Titolo unità didattica: Introduzione al linguaggio C [03]

Titolo modulo : Variabili e tipi in C [03-C]

Sviluppo di semplici programmi C

Argomenti trattati:

- ✔ Differenze tra i linguaggi
- ✔ Editor, compilatore e debugger
- ✔ tipi di dati scalari in C
- ✔ variabili e costanti in C
- ✔ operazione di assegnazione in C
- ✔ operatori aritmetici ed espressioni in C

Prerequisiti richiesti: AP-02-*-T, AP-03-02-C

Linguaggio macchina, assembly e di alto livello

Paola Barra

Linguaggio macchina, o linguaggio binario:

```
100101011001
010011111011
111010101101
01010101010
```

Linguaggio assembly

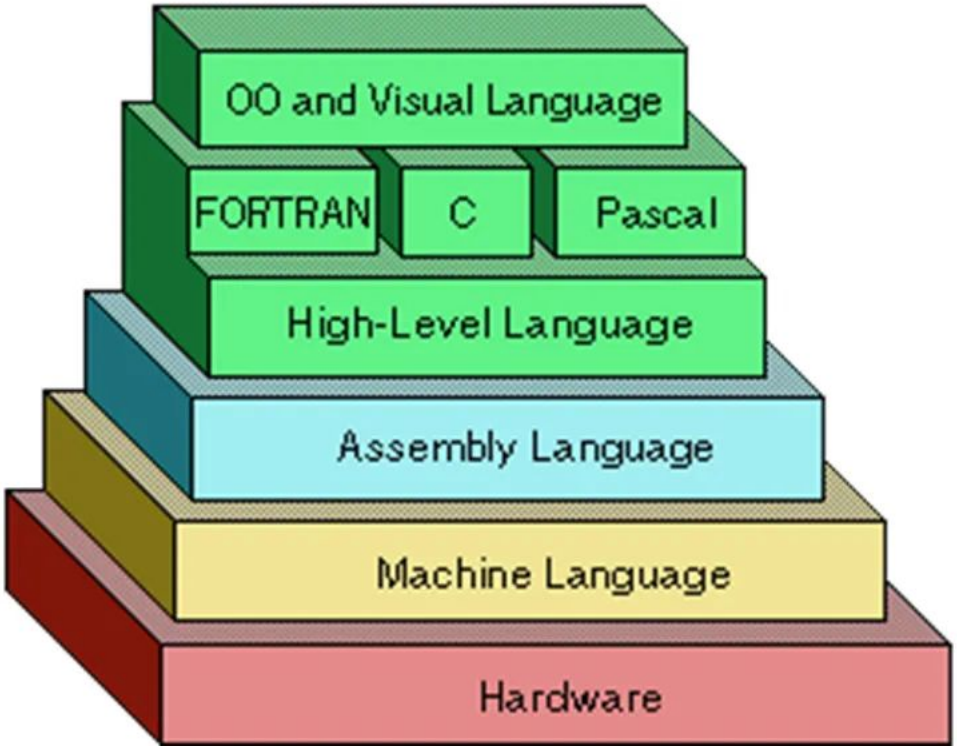
```
      movl   $0, %ecx
.loop:
      cmpl   $1, %edx
      jle    .endloop
      addl   $1, %ecx
      movl   %edx, %eax
      andl   $1, %eax
      je     .else
      movl   %edx, %eax
      addl   %eax, %edx
      addl   %eax, %edx
      addl   $1, %edx
      jmp    .endif
.else:
      sarl   $1, %edx
.endif:
      jmp    .loop
.endloop:
```

Linguaggio di programmazione:

```
void CompilerGCC::RemoveBuildProgressBar()
{
    if (m_BuildProgress)
    {
        wxSizer* s = m_Log->GetSizer();
        if (s)
        {
            s->Detach(m_BuildProgress);
            m_BuildProgress->Destroy();
            m_BuildProgress = 0;
            s->Layout();
        }
    }
}
```

High-Level Languages:

1. Python
2. JavaScript
3. Ruby
4. PHP
5. Java
6. C#



Mid to High-Level Languages:

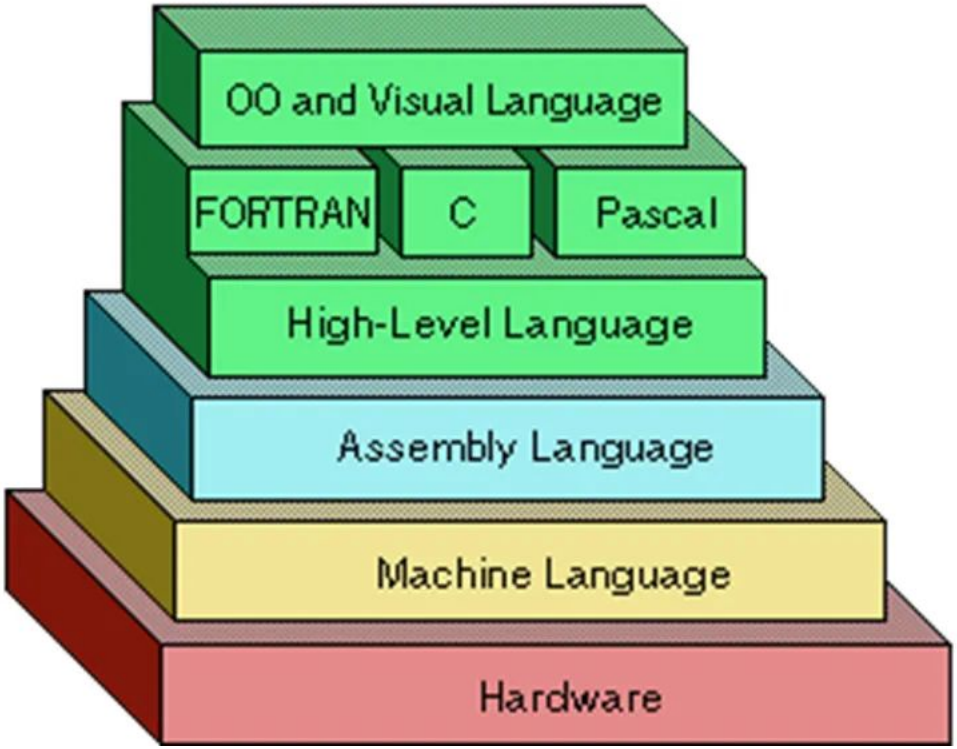
1. Go
2. Scala
3. Erlang
4. Clojure
5. Haskell

Mid-Level Languages:

1. Rust
2. C++
3. OCaml
4. Scheme

Low-Level Languages:

1. Zig
2. C
3. Assembly
4. Machine Code



Assembly VS linguaggio macchina

Ogni computer può comprendere direttamente il proprio linguaggio macchina.

Il **linguaggio Assembly**, invece, è una rappresentazione delle stesse istruzioni in una forma testuale più comprensibile dall'uomo.

L'assembly è stato creato per facilitare l'uso di 0 e 1 con abbreviazioni in inglese.

Ad esempio, l'istruzione seguente somma il contenuto dei registri EAX e EBX mettendo il risultato nel registro EAX.

```
00000011 11000011
```

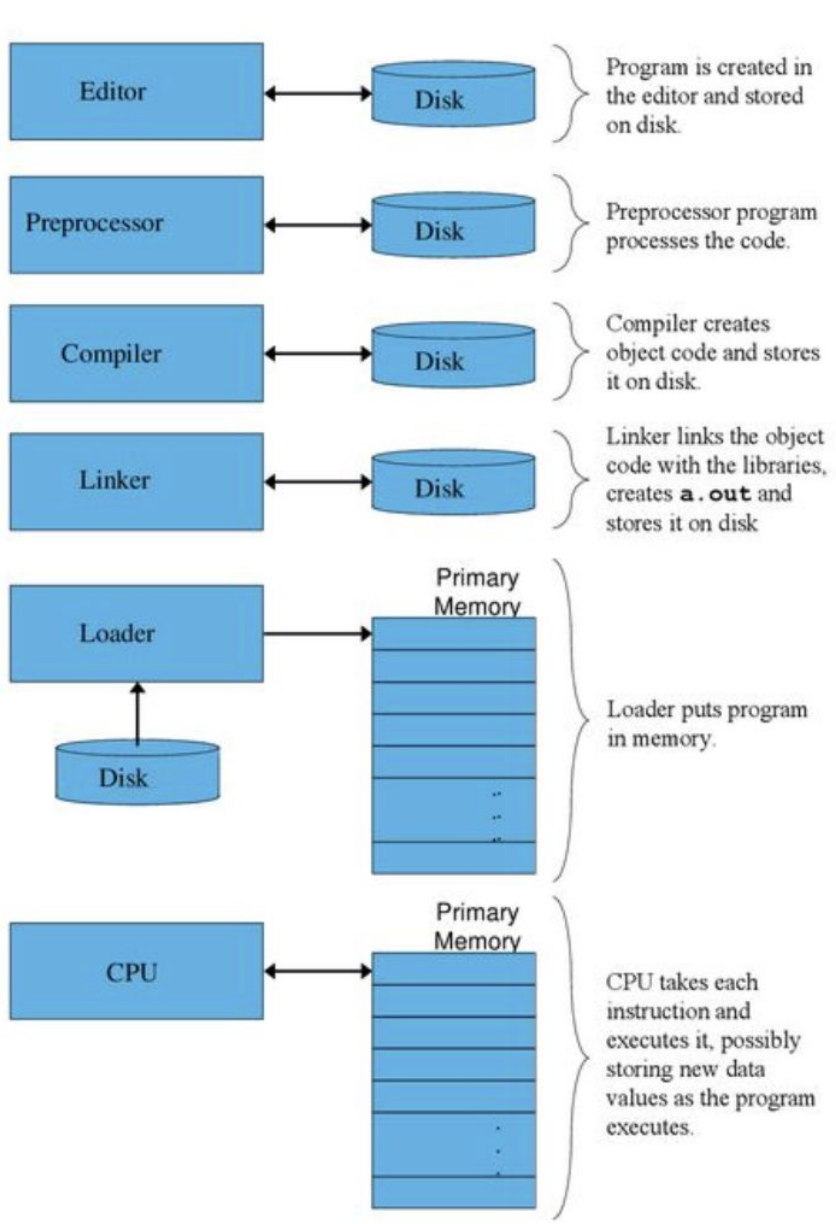
per compiere la stessa operazione di somma dei registri EAX e EBX basta scrivere in Assembly

```
MOV EAX, EBX
```

Editor
compilatore
debugger

Fasi di un programma C

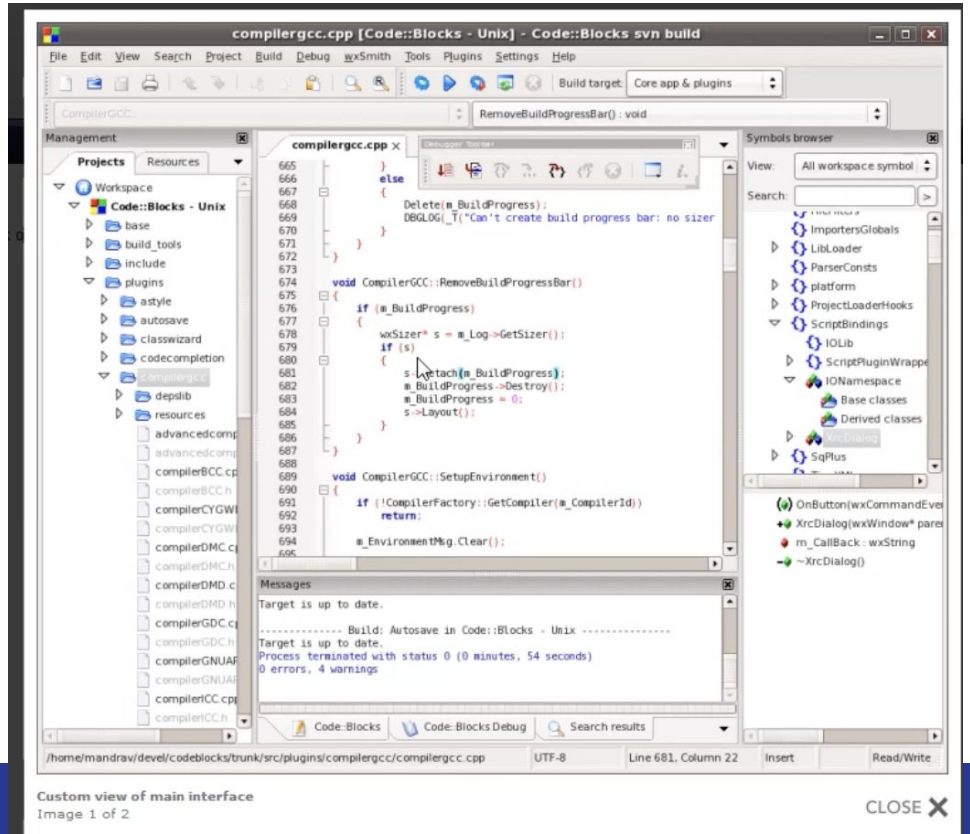
1. *Edit*
2. *Preprocess*
3. *Compile*
4. *Link*
5. *Load*
6. *Execute*

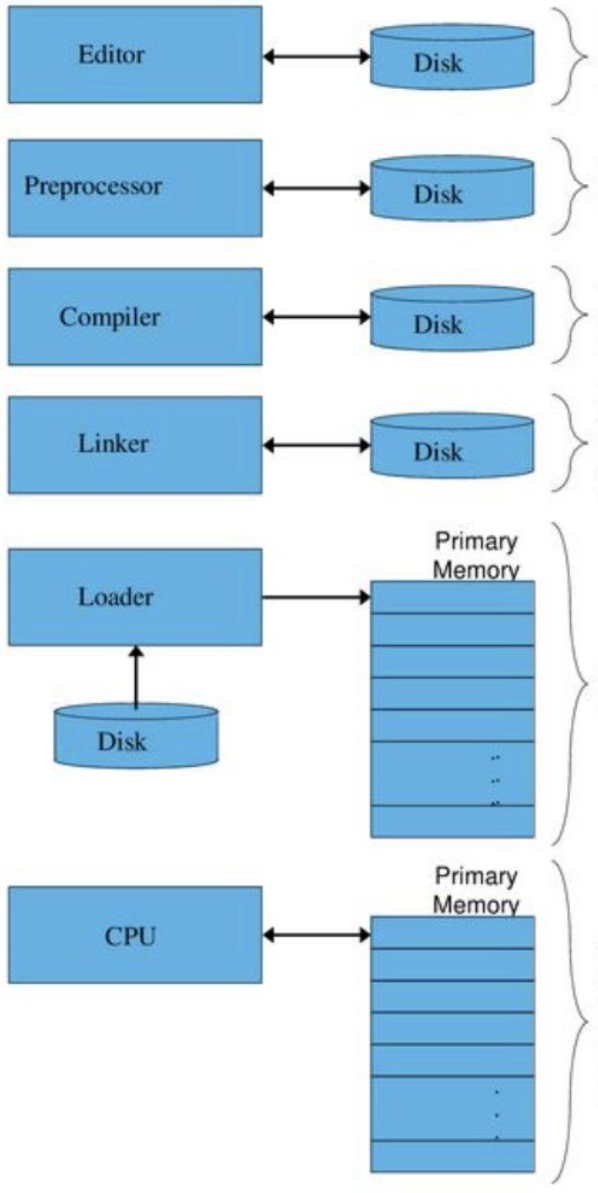


L’editor : IDE (Ambiente di sviluppo integrato)

All’interno dell’editor scriveremo il codice.

L’editor ci aiuta dandoci indicazioni,
suggerendo il codice, segnalando gli errori.





Il compilatore

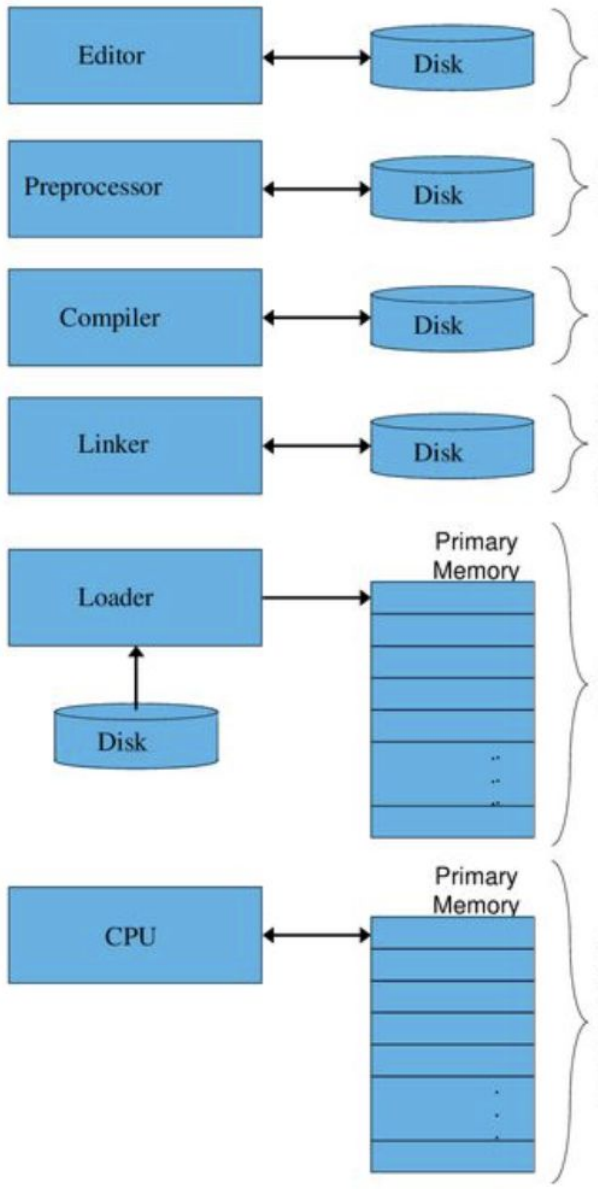
Il processore non è capace di capire il linguaggio di programmazione.

Il linguaggio di programmazione è qualcosa di comprensibile per l'uomo ma molto lontano dal linguaggio della macchina.

Il compilatore traduce il nostro programma "il codice sorgente" in un linguaggio comprensibile al processore "programma eseguibile" : una sequenza di 0 e 1.

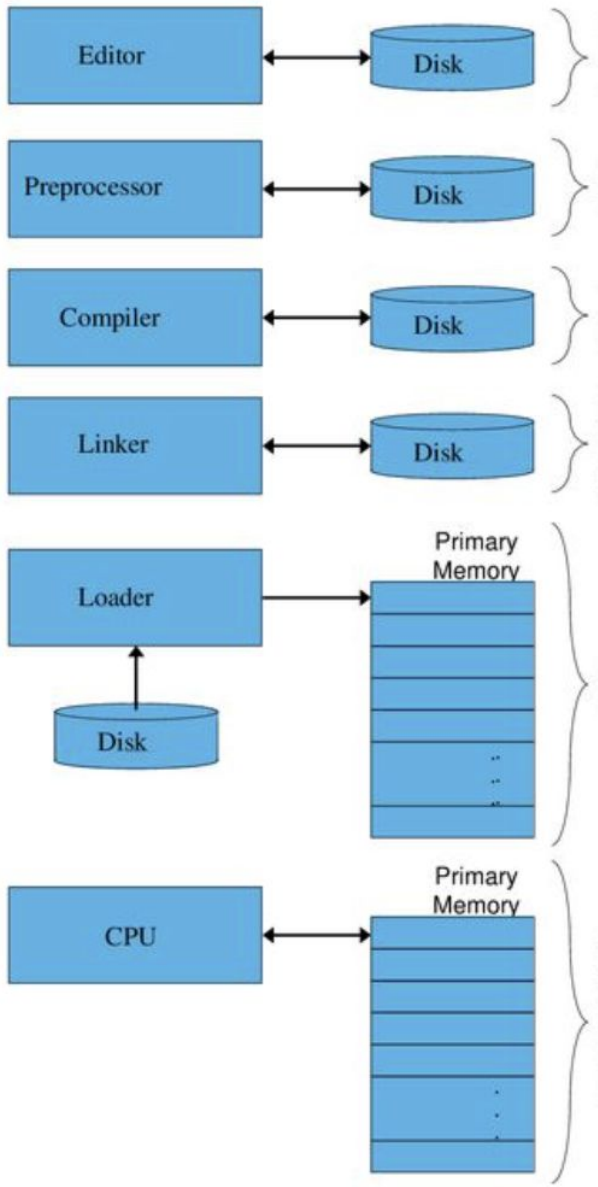
Una volta eseguito prende il codice sorgente e controlla che non ci siano errori di tipo formali.

Nel caso in cui ci fossero errori questi verranno segnalati al programmatore che dovrà correggerli



Il preprocessore

Prima della traduzione viene eseguito il preprocessore che crea una serie di manipolazione nel file come inclusione di altri file nel programma o sostituzione di simboli speciali con testo nel programma.



Linking

I programmi scritti in C contengono tipicamente dei riferimenti a funzioni definite altrove, come le librerie.

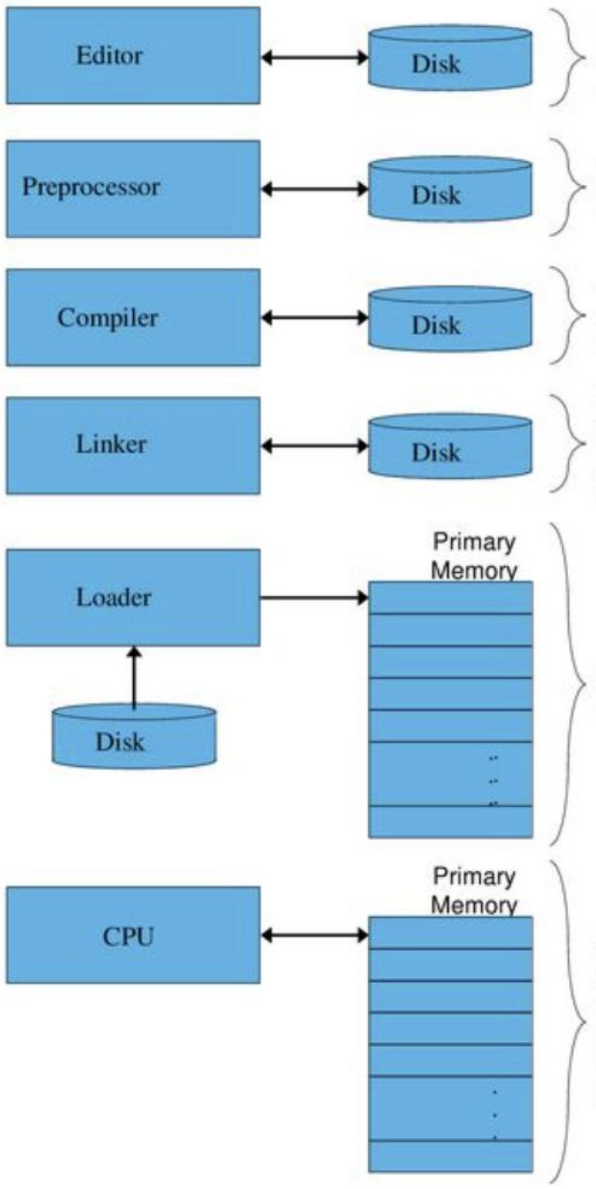
Quindi il codice oggetto prodotto dal compilatore conterrà dei “buchi” dovuti a queste parti mancanti.

Il linker collega il codice oggetto con quello delle funzioni mancanti per produrre un’immagine eseguibile.

Il comando da terminare linux è cc o gcc quindi

gcc main.c

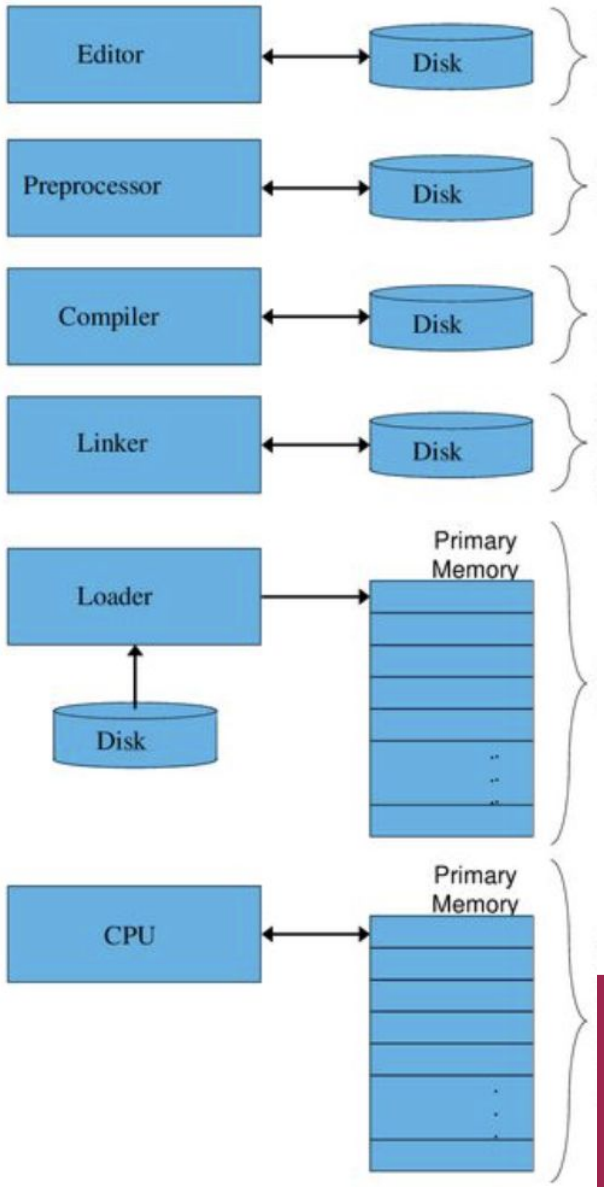
questo produrrà il file eseguibile **a.out**



Caricamento

Prima di essere eseguito un programma dovrà essere caricato in memoria.

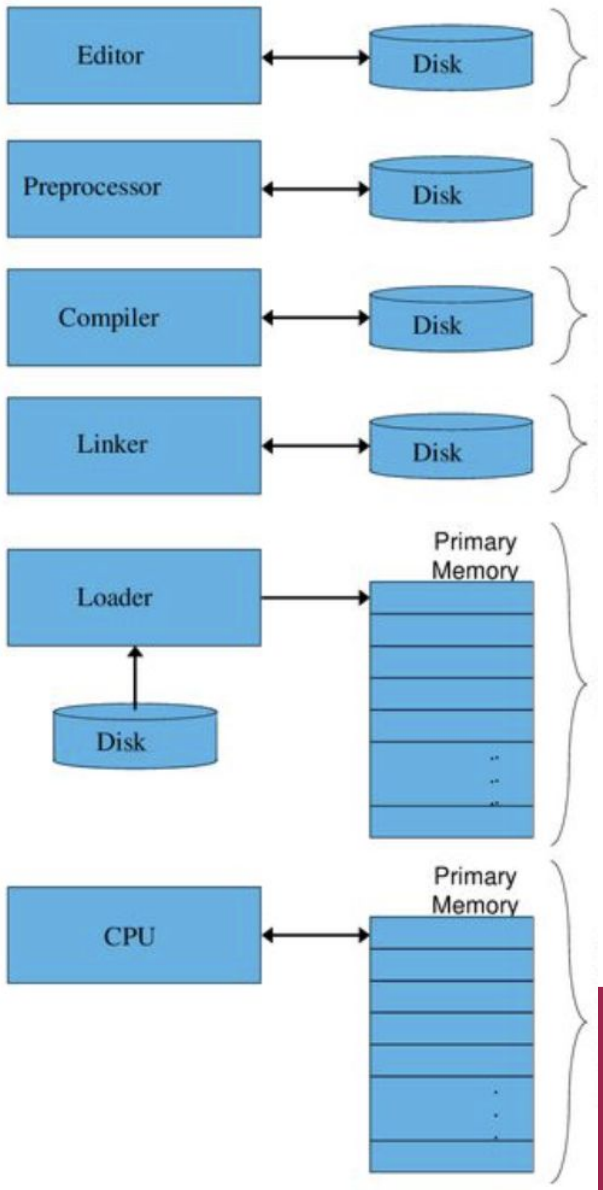
Quindi questa fase prende il file eseguibile e lo carica in memoria.



CPU

una volta caricato in memoria il file può essere eseguito.

Da terminale UNIX basta chiamare **a.out**



Riassumendo…

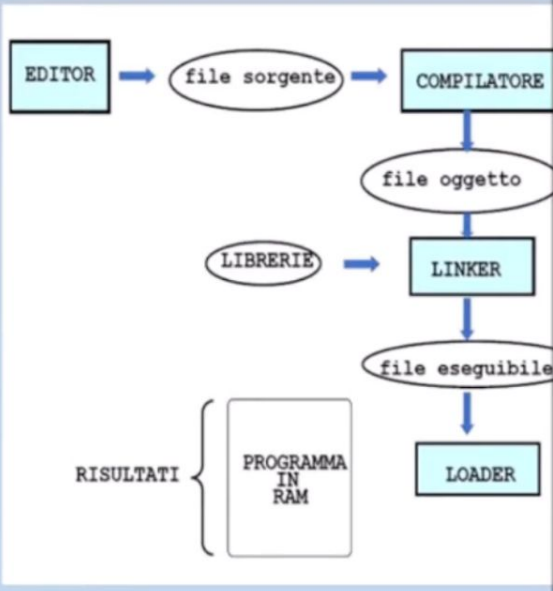
Compilatore

E' un software che traduce le istruzioni scritte in un linguaggio di programmazione in linguaggio comprensibile al microprocessore, ovvero in Assembler.

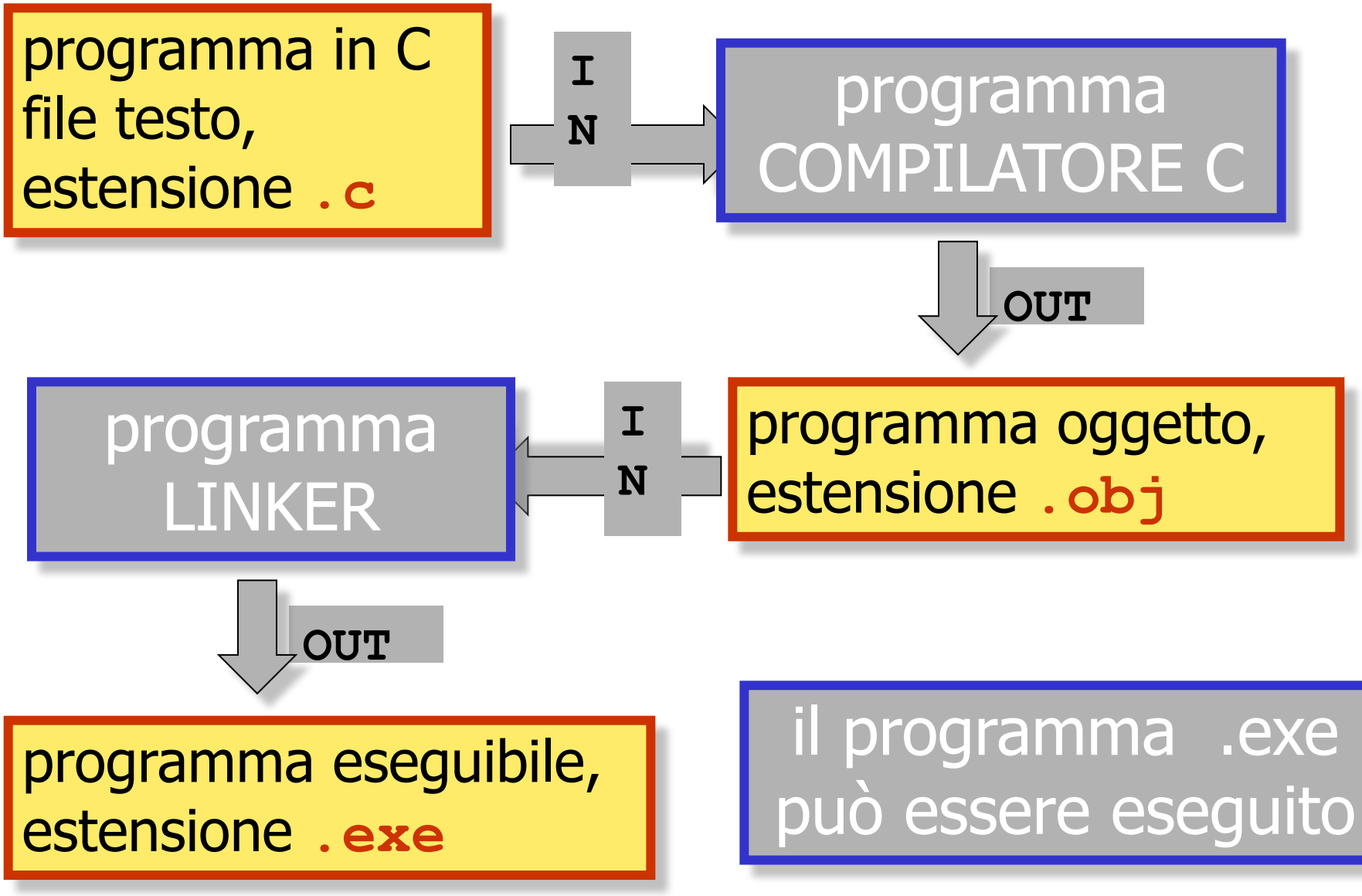
Il sorgente (scritto dal programmatore) viene letto dal compilatore, che effettua il controllo sintattico sulle istruzioni e le traduce in linguaggio macchina producendo un secondo file, detto **object file** che, comunque, non è ancora eseguibile, in quanto non incorpora il codice binario delle funzioni esterne al linguaggio che solitamente sono raccolti in nelle librerie.

E' dunque necessario un altro software: ll **linker**, che incorpora nell'object file, i codici eseguibili dalle librerie producendo un terzo file direttamente eseguibile.
Compilazione e linkaggio solitamente sono assemblati in un unico software.

In caso di errori, il compilatore li segnala e non produce alcun object file.
Il programmatore deve analizzare il sorgente, correggere gli errori e ritentare la compilazione sino a quando vengono rilevati errori e viene prodotto un object file.
Anche il linker può rilevare errori per esempio non trovare nella libreria una funzione. Il programmatore cercherà l'errore per correggerlo ed effettuerà nuovamente la compilazione.



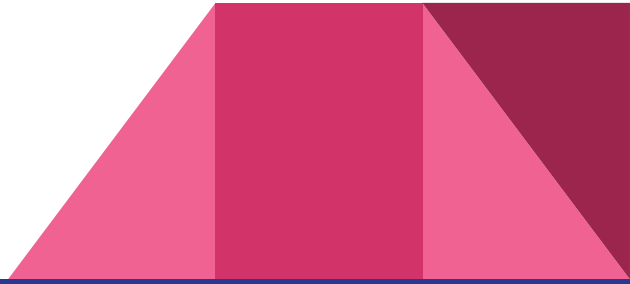
processo di **compilazione-linking-esecuzione**



Il debugger

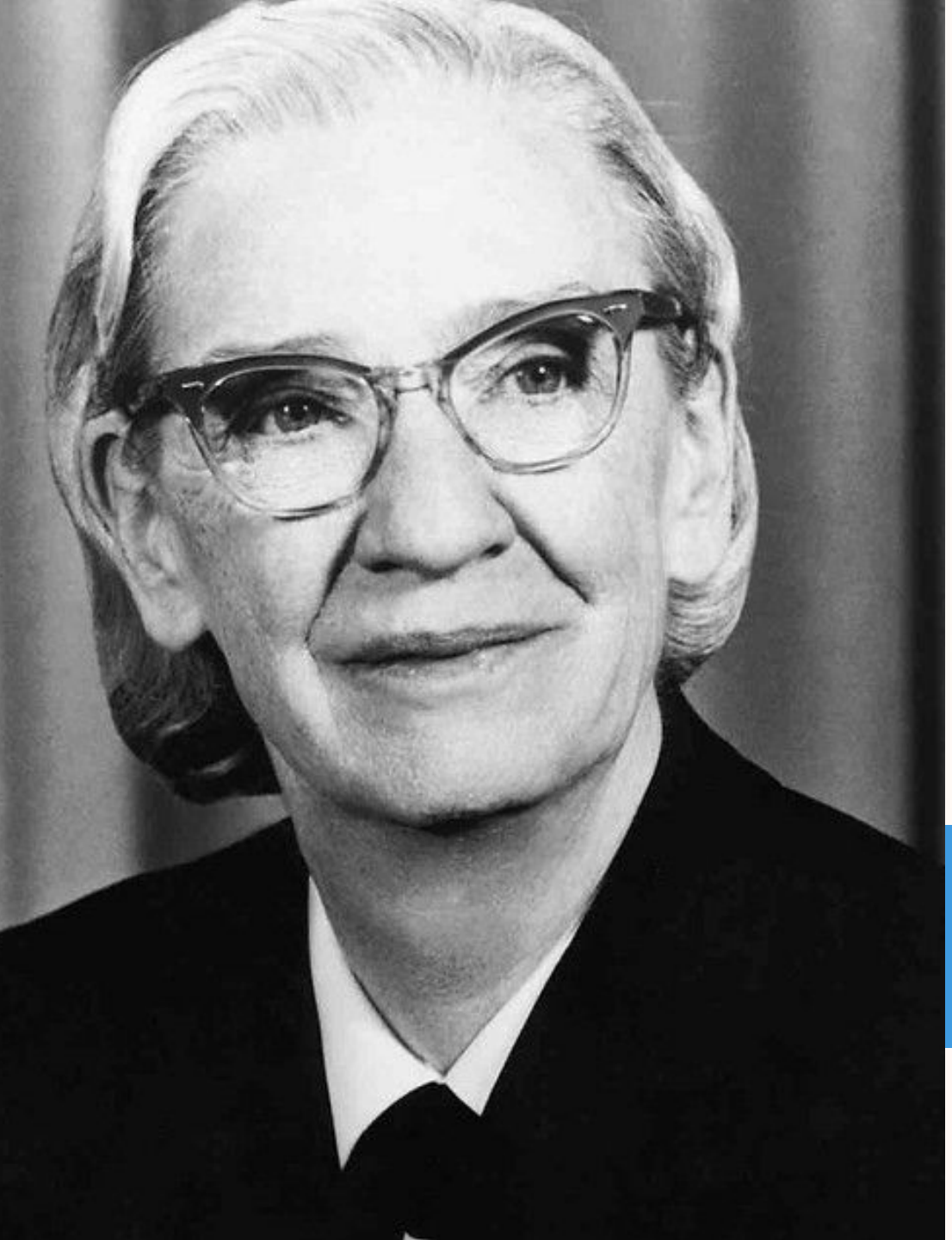
Un **debugger** in informatica è un programma/software specificatamente progettato per l'analisi e l'eliminazione dei bug (debugging), ovvero errori di programmazione interni al codice di altri programmi.

Assieme al compilatore è fra i più importanti strumenti di sviluppo a disposizione di un programmatore, spesso compreso all'interno di un ambiente integrato di sviluppo (IDE), in quanto in grado di aiutare il programmatore ad individuare errori di semantica all'interno del codice sorgente del programma, altrimenti di difficile individuazione in fase di runtime.



Cosa è un bug?

apriamo una piccola parentesi…



Grace Murray Hopper è passata alla storia come una delle figure più eminenti dell'informatica, la prima ad aver pensato e realizzato un linguaggio di programmazione indipendente dalla macchina (il COBOL) e ad aver inventato il metodo del debugging (eliminazione dei bug informatici attraverso analisi periodiche e continue del codice sorgente del programma).

Grace Hopper

New York 1906 - 1992

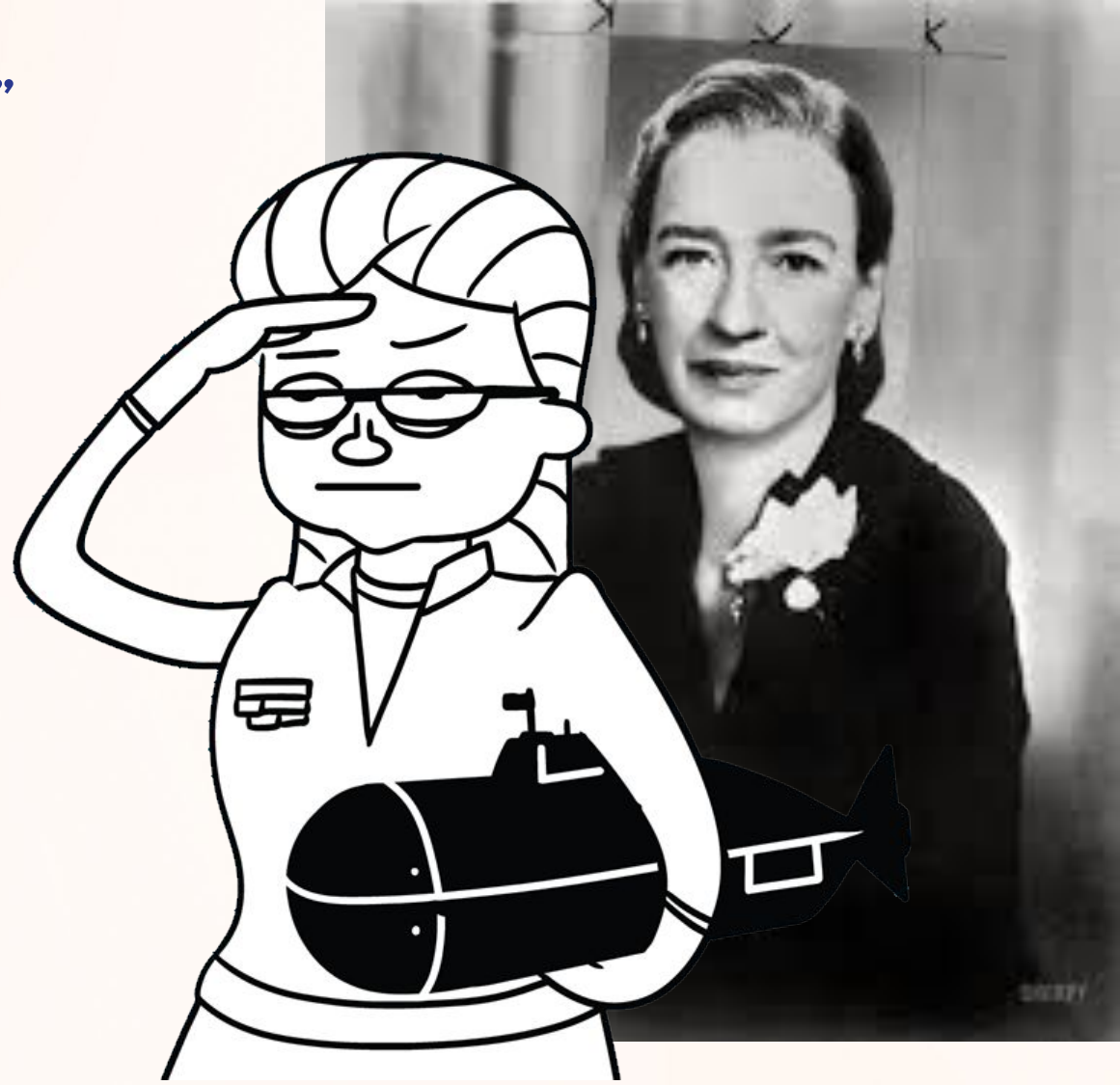
Women Techmakers

International Women's Day 2022

Ragazzina vivace e “sveglia”

Praticava basket, Hockey su prato, Pallanuoto... Sport “maschili”.

1941 Diventa professore di matematica
Nonostante frequentasse scuole private femminili, che insistevano molto su un certo tipo di educazione “per signore”, suo padre la incentivò sempre a lasciarsi alle spalle gli stereotipi dei ruoli femminili. Praticava basket, hockey su prato e pallanuoto.



Nel 1941 Grace lascia famiglia e insegnamento per arruolarsi volontaria in Marina.

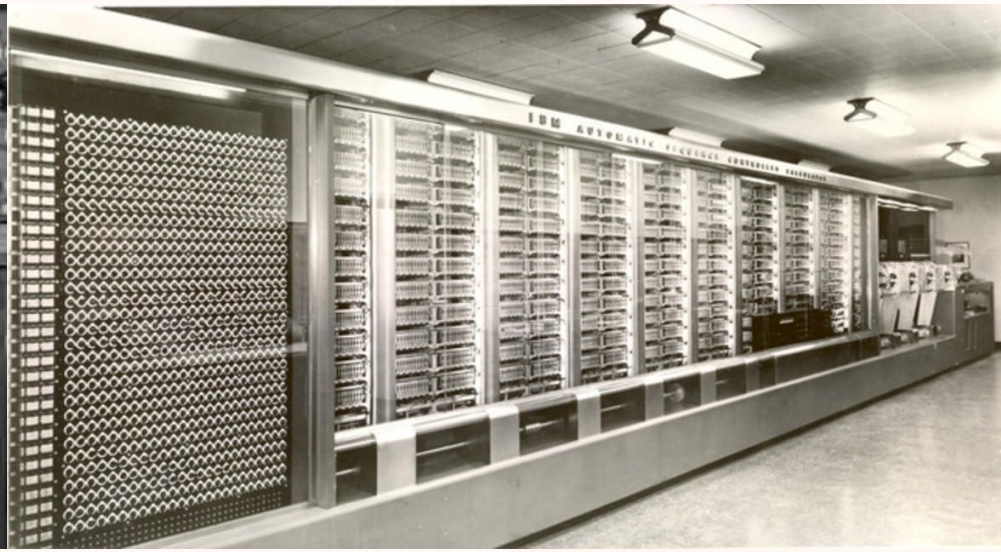
COBOL

Mark I
4,5 t, 16 m x 2,4 m

Mark I: 4 tonnellate e mezzo, 16 m x 2,4 m.

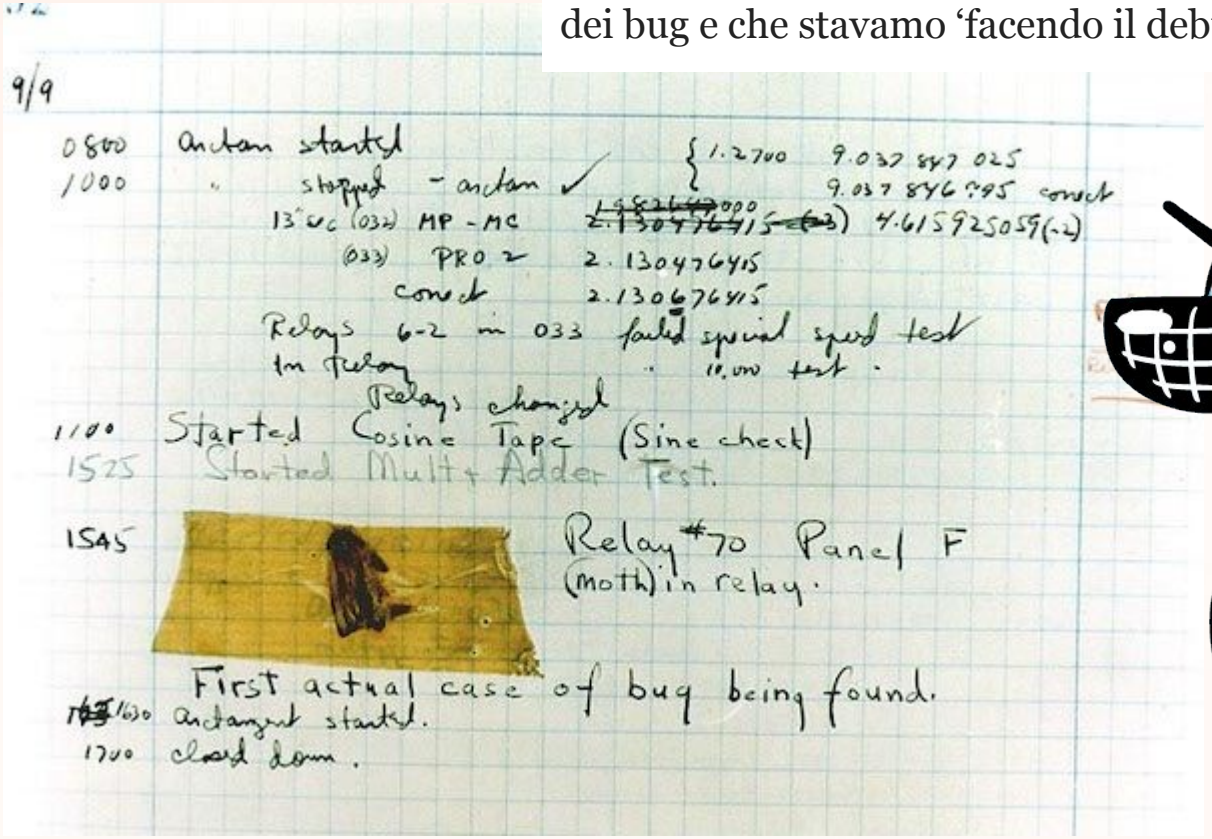
Hopper ha insegnato a quella macchina, un'infinità di operazioni attraverso comandi impressi su nastro perforato.

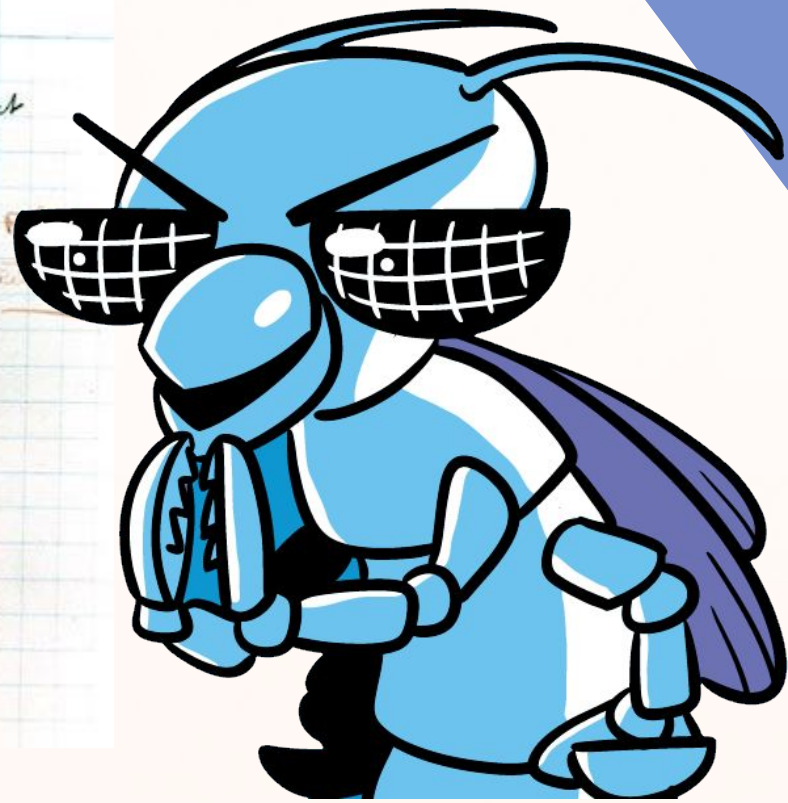
Il risultato è un testo epocale, A Manual of Operation for the Automatic Sequence Controlled Calculator, pubblicato nel 1946. **La Bibbia dei computer.**



Debugging

Il termine *bug*, è vero, era stato impiegato sin dai tempi di Thomas Edison per indicare problemi e malfunzionamenti meccanici, ma con questo episodio esso veniva finalmente introdotto nel linguaggio della nascente informatica, tanto che, avrebbe continuato Grace, “da quel momento in poi, quando qualcosa non andava, dicevamo che c’erano dei bug e che stavamo ‘facendo il debugging’”.

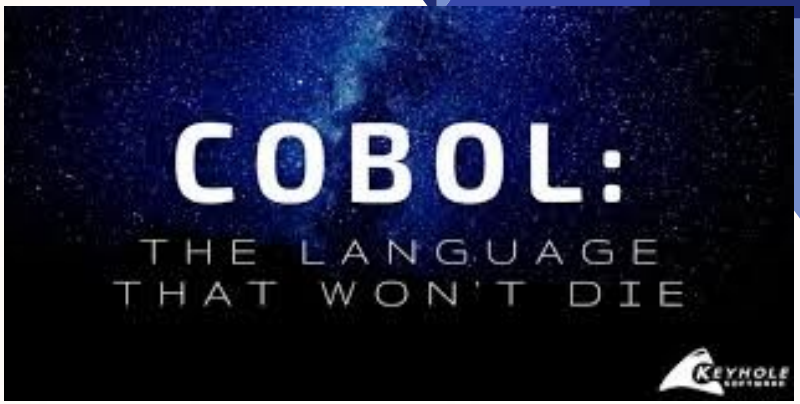




“Avevamo un paio di pinzette”, avrebbe ricordato Grace. “Trovammo una falena di circa quattro pollici di apertura alare e, con molta attenzione, la tirammo fuori e la mettemmo nel nostro ‘registro di bordo’, bloccandola con dello scotch”

COBOL oggi

Nonostante abbia compiuto da poco 50 anni, la programmazione Cobol è ancora al centro di diversi settori di business, in particolare quello bancario e governativo.



Concentrandoci solo sul settore bancario: circa il 43% dei sistemi bancari sono scritti in Cobol, circa l'80% delle transazioni di persona avvengono utilizzando il linguaggio Cobol; circa il 95% delle operazioni ATM si basano su codice Cobol.

….chiusa parentesi

Un semplice programma

I commenti

Ogni funzione dovrebbe essere preceduta da un commento che ne descriva lo scopo!

```
/* Questo è un commento */

//  anche questo è un commento !!

/* Questo commento può essere chiuso */  printf(“Ciao! \n ”);


/* Questo è un
        commento
        su
        più
        righe */


// questo commento non può essere chiuso printf(“Ciao! \n ”);
```

Le parentesi graffe { }

indicano il corpo della funzione.

Quello racchiuso nelle parentesi graffe si chiama “corpo” della funzione.

{ la parentesi graffa aperta indica l’inizio della funzione

} la parentesi graffa chiusa indica la fine della funzione

```
int main(){

    // stampo una frase
    printf(“una frase”):

}
```

printf (“ Hello world \n”) ;

E’ un’istruzione che contiene la funzione “printf”, i suoi argomenti racchiusi tra parentesi tonde, e si chiude con il punto e virgola ;

Ordina la computer di eseguire un’azione: visual izzare sullo schermo una stringa di caratteri indicati tra le virgolette.

Ogni funzione ha delle parentesi tonde nelle quali è racchiuso l’argomento della funzione.

\n non viene visualizzata perchè rappresenta una sequenza di escape che indica il ritorno a capo.

SEQUENZA DI ESCAPE	DESCRIZIONE
\n	Newline (nuova riga). Posiziona il cursore all'inizio della riga successiva
\t	Tabulazione orizzontale. Muove il cursore alla tabulazione successiva
\r	Ritorno a carrello, Posiziona il cursore all'inizio della riga corrente (non lo fa avanzare a quella successiva)
\a	Allarme! Fa suonare il cicalino del sistema
\\	Backslash. Visualizza un carattere backslash in una istruzione printf
\'	Apice singolo. Visualizza un carattere apice singolo in una istruzione printf
\"	Virgolette. Visualizza le virgolette in una istruzione printf

Non ci confondiamo tra / e \

/ slash : da utilizzare per i commenti

\ backslash : da utilizzare per le sequenze di escape

I caratteri \, ‘ e “ sono speciale quindi per essere visualizzati in una printf hanno bisogno di essere preceduti da \

Buone abitudini!

L’ultimo carattere di un argomento di una printf dovrebbe essere sempre \n

Indentare Il corpo delle funzioni in 3 spazi

Esercizi

```
#include<stdio.h>

int main(){
        printf(“Il\nmio\nprogramma\nC\n”)
}
```

```
#include<stdio.h>

int main(){
        printf(“Il mio “);
        printf(“programma C\n”);
}
```

Sommare due interi

Somma due interi

```
#include <stdio.h>

int main(){
        int intero1, intero2, somma;

        printf(“Inserisci il primo intero \n”);
        scanf(“%d”, &intero1 );

        printf(“Inserisci il secondo intero \n”);
        scanf(“%d”, &intero2 );

        somma = intero1 + intero2;
        printf(“La somma è %d \n”, somma);
}
```

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

 # è una direttiva del preprocessore del C.

Questa riga indica al preprocessore di includere nel programma il contenuto del file stdio.h

```
int intero1, intero2, somma;
```

è una dichiarazione.

intero1, intero2 e somma sono i nomi delle variabili.

Una variabile è una locazione di memoria in cui viene immagazzinato un valore.

Queste variabili contengono valori interi

Come scegliere il nome di una variabile?

Può contenere lettere, numeri o _ underscore.

Non deve contenere spazi.

Non può iniziare con un numero

Deve avere una lunghezza inferiore ai 31 caratteri

I nomi di variabili sono ***case sensitive*** *- le lettere minuscole e maiusole sono diverse*

- a1 è diverso da A1

Come scegliere il nome di una variabile?

buone norme...

Scegliere un nome che ne sieghi il contenuto (come somma), per migliorare la leggibilità del codice.

generalmente la prima lettera è una minuscola

le variabili formate da più parole vanno separate con _
(esempio: totale_commissioni)

___

```
scanf("%d", &intero1);
```

%d indica il tipo di dato che dovrà essere immesso dall'utente.

%d è la specifica di conversione dell' intero, la funzione si aspetta un intero.

d sta per intero decimale.

& si chiama operatore di indirizzo

&intero1 indica la locazione di memoria in cui è stata immagazzinata la variabile intero1

Per ora ricordate di aggiungere & quando chiamate una variabile con scanf, in seguito studieremo nel dettaglio il perchè.

```
somma = intero1 + intero2;
```

il calcolo di una istruzione di assegnamento deve essere a destra dell'operatore =

```
printf("La somma è %d \n", somma);
```

(Gli argomenti nella funzione sono separati da virgola)

Nel primo argomento ci sono alcuni caratteri letterali che dovranno essere visualizzati, e la specifica di conversione %d indica che sarà visualizzato un intero.

Il secondo argomento specifica l'intero che ci aspettiamo di visualizzare.

Notare come la modalità è simile in printf e scanf.

Variabili e tipi in C

Tipi di dati scalari in C
(tipi semplici built-in)

- ✔ tipo **intero**
- ✔ tipo **reale**
- ✔ tipo **carattere**

in C è possibile definire **nuovi tipi** di dati scalari
(tipi **user-defined**)

tipo **intero** in C

```
int
```

```
short
long
unsigned short
unsigned int
unsigned long
```

tipo **intero** in C	insiemi dei valori (pc classici, celle di 32 bit)
`int`	-2˙147˙483˙648, +2˙147˙483˙647
`short`	-32˙768, +32˙767
`long`	come `int`
`unsigned short`	0, +65˙535
`unsigned int`	0, +4˙294˙967˙295
`unsigned long`	come `int`

tipo **intero** in C

specificazione di un valore del tipo

numeri interi positivi

`[+]ddddddddddd`

numeri interi negativi

`-ddddddddddd`

il numero delle cifre (`d`) dipende dal particolare tipo C
(dal numero di bit per la rappresentazione in memoria)

tipo **intero** in C

il comando C

```
sizeof(tipo)
```

restituisce il **numero di byte** necessario per la rappresentazione di un valore del tipo

```
sizeof(int)
sizeof(unsigned short)
sizeof(long)
```

tipo **reale** in C

`float`

singola precisione,
8 cifre significative

`double`

doppia precisione,
16 cifre significative

`long double`

tipo **reale** in C

specificazione di un valore del tipo

valori `float` positivi

`[+]dddd.ddddF`

`[+]dddd.ddddE[±]eeF`

valori `float` negativi

`-dddd.ddddF`

`-dddd.ddddE[±]eeF`

il numero delle cifre significative (`d`) è al più 8

il numero delle cifre dell'esponente (`e`) è al più 2

tipo **reale** in C

specificazione di un valore del tipo

valori `double` positivi

`[+]ddddddddd.ddddddddd`

`[+]ddddddddd.dddddddddE[±]eee`

valori `double` negativi

`-ddddddddd.ddddddddd`

`-ddddddddd.dddddddddE[±]eee`

il numero delle cifre significative (`d`) è al più 16

il numero delle cifre dell'esponente (`e`) è al più 3

tipo **reale** in C

il comando C

```
sizeof(tipo)
```

restituisce il **numero di byte** necessario per la rappresentazione di un valore del tipo

```
sizeof(float)
sizeof(double)
sizeof(long double)
```

tipo **carattere** in C

`char`

un solo carattere dell'alfabeto esteso

codifica ASCII → 1 carattere 1 byte

alfabeto esteso: 128 caratteri

000	001	002	003	004	005	006	007	008
009	010	011	012	013	014	015	016	017
018	019	020	021	022	023	024	025	026
027	028	029	030	031	032	033 !	034 "	035 #
036 $	037 %	038 &	039 '	040 (	041)	042 *	043 +	044 ,
045 -	046 .	047 /	048 0	049 1	050 2	051 3	052 4	053 5
054 6	055 7	056 8	057 9	058 :	059 ;	060 <	061 =	062 >
063 ?	064 @	065 A	066 B	067 C	068 D	069 E	070 F	071 G
072 H	073 I	074 J	075 K	076 L	077 M	078 N	079 O	080 P
081 Q	082 R	083 S	084 T	085 U	086 V	087 W	088 X	089 Y
090 Z	091 [	092 \	093]	094 ^	095 _	096 `	097 a	098 b
099 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k
108 l	109 m	110 n	111 o	112 p	113 q	114 r	115 s	116 t
117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }
126 ~	127							

tabella ASCII

Il codice ASCII

Il codice ASCII (acronimo di *American Standard Code for Information Interchange*) è un **sistema di codifica a 7 bit**, **capace di definire 128 caratteri**. Inizialmente veniva utilizzato nei calcolatori ma, con il passare del tempo, è stato anche implementato anche nei sistemi operativi dei computer e di altri dispositivi. È proprio grazie a questo **sistema di codifica di caratteri e simboli** (e alle sue successive evoluzioni) se oggi possiamo scrivere e-mail, testi…

Come funziona il codice ASCII

- Caratteri di comando (da 0 a 31, 127): si tratta di caratteri non stampabili e servono per inviare comandi al PC. Un esempio è il comando per spostare il cursore uno spazio indietro. Il codice a 7 bit in questo caso è 00001000 e lo stesso comando si può inviare tenendo premuto il tasto ALT della tastiera e premendo in successione sul tastierino numerico i numeri 0 e 8. Attualmente non utilizziamo il codice binario o la combinazione di tasti indicata, ci basta premere semplicemente il tasto backspace.

- Caratteri speciali (da 32 a 47, da 58 a 64, da 91 a 96 e da 123 a 126): sono caratteri speciali stampabili che non corrispondono a numeri o lettere. Si tratta dei segni di punteggiatura. In questo gruppo rientra anche lo spazio che, sebbene non sia visibile, è stampabile: Questo è il motivo per cui lo spazio non rientra nel gruppo dei caratteri da comando.

- Cifre (da 48 a 57): comprende le 10 cifre decimali da 0 a 9.

- Lettere: da 65 a 90 per lettere maiuscole e da 97 a 122 per le lettere minuscole.

000	001	002	003	004	005	006	007	008
009	010	011	012	013	014	015	016	017
018	019	020	021	022	023	024	025	026
027	028	029	030	031	032	033 !	034 "	035 #
036 $	037 %	038 &	039 '	040 (	041)	042 *	043 +	044 ,
045 -	046 .	047 /	048 0	049 1	050 2	051 3	052 4	053 5
054 6	055 7	056 8	057 9	058 :	059 ;	060 <	061 =	062 >
063 ?	064 @	065 A	066 B	067 C	068 D	069 E	070 F	071 G
072 H	073 I	074 J	075 K	076 L	077 M	078 N	079 O	080 P
081 Q	082 R	083 S	084 T	085 U	086 V	087 W	088 X	089 Y
090 Z	091 [	092 \	093]	094 ^	095 _	096 `	097 a	098 b
099 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k
108 l	109 m	110 n	111 o	112 p	113 q	114 r	115 s	116 t
117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }
126 ~	127							

tabella ASCII

tipo **carattere** in C

specificazione di un valore del tipo

un solo carattere dell'**alfabeto esteso**

```
'k'
```

l'alfabeto esteso contiene caratteri minuscoli, maiuscoli, simboli speciali, etc…

```
'K'    'a'    'A'
'5'    ' '    '%'
'\''   '\n'   '\\'
```

tipi **scalari** in C:
spazio di memoria (in byte) per la rappresentazione dei valori

	`sizeof`
`int`	4 byte
`float`	4 byte
`double`	8 byte
`char`	1 byte

pc classici, celle di 32 bit

il tipo **logico** in C **non esiste**

in sostituzione, si usa il tipo `int`, con la convenzione che

falso ⟺ 0

vero ⟺ 1

qualunque valore **non nullo** è interpretato come **vero**

Dove vanno le variabili in C?

una variabile C è caratterizzata dal

- **nome** (identificatore)
- **valore** associato
- **tipo**
- **indirizzo** della cella di memoria a partire dal quale è memorizzato il valore

Le variabili in C

una variabile C è caratterizzata dal

- **nome** (identificatore)
- **valore** associato
- **tipo**
- **indirizzo** della cella di memoria a partire dal quale è memorizzato il valore

attenzione!

nome, tipo e indirizzo non possono essere modificati

Le variabili in C

- a ogni variabile è associata una **cella di memoria** o più celle consecutive, a seconda del suo tipo
- l'**indirizzo** di una variabile è quello della prima cella

variabili in C

metafora della scatola etichettata in uno scaffale

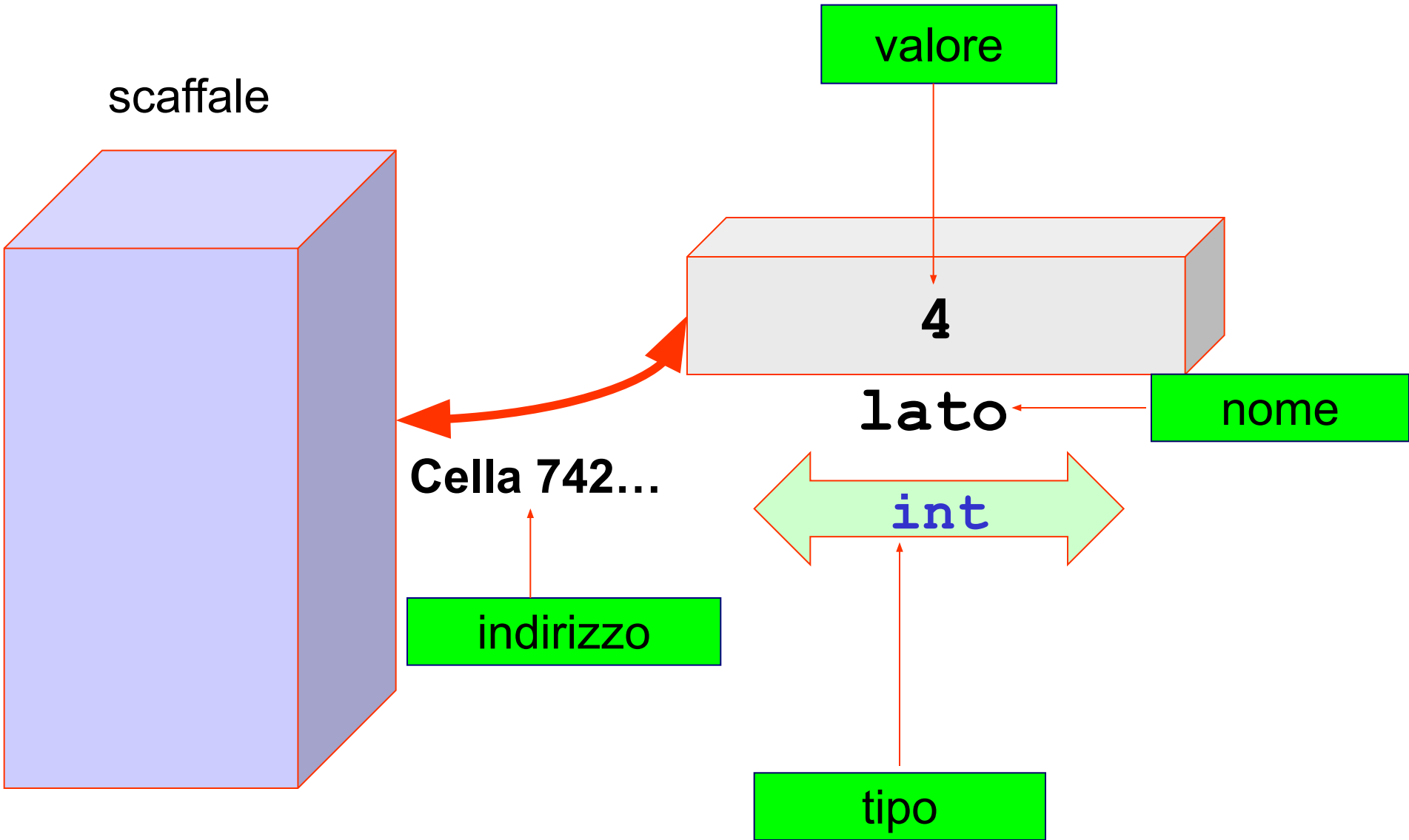
nome ⇨ etichetta

valore ⇨ contenuto della scatola

tipo ⇨ contenuti (e capienza) possibili della scatola

indirizzo ⇨ posizione della scatola nello scaffale

variabili in C



dichiarazione di variabili in C

```
<tipo> <variabili>;
```

Esempio:

```
int eta_anni;
float raggio, circonferenza;
double lato;
char lettera_alfabeto,simbolo;
```

dichiarazione di variabili in C

```
<tipo> <variabili>;
```

Esempio:

```
unsigned int eta;
long double velocita_luce;
short indice_riga;
long fattoriale;
int p;
```

costanti in C

una costante è un'associazione non modificabile che associa in modo permanente un valore a un identificatore

```
const float pi_greco = 3.1415926F;
const double pi = 3.14159265258416;
const int n = 100;
```

assegnazione in C

```
<variabile> = <espressione>;
```

simbolo di assegnazione

Esempio:

```
int eta_anni,p;
float raggio;
double lato;
char lettera_alfabeto;
```

```
eta_anni = 27;
raggio = 59.4F;
lato = 3.12;
lettera_alfabeto = 'g';
p = 1;
```

assegnazione in C

```
<variabile> = <espressione>;
```

```
var cognome: string
cognome := "Rossi"
```

```
char cognome;
cognome = 'Rossi';
cognome = "Rossi";
```

le stringhe in C sono considerate **valori strutturati**
(verranno trattate nell'ambito delle **strutture dati**)

dichiarazione/inizializzazione in C

```
<tipo> <variabile> = <valore>;
```

Esempio:

```
int eta_anni = 27;
float raggio = 59.4F;
double lato = 3.12;
char lettera_alfabeto = 'g';
int p = 1;
```

assegnazione in C

```
<variabile> = <espressione>;
```

operatori aritmetici `+,-,*,/,%`

Esempio:

```
eta_anni = (27-7)/2;
eta_anni = (28-7)/2;
eta_anni = 27-7/2;
raggio = 2.0F*15.4F;
lato = (3.0-0.1)/(3.-1E-1);
eta_anni = 27%5
```

```
10
10
24
30.8
1.0
2
```

Aritmetica in C

Operazioni in C	**Operatore aritmetico**
addizione	+
sottrazione	-
Moltiplicazione	*
Divisione	\
Modulo	%

il **modulo** (tra un **dividendo di tipo intero** e un **divisore di tipo intero**) fornisce come risultato il **resto** della **divisione intera**

Aritmetica in C

Gli operatori aritmetici sono tutti operatori binari

La divisione tra interi restituisce un intero.
(Viene troncata la parte decimale)

Un tentativo di divisione per 0 è un errore fatale: provoca la terminazione immediata del programma

Regole generali per la priorità degli operatori artimetici

Operatori in C

Operatori	Operazione	Priorità
( )	Parentesi	Sono valutate per prime. Nel caso che le parentesi siano nidificate, saranno valutate prima le espressioni della coppia più interna. Altrimenti da sinistra verso destra.
* / %	Moltiplicazione Divisione Modulo	Sono valutate per seconde. Nel caso che ce ne siano molte, saranno valutate da sinistra a destra
+ -	Addizione Sottrazione	Sono valutate per ultime. Nel caso che ce ne siano molte, saranno valutate da sinistra a destra

operatori aritmetici in C

`+`	addizione (per tutti i tipi)
`-`	sottrazione (per tutti i tipi)
`*`	moltiplicazione (per tutti i tipi)
`/`	divisione (`float`, `double`, `long double`)
`/`	divisione intera (per i tipi interi)
`%`	modulo (per i tipi interi), resto divisione intera

la **divisione intera** (tra un **dividendo di tipo intero** e un **divisore di tipo intero**) fornisce come risultato la **parte intera del quoziente**

`(28-7)/2`	`10`
`1/2`	`0`
`1.0/2.0`	`0.5`

operatori aritmetici in C

`+`	addizione (per tutti i tipi)
`-`	sottrazione (per tutti i tipi)
`*`	moltiplicazione (per tutti i tipi)
`/`	divisione (`float`, `double`, `long double`)
`/`	divisione intera (per i tipi interi)
`%`	modulo (per i tipi interi), resto divisione intera

il **modulo** (tra un **dividendo di tipo intero** e un **divisore di tipo intero**) fornisce come risultato il **resto** della **divisione intera**

`23%7`	`2`	`7%7`	`0`	`1%2`	`1`	`10%2`	`0`	`11%2`	`1`

operatori aritmetici in C

regole di precedenza:

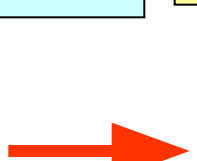
moltiplicazione, divisione e modulo vengono eseguite **prima** di addizione e sottrazione

```
27-8/2
```
23

```
6.0/3.0-1.0
```
1.0

```
16%3+2
```
3

$$\frac{3 \cdot 7 - 1}{4 \cdot 2 + 2}$$



~~`3*7-1/4*2+2`~~

operatori aritmetici in C

regole di precedenza:

per superare le regole di precedenza è necessario usare le parentesi (tonde)

$$\frac{3 \cdot 7 - 1}{4 \cdot 2 + 2}$$

```
(3*7-1)/(4*2+2)
```

operatori aritmetici in C

regole di precedenza:

se gli operatori hanno la stessa precedenza, allora l'espressione viene valutata da **sinistra verso destra**

```
59*100/60
```

```
98
```

valore `int`

```
59.0*100.0/60.0
```

```
98.3333333333333
```

valore `double`

```
59.0F*100.0F/60.0F
```

```
98.333333
```

valore `float`

programma C per il calcolo della **circonferenza** di un cerchio, fissato il suo raggio

commento

```
#include <stdio.h>
/* calcolo circonferenza di un cerchio */
void main ()
{
 const float pi_greco = 3.1415926F;
 float raggio, circon;

 raggio = 2.0F;
 circon = 2.0F * pi_greco * raggio;
 printf ("circonferenza=%f\n",circon);
}
```

visualizzazione su schermo

```
circonferenza=12.566370
Press any key to continue_
```

Calcolo della circonferenza

```
#include <stdio.h>
/* calcolo circonferenza di un cerchio */
/* versione doppia precisione */

int main(){

        const double pi_greco = 3.14159265258416;
        double raggio, circon;

        raggio = 2.0; /* il Raggio e' fissato */
        circon = 2.0 * pi_greco * raggio;
        printf ("circonferenza=%23.14lf\n", circon);
}
```

Calcolo della circonferenza

```
#include <stdio.h>
/* calcolo circonferenza di un cerchio */
/* versione doppia precisione */

int main(){
```

parte **dichiarativa**

```
    const double pi_greco = 3.14159265258416;
    double raggio, circon;

    raggio = 2.0; /* il Raggio e' fissato */
    circon = 2.0 * pi_greco * raggio;
    printf ("circonferenza=%23.14lf\n", circon);
}
```

parte **esecutiva**

Esercizi per casa

1. Scrivere un programma che chieda in input due numeri e visualizzi la loro somma, prodotto, differenza, divisione e modulo.

2. Scrivere un programma che prenda in input dalla tastiera 3 numeri diversi, quindi visualizzi la loro somma, la media e il prodotto.

3. Scrivere un programma che visualizzi a schermo un tiangolo di * , cosi:
   *
   **
   ***
   ****

4. Scrivere un programma che legga il raggio di un cerchio e visualizzi il diametro, la circonferenza e l'area dello stesso. Usate il valore costante 3,14159 per pi. All'interno dell'istruzione printf usare la specifica di conversione %f per i numeri con la virgola.