

STRUTTURA DI UN PROGRAMMA C

- In prima battuta, la struttura di un programma C è definita nel modo seguente:

```
<programma> ::=  
  {<unità-di-traduzione>}  
  <main>  
  {<unità-di-traduzione>}
```

- Intuitivamente un programma in C è definito da tre parti:

- una o più unità di traduzione,
- il programma vero e proprio (main)
- una o più unità di traduzione

STRUTTURA DI UN PROGRAMMA C

- La parte <main> è l'unica **obbligatoria**, ed è definita come segue:

```
<main> ::=  
  main()  
  {[<dichiarazioni-e-definizioni>]  
  [<sequenza-istruzioni>]  
  }
```

- Intuitivamente il main è definito dalla parola chiave `main()` e racchiuso tra parentesi graffe al cui interno troviamo
 - le dichiarazioni e definizioni
 - una sequenza di istruzioni

STRUTTURA DI UN PROGRAMMA C

- <dichiarazioni-e-definizioni>

introducono i *nomi* di costanti, variabili, tipi definiti dall'utente

- <sequenza-istruzioni>

sequenza di frasi del linguaggio ognuna delle quali è un'istruzione

Il `main()` è una particolare unità di traduzione (una funzione).

STRUTTURA DI UN PROGRAMMA C

- **set di caratteri** ammessi in un programma dipende dall'implementazione; solitamente ASCII + estensioni

- **identificatori**

sequenze di caratteri tali che

```
<Identificatore> ::=  
  <Lettera> {<Lettera> | <Cifra> }
```

- Intuitivamente un identificatore è una sequenza (di lunghezza maggiore o uguale a 1) di lettere e cifre che inizia obbligatoriamente con una lettera.

COMMENTI

- **commenti**

sequenze di caratteri racchiuse fra i delimitatori
`/*` e `*/`

```
<Commento> ::= /* <frase> */  
<frase> ::= {<parola> }  
<parola> ::= {<carattere> }
```

- i commenti non possono essere innestati.

VARIABILI

- Una **variabile** è un'astrazione della **cella di memoria**.

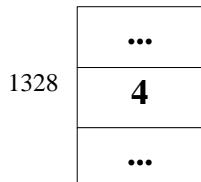
- Formalmente, è un simbolo **associato a un indirizzo fisico (L-value)**...

simbolo	indirizzo
X	1328

Perciò, l' **L-value** di x è 1328 (**fisso e immutabile!**).

VARIABILI

... che *denota un valore (R-value)*.



..e l' **R-value** di x è *attualmente* 4 (può cambiare).

DEFINIZIONE DI VARIABILE

- Una variabile utilizzata in un programma deve essere definita.
- La definizione è composta da
 - il nome della variabile (identificatore)
 - il **tipo** dei valori (R-value) che possono essere denotati alla variabile

DEFINIZIONE DI VARIABILE: ESEMPI

Definizione di una variabile:

```
<tipo> <identificatore>;  
  
int x; /* x deve denotare un valore intero */  
  
float y; /* y deve denotare un valore reale */  
  
char ch; /* ch deve denotare un carattere */
```

INIZIALIZZAZIONE DI UNA VARIABILE

- Contestualmente alla *definizione* è possibile *specificare un valore iniziale* per una variabile
- Inizializzazione di una variabile:

```
<tipo> <identificatore> = <espr> ;
```
- Esempio

```
int x = 32;  
double speed = 124.6;
```

VARIABILI & ESPRESSIONI

Una variabile

- può comparire in una espressione
- può assumere un valore dato dalla valutazione di un'espressione

```
double speed = 124.6;  
double time = 71.6;  
double km = speed * time;
```

CARATTERISTICHE DELLE VARIABILI

- **campo d'azione (scope):** è la parte di programma in cui la variabile è nota e può essere manipolata
 - in C, Pascal: determinabile *staticamente*
 - in LISP: determinabile *dinamicamente*
- **tipo:** specifica la *classe di valori* che la variabile può assumere (e quindi gli operatori applicabili)

CARATTERISTICHE DELLE VARIABILI

- **tempo di vita**: è l'intervallo di tempo in cui rimane valida l'associazione simbolo/indirizzo fisico (L-VALUE)
 - in FORTRAN: allocazione *statica*
 - in C, Pascal: allocazione *dinamica*
- **valore**: è rappresentato (secondo la codifica adottata) nell'area di memoria associata alla variabile

ESEMPIO: Un semplice programma

Problema:

“Data una temperatura espressa in gradi Celsius, calcolare il corrispondente valore espresso in gradi Fahrenheit”

Approccio:

- si parte dal **problema e dalle proprietà note sul dominio dei dati**

ESEMPIO: Un semplice programma

Specifiche della soluzione:

$$c * 9/5 = f - 32$$

oppure

$$c = (f - 32) * 5/9$$

$$f = 32 + c * 9/5$$

ESEMPIO: Un semplice programma

L'Algoritmo corrispondente:

- Dato **c**
- calcolare **f** sfruttando la relazione $f = 32 + c * 9/5$

solo a questo punto

- si codifica l'algoritmo nel linguaggio scelto.

ESEMPIO: Un semplice programma

```
main(){  
    float c=18; /* Celsius */  
    float f = 32 + c * 9/5;  
}
```



NOTA: per ora abbiamo a disposizione solo il modo per inizializzare le variabili. Mancano, ad esempio, la possibilità di modificare una variabile, costrutti per l'input output

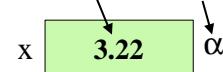
VARIABILI NEI LINGUAGGI IMPERATIVI

Una **variabile** in un linguaggio imperativo

- **non è solo un sinonimo per un dato** come in matematica

• **è un'astrazione della cella di memoria**

- associata a due diverse informazioni:
 - il **contenuto (R-value)**
 - l'**indirizzo a cui si trova (L-value)**



ESPRESSIONI CON EFFETTI COLLATERALI

- Le espressioni che contengono variabili, oltre a denotare un valore, possono a volte comportare *effetti collaterali* sulle variabili coinvolte.
- Un *effetto collaterale* è una modifica del valore della variabile (R-value) causata da *particolari operatori*:
 - operatori di *assegnamento*
 - operatori di *incremento e decremento*

ASSEGNAZIONE

- Ad una variabile può essere assegnato un valore nel corso del programma e non solo all'atto della inizializzazione.
- Assegnamento di una variabile: SINTASSI
`<identificatore> = <espr> ;`
- L'assegnamento è l'astrazione della modifica distruttiva del contenuto della cella di memoria denotata dalla variabile.

`int x = 32;`  `x` 
`...`  `x` 

ASSEGNAZIONE

- L'assegnamento è un *particolare tipo di espressione*
 – come tale denota comunque un valore!
con un effetto collaterale: quello di cambiare il valore della variabile.
- Esempi di espressioni di assegnamento:
 $j = 0$ $k = j + 1$
- Se k valeva 2, l'espressione $k = j + 1$
 - denota il valore 1 (risultato della valutazione dell'espressione)
 - e cambia il valore di k , che d'ora in poi vale 1 (non più 2)

L'assegnamento è distruttivo

ASSEGNAZIONE & VARIABILI

Una variabile in una espressione di assegnamento:

- è interpretata come il suo R-value, se compare a destra del simbolo = 
- è interpretata come il suo L-value, se compare a sinistra del simbolo = 

ASSEGNAZIONE & VARIABILI

Se x valeva 2, l'espressione

$$x = x + 1$$

- denota il valore 3
- e cambia in 3 il valore di x
 - il simbolo x a destra dell'operatore = denota il valore attuale (R-value) di x , cioè 2
 - il simbolo x a sinistra dell'operatore = denota la cella di memoria associata a x (L-value), a cui viene assegnato il valore dell'espressione di destra (3)
 - l'espressione nel suo complesso denota il valore della variabile dopo la modifica, cioè 3.

OPERATORI DI ASSEGNAZIONE COMPATTI

Il C introduce una *forma particolare di assegnamento* che **ingloba anche un'operazione** aritmetica:

`l-espr OP= <espressione>`
 è "quasi equivalente" a
`l-espr = l-espr OP <espressione>`
 dove **OP** indica un operatore fra
 $+, -, *, /, \%, >-, <-, \&, ^, |$

OPERATORI DI ASSEGNAZIONE COMPATTI

Esempi

```
k += j    equivale a  k = k + j
k *= a + b  equivale a  k = k*(a+b)*/
```

• Perché "quasi" equivalente ?

- nel primo caso, *l-expr* viene valutata *una sola volta*
- nel secondo, invece, viene valutata *due volte*
- Quindi, le due forme sono *equivalenti solo se la valutazione di l-expr non comporta effetti collaterali*

INCREMENTO E DECREMENTO

Gli operatori di incremento e decremento sono *usabili in due modi*

- **come pre-operatori:** `++v`
prima incremento e poi uso
- **come post-operatori:** `v++`
prima uso e poi incremento

ESEMPI

```
• int i, k = 5;
i = ++k /* i vale 6, k vale 6 */
• int i, k = 5;
i = k++ /* i vale 5, k vale 6 */
• int i=4, j, k = 5;
j = i + k++; /* j vale 9, k vale 6 */
• int j, k = 5;
j = ++k - k++; /* DA NON USARE */
/* j vale 0, k vale 7 */
```

COMPATIBILITÀ DI TIPO

- In un assegnamento, l'identificatore di variabile e l'espressione devono essere dello stesso tipo.
 - Nel caso di tipi diversi, se possibile si effettua la conversione implicita, altrimenti l'assegnamento può generare perdita di informazione
- ```
int x;
char y;
double r;

x = y; /* char -> int */
x = y+x;
r = y; /* char -> int -> double */
x = r; /* troncamento */
```

## ESEMPIO

```
main()
{
 /* parte dichiarazioni variabili */
 int X,Y;
 unsigned int Z;
 float SUM;
 /* segue parte istruzioni */
 X=27;
 Y=343;
 Z = X + Y -300;
 X = Z / 10 + 23;
 Y = (X + Z) / 10 * 10;
 /* qui X=30, Y=100, Z=70 */
 X = X + 70;
 Y = Y % 10;
 Z = Z + X -70;
 SUM = Z * 10;
 /* qui X=100, Y=0, Z=100 , SUM =1000.0*/
}
```

## INPUT/OUTPUT

- L'immissione dei dati di un programma e l'uscita dei suoi risultati avvengono attraverso operazioni di lettura e scrittura.
- Il C non ha istruzioni predefinite per l'input/output.
- In ogni versione ANSI C, esiste una *Libreria Standard (stdio)* che mette a disposizione alcune funzioni ( dette *funzioni di libreria*) per effettuare l'input e l'output.

## INPUT/OUTPUT

- Le dichiarazioni delle funzioni messe a disposizione da tale libreria devono essere essere incluse nel programma:  
`#include <stdio.h>`
  - #include è una direttiva per il **preprocessore C**:
  - nella fase precedente alla compilazione del programma ogni direttiva "#..." viene eseguita, provocando delle modifiche testuali al programma sorgente. Nel caso di `#include <nomefile>`:
    - viene sostituita l'istruzione stessa con il contenuto del file specificato.
- Dispositivi standard di input e di output:**
  - per ogni macchina, sono periferiche predefinite (generalmente tastiera e video).

## INPUT/OUTPUT

- Il C vede le informazioni lette/scritte da/verso i dispositivi standard di I/O come file **sequenziali**, cioè **sequenze di caratteri** (o stream).
  - Gli *stream* di input/output possono contenere dei caratteri di controllo:
    - End Of File (EOF)
    - End Of Line (EOL)
- Sono disponibili funzioni di libreria per:**
  - Input/Output a caratteri
  - Input/Output a stringhe di caratteri
  - Input/Output con formato

|   |   |   |   |   |   |     |   |   |   |     |     |
|---|---|---|---|---|---|-----|---|---|---|-----|-----|
| a | 7 | 1 | 3 | 7 | * | b   | 5 | 7 | * | c   | EOF |
|   |   |   |   |   |   | EOL |   |   |   | EOL | EOF |

## INPUT/OUTPUT CON FORMATO

- Nell'I/O con formato occorre specificare il formato (*tipo*) dei dati che si vogliono leggere oppure stampare.
- Il formato stabilisce:
  - come interpretare la sequenza dei caratteri immessi dal dispositivo di ingresso (nel caso della lettura)
  - con quale sequenza di caratteri rappresentare in uscita i valori da stampare (nel caso di scrittura)

## LETTURA CON FORMATO: scanf

- E' una particolare forma di assegnamento: la scanf assegna i valori letti alle variabili specificate come argomenti (nell'ordine di lettura).  
`scanf(<stringa-formato>, <sequenza-variabili>);`
- Ad esempio:  

```
int X;
float Y;
scanf("%d%f", &X, &Y);
```

## LETTURA CON FORMATO: scanf

- Scarf legge una serie di valori in base alle specifiche contenute in *<stringa-formato>* e memorizza i valori letti nelle variabili
  - restituisce il numero di valori letti e memorizzati, oppure EOF in caso di *end of file*
  - Gli identificatori delle variabili a cui assegnare i valori sono sempre preceduti dal simbolo &.
  - La *<stringa\_formato>* può contenere dei caratteri qualsiasi (che vengono scartati, durante la lettura), che si prevede vengano immessi dall'esterno, insieme ai dati da leggere.
    - `scanf("%d:%d:%d", &A, &B, &C);`
  - richiede che i tre dati da leggere vengano immessi separati dal carattere ":".

## SCRITTURA CON FORMATO: printf

- La `printf` viene utilizzata per fornire in uscita il valore di una variabile, o, più in generale, il risultato di una espressione.
- Anche in scrittura è necessario specificare (mediante una *stringa di formato*) il formato dei dati che si vogliono stampare.

```
printf(<stringa-formato>,<sequenza-elementi>)
```

## SCRITTURA CON FORMATO: `printf`

- `printf` scrive una serie di valori in base alle specifiche contenute in `<stringa-formato>`.
- I valori visualizzati sono i risultati delle espressioni che compaiono come argomenti
- La `printf` restituisce il numero di caratteri scritti.
- La stringa di formato della `printf` può contenere sequenze costanti di caratteri da visualizzare.

## FORMATI COMUNI

- Formati più comuni: ne vedremo altri più avanti

|                      |    |
|----------------------|----|
| int                  | %d |
| float                | %f |
| carattere singolo    | %c |
| stringa di caratteri | %s |

- Caratteri di controllo:

|                 |    |
|-----------------|----|
| newline         | \n |
| tab             | \t |
| backspace       | \b |
| form feed       | \f |
| carriage return | \r |

- Per la stampa del carattere ' % ' si usa: %%

## ESEMPIO

- ```
main()
{
    int k;
    scanf("%d",&k);
    printf("Quadrato di %d: %d",k,k*k);
}
```
- Se in ingresso viene immesso il dato:
3 viene letto tramite la `scanf` e assegnato a `k`
 - La `printf` stampa:
Quadrato di 3: 9

ESEMPIO

```
scanf("%c%c%c%d%f", &c1,&c2,&c3,&i,&x);
```

- Se in ingresso vengono dati:
ABC 3 7.345
- La `scanf` effettua i seguenti assegnamenti:

char c1	'A'
char c2	'B'
char c3	'C'
int i	3
float x	7.345

ESEMPIO

```
char Nome='F';
char Cognome='R';
printf("%s\n%c. %c. \n\n%s\n",
       "Programma scritto da:",
       Nome, Cognome,"Fine");
```

vengono stampate le seguenti linee
Programma scritto da:
F. R.
Fine

ESEMPIO

```
main(){
    float c, f; /* Celsius e Fahrenheit */
    printf("Inserisci la temperatura da convertire");
    scanf("%f", c);
    f = 32 + c * 9/5;
    printf("Temperatura Fahrenheit %f", f);
}
```