Capitolo 2

Nozioni di base

Programma: Stampare un gioco di parole

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C: that is the question.\n");
  return 0;
}
```

- Possiamo memorizzare il programma nel file `pun.c`
- Il nome del file non ha importanza, ma l'estensione `.c` è richiesta

Compilare e linkare

- Per creare un programma eseguibile occorrono 3 operazioni:
  - ***Preprocessare.*** Il ***preprocessor*** serve per gestire comandi che iniziano con il carattere # (detti ***direttive***)
  - ***Compilare.*** Il ***compilatore*** traduce il programma in istruzioni macchina (***codice oggetto***).
  - ***Linking.*** Il ***linker*** mette insieme il codice oggetto prodotto dal compilatore con altro codice oggetto (librerie) per avere un programma completo ed eseguibile.
- Solitamente il preprocessore è parte del compilatore e spesso anche il linking viene fatto dal compilatore

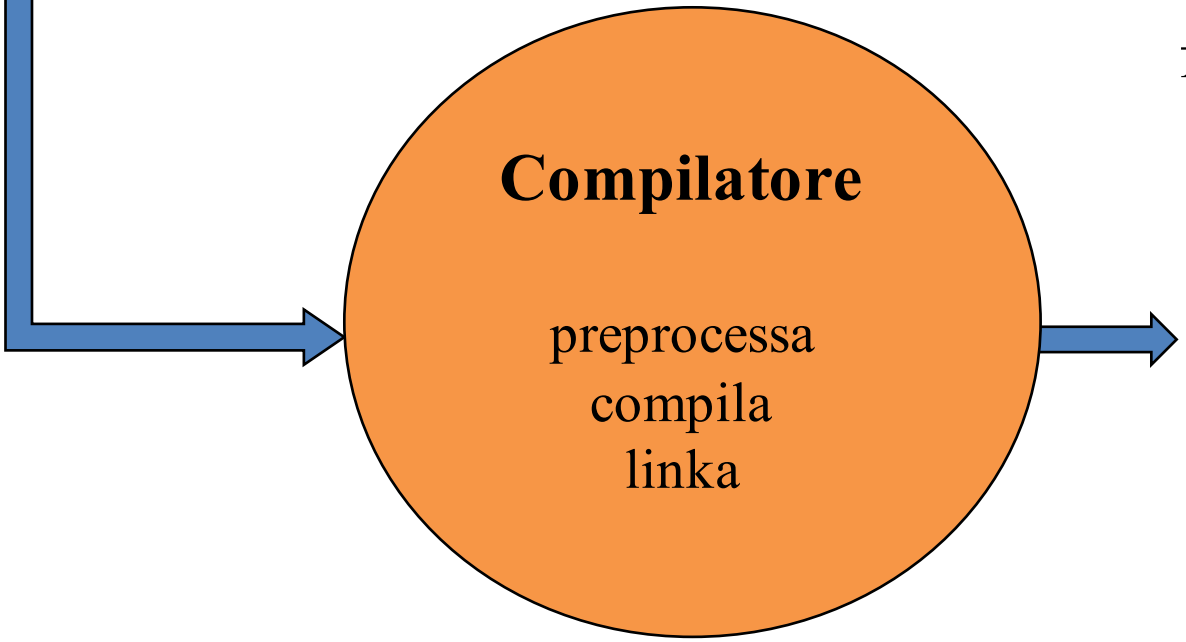
Programma eseguibile

file `pun.c`

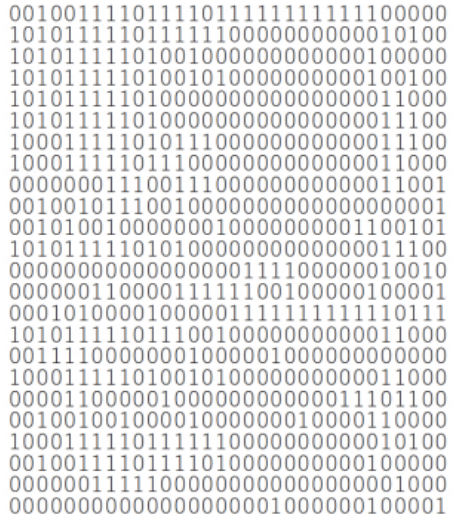
```
#include <stdio.h>

int main(void)
{
  printf("To C,…\n");
  return 0;
}
```

codice sorgente



file `pun`



codice eseguibile

Compilare e linkare con `cc`

- Per compilare (e linkare) il programma `pun.c` in un sistema UNIX, possiamo usare il seguente comando da una finestra shell:

```
% cc pun.c
```

(Il carattere `%` è detto *prompt*).

- Con il compilatore `cc` l'operazione di link viene realizzata automaticamente

Compilare e linkare con `cc`

- L'output del compilatore `cc` è un file eseguibile il cui nome (di default) è `a.out`

- L'opzione `-o` permette di specificare il nome del file di output

- Il seguente comando compila (ed esegue il link) e scrive l'output (versione eseguibile di `pun.c`) nel file `pun`:

```
% cc -o pun pun.c
```

Il compilatore GCC

- GCC è uno dei compilatori C più utilizzati
  - GNU C Compiler
  - GNU, GNU is Not Unix, Free Software Foundation

- GCC viene fornito insieme al sistema operativo Linux ma è disponibile anche per altri sistemi

- Si usa allo stesso modo di `cc`:

```
% gcc -o pun pun.c
```

Integrated Development Environment

- Un ***IDE (integrated development environment)*** è un pacchetto software che permette di editare i file sorgenti, compilare, linkare, eseguire e far il debug di un programma utilizzando un'unica interfaccia

- In questo corso non ne utilizzeremo
  - shell, editor (vi, emacs …) e compilatore gcc

Forma generale di un programma

- Forma generale di un programma C (semplice):

direttive

```
int main(void)
{
   istruzioni
}
```

Forma generale di un programma

- Il C usa le parentesi graffe `{` e `}` per delimitare un blocco di istruzioni
  - Altri linguaggi usano parole del tipo `begin` ed `end`

- Anche il più semplice dei programmi C è costruito su 3 elementi:
  - Direttive
  - Funzioni
  - Istruzioni

Direttive

- Prima della compilazione vera e propria
  - Un preprocessore modifica il testo del file sorgente
- Le *direttive* sono i comandi per il preprocessore

- Esempio:

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

- `<stdio.h>` è un ***header*** (file) che contiene informazioni sulle funzioni di libreria per l'I/O

Direttive

- Le direttive iniziano sempre con il carattere #
- Ogni direttiva deve essere messa su una singola linea
- Non ci sono punti e virgola né altri caratteri che indicano la fine

Funzioni

- ***Funzione***
  - Una sequenza di istruzioni raggruppate sotto un unico nome (il nome della funzione)

- ***Funzioni di libreria***
  - Fornite come parte del linguaggio C

- Una funzione che calcola un valore utilizza l'istruzione `return` per specificare qual è il valore "restituito" dalla funzione:

```
return x + 1;
```

La funzione `main`

- La funzione `main` è obbligatoria
- `main` è speciale
  - Viene eseguita automaticamente quando si esegue il programma
- `main` restituisce un codice di uscita
  - Normalmente il valore 0 per indicare il corretto funzionamento del programma
- Se non c'è l'istruzione `return` che termina l'esecuzione di `main`
  - Alcuni compilatori producono un messaggio di avvertimento

Istruzioni

- Un'***istruzione*** è un comando da eseguire
- `pun.c` utilizza solo due istruzioni
  - Una è l'istruzione `return`
  - L'altra è una ***chiamata di funzione.***
- "***Chiamare***" una funzione significa eseguire le istruzioni specificate nella definizione della funzione
- `pun.c` chiama la funzione `printf` per visualizzare una stringa di caratteri:

```
printf("To C, or not to C: that is the question.\n");
```

Istruzioni

- Ogni istruzione viene terminata con “;” (punto e virgola)
  - può occupare diverse linee
- Le direttive invece occupano un’intera linea
  - Terminano quando termina la linea

Stampare le stringhe

- La funzione `printf` stampa (a video) una ***stringa letterale*** —cioè caratteri racchiusi tra doppi apici—senza stampare i doppi apici
- `printf` non fa andare automaticamente alla linea successiva

- Per andare a capo dobbiamo dire alla `printf` di "andare", aggiungendo il carattere newline "`\n`"

Stampare le stringhe

- L'istruzione

```
printf("To C, or not to C: that is the question.\n");
```

potrebbe essere rimpiazzata da due `printf`:

```
printf("To C, or not to C: ");
printf("that is the question.\n");
```

- Il carattere di newline potrebbe apparire più di una volta:

```
printf("Brevity is the soul of wit.\n  --Shakespeare\n");
```

Commenti

- Un ***commento*** inizia con `/*` e termina con `*/`

```
/* This is a comment */
```

- I commenti possono apparire (quasi) ovunque nel testo del programma
  - Su linee separate dalle linee che contengono istruzioni
  - Mischiate con i comandi (sulla stessa linea)

- Un commento può occupare più di una singola linea:

```
/* Name: pun.c
   Purpose: Prints a bad pun.
   Author: K. N. King */
```

Commenti

- *Avvertimento:*
  - Dimenticare di "chiudere" un commento fa ignorare parte del programma.

```
printf("My ");     /* Se si dimentica di chiudere...
printf("cat ");
printf("has ");    /* il commento finisce qui */
printf("fleas");
```

Commenti in C99

- Nello standard C99, i commenti possono essere inseriti anche in questo modo:

```
// This is a comment
```

- Con questa sintassi il commento finisce con la riga
- Vantaggi del commento `//` :
  - Più sicuro: non c'è la possibilità di "dimenticare" di chiuderlo
  - Commenti su più linee sono più evidenti
    - Richiedono uno `//` su ogni linea

```
// Name: pun.c
//       Purpose: Prints a bad pun.
//       Author: K. N. King
```

Variabili ed assegnamenti

- Durante l'esecuzione di un programma è necessario memorizzare dati temporanei
- Le locazioni di memoria utilizzate per questo scopo vengono dette ***variabili***
- Ogni ***variabile*** è identificata da un nome (identificativo) che è una sequenza alfanumerica che inizia con un carattere alfabetico

Tipi

- Ogni variabile deve avere un ***tipo***
- Esistono molti tipi fra cui `int` e `float`
- Il tipo `int` (abbreviazione di *intero*) può memorizzare numeri interi
  - Ad esempio: 0, 1, 392, –2553
  - Il più grande valore memorizzabile in una variabile di tipo `int` dipende dal sistema che si usa
  - Tipicamente è 2.147.483.647 (2^{32}-1) ma può essere anche molto più piccolo: 32.767 (2^{16}-1)

Tipi

- Una variabile di tipo `float` (abbreviazione di *floating-point*) può memorizzare numeri con parte decimale
  - Esempio: 379.125
  - Nota: utilizziamo il punto, non la virgola
- E anche numeri interi più grandi di quelli memorizzabili con `int`
- Aspetti negativi del tipo `float`:
  - Operazioni aritmetiche più lente
  - Approssimazione dei risultati

Dichiarazione di variabili

- Una variabile deve essere ***dichiarata*** prima di poter essere usata
- Possiamo dichiarare una variabile alla volta:

```
int height;
float profit;
```

- Oppure più variabili che hanno lo stesso tipo:

```
int height, length, width, volume;
float profit, loss;
```

Dichiarazione variabili

- Le dichiarazioni di variabili di una funzione (es. `main`) devono precedere le istruzioni:

```
int main(void)
{
   dichiarazioni    (declarations)

   istruzioni       (statements)
}
```

- Nello standard C99, le dichiarazioni non devono necessariamente precedere le istruzioni

Assegnamento

- Ad una variabile può essere ***assegnato*** un valore:

```
height = 8;
```

Il numero `8` è una ***costante.***

- Prima di poter assegnare un valore ad una variabile occorre dichiarare la variabile

Assegnamento

- Una costante assegnata ad una variabile di tipo `float` può contenere numeri dopo il punto (virgola) decimale:

```
profit = 2150.48;
```

- È buona prassi aggiungere la lettera `f` ad una costante che rappresenta un valore da assegnare ad una variable `float`:

```
profit = 2150.48f;
```

  Non farlo potrebbe generare degli avvertimenti (warnings) nella fase di compilazione

Assegnamento

- Ad una variabile di tipo `int` va assegnato un valore `int`
- Ad una varibile di tipo `float` va assegnato un valore `float`.
- Mischiare tipi diversi (es., assegnare un valore `int` ad una variabile `float` oppure un valore `float` ad una variabile `int`) è possibile, ma non sempre "sicuro"

Assegnamento

- Dopo aver assegnato un valore ad una variabile possiamo usare la variabile per altri calcoli:

```
height = 8;
length = 12;
width = 10;
volume = height * length * width;
  /* volume is now 960 */
```

- La parte destra di un assegnamento (cioè la parte dopo il carattere =) può essere una formula (detta ***espressione,*** nella terminologia C) che coinvolge costanti, variabili ed operatori.

Stampare il valore di una variabile

- `printf` può essere usata per stampare il valore di una variabile

- Per stampare il messaggio

  `Height: 37`

  dove *37* è il valore attuale della variabile `height` si può usare la seguente `printf`:

```
printf("Height: %d\n", height);
```

- `%d` è un segnaposto che indica dove il valore della variabile `height` deve essere stampato

Stampare il valore di una variabile

- `%d` funziona per variabili di tipo `int`; per il tipo `float` si usa il segnaposto `%f`
- Per default, `%f` stampa un numero con 6 cifre decimali
- Per cambiare questo formato si possono usare dei *modificatori*: specificando *.p* fra `%` e `f` si fanno stampare *p* cifre decimali
- Esempio

```
printf("Profit: $%.2f\n", profit);
```

stampa

```
Profit: $2150.48
```

Stampare il valore di una variabile

- In una singola `printf` si possono stampare quante variabili si vogliono

```
printf("Height: %d  Length: %d\n", height, length);
```

Programma: peso dimensionale

- Le compagnie di spedizioni spesso applicano delle tariffe aggiuntive su scatole grandi in base alle dimensioni
- Ogni Kg può avere una dimensione massima di 166 cm cubici. Quindi, in pratica si divide il volume per 166 per avere il peso "dimensionale".
- La tariffa viene applicata sul peso dimensionale (anche se il peso reale è minore)
- Il programma `dweight.c` calcola il peso dimensionale di una scatola di dimensione 12x10x8 cm:

Programma: peso dimensionale

- L'operatore di divisione è il simbolo / , quindi la formula per il peso dimensionale è

```
weight = volume / 166;
```

- Tuttavia in C, quando un intero viene diviso per un altro intero il risultato è anch'esso un intero e quindi viene "troncato", cioè la parte decimale viene persa
  - Il volume di una scatola 12cm × 10cm × 8cm è 960 cm cubici
  - Se si divide 960 per 166 si dovrebbe ottenere 5.783
  - In C, usando variabili intere, si ottiene 5

Programma: peso dimensionale

- Una possibile soluzione è aumentare il volume reale di 165 prima di dividere per 166:

```
weight = (volume + 165) / 166;
```

- In questo modo
  - Un volume reale fra 1 e 165 dà come risultato 1
  - Anche 166 darebbe come risultato 331/166, cioè 1
  - Mentre 167 darebbe come risultato 332/166, cioè 2
  - E così via

dweight.c

```
/* Computes the dimensional weight of a 12 x 10 x 8 box */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  height = 8;
  length = 12;
  width = 10;
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Dimensions: %dx%dx%d\n", length, width, height);
  printf("Volume (cubic centimeters): %d\n", volume);
  printf("Dimensional weight (kilograms): %d\n", weight);

  return 0;
}
```

Programma: peso dimensionale

```
Output:

  Dimensions: 12x10x8
  Volume (cubic centimeters): 960
  Dimensional weight (kilograms): 6
```

Inizializzazione

- Il linguaggio C non garantisce che il valore iniziale di una variabile sia zero
  - Anche se spesso è così
- Per essere sicuri occorre ***inizializzare*** la variabile, cioè assegnare un valore iniziale prima di usarla
- Usare una variabile non inizializzata può produrre risultati non prevedibili
- In alcuni casi il programma può andare in ***crash***

Inizializzazione

- Il valore iniziale può essere specificato nella dichiarazione:

```
int height = 8;
```

- Più variabili possono essere inizializzate sulla stessa linea:

```
int height = 8, length = 12, width = 10;
```

- È obbligatorio inizializzare ogni singola variabile

```
int height, length, width = 10;
  /* initializza solo width */
```

Stampare il risultato di un'espressione

- `printf` può stampare il risultato di un'espressione

- Le istruzioni

```
volume = height * length * width;
printf("%d\n", volume);
```

potrebbero essere sostituite da

```
printf("%d\n", height * length * width);
```

Input

- `scanf` è la controparte di `printf`.
- `scanf` necessita di una ***stringa di formato*** per specificare il formato dei dati in input
- Ecco un esempio di `scanf` per leggere un valore `int`:

```
scanf("%d", &i);
/* reads an integer; stores into i */
```

- Il simbolo `&` anteposto al nome della variabile che conterrà il valore di input è di solito (ma non sempre) richiesto dalla `scanf`.

Input

- Per leggere il valore di una variabile `float` dobbiamo usare `%f` al posto di `%d`:

```
scanf("%f", &x);
```

- `"%f"` dice alla funzione `scanf` di leggere un valore nel formato `float` (cioè un numero che può avere anche delle cifre decimali)

Programma: peso dimensionale

- `dweight2.c` è una versione migliorata del programma precedente
  - Utilizziamo la scanf per prendere in input le dimensioni della scatola

- Ogni chiamata a `scanf` è preceduta da una `printf` che visualizza un ***prompt***

dweight2.c

```
/* Computes the dimensional weight of a box from input provided by the user */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  printf("Enter height of box: ");
  scanf("%d", &height);
  printf("Enter length of box: ");
  scanf("%d", &length);
  printf("Enter width of box: ");
  scanf("%d", &width);
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Volume (centimeters): %d\n", volume);
  printf("Dimensional weight (kilograms): %d\n", weight);

  return 0;
}
```

Programma: peso dimensionale

- Esempio di output:

```
Enter height of box: 8
Enter length of box: 12
Enter width of box: 10
Volume (centimeters): 960
Dimensional weight (kilograms): 6
```

- Si noti che il "prompt" (es. "`Enter height of box:`") non deve finire con il newline ("`\n`")

Definizione di nomi e costanti

- `dweight.c` e `dweight2.c` utilizzano la costante 166
  - Il suo significato potrebbe non essere chiaro

- Possiamo usare una ***macro***, in cui associamo il valore ad un nome

```
#define CMCUBICI_PER_KG 166
```

Definizione di nomi e costanti

- Il preprocessore rimpiazza tutte le occorrenze del nome con il valore numerico prima della compilazione

- Il testo

```
weight = (volume + CMCUBICI_PER_KG - 1) / CMCUBICI_PER_KG;
```

dopo il preprocessing diventa

```
weight = (volume + 166 - 1) / 166;
```

Definizione di nomi e costanti

- Il valore di una macro può essere un'espressione:

```
#define RECIPROCAL_OF_PI (1.0f / 3.14159f)
```

- Se contiene operatori l'espressione deve essere racchiusa fra parentesi tonde.

- È prassi comune usare solo lettere maiuscole (e l'underscore) per i nomi delle macro

Programma: da Fahrenheit a Celsius

- Il programma `celsius.c` chiede in input una temperatura (espressa in Fahrenheit); dopodiché stampa il valore equivalente in Celsius

- Esempio di output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

- Il programma gestisce anche le temperature che non sono numeri interi

celsius.c

```
/* Converts a Fahrenheit temperature to Celsius */

#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

int main(void)
{
  float fahrenheit, celsius;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &fahrenheit);

  celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;

  printf("Celsius equivalent: %.1f\n", celsius);

  return 0;
}
```

Programma: da Fahrenheit a Celsius

- Definire `SCALE_FACTOR` come `(5.0f / 9.0f)` invece di `(5 / 9)` è importante.
- Si noti l'uso di `%.1f` per visualizzare la temperatura in Celsius con una sola cifra decimale

Identificatori

- I nomi di variabili, funzioni, macro e altre entità, sono detti ***identificatori***
- Un identificatore può contenere lettere, numeri e trattini bassi (underscore), ma ***deve iniziare*** con una lettera o un underscore:

```
times10   get_next_char   _done
```

  è buona prassi evitare di utilizzare identificatori che iniziano con un underscore
- Esempi di identificatori non ammessi:

```
10times   get-next-char
```

Identificatori

- Il linguaggio C è ***case-sensitive:***
  - Fa distinzione fra lettere minuscole e lettere maiuscole

- Ad esempio, i seguenti identificatori sono tutti diversi:

```
job   joB   jOb   jOB   Job   JoB   JOb   JOB
```

Identificatori

- È prassi comune utilizzare solo lettere minuscole (e underscore come separatore) per i nomi di variabili e funzioni:

```
symbol_table   current_page   name_and_address
```

- Alcuni programmatori preferiscono usare una lettera maiuscola per separare le parole

```
symbolTable   currentPage   nameAndAddress
```

- Non c'è limite alla lunghezza di un identificatore

Parole chiave (Keywords)

- Le seguenti parole chiave non possono essere usate come identificatori:

`auto`	`break`	`case`	`char`	C89
`const`	`continue`	`default`	`do`	
`double`	`else`	`enum`	`extern`	
`float`	`for`	`goto`	`if`	
`int`	`long`	`register`	`unsigned`	
`return`	`short`	`signed`	`sizeof`	
`static`	`struct`	`switch`	`typedef`	
`union`	`void`	`volatile`	`while`	
`inline`	`restrict`	`_Bool`	`_Complex`	C99
`_Imaginary`				
`_Alignas`	`_Alignof`	`_Atomic`	`_Generic`	C11
`_NoReturn`	`_StaticAssert`	`_ThreadLocal`		

Keywords

- Le parole chiave hanno lettere minuscole
  - con le eccezioni di quelle che iniziano per “underscore” (`_Bool`,`_Complex`,...)

- Anche le funzioni di libreria hanno nomi con lettere minuscole (es., `printf`)

Programma C

- Un programma C è una sequenza di ***token***
  - ***gruppi di caratteri che non possono essere separati tra loro senza cambiarne significato***
- Un token è un/una:
  - Identificatore
  - Keyword
  - Operatore
  - Punteggiatura
  - Costante
  - Stringa letterale

Programma C

- L'istruzione

```
printf("Height: %d\n", height);
```

è formata da una sequenza di 7 token:

1.	`printf`	Identificatore
2.	`(`	Punteggiatura
3.	`"Height: %d\n"`	Stringa letterale
4.	`,`	Punteggiatura
5.	`height`	Identificatore
6.	`)`	Punteggiatura
7.	`;`	Punteggiatura

Programma C

- Lo spazio bianco fra i token è irrilevante.
- Si può eliminare completamente lo spazio bianco, tranne quando facendo ciò si produrrebbe la fusione di due token in un unico token:

```
/* Converts a Fahrenheit temperature to Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f/9.0f)
int main(void){float fahrenheit,celsius;printf(
"Enter Fahrenheit temperature: ");scanf("%f", &fahrenheit);
celsius=(fahrenheit-FREEZING_PT)*SCALE_FACTOR;
printf("Celsius equivalent: %.1f\n", celsius);return 0;}
```

Programma C

- Si ricordi che le direttive vanno su linee separate
- Scrivere i programmi senza spazio bianco non è una buona prassi
  - Diventa difficilissimo leggerli

- Per facilitare la leggibilità del programma anzi è buona norma usare gli spazi bianchi per ***indentare*** correttamente i blocchi di istruzioni
  - questa "buona norma" dovrebbe essere considerata una regola

Programma C

- Lo spazio bianco è fatto dai caratteri:
  - spazio, tab e newline
- In conclusione:
  - *Le istruzioni possono essere disposte* su un qualsiasi numero di linee
  - *Spazi fra i token* (es., prima e dopo un operatore) facilitano la lettura
  - L'*indentazione* facilita l'individuazione dei blocchi innestati.
  - *Linee completamente bianche* possono dividere il testo in blocchi logici

Programma C

- Attenzione:
  - Spazio bianco può essere inserito fra due token
  - NON all'interno di un singolo token
    - In quel caso cambierebbe il significato del programma

- Scrivere

```
fl oat fahrenheit, celsius;  /*** WRONG ***/
```

oppure

```
fl
oat fahrenheit, celsius;     /*** WRONG ***/
```

produce un errore in fase di compilazione

… arrivederci alla prossima lezione