



Programmazione I

Il Linguaggio C

Elementi Fondamentali

Daniel Riccio

Università di Napoli, Federico II

6 ottobre 2021



Sommario



- Argomenti
 - Il Linguaggio C
 - Gli identificatori
 - i Tipi primitivi
 - Gli operatori

Il Linguaggio C



Sviluppato tra il 1969 ed il 1973 presso gli AT&T Bell Laboratories (Ken Thompson, B. Kernighan, Dennis Ritchie)

- Per uso interno
- Legato allo sviluppo del sistema operativo Unix



Ken
Thompson



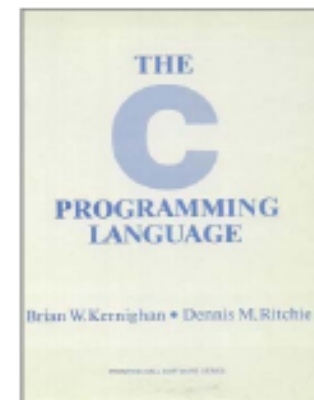
Brian
Kernighan



Dennis
Ritchie

Nel 1978 viene pubblicato “The C Programming Language”, prima specifica ufficiale del linguaggio

- Detto “K&R”



Il Linguaggio C



Il C è un linguaggio:

- **Imperativo** ad alto livello
... ma anche poco astratto
- **Strutturato**
... ma con eccezioni
- **Tipizzato**
Ogni oggetto ha un tipo
- **Elementare**
Poche keyword
- **Case sensitive**
Maiuscolo diverso da minuscolo negli identificatori!
- **Portabile**
- **Standard ANSI**

Un programma viene inteso come un insieme di istruzioni, ciascuna delle quali può essere pensata come un "ordine" che viene impartito alla macchina.

Non sono presenti salti incondizionati, del tipo "go to".
Teorema di Böhm-Jacopini:
qualunque algoritmo può essere implementato utilizzando tre sole strutture di controllo: la *sequenza*, la *selezione* ed il *ciclo* (iterazione).

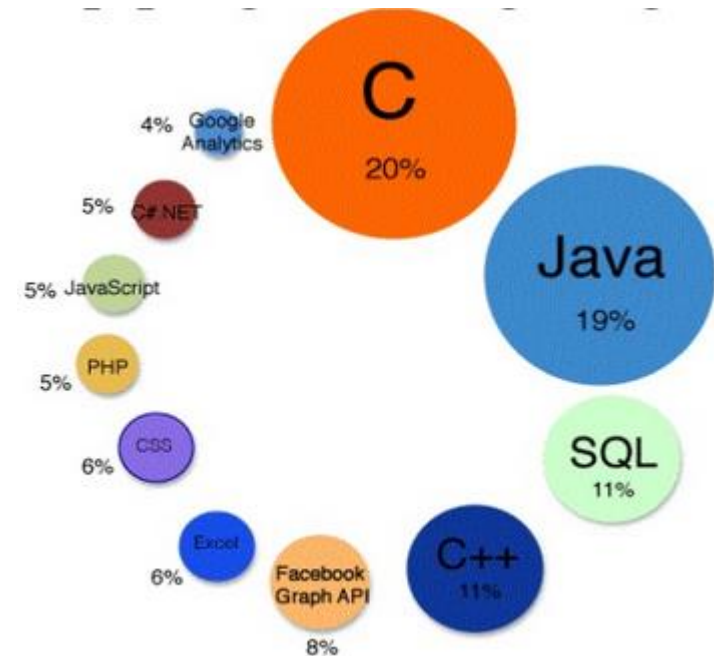


Diffusione attuale



I **linguaggi** attualmente più diffusi al mondo sono:

- C
- C++, un'evoluzione del C
- Java, la cui sintassi è tratta da C++
- C#, estremamente simile a Java e C++



Il **linguaggio C** è uno dei linguaggi più diffusi

La sintassi del linguaggio C è ripresa da tutti gli altri linguaggi principali

Un esempio canonico

Programma

main.c

```
1  /*****
2
3  Esempi per il corso di Programmazione I
4
5  *****/
6  #include <stdio.h>
7
8  int main()
9  {
10     // questo programma stampa il messaggio
11     // "Hello World"
12     printf("Hello World");
13
14     return 0;
15 }
16
```

Esecuzione

Hello World

...Program finished with exit code 0
Press ENTER to exit console.



La struttura di un programma C

Parte dichiarativa globale

`main()`

Parte dichiarativa locale

Parte esecutiva

Programma

main.c

```
1  /*****
2
3  Esempi per il corso di Programmazione I
4
5  *****/
6  #include <stdio.h>
7
8  int main()
9  {
10     // questo programma stampa il messaggio
11     // "Hello World"
12     printf("Hello World");
13
14     return 0;
15 }
16
```

Un **programma** può essere concepito come l'unione di due parti:

- **dichiarativa** – descrive i dati che devono essere manipolati
- **esecutiva** – descrive le azioni che devono essere eseguite

Il pre-processor



La compilazione C passa attraverso un passo preliminare che precede la vera e propria traduzione in linguaggio macchina

Il programma che realizza questa fase è detto pre-processor, la cui funzione principale è l'espansione delle direttive che iniziano con il simbolo **#**

Direttive principali:

#include
#define

Esempio:

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

int main()
{
    // programma vuoto
    return 0;
}
```


Il pre-processor (#include)



Sintassi:

#include <**file**> per includere un file di sistema

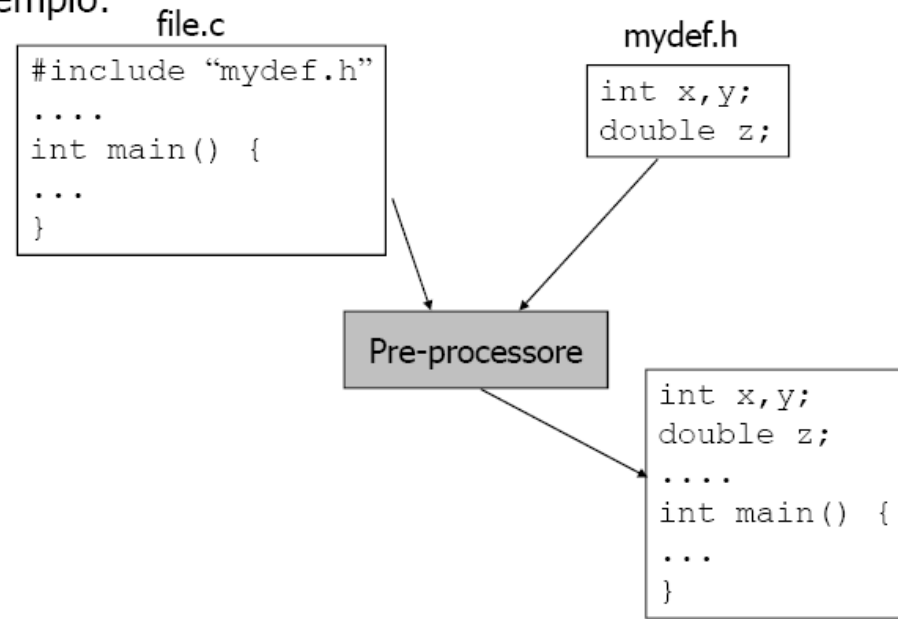
#include "**file**" per includere un file definito dal programmatore

Esempio:

#include "mydef.h"

- Significato:
<**file**> viene espanso ed incluso per intero nel file sorgente

Esempio:



Il pre-processor (#define)



#define <costante> <valore>

- <costante>**: Identificatore della costante simbolica
Convenzionalmente indicato tutto in maiuscolo
- <valore>**: Un valore da assegnare alla costante

- Utilizzo:
 - Definizione di costanti simboliche
 - Maggiore leggibilità
 - Maggiore flessibilità

Il cambiamento del valore della costante si applica a tutto il file!

Esempio:

```
#define PI 3.1415
```

```
...
```

```
double raggio = 4.7432 ;
```

```
double Area = PI *r*r;
```

Funzioni di I/O



In C, le operazioni di I/O non sono gestite tramite vere e proprie istruzioni, bensì mediante opportune funzioni (il concetto di funzione verrà introdotto successivamente)

L'utilizzo di queste istruzioni richiede l'inserimento di una direttiva `#include <stdio.h>` all'inizio del file sorgente

Significato: “includi il file `stdio.h`”, che contiene alcune dichiarazioni, fra le quali, quelle di:

```
printf(<formato>,<arg1>,...,<argn>);
```

e

```
scanf(<formato>,<arg1>,...,<argn>);
```



Funzioni di I/O (printf)

La sintassi di **printf** è:

```
printf(<formato>,<arg1>,...,<argn>);
```

dove

<formato> è una sequenza di caratteri che determina il formato di stampa di ognuno dei vari argomenti

nella forma %<carattere>

%d intero

%u unsigned

%s stringa

%c carattere

%x esadecimale

%o ottale

%f float

%g double

Esempio:

```
int a = 3;
```

```
double b = 2.5;
```

```
...
```

```
printf("Primo valore: %d", a);
```

```
printf("Secondo valore: %f", b);
```

<arg1>,...,<argn> sono le quantità (espressioni) che si vogliono stampare, associate alle direttive di formato nello stesso ordine.



Funzioni di I/O (scanf)

La sintassi di **scanf** è:

```
scanf(<formato>,<arg1>,...,<argn>);
```

dove

<formato> è una sequenza di caratteri che determina il formato di stampa di ognuno dei vari argomenti

nella forma %<carattere>

- %d** intero
- %u** unsigned
- %s** stringa
- %c** carattere
- %x** esadecimale
- %o** ottale
- %f** float
- %g** double

Esempio:

```
int a;
```

```
double b;
```

```
...
```

```
scanf("Primo valore: %d", &a);
```

```
scanf("Secondo valore: %f", &b);
```

<arg1>,...,<argn> sono le variabili, in cui si vogliono memorizzare i valori, associate alle direttive di formato nello stesso ordine.



Sequenze di escape

La tabella ASCII comprende un certo numero di valori ai quali non corrisponde un carattere stampabile.

Per rappresentare questi caratteri si utilizza una **sequenza di escape**, dove il carattere **backslash** (\), detto appunto carattere di escape, assume il ruolo di segnalatore.

Sequenza di escape	Rappresenta
\a	Segnale acustico (avviso)
\b	BACKSPACE
\f	Modulo continuo
\n	Nuova riga
\r	Ritorno a capo
\t	Tabulazione orizzontale
\v	Tabulazione verticale
\'	Virgoletta singola
\"	Virgolette doppie



Esempio di uso delle Funzioni di I/O

main.c

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int b;
7     int h;
8
9     int prodotto;
10    int area;
11
12    // Richiediamo i dati di input all'utente
13    printf("Inserisci la base del triangolo: ");
14    scanf("%d", &b);
15    printf("Inserisci l'altezza del triangolo: ");
16    scanf("%d", &h);
17
18    // Calcoliamo il prodotto b*xh
19    prodotto = b*h;
20
21    // Calcoliamo l'area dividendo il prodotto per 2
22    area = prodotto/2;
23
24    // Mostriamo a video il risultato
25    printf("L'area del triangolo è: %d", area);
26
27    return 0;
28 }
```

Direttiva che include printf e scanf

Variabili per memorizzare i dati

Uso di printf e scanf per l'inserimento dei dati

Operazioni matematiche per il calcolo del risultato

Uso di printf per la stampa del risultato

Esempio di uso delle Funzioni di I/O



main.c

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int b;
7     int h;
8
9     int prodotto;
10    int area;
11
12    // Richiediamo i dati di input all'utente
13    printf("Inserisci la base del triangolo: ");
14    scanf("%d", &b);
15    printf("Inserisci l'altezza del triangolo: ");
16    scanf("%d", &h);
17
18    // Calcoliamo il prodotto bxh
19    prodotto = b*h;
20
21    // Calcoliamo l'area dividendo il prodotto per 2
22    area = prodotto/2;
23
24    // Mostriamo a video il risultato
25    printf("L'area del triangolo è: %d", area);
26
27    return 0;
28 }
```

Esecuzione:

```
Inserisci la base del triangolo: 
```

```
Inserisci la base del triangolo: 4
Inserisci l'altezza del triangolo: 
```

```
Inserisci la base del triangolo: 4
Inserisci l'altezza del triangolo: 8
L'area del triangolo è: 16
```

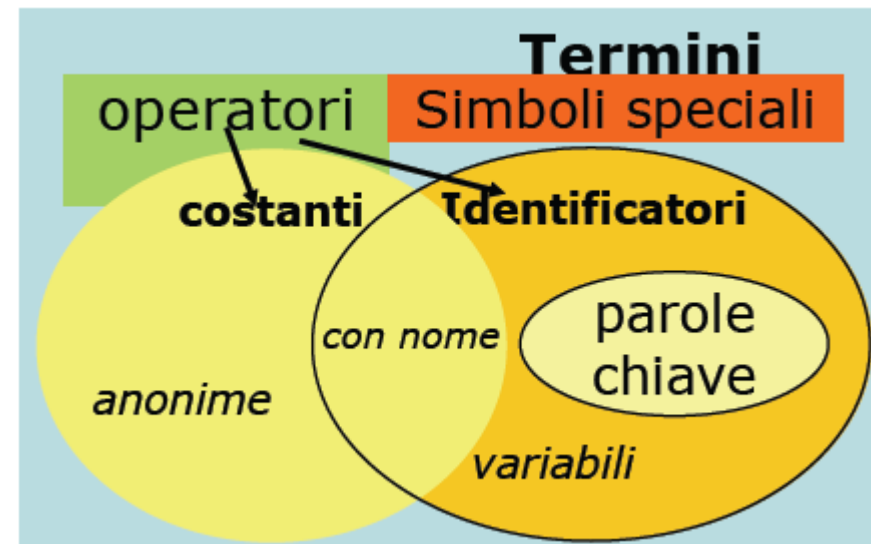

Elementi del linguaggio C



Da un punto di vista lessicale, un programma è una sequenza di Termini, detti **tokens**. Il compilatore deve riconoscere i termini del linguaggio per le successive fasi di analisi.

Tipi di termini

- **Identificatori**
- **Parole chiave**
- **Costanti**
- **Espressioni**
- **Operatori**
- **Simboli o segni speciali**



Sono ignorati

- Spazi bianchi, tabulatori, newlines, e commenti

Gli identificatori

Identificatore è un termine usato dal programmatore per indicare funzioni, variabili, oggetti, costanti, etc.

Ogni identificatore è formato da una sequenza di caratteri di tipo **lettere** o **cifre** o “_” (*underscore*)

- Il primo carattere deve essere una **lettera** o _
- Caratteri maiuscoli e minuscoli sono diversi
- 31 caratteri
- Identificatori non validi:
 - **un amico** (contiene uno spazio)
 - **un'amica** (contiene un apostrofo)
 - **piano*forte** (contiene un simbolo)
 - **for** (è una *parola-chiave del C*)

Gli identificatori devono **ricordare mnemonicamente** gli oggetti cui si riferiscono



Gli identificatori

La **parole chiave** è un termine che ha un significato particolare per il compilatore C.

Possono essere adoperati dal programmatore solo come previsto dal linguaggio

Esempi di parole chiave sono:

- **main**: indica che il testo che segue tra parentesi graffe rappresenta il codice sorgente del programma.
- **const**: definisce il nome che segue come dato costante.
- **float**: definisce il nome seguente come variabile a virgola mobile (singola precisione).
- **if, then, else**: definisce costrutti di controllo del linguaggio
- ...

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Le parole chiave sono indicate in grassetto

I dati numerici, le variabili e le costanti



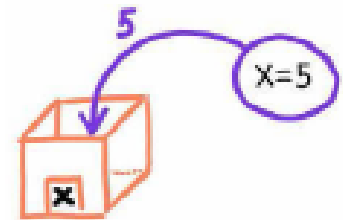
Sono quelli più usati in ambito scientifico nei moderni sistemi di elaborazione ... tutti gli altri tipi di dato sono trasformati in dati numerici

In **matematica**, una variabile è un carattere alfabetico che rappresenta un numero arbitrario, sconosciuto, o non completamente specificato.

$$2x = 3;$$

$$x = \frac{3}{2};$$

In **informatica**, una **variabile** è una porzione di memoria destinata a contenere dei dati, che potranno essere **letti o modificati** durante l'esecuzione di un programma.



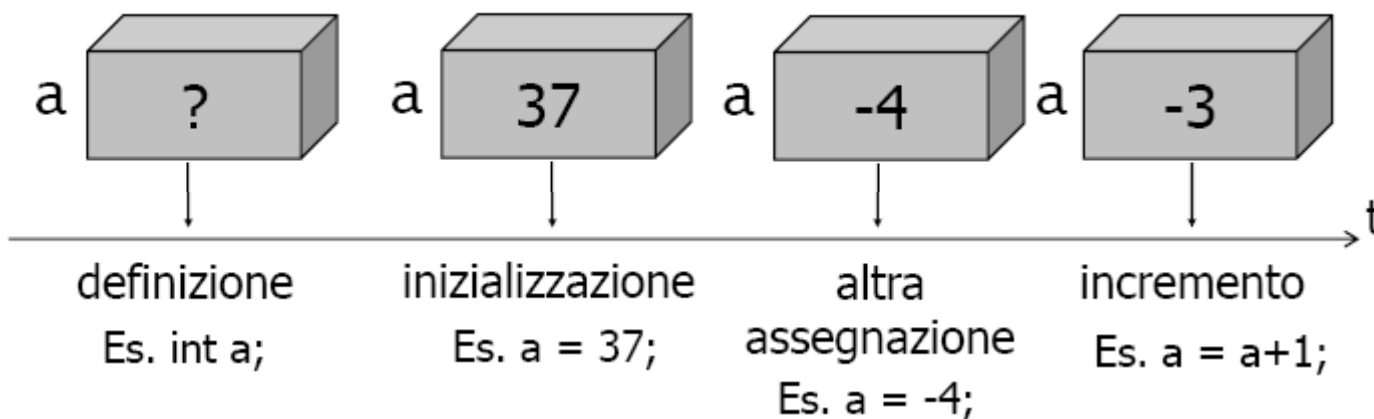
In **informatica**, una **costante** è una porzione di memoria destinata a contenere dei dati, che potranno essere **solo letti e non modificati** durante l'esecuzione di un programma.

Il contenuto di una variabile



Ogni variabile, in ogni istante di tempo, possiede un certo Valore

- Le variabili appena definite hanno valore ignoto
 - Variabili non inizializzate
- In momenti diversi il valore può cambiare



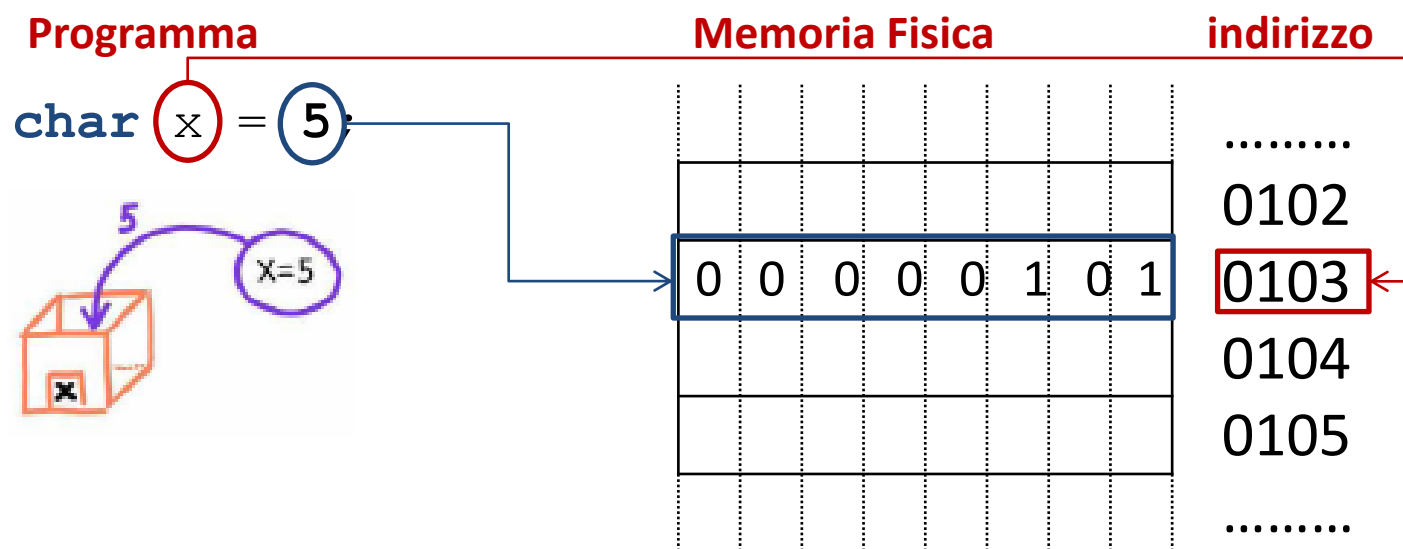


I dati numerici, le variabili e le costanti

In C, tutti i dati devono essere **dichiarati** prima di essere utilizzati!

La dichiarazione di un dato richiede:

- L'**allocazione di uno spazio in memoria atto a contenere il dato**
- L'**assegnazione di un nome a tale spazio in memoria**

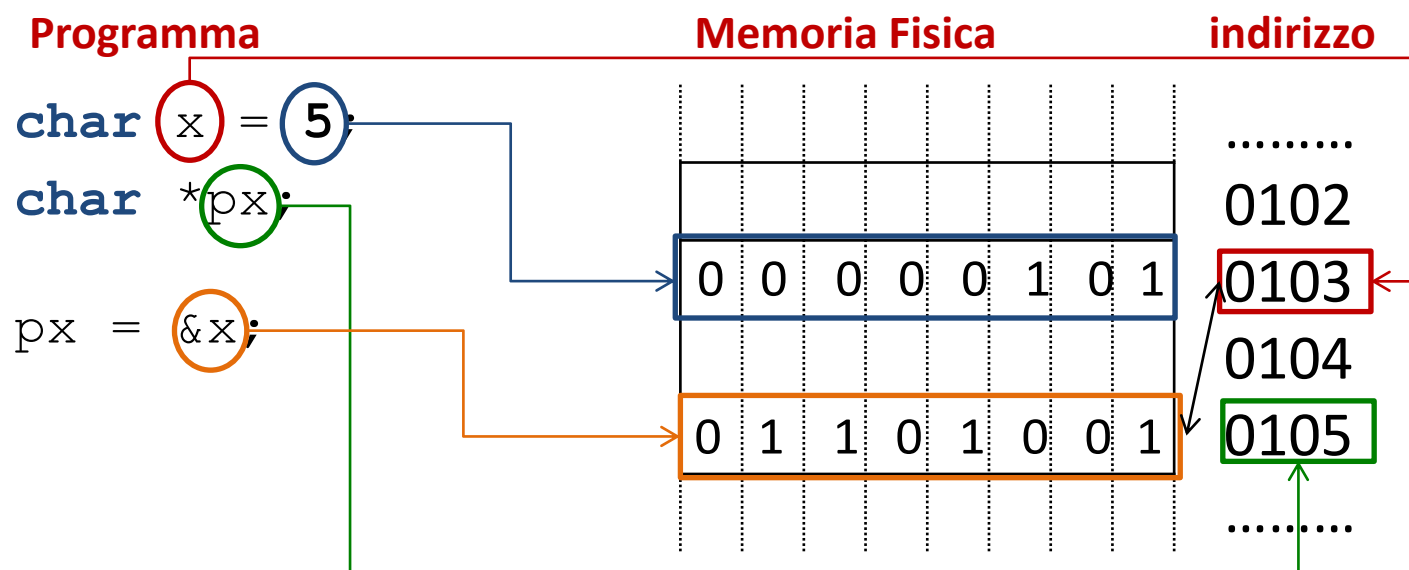


Ad ogni istanza di **x** nel programma viene associato contenuto della cella di memoria, ovvero **5**
È possibile conoscere l'indirizzo della cella di memoria associata a **x**, scrivendo **&x**

Il puntatore



L'**indirizzo** della cella, in cui è memorizzata la variabile **x** è esso stesso un dato numerico e può essere a sua volta memorizzato in una variabile. Una variabile che contiene l'indirizzo di una cella di memoria prende il nome di puntatore alla variabile **x**.



Ad ogni istanza di **x** nel programma viene associato contenuto della cella di memoria, ovvero 5
È possibile conoscere l'indirizzo della cella di memoria associata a **x**, scrivendo **&x**



Dichiarazione di identificatori

In C, dichiarare un dato significa, specificarne:

- **Nome** - definisce un identificativo unico per la variabile
- **Tipo** - l'insieme dei valori che può assumere e l'insieme delle operazioni che possono applicarsi a tali valori
- **Modalità di accesso** - valore corrente della variabile

Esempi di nomi:

a	b	a1	a2	
num	n	N	somma	max
area	perimetro	perim		
n_elementi	Nelementi	risultato		
trovato	nome	risposta		

Tipi di base (primitivi)



Sono quelli forniti direttamente dal C

- char** caratteri ASCII ('a', '%') o interi nell'intervallo **[-128,127]**

- int** interi (complemento a 2) nell'intervallo **[-2.147.483.648 ; 2.147.483.647]**

- float** reali (floating point singola precisione, 7 cifre significative) in valore assoluto in **[1,17 x 10⁻³⁸ e 3,40 x 10³⁸]**

- double** reali (floating point doppia precisione, 15 cifre significative) in valore assoluto in **[1,17 x 10⁻³⁰⁸ e 1,79 x 10³⁰⁸]**

La dimensione precisa di questi tipi dipende dall'architettura (non definita dal linguaggio), tuttavia la dimensione di un **char** è sempre **8 bit** (1 Byte).

Tipi di base (primitivi)



La **dichiarazione di tipo** informa il compilatore sul tipo assegnato ad un identificatore e ha come forma generale

Dichiarazione di tipo di un identificatore

specificatore ... qualificatore ... tipo ident

Fornisce altre caratteristiche dell'identificatore

Indica come deve essere allocato il contenuto

tipo standard

Qualificatori: short, long, signed, unsigned

Specificatori: const, extern, static, volatile,...

Interi

- [signed/unsigned] short [int]
- [signed/unsigned] int
- [signed/unsigned] long [int]

La **definizione di tipo** è una dichiarazione che comporta l'allocazione di un'area di memoria per l'identificatore ma non l'inizializzazione del suo contenuto (indefinito)



Dimensione in byte di un tipo (sizeof)

L'operatore **sizeof()** calcola il numero di byte utilizzato dai tipi di dato di base

Sintassi:

sizeof (<tipo>)

Restituisce il numero di byte occupato da <tipo>

Esempio:

```
unsigned int size;  
size = sizeof(float); /* size = 4 */
```

L'uso dell'operatore **sizeof()** può essere esteso al calcolo dello spazio occupato da espressioni, vettori e strutture

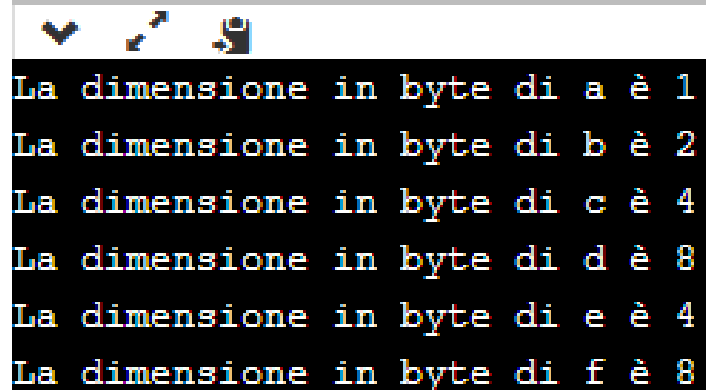
Esempio di uso di sizeof()



main.c

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     // Dichiariamo tutte le variabili
7     char a;
8     short int b;
9     int c;
10    long int d;
11    float e;
12    double f;
13
14    // Stampiamo la dimensione in byte
15    // di ciascuna variabile
16    printf("La dimensione in byte di a è %d\n", sizeof(a));
17    printf("La dimensione in byte di b è %d\n", sizeof(b));
18    printf("La dimensione in byte di c è %d\n", sizeof(c));
19    printf("La dimensione in byte di d è %d\n", sizeof(d));
20    printf("La dimensione in byte di e è %d\n", sizeof(e));
21    printf("La dimensione in byte di f è %d\n", sizeof(f));
22
23    return 0;
24 }
```

Esecuzione:



```
La dimensione in byte di a è 1
La dimensione in byte di b è 2
La dimensione in byte di c è 4
La dimensione in byte di d è 8
La dimensione in byte di e è 4
La dimensione in byte di f è 8
```

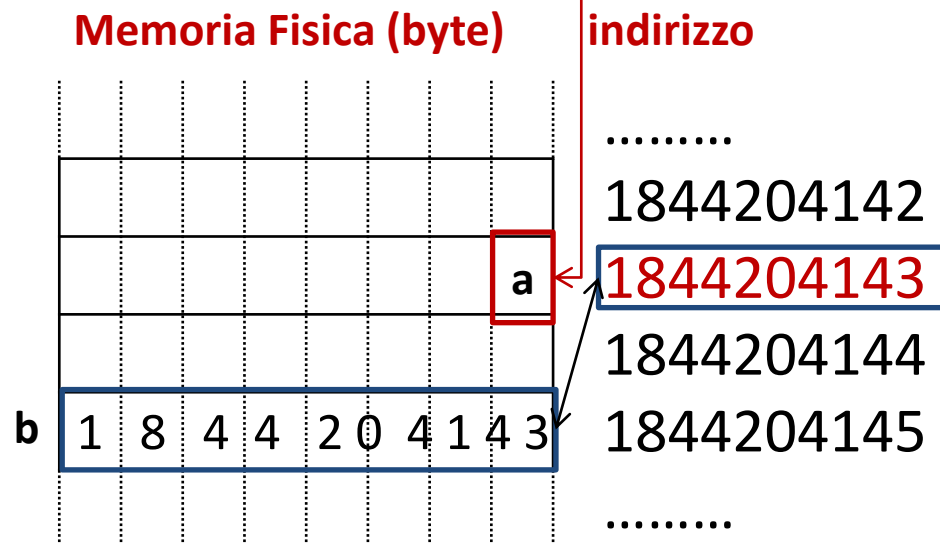
Esempio di uso di sizeof()



```
main.c
1
2 #include <stdio.h>
3
4 int main()
5 {
6     // Dichiariamo le variabili
7     char a 56;
8     char b;
9
10    // Assegnamo al puntatore b l'indirizzo della
11    // cella di memoria che contiene a
12    b = &a;
13
14    // Stampiamo il valore di a
15    printf("Il valore di a è %d\n", a);
16
17    // Stampiamo l'indirizzo della cella che contiene a
18    printf("L'indirizzo di a è %d\n", b);
19
20    // Stampiamo il contenuto della cella che
21    // contiene a
22    printf("Il valore di a è %d\n", *b);
23
24    // Verifichiamo la dimensione del tipo di a
25    printf("In numero di byte per a è %d\n", sizeof(a));
26
27    // Verifichiamo la dimensione del tipo di b
28    printf("In numero di byte per b è %d\n", sizeof(b));
29
30    return 0;
31 }
```

Esecuzione:

```
Il valore di a è 56
L'indirizzo di a è 1844204143
Il valore di a è 56
In numero di byte per a è 1
In numero di byte per b è 8
```



Dichiarazione di identificatori



L'**Inizializzazione** assegna un valore iniziale ad un identificatore già definito

```
ident = valore;
```



Operatore di assegnazione

La dichiarazione di un identificatore deve precedere il suo primo utilizzo, e può essere in ogni parte del programma

È sempre consigliabile dichiarare tutte le costanti e le variabili **nella parte iniziale** del programma (o della funzione che le utilizza), così è più facile identificare nel programma la **sezione delle dichiarazioni**

Dichiarazione e inizializzazione possono essere combinate

Dichiarazione di identificatori

Dichiarazione di variabile:

```
int numero;
```

Dichiarazione di variabile e successiva inizializzazione

```
int numero;
```

```
...
```

```
numero = 54;
```

Dichiarazione di più variabili dello stesso tipo

```
int numero, secondo_numero, terzo;
```

Dichiarazione di variabile con contestuale inizializzazione

```
int numero = 54;
```

Dichiarazione di più variabili e inizializzazione

```
int numero = 54, secondo_numero, terzo = 15;
```


Dichiarazione di identificatori



main.c

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     // Dichiariamo le variabili
7     char a;
8     int b;
9     float c=2.5;
10    double d=5.6, e, f=3.4;
11
12    // Assegnamo un valore alle variabili non inizializzate
13    a = 'T';
14    b = 32768;
15    e = 3.14159265;
16
17    // Stampiamo il contenuto delle variabili
18    printf("Il contenuto di a è %c\n", a);
19    printf("Il contenuto di b è %d\n", b);
20    printf("Il contenuto di c è %f\n", c);
21    printf("Il contenuto di d è %f\n", d);
22    printf("Il contenuto di e è %f\n", e);
23    printf("Il contenuto di f è %f\n", f);
24    printf("Su una sola riga\na:%c b:%d c:%f d:%f e:%.4f f:%f\n", a, b, c, d, e, f);
25
26    return 0;
27 }
```

Esecuzione:

```
Il contenuto di a è T
Il contenuto di b è 32768
Il contenuto di c è 2.500000
Il contenuto di d è 5.600000
Il contenuto di e è 3.141593
Il contenuto di f è 3.400000
Su una sola riga
a:T b:32768 c:2.500000 d:5.600000 e:3.1416 f:3.400000
```


Simboli speciali



Uno o anche due caratteri consecutivi che sono usati per scopi particolari come ad esempio:

- il **punto e virgola** (;) che serve ad indicare la fine di una istruzione
- le **parentesi graffe** ({ }) che indicano inizio e fine di una istruzione composta
- la **virgola** (,) che è usata come separatore

es:

```
float n, raggio;
```

invece di

```
float n;  
float raggio;
```



Visibilità di un identificatore

Ogni variabile è utilizzabile all'interno di un preciso ambiente di visibilità (**scope**)

- Variabili globali
 - Definite all'esterno del main()
- Variabili locali
 - Definite all'interno del main()
 - Più in generale, definite all'interno di un blocco

Esempio: un blocco di istruzioni è racchiuso fra parentesi graffe

```
{  
    int a = 4;  
  
    a = a + 1;  
    printf("il valore di a è: %d\n", a);  
}
```

un blocco di istruzioni

Visibilità di un identificatore



main.c

```
1
2 #include <stdio.h>
3
4 int a=5;
5
6 int main()
7 {
8     // Dichiariamo le variabili
9     int b=6;
10    int c=7;
11
12    {
13        int b=12; // dichiariamo una nuova variabile b
14
15        // Stampiamo il valore delle variabili nel blocco
16        printf("Il valore di a nel blocco è %d\n", a);
17        printf("Il valore di b nel blocco è %d\n", b);
18        printf("Il valore di c nel blocco è %d\n", c);
19    }
20
21    // Stampiamo il valore delle variabili fuori dal blocco
22    printf("Il valore di a fuori dal blocco è %d\n", a);
23    printf("Il valore di b fuori dal blocco è %d\n", b);
24    printf("Il valore di c fuori dal blocco è %d\n", c);
25
26    return 0;
27 }
```

Esecuzione:

```
Il valore di a nel blocco è 5
Il valore di b nel blocco è 12
Il valore di c nel blocco è 7
Il valore di a fuori dal blocco è 5
Il valore di b fuori dal blocco è 6
Il valore di c fuori dal blocco è 7
```

Le espressioni



Le **Espressioni** rappresentano il valore che si ottiene applicando opportune operazioni ben definite ad uno o più operandi che possono essere costanti o variabili

In una espressione le operazioni vengono indicate con particolari simboli detti operatori

Gli **operatori** possono essere

- **unari**: agiscono su un solo operando **!a**, **++i**
- **binari**: agiscono su due operandi (destro e sinistro) **a+b**, **a<=b**
- **ternari**: agiscono su tre operandi

Gli **operatori** possono essere

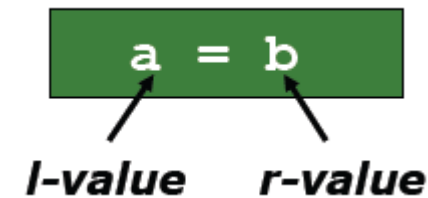
- | | |
|-------------------|---------------------------|
| – di assegnazione | – logici |
| – aritmetici | – incremento e decremento |
| – binari | – condizionali |
| – relazionali | |

Espressioni di assegnazione

L'**operatore di assegnazione** = copia il contenuto dell'operando destro (detto **r-value**) nell'operando sinistro (detto **l-value**)

r-value è una qualsiasi espressione
con valore un tipo standard

l-value è una variabile



Il tipo di **r-value** deve essere lo stesso o implicitamente convertibile nel tipo di **l-value**

Il valore dell'**espressione assegnazione** è **r-value**

Esempio: **a = b + 3**

Espressioni di assegnazione

È possibile combinare l'istruzione di assegnazione con gli operatori aritmetici.

Sintassi:

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

Operatori:

```
+=    -=    *=    /=    %=
```

– Significato: assegnazione + operazione.

Esempi:

```
x += 5;
```

```
/* equivalente a x = x + 5 */
```

```
y -= x;
```

```
/* equivalente a y = y - x */
```

Operazioni di incremento e decremento



Per le assegnazioni composte più comuni sono previsti degli operatori espliciti:

++	--
----	----

Significato:

++ ➡ +=1

-- ➡ -=1

Esempi:

`x++;` `/* equivale a x = x + 1 */`

`valore--;` `/* equivale a valore = valore - 1 */`

Operazioni di incremento e decremento



Possono essere utilizzati sia in notazione **prefissa** che in notazione **postfissa**

Prefissa: la variabile viene modificata prima di essere utilizzata nell'espressione

Postfissa: la variabile viene modificata solo dopo averla utilizzata nell'espressione

Esempio:

assumendo $x = 4$:

Se si esegue $y = x++$, si otterrà come risultato $x = 5$ e $y = 4$;

Se si esegue $y = ++x$, si otterrà come risultato $x = 5$ e $y = 5$;



Espressioni aritmetiche

Gli **operatori aritmetici** eseguono le principali operazioni matematiche (somma) **+**, (differenza) **-**, (moltiplicazione) *****, (divisione) **/**

Se la divisione è fra due numeri interi, il risultato dell'operazione è ancora un numero intero (troncamento).

Esempio: **27 / 4** dà come risultato **6** (anziché **6.75**).

Il resto di una divisione fra numeri interi si calcola con l'operatore binario **%**. Ad esempio, **27 % 4** dà come risultato **3**

Operazione	Simbolo algebrico	Simbolo in C	Espressione algebrica	Espressione in C
Addizione	+	+	$a+b$	$a+b$
Sottrazione	-	-	$a-b$	$a-b$
Moltiplicazione	x	*	ab	$a*b$
Divisione	:	/	$a:b$	a/b
Modulo	mod	%	$a \bmod b$	$a \% b$

Regole di precedenza



$$5 + 3 * 2$$

Risultato = 16 o 11?

1. Svolgere le parentesi
2. Moltiplicazione, divisione e modulo
3. Addizione e sottrazione

Più operazioni dello stesso tipo vanno risolte da sinistra a destra.

Esempio:

$$5 + 4 - 3 * 4 / 2$$

$$5 + 4 - 12 / 2$$

$$5 + 4 - 6$$

$$9 - 6 = 3$$



Tipo di una espressione

Se le costanti e le variabili sono tutte dello stesso tipo allora anche il valore dell'espressione sarà dello stesso tipo.

Se tutte le grandezze presenti nell'espressione sono di tipo numerico, anche se diversi, sarà il compilatore ad effettuare tutte le opportune conversioni di tipo.

Esempio:

int n = 3;

float f = 2.5;

double d = 4.8;

$(n+f)*d$ di che tipo è?

n è convertito a **float**,

la somma $(n+f)$ è convertita a **double**

$(n+f)*d$ è di tipo **double**

Regola: Non adoperare espressioni in cui sono presenti variabili di tipo diverso.

Operatori relazionali



Gli **operatori relazionali** sono:

- > (strettamente maggiore)
- >= (maggiore o uguale)
- < (strettamente minore)
- <= (minore o uguale)
- == (uguale)
- != (diverso)

Eseguono il confronto fra i valori dei due operandi (di qualsiasi tipo standard) e restituiscono un valore booleano (vero/falso):

<code>a > b</code>	restituisce true se <code>a</code> é maggiore di <code>b</code>
<code>a >= b</code>	restituisce true se <code>a</code> é maggiore o uguale a <code>b</code>
<code>a < b</code>	restituisce true se <code>a</code> é minore di <code>b</code>
<code>a <= b</code>	restituisce true se <code>a</code> é minore o uguale a <code>b</code>
<code>a == b</code>	restituisce true se <code>a</code> é uguale a <code>b</code>
<code>a != b</code>	restituisce true se <code>a</code> é diverso da <code>b</code>

Esempio:

bool bvar = (**7** > **3**);
(in bvar viene memorizzato **true**)

bool bvar = (**7** == **3**);
(in bvar viene memorizzato **false**)

Esercizi



Scrivere dei programmi che calcolano e stampano a video:

- 1) Area e perimetro di un rettangolo dati i due lati
- 2) Area del trapezio date le due basi e l'altezza
- 3) Circonferenza e area del cerchio dato il raggio

Bonus:

Scrivere un programma che prende in input tre valori interi positivi e stampa a video il valore massimo.

Scrivere un programma che stampa a video l'architettura a N bit del processore su cui esso viene eseguito (es. 32, 64, ...)