Programmazione in C

Unità
Caratteri e stringhe

Caratteri e stringhe

- Dati testuali
- Il tipo char
- Vettori di caratteri
- Operazioni elementari sulle stringhe
- Funzioni di libreria
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 1 e 5
  - Cabodi, Quer, Sonza Reorda: capitolo 5
  - Dietel & Dietel: capitolo 8
- Dispense
  - Scheda: "Caratteri e stringhe in C"

Caratteri e stringhe

Dati testuali

Dati testuali

- Tipi di dato testuali
- Caratteri
- Stringhe

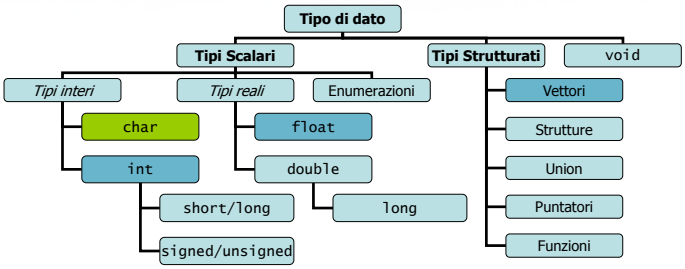
Dati testuali

Tipi di dato testuali

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

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

Dati testuali

Caratteri

Caratteri in C

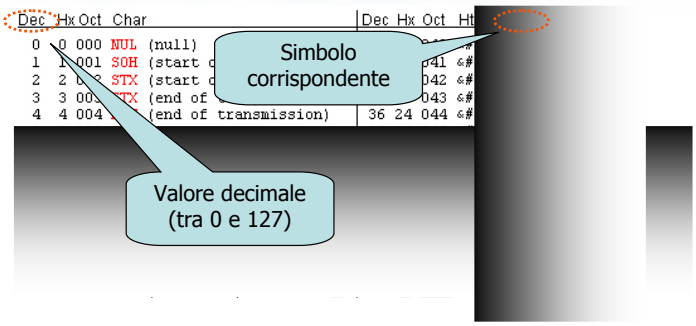
- Ogni carattere viene rappresentato dal proprio codice ASCII
- Sono sufficienti 7 bit per rappresentare ciascun carattere
  - Il C usa variabili di 8 bit (1 byte)
- Non sono previste le lettere accentate né altri simboli diacritici
  - Richiedono estensioni speciali e librerie specifiche

Codice ASCII

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

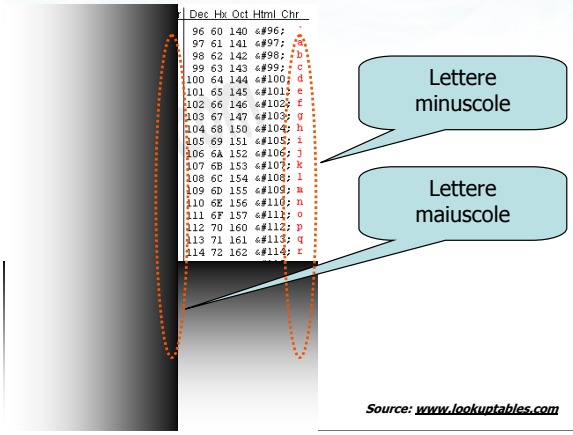
Source: www.lookuptables.com

Codice ASCII



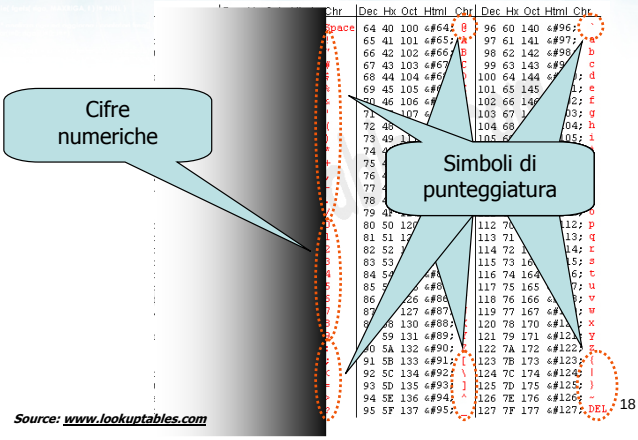
Source: www.lookuptables.com

Codice ASCII



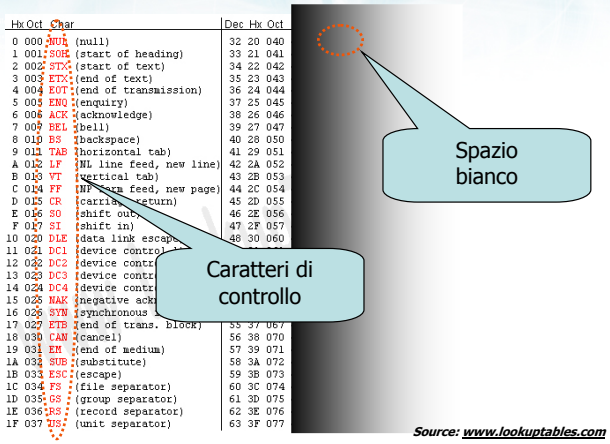
Source: www.lookuptables.com

Codice ASCII



Source: www.lookuptables.com

Codice ASCII



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

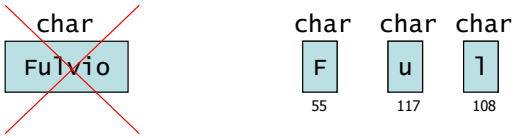
Errore frequente

- Non confondere il carattere ASCII che rappresenta una cifra numerica con il valore decimale associato a tale cifra

- Per chiarezza useremo gli apici per indicare i caratteri

Errore frequente

- Pensare che un singolo carattere possa memorizzare più simboli



Dati testuali

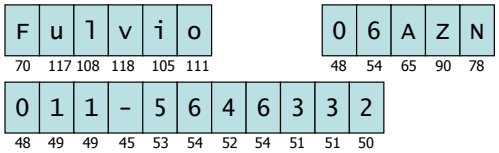
Stringhe

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

Caratteri e stringhe

Il tipo char

Il tipo char

- Variabili char
- Input/output di char
- Operazioni sui char
- Esercizio "Quadrati di lettere"

Il tipo char

Variabili char

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

```
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

```
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*
'\x*hh*'	*hh*	Immette in carattere ASCII con codice (esadecimale) *hh*

Il tipo char

Input/output di char

Input/output di char

- 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)

- 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)

- 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() ;
```

```
Dato: _
```

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') ;
```


Soluzione proposta

```
char ch, temp ;

printf("Dato: ");

ch = getchar() ; /

/* elimina eventua
ed il \n che sicur
do {
    temp = getchar() ;
} while (temp != '\n') ;
```

```
/* forma più compatta */
while ( getchar()!='\n' )
  /*niente*/ ;
```


Il tipo char

Operazioni sui char

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)

char-int.c

```
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) ;
```

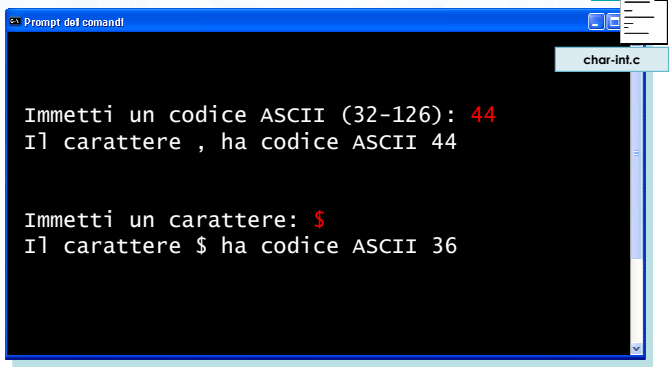
Esempio (2/3)

char-int.c

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

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

i = ch ;

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


Esempio (3/3)

char-int.c



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) ;
```


Il tipo char

Esercizio "Quadrati di lettere"

Esercizio "Quadrati di lettere"

- Si scriva un programma in linguaggio C che stampi su video una serie di quadrati, composti dalle successive lettere dell'alfabeto, di dimensioni sempre crescenti:
  - Un quadrato 1x1 di lettere A
  - Un quadrato 2x2 di lettere B
  - Un quadrato 3x3 di lettere C
  - ...eccetera

Analisi

```
Prompt dei comandi
Quanti quadrati vuoi stampare? 4
A

BB
BB

CCC
CCC
CCC

DDDD
DDDD
DDDD
DDDD
```

Soluzione (1/2)

quadrati.c

```
int i, N ;
int riga, col ;
char ch ;

printf("Quanti quadrati? ") ;
scanf("%d", &N) ;

while(N<1 || N>26)
{
  printf("Deve essere tra 1 e 26\n");
  printf("Quanti quadrati? ") ;
  scanf("%d", &N) ;
}
```


Soluzione (2/2)

quadrati.c

```
for( i=0; i<N; i++ )
{
    /* stampa un quadrato
       di dimensione (i+1) */

    ch = i + 'A' ;

    for(riga=0; riga<i+1; riga++)
    {
        for(col=0; col<i+1; col++)
            putchar(ch);
        putchar('\n') ;
    }
    putchar('\n') ;
}
```

Caratteri e stringhe

Vettori di caratteri

Vettori di caratteri

- Il tipo stringa
- Terminatore nullo
- Input/output di stringhe

Vettori di caratteri

Il tipo stringa

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

```
Prompt dei comandi
Come ti chiami? Fulvio

Buongiorno, Fulvio!
```


Soluzione (1/3)

saluti.c

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

printf("Come ti chiami? ") ;

N = 0 ;
```

Soluzione (2/3)

saluti.c

```
ch = getchar() ;

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


Soluzione (3/3)

saluti.c

```
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

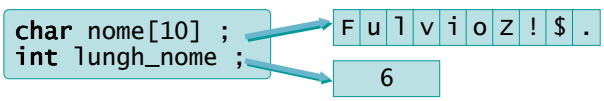
Vettori di caratteri

Terminatore nullo

Lunghezza di una stringa

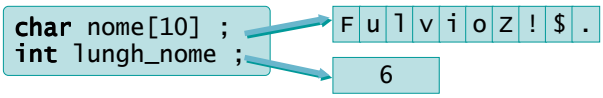
- Vi sono due tecniche per determinare la lunghezza di una stringa
  1. utilizzare una variabile intera che memorizzi il numero di caratteri validi

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

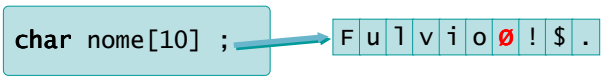


Lunghezza di una stringa

- 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

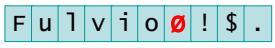


Carattere terminatore

- 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
  - `nome[6] = 0 ;`
  - `nome[6] = '\0' ;`



Vantaggi

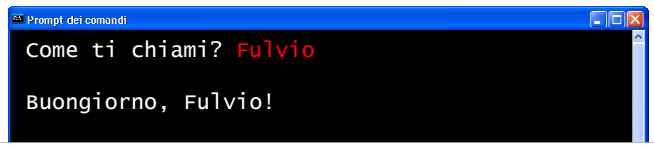
- Non è necessaria un'ulteriore variabile intera per ciascuna stringa
- L'informazione sulla lunghezza della stringa è interna al vettore stesso
- Tutte le funzioni della libreria standard C rispettano questa convenzione
  - Si aspettano che la stringa sia terminata
  - Restituiscono sempre stringhe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

saluti0.c

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

printf("Come ti chiami? ") ;

i = 0 ;
```


Soluzione (2/3)

saluti0.c

```
i = 0 ;

ch = getchar() ;

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


Soluzione (3/3)

saluti0.c

```
printf("Buongiorno, ") ;

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

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


Vettori di caratteri

Input/output di stringhe

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

Caratteri e stringhe

Operazioni elementari sulle stringhe

Operazioni elementari sulle stringhe

- Lunghezza
- Copia di stringhe
- Concatenazione di stringhe
- Confronto di stringhe
- Ricerca di sotto-stringhe
- Ricerca di parole

Operazioni elementari sulle stringhe

Lunghezza

Lunghezza di una stringa

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

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

s	S	a	l	v	e	ø	3	r	W	t
	0	1	2	3	4	5				

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) ;
```

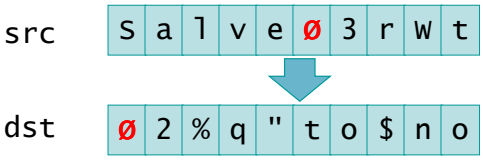
Operazioni elementari sulle stringhe

Copia di stringhe

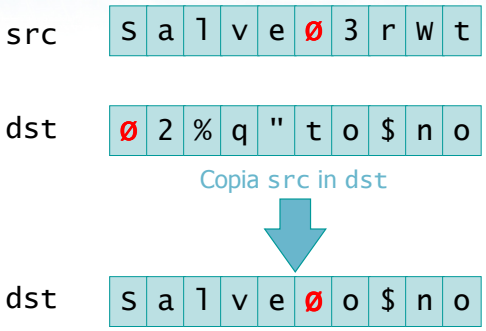
Copia di stringhe

- 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);
```


Operazioni elementari sulle stringhe

Concatenazione di stringhe

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`

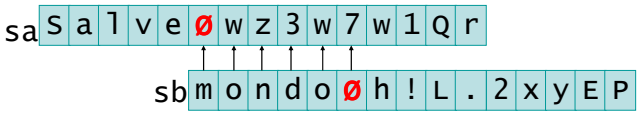
Esempio

Algoritmo di concatenazione

- Trova la fine della prima stringa

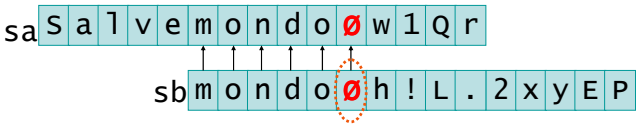
Algoritmo di concatenazione

- 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, "!");`

Operazioni elementari sulle stringhe

Confronto di stringhe

Confronto di stringhe

- Il confronto di due stringhe (es.: `sa` e `sb`), mira a determinare se:
  - Le due stringhe sono uguali
    - hanno uguale lunghezza e sono composte dagli stessi caratteri nello stesso ordine
  - Le due stringhe sono diverse
  - La stringa `sa` precede la stringa `sb`
    - secondo l'ordine lessicografico imposto dal codice ASCII
    - parzialmente compatibile con l'ordine alfabetico
  - La stringa `sa` segue la stringa `sb`

Confronto di uguaglianza

- Ogni carattere di `sa` deve essere uguale al carattere corrispondente di `sb`
- Il terminatore nullo deve essere nella stessa posizione
- I caratteri successivi al terminatore vanno ignorati

sa	S	a	l	v	e	Ø	o	4	d	1	Ø	w	1	Q	r
sb	S	a	l	v	e	Ø	h	!	L	.	2	x	y	E	P

Confronto di uguaglianza

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

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```


Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[M
int ugual
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Flag: ricerca di universalità della condizione `sa[i]==sb[i]`

Confronto di uguaglianza

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

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Cicla fino al terminatore di `sa` o di `sb` (il primo che si incontra)

Confronto di uguaglianza

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

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Verifica che tutti i caratteri incontrati siano uguali. Se no, poni a 0 il flag `uguali`.

Confronto di uguaglianza

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

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

In questo punto sicuramente una delle due stringhe è arrivata al terminatore. Se non lo è anche l'altra, allora non sono uguali!

Confronto di ordine

- Verifichiamo se `sa` "è minore di" `sb`. Partiamo con `i=0`
- Se `sa[i]<sb[i]`, allora `sa` è minore
- Se `sa[i]>sb[i]`, allora `sa` non è minore
- Se `sa[i]=sb[i]`, allora bisogna controllare i caratteri successivi (`i++`)
- Il terminatore nullo conta come "minore" di tutti

sa	S	a	l	v	e	Ø	o	4	d	1	Ø	w	1	Q	r

sb	S	a	l	u	t	e	Ø	!	L	.	2	x	y	E	P

Confronto di ordine (1/2)

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int minore ;
int i ;
...
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
          && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}
```


Confronto di ordine (2/2)

```
if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    printf("%s e' minore di %s\n",
           sa, sb ) ;
```


Commenti

```
minore = 0 ;
for( i=0 ; sa[i]
          && minore
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}

if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    ...
```

Ricerca di esistenza della condizione `sa[i]<sb[i]`.

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore
    if(sa[i]>sb
        minore
}

if(minore==0 &&
    minore=1 ;

if(minore==1)
    ...
```

Cicla fino al primo terminatore nullo, oppure fino a che non si "scopre" chi è minore.
In altre parole, continua a ciclare solo finché le stringhe "sembrano" uguali.

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=
           && mino
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}
                  sa
if(minore==1)
    ...
```

Sicuramente sa è minore di sb
Flag: `minore = 1`

Se flag `minore==0` continua a ciclare

Sicuramente sa non è minore di sb
Flag: `minore = -1`

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore = 1
    if(sa[i]>sb[i])
        minore = -1
}

if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    ...
```

Se finora erano uguali, ma sa è più corta di sb, allora sa è minore

La funzione `strcmp`

- Nella libreria standard C, includendo `<string.h>`, è disponibile la funzione `strcmp`, che effettua il confronto di stringhe
  - Primo parametro: prima stringa
  - Secondo parametro: seconda stringa
  - Valore restituito:
    - `<0` se la prima stringa è minore della seconda
    - `==0` se le stringhe sono uguali
    - `>0` se la prima stringa è maggiore della seconda

Confronti vari

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

...

ris = strcmp(sa, sb) ;

if(ris<0)
    printf("%s minore di %s\n", sa, sb);
if(ris==0)
    printf("%s uguale a %s\n", sa, sb);
if(ris>0)
    printf("%s maggiore di %s\n", sa, sb);
```

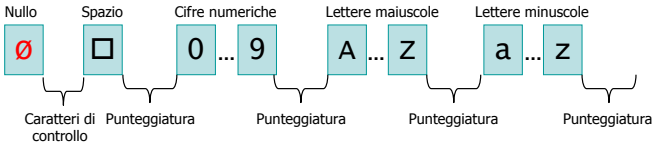
Suggerimento

- Per ricordare il significato del valore calcolato da `strcmp`, immaginare che la funzione faccia una "sottrazione" tra le due stringhe
  - `sa - sb`
    - Negativo: sa minore
    - Positivo: sa maggiore
    - Nullo: uguali

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int ris ;
...
ris = strcmp(sa, sb) ;
```

Ordinamento delle stringhe

- La funzione `strcmp` lavora confrontando tra loro i codici ASCII dei caratteri
- Il criterio di ordinamento è quindi dato dalla posizione dei caratteri nella tabella ASCII



Conseguenze

- Ogni lettera maiuscola precede ogni lettera minuscola
  - `Ciao` precede `ciao`
  - `Zulu` precede `apache`
- Gli spazi contano, e precedono le lettere
  - `Qui Quo Qua` precede `QuiQuoQua`
- I simboli di punteggiatura contano, ma non vi è una regola intuitiva
- L'ordinamento che si ottiene è lievemente diverso da quello "standard" alfabetico

Operazioni elementari sulle stringhe

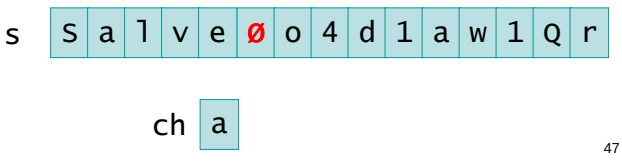
Ricerca di sotto-stringhe

Ricerca in una stringa

- È possibile concepire diversi tipi di ricerche da compiersi all'interno di una stringa:
  - Determinare se un determinato carattere compare all'interno di una stringa data
  - Determinare se una determinata stringa compare integralmente all'interno di un'altra stringa data, in una posizione arbitraria

Ricerca di un carattere (1/2)

- Detti:
  - `s` una stringa arbitraria
  - `ch` un carattere qualsiasi
- Determinare se la stringa `s` contiene (una o più volte) il carattere `ch` al suo interno, in qualsiasi posizione



Ricerca di un carattere (2/2)

```
const int MAX = 20 ;
char s[MAX] ;
char ch ;
int trovato ;
int i ;

...

trovato = 0 ;
for( i=0 ; s[i]!=0 && trovato==0; i++ )
{
    if( s[i]==ch )
        trovato = 1 ;
}
```

La funzione strchr (1/2)

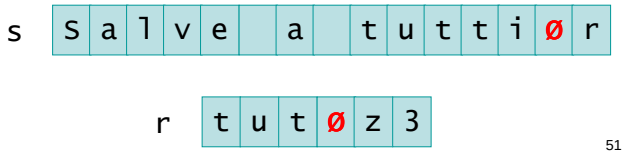
- Nella libreria standard C, includendo <string.h>, è disponibile la funzione strchr, che effettua la ricerca di un carattere
  - Primo parametro: stringa in cui cercare
  - Secondo parametro: carattere da cercare
  - Valore restituito:
    - !=NULL se il carattere c'è
    - ==NULL se il carattere non c'è

La funzione strchr (2/2)

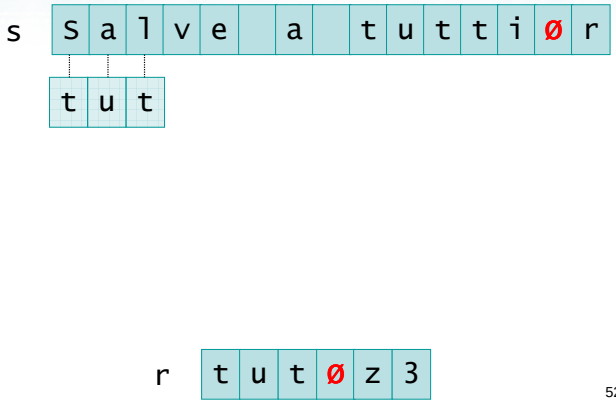
```
const int MAX = 20 ;
char s[MAX] ;
char ch ;

...

if(strchr(s, ch)!=NULL)
   printf("%s contiene %c\n", s, ch) ;
```


Ricerca di una sotto-stringa

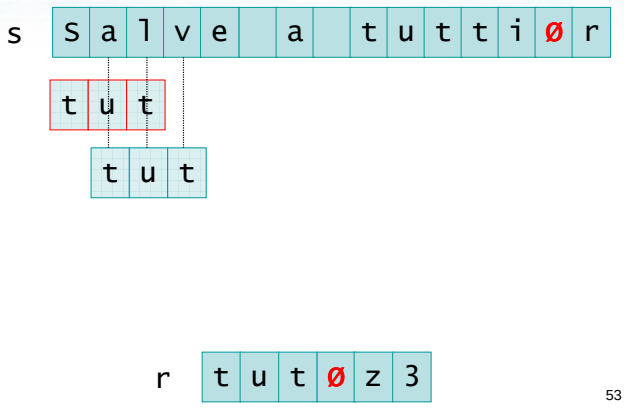
- Detti:
  - s una stringa arbitraria
  - r una stringa da ricercare
- Determinare se la stringa s contiene (una o più volte) la stringa r al suo interno, in qualsiasi posizione



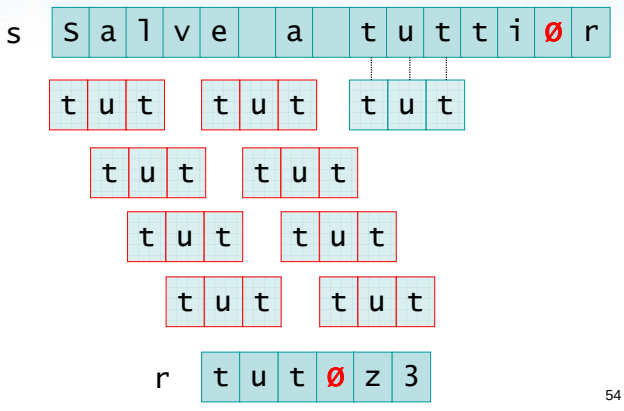
Esempio



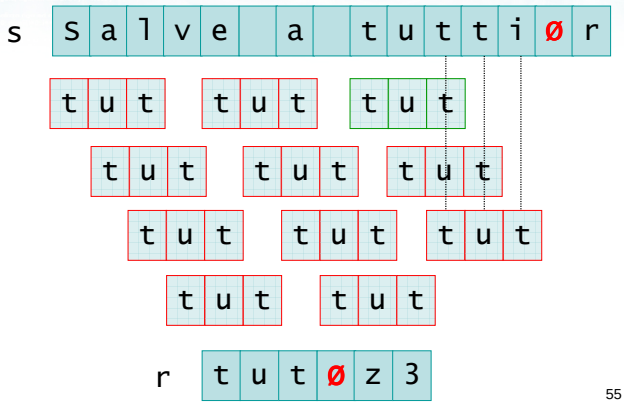
Esempio



Esempio

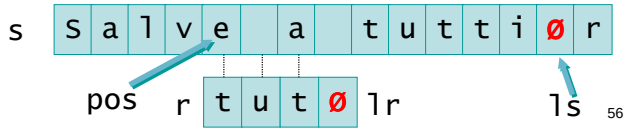


Esempio



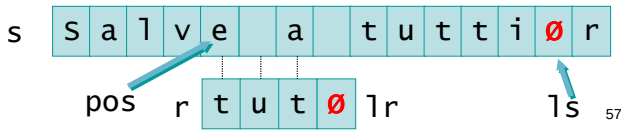
Algoritmo di ricerca

- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione possibile di `r` all'interno di `s`: `pos = 0...ls-lr` (compresi)



Algoritmo di ricerca

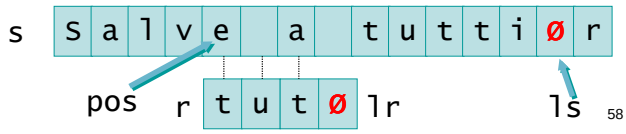
- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione possibile di `r` all'interno di `s`: `pos = 0...ls-lr` (compresi)
  - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`, tra `pos` e `pos+lr-1`
  - Se sì, `trovato = 1`



Algoritmo di ricerca

- `lr = strlen(r);`
- `trovato = 0`
- Per ogni posizione po
  `pos = 0...ls-lr` (compresi)
  - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
  - Se sì, `trovato = 1`

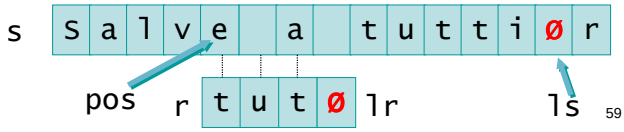
```
diversi = 0 ;
for(i=0; i<lr; i++)
    if(r[i]!=s[pos+i])
        diversi = 1 ;
```



Algoritmo di ricerca

- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione po
  `pos = 0...ls-lr`
  - Controlla se i caratt
    coincidono con i ca
    `pos+lr-1`
  - Se sì, `trovato = 1`

```
if(diversi==0)
    trovato=1 ;
```



Ricerca di una sotto-stringa (1/2)

substr.c

```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;
int lr, ls, pos ;
int i ;
int trovato, diversi ;

...

ls = strlen(s);
lr = strlen(r);
```

Ricerca di una sotto-stringa (2/2)


```
trovato = 0 ;
for(pos=0; pos<=ls-lr; pos++)
{
  /* confronta r[0...lr-1] con  s[pos...pos+lr-1] */
    diversi = 0 ;
    for(i=0; i<lr; i++)
        if(r[i]!=s[pos+i])
            diversi = 1 ;

    if(diversi==0)
        trovato=1 ;
}

if(trovato==1)
    printf("Trovato!\n");
```


La funzione strstr (1/2)

- Nella libreria standard C, includendo <string.h>, è disponibile la funzione strstr, che effettua la ricerca di una sottostringa
  - Primo parametro: stringa in cui cercare
  - Secondo parametro: sotto-stringa da cercare
  - Valore restituito:
    - !=NULL se la sotto-stringa c'è
    - ==NULL se la sotto-stringa non c'è

La funzione strstr (2/2)

```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;

...

if(strstr(s, r)!=NULL)
    printf("Trovato!\n");
```


Operazioni elementari sulle stringhe

Ricerca di parole

Ricerca di parole

- Talvolta non interessa trovare una qualsiasi sotto-stringa, ma solamente verificare se una **parola completa** è presente in una stringa

s1	C	i	a	o		n	o	n	n	o	ø	t	2	"	r
s2	o	g	g	i		n	o	n		c	'	e	'	ø	4

r	n	o	n	ø	z	3

Definizioni (1/2)

- **Lettera**: carattere ASCII facente parte dell'alfabeto maiuscolo ('A'...'Z') o minuscolo ('a'...'z')
- **Parola**: insieme consecutivo di lettere, separato da altre parole mediante spazi, numeri o simboli di punteggiatura

Definizioni (2/2)

- **Inizio di parola**: lettera, prima della quale non vi è un'altra lettera
  - Non vi è un altro carattere (inizio stringa)
  - Vi è un altro carattere, ma non è una lettera
- **Fine di parola**: lettera, dopo la quale non vi è un'altra lettera
  - Non vi è un altro carattere (fine stringa)
  - Vi è un altro carattere, ma non è una lettera

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```

```
if( pos == 0 ||
    !( (s[pos-1]>='a' && s[pos-1]<='z') ||
       (s[pos-1]>='A' && s[pos-1]<='Z') )
  )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```

```
if( pos == ls-lr ||
    !( (s[pos+lr]>='a' && s[pos+lr]<='z') ||
       (s[pos+lr]>='A' && s[pos+lr]<='Z') )
  )
```

Ricerca di una parola (1/2)

parola.c

```
trovato = 0 ;
for(pos=0; pos<=ls-lr; pos++)
{
    if( pos==0 ||
        !( (s[pos-1]>='a' &&
            s[pos-1]<='z') ||
           (s[pos-1]>='A' &&
            s[pos-1]<='Z') ) )
    {
        diversi = 0 ;
        for(i=0; i<lr; i++)
            if(r[i]!=s[pos+i])
                diversi = 1 ;
```


Ricerca di una parola (2/2)

parola.c

```
        if( diversi==0 &&
            ( pos == ls-lr ||
              !( (s[pos+lr]>='a' &&
                  s[pos+lr]<='z') ||
                 (s[pos+lr]>='A' &&
                  s[pos+lr]<='Z') )
            ) )
        {
            trovato=1 ;
        }
    }
}
```


La funzione strparola

- Nella libreria standard C non esiste alcuna funzione che svolga automaticamente la ricerca di una parola intera!!!
- Occorre identificare, ogni volta, se il compito da svolgere è riconducibile ad una o più funzioni di libreria
- Eventualmente si combinano tra loro più funzioni di libreria diverse
- In alcuni casi occorre però ricorrere all'analisi carattere per carattere

Caratteri e stringhe

Funzioni di libreria

Funzioni di libreria

- Introduzione
- Lunghezza di stringhe
- Classificazione di caratteri
- Trasformazione di caratteri
- Copia e concatenazione
- Confronto di stringhe
- Ricerca in stringhe
- Conversione numero-stringa

Funzioni di libreria

Introduzione

Librerie sulle stringhe

- La libreria standard C dispone di molte funzioni predisposte per lavorare su caratteri e stringhe
- Tali funzioni si trovano prevalentemente in due librerie:
  - `<ctype.h>` funzioni operanti su caratteri
  - `<string.h>` funzioni operanti su stringhe
- Tutte le funzioni di libreria accettano e generano stringhe correttamente terminate

Suggerimenti

- Quando possibile, utilizzare sempre le funzioni di libreria
  - Sono più veloci
  - Sono maggiormente collaudate
- In ogni caso, ricordare che è sempre possibile effettuare le operazioni direttamente:
  - Sui caratteri, ricorrendo alla codifica ASCII
  - Sulle stringhe, ricorrendo alla rappresentazione come vettori di caratteri

Rappresentazione

Nome funzione	**`strlen`**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`int`** : la lunghezza della stringa
Descrizione	Calcola la lunghezza della stringa `s`
Esempio	`lun = strlen(s) ;`

Convenzioni

- Assumiamo che nel seguito di questa lezione siano valide le seguenti definizioni

```
const int MAX = 20 ;
char s[MAX] ;
char s1[MAX] ;
char s2[MAX] ;
char r[MAX] ;
int lun ;
int n ;
char ch ;
float x ;
```


Funzioni di libreria

Lunghezza di stringhe

Lunghezza di stringhe

- Definite in `<string.h>`
- Determina la lunghezza di una stringa data
  - strlen

strlen

Nome funzione	**strlen**
Libreria	`#include <string.h>`
Parametri in ingresso	s : stringa
Valore restituito	**int** : la lunghezza della stringa
Descrizione	Calcola la lunghezza della stringa s
Esempio	`lun = strlen(s) ;`

Funzioni di libreria

Classificazione di caratteri

Classificazione di caratteri

- Definite in `<ctype.h>`
- Analizzano un singolo carattere, identificandone la tipologia
  - Lettera
    - Maiuscola
    - Minuscola
  - Cifra
  - Punteggiatura
  - isalpha
  - isupper
  - islower
  - isdigit
  - isalnum
  - isxdigit
  - ispunct
  - isgraph
  - isprint
  - isspace
  - iscntrl

isalpha

Nome funzione	**isalpha**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera maiuscola o minuscola (A...Z, a...z), "falso" altrimenti
Esempio	`if(isalpha(ch))` `{ ... }`

isupper

Nome funzione	**isupper**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera maiuscola (A...Z), "falso" altrimenti
Esempio	`if(isupper(ch))` `{ ... }`

islower

Nome funzione	**islower**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera minuscola (a...z), "falso" altrimenti
Esempio	`if(islower(ch))` `{ ... }`

isdigit

Nome funzione	**isdigit**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una cifra numerica (0...9), "falso" altrimenti
Esempio	`if(isdigit(ch))` `{ ... }`

isalnum

Nome funzione	**isalnum**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera oppure una cifra numerica, "falso" altrimenti. Equivalente a `isalpha(ch)||isdigit(ch)`
Esempio	`if(isalnum(ch))` `{ ... }`

isxdigit

Nome funzione	**isxdigit**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una una cifra numerica oppure una lettera valida in base 16 (a...f, A...F), "falso" altrimenti.
Esempio	`if(isxdigit(ch))` `{ ... }`

ispunct

Nome funzione	`ispunct`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un simbolo di punteggiatura (!"#$%&'()*+,-./:;<=>?@ [\]^_`{|}~), "falso" altrimenti.
Esempio	`if(ispunct(ch))` `{ ... }`

isgraph

Nome funzione	`isgraph`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un qualsiasi simbolo visibile (lettera, cifra, punteggiatura), "falso" altrimenti.
Esempio	`if(isgraph(ch))` `{ ... }`

isprint

Nome funzione	`isprint`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un qualsiasi simbolo visibile oppure lo spazio, "falso" altrimenti.
Esempio	`if(isprint(ch))` `{ ... }`

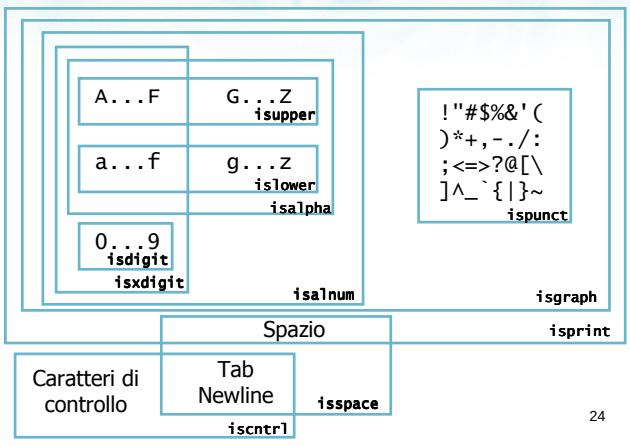
isspace

Nome funzione	`isspace`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è invisibile (spazio, tab, a capo), "falso" altrimenti.
Esempio	`if(isspace(ch))` `{ ... }`

iscntrl

Nome funzione	`iscntrl`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se `ch` è un carattere di controllo (ASCII 0...31, 127), "falso" altrimenti.
Esempio	`if(iscntrl(ch))` `{ ... }`

Vista d'insieme



Funzioni di libreria

Trasformazione di caratteri

Trasformazione di caratteri

- Definite in `<ctype.h>`
- Convertono tra lettere maiuscole e lettere minuscole
  - toupper
  - tolower

toupper

Nome funzione	**toupper**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	**char** : carattere maiuscolo
Descrizione	Se `ch` è una lettera minuscola, ritorna l'equivalente carattere maiuscolo, se no ritorna `ch` stesso
Esempio	`for(i=0; s[i]!=0; i++)` `  s[i] = toupper(s[i]) ;`

tolower

Nome funzione	**tolower**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	**char** : carattere maiuscolo
Descrizione	Se `ch` è una lettera minuscola, ritorna l'equivalente carattere maiuscolo, se no ritorna `ch` stesso
Esempio	`for(i=0; s[i]!=0; i++)` `  s[i] = tolower(s[i]) ;`

Funzioni di libreria

Copia e concatenazione

Copia e concatenazione

- Definite in `<string.h>`
- Trasferiscono il contenuto di una stringa in un'altra
  - Sostituendolo
  - Accodandolo
  - strcpy
  - strncpy
  - strcat
  - strncat

strcpy

Nome funzione	**strcpy**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa
Valore restituito	nessuno utile
Descrizione	Copia il contenuto della stringa `src` all'interno della stringa `dst` (che deve avere lunghezza sufficiente).

Esempio	`strcpy(s1, s2) ;`
	`strcpy(s, "") ;`
	`strcpy(s1, "ciao") ;`

strncpy

Nome funzione	**strncpy**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa `n` : numero max caratteri
Valore restituito	nessuno utile
Descrizione	Copia il contenuto della stringa `src` (massimo n caratteri) all'interno della stringa `dst`.
Esempio	`strncpy(s1, s2, 20) ;` `strncpy(s1, s2, MAX) ;`

strcat

Nome funzione	**strcat**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa
Valore restituito	nessuno utile
Descrizione	Accoda il contenuto della stringa `src` alla fine della stringa `dst` (che deve avere lunghezza sufficiente).
Esempio	`strcat(s1, s2) ;` `strcat(s1, " ") ;`

strncat

Nome funzione	**strncat**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa `n` : numero max caratteri
Valore restituito	nessuno utile
Descrizione	Accoda il contenuto della stringa `src` (massimo n caratteri) alla fine della stringa `dst`.
Esempio	`strncat(s1, s2) ;`

Funzioni di libreria

Confronto di stringhe

Confronto di stringhe

- Definite in `<string.h>`
- Confrontano due stringhe sulla base dell'ordine lessicografico imposto dalla tabella dei codici ASCII
  - strcmp
  - strncmp

strcmp

Nome funzione	**strcmp**
Libreria	`#include <string.h>`
Parametri in ingresso	`s1` : stringa `s2` : stringa
Valore restituito	**`int`** : risultato confronto
Descrizione	Risultato <0 se `s1` precede `s2` Risultato ==0 se `s1` è uguale a `s2` Risultato >0 se `s1` segue `s2`
Esempio	`if(strcmp(s, r)==0) {...}` `while(strcmp(r,"fine")!=0) {...}`

strncmp

Nome funzione	**strncmp**
Libreria	`#include <string.h>`
Parametri in ingresso	`s1` : stringa `s2` : stringa `n` : numero max caratteri
Valore restituito	**`int`** : risultato confronto
Descrizione	Simile a `strcmp`, ma confronta solo i primi n caratteri, ignorando i successivi.
Esempio	`if(strncmp(r, "buon", 4)==0)` (buongiorno, buonasera, buonanotte)

Funzioni di libreria

Ricerca in stringhe

Ricerca

- Definite in `<string.h>`
- Ricercano all'interno di una stringa data
  - Se compare un carattere
  - Se compare una sotto-stringa
  - Se compare una sequenza qualsiasi composta di caratteri dati

- strchr
- strstr
- strspn
- strcspn

strchr

Nome funzione	**strchr**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `ch` : carattere
Valore restituito	`==NULL` oppure `!=NULL`
Descrizione	Risultato `!=NULL` se il carattere `ch` compare nella stringa. Risultato `==NULL` se non compare.
Esempio	`if(strchr(s, '.')!=NULL)...` `if(strchr(s, ch)==NULL)...`

strstr

Nome funzione	**strstr**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	`==NULL` oppure `!=NULL`
Descrizione	Risultato `!=NULL` se la sotto-stringa `r` compare nella stringa `s`. Risultato `==NULL` se non compare.
Esempio	`if(strstr(s, "xy")!=NULL)...` `if(strstr(s, s1)==NULL)...`

strspn

Nome funzione	**strspn**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	**`int`** : lunghezza sequenza iniziale
Descrizione	Calcola la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri presenti in `r` (in qualsiasi ordine).
Esempio	`lun = strspn(s, " ") ;` `lun = strspn(s, " :,;.") ;`

strcspn

Nome funzione	**strcspn**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	**`int`** : lunghezza sequenza iniziale
Descrizione	Calcola la lunghezza della parte iniziale di `s` che è composta esclusivamente da caratteri **non presenti** in `r` (in qualsiasi ordine).
Esempio	`lun = strcspn(s, " ") ;` `lun = strcspn(s, " :,;.") ;`

Funzioni di libreria

Conversione numero-stringa

Conversioni numero-stringa

- Definite in `<stdlib.h>`
- Mettono in relazione un valore numerico (intero o reale) con la sua rappresentazione come caratteri all'interno di una stringa
  - "372" ↔ 372 (int)
  - "3.0" ↔ 3.0 (float)
  - atoi
  - atof
- In futuro:
  - sscanf
  - sprintf

atoi

Nome funzione	**atoi**
Libreria	`#include <stdlib.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`int`** : valore estratto
Descrizione	Analizza la stringa `s` ed estrae il valore intero in essa contenuto (a partire dai primi caratteri).
Esempio	`n = atoi(s) ;` `n = atoi("232abc") ;`

atof

Nome funzione	**atof**
Libreria	`#include <stdlib.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`double/float`** : valore estratto
Descrizione	Analizza la stringa `s` ed estrae il valore reale in essa contenuto (a partire dai primi caratteri).
Esempio	`x = atof(s) ;` `x = atof("2.32abc") ;`

Caratteri e stringhe

Esercizi proposti

Esercizi proposti

- Esercizio "Parola palindroma"
- Esercizio "Iniziali maiuscole"
- Esercizio "Alfabeto farfallino"

Esercizi proposti

Esercizio "Parola palindroma"

Esercizio "Parola palindroma"

- Sia data una parola inserita da tastiera.
  - Si consideri che la parola può contenere sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al massimo 30 caratteri
- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la parola inserita
  - Aggiornare la parola in modo che tutti i caratteri siano minuscoli, e visualizzarla
  - Verificare se la parola è palindroma

Palindromia

- Una parola è detta **palindroma** se può essere letta indifferentemente da sinistra verso destra e da destra verso sinistra
  - Esempi:

o	t	t	o

m	a	d	a	m

Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa minuscolo
- Verifica se è palindroma
- Stampa se è palindroma

Analisi

- Acquisisci parola
- Stampa parola
- Converti in
- Stampa mi
- Verifica se
- Stampa se

```
const int MAX = 30 ;
char parola[MAX+1] ;
printf("Inserisci parola: ") ;
scanf("%s", parola) ;
```


Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa mi
- Verifica se
- Stampa se

```
printf("Parola inserita: %s\n",
   parola) ;
```


Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa minuscolo
- Veri
- Star

```
char minusc[MAX+1] ;
int i ;

strcpy(minusc, parola) ;

for(i=0; minusc[i]!=0; i++)
{
    minusc[i] = tolower( minusc[i] ) ;
}
```

Analisi

- Acquisisci p
- Stampa par
- Converti in
- Stampa minuscolo
- Verifica se è palindroma
- Stampa se è palindroma

```
printf("Parola minuscola: %s\n",
   minusc) ;
```


Analisi

- Acquisisci p
- Stampa par
- Converti in
- Stampa mi
- Verifica se
- Stampa se

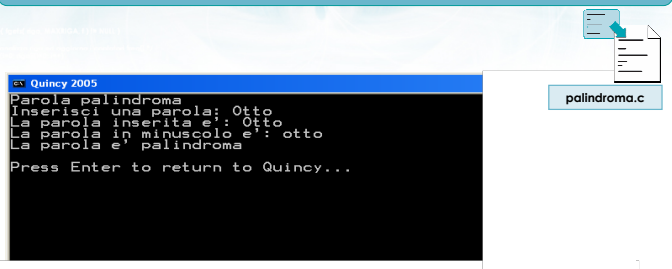
```
int i, j, palin, lun ;

i = 0 ;
j = strlen( minusc ) - 1 ;

palin = 1 ;

while( i<j && palin==1 )
{
    if(minusc[i] != minusc[j])
        palin = 0 ;
    i++ ;  j-- ;
}
```


Analisi

- Acquisisci parola
- Stampa
- Conver
- Stampa
- Verifica se è palindroma
- Stampa se è palindroma

```
if(palin==1)
    printf("E' palindroma\n") ;
else
    printf("Non e' palindroma\n") ;
```

Soluzione




```
Quincy 2005
Parola palindroma
Inserisci una parola: Otto
La parola inserita e': Otto
La parola in minuscolo e': otto
La parola e' palindroma

Press Enter to return to Quincy...
```


Esercizi proposti

Esercizio "Iniziali maiuscole"

Esercizio "Iniziali maiuscole" (1/2)

- Scrivere un programma che legga una frase introdotta da tastiera
  - La frase è terminata dall'introduzione del carattere di invio
  - La frase contiene sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al più 100 caratteri

Esercizio "Iniziali maiuscole" (2/2)

- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la frase inserita
  - Costruire una nuova frase in cui il primo carattere di ciascuna parola nella frase di partenza è stato reso maiuscolo. Tutti gli altri caratteri devono essere resi minuscoli
  - Visualizzare la nuova frase

Esempio

```
cHe bElLA gIOrnaTa
```

```
Che Bella Giornata
```


Analisi

- La frase inserita può contenere degli spazi: occorrerà usare `gets` e non `scanf`
- Ogni lettera inziale di parola va convertita in maiuscolo
- Ogni lettera non iniziale di parola va convertita in minuscolo
- Ogni altro carattere va lasciato immutato

Conversione delle lettere

iniziali.c

```
for(i=0; frase[i]!=0; i++)
{
    if( isalpha(frase[i]) &&
        ( i==0 || !isalpha(frase[i-1]) ) )
    {
        frase[i] = toupper( frase[i] ) ;
    }
    else
    {
        frase[i] = tolower( frase[i] ) ;
    }
}
```


Esercizi proposti

Esercizio "Alfabeto farfallino"

Esercizio "Alfabeto farfallino" (1/2)

- Scrivere un programma che legga una frase introdotta da tastiera
  - La frase è terminata dall'introduzione del carattere di invio
  - La frase contiene sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al più 100 caratteri

Esercizio "Alfabeto farfallino" (2/2)

- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la frase inserita
  - Costruire una nuova frase nel cosiddetto "alfabeto farfallino"
  - Visualizzare la nuova frase

L'alfabeto farfallino

- La traduzione nell'alfabeto farfallino di una parola segue le seguenti regole:
  - Tutte le consonanti sono tradotte in modo identico
  - Ogni vocale è tradotta uguale a se stessa, seguita da **altri due** caratteri:
    - la lettera 'f' (se la vocale è minuscola) o la lettera 'F' (se la vocale è maiuscola)
    - una copia della vocale stessa
- Esempi:
  - a ⟶ afa
  - con ⟶ cofon

Esempio

Vacanze di NATALE

↓

Vafacafanzefe difi NAFATAFALEFE

Approccio risolutivo

- Copiamo la stringa `frase` in una nuova stringa `farfa`
- `lun=0`
- Per ogni carattere `frase[i]`
  - Se non è una vocale, va accodato a `farfa[lun]`
  - Se è una vocale, occorre accodare a `farfa[lun]` i 3 caratteri: `frase[i]`, poi `'f'` o `'F'`, poi ancora `frase[i]`
  - Incrementare `lun` (di 1 oppure 3)
- Infine aggiungere a `farfa[lun]` il terminatore nullo

Conversione alfabeto (1/2)

farfallino.c

```
lun = 0 ;
for(i=0; frase[i]!=0; i++)
{   if( isalpha(frase[i]) )
    {   /* lettera alfabetica */
        ch = tolower(frase[i]) ;
        if( ch=='a' || ch=='e' || ch=='i'
            || ch=='o' || ch=='u' )
        {   /* vocale: trasforma */
          1 farfa[lun] = frase[i] ;
          2 if(isupper(frase[i]))
                farfa[lun+1] = 'F' ;
            else  farfa[lun+1] = 'f' ;
          3 farfa[lun+2] = frase[i] ;

            lun = lun + 3 ;
        }
```


Conversione alfabeto (2/2)

farfallino.c

```
        else
        {
            /* consonante: copia */
            farfa[lun] = frase[i] ;
            lun++ ;
        }
    }
    else
    {
        /* altro carattere: copia */
        farfa[lun] = frase[i] ;
        lun++ ;
    }
}
farfa[lun] = 0 ; /* terminatore */
```

Caratteri e stringhe

Sommario

Argomenti trattati

- Caratteri e stringhe
- Il tipo `char`
- Vettori di `char`
- Stringhe: parole, frasi
- Operazioni fondamentali sulle stringhe
  - Classificazione
  - Ricerca
  - Copia e concatenazione
  - Conversione

Tecniche di programmazione

- Terminatore nullo
- Librerie `<string.h>` e `<ctype.h>`
- Manipolazione di stringhe come vettori di caratteri
- Manipolazione di stringhe attraverso le funzioni di libreria
- Identificazione di parole all'interno di frasi

Stringhe e vettori

- Molte operazioni sulle stringhe sono ricondotte ad analoghe operazioni sui vettori
- Molti problemi "astratti" su vettori numerici assumono forma più "concreta" nel caso delle stringhe
- I cicli sono solitamente governati dal controllo del terminatore di fine stringa

Errore frequente

- L'input/output nelle stringhe è spesso problematico
- Utilizzando la funzione `scanf`, in presenza di spazi interni alla stringa rimarranno dei caratteri "non letti", che daranno fastidio alle successive `scanf`
- Quando possibile, ricorrere alla funzione `gets` per la lettura di una stringa

Errore frequente

- Le stringhe hanno lunghezza variabile; i vettori che le contengono hanno lunghezza fissa
- È possibile, con una chiamata a `strcpy` o `strcat`, scrivere oltre la dimensione del vettore
  - Grave errore di programmazione, che può portare alla corruzione di dati in altre variabili
- Abituarsi a verificare la lunghezza prima di copiare le stringhe

```
if(strlen(a) + strlen(b) + 1 <= MAX)
    strcat(a,b) ;
else
    ERRORE!!!
```

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo