Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2021-2022

Dichiarazione di variabili e tipi di dati

Catia Prandi - catia.prandi2@unibo.it

Credit: **Pietro Di Lena**

```
long time, ago; /* in a galaxy far, far away
```

©George Lucas

Dichiarazione di variabili

- Una **variabile** è una porzione di memoria contenente dati che possono eventualmente essere modificati durante l'esecuzione del programma.

Dichiarazione di variabili

- Una **variabile** è una porzione di memoria contenente dati che possono eventualmente essere modificati durante l'esecuzione del programma.
- Ogni variabile deve essere **dichiarata** prima di poter essere utilizzata.

Dichiarazione di variabili

- Una **variabile** è una porzione di memoria contenente dati che possono eventualmente essere modificati durante l'esecuzione del programma.
- Ogni variabile deve essere **dichiarata** prima di poter essere utilizzata.
- La dichiarazione di variabile è un'*istruzione* che permette di associare alla variabile:
  - un **identificativo**, che rappresenta il *nome* (univoco) della variabile
  - un **tipo**, che definisce la *dimensione* della variabile in termini di occupazione di memoria, oltre al range di valori ammissibili per tale tipo.

Dichiarazione di variabili

- Una **variabile** è una porzione di memoria contenente dati che possono eventualmente essere modificati durante l'esecuzione del programma.
- Ogni variabile deve essere **dichiarata** prima di poter essere utilizzata.
- La dichiarazione di variabile è un'*istruzione* che permette di associare alla variabile:
  - un **identificativo**, che rappresenta il *nome* (univoco) della variabile
  - un **tipo**, che definisce la *dimensione* della variabile in termini di occupazione di memoria, oltre al range di valori ammissibili per tale tipo.
- Sintassi semplificata per la dichiarazione (**definizione**) di variabili:

```
<Tipo> <Identificatore>;
```

```
1 char symbol;
2 float height;
3 int x;
4 int X;
5 double temperature, pressure;
```

- L'istruzione a riga 1 dichiara una variabile di tipo `char` (carattere) con nome *symbol*.
- L'istruzione a riga 2 dichiara una variabile di tipo `float` (numero reale) con nome *height*.
- Le istruzioni a riga 3 e 4 dichiarano due variabili di tipo `int` (numero intero), una con nome x ed una con nome X.
- Le dichiarazioni di variabili dello stesso tipo possono essere *contratte* in un'unica riga di codice, separando i nomi delle variabili con una virgola (vedi riga 5).

Nota: Dichiarazione vs Definizione

- Lo standard C fa una precisa distinzione tra i concetti **dichiarazione** e **definizione**.
  - **Dichiarazione**: specifica l'interpretazione e gli attributi di un identificatore.
  - **Definizione**: dichiara un identificatore e riserva spazio in memoria per tale identificatore.
- Vedremo più avanti esempi espliciti di dichiarazioni che non sono definizioni.
- Ci capiterà di utilizzare la terminologia in modo improprio:
  - Quando parliamo di **dichiarazione di variabile**, a meno di casi specifici (che vedremo più avanti), intendiamo **definizione di variabile**

Inizializzazione di variabili

- La **definizione** di una variabile consente al compilatore di individuare un'area di memoria da riservare alla variabile.
- Ogni riferimento al nome della variabile all'interno del codice è un riferimento al contenuto di tale locazione di memoria.

Inizializzazione di variabili

- La **definizione** di una variabile consente al compilatore di individuare un'area di memoria da riservare alla variabile.
- Ogni riferimento al nome della variabile all'interno del codice è un riferimento al contenuto di tale locazione di memoria.
- Al momento della dichiarazione/definizione, ad una variabile non viene assegnato nessun valore: non possiamo fare nessuna assunzione sul contenuto della locazione di memoria associata alla variabile. Diciamo che il valore della variabile è non definito.

Inizializzazione di variabili

- La **definizione** di una variabile consente al compilatore di individuare un'area di memoria da riservare alla variabile.
- Ogni riferimento al nome della variabile all'interno del codice è un riferimento al contenuto di tale locazione di memoria.
- Al momento della dichiarazione/definizione, ad una variabile non viene assegnato nessun valore: non possiamo fare nessuna assunzione sul contenuto della locazione di memoria associata alla variabile. Diciamo che il valore della variabile è non definito.
- E' possibile **inizializzare** il contenuto una variabile al momento della definizione utilizzando **l'operatore di assegnamento** =

```
1 char symbol = 'A';
2 float height = 1.75;
3 int x = -1;
4 int X;
5 double temperature = 100, pressure = 1.5;
```

Nell'esempio sopra, tutte le variabili sono inizializzate, tranne X.

- L'assegnamento può essere effettuato tramite costanti letterali (come nell'esempio) o tramite espressioni più complesse (che vedremo in seguito).

Tipi di dati

- Il C è un linguaggio **tipizzato**, ovvero richiede che venga associato un tipo ad ogni dato utilizzato nel codice.

Tipi di dati

- Il C è un linguaggio **tipizzato**, ovvero richiede che venga associato un tipo ad ogni dato utilizzato nel codice.
- La tipizzazione permette al compilatore:
  - di capire come **interpretare** il contenuto della locazione di memoria riservata alla variabile,
  - di verificare che ci sia **coerenza** nell'uso dei dati. Ad esempio, la seguente inizializzazione (assegnamento di una stringa letterale ad una variabile di tipo carattere) non è permessa in C:

```
char symbol = "wrong initialization";
```

Tipi di dati

- Il C è un linguaggio **tipizzato**, ovvero richiede che venga associato un tipo ad ogni dato utilizzato nel codice.
- La tipizzazione permette al compilatore:
  - di capire come **interpretare** il contenuto della locazione di memoria riservata alla variabile,
  - di verificare che ci sia **coerenza** nell'uso dei dati. Ad esempio, la seguente inizializzazione (assegnamento di una stringa letterale ad una variabile di tipo carattere) non è permessa in C:

```
char symbol = "wrong initialization";
```

- Il linguaggio C può essere definito come un linguaggio **debolmente tipizzato**, in quanto presenta dei meccanismi impliciti ed espliciti per trattare in modo flessibile le regole di *conversione* tra tipi di dati (vedremo in seguito tali regole).
- Nello standard ISO C89, sono ammessi un numero ristretto di **specificatori** e **modificatori** di tipo, che possono essere *parzialmente combinati* tra di loro per definire **tipi di dati** basilari del linguaggio C.
- Sia gli specificatori che i modificatori di tipo sono identificati tramite **keyword** del linguaggio.

Tipi di dati: specificatori e modificatori

- **Specificatori** di tipo: classi primarie di memorizzazione del linguaggio C.
  - `void`: tipo di dato *indefinito*, ovvero *non specificato*,
  - `char`: rappresenta i caratteri nel set a disposizione sul calcolatore,
  - `int`: rappresenta numeri interi,
  - `float`: rappresenta numeri in *virgola mobile* (decimali) a *precisione singola*,
  - `double`: rappresenta numeri *virgola mobile* a *precisione doppia*.

Tipi di dati: specificatori e modificatori

- **Specificatori** di tipo: classi primarie di memorizzazione del linguaggio C.
  - `void`: tipo di dato *indefinito*, ovvero *non specificato*,
  - `char`: rappresenta i caratteri nel set a disposizione sul calcolatore,
  - `int`: rappresenta numeri interi,
  - `float`: rappresenta numeri in *virgola mobile* (decimali) a *precisione singola*,
  - `double`: rappresenta numeri *virgola mobile* a *precisione doppia*.
- **Modificatori** di tipo: modificano il range di valori ammissibili per il tipo
  - `short`: riduce il numero di valori memorizzabili,
  - `long`: aumenta il numero di valori memorizzazabili,
  - `signed`: il tipo di dato è dotato di segno,
  - `unsigned`: il tipo di dato non è dotato di segno.

Combinazioni di specificatori e modificatori

- Combinazioni sintatticamente valide di specificatori e modificatori.
- Le combinazioni sulla stessa riga indicano tipi perfettamente equivalenti.

Combinazioni	**Classe**
`void`	Tipo indefinito

`char`
`signed char`
`unsigned char`
`short`, `signed short`, `short int`, `signed short int`
`unsigned short`, `unsigned short int`
`int`, `signed`, `signed int` (o anche nessun tipo)
`unsigned`, `unsigned int`
`long`, `signed long`, `long int`, `signed long int`
`unsigned long`, `unsigned long int`
	Tipi interi

`float`
`double`
`long double`
	Tipi in virgola mobile

Tipi di dati: regole sintattiche aggiuntive

- Gli specificatori e i modificatori di tipo possono essere posizionati in qualsiasi ordine all'interno di una dichiarazione di tipo.
- E' possibile dichiarare solo il tipo di dato senza specificare un identificatore. Tali dichiarazioni sono sintatticamente valide ma totalmente inutili.
- Il tipo di dato `void` non può essere assegnato ad un nome di variabile (la compilazione termina in errore).
  - Vedremo più avanti come utilizzare lo specificatore di tipo `void`.
- Nel seguente esempio, tutte le dichiarazioni sono valide, tranne quella a riga 7.

```
1  int main() {
2   int short x,y=0;// Equivalente a "short int x,y=0;"
3   double long z;  // Equivalente a "long double z;"
4   char unsigned a;// Equivalente a "unsigned char a;"
5   long int;       // Dichiarazione "vuota"
6   void;           // Dichiarazione "vuota"
7   void b;         // Dichiarazione non valida
8
9   return 0;
10 }
```

Memorizzazione delle informazioni sulla memoria del calcolatore (cenni)

- Per poter comprendere e saper utilizzare correttamente costanti, variabili e tutti gli aspetti del linguaggio che hanno attinenza con i tipi di dati, è necessario comprendere come tali dati vengano *memorizzati* sul calcolatore.

Memorizzazione delle informazioni sulla memoria del calcolatore (cenni)

- Per poter comprendere e saper utilizzare correttamente costanti, variabili e tutti gli aspetti del linguaggio che hanno attinenza con i tipi di dati, è necessario comprendere come tali dati vengano *memorizzati* sul calcolatore.
- Il calcolatore ha una **memoria finita**, quindi, per un dato tipo è possibile rappresentare solo un numero ristretto di valori possibili. Ad esempio, il tipo di dato `int` non ci permette di rappresentare l'intero dominio dei **numeri interi**.

Memorizzazione delle informazioni sulla memoria del calcolatore (cenni)

- Per poter comprendere e saper utilizzare correttamente costanti, variabili e tutti gli aspetti del linguaggio che hanno attinenza con i tipi di dati, è necessario comprendere come tali dati vengano *memorizzati* sul calcolatore.
- Il calcolatore ha una **memoria finita**, quindi, per un dato tipo è possibile rappresentare solo un numero ristretto di valori possibili. Ad esempio, il tipo di dato `int` non ci permette di rappresentare l'intero dominio dei **numeri interi**.
- La memoria di un calcolatore può essere astrattamente considerata come una *sequenza finita di celle*:
  - ogni cella può memorizzare un **bit** (Binary digIT). Il bit è *l'unità di informazione* basilare per i calcolatori e può assumere due soli valori: 0 oppure 1.
  - le celle di memoria sono raggruppate in **parole** di lunghezza prefissata. Una parola è chiamata **byte** e consiste di **8 bit**. Il byte è la più piccola quantità di memoria direttamente *indirizzabile* in molte architetture di calcolatori.
  - Esempi:
    - con 1 bit è possibile memorizzare $2^1 = 2$ valori distinti;
    - con 1 byte (= 8 bit) è possibile memorizzare $2^8 = 256$ valori distinti;
    - con 2 byte (= 16 bit) è possibile memorizzare $2^{16} = 65,536$ valori distinti;
    - con 4 byte (= 32 bit) è possibile memorizzare $2^{32} = 4294967296$ valori.
- Date le limitazioni fisiche e le scelte progettuali adottate per i calcolatori, il **numero di valori distinti** che è possibile memorizzare in un *tipo di dato* dipende dal **numero di byte** associati a tale tipo.

Dimensioni e range di valori dei tipi di dati base

- Lo standard ISO C89 definisce dimensioni minime (in byte) per i tipi di dato base del linguaggio C. Le dimensioni effettive dipendono dall'hardware della macchina: processori a 32 bit o 64 bit tipicamente utilizzano dimensioni differenti.

Tipo	**Byte**	**Min**	**Max**
`signed char`	1	-128	127
`unsigned char`	1	0	255
`short int`	2	-32767	32767
`unsigned short int`	2	0	65535
`int`	4	-2147483647	2147483647
`unsigned int`	4	0	4294967295
`long int`	4	-2147483647	2147483647
`unsigned long int`	4	0	4294967295
`float`	4	$1.175494 \times^{-38}$	$3.402823 \times^{38}$
`double`	8	$2.225074 \times^{-308}$	$1.797693 \times^{308}$
`long double`	16	$3.362103 \times 10^{-4932}$	$1.189731 \times 10^{4932}$

Tabella: Dimensioni **tipiche** su processore a 32 bit

Dimensioni e range di valori dei tipi di dati base

- Lo standard ISO C89 definisce dimensioni minime (in byte) per i tipi di dato base del linguaggio C. Le dimensioni effettive dipendono dall'hardware della macchina: processori a 32 bit o 64 bit tipicamente utilizzano dimensioni differenti.

Tipo	Byte	Min	Max
`signed char`	1	-128	127
`unsigned char`	1	0	255
`short int`	2	-32767	32767
`unsigned short int`	2	0	65535
`int`	4	-2147483647	2147483647
`unsigned int`	4	0	4294967295
`long int`	4	-2147483647	2147483647
`unsigned long int`	4	0	4294967295
`float`	4	$1.175494 \times^{-38}$	$3.402823 \times^{38}$
`double`	8	$2.225074 \times^{-308}$	$1.797693 \times^{308}$
`long double`	16	$3.362103 \times 10^{-4932}$	$1.189731 \times 10^{4932}$

Tabella: Dimensioni **tipiche** su processore a 32 bit

- Il tipo `char` può essere di default `signed` o `unsigned`, a seconda dell'implementazione.
- I tipi in virgola mobile non possono essere `unsigned`. La tabella mostra in valore assoluto il minimo (escludendo lo 0) e massimo numero rappresentabile per `float`, `double` e `long double`.

Il tipo di dato **char**

- Nota storica: il byte (8 bit = 256 valori distinti) è l'unità di misura storicamente utilizzata per poter memorizzare un singolo carattere di testo su un computer.
- In C la distinzione tra tipo di dato **carattere** e tipo di dato **intero** è alquanto sfumata. Il tipo di dato `char` rappresenta effettivamente un intero è può essere utilizzato per memorizzare un limitato numero di valori interi: `signed` (da -128 a 127) e `unsigned` (da 0 a 255).
- La *conversione* tra numero intero e carattere dipende da una *tabelle di codifica* disponibile sulla macchina. La tabella di codifica comunemente utilizzata è lo standard **ASCII** (American Standard Code for Information Interchange).

Il tipo di dato **char**

- Nota storica: il byte (8 bit = 256 valori distinti) è l'unità di misura storicamente utilizzata per poter memorizzare un singolo carattere di testo su un computer.
- In C la distinzione tra tipo di dato **carattere** e tipo di dato **intero** è alquanto sfumata. Il tipo di dato `char` rappresenta effettivamente un intero è può essere utilizzato per memorizzare un limitato numero di valori interi: `signed` (da -128 a 127) e `unsigned` (da 0 a 255).
- La *conversione* tra numero intero e carattere dipende da una *tabelle di codifica* disponibile sulla macchina. La tabella di codifica comunemente utilizzata è lo standard **ASCII** (American Standard Code for Information Interchange).
- Lo standard ASCII originale utilizza solo 7 bit (128 valori distinti) per la codifica dei caratteri. Tale codifica include tutte le lettere dell'alfabeto inglese, segni di punteggiatura, caratteri di spaziatura e codici di controllo.
- La codifica ASCII è stata successivamente incorporata in codifiche ad 8 bit per poter estendere a 256 il numero di caratteri disponibili.

Il tipo di dato **char**

- Nota storica: il byte (8 bit = 256 valori distinti) è l'unità di misura storicamente utilizzata per poter memorizzare un singolo carattere di testo su un computer.
- In C la distinzione tra tipo di dato **carattere** e tipo di dato **intero** è alquanto sfumata. Il tipo di dato `char` rappresenta effettivamente un intero è può essere utilizzato per memorizzare un limitato numero di valori interi: `signed` (da -128 a 127) e `unsigned` (da 0 a 255).
- La *conversione* tra numero intero e carattere dipende da una *tabelle di codifica* disponibile sulla macchina. La tabella di codifica comunemente utilizzata è lo standard **ASCII** (American Standard Code for Information Interchange).
- Lo standard ASCII originale utilizza solo 7 bit (128 valori distinti) per la codifica dei caratteri. Tale codifica include tutte le lettere dell'alfabeto inglese, segni di punteggiatura, caratteri di spaziatura e codici di controllo.
- La codifica ASCII è stata successivamente incorporata in codifiche ad 8 bit per poter estendere a 256 il numero di caratteri disponibili.
- Esempi di codifica ASCII a 7 bit:
  - i codici dallo 0 a 31 e il 127 sono caratteri non stampabili (codici di controllo);
  - l'intero 48 corrisponde al carattere '0';
  - l'intero 65 corrisponde al carattere 'A';
  - l'intero 66 corrisponde al carattere 'B';
  - l'intero 97 corrisponde al carattere 'a'.

Tabella ASCII a 7 bit

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Il tipo booleano

- Nello standard ISO C89 non è previsto il tipo di dato **booleano**, cioè un tipo di dato che può assumere solo due valori, **vero** e **falso** (o, alternativamente, 0 e 1).
- Il tipo di dato booleano (`bool` o `_Bool`) è stato aggiunto successivamente nello standard ISO C99.
- Nel linguaggio C i valori di tipo booleano (vero/falso) possono essere espressi utilizzando i tipi di dato interi (`char, int, long`).
- **Regola** generale del linguaggio C:
  1. il valore intero **0** indica **falso**;
  2. un qualsiasi valore intero **diverso da 0** indica **vero**.

Overflow

- E' importante conosce il range di valori ammissibili per un tipo di dato per evitare problemi di **overflow** durante l'esecuzione del codice. Cosa succede nel seguente esempio? Qual è il valore di `x` dopo l'esecuzione dell'istruzione a riga 4?

```
1 // Assumiamo che la dimensione in byte
2 // di uno short int sia 2
3 int main() {
4   short int x = 4294967295;
5
6   return 0;
7 }
```

Overflow

- E' importante conosce il range di valori ammissibili per un tipo di dato per evitare problemi di **overflow** durante l'esecuzione del codice. Cosa succede nel seguente esempio? Qual è il valore di `x` dopo l'esecuzione dell'istruzione a riga 4?

```
1 // Assumiamo che la dimensione in byte
2 // di uno short int sia 2
3 int main() {
4   short int x = 4294967295;
5
6   return 0;
7 }
```

Non definito!

- Non possiamo fare nessun tipo di assunzione sul valore della variabile `x` a riga 4.
- In generale, l'overflow non è un comportamento desiderabile in un programma e deve essere evitato.

L'operatore `sizeof` e i file header `limits.h` e `float.h`

- Come abbiamo detto, le dimensioni esatte dei tipi di dati base sono specifiche dell'implementazione e possono variare su computer differenti.
  - **Non possiamo contare su valore limite predefinito per un tipo di dato.**
- In C abbiamo due possibilità differenti per verificare la dimensione dei dati su una specifica macchina:
  - l'operatore (keyword) `sizeof` permette di determinare l'occupazione di memoria in byte di un dato tipo o variabile;
  - le dimensioni e proprietà dei vari tipi di dati sulla macchina sono specificate nei file header `limits.h` e `float.h`.

L'operatore sizeof

- L'operatore `sizeof` applicato ad un nome di variabile o tipo, *ritorna* la dimensione in byte dell'oggetto passato come argomento.
- Esempio:

```
1  int main() {
2   char x = 'a';
3   unsigned int y;
4
5   unsigned long int char_size = sizeof(char);
6   unsigned long int x_size    = sizeof x;
7   unsigned long int uint_size = sizeof(unsigned int);
8   unsigned long int y_size    = sizeof(y);
9
10  return 0;
11 }
```

  - Le variabili `char_size` e `x_size` conterranno lo stesso valore dopo l'esecuzione delle istruzioni a riga 5 e 6, rispettivamente. Il contenuto di tali variabili sarà la dimensione in byte del tipo di dato `char`.
  - Allo stesso modo, le variabili `uint_size` e `y_size` conterranno la dimensione in byte del tipo di dato `unsigned int`.
- Da notare nell'esempio che, quando l'argomento di `sizeof` è un nome di variabile, è possibile omettere le parentesi tonde (confronta riga 6 e riga 8). Questo non è possibile se l'argomento è un tipo di dato.

Gli header `limits.h` e `float.h`

- Le proprietà/limiti dei tipi di dati sono disponibili nei file header `limits.h` e `float.h`.
  - Il file `limits.h` contiene informazioni relative ai tipi di dati interi.
  - Il file `float.h` contiene informazioni relative ai tipi di dati in virgola mobile.
- Tali file devono essere inclusi nel file sorgente con la direttiva `#include` per poterne utilizzare il contenuto.

```
1 #include <limits.h>
2 #include <float.h>
```

- Contengono costanti simboliche definite tramite **macro** (le vedremo più avanti) relative alle limitazioni numeriche del calcolatore/compilatore.

limits.h

Alcune macro contenute in `limits.h`.

Macro	Descrizione
`SCHAR_MIN`	Valore minimo per `signed char`
`SCHAR_MAX`	Valore massimo per `signed char`
`UCHAR_MAX`	Valore massimo per `unsigned char`
`CHAR_MIN`	Valore minimo per `char`
`CHAR_MAX`	Valore massimo per `char`
`SHRT_MIN`	Valore minimo per `short int`
`SHRT_MAX`	Valore massimo per `short int`
`USHRT_MAX`	Valore massimo per `unsigned short int`
`INT_MIN`	Valore minimo per `int`
`INT_MAX`	Valore massimo per `int`
`UINT_MAX`	Valore massimo per `unsigned int`
`LONG_MIN`	Valore minimo per `long int`
`LONG_MAX`	Valore massimo per `long int`
`ULONG_MAX`	Valore massimo per `unsigned long int`

float.h

Alcune macro contenute in `float.h`.

Macro	Descrizione
`FLT_MIN`	Valore minimo per `float`
`FLT_MAX`	Valore massimo per `float`
`DBL_MIN`	Valore minimo per `double`
`DBL_MAX`	Valore massimo per `double`
`LDBL_MIN`	Valore minimo per `long double`
`LDBL_MAX`	Valore massimo per `long double`

La funzione `printf()`

- La funzione `printf()` (formatted print) permette di stampare *output formattato* sul terminale.

La funzione `printf()`

- La funzione `printf()` (formatted print) permette di stampare *output formattato* sul terminale.
- Non appartiene direttamente al linguaggio C ma è una *funzione di libreria*. La sua definizione è specificata nell'header `stdio.h`, che è necessario *includere* nel file sorgente ogni volta che si intende utilizzare tale funzione.
- Così come tutte le funzioni nella libreria standard del C, il comportamento della `printf()` è definito dallo standard ISO C89.
- Abbiamo già visto come utilizzare la `printf()` per stampare la stringa `"Hello, World!"` sul terminale.

La funzione `printf()`

- La funzione `printf()` (formatted print) permette di stampare *output formattato* sul terminale.
- Non appartiene direttamente al linguaggio C ma è una *funzione di libreria*. La sua definizione è specificata nell'header `stdio.h`, che è necessario *includere* nel file sorgente ogni volta che si intende utilizzare tale funzione.
- Così come tutte le funzioni nella libreria standard del C, il comportamento della `printf()` è definito dallo standard ISO C89.
- Abbiamo già visto come utilizzare la `printf()` per stampare la stringa `"Hello, World!"` sul terminale.
- Più nel dettaglio, `printf()` può avere un numero variabile di argomenti. Il primo argomento è una *stringa di formato* che può contenere caratteri speciali, chiamati **specificatori di formato**.
- Gli specificatori di formato sono sequenze speciali di caratteri che iniziano sempre con il simbolo `%` e che indicano le modalità di scrittura del dato sul terminale.
- I dati relativi agli specificatori di formato dichiarati nella stringa sono passati come ulteriori argomenti alla funzione.

Specificatori di formato della funzione `printf()`

- Ci sono numerosi specificatori di formato in C (vedi lezione di laboratorio). Alcuni esempi:

Specificatore	Tipi di dato gestito
`%c`	carattere (`char`)
`%d o %i`	intero con segno (`int, signed int, char ..`)
`%u`	intero senza segno (`unsigned int, unsigned char`)
`%l`	doppia precisione (`long`)
`%f`	virgola mobile (`float, double`)
`%e`	notazione esponenziale (`float, double`)
`%s`	stringhe

- Esempio di stampa della stringa `"Hello, World!"` utilizzando lo specificatore di formato per le stringhe:

```
1 #include<stdio.h>
2
3 int main() {
4  printf("%s\n","Hello, World!");
5
6  return 0;
7 }
```

- Nella stringa di formato della `printf()` è possibile utilizzare le **sequenze di escape** (come `'\n'`, `'\t'`) per formattare l'output (vedi "Costanti carattere" nelle slide precedenti).

Esempio: stampa del contenuto di variabili e costanti numeriche

Stampa il contenuto delle variabili dichiarate nel `main()` e della costante numerica 3.14, utilizzando correttamente gli specificatori di formato.

```
1  #include <stdio.h>
2
3  int main() {
4   char symbol = 'A';
5   float height = 1.75;
6   int x = -1;
7   int X;
8   double temp = 100, pressure = 1.5;
9
10  printf("Value of \"symbol\" variabile :\t%c\n",symbol);
11  printf("Value of \"heigth\" variabile :\t%f\n",height);
12  printf("Value of \"x\" variabile :\t%d\n",x);
13  printf("Value of \"X\" variabile :\t%d\n",X);
14  printf("Value of \"temp\" variabile :\t%f\n",temp);
15  printf("Value of \"pressure\" variabile :\t%f\n",pressure);
16  printf("Value of \"Pi\" constant :\t%f\n",3.14);
17
18  return 0;
19 }
```

- La variabile `X` (riga 7) non è inizializzata, pertanto il suo contenuto non è predicibile a priori.
- Il carattere " nella stringa di formato deve essere indicato come sequenza di escape (`'\"'`).
- La sequenza di escape `'\t'` inserisce uno spazio di tabulazione dopo : in modo da allineare i valori stampati.

Esempio: overflow

Stampa il contenuto della variabile `x` di tipo `short int` dopo una istruzione che la manda in overflow.

```c
#include <stdio.h>

int main() {
 short int x = 4294967295;
 printf("The value of x variable is: %d\n",x);
 return 0;
}
```

Esempio: sizeof

Stampa la dimensione in byte dei tipi di dato `char` e `unsigned int`.

```c
#include<stdio.h>

int main() {
 char x = 'a';
 unsigned int y;

 unsigned long int char_size = sizeof(char);
 unsigned long int x_size    = sizeof x;
 unsigned long int uint_size = sizeof(unsigned int);
 unsigned long int y_size    = sizeof(y);

 printf("char_size = %lu byte,  x_size = %lu byte\n",char_size,x_size);
 printf("uint_size = %lu bytes, y_size = %lu bytes\n",uint_size,y_size);
 return 0;
}
```

Esempio: limiti e occupazione di memoria dei tipi di dato base

Stampa valore minimo/massimo e occupazione di memoria in byte dei tipo di dato `char`, `int`, `float`, `double`.

```c
#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(){
 printf("Type: %-7s Min: %-13d Max: %-13d ","char",CHAR_MIN,CHAR_MAX);
 printf("Bytes: %lu\n",sizeof(char));
 printf("Type: %-7s Min: %-13d Max: %-13d ","int",INT_MIN,INT_MAX);
 printf("Bytes: %lu\n",sizeof(int));
 printf("Type: %-7s Min: %-13e Max: %-13e ","float",FLT_MIN,FLT_MAX);
 printf("Bytes: %lu\n",sizeof(float));
 printf("Type: %-7s Min: %-13e Max: %-13e ","double",DBL_MIN,DBL_MAX);
 printf("Bytes: %lu\n",sizeof(double));

 return 0;
}
```

Da notare:

- inclusione dei file header `limits.h` e `float.h`;
- opzioni di formattazione per definire la dimensione di stampa. Ad esempio, **%-7**s stampa una stringa usando almeno 7 caratteri (aggiunge spazi bianchi se la stringa è più corta) e la allinea a sinistra (-).