Programmazione
Il linguaggio C

Liceo G.B. Brocchi - Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

Cosa significa *programmare*

- **Programmare** significa *fornire istruzioni ad un computer per fargli portare a termine un compito determinato*

- L'essere umano scrive programmi per i computer dagli anni '40, utilizzando dei linguaggi particolari, detti **linguaggi di programmazione**

- I linguaggi di programmazione, a differenza dei *linguaggi naturali* (come l'inglese o l'italiano), sono dei linguaggi **con una sintassi molto rigida e non ambigui**, proprio perché devono essere *compresi* da una macchina, che non possiede il *buon senso umano*

Diversi linguaggi di programmazione

- Abbiamo già visto un linguaggio di programmazione, il **linguaggio macchina/assembly** (linguaggio a basso livello)
- In realtà ogni microprocessore ha il suo set di istruzioni, e quindi il proprio linguaggio macchina

- È estremamente scomodo scrivere programmi nel linguaggio macchina/assembly del processore

- Per questo, già a partire dagli anni '50, sono stati inventati i cosiddetti **linguaggi ad alto livello** (i più antichi: Fortran, COBOL, ALGOL… i più moderni: C++, Java, Scala, Python, PHP, Haskell)

Compilatori

- I linguaggi di programmazione sono formalismi (i.e. delle lingue artificiali e matematiche) utilizzati per descrivere procedimenti di calcolo/algoritmi
- Il codice scritto nei linguaggi di programmazione utilizzati comunemente deve essere tradotto in una forma comprensibile da un calcolatore elettronico
- Per questo passaggio di traduzione servono programmi detti **compilatori** e **interpreti**

- La traduzione da codice assembly a codice binario, invece, è effettuata da programmi molto più semplici detti **assemblatori**

Compilatori

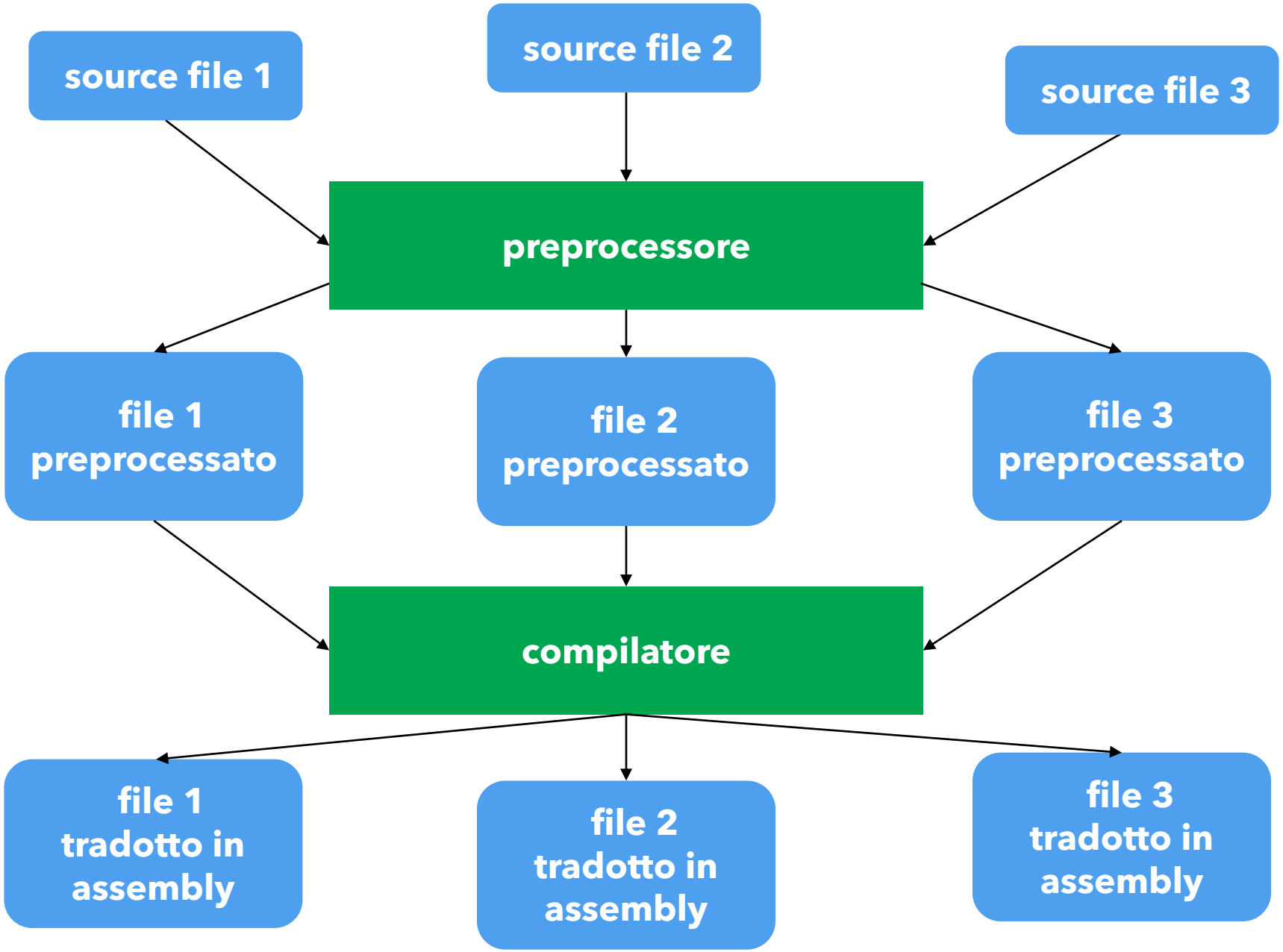
- Un compilatore è un programma (*software*) elaboratore di linguaggi che legge un programma scritto in un certo linguaggio di programmazione (*linguaggio sorgente*) e lo traduce in un altro linguaggio, diverso, detto *linguaggio destinazione*. Se il codice destinazione è codice macchina per una determinata CPU, allora questo codice è un programma eseguibile da un computer e può essere *eseguito/lanciato/runnato*. **I linguaggi C e C++ sono linguaggi compilati**
- I programmatori (*sviluppatori software*) normalmente commettono diversi errori di sintassi quando programmano: compito del compilatore è segnalare gli errori che incontra durante il processo di traduzione

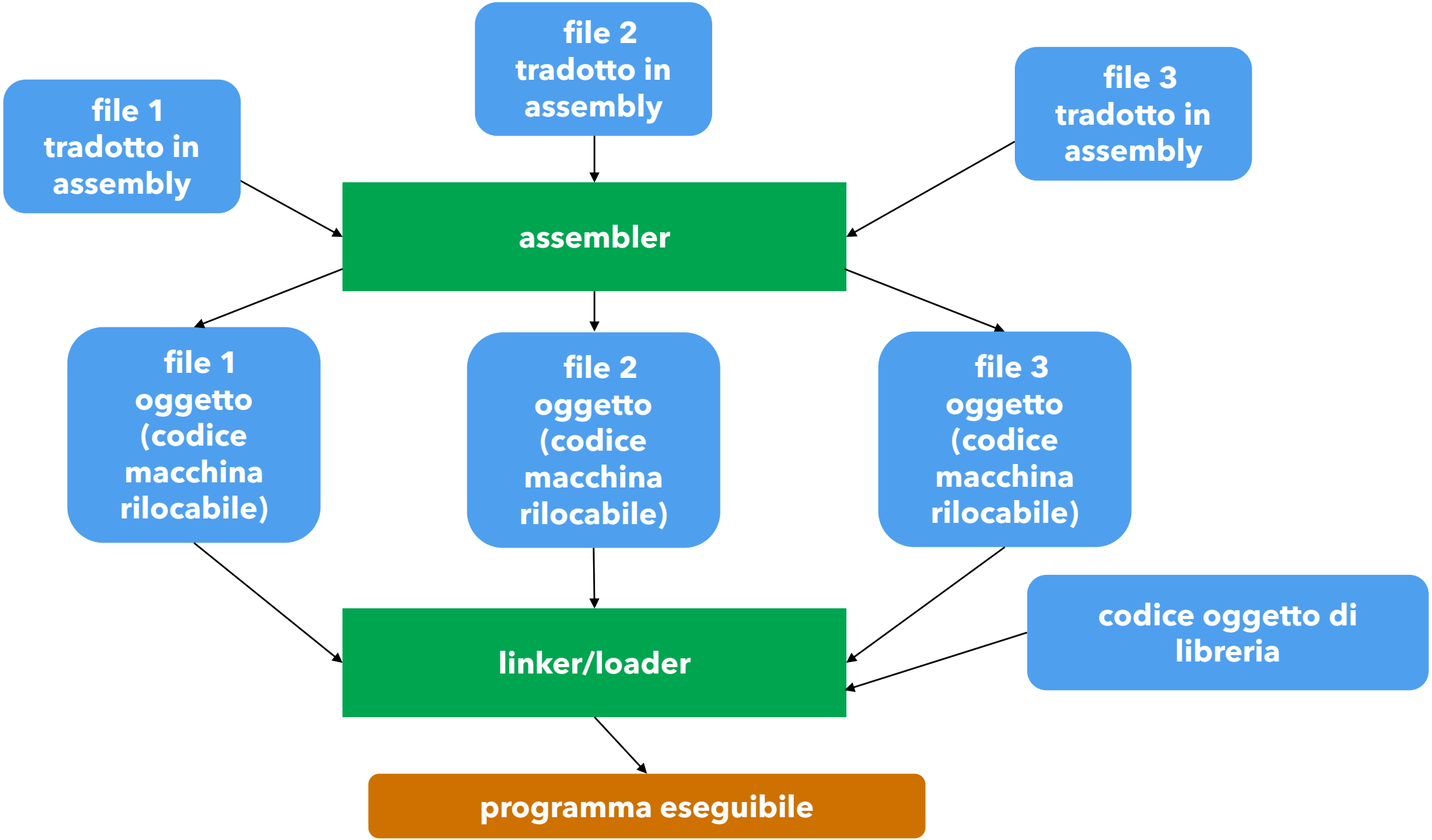
Interpreti

- Un interprete, a differenza di un compilatore, non produce un programma destinazione, ma esegue direttamente le istruzioni descritte nel programma sorgente. Esempi di linguaggi interpretati sono: Python, Perl, PHP, il linguaggio di scripting Bash etc…
- **NB**: La *shell* che utilizzate come interfaccia del sistema operativo è un interprete
- Generalmente, un programma compilato in linguaggio macchina è molto più efficiente di un programma sorgente interpretato

Compilatori

- Vedremo che un programma sorgente di un programma realistico viene spesso suddiviso in moduli memorizzati in *file* diversi
- Il **preprocessore** si occupa di effettuare una prima elaborazione dei file sorgenti
- L'output del preprocessore costituisce l'input del compilatore. Per ogni file sorgente preprocessato, il compilatore produce generalmente un file in linguaggio assembly
- I programmi assembly vengono dati in pasto all'assembler, che si occupa di tradurre l'assembly in codice macchina. I file in codice macchina ottenuti a questo punto vengono detti **file oggetto contenenti codice rilocabile**
- **NB**: **i singoli file oggetto non sono ancora programmi eseguibili!**
- Un programma chiamato **linker** si occupa di collegare i diversi file oggetto in un unico file contenente codice macchina eseguibile
- Infine, il **loader** si occupa di caricare in memoria RAM il file eseguibile, rendendo il programma pronto per l'esecuzione


source file 1
source file 2
source file 3
preprocessore
file 1 preprocessato
file 2 preprocessato
file 3 preprocessato
compilatore
file 1 tradotto in assembly
file 2 tradotto in assembly
file 3 tradotto in assembly


file 1 tradotto in assembly
file 2 tradotto in assembly
file 3 tradotto in assembly
assembler
file 1 oggetto (codice macchina rilocabile)
file 2 oggetto (codice macchina rilocabile)
file 3 oggetto (codice macchina rilocabile)
linker/loader
codice oggetto di libreria
programma eseguibile

Il linguaggio C

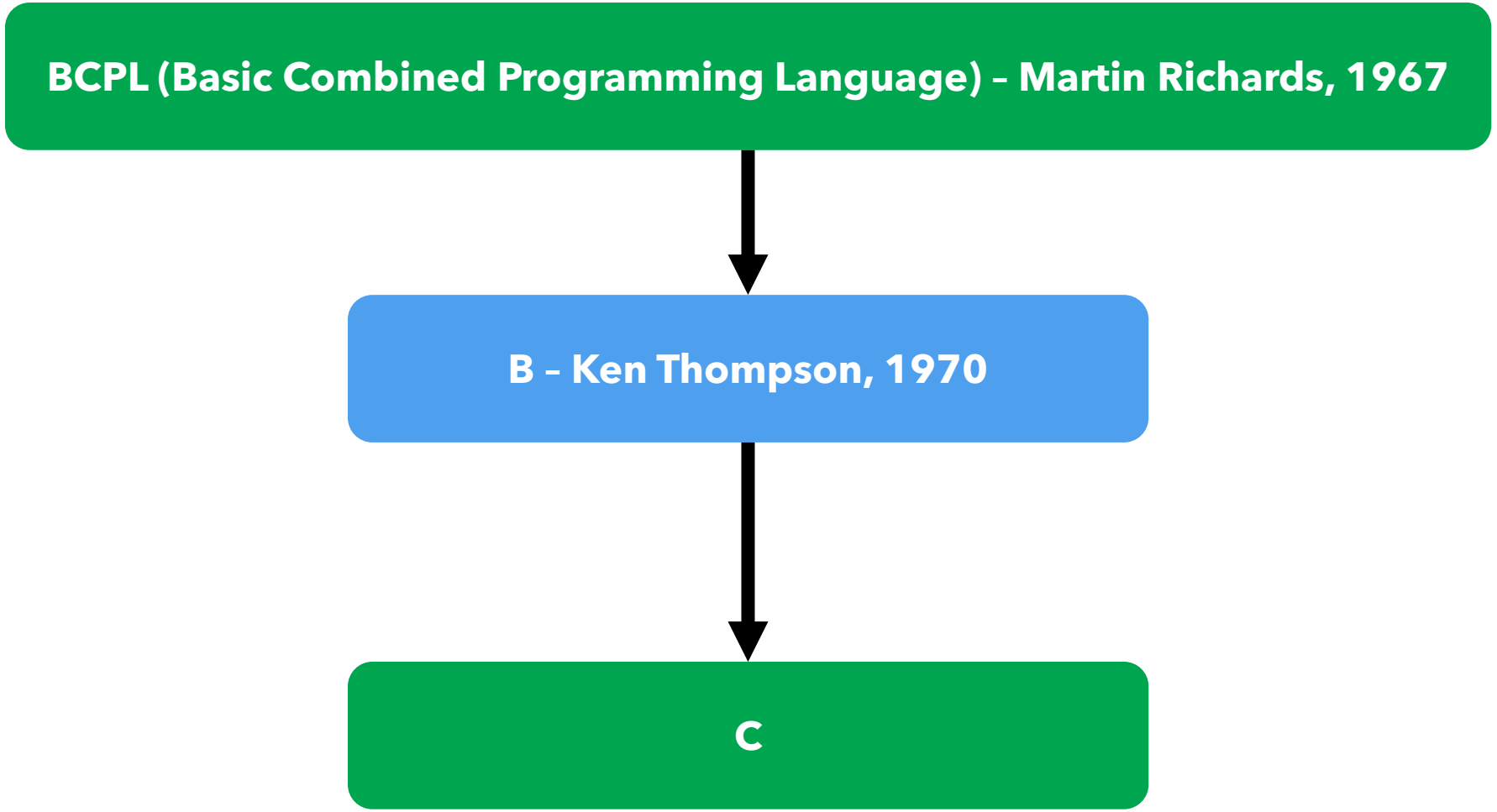
- Il **linguaggio C** è un linguaggio di programmazione ad alto livello progettato ed implementato da Dennis Ritchie negli anni '70, per il sistema operativo **UNIX**, sulla macchina **DEC PDP-11**

- Il C è un linguaggio compilato ad alto livello, *imperativo* e *procedurale*, e permette di programmare in modo strutturato

- Non è pensato per applicazioni specifiche: è *general-purpose*, ossia può essere utilizzato per sviluppare qualsiasi tipo di applicazione

- Viene utilizzato in particolare per la programmazione di sistema, ossia per lo sviluppo di sistemi operativi e altri software di base

Un po’ di storia



Iniziamo a programmare!

- Iniziamo da un grande classico:
  - `scriviamo un programma che stampa sullo schermo la stringa «hello, world» e termina`

- Per scrivere un programma dobbiamo utilizzare:
  - un linguaggio di programmazione: il nostro linguaggio sarà il C
  - un *text editor*
  - il linguaggio C è ad alto livello e compilato, quindi non comprensibile direttamente dalla CPU, per cui ci serve un software che traduca i nostri programmi in linguaggio macchina: questo software si chiama **compilatore**

Assembly, che spavento

```
        global    _start

        section   .text
_start: mov       rax, 1
        mov       rdi, 1
        mov       rsi, message
        mov       rdx, 13
        syscall
        mov       rax, 60
        xor       rdi, rdi
        syscall

        section   .data
message: db       "hello, world", 10
```

In assembly si lavora direttamente con i registri del processore (`rax`, `rdi`, `rdx` etc...) e con gli indirizzi di memoria

È estremamente difficile programmare e leggere codice scritto in assembly. Al giorno d'oggi si utilizza solo in contesti molto particolari. Il codice cambia per ogni processore

Ovviamente il nostro linguaggio principale non sarà l'assembly. Comunque cercheremo di capirne i principi di funzionamento

Assembly, che spavento

```
        global    _start

        section   .text
_start: mov       rax, 1
        mov       rdi, 1
        mov       rsi, message
        mov       rdx, 13
        syscall
        mov       rax, 60
        xor       rdi, rdi
        syscall

        section   .data
message: db       "hello, world", 10
```

per eseguire questo programma (salvato come *hello.asm*) bisogna prima *assemblarlo in linguaggio macchina* e *linkarlo*, con questi comandi:

```
nasm -felf64 hello.asm && ld hello.o
```

Poi si può eseguire con:

```
./a.out
```

Iniziamo a programmare!

```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("hello, world\n");

    return 0;
}
```

questo è il codice sorgente (*source code*) del nostro primo programma in linguaggio C

va scritto su un file di testo creato con un *text editor* (su Linux: *emacs*, *nano*, *gedit*, *vim*, *emacs* su Windows: *Notepad++*, *Sublime*)

più avanti potrete utilizzare anche strumenti più avanzati (*IDE*), come *Geany*

lanciare il comando:

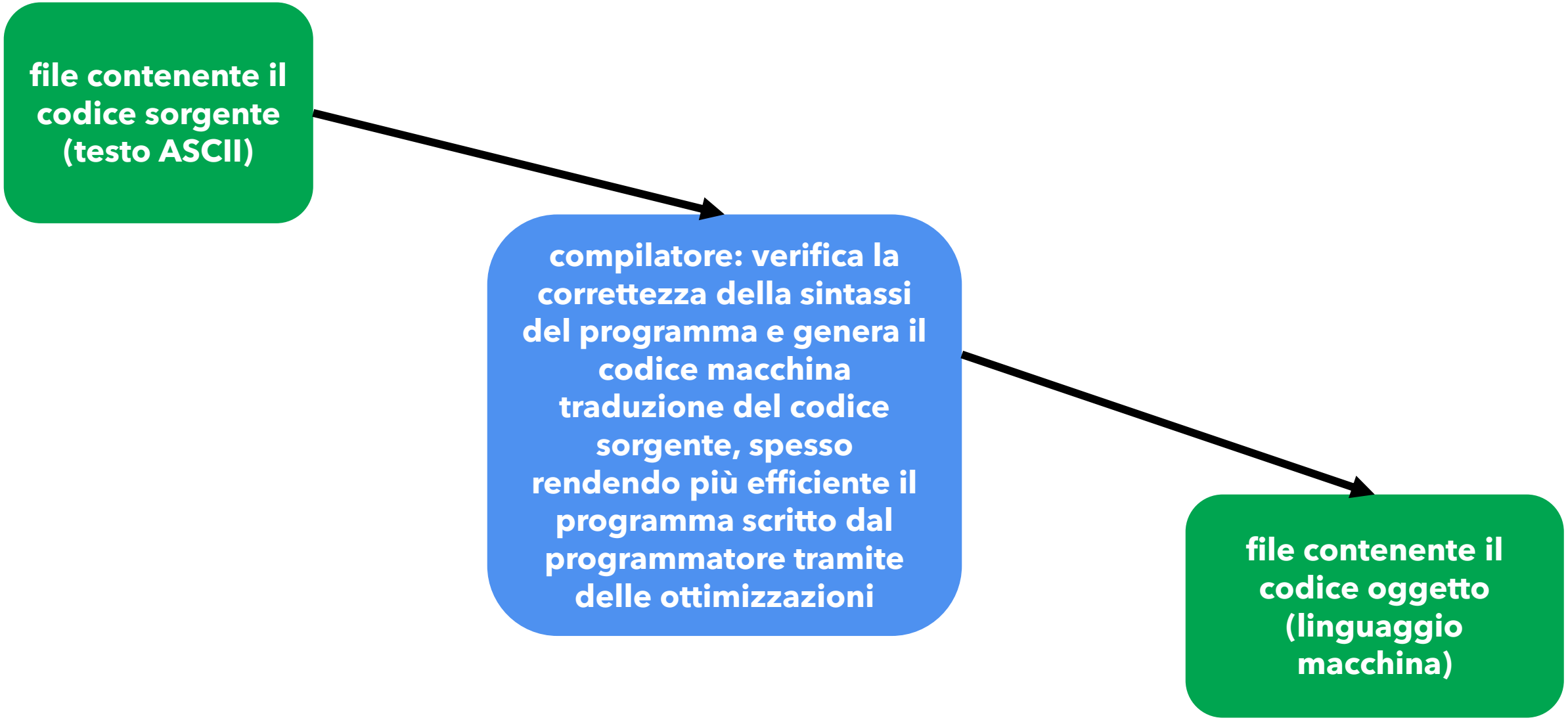
```
gedit first_program.c &
```

oppure

```
emacs first_program.c &
```

viene creato un file di testo con nome `first_program.c`

Il compilatore



Il compilatore `gcc`

- Utilizzeremo il compilatore **`gcc`** (*GNU Compiler Collection*). Per noi gcc è il compilatore del C di GNU/Linux. In realtà è una raccolta di compilatori per diversi linguaggi

- Il comando da shell per compilare il programma `first_program.c` è:

```
gcc first_program.c
```

- L'output prodotto da gcc è questo file:

```
a.out
```

- `a.out` (**a**ssembler **out**put) è un file eseguibile (su Linux un file **ELF**, su Windows un **.exe**) nel linguaggio macchina del vostro processore, ossia un vero e proprio programma che potete mandare in esecuzione dalla shell, così:

```
./a.out
```

Il compilatore `gcc`

- Possiamo dare all'eseguibile un nome migliore di `a.out` (nome di default)

- Il comando da shell per compilare il programma `first_program.c` è:

```
gcc -o first_program first_program.c
```

- L'output prodotto da `gcc` è questo file:

```
first_program
```

- `first_program` è un file eseguibile nel linguaggio macchina del vostro processore, ossia un vero e proprio programma che potete mandare in esecuzione dalla shell, così:

```
./first_program
```

- Sullo schermo (*standard output*) comparirà la stringa `hello, world` seguita da un carattere di caporiga (*newline*)

Struttura di un programma C

```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("hello, world\n");

    return 0;
}
```

direttiva per il preprocessore
il preprocessore si occuperà di includere l'*header* `stdio.h` della libreria standard per l'input/output

`main` è una funzione che restituisce un intero e riceve 2 argomenti
l'esecuzione del programma inizia dalla funzione `main`. Un programma, per essere eseguito, deve avere la funzione `main` da qualche parte

il `main` è composto da istruzioni, o *statement*
la funzione `printf` della libreria standard del C permette di stampare su *standard output*, ossia sullo schermo. L'istruzione `return` restituisce il controllo alla *shell* da cui era stato lanciato il programma. Tutte le istruzioni terminano con `;` (*semicolon*)

le parentesi graffe (*curly brackets*) definiscono un blocco

Struttura di un programma C

```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("hello, world\n");

    return 0;
}
```

argomenti della funzione `main`, per ora non considerateli

NB: le funzioni di libreria come `printf` sono state sviluppate da programmatori molto esperti. Fanno cose difficili/noiose da riscrivere. La libreria standard C su GNU/Linux si chiama `glibc`

costante stringa di caratteri (*string literal*). Le costanti stringhe vanno comprese tra apici doppi (*double quotes*)

`stdio.h` sta per *standard input-output header* e contiene diverse «cose» difficili da capire per ora, ma necessarie per effettuare I/O. Stampare una stringa sullo schermo è un'operazione di input/output. Se volete vedere il contenuto di `stdio.h`, lanciate `emacs /usr/include/stdio.h &`

Struttura di un programma C

il codice all'interno di un blocco va **indentato**
indentazione: all'apertura di un nuovo blocco, il codice va spostato a destra di
un certo numero di spazi (generalmente 2 o 4)

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("hello, world\n");

    return 0;
}
```

una parentesi graffa (in inglese *curly bracket*) aperta determina l'apertura di un blocco,
una parentesi graffa chiusa ne determina la chiusura.
Le parentesi devono essere bilanciate: ad ogni apertura deve corrispondere una chiusura:
l'annidamento dei blocchi è potenzialmente infinito, esempio: `{{{{{}}}}}`

Struttura di un programma C

indentazione: dopo la chiusura di un blocco, si torna al livello di indentazione del blocco «genitore»

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("hello, world\n");

    return 0;
}
```

Struttura di un programma C

"hello, world\n"

è una stringa composta dai seguenti caratteri (i caratteri singoli vanno racchiusi tra apici singoli):

```
'h'
'e'
'l'
'l'
'l'
'o'
','
' '
'w'
'o'
'r'
'l'
'd'
'\n'
```

come potete notare, l'ultimo carattere è un po' particolare. Sicuramente avete visto che in output non viene stampato \n

Però se provate a toglierlo, la stringa «hello world» compare «attaccata» all'intestazione della shell

'\n' è un carattere speciale, detto *newline character*, che fa avanzare l'output al margine sinistro della riga successiva

Lo stesso programma scritto diversamente

```c
#include <stdio.h>

int main(int argc, char *argv[]){
  putchar('h');
  putchar('e');
  putchar('l');
  putchar('l');
  putchar('o');
  putchar(',');
  putchar(' ');
  putchar('w');
  putchar('o');
  putchar('r');
  putchar('l');
  putchar('d');
  putchar('\n');

  return 0;
}
```

la funzione `putchar`, definita sempre nella libreria standard, permette di stampare su standard output un singolo carattere. Notare che i caratteri singoli vengono racchiusi tra apici singoli!

Interagire con il compilatore

```
#include <stdio.h>

int main(int argc, char *argv[])
}
```

Il compilatore segnala diversi tipi di errore, tra cui gli errori di sintassi. Proviamo a scrivere qualche programma sbagliato e chiediamo al compilatore cosa ne pensa

```
cyofanni@LAPTOP-I0S1RKRC:~/Desktop/high-school-cs-class/c_lectures$ gcc -o hello_world_putchar hello_world_putchar.c
hello_world_putchar.c: In function ‘main’:
hello_world_putchar.c:4:1: error: expected declaration specifiers before ‘}’ token
    4 | }
      | ^
hello_world_putchar.c:5: error: expected ‘{’ at end of input
cyofanni@LAPTOP-I0S1RKRC:~/Desktop/high-school-cs-class/c_lectures$
```

Interagire con il compilatore

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  }
```

Ho aggiunto i numeri di riga per chiarezza.
I messaggi del compilatore non sono sempre simpatici… vi sta segnalando che manca una parentesi graffa da qualche parte. Trovate l'errore.

```
cyofanni@LAPTOP-I0S1RKRC:~/Desktop/high-school-cs-class/c_lectures$ gcc -o hello_world_putchar hello_world_putchar.c
hello_world_putchar.c: In function ‘main’:
hello_world_putchar.c:4:1: error: expected declaration specifiers before ‘}’ token
    4 | }
      | ^
hello_world_putchar.c:5: error: expected ‘{’ at end of input
cyofanni@LAPTOP-I0S1RKRC:~/Desktop/high-school-cs-class/c_lectures$
```

Interagire con il compilatore

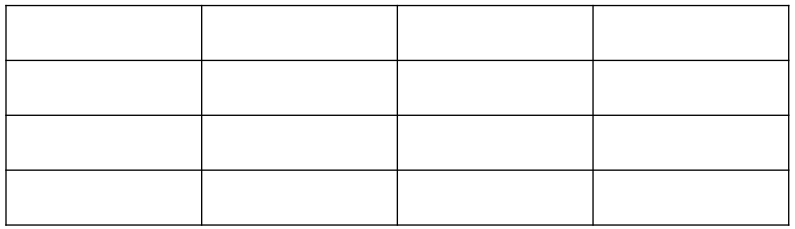
```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4      printf("hello world\n")
5  }
```

qui il messaggio è più chiaro: manca il punto e virgola terminatore dell'istruzione

```
hello_world_putchar.c:5: error: expected '{' at end of input
cyofanni@LAPTOP-I0S1RKRC:~/Desktop/high-school-cs-class/c_lectures$ gcc -o hello_world_putchar hello_world_putchar.c
hello_world_putchar.c: In function 'main':
hello_world_putchar.c:4:30: error: expected ';' before '}' token
    4 |         printf("hello world")
      |                              ^
      |                              ;
    5 | }
      | ~
```

Le variabili

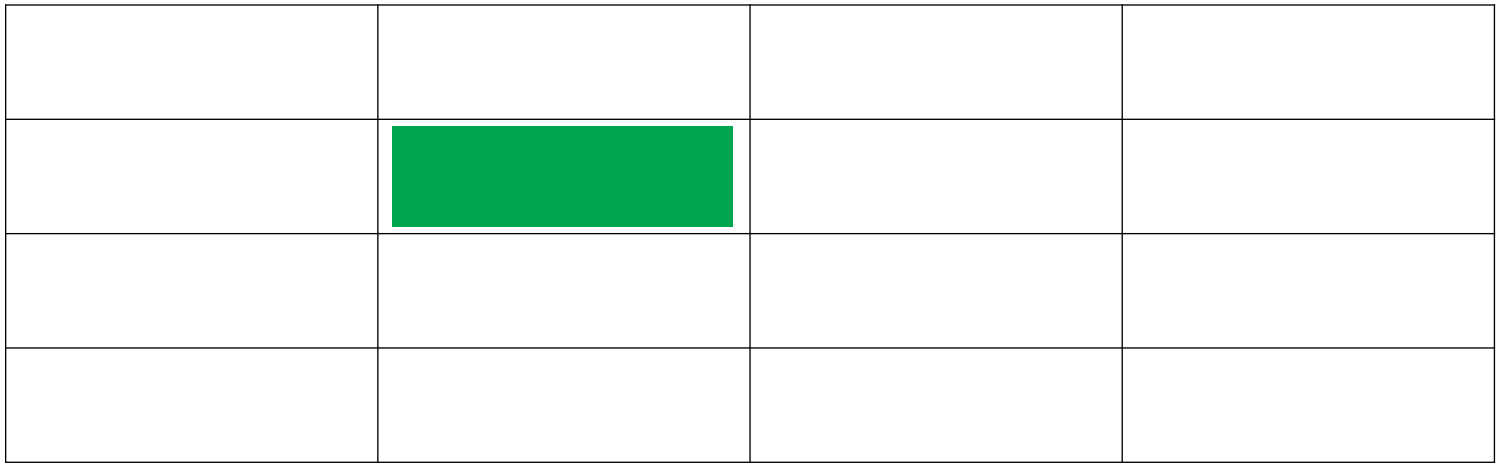
- Nei linguaggi di programmazione ad alto livello, **una variabile è l'identificativo (il nome) di un'area della memoria di lavoro di una certa dimensione**
- A basso livello, la memoria è composta da celle localizzabili tramite una posizione, chiamata indirizzo. In molte architetture, la dimensione delle celle è 1 byte
- Ora che programmiamo ad alto livello, daremo dei nomi umanamente comprensibili a queste celle di memoria. La figura seguente rappresenta astrattamente le celle della memoria di lavoro:



Le variabili

- I linguaggi ad alto livello come il C permettono di dare dei nomi alle celle di memoria. Se in un programma decidiamo di utilizzare l'identificativo **a** come variabile, il compilatore metterà a disposizione del programma una o più celle consecutive che nel programma verranno sempre identificate come **a**

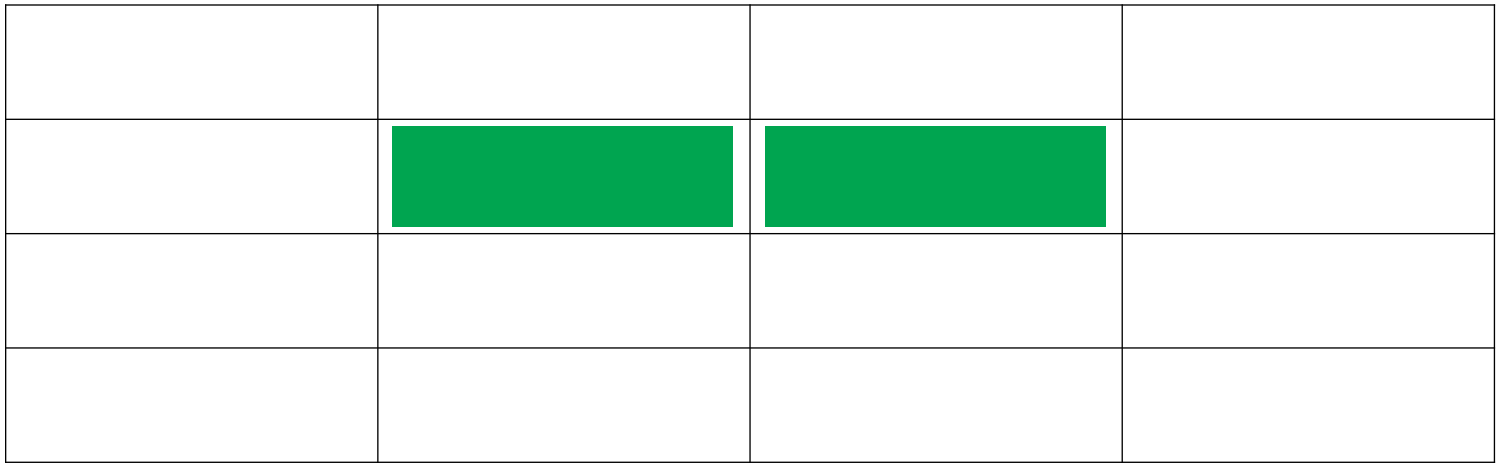
nel programma, il nome **a** corrisponderà alla cella colorata in verde



Le variabili

- Come vedremo, una variabile potrebbe occupare anche più celle di memoria consecutive
- Facciamo un esempio con una variabile **var** che occupa 2 celle:

nel programma, **var** corrisponderà alle celle colorate in verde

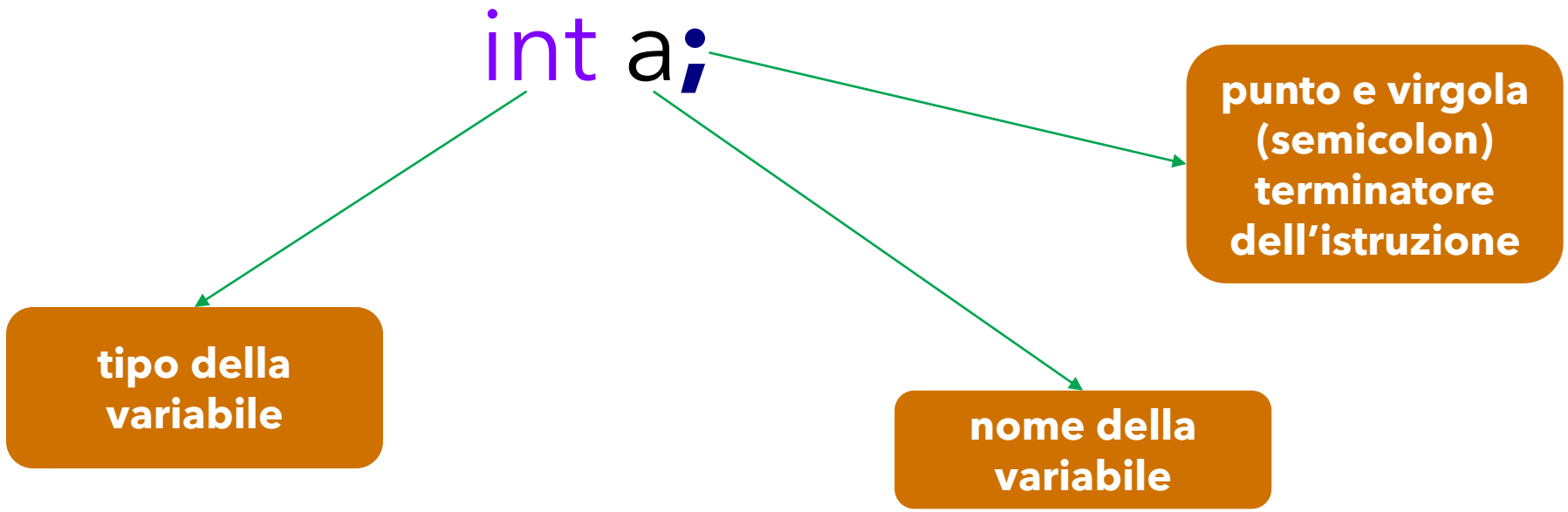


Le variabili

- Da cosa dipende il numero di celle occupato da una variabile?
- Dipende dal **tipo della variabile**
- Informalmente, **il tipo di una variabile è il significato che l'essere umano dà ai byte memorizzati nelle celle di memoria**
- I programmi operano su diverse tipologie di dati:
  - numeri interi senza segno (naturali)
  - numeri interi con segno
  - numeri reali (fino ad una certa precisione)
  - caratteri (stampabili e di controllo)
  - stringhe di caratteri
  - etc…
- Iniziamo a lavorare con i numeri interi utilizzando le variabili in C

Le variabili

- In C, le variabili vanno **dichiarate**. Per dichiarare una variabile **intera con segno** di nome **a**, l'istruzione da scrivere è la seguente (all'interno della funzione main):



Da vedere a casa

- "C" Programming Language: Brian Kernighan – Computerphile
- Why C is so Influential - Computerphile