

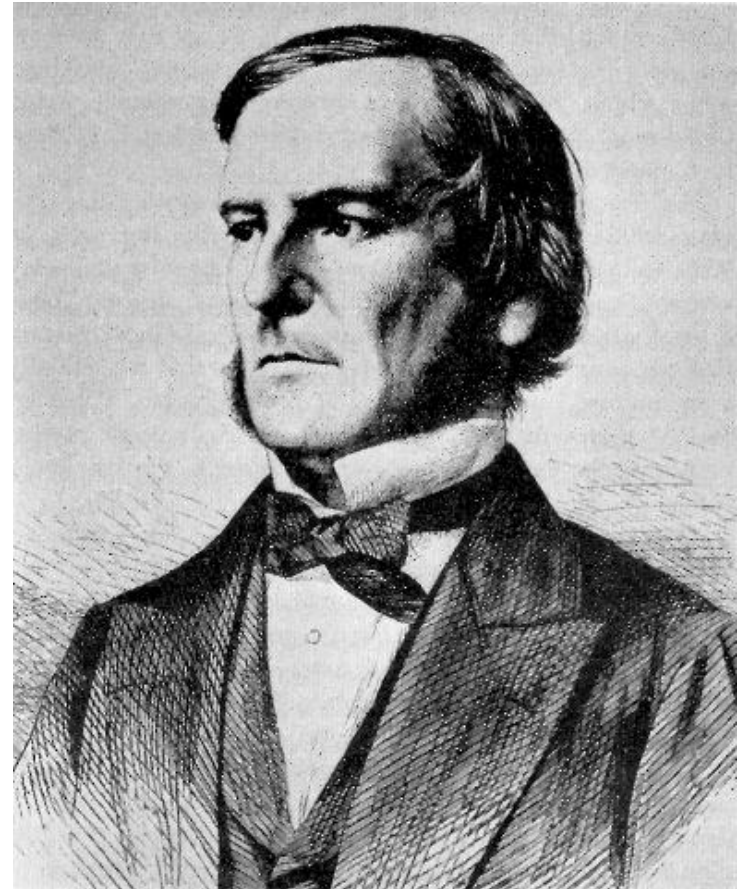
# **Algebra di Boole ed elementi di Logica**

## **Fine introduzione all'informatica**

Credits to A. Fuggetta, A. Campi e P. Pinoli

# Cenni all'algebra di Boole

- L'*algebra di Boole* (inventata da G. Boole, britannico, seconda metà '800), o *algebra della logica*, si basa su *operazioni logiche*
- Le operazioni logiche sono applicabili a *operandi logici*, cioè a operandi in grado di assumere solo i valori **vero** e **falso**
- Si può rappresentare vero con il bit **1** e falso con il bit **0** (convenzione di *logica positiva*)



Fonte: Wikipedia

# Operazioni logiche fondamentali

- Operatori logici binari (con 2 operandi logici)
  - Operatore **or**, o *somma logica*
  - Operatore **and**, o *prodotto logico*
- Operatore logico unario (con 1 operando)
  - Operatore **not**, o *negazione*, o *inversione*
- Poiché gli operandi logici ammettono due soli valori, si può definire compiutamente ogni operatore logico tramite una **tabella** di associazione operandi-risultato

# Operatori logici di base e loro tabelle di verità

| <u>A</u>       | <u>B</u> | <u>A or B</u> | <u>A</u>          | <u>B</u> | <u>A and B</u> |             |              |
|----------------|----------|---------------|-------------------|----------|----------------|-------------|--------------|
| 0              | 0        | 0             | 0                 | 0        | 0              |             |              |
| 0              | 1        | 1             | 0                 | 1        | 0              | <u>A</u>    | <u>not A</u> |
| 1              | 0        | 1             | 1                 | 0        | 0              | 0           | 1            |
| 1              | 1        | 1             | 1                 | 1        | 1              | 1           | 0            |
| (somma logica) |          |               | (prodotto logico) |          |                | (negazione) |              |

Le tabelle elencano tutte le possibili combinazioni in ingresso e il risultato associato a ciascuna combinazione

# Espressioni logiche (o booleane)

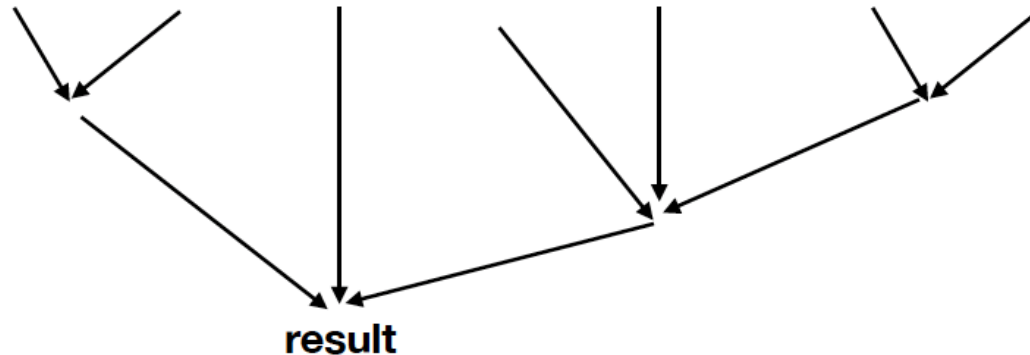
- Come le espressioni algebriche, costruite con:
  - Variabili logiche (letterali): p. es.  $A, B, C = 0$  oppure  $1$
  - Operatori logici: and, or, not
- Esempi:
  - $A \text{ or } (B \text{ and } C)$
  - $(A \text{ and } (\text{not } B)) \text{ or } (B \text{ and } C)$
- **Precedenza:** l'operatore "not" precede l'operatore "and", che a sua volta precede l'operatore "or"
  - $A \text{ and not } B \text{ or } B \text{ and } C = (A \text{ and } (\text{not } B)) \text{ or } (B \text{ and } C)$
  - (occhio alle parentesi!)
- Per ricordarlo, si pensi OR come "+" (più), AND come "x" (per) e NOT come "-" (cambia segno)

# Regole di precedenza

Operator precedence:

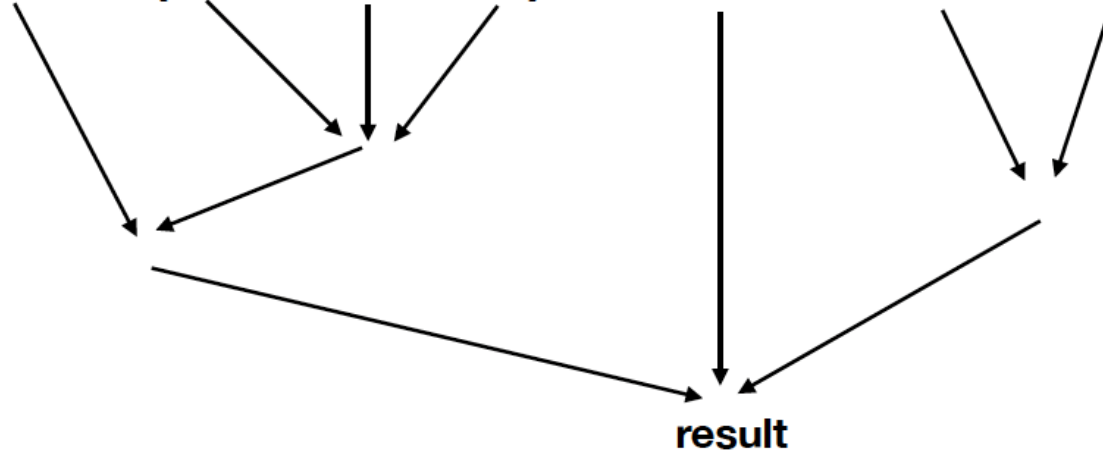
**NOT** precedes **AND** precedes **OR**

***NOT X OR Y AND NOT Z***



# Regole di precedenza: occhio alle parentesi!

**NOT (X OR Y) AND NOT Z**



# Tabelle di verità delle espressioni logiche

| <b>A</b> | <b>B</b> | <b><u>not ( ( A or B) and ( not A ) )</u></b> |
|----------|----------|---|
| 0        | 0        | 1   |
| 0        | 1        | 0   |
| 1        | 0        | 1   |
| 1        | 1        | 1   |

Specificano i valori di verità  
per tutti i possibili valori  
delle variabili



# Esempio di tabella di verità

## A and B or not C

| A B C | <b>X</b> = A and B | <b>Y</b> = not C | <b>X</b> or <b>Y</b> |            |
|-------|--------------------|------------------|----------------------|------------|
| 0 0 0 | 0 and 0 = 0        | not 0 = 1        | 0 or 1               | <b>= 1</b> |
| 0 0 1 | 0 and 0 = 0        | not 1 = 0        | 0 or 0               | <b>= 0</b> |
| 0 1 0 | 0 and 1 = 0        | not 0 = 1        | 0 or 1               | <b>= 1</b> |
| 0 1 1 | 0 and 1 = 0        | not 1 = 0        | 0 or 0               | <b>= 0</b> |
| 1 0 0 | 1 and 0 = 0        | not 0 = 1        | 0 or 1               | <b>= 1</b> |
| 1 0 1 | 1 and 0 = 0        | not 1 = 0        | 0 or 0               | <b>= 0</b> |
| 1 1 0 | 1 and 1 = 1        | not 0 = 1        | 1 or 1               | <b>= 1</b> |
| 1 1 1 | 1 and 1 = 1        | not 1 = 0        | 1 or 0               | <b>= 1</b> |

# Esercizio

- Risolvere le seguenti espressioni con tabella di verità:
  1.  $\text{not } A \text{ or } B \text{ and not } A$
  2.  $B \text{ or not } C \text{ and } A \text{ or not } C$

# A che cosa servono le espressioni logiche?

- A *modellare* alcune (non tutte) forme di *ragionamento*
  - $A =$  è vero che 1 è maggiore di 2 ? (sì o no, qui è no) = 0
  - $B =$  è vero che 2 più 2 fa 4 ? (sì o no, qui è sì) = 1
  - $A \text{ and } B =$  è vero che 1 sia maggiore di 2 e che 2 più 2 faccia 4 ?  
Si ha che  $A \text{ and } B = 0 \text{ and } 1 = 0$ , dunque no
  - $A \text{ or } B =$  è vero che 1 sia maggiore di 2 o che 2 più 2 faccia 4 ?  
Si ha che  $A \text{ or } B = 0 \text{ or } 1 = 1$ , dunque sì
- OR, AND e NOT vengono anche chiamati *connettivi logici*, perché funzionano come le congiunzioni coordinanti "o" ed "e", e come la negazione "non", del linguaggio naturale
- Si modellano ragionamenti (o *deduzioni*) basati solo sull'uso di "o", "e" e "non" (non è molto, ma è utile)

# Che cosa non si può modellare tramite espressioni logiche?

- Le espressioni logiche (booleane) *non modellano*:
  - Domande *esistenziali*: "**c'è almeno** un numero reale  $x$  tale che il suo quadrato valga  $-1$  ?" (si sa bene che *non c'è*)  
 $\exists x \mid x^2 = -1$  è falso
  - Domande *universali*: "**ogni** numero naturale è la somma di quattro quadrati di numeri naturali ?" (si è dimostrato *di sì*)  
 $\forall x \mid x = a^2 + b^2 + c^2 + d^2$  è vero ("teorema dei 4 quadrati")  
Più esattamente andrebbe scritto:  $\forall x \exists a, b, c, d \mid x = a^2 + b^2 + c^2 + d^2$
- $\exists$  e  $\forall$  sono chiamati "operatori di quantificazione", e sono ben diversi da *or*, *and* e *not*
- La parte della logica che tratta solo degli operatori *or*, *and* e *not* si chiama **calcolo proposizionale**
- Aggiungendo gli operatori di quantificazione, si ha il **calcolo dei predicati** (che è molto più complesso)

# Tautologie e Contraddizioni

- *Tautologia*

- Una espressione logica che è sempre **vera**, per qualunque combinazione di valori delle variabili
  - Esempio: principio del "terzo escluso": **A or not A** (*tertium non datur*, non si dà un terzo caso tra l'evento A e la sua negazione)

- *Contraddizione*

- Una espressione logica che è sempre **falsa**, per qualunque combinazione di valori delle variabili
  - Esempio: principio di "non contraddizione": **A and not A** (l'evento A e la sua negazione non possono essere entrambi veri)

# Equivalenza tra espressioni

- Due espressioni logiche si dicono **equivalenti** (e si indica con  $\Leftrightarrow$ ) **se hanno la medesima tabella di verità**. La verifica è *algoritmica*. Per esempio:

| A B | not A and not B | $\Leftrightarrow$ | not (A or B) |
|-----|-----------------|-------------------|--------------|
| 0 0 | 1 and 1 = 1     |                   | not 0 = 1    |
| 0 1 | 1 and 0 = 0     |                   | not 1 = 0    |
| 1 0 | 0 and 1 = 0     |                   | not 1 = 0    |
| 1 1 | 0 and 0 = 0     |                   | not 1 = 0    |

- Espressioni logiche equivalenti modellano gli stessi *stati di verità* a fronte delle medesime variabili

# Proprietà dell'algebra di Boole

- L'algebra di Boole gode di svariate *proprietà*, formulabili sotto specie di *identità*
  - (cioè formulabili come equivalenze tra espressioni logiche, valide per qualunque combinazione di valori delle variabili)
- Esempio celebre: le **Leggi di De Morgan**
  - $\text{not}(A \text{ **and** } B) = \text{not } A \text{ **or** } \text{not } B$  (1ª legge)
  - $\text{not}(A \text{ **or** } B) = \text{not } A \text{ **and** } \text{not } B$  (2ª legge)

# Altre proprietà

- Alcune proprietà somigliano a quelle dell'algebra numerica tradizionale:
  - Proprietà *associativa*:  $A \text{ or } (B \text{ or } C) = (A \text{ or } B) \text{ or } C$  (idem per AND)
  - Proprietà *commutativa*:  $A \text{ or } B = B \text{ or } A$  (idem per AND)
  - Proprietà *distributiva* di AND rispetto a OR:  
 $A \text{ and } (B \text{ or } C) = A \text{ and } B \text{ or } A \text{ and } C$
  - ... e altre ancora
- Ma parecchie altre sono alquanto insolite...
  - Proprietà *distributiva* di OR rispetto a AND:  
 $A \text{ or } B \text{ and } C = (A \text{ or } B) \text{ and } (A \text{ or } C)$
  - Proprietà di *assorbimento* (A assorbe B):  
 $A \text{ or } A \text{ and } B = A$
  - *Legge dell'elemento 1*:  $\text{not } A \text{ or } A = 1$
  - ... e altre ancora



# Uso delle proprietà

- *Trasformare* un'espressione logica in un'altra, differente per aspetto ma equivalente:

|   |   |
|---|---|
| $\text{not } A \text{ and } B \text{ or } A =$                                  | (assorbimento)                              |
| $= \text{not } A \text{ and } B \text{ or } (A \text{ or } A \text{ and } B) =$ | (togli le parentesi)                        |
| $= \text{not } A \text{ and } B \text{ or } A \text{ or } A \text{ and } B =$   | (commutativa)                               |
| $= \text{not } A \text{ and } B \text{ or } A \text{ and } B \text{ or } A =$   | (distributiva)                              |
| $= (\text{not } A \text{ or } A) \text{ and } B \text{ or } A =$                | (legge dell'elemento 1)                     |
| $= \textbf{true} \text{ and } B \text{ or } A =$                                | (vero and $B = B$ )                         |
| $= B \text{ or } A$   | è più semplice dell'espressione originale ! |

- Si *verifichi* l'equivalenza con le tabelle di verità!
- Occorre conoscere un'ampia lista di proprietà e si deve riuscire a "vederle" nell'espressione (qui è il difficile)

# Valutazione cortocircuitata

- La valutazione cortocircuitata delle espressioni è una tecnica utilizzata in programmazione elettronica e informatica per migliorare l'efficienza nell'elaborazione di espressioni booleane
- **Valutazione cortocircuitata dell'AND:** se il primo termine risulta falso, l'espressione risulta falsa senza che gli altri termini vengano controllati
- **Valutazione cortocircuitata dell'OR:** se il primo termine risulta vero, l'espressione è vera indipendentemente dagli altri valori

# Dalla tabella di verità alle espressioni

- Esistono principalmente due algoritmi:
  1. Somma di prodotti (**SOP**)
  2. Prodotto di somme (**POS**)
- Il risultato che producono è *equivalente*
- (Non li vedremo in questo corso)

# Quiz

Vai su **wooclap.com** e usa il codice **XGHTDZ**



Rispondete in coppia:



- Qual è il risultato dell'operazione 1 and 0:

- a) 0
- b) 1
- c) 10
- d) Non esiste

Risposta: ①



- Qual è il risultato dell'operazione 1 or 0:

- a) 0
- b) 1
- c) Non definito
- d) x

Risposta: ②

Clicca sullo schermo di proiezione per avviare la domanda

- La seguente proposizione è una tautologia o una contraddizione (risolvete tramite tabella di verità):

$E = A \text{ OR NOT}(A \text{ AND } B)$



# Tool online

- <https://www.emathhelp.net/en/calculators/discrete-mathematics/boolean-algebra-calculator/>
- <https://www.boolean-algebra.com>

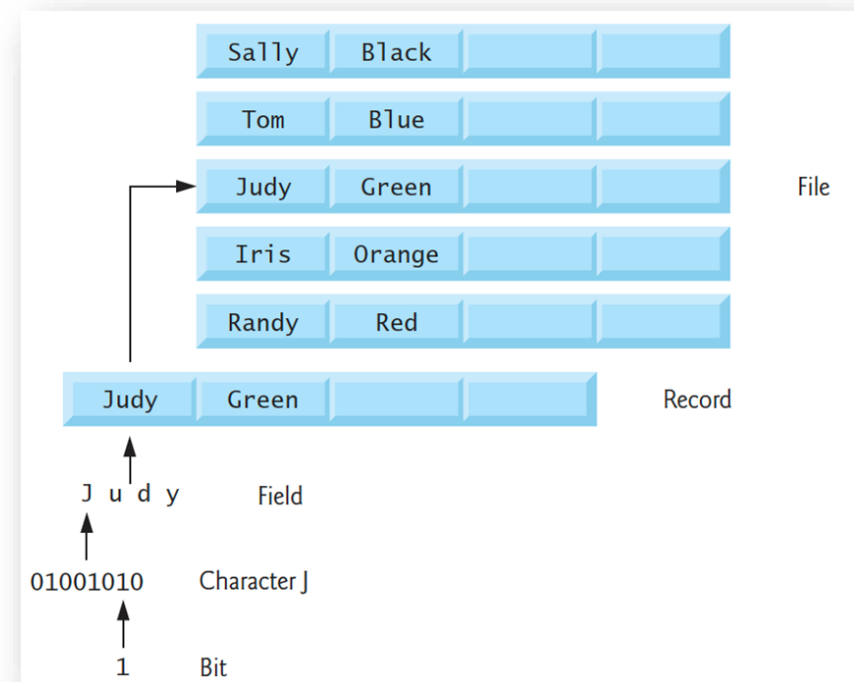
# Intervallo



Fonte: PlaygroundAI

# Gerarchia dei dati

- I dati elaborati dai computer formano una gerarchia che cresce in **grandezza** e **complessità**, partendo dai **bit** fino ad arrivare ai **caratteri** e i **campi**
- Ogni livello ha più significato e contesto, permettendo la rappresentazione di informazioni sempre più complesse
- Questa struttura gerarchica è essenziale per comprendere come i dati vengono organizzati e gestiti nei sistemi informatici



Fonte: Deitel & Deitel

# Caratteri

- Lavorare con i dati nella forma a basso livello dei bit è difficile
- Invece, le persone preferiscono lavorare con cifre decimali (0-9), lettere (A-Z e a-z) e simboli speciali come:

\$ @ % & \* ( ) - + " : ; , ? /

- Le cifre, le lettere e i simboli speciali sono noti come **caratteri**
- Il set di caratteri del computer contiene i caratteri usati per scrivere programmi e rappresentare elementi di dati
- C utilizza per impostazione predefinita il set di caratteri ASCII (American Standard Code for Information Interchange)
- C supporta anche i caratteri Unicode®, composti da uno, due, tre o quattro byte (rispettivamente 8, 16, 24 o 32 bit)



# Campi

- Così come i caratteri sono composti da bit, i campi sono composti da caratteri o byte
- Un campo è un gruppo di caratteri o byte che trasmette un significato
- Un campo composto da lettere maiuscole e minuscole potrebbe rappresentare il nome di una persona
- Un campo composto da cifre decimali potrebbe rappresentare l'età di una persona in anni

# Record

- Diversi campi correlati possono essere utilizzati per comporre un **record**
- Un record di un dipendente potrebbe essere composto da:
  - Numero di identificazione del dipendente (un numero intero)
  - Nome (un gruppo di caratteri)
  - Indirizzo (un gruppo di caratteri)
  - Tariffa oraria (un numero con un punto decimale)
  - Guadagni accumulati nell'anno (un numero con un punto decimale)
  - Importo delle tasse trattenute (un numero con un punto decimale)
- Un record è quindi un gruppo di campi correlati

# Files

- Un **file** è un gruppo di record correlati
- Un file contiene dati arbitrari in formati arbitrari
- Alcuni sistemi operativi considerano un file semplicemente come una sequenza di byte;
  - qualsiasi organizzazione dei byte in un file è una struttura creata dal programmatore del software



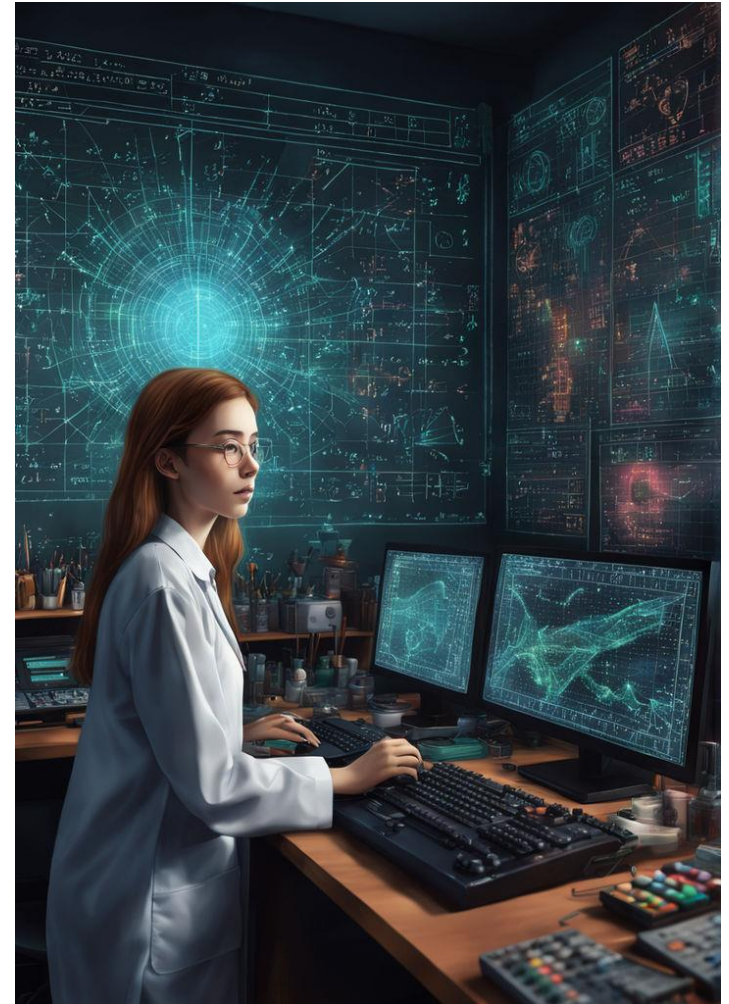
Fonte: PlaygroundAI

# Databases

- Un **database** è una raccolta di dati organizzata per un facile accesso e manipolazione
- Il modello più popolare è il database relazionale, in cui i dati sono memorizzati in tabelle semplici
- Una **tabella** include record e campi
- Una tabella di studenti potrebbe includere:
  - Nome, cognome, corso di laurea, anno, numero di matricola e media voti.
- I dati di ciascuno studente costituiscono un **record**, e le singole informazioni all'interno di ogni record sono i **campi**

# Big Data

- La quantità di dati prodotta a livello mondiale è **enorme**, e la sua crescita è in accelerazione
- Le applicazioni di Big Data gestiscono quantità massicce di dati
- Questo settore è in rapida **crescita**, creando molte opportunità per gli sviluppatori software
- Milioni di posti di lavoro nel campo della tecnologia dell'informazione (IT) a livello globale già supportano le **applicazioni di Big Data**



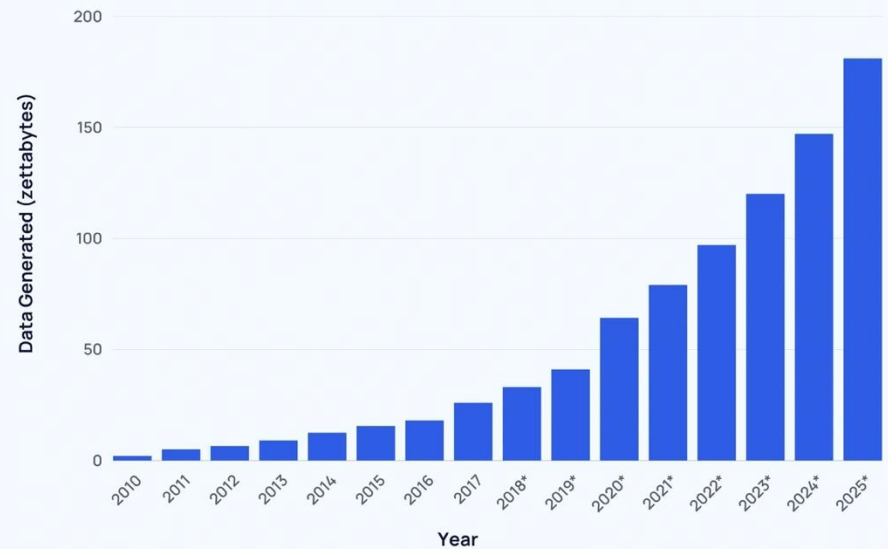
Fonte: PlaygroundAI

# Zettabyte era

| Category     | Proportion of Internet Data Traffic |
|--------------|-------------------------------------|
| Video        | 53.72%                              |
| Social       | 12.69%                              |
| Gaming       | 9.86%                               |
| Web browsing | 5.67%                               |
| Messaging    | 5.35%                               |
| Marketplace  | 4.54%                               |
| File sharing | 3.74%                               |
| Cloud        | 2.73%                               |

Fonte: Exploding Topics

## Global Data Generated Annually



- 1 Zettabyte =  $10^{21}$  bytes

# Linguaggi di programmazione

- I programmatori scrivono istruzioni in vari linguaggi di programmazione, alcuni **direttamente** comprensibili dai computer e altri che richiedono passaggi intermedi di **traduzione**
- Oggi sono in uso centinaia di tali linguaggi, che possono essere suddivisi in tre tipi generali:
  - Linguaggio macchina
  - Linguaggio assembly
  - Linguaggio di alto livello

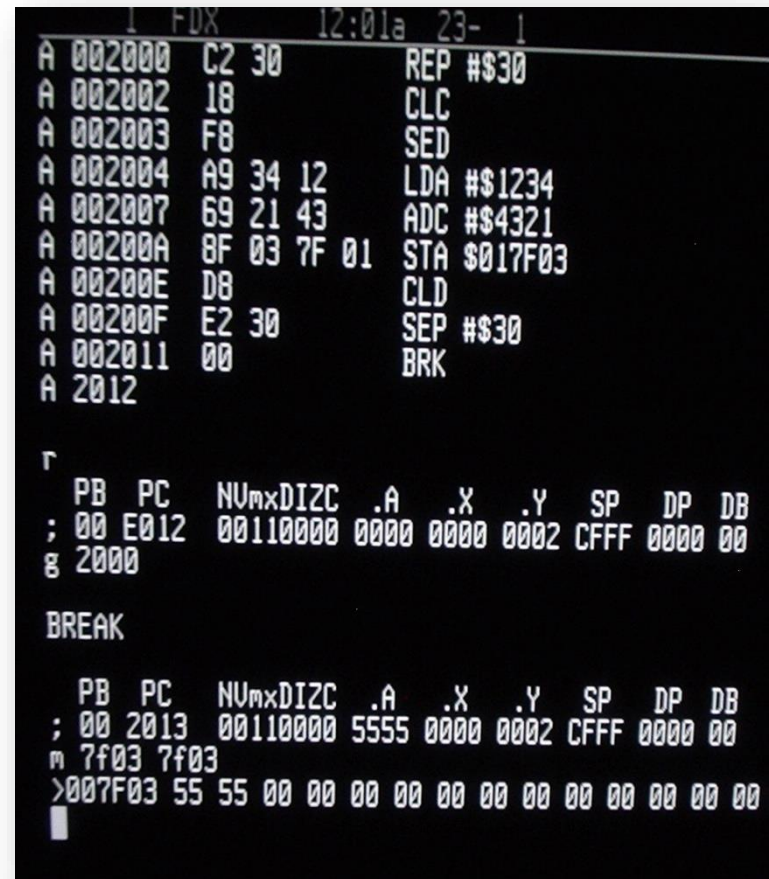


Fonte: PlaygroundAI



# Linguaggio macchina

- Ogni computer può comprendere **direttamente** il proprio linguaggio macchina, definito dal suo design hardware
- Il linguaggio macchina generalmente consiste di una **stringa di numeri** (0 e 1) che istruiscono il computer su come eseguire le operazioni elementari una alla volta
- I linguaggi macchina sono **machine-dependent** - ognuno può essere utilizzato su un solo tipo di macchina
- Molto difficili da comprendere per gli esseri umani!



The image shows a screenshot of a debugger window, likely for the Intel 8086 architecture. The top section displays assembly instructions with their addresses and operands. The bottom section shows the current state of registers and flags.

```
1 FDX 12:01a 23- 1
A 002000 C2 30 REP #$30
A 002002 18 CLC
A 002003 F8 SED
A 002004 A9 34 12 LDA #$1234
A 002007 69 21 43 ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8 CLD
A 00200F E2 30 SEP #$30
A 002011 00 BRK
A 2012

r
PB PC NUMxDIZC .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC NUMxDIZC .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00
```

Fonte: Wikipedia



# Linguaggio Assembly e Assembler

- Programmare in linguaggio macchina era troppo **lento** e tedioso per la maggior parte dei programmatori
- Invece di utilizzare stringhe di numeri comprensibili direttamente dai computer, i programmatori iniziarono a usare **abbreviazioni simili all'inglese** per rappresentare operazioni elementari
- Queste abbreviazioni formarono la base dei linguaggi **assembly**
- Furono sviluppati programmi traduttori chiamati **assembler** per convertire i programmi in linguaggio assembly in linguaggio macchina alla velocità del computer

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

C000                                ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS    #STACK

                                *****
                                * FUNCTION: INITA - Initialize ACIA
                                * INPUT: none
                                * OUTPUT: none
                                * CALLS: none
                                * DESTROYS: acc A

0013          RESETA EQU    %00010011
0011          CTLREG EQU    %00010001

C003 86 13          INITA  LDA A  #RESETA  RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11          LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON  GO TO START OF MONITOR

                                *****
                                * FUNCTION: INCH - Input character
                                * INPUT: none
                                * OUTPUT: char in acc A
                                * DESTROYS: acc A
                                * CALLS: none
                                * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH   LDA A  ACIA      GET STATUS
C013 47          ASR A                  SHIFT RDRF FLAG INTO CARRY
C014 24 FA          BCC  INCH          RECIEVE NOT READY
C016 B6 80 05          LDA A  ACIA+1    GET CHAR
C019 84 7F          AND A  #$7F        MASK PARITY
C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                                *****
                                * FUNCTION: INHEX - INPUT HEX DIGIT
                                * INPUT: none
                                * OUTPUT: Digit in acc A
                                * CALLS: INCH
                                * DESTROYS: acc A
                                * Returns to monitor if not HEX input

C01E 8D F0  INHEX   BSR    INCH        GET A CHAR
C020 81 30          CMP A  #'0         ZERO
C022 2B 11          BMI    HEXERR      NOT HEX
C024 81 39          CMP A  #'9         NINE
C026 2F 0A          BLE    HEXRTS      GOOD HEX
C028 81 41          CMP A  #'A         NOT HEX
C02A 2B 09          BMI    HEXERR
C02C 81 46          CMP A  #'F
C02E 2E 05          BGT    HEXERR
C030 80 07          SUB A  #7          FIX A-F
C032 84 0F  HEXRTS  AND A  #$0F        CONVERT ASCII TO DIGIT
C034 39          RTS

C035 7E C0 AF  HEXERR JMP    CTRL      RETURN TO CONTROL LOOP
```

# Linguaggi di alto livello e Compilatori

- Per accelerare il processo di programmazione, sono stati sviluppati i **linguaggi di alto livello**, in cui singole istruzioni possono realizzare compiti sostanziali
- Un tipico programma in linguaggio di alto livello contiene molte istruzioni, conosciute come **codice sorgente** del programma
- I programmi traduttori chiamati **compilatori** convertono il codice sorgente in linguaggio di alto livello in linguaggio macchina
- I linguaggi di alto livello permettono di scrivere istruzioni che sembrano quasi come l'inglese quotidiano e contengono notazioni matematiche comuni

High-level program

```
class Triangle {  
    ...  
    float surface()  
        return b*h/2;  
}
```

Low-level program

```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

Executable Machine code

```
0001001001000101  
0010010011101100  
10101101001...
```

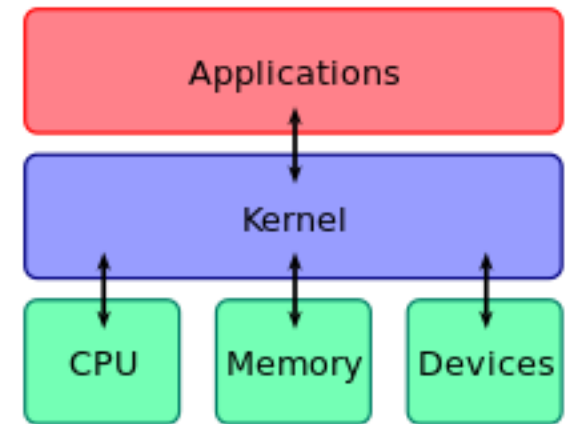
Fonte: Charles Dahab @Medium

# Interpreter

- Compilare un ampio programma in linguaggio di alto livello in linguaggio macchina può richiedere un considerevole tempo di elaborazione
- Gli **interpreter** eseguono i programmi in linguaggio di alto livello direttamente
- Gli interpreter evitano i ritardi della compilazione, ma il codice viene eseguito più **lentamente** rispetto ai programmi compilati
- Alcuni linguaggi di programmazione, come Java e Python, utilizzano una combinazione intelligente di compilazione e interpretazione per eseguire i programmi

# Sistemi operativi

- I sistemi operativi sono **software** che rendono l'uso dei computer più comodo per utenti, sviluppatori di software e amministratori di sistema
- Forniscono servizi che permettono alle applicazioni di eseguire in modo sicuro, efficiente e simultaneo
- Il software che contiene i componenti principali del sistema operativo è chiamato **kernel**
- **Linux, Windows e macOS** sono sistemi operativi desktop popolari
- Ognuno di essi è *parzialmente scritto* in C
- I sistemi operativi mobili più popolari per smartphone e tablet sono **Android** di Google e **iOS** di Apple



Fonte: Wikipedia



Fonte: AlmaBetter

# Windows e Linux

- **Windows:** Sviluppato da Microsoft, sistema operativo proprietario.
- **Linux:** Sviluppato dalla comunità open source
- **Compatibilità e Utilizzo:**
  - **Windows:** Ampio supporto per hardware e software commerciale, comune su desktop e laptop, adatto anche per giochi.
  - **Linux:** Ampio supporto per hardware e software open source, prevalente su server e sistemi embedded, meno supporto per software commerciale.
- **Sicurezza:**
  - **Windows:** Aggiornamenti regolari e integrazione con Microsoft Defender, ma storicamente più vulnerabile a malware.
  - **Linux:** Considerato altamente sicuro, grazie alla sua struttura e alla comunità di sviluppo attiva che contribuisce alla rapida risoluzione di vulnerabilità.
- **Costo:**
  - **Windows:** Licenza a pagamento, con costi associati per aggiornamenti e supporto.
  - **Linux:** Gratuito, con possibilità di supporto a pagamento in alcune distribuzioni commerciali.

# Paradigma Open-source

- **Accesso al codice:** Chiunque può visualizzare, modificare e distribuire il codice sorgente, favorendo trasparenza e innovazione
- **Collaborazione:** Sviluppato da una comunità globale, permette contributi rapidi e risoluzione di problemi
- **Libertà d'uso:** Gli utenti possono personalizzare il software senza le restrizioni delle licenze proprietarie
- **Indipendenza:** Riduce la dipendenza dai fornitori e promuove sostenibilità con modelli di business alternativi
- **Sicurezza:** La trasparenza del codice facilita la rapida identificazione e correzione delle vulnerabilità



Fonte: Wikipedia

# Organizzazioni Open-Source

- **GitHub:** Fornisce strumenti per la gestione di progetti open-source, con milioni di progetti in fase di sviluppo
- **The Apache Software Foundation:** Originariamente creatori del server web Apache, ora supervisiona oltre 350 progetti open-source, inclusi molte tecnologie per l'infrastruttura dei Big Data
- **The Eclipse Foundation:** Supporta l'Eclipse Integrated Development Environment (IDE), che aiuta i programmatori nello sviluppo del software
- **The Mozilla Foundation:** Creatori del browser web Firefox e promotori di software open-source per la navigazione e altre applicazioni
- **OpenML:** Si concentra su strumenti e dati open-source per il machine learning, facilitando la collaborazione e l'innovazione nel campo dell'apprendimento automatico
- **Python Software Foundation:** Responsabile del linguaggio di programmazione Python, sostenendo la crescita e lo sviluppo della sua comunità e delle sue librerie

# Open-Source A.I.

- **Piattaforma collaborativa:**  
Hugging Face facilita la condivisione e l'esplorazione di progetti open-source nel machine learning
- **Empowerment e condivisione:**  
Supporta ingegneri e scienziati nell'apprendimento e nella collaborazione per un'AI aperta ed etica
- **Cuore dell'AI:** Leader nello sviluppo e nella condivisione di modelli di linguaggio di grandi dimensioni (**LLMs**)



## Hugging Face

Downloads last month  
2,736,285

🔗 Safetensors ⓘ Model size 8.03B params Tensor type BF16 ↗

⚡ Inference API ⓘ Warm ▾

📄 Text Generation Examples ▾

Input a message to start chatting with  
meta-llama/Meta-Llama-3.1-8B-Instruct.

Your sentence here... Send

🔗 [meta-llama/Meta-Llama-3.1-8B-Instruct](#)

Fonte: Hugging Face



# Quiz

Vai su **wooclap.com** e usa il codice **XGHTDZ**



Rispondete in coppia:



Rispondete VERO o FALSO:

- Un terabyte è più grande di un petabyte: ①
- Un kilobyte è esattamente 1000 byte: ②
- Un gigabyte è 1024 megabyte: ③
- Qualsiasi computer può comprendere direttamente solo il proprio linguaggio macchina, definito dal design dell'hardware: ④
- I programmi traduttori chiamati assembler, convertono i programmi in linguaggio assembly in linguaggio macchina alla velocità del computer: ⑤
- I programmi interpreti, sviluppati per eseguire direttamente i programmi in linguaggio di alto livello, evitano il ritardo della compilazione e vengono eseguiti più rapidamente rispetto ai programmi compilati: ⑥
- Linux è di gran lunga il sistema operativo desktop più utilizzato al mondo: ⑦
- Un campo può contenere tanti record: ⑧

Clicca sullo schermo di proiezione per avviare la domanda



# Il linguaggio C

- Evoluzione di due linguaggi del passato: BCPL (1967) e B (1970)
- Pensato per sistemi operativi e compilatori, inventato nel 1972 nei Bell Laboratories da Dennis Ritchie
- Linguaggio di sviluppo di UNIX, ora implementato in molti sistemi operativi moderni
- **Hardware-independent:** I programmi scritti in C possono essere lanciate sulla maggior parte dei computer con poche modifiche
- Per ottimizzare performance:
  - Sistemi operativi
  - Sistemi embedded
  - Sistemi di comunicazione

# Libreria standard C

- I programmi in C sono costituiti da elementi chiamati **funzioni**
- È possibile programmare tutte le funzioni necessarie per formare un programma in C
- Tuttavia, la maggior parte dei programmatori C sfrutta la ricca collezione di funzioni esistenti nella **libreria standard** del C
- Ci sono due aspetti principali dell'apprendimento della programmazione in C:
  - Apprendere il linguaggio C stesso
  - Apprendere come utilizzare le funzioni nella libreria standard del C

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

int main()
{
    printf("Hello World");

    return 0;
}
```

# Programmare in C

- Quando programmiamo in C, utilizziamo tipicamente i seguenti elementi costitutivi:
  - Funzioni della libreria standard del C
  - Funzioni della libreria C open-source
  - Funzioni che creiamo
- Funzioni create da altre persone e messe a disposizione
- Ci concentriamo sull'utilizzo della libreria standard del C esistente per ottimizzare gli sforzi nello sviluppo del programma e evitare di "reinventare la ruota".
- Questo è chiamato **riutilizzo del software**

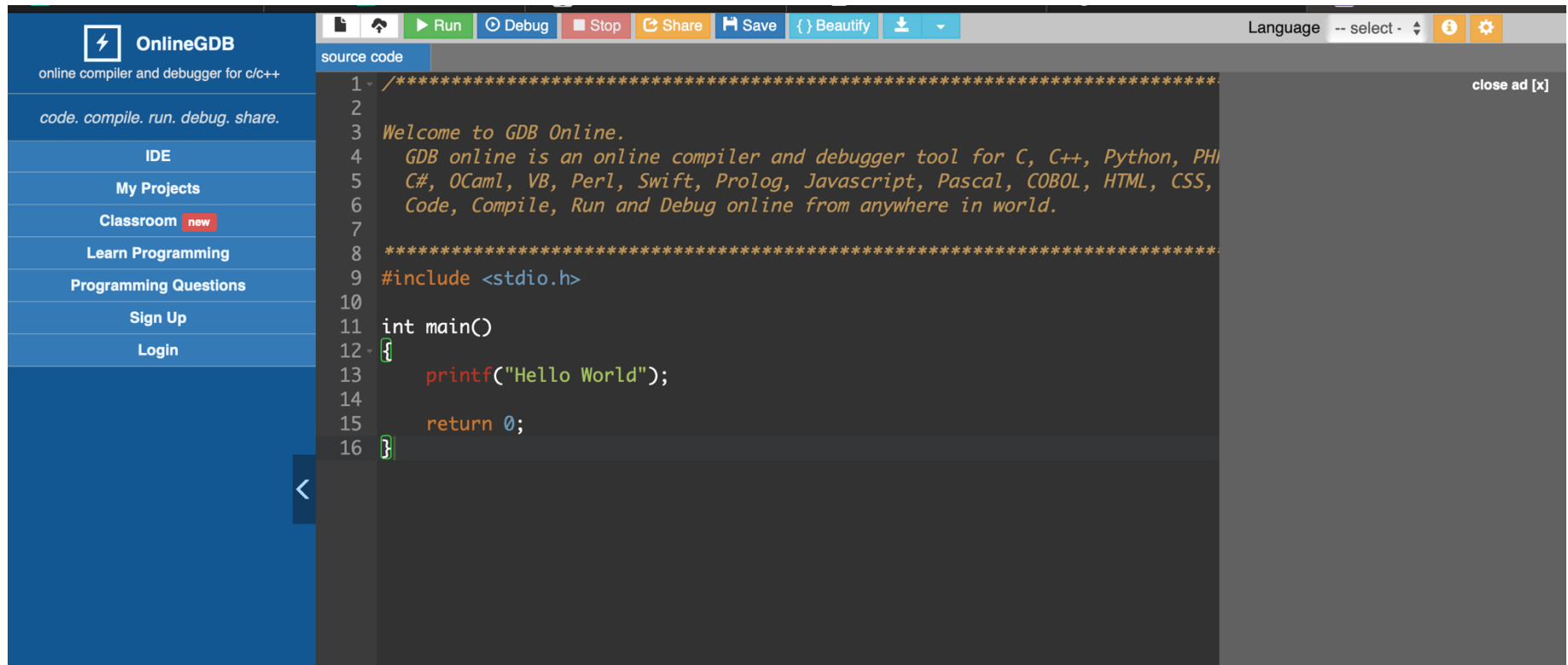
# Sviluppo di un programma C

- I sistemi C generalmente consistono di diverse parti:
  - Un ambiente di sviluppo del programma
  - Il linguaggio
  - La libreria standard del C
- I programmi C tipicamente passano attraverso sei fasi per essere eseguiti:

- 1. Modifica (Edit):** Scrittura del codice sorgente
- 2. Preprocesso (Preprocess):** Elaborazione del codice sorgente per includere file e gestire macro
- 3. Compilazione (Compile):** Traduzione del codice sorgente in codice oggetto
- 4. Collegamento (Link):** Unione del codice oggetto con le librerie per creare l'eseguibile
- 5. Caricamento (Load):** Trasferimento dell'eseguibile in memoria
- 6. Esecuzione (Execute):** Esecuzione del programma

# Fase 1: Creare il programma

- Consiste nell'utilizzare un software di editor per scrivere il programma



The screenshot displays the OnlineGDB web interface. On the left is a blue sidebar with navigation links: OnlineGDB (online compiler and debugger for c/c++), code, compile, run, debug, share, IDE, My Projects, Classroom (new), Learn Programming, Programming Questions, Sign Up, and Login. The top of the editor features a toolbar with icons for Run, Debug, Stop, Share, Save, Beautify, and download. The main area shows a C program with the following code:

```
1  /*****  
2  
3  Welcome to GDB Online.  
4  GDB online is an online compiler and debugger tool for C, C++, Python, PHP,  
5  C#, OCaml, VB, Perl, Swift, Prolog, Javascript, Pascal, COBOL, HTML, CSS,  
6  Code, Compile, Run and Debug online from anywhere in world.  
7  
8  *****/  
9  #include <stdio.h>  
10  
11  int main()  
12  {  
13      printf("Hello World");  
14  
15      return 0;  
16  }
```

## Fase 2 e 3: Preprocessing e Compilazione del programma C

- Viene lanciato il comando per compilare il programma
- Il **compilatore** traduce il codice in linguaggio macchina
- Il comando di compilazione invoca prima un **preprocessore**
- Il preprocessore esegue manipolazioni testuali sui file sorgente del programma, come:
  - Inserimento del contenuto di altri file
  - Sostituzioni di testo

## Fase 2 e 3: Preprocessing e Compilazione del programma C

- Un **errore di sintassi** si verifica quando il compilatore non riesce a riconoscere una dichiarazione perché viola le regole del linguaggio
- Il compilatore emette un messaggio di errore per aiutare a individuare e correggere la dichiarazione errata
- Lo standard del C non specifica la formulazione dei messaggi di errore
- Gli errori di sintassi sono anche chiamati **errori di compilazione** o **errori di tempo di compilazione**
- Un **errore logico** si verifica quando il programma compila ed esegue senza errori, ma produce risultati sbagliati a causa di una logica errata nel codice. Questi errori sono più difficili da rilevare e correggere rispetto agli errori di sintassi.



# Fase 4: Linking

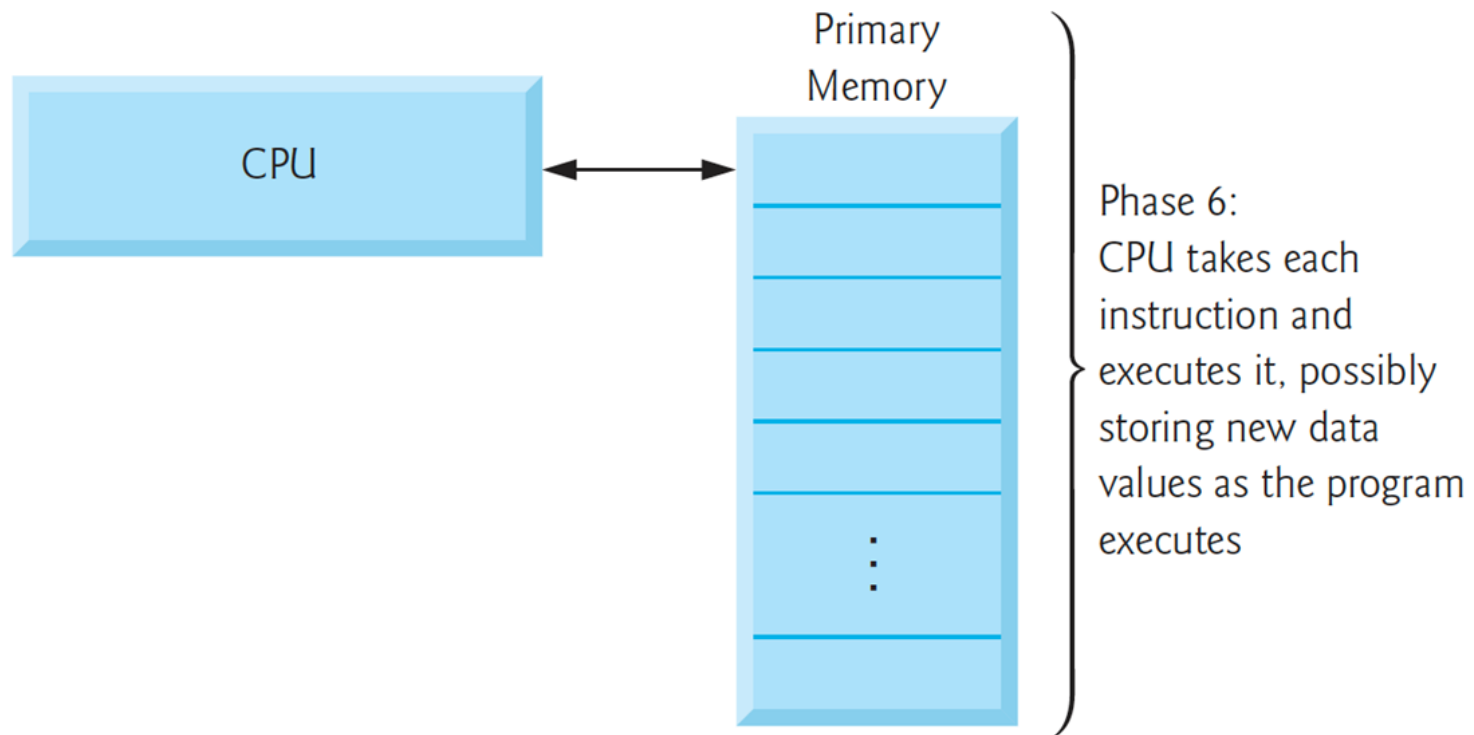
- I programmi in C utilizzano funzioni definite altrove:
  - Librerie standard
  - Librerie open-source
  - Librerie private di un progetto specifico
- Il codice oggetto prodotto dal compilatore C contiene tipicamente "buchi"
- Il linker collega il codice oggetto del programma con il codice delle funzioni mancanti per produrre **un'immagine eseguibile** (senza pezzi mancanti)

# Fase 5: Loading

- Prima che un programma possa essere eseguito, il sistema operativo deve caricarlo in memoria
- Il loader trasferisce l'immagine eseguibile dal disco alla memoria
- Anche i componenti aggiuntivi delle librerie condivise che supportano il programma vengono caricati

# Fase 6: Execution

- Infine il computer, sotto il controllo della CPU, esegue il programma un'istruzione alla volta



# Problemi durante l'esecuzione

- Gli errori che si verificano durante l'esecuzione dei programmi sono chiamati **errori di runtime** (o errori di tempo di esecuzione)
- Gli **errori fatali** causano l'interruzione immediata del programma senza che questo completi il suo compito con successo
- Gli **errori non fatali** permettono ai programmi di completare l'esecuzione, ma spesso producono risultati errati

# Standard Input, Standard Output e Standard Error

- La maggior parte dei programmi C gestisce input e/o output di dati
- Alcune funzioni C ricevono input da **stdin** (il flusso di input standard), che è normalmente la tastiera
- I dati vengono spesso inviati a **stdout** (il flusso di output standard), che è normalmente lo schermo del computer
- I dati possono anche essere inviati a dispositivi come dischi e stampanti
- Esiste anche un flusso di errore standard chiamato **stderr**, che è normalmente collegato allo schermo e viene utilizzato per visualizzare i messaggi di errore

# Recap

- Algebra di Boole, operatori logici di base e tabelle di verità
  - Proprietà dell'algebra di Boole
  - Leggi di De Morgan
  - Equivalenze tra espressioni
- Gerarchia dei dati in un computer
- Linguaggi di programmazione e sistemi operativi
- Introduzione al linguaggio C e fasi di sviluppo di un programma