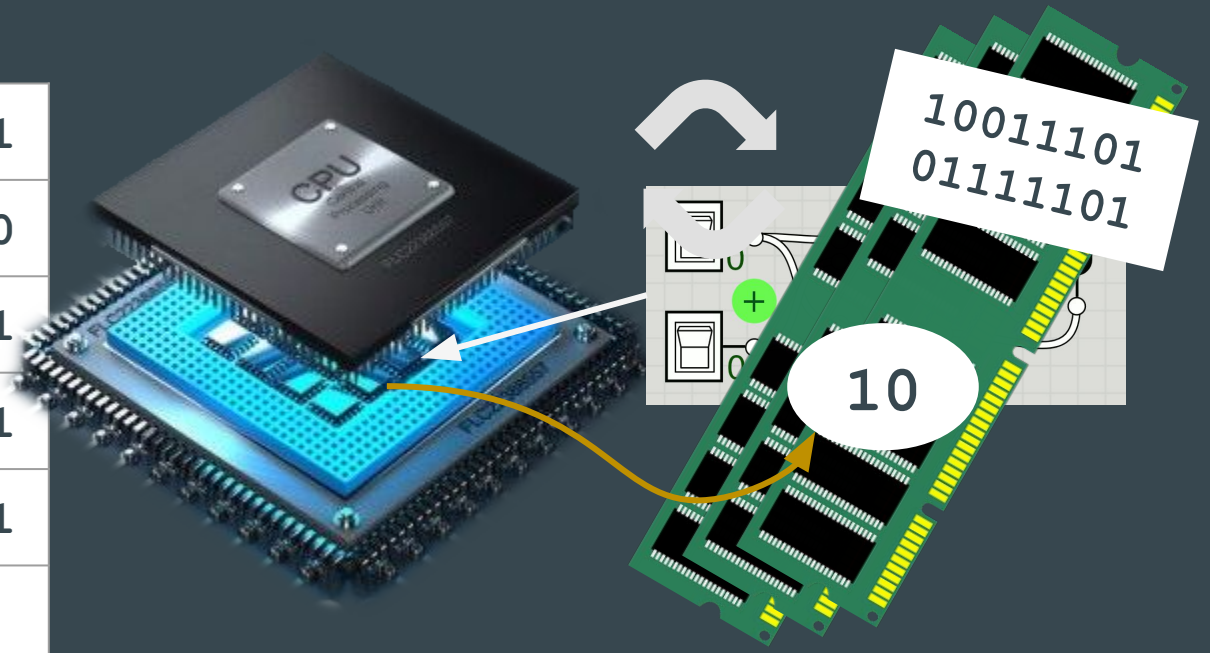
Tamanho em pixels:

- Imagens típicas (fotos, fundo de telas) são constituídas por milhões de pixels.
- Vídeos típicos com 30 quadros por segundo conseguem ultrapassar 3 bilhões de pixels por minuto.

Instruções para a CPU/Memória

ADD	10011101
SUBTRACT	11100100
MU	01[illegible]01
MOVE	11000111
STORE	01111101
. . .	. . .

STORE



Sequência de instruções ≡ CÓDIGO BINÁRIO

Abstraindo a programação

Cartão perfurado

Binário

```
00011000
10011000
01011100
01110000
01001101
11000010
10111011
10101001
11101000
01101110
10110100
10000000
11000001
10100101
00010110
11100100
11101100
00001000
00000100
01111101
```

Binário

Assembly

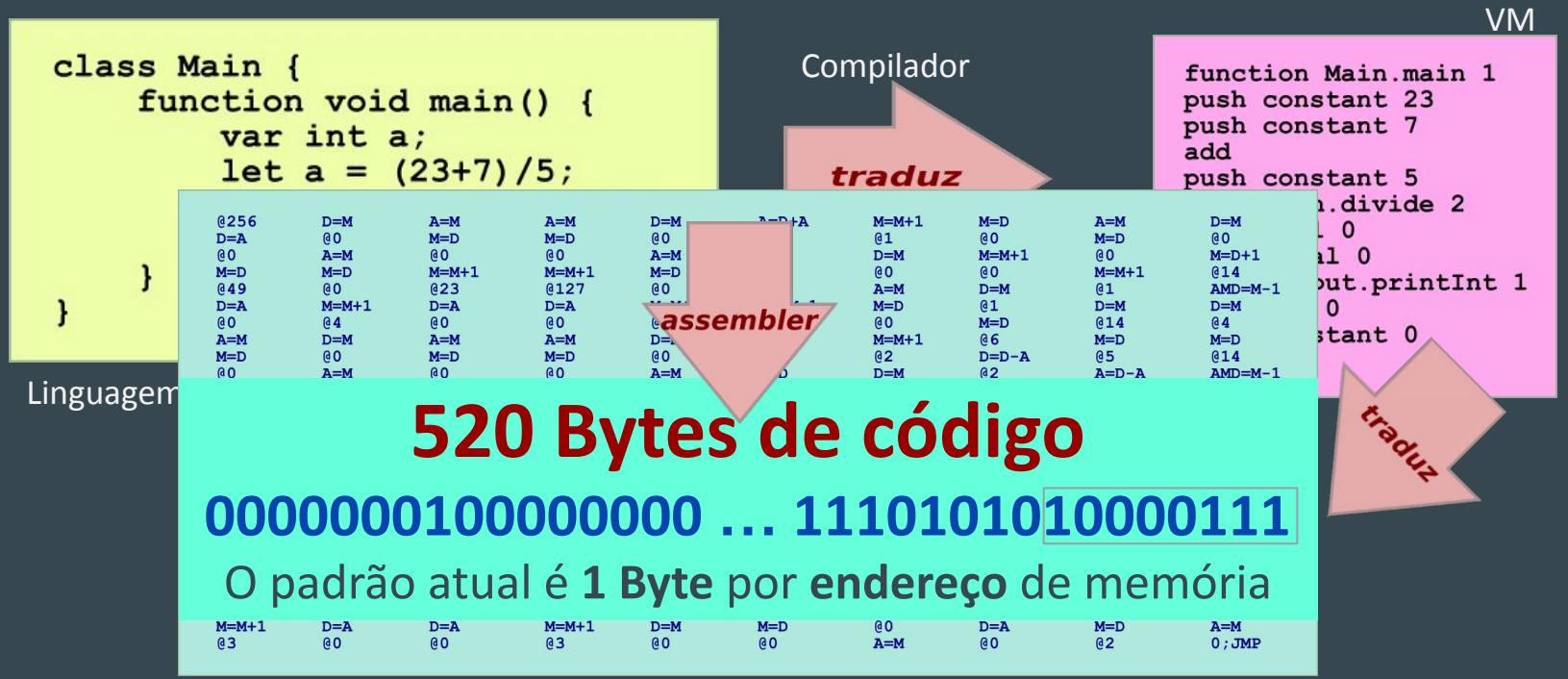
```
mov ah,0
int 1ah
mov ax,dx
mov cl,4
shl dx,cl
mov bx,ax
mov cl,12
shr bx,cl
add dx,bx
mov cl,4
shl ax,cl
div bx
jnz p020
call p100
push ax
dec ax
cmp ax,0
jnz p018
mov ax,0ffffh
mov atime,ax
```

(curiosidade) Referência em armazenamento

Cartão perfurado

Cartão SD

Compiladores, Interpretadores, Máquinas Virtuais



Linguagem JACK para o computador HACK
Adaptado de: https://www.nand2tetris.org/course

Linguagens de programação (alto nível)

python

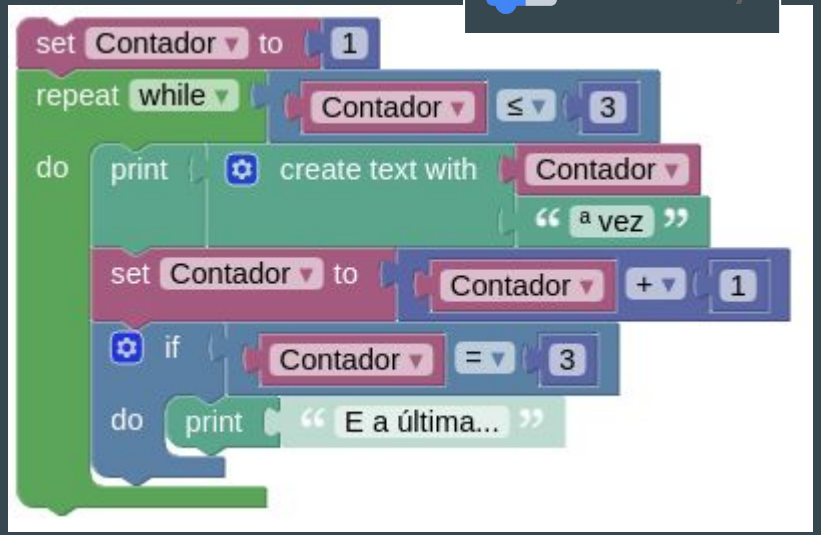
```python
Contador = 1
while Contador <= 3:
    print(f'{Contador} ª vez')
    Contador += 1
    if Contador == 3:
        print('E a última...')
```

JS JavaScript

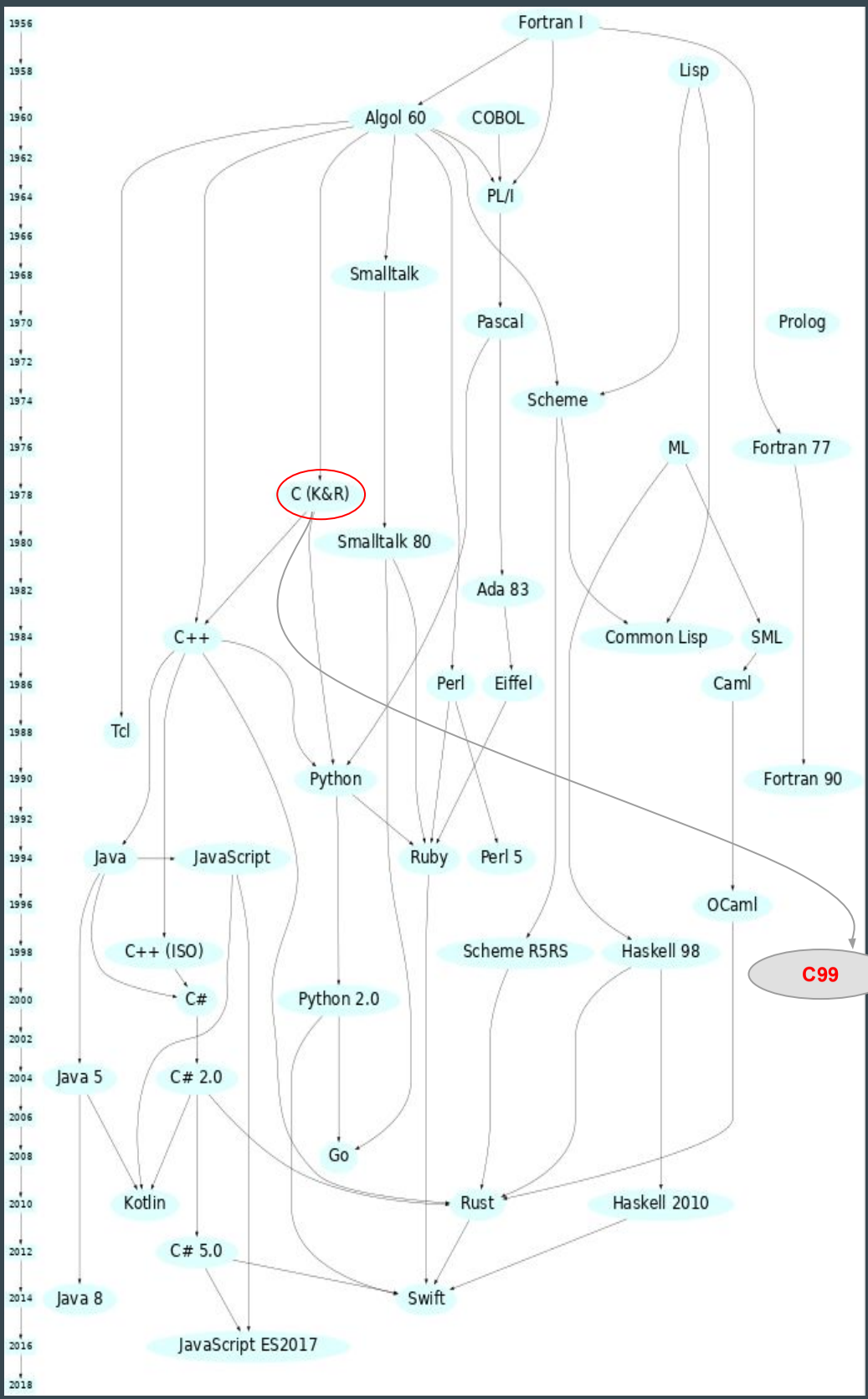
```javascript
var Contador;

Contador = 1;
while (Contador <= 3) {
    window.alert(String(Contador) + 'ª vez');
    Contador = Contador + 1;
    if (Contador == 3) {
        window.alert('E a última...');
    }
}
```

Blockly

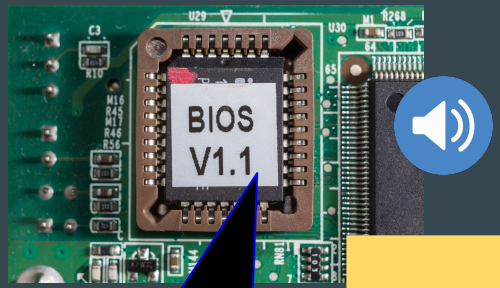


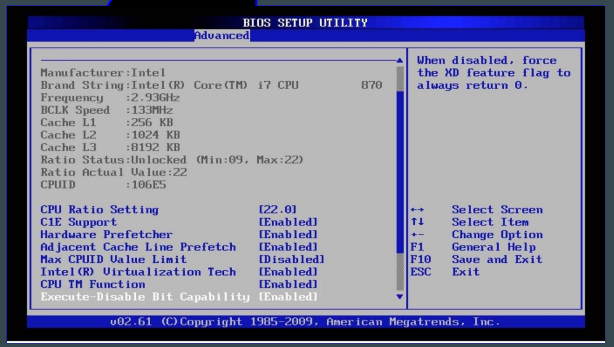
Genealogia das Principais Linguagens de Programação



Camadas de Software

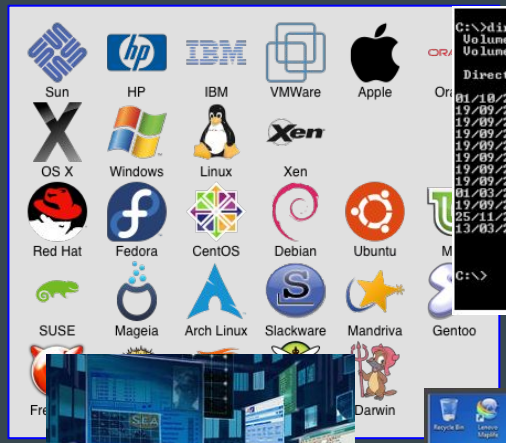
Camadas de Software, inicialização



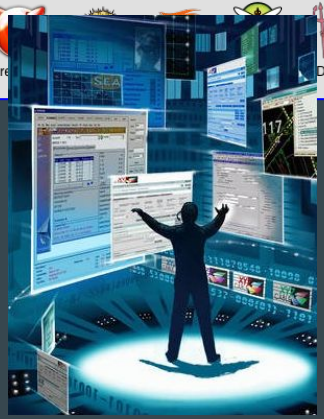


Camadas de Software, utilização e desenvolvimento

Aplicações Diversas



Interface Textual



Interface Natural

Interface Gráfica

Ambientes de Desenvolvimento Integrado (IDE)

IDEs não são Compiladores!

≠

Paradigmas de Programação

Classificação das linguagens de programação (I)

Programação imperativa: descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Substitui as funções do código de máquina por mnemônicas, endereços de memória absolutos referenciados por identificadores. (e.g. *Assembly*, também conhecida por linguagem de montagem)

Programação estruturada, procedural ou procedimental (inclui imperativa): descreve, passo a passo, o procedimento a ser seguido para resolver certo problema. A eficácia e a eficiência de cada solução é subjetiva e altamente dependente da experiência, habilidade e criatividade do programador. Faz uso de estruturas de ***sequência***, de ***decisão*** e de ***iteração***. (e.g. *C*)

Programação orientada a objeto: os dados e as rotinas para manipulá-los são mantidos numa unidade chamada objeto. O utilizador só pode acessar os dados através das sub-rotinas disponíveis, chamadas métodos, o que permite alterar o funcionamento interno do objeto sem afetar o código que o consome. Principais conceitos: abstração digital, encapsulamento, herança e polimorfismo. (e.g. *Java*)

Classificação das linguagens de programação (II)

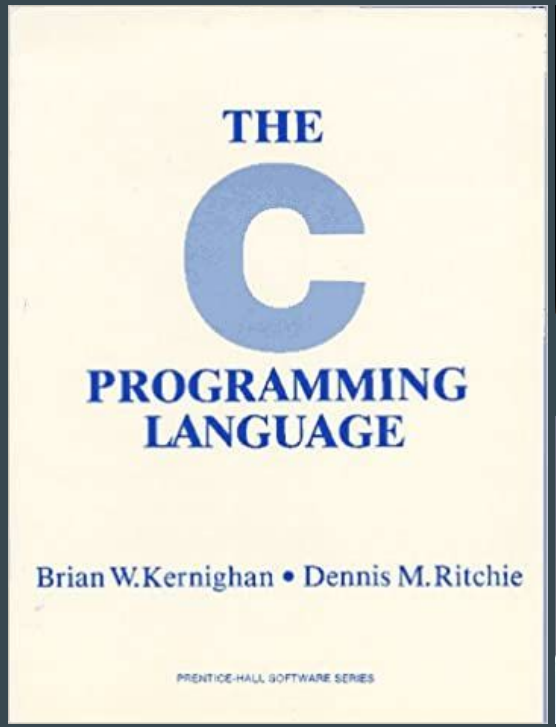
Programação declarativa: o problema é descrito ao computador, não o modo de resolução. O programa é estruturado como uma coleção de propriedades para encontrar o resultado esperado, e não um procedimento a se seguir. (e.g. *SQL*)

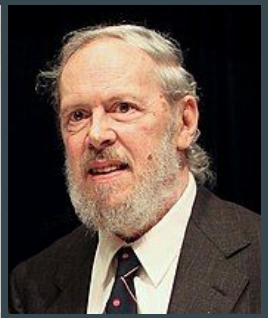
Programação funcional: usam funções, blocos de código construídos para agir como funções matemáticas. Desencoraja-se a mudança do valor das variáveis através de atribuição, fazendo grande uso de recursividade para isso. (e.g. *Haskell*)

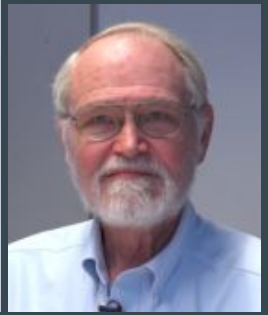
Programação lógica: fatos sobre o domínio do problema são expressados como fórmulas lógicas, e os programas são executados ao se aplicar regras de inferência nas fórmulas até que uma resposta é encontrada, ou a coleção de fórmulas é provada inconsistente. (e.g. *Prolog*)

Razões para aprender C

Breve histórico de C







Dennis Ritchie, trabalhando a partir da linguagem B de Ken Thompson, projeta e finaliza o primeiro compilador C em 1971.

Em 1978, junto com **Brian Kernighan**, publicou o livro que é considerado o lançamento da linguagem C de programação.

A partir de 1983, um esforço de padronização da linguagem começa pelo *American National Standards Institute*, a era do **ANSI C**, tendo como representante o padrão C89.

Em 1990, a ISO/IEC ratificou a norma que ficou conhecida como C90.

Em 1999, o lançamento da **ISO/IEC 9899:1999** define a versão **C99** mais madura e suportada pela maioria absoluta de compiladores em uso na atualidade.

Outras versões foram lançadas, como C11, C18 e C2x (em preparação), mas ainda falta alcançar a maioria dos compiladores em uso.

Sobre C

- C é uma **linguagem de programação procedural de propósito geral**, tem os recursos de programação de alto nível fornecidos pela maioria das linguagens de programação procedurais – variáveis fortemente tipadas, constantes, tipos de dados padrão, tipos enumerados, um mecanismo para definir seus próprios tipos, estruturas agregadas, estruturas de controle, recursão e modularização de programa. Apenas recentemente adicionou um tipo de dados booleano.
- C **não suporta** conjuntos de dados, classe ou objetos, funções aninhadas, nem índices por faixa ou seu uso como subscritos de matriz. C também não trabalha com coletor de lixo, **tampouco é considerada uma linguagem segura**.
- C tem, no entanto, **compilação separada**, **compilação condicional**, **operadores bit a bit**, **aritmética de ponteiros**, e **entrada e saída independentes de linguagem**. C possui um excelente grau de controle sobre **representações de dados**, de **memória** e de **uso da CPU**, com desempenho muito próximo ao de código de máquina (estimado em 1% a 2% de perda).
- C é conhecido como uma excelente linguagem de programação para sistemas operacionais. É também a linguagem de programação de escolha para nível de **sistemas**, **engenharias** e **programação científica**.
- Ainda hoje, C continua sendo a caixa de ferramentas do programador por excelência, **capaz de resolver praticamente qualquer tipo de problema em praticamente qualquer tipo de hardware**.

Benefícios de aprender a programar em C (I)

- **Ajuda a entender como um computador funciona**
  Você será capaz de entender e visualizar o funcionamento interno dos sistemas de computador, sua arquitetura e os conceitos gerais que orientam a programação, incluindo algoritmos, permitindo desenvolver programas mais complexos e abrangentes.

- **A maioria das linguagens de programação pode interagir com C**
  Linguagens como JavaScript, Python e Java, por exemplo, podem interagir com programas baseados em C. Além disso, C é útil ao tentar comunicar ideias e conceitos em programação devido à sua natureza universal.

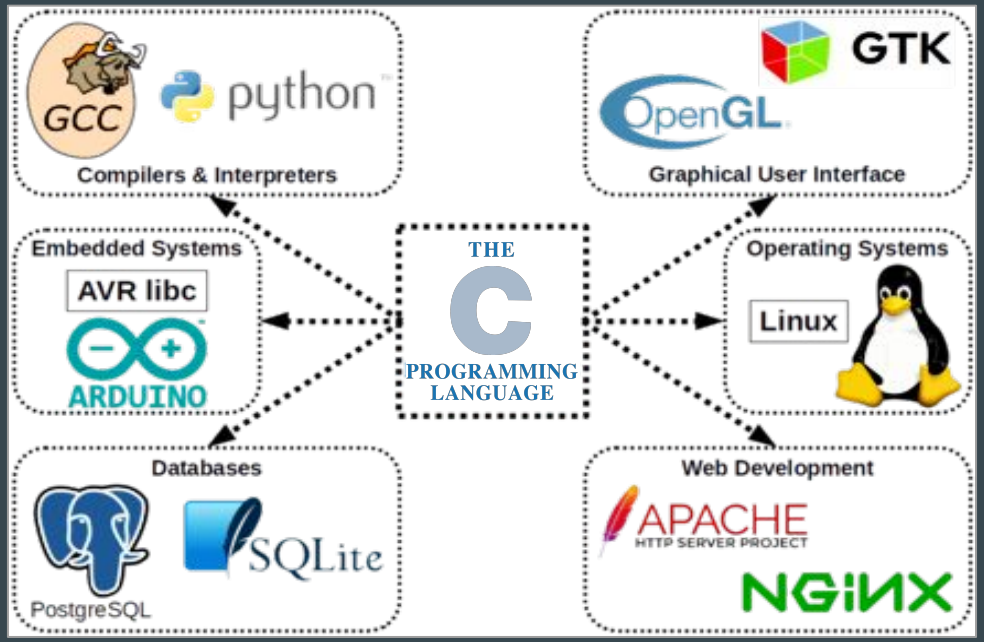
- **Permite trabalhar em projetos de código aberto**
  Muitas outras linguagens de programação se inspiram em C. Portanto, você também pode trabalhar em grandes projetos de código aberto e beneficiar milhões de programadores em todo o mundo.

Benefícios de aprender a programar em C (II)

- **Será mais fácil aprender outras linguagens de programação**
  Pelo fato de que muitas linguagens de programação são baseadas ou relacionadas a C, fica mais fácil o processo de aprendizado de outras linguagens.

- **A estrutura simplifica o teste e a depuração**
  A linguagem de programação C é uma linguagem fortemente tipada permitindo ao compilador impor o uso adequado desses tipos declarados e capturar uma variedade de bugs em tempo de compilação. Além disso, suporta o conceito de estruturas definidas pelo usuário usadas para encapsular dados e código de forma modular, melhorando a eficiência dos testes e facilitando a avaliação de defeitos.

- **É uma linguagem de programação eficiente**
  Existem apenas 32 palavras-chave contidas em C, bem como funções internas e tipos de dados. Também possui bibliotecas padrão que permitem o acesso a todos esses recursos e funções em qualquer ponto do programa. Além disso, C é uma linguagem extremamente eficiente cujos binários compilados são executados rapidamente e com sobrecarga mínima.



Escritos em C: os sistemas operacionais mais populares do mundo - Linux, Windows e Mac OS-X, suas interfaces e sistemas de arquivos; a infra-estrutura da Internet, incluindo a maioria de seus protocolos de rede, servidores web e sistemas de e-mail; bibliotecas de software que fornecem interfaces gráficas e ferramentas e algoritmos numéricos, estatísticos, de criptografia e de compressão eficientes são escritos em C; e o software para a maioria dos dispositivos embarcados, incluindo carros, aeronaves, robôs, aparelhos inteligentes, sensores, telefones celulares e consoles de jogos; etc.

GNU Compiler Collection

O **GNU Compiler Collection** (GCC) é um **compilador com otimização de instruções** produzido pelo Projeto GNU que suporta **várias linguagens de programação, arquiteturas de hardware e sistemas operacionais**.

Quando foi lançado em 1987, o GCC 1.0 foi chamado de **GNU C Compiler**, já que lidava apenas com a linguagem de programação C. Foi estendido para compilar C++, Objective-C, Objective-C++, Fortran, Ada, D e Go, entre outras linguagens de programação, incluindo as especificações OpenMP e OpenACC suportadas nos compiladores C e C++.

O GCC **foi portado para mais plataformas e arquiteturas de conjuntos de instruções do que qualquer outro compilador** e é amplamente implantado como uma ferramenta no desenvolvimento de software livre e proprietário. O GCC também está disponível para muitos sistemas embarcados, incluindo chips baseados em ARM e baseados em Power ISA.

GCC - Arquiteturas

-march=CPU[,+EXTENSION...] : *generate code for CPU and EXTENSION, CPU is one of:*

generic32, generic64, i386, i486, i586, i686, pentium, pentiumpro, pentiumii, pentiumiii, pentium4, prescott, nocona, core, core2, corei7, l1om, k1om, iamcu, k6, k6_2, athlon, opteron, k8, amdfam10, bdver1, bdver2, bdver3, bdver4, znver1, znver2, znver3, btver1, btver2

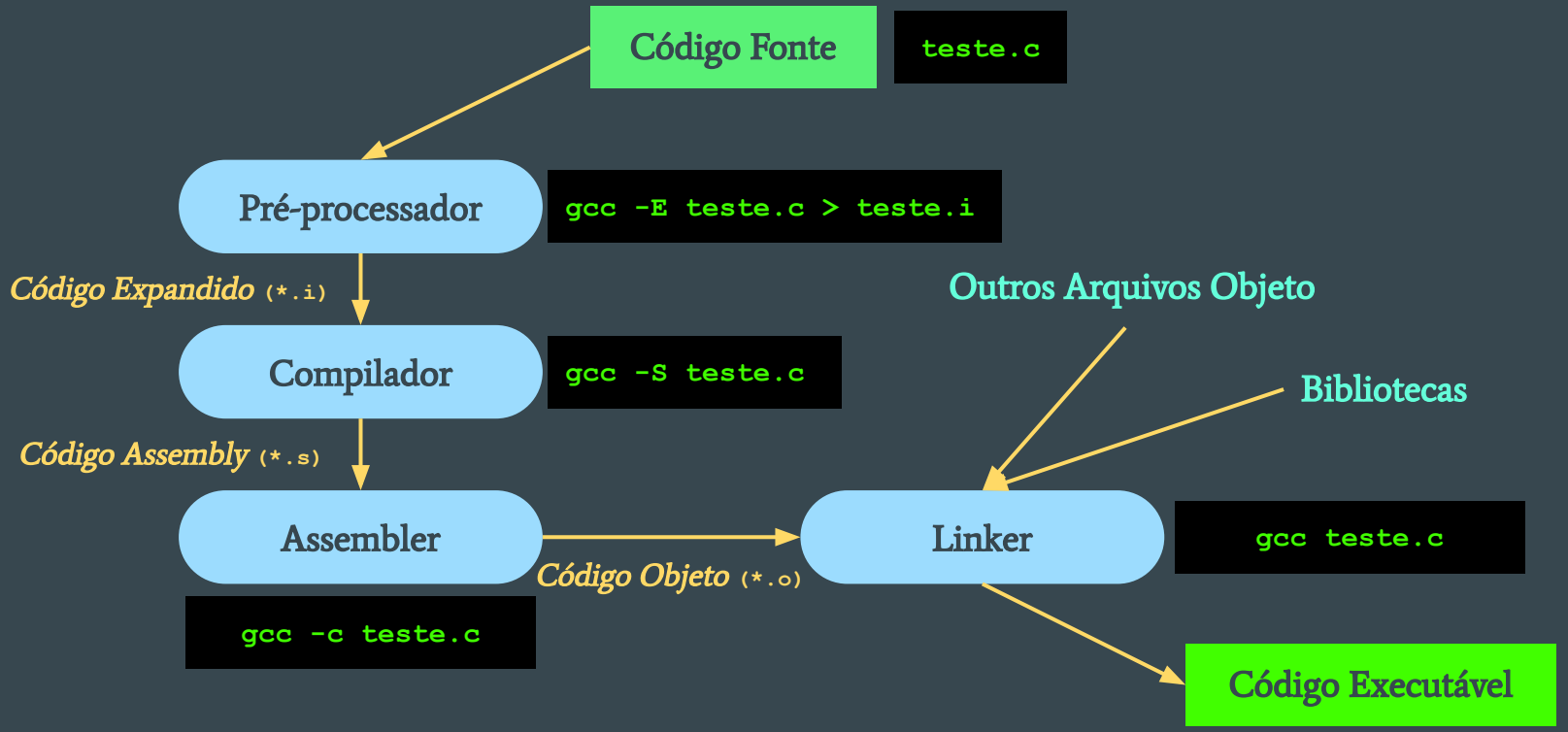
EXTENSION is combination of:

8087, 287, 387, 687, cmov, fxsr, mmx, sse, sse2, sse3, sse4a, ssse3, sse4.1, sse4.2, sse4, avx, avx2, avx512f, avx512cd, avx512er, avx512pf, avx512dq, avx512bw, avx512vl, vmx, vmfunc, smx, xsave, xsaveopt, xsavec, xsaves, aes, pclmul, fsgsbase, rdrnd, f16c, bmi2, fma, fma4, xop, lwp, movbe, cx16, ept, lzcnt, popcnt, hle, rtm, invpcid, clflush, nop, syscall, rdtscp, 3dnow, 3dnowa, padlock, svme, sse4a, abm, bmi, tbm, adx, rdseed, prfchw, smap, mpx, sha, clflushopt, prefetchwt1, se1, clwb, avx512ifma, avx512vbmi, avx512_4fmaps, avx512_4vnniw, avx512_vpopcntdq, avx512_vbmi2, avx512_vnni, avx512_bitalg, avx_vnni, clzero, mwaitx, ospke, rdpid, ptwrite, ibt, shstk, gfni, vaes, vpclmulqdq, wbnoinvd, pconfig, waitpkg, cldemote, amx_int8, amx_bf16, amx_tile, movdiri, movdir64b, avx512_bf16, avx512_vp2intersect, tdx, enqcmd, serialize, rdpru, mcommit, sev_es, tsxldtrk, kl, widekl, uintr, hreset, no87, no287, no387, no687, nocmov, nofxsr, nommx, nosse, nosse2, nosse3, nosse4a, nossse3, nosse4.1, nosse4.2, nosse4, noavx, noavx2, noavx512f, noavx512cd, noavx512er, noavx512pf, noavx512dq, noavx512bw, noavx512vl, noavx512ifma, noavx512vbmi, noavx512_4fmaps, noavx512_4vnniw, noavx512_vpopcntdq, noavx512_vbmi2, noavx512_vnni, noavx512_bitalg, noavx_vnni, noibt, noshstk, noamx_int8, noamx_bf16, noamx_tile, nomovdiri, nomovdir64b, noavx512_bf16, noavx512_vp2intersect, notdx, noenqcmd, noserialize, notsxldtrk, nokl, nowidekl, nouintr, nohreset

-mtune=CPU : *optimize for CPU, CPU is one of:*

generic32, generic64, i8086, i186, i286, i386, i486, i586, i686, pentium, pentiumpro, pentiumii, pentiumiii, pentium4, prescott, nocona, core, core2, corei7, l1om, k1om, iamcu, k6, k6_2, athlon, opteron, k8, amdfam10, bdver1, bdver2, bdver3, bdver4, znver1, znver2, znver3, btver1, btver2

Etapas de Compilação

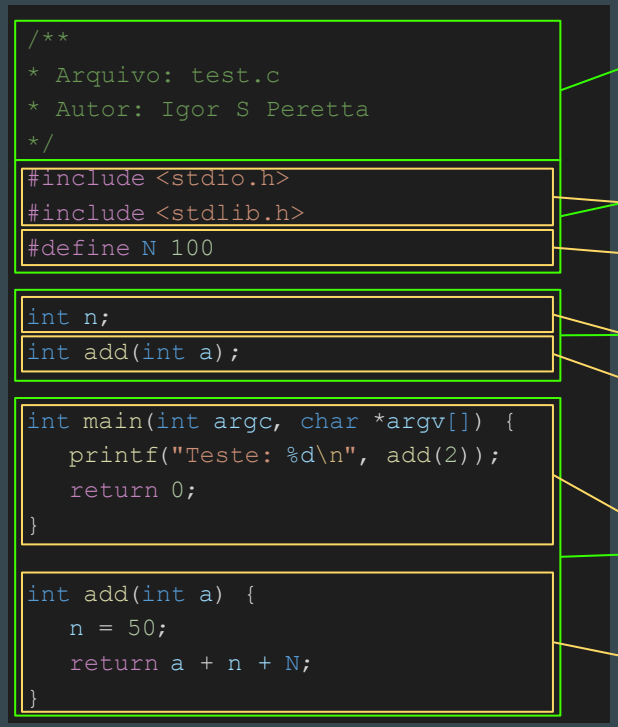


Programando em C

Estrutura Básica de um Programa em C

- **Documentação:**
  - Consiste na descrição do programa, nome do programador e data de criação. Estes são geralmente escritos na forma de comentários.
- **Diretivas de pré-processamento:**
  - **Link**: Todos os arquivos de cabeçalho estão incluídos nesta seção que contém diferentes funções das bibliotecas. Uma cópia desses arquivos de cabeçalho é inserida em seu código antes da compilação.
  - **Definições**: Inclui diretiva de pré-processador, que contém constantes simbólicas.
- **Declarações:**
  - **Declarações Globais**: Inclui declaração de variáveis globais, variáveis globais estáticas e declarações (assinaturas) de função.
- **Programa** (função 'main' é o *ponto de entrada*):
  - **Main**: Para cada programa C, a execução começa a partir da função main(). É obrigatório incluir uma função main() em cada programa C.
  - **Subprogramas**: Inclui todas as funções definidas pelo usuário (funções fornecidas pelo usuário). Eles podem conter as funções embutidas e as definições de função declaradas na seção Declaração Global. Eles são chamados na função main().

Arquivo texto com código-fonte (extensão **.c**)



```c
/**
* Arquivo: test.c
* Autor: Igor S Peretta
*/
#include <stdio.h>
#include <stdlib.h>
#define N 100

int n;
int add(int a);

int main(int argc, char *argv[]) {
   printf("Teste: %d\n", add(2));
   return 0;
}

int add(int a) {
   n = 50;
   return a + n + N;
}
```

Documentação

Diretivas de pré-processamento

Links (#include)

Definições (#define)

Declarações

Variáveis Globais

Protótipos de funções

Programa

Ponto de Entrada

Sub-Programas

Compilando e executando (terminal, powershell)

```
$ ls -la
-rw-r--r--. 1 igor igor   297 Aug 24 17:35 test.c
$ gcc test.c -std=c99 -pedantic-errors -o test.exe
$ ./test.exe
Teste: 150
```

executar

resultado

(opcional) flag `-o` para explicitar o nome do executável (se omitido, o padrão é `a.out`)

Flags do compilador para garantir o padrão **C99**

Nome do arquivo com código-fonte

Templates para C

```c
/**
* Arquivo:
* Autor:
* Matricula:
* Criado em:
*/

#include <stdio.h>

int main(void) {

    return 0;
}
```

```c
/**
* Arquivo:
* Autor:
* Matricula:
* Criado em:
*/

#include <stdio.h>

int main(int argc, char *argv[]) {

    return 0;
}
```

```c
/**
* Arquivo:
* Autor:
* Matricula:
* Criado em:
*/

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    return EXIT_SUCCESS;
}
```

Se falhar, use `EXIT_FAILURE`

Tamanhos de arquivos por etapa de compilação

```c
/**
* Arquivo: teste.c
* Autor: Igor Peretta
*/
#include <stdio.h>
int main(int argc, char *argv[]) {
  printf("Hello world!\n");
  return 0;
}
```

teste.c (texto) 145 Bytes
teste.i (texto) ~17 kBytes
teste.s (texto, assembly) 512 Bytes
teste.o (binário, objeto) 1,5 kBytes
teste (binário, executável) ~25 kBytes