Elementi Di Informatica E Programmazione

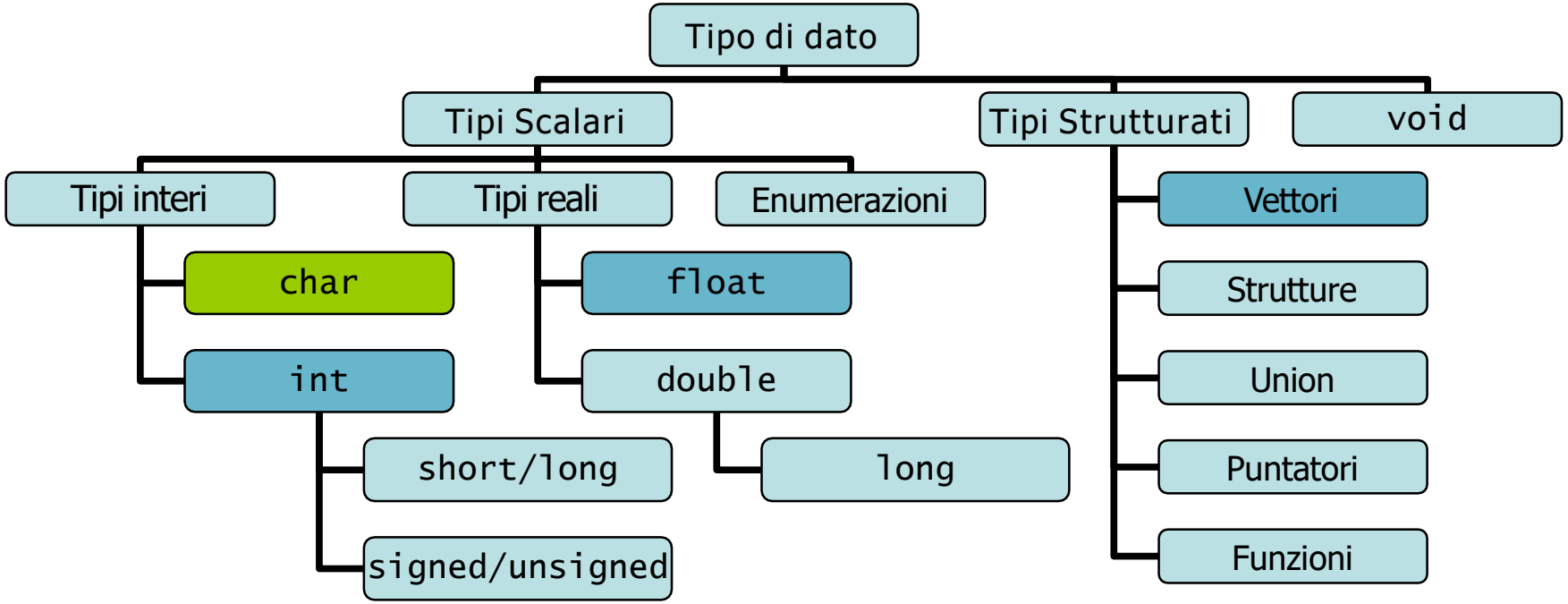
Prof. Andrea Loreggia

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Tipi di dato testuali

- I programmi visti finora erano in grado di elaborare esclusivamente informazioni numeriche
- Numeri interi (int), numeri reali (float) Variabili singole o vettori
- In molti casi è necessario elaborare informazioni di tipo testuale
- Vuoi continuare (s/n)?
- Conta le parole di un testo scritto
- Gestisci una rubrica di nomi e numeri di telefono
- …

Il sistema dei tipi C



Rappresentazione dei testi

- Il calcolatore è in grado di rappresentare i caratteri alfabetici, numerici ed i simboli speciali di punteggiatura
- Ad ogni diverso carattere viene assegnato, convenzionalmente, un codice numerico corrispondente
- Il programma in C lavora sempre con i codici numerici
- Le funzioni di input/output sono in grado di accettare e mostrare i caratteri corrispondenti

Codice ASCII

UNIVERSITÀ DEGLI STUDI DI BRESCIA

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.lookuptables.com

Caratteri e stringhe

- Il codice ASCII permette di rappresentare un singolo carattere

y	7	W	!	%

- Nelle applicazioni pratiche spesso serve rappresentare sequenze di caratteri: stringhe

F	u	l	v	i	o

0	6	A	Z	N

0	1	1	-	5	6	4	6	3	3	2

Dualità caratteri - numeri

- Ogni carattere è rappresentato dal suo codice ASCII

y	7	W	!	%
121	55	87	33	37

- Ogni stringa è rappresentata dai codici ASCII dei caratteri di cui è composta

F	u	l	v	i	o
70	117	108	118	105	111

0	6	A	Z	N
48	54	65	90	78

0	1	1	-	5	6	4	6	3	3	2
48	49	49	45	53	54	52	54	51	51	50

Caratteri in C

- Ogni carattere viene rappresentato dal proprio codice ASCII
- Sono sufficienti 7 bit per rappresentare ciascun carattere
  - Il C usa variabili di 8 bit (1 byte)
- Non sono previste le lettere accentate né altri simboli diacritici
  - Richiedono estensioni speciali e librerie specifiche

Codice ASCII

UNIVERSITÀ DEGLI STUDI DI BRESCIA

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.lookuptables.com

Caratteristiche del codice ASCII

- Le lettere maiuscole sono tutte consecutive, ed in ordine alfabetico
- Le lettere minuscole sono tutte consecutive, ed in ordine alfabetico
- Le lettere maiuscole vengono "prima" delle minuscole
- Le cifre numeriche sono tutte consecutive, in ordine dallo 0 al 9
- I simboli di punteggiatura sono sparsi

Caratteri di controllo

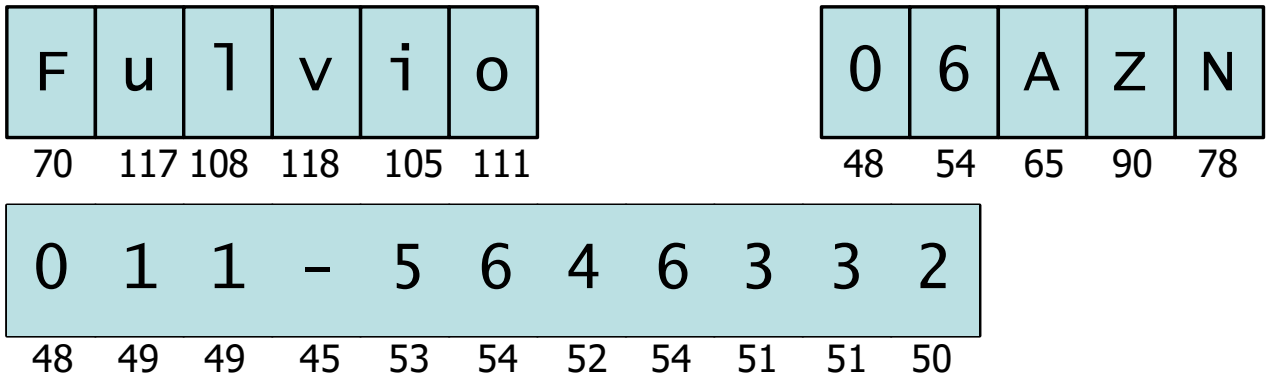
- Caratteri speciali, non visualizzabili
- Rappresentano comandi di stampa, e non simboli da stampare
- Esempi:
  - 7 – BEL: emetti un "bip"
  - 8– BS: cancella l'ultimo carattere
  - 10 – LF: avanza di una riga
  - 13 – CR: torna alla prima colonna
  - 27 – ESC: tasto "Esc"
- Per alcuni esiste una sequenza di escape in C: `\n`

Stringhe

- Una stringa è una struttura dati capace di memorizzare sequenze di caratteri
- In C non esiste un tipo di dato specifico
- Si usano vettori di caratteri
- La lunghezza di una stringa è tipicamente variabile durante l'esecuzione del programma
  - Occorrerà gestire l'occupazione variabile dei vettori di caratteri

Caratteristiche delle stringhe

- Memorizzate come singoli caratteri, ma il loro significato è dato dall'intera sequenza di caratteri
- Lunghezza variabile
- Mix di lettere/cifre/punteggiatura/spazi
- Solitamente non contengono caratteri di controllo



Manipolazione delle stringhe

- Occorre trattare l'insieme di caratteri memorizzato nel vettore come un'unica "variabile"
- Ogni operazione elementare sulle stringhe coinvolgerà tipicamente dei cicli che scandiscono il vettore
- Molte funzioni di libreria sono già disponibili per compiere le operazioni più frequenti ed utili

Errore frequente

- Non confondere una stringa composta da cifre numeriche con il valore decimale associato a tale sequenza

`int`

137

`char`

1	3	7
49	51	55

Variabili char

- I caratteri in C si memorizzano in variabili di tipo `char`

```
char lettera ;
```

- Le costanti di tipo `char` si indicano ponendo il simbolo corrispondente tra singoli apici

```
lettera = 'Q' ;
```

Apici

- Non confondere i 3 tipi di apici presenti sulla tastiera:

Apice singolo (apostrofo)	'	In C, delimita singoli caratteri
Apice doppio (virgolette)	"	In C, delimita stringhe di caratteri
Apice rovesciato (accento grave)	`	Non utilizzato in C

Dualità dei char

- Sintatticamente, i char non sono altro che degli int di piccola dimensione
- Ogni operazione possibile su un int, è anche possibile su un char
- Ovviamente solo alcune di tali operazioni avranno senso sull'interpretazione testuale (ASCII) del valore numerico

Esempi

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
int i ;
char c ;

c = 'A' ;
```

Esempi


```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
```

Esempi


```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
```

Esempi


```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
```

Esempi


```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
```

Esempi


```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
if (c == 'Z') ...
```

Esempi

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
if (c == 'Z') ...
for( c='A'; c<='Z'; c++) ...
```

Caratteri speciali

- Per alcuni caratteri di controllo il linguaggio C definisce una particolare sequenza di escape per poterli rappresentare

C	ASCII	Significato
`'\n'`	LF – 10	A capo
`'\t'`	TAB – 9	Tabulazione
`'\b'`	BS – 8	Backspace – cancella ultimo car.
`'\a'`	BEL – 7	Emette un "bip"
`'\r'`	CR – 13	Torna alla prima colonna

Punteggiatura speciale in C

- Alcuni caratteri hanno un significato particolare dentro gli apici. Per poterli inserire come carattere esistono apposite sequenze di escape

C	ASCII	Significato
'\\'	\	Immette un backslash
'\''	'	Immette un apice singolo
'\"'	"	Immette un apice doppio
'\ooo'	ooo	Immette in carattere ASCII con codice (ottale) ooo
'\xhh'	hh	Immette in carattere ASCII con codice (esadecimale) hh

Input/output di char

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Esistono due insiemi di funzioni che permettono di leggere e stampare variabili di tipo char:
  - Le funzioni `printf/scanf`, usando lo specificatore di formato `"%c"`
  - Le funzioni `putchar` e `getchar`
- In entrambi i casi è sufficiente includere la libreria `<stdio.h>`
- È possibile mescolare liberamente le due famiglie di funzioni

Stampa di caratteri


```
char ch ;

printf("%c", ch) ;
```

```
char ch ;

putchar(ch) ;
```

Lettura di caratteri


```
char ch ;

scanf("%c", &ch) ;
```

```
char ch ;

ch = getchar() ;
```

Suggerimenti (1/2)

UNIVERSITÀ DEGLI STUDI DI BRESCIA

- La funzione printf è più comoda quando occorre stampare altri caratteri insieme a quello desiderato
- printf("La risposta e': %c\n", ch) ; printf("Codice: %c%d\n", ch, num) ;
- La funzione putchar è più comoda quando occorre stampare semplicemente il carattere
- for(ch='a'; ch<='z'; ch++) putchar(ch) ;

Suggerimenti (2/2)

UNIVERSITÀ DEGLI STUDI DI BRESCIA

- La funzione `getchar` è generalmente più comoda in tutti i casi

```
printf("Vuoi continuare (s/n)? ");
ch = getchar() ;
```

Bufferizzazione dell'input-output

- Tutte le funzioni della libreria <stdio.h>
- gestiscono l'input-output in modo bufferizzato
- Per maggior efficienza, i caratteri non vengono trasferiti immediatamente dal programma al terminale (o viceversa), ma solo a gruppi
- È quindi possibile che dopo una putchar, il carattere non compaia immediatamente sullo schermo
- Analogamente, la getchar non restituisce il carattere finché l'utente non preme invio

Conseguenza pratica

```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

Il programma stampa l'invito ad inserire un dato

Conseguenza pratica


```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: _
```

`getchar` blocca il programma in attesa del dato

Conseguenza pratica


```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: a_
```

L'utente immette `'a'`, il programma non lo riceve

Conseguenza pratica


```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: a
_
```

L'utente immette Invio, il programma prosegue

Conseguenza pratica

```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: a
_
```

Ora ch='a', il programma fa un'altra getchar()

Conseguenza pratica

```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: a
_
```

Il programma non si blocca in attesa dell'utente

Conseguenza pratica


```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: a
_
```

C'era già un carattere pronto: Invio! ch2='\n'

Consigli pratici

- Ricordare che l'utente deve sempre premere Invio, anche se il programma richiede un singolo carattere
- Ricordare che, se l'utente inserisce più di un carattere, questi verranno restituiti uno ad uno nelle getchar successive
- Ricordare che l'Invio viene letto come tutti gli altri caratteri

Soluzione proposta


```
char ch, temp ;

printf("Dato: ");

ch = getchar() ; /* leggi il dato */

/* elimina eventuali caratteri successivi
ed il \n che sicuramente ci sarà */
do {
     temp = getchar() ;
} while (temp != '\n') ;
```

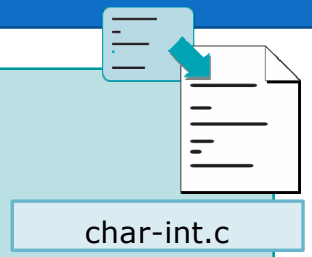
Operazioni sui char

- Le operazioni lecite sui `char` derivano direttamente dalla combinazione tra
  - Le operazioni permesse sugli `int`
  - La disposizione dei caratteri nella tabella ASCII
  - Le convenzioni lessicali della nostra lingua scritta

Conversione ASCII-Carattere

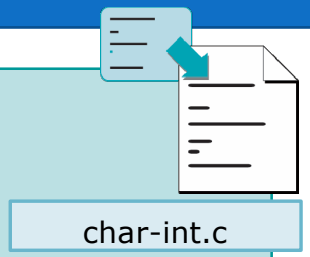
- Una variabile di tipo `char` è allo stesso tempo
  - Il valore numerico del codice ASCII del carattere
    - `printf("%d", ch) ;`
    - `i = ch ;`
    - `ch = j ;`
    - `ch = 48 ;`
  - Il simbolo corrispondente al carattere ASCII
    - `printf("%c", ch) ;`
    - `putchar(ch) ;`
    - `ch = 'Z' ;`
    - `ch = '4' ;`

Esempio (1/3)



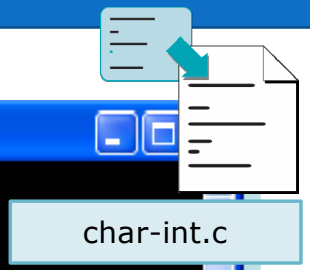

```
int i ;
char ch ;

printf("Immetti codice ASCII (32-126): ");

scanf("%d", &i) ;

ch = i ;

printf("Il carattere %c ha codice %d\n",
        ch, i) ;
```

Esempio (2/3)




```
printf("Immetti  un carattere: ") ;
ch = getchar() ;

while( getchar() != '\n' )
    /**/ ;

i = ch ;

printf("Il carattere %c ha codice %d\n",
        ch, i) ;
```

Esempio (3/3)



Prompt dei comandi

```
Immetti un codice ASCII (32-126): 44
Il carattere , ha codice ASCII 44


Immetti un carattere: $
Il carattere $ ha codice ASCII 36
```

Scansione dell'alfabeto

- È possibile generare tutte le lettere dell'alfabeto, in ordine, grazie al fatto che nella tabella ASCII esse compaiono consecutive e ordinate

```
char ch ;

for( ch = 'A' ; ch <= 'Z' ; ch++ )
    putchar(ch) ;

putchar('\n') ;
```

Verifica se è una lettera

- Per sapere se un carattere è alfabetico, è sufficiente verificare se cade nell'intervallo delle lettere (maiuscole o minuscole)

```
if( ch>='A' && ch<='Z' )
    printf("%c lettera maiuscola\n", ch) ;

if( ch>='a' && ch<='z' )
    printf("%c lettera minuscola\n", ch) ;

if( (ch>='A' && ch<='Z') ||
    (ch>='a' && ch<='z') )
    printf("%c lettera\n", ch) ;
```

Verifica se è una cifra

- Per sapere se un carattere è numerico ('0'-'9'), è sufficiente verificare se cade nell'intervallo delle cifre

```
if( ch>='0' && ch<='9' )
    printf("%c cifra numerica\n", ch) ;
```

Valore di una cifra

- Conoscere il valore decimale di un carattere numerico (`'0'-'9'`), è sufficiente calcolare la "distanza" dalla cifra `'0'`

```
if( ch>='0' && ch<='9' )
{
    printf("%c cifra numerica\n", ch) ;
    val = ch - '0' ;
    printf("Il suo valore e': %d", val ) ;
}
```

Da minuscolo a maiuscolo (1/2)

- I codici ASCII delle lettere maiuscole e delle minuscole differiscono solamente per una costante:
- 'A' = 65 ... 'Z' = 90
- 'a' = 97 ... 'z' = 122
- Se ch è una lettera minuscola
- ch - 'a' è la sua posizione nell'alfabeto
- (ch - 'a') + 'A' è la corrispondente lettera maiuscola

Da minuscolo a maiuscolo (2/2)

- Possiamo interpretare la conversione come una traslazione della quantità (`'A'-'a'`)

```
if( ch>='a' && ch<='z' )
{
    printf("%c lettera minuscola\n", ch) ;
    ch2 = ch + ('A'-'a') ;
    printf(La maiuscola e': %c\n", ch2) ;
}
```

Confronto alfabetico

- Se due caratteri sono entrambi maiuscoli (o entrambi minuscoli) è sufficiente confrontare i rispettivi codici ASCII

```
if( ch < ch2 )
  printf("%c viene prima di %c", ch, ch2) ;
else
  printf("%c viene prima di %c", ch2, ch) ;
```

Stringhe in C

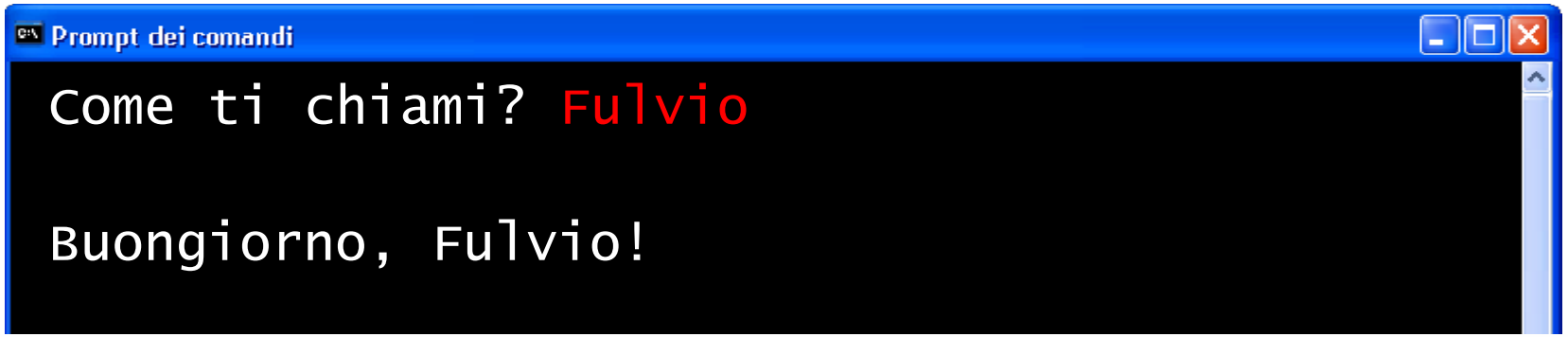
- Nel linguaggio C non è supportato esplicitamente alcun tipo di dato "stringa"
- Le informazioni di tipo stringa vengono memorizzate ed elaborate ricorrendo a semplici vettori di caratteri

```
char saluto[10] ;
```

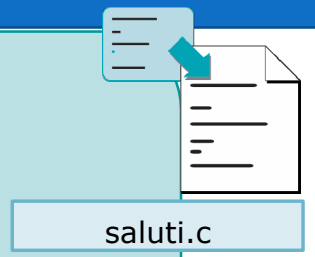
B	u	o	n	g	i	o	r	n	o

Esempio

- Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max 20 caratteri), e stampi a video un saluto per l'utente stesso

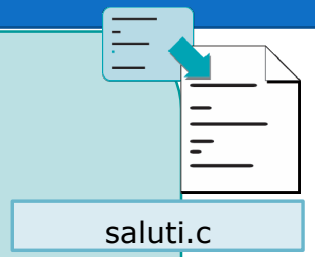


Soluzione (1/3)



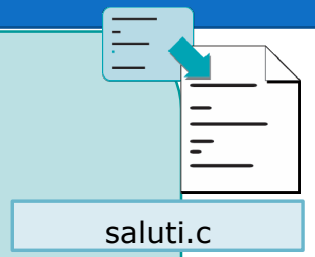

```
const int MAX = 20 ;
char nome[MAX] ;
int N ;
char ch ;
int i ;

printf("Come ti chiami? ") ;

N = 0 ;
```

Soluzione (2/3)




```
ch = getchar() ;

while( ch != '\n' && N<MAX )
{
     nome[N] = ch ;
     N++ ;
     ch = getchar() ;
}
```

Soluzione (3/3)




```
printf("Buongiorno, ") ;

for(i=0; i<N; i++)
    putchar( nome[i] ) ;

printf("!\n") ;
```

Commenti (1/2)

- Qualsiasi operazione sulle stringhe si può realizzare agendo opportunamente su vettori di caratteri, gestiti con occupazione variabile
- Così facendo, però vi sono alcuni svantaggi
- Per ogni vettore di caratteri, occorre definire un'opportuna variabile che ne indichi la lunghezza
- Ogni operazione, anche elementare, richiede l'uso di cicli for/while

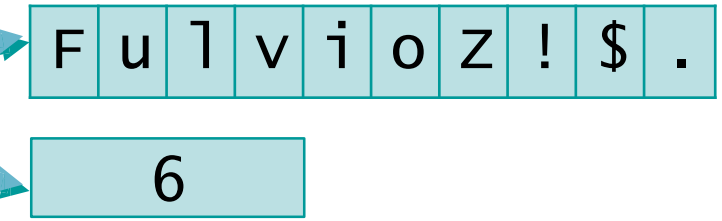
Commenti (2/2)

- Alcune convenzioni ci possono aiutare
  - Gestire in modo standard i vettori di caratteri usati per memorizzare stringhe
  - Apprendere le tecniche solitamente utilizzate per compiere le operazioni più frequenti
- Molte funzioni di libreria seguono queste convenzioni
  - Conoscere le funzioni di libreria ed utilizzarle per accelerare la scrittura del programma

Lunghezza di una stringa

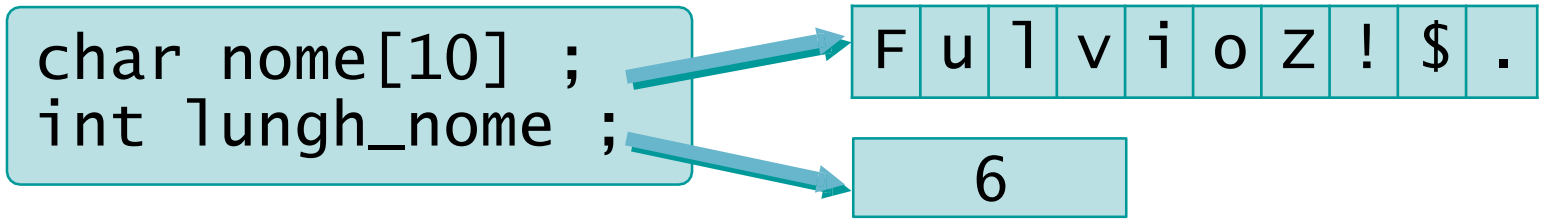
- Vi sono due tecniche per determinare la lunghezza di una stringa
  1. utilizzare una variabile intera che memorizzi il numero di caratteri validi

```
char nome[10] ;
int lungh_nome ;
```

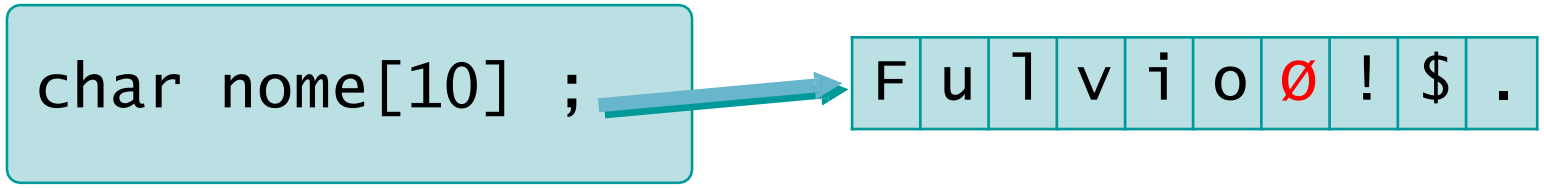


Lunghezza di una stringa

- Vi sono due tecniche per determinare la lunghezza di una stringa
  1. utilizzare una variabile intera che memorizzi il numero di caratteri validi



  2. utilizzare un carattere "speciale", con funzione di terminatore, dopo l'ultimo carattere valido



Carattere terminatore

- Il carattere “terminatore” deve avere le seguenti caratteristiche
- Fare parte della tabella dei codici ASCII
- Deve essere rappresentabile in un char
- Non comparire mai nelle stringhe utilizzate dal programma
- Non deve confondersi con i caratteri “normali”
- Inoltre il vettore di caratteri deve avere una posizione libera in più, per memorizzare il terminatore stesso

Terminatore standard in C

- Per convenzione, in C si sceglie che tutte le stringhe siano rappresentate mediante un carattere terminatore
- Il terminatore corrisponde al carattere di codice ASCII pari a zero (Stringhe ASCII-Z)
  - `nome[6] = 0 ;`
  - `nome[6] = '\0' ;`

F	u	l	v	i	o	Ø	!	$	.

Vantaggi

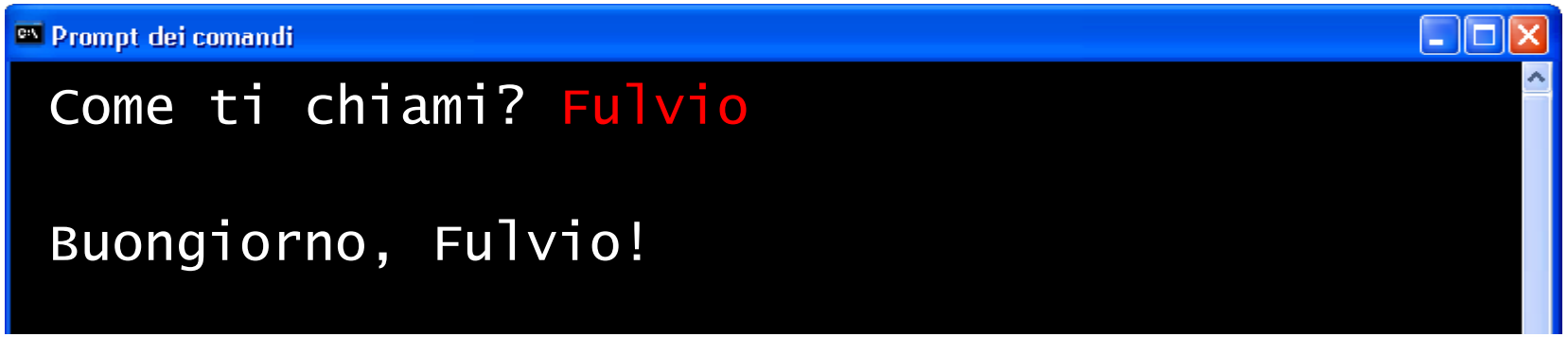
- Non è necessaria un'ulteriore variabile intera per ciascuna stringa
- L'informazione sulla lunghezza della stringa è interna al vettore stesso
- Tutte le funzioni della libreria standard C rispettano questa convenzione
- Si aspettano che la stringa sia terminata Restituiscono sempre stringhe terminate

Svantaggi

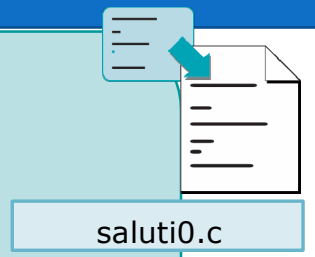
- Necessario 1 byte in più
  - Per una stringa di N caratteri, serve un vettore di N+1 elementi
- Necessario ricordare di aggiungere sempre il terminatore
- Impossibile rappresentare stringhe contenenti il carattere ASCII 0

Esempio

- Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max 20 caratteri), e stampi a video un saluto per l'utente stesso

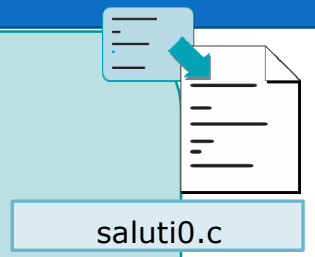


Soluzione (1/3)



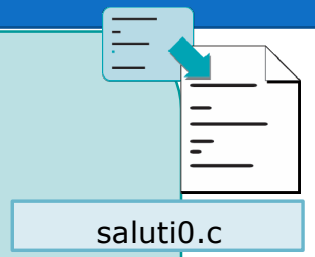

```
const int MAX = 20 ;
char nome[MAX+1] ;
char ch ;
int i ;

printf("Come ti chiami? ") ;

i = 0 ;
```

Soluzione (2/3)




```
i = 0 ;

ch = getchar() ;

while( ch != '\n' && i<MAX )
{
     nome[i] = ch ;
     i++ ;
     ch = getchar() ;
}
/* aggiunge terminatore nullo */
nome[i] = '\0' ;
```

Soluzione (3/3)




```
printf("Buongiorno, ") ;

for(i=0; nome[i]!='\0'; i++)
    putchar( nome[i] ) ;

printf("!\n") ;
```

I/O di stringhe

- Diamo per scontato di utilizzare la convenzione del terminatore nullo
- Si possono utilizzare
- Funzioni di lettura e scrittura carattere per carattere
- Come nell'esercizio precedente
- Funzioni di lettura e scrittura di stringhe intere
- scanf e printf gets e puts

Lettura di stringhe con scanf

- Utilizzare la funzione `scanf` con lo specificatore di formato `"%s"`
- La variabile da leggere deve essere il nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
  - Non utilizzare la &
- Legge ciò che viene immesso da tastiera, fino al primo spazio o fine linea (esclusi)
  - Non adatta a leggere nomi composti (es. `"Pier Paolo"`)

Esempio


```
const int MAX = 20 ;
char nome[MAX+1] ;

printf("Come ti chiami? ") ;

scanf("%s", nome) ;
```

Lettura di stringhe con gets

- La funzione gets è pensata appositamente per acquisire una stringa
- Accetta un parametro, che corrisponde al nome di un vettore di caratteri
- Non utilizzare le parentesi quadre
- Legge ciò che viene immesso da tastiera, fino al fine linea (escluso), e compresi eventuali spazi
- Possibile leggere nomi composti (es. "Pier Paolo")

Esempio


```
const int MAX = 20 ;
char nome[MAX+1] ;

printf("Come ti chiami? ") ;

gets(nome) ;
```

Scrittura di stringhe con printf

- Utilizzare la funzione `printf` con lo specificatore di formato `"%s"`
- La variabile da stampare deve essere il nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
- È possibile combinare la stringa con altre variabili nella stessa istruzione

Esempio


```
printf("Buongiorno, ") ;
printf("%s", nome) ;
printf("!\n") ;


printf("Buongiorno, %s!\n", nome) ;
```

Scrittura di stringhe con puts

- La funzione puts è pensata appositamente per stampare una stringa
- La variabile da stampare deve essere il nome di un vettore di caratteri
- Non utilizzare le parentesi quadre
- Va a capo automaticamente
- Non è possibile stampare altre informazioni sulla stessa riga

Esempio


```
printf("Buongiorno, ") ;
puts(nome) ;

/* No!! printf("!\n") ; */
```

Conclusione

- Utilizzare sempre la convenzione del terminatore nullo
- Ricordare di allocare un elemento in più nei vettori di caratteri
- Utilizzare quando possibile le funzioni di libreria predefinite
- In lettura, prediligere gets
- In scrittura
  - printf è indicata per messaggi composti
  - puts è più semplice se si ha un dato per riga

Operazioni elementari sulle stringhe

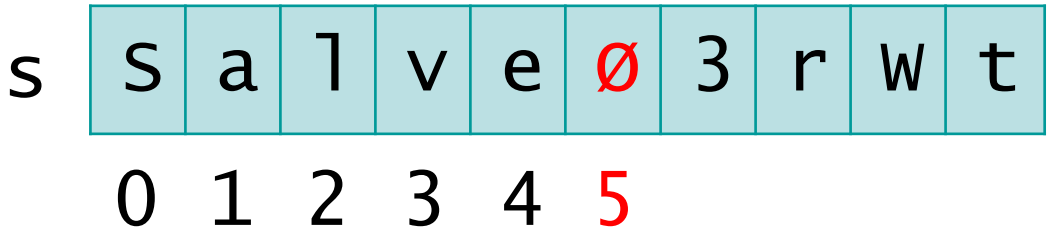
UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Lunghezza
- Copia di stringhe
- Concatenazione di stringhe

Lunghezza di una stringa

- La lunghezza di una stringa si può determinare ricercando la posizione del terminatore nullo

```
char s[MAX+1] ;
   int lun ;
```



Calcolo della lunghezza


```
const int MAX = 20 ;
char s[MAX+1] ;
int lun ;
int i ;

... /* lettura stringa */

for( i=0 ; s[i] != 0 ; i++ )
     /* Niente    ;
     */
lun = i ;
```

La funzione strlen

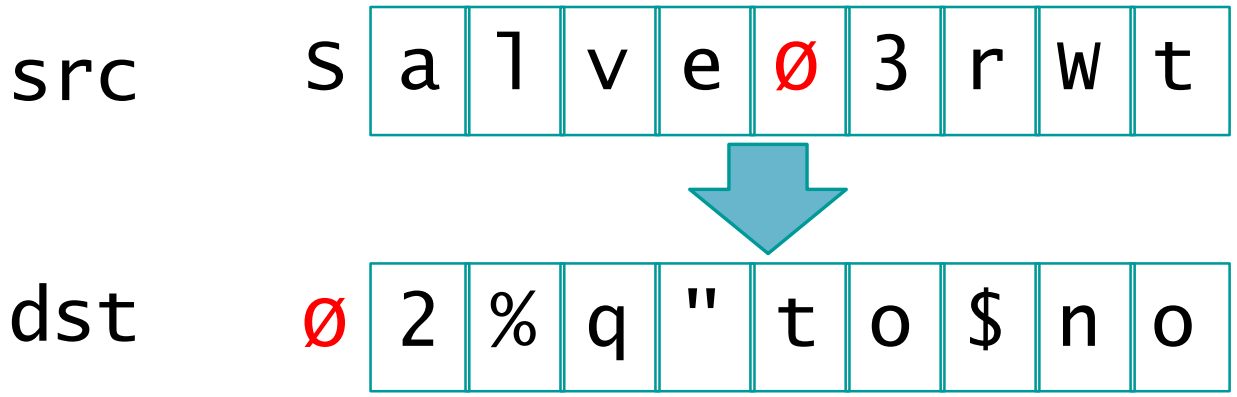
- Nella libreria standard C è disponibile la funzione `strlen`, che calcola la lunghezza della stringa passata come parametro
- Necessario includere `<string.h>`

```
const int MAX = 20 ;
char s[MAX+1] ;
int lun ;

... /* lettura stringa */

lun = strlen(s) ;
```

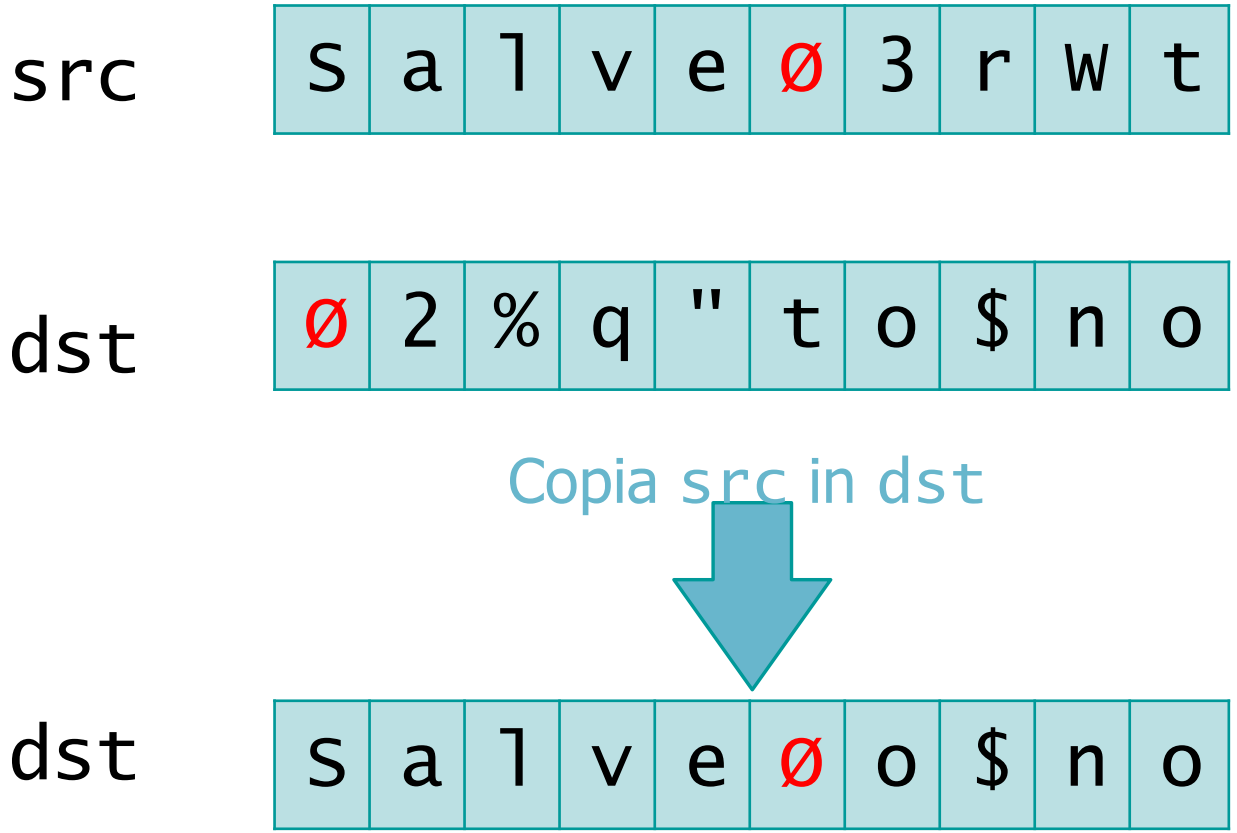
Copia di stringhe

- L'operazione di copia prevede di ricopiare il contenuto di una prima stringa "sorgente", in una seconda stringa "destinazione"

```
char src[MAXS+1] ;
char dst[MAXD+1] ;
```



Risultato della copia



Copia


```
const int MAXS = 20, MAXD = 30 ;
char src[MAXS+1] ;
char dst[MAXD+1] ;
int i ;

... /* lettura stringa src */

for( i=0 ; src[i] != 0 ; i++ )
     dst[i] = src[i] ; /* copia */

dst[i] = 0 ; /* aggiunge terminatore */
```

La funzione strcpy

- Nella libreria standard C, includendo `<string.h>`, è disponibile la funzione `strcpy`, che effettua la copia di stringhe
  - Primo parametro: stringa destinazione
  - Secondo parametro: stringa sorgente

```
const int MAXS = 20, MAXD = 30 ;
char src[MAXS+1] ;
char dst[MAXD+1] ;

... /* lettura stringa src */

strcpy(dst, src) ;
```

Avvertenze

- Nella stringa destinazione vi deve essere un numero sufficiente di locazioni libere
  - `MAXD+1 >= strlen(src)+1`
- Il contenuto precedente della stringa destinazione viene perso
- La stringa sorgente non viene modificata
- Il terminatore nullo
  - Deve essere aggiunto in coda a `dst`
  - La `strcpy` pensa già autonomamente a farlo

Errore frequente

- Per effettuare una copia di stringhe non si può assolutamente utilizzare l'operatore =
- Necessario usare `strcpy`

```
dst = src ;
```

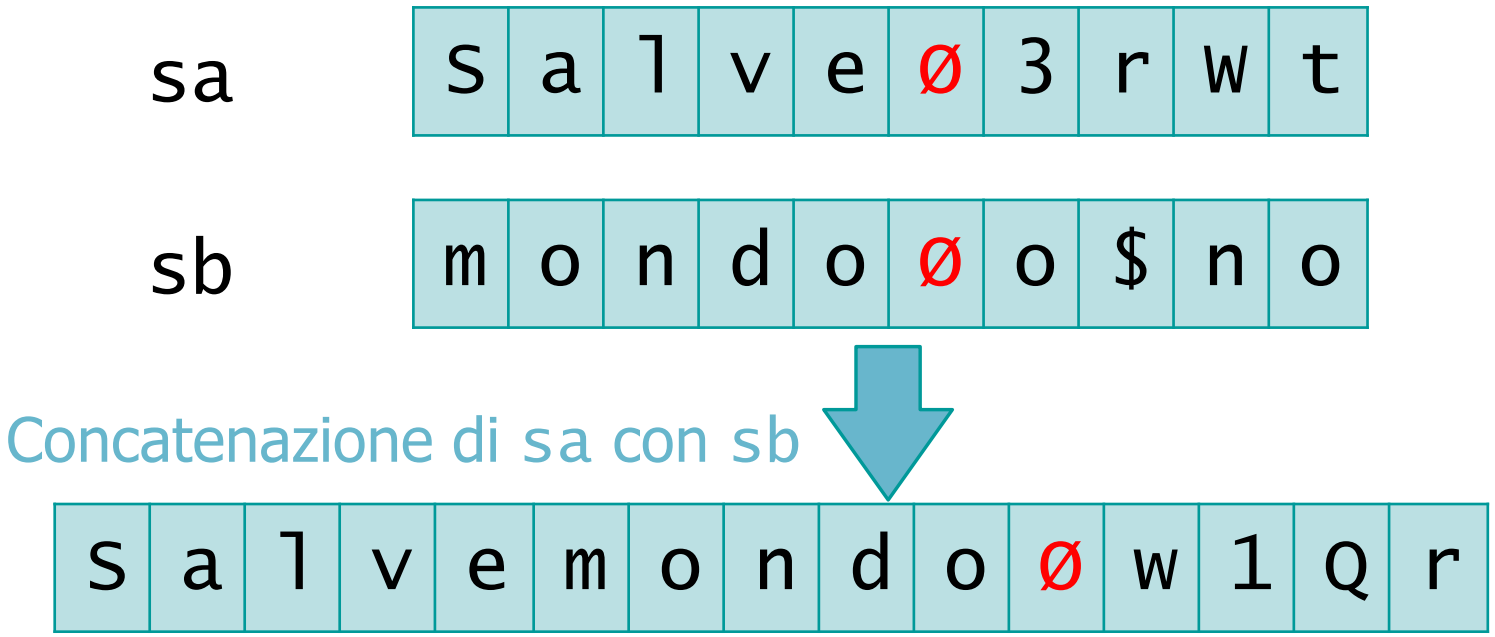
```
dst[] = src[] ;
```

```
dst[MAXD] = src[MAXC] ;
```

```
strcpy(dst, src);
```

Concatenazione di stringhe

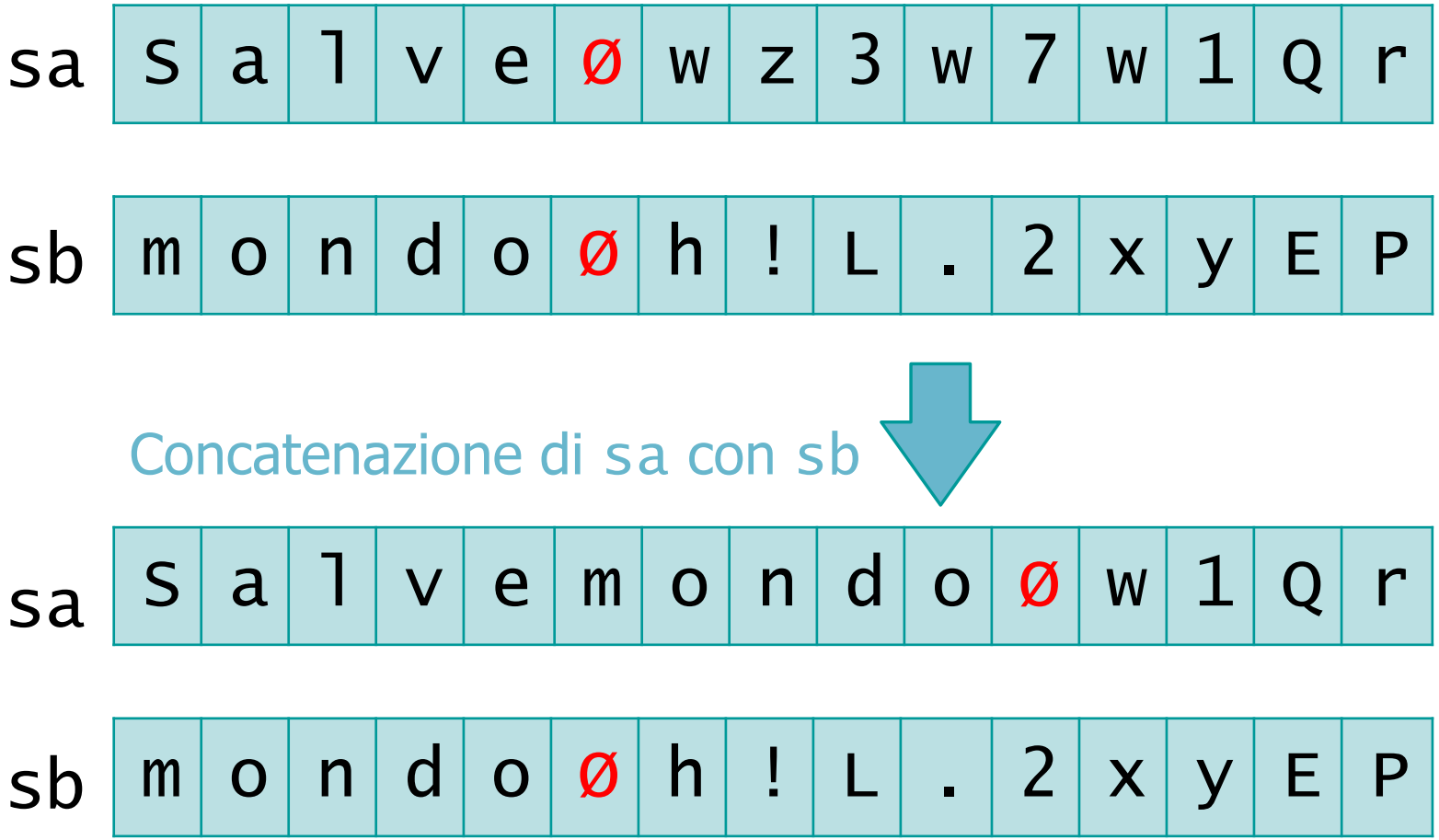
- L'operazione di concatenazione corrisponde a creare una nuova stringa composta dai caratteri di una prima stringa, seguiti dai caratteri di una seconda stringa



Semplificazione

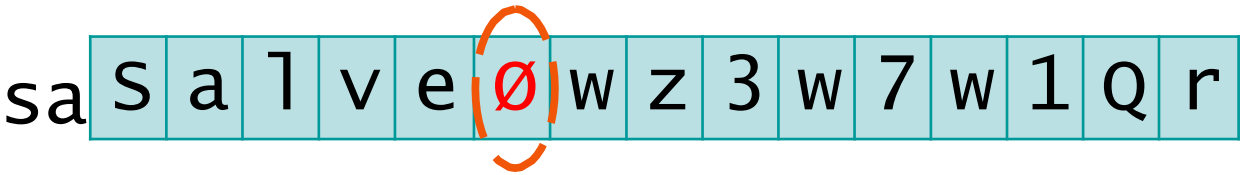
- Per maggior semplicità, in C l'operazione di concatenazione scrive il risultato nello stesso `vettore` della prima stringa
- Il valore precedente della prima stringa viene così perso
- Per memorizzare altrove il risultato, o per non perdere la prima stringa, è possibile ricorrere a stringhe temporanee ed alla funzione `strcpy`

Esempio


sa
S a l v e Ø w z 3 w 7 w 1 Q r
sb
m o n d o Ø h ! L . 2 x y E P
Concatenazione di sa con sb
sa
S a l v e m o n d o Ø w 1 Q r
sb
m o n d o Ø h ! L . 2 x y E P

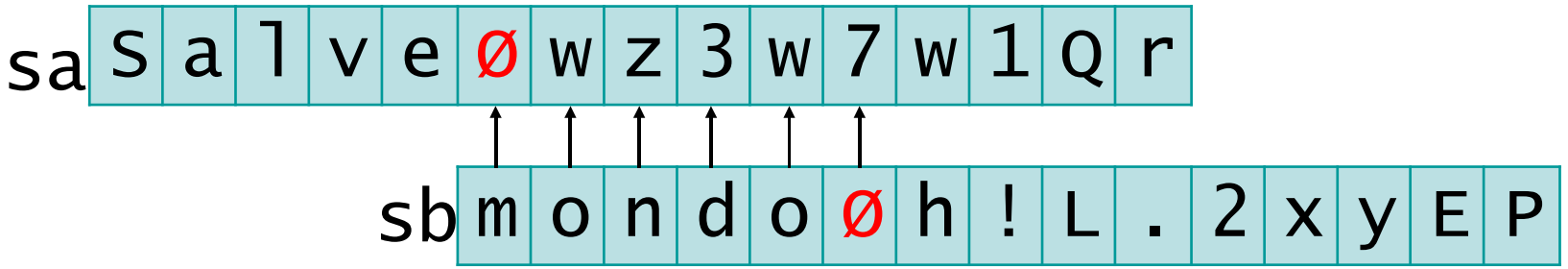
Algoritmo di concatenazione

- Trova la fine della prima stringa



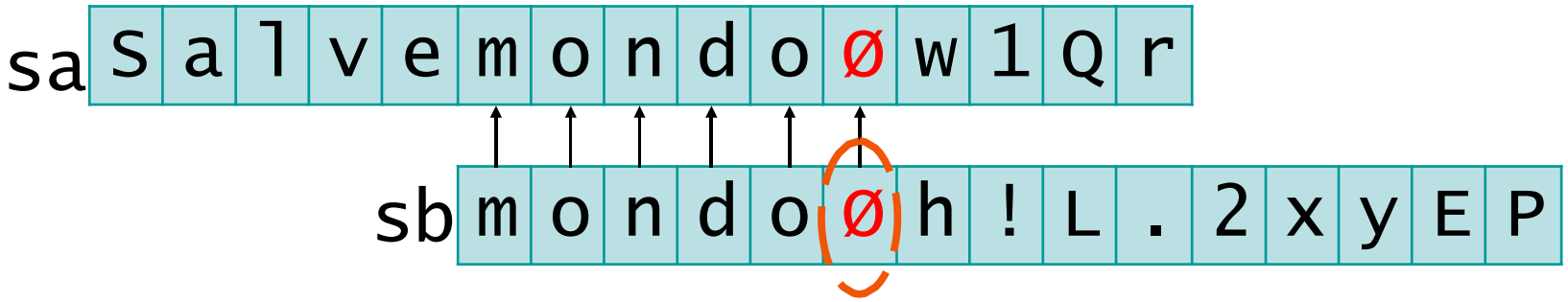
Algoritmo di concatenazione

- Trova la fine della prima stringa
- Copia la seconda stringa nel vettore della prima, a partire della posizione del terminatore nullo (sovrascrivendolo)



Algoritmo di concatenazione

- Trova la fine della prima stringa
- Copia la seconda stringa nel vettore della prima, a partire della posizione del terminatore nullo (sovrascrivendolo)
- Termina la copia non appena trovato il terminatore della seconda stringa



Concatenazione


```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int la ;
int i ;

... /* lettura stringhe */

la = strlen(sa) ;

for( i=0 ; sb[i] != 0 ; i++ )
    sa[la+i] = sb[i] ; /* copia */

sa[la+i] = 0 ; /* terminatore */
```

La funzione strcat

- Nella libreria standard C, includendo
- <string.h>, è disponibile la funzione strcat, che effettua la concatenazione di stringhe
- Primo parametro: prima stringa (destinazione)
- Secondo parametro: seconda stringa

- const int MAX = 20 ; char sa[MAX] ;
- char sb[MAX] ;
- ... /* lettura stringhe */ strcat(sa, sb) ;

Avvertenze (1/2)

- Nella prima stringa vi deve essere un numero sufficiente di locazioni libere
  - `MAX+1 >= strlen(sa)+strlen(sb)+1`
- Il contenuto precedente della prima stringa viene perso
- La seconda stringa non viene modificata
- Il terminatore nullo
  - Deve essere aggiunto in coda alla prima stringa
  - La `strcat` pensa già autonomamente a farlo

Avvertenze (2/2)

- Per concatenare 3 o più stringhe, occorre farlo due a due:
  - `strcat(sa, sb);`
  - `strcat(sa, sc);`
- È possibile concatenare anche stringhe costanti
  - `strcat(sa, "!");`