TIPI STRUTTURATI: STRINGHE

Fondamenti di Programmazione 2021/2022

Francesco Tortorella

Array monodimensionali

- Un array è un insieme di variabili, tutte dello stesso tipo, identificato da un nome unico. Gli elementi dell'array sono disposti in memoria in posizioni consecutive.

```
int v[20];
float w[10];
```

- Quali sono le caratteristiche di un array di caratteri?

Insiemi di caratteri

- Ad una variabile di tipo **`char`** si può assegnare un qualsiasi carattere:

```
char ch;

ch = 'a';    /* lower-case a */
ch = 'A';    /* upper-case A */
ch = '0';    /* zero         */
ch = ' ';    /* space        */
```

- Si noti le costanti **`char`** sono espresse fra apici ' ', non virgolette " ".

Operazioni con i caratteri

- È possibile fare “operazioni” con i caratteri in quanto il linguaggio C tratta i caratteri come interi
- Nel codice ASCII, i codici dei caratteri vanno da 0000000_2 a 1111111_2, che corrispondono agli interi decimali da 0 to 127.

Il carattere `'a'` corrisponde al valore 97,

`'A'` al valore 65,

`'0'` al valore 48,

`' '` al valore 32.

Codice ASCII

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	‘
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del

Operazioni con i caratteri

- Quando un carattere viene usato in un'espressione, esso viene valutato con la sua rappresentazione come numero intero
- Ad esempio, usando il codice ASCII:

```
char ch;
int i;

i = 'a';          /* i diventa 'a' cioè 97 */
ch = 65;          /* ch diventa 'A' */
ch = 'A';         /* ch diventa 'A' – meglio! */
ch = ch + 1;      /* ch diventa 'B' */
ch++;             /* ch diventa 'C' */
```

Operazioni con i caratteri

- I caratteri possono essere confrontati (visto che sono di fatto numeri).
- Questa caratteristica può essere utilmente sfruttata.
- Che cosa fa questa istruzione?

```
if (ch >= 'a' && ch <= 'z')
   ch = ch - 'a' + 'A';
```

Operazioni con i caratteri

- Che cosa fa questo ciclo?

```
for (ch = 'A'; ch <= 'Z'; ch++)
  printf("%c\n", ch);
```

Funzioni per i caratteri

- La funzione di libreria **`toupper`** converte un carattere da minuscolo a maiuscolo:

```
ch = toupper(ch);
```

- **`toupper`** restituisce la versione maiuscola del suo argomento
- **`tolower`** restituisce invece la versione minuscola
- Per usare la funzione toupper è necessario includere il file **`ctype.h`**

```
#include <ctype.h>
```

- La libreria standard del C fornisce molte altre funzioni utili per manipolare i singoli caratteri e le stringhe di caratteri

Funzioni per i caratteri

```
#include <stdio.h>
#include <ctype.h>
int main(){
   char small_ch, capital_ch;
   small_ch = 'a';
   capital_ch = toupper(small_ch);
   printf("%c %c\n", small_ch, capital_ch);
}
```

Funzioni di I/O per i caratteri

- Lo specificatore **`%c`** permette alle funzioni **`scanf`** e **`printf`** di leggere e stampare caratteri:

```
char ch;

scanf("%c", &ch);   /* legge un carattere */
printf("%c", ch);   /* scrive un carattere */
```

- **`scanf`** non ignora i caratteri di spazio bianco (come fa per i numeri)
- Per forzare la scanf a saltare uno spazio prima di leggere il carattere è necessario esplicitare la presenza dello spazio nell'input prima dello specificatore **`%c`**:

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

Funzioni di I/O per i caratteri

- Poichè **`scanf`** non ignora gli spazi bianchi è facile accorgersi della fine di una linea di input: basta controllare che il carattere appena letto sia il carattere “newline”
- Ecco un ciclo che legge ed ignora tutti i rimanenti caratteri fino al newline:

```
do {
   scanf("%c", &ch);
} while (ch != '\n');
```

- La prossima chiamata a **`scanf`** leggerà il primo carattere della prossima linea di input

Funzioni di I/O per i caratteri

- Per l'input e l'output di caratteri singoli si possono usare le funzioni **`getchar`** e **`putchar`** al posto di **`scanf`** e **`printf`**
- **`putchar`** scrive un carattere:

```
putchar(ch);
```

- Invece **`getchar`** legge un nuovo carattere dall'input ad ogni chiamata; il carattere letto è fornito come valore restituito dalla funzione:

```
ch = getchar();
```

  - **`getchar`** restituisce un **`int`** non un **`char`**, quindi **`ch`** dovrà avere tipo **`int`**.
  - Come **`scanf`**, **`getchar`** non tralascia lo spazio bianco

Funzioni di I/O per i caratteri

- Usare **`getchar`** e **`putchar`** (al posto di **`scanf`** e **`printf`**) rende i programmi più chiari e semplici.
  - **`getchar`** e **`putchar`** sono molto più semplici di **`scanf`** e **`printf`**, che sono progettate per leggere e scrivere molti tipi di dati in molti formati
  - Spesso sono implementate come macro per renderle ancora più veloci
- **`getchar`** ha anche un altro vantaggio. Poichè restituisce il valore del carattere che legge permette di semplificare il codice sorgente in molte situazioni tipiche

Funzioni di I/O per i caratteri

- Consideriamo il ciclo con la **scanf** che abbiamo usato per ignorare i caratteri rimanenti fino al newline

```
do{
   scanf("%c", &ch);
} while (ch != '\n');
```

- Riscrivendo il ciclo con **getchar** otteniamo:

```
do{
   ch = getchar();
} while (ch != '\n');
```

Funzioni di I/O per i caratteri

- La chiamata a **`getchar`** può essere spostata nell'espressione di controllo del ciclo:

```
while ((ch = getchar()) != '\n')
   ;
```

- La variabile **`ch`** non è necessaria; è sufficiente che il valore restituito da **`getchar`** sia confrontato con il carattere newline:

```
while (getchar() != '\n')
   ;
```

Funzioni di I/O per i caratteri

- **`getchar`** è utile in cicli che ignorano caratteri fino al raggiungimento di un un particolare carattere
- Esempio: istruzione che usa **`getchar`** per saltare un numero qualsiasi di spazi bianchi:

```
while ((ch = getchar()) == ' ')
    ;
```

- Al termine del ciclo, **`ch`** conterrà il primo carattere dell'input diverso dallo spazio

Funzioni di I/O per i caratteri

- Bisogna fare attenzione quando si usa sia **`getchar`** che **`scanf`**.
- **`scanf`** normalmente lascia nel buffer di input i caratteri che ha "controllato" ma non ha letto come, ad esempio, il newline:

```
int i, ch;
printf("Inserisci un intero: ");
scanf("%d", &i);
printf("Inserisci un carattere: ");
ch = getchar();
```

- In questo caso **`getchar`** leggerà il newline.

Funzioni di I/O per i caratteri

- Per risolvere il problema, possiamo svuotare esplicitamente il buffer da tutto quanto è stato lasciato da scanf:

```
int i, ch;
printf("Inserisci un intero: ");
scanf("%d", &i);
while(getchar() != '\n')
    ;

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

- In questo caso **`getchar`** leggerà correttamente il carattere richiesto.

Esempio: calcolare la lunghezza di una frase

- Supponiamo di voler calcolare la lunghezza di una frase inserita dall'utente

```
Inserisci una frase: Pippo dammi la mela.
Numero caratteri: 20
```

- La lunghezza della frase include gli spazi ed i caratteri di punteggiatura, ma non il newline alla fine della riga inserita da tastiera.
- Possibile usare sia **`scanf`** che **`getchar`** per leggere i caratteri

Esempio: calcolare la lunghezza di una frase

```
#include <stdio.h>

int main(void)
{
  char ch;
  int len = 0;

  printf("Inserisci una frase: ");
  ch = getchar();
  while (ch != '\n') {
    len++;
    ch = getchar();
  }

  printf("Numero caratteri: %d\n", len);

  return 0;
}
```

Possibile fare meglio?

Esempio: calcolare la lunghezza di una frase

```
#include <stdio.h>

int main(void)
{
  char ch;
  int len = 0;

  printf("Inserisci una frase: ");

  while (getchar() != '\n') {
    len++;
  }
  printf("Numero caratteri: %d\n", len);

  return 0;
}
```

Esercizio

- Scrivere un programma che riceve in input una frase e modifica in maiuscolo tutti caratteri alfabetici minuscoli.
- Gli altri caratteri resteranno invariati.

Esercizio

```
#include <stdio.h>

int main(void)
{
  char ch;
  while ((ch = getchar()) !='\n')
    if(ch >= 'a' && ch <= 'z')
        putchar(ch+'A'-'a');
    else
        putchar(ch);
  return 0;
}
```

Le Stringhe

Le stringhe

- Le stringhe sono **sequenze** di caratteri
- Possono essere
  - costanti (dette letterali)
  - variabili (cioè variabili il cui valore è una stringa)

- **Le stringhe sono realizzate tramite array di caratteri**
  - Il carattere speciale **'\0'** segnala la fine della stringa

Stringhe letterali

- Una **stringa letterale** è una sequenza di caratteri racchiusi fra doppi apici:

```
"Essere o non essere, questo è il problema."
```

- Stringhe letterali possono contenere sequenze di escape
- Sono spesso usate nelle stringhe di formato per **`printf`** e **`scanf`**

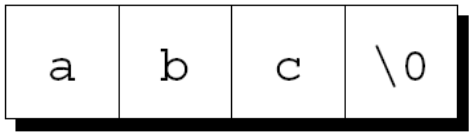
```
"buongiorno \n a tutti !!\n"
```

Memorizzazione delle stringhe letterali

- Una stringa letterale di lunghezza contenente N caratteri richiede N + 1 byte di memoria
- Agli N caratteri della stringa va infatti aggiunto il carattere '\0' che segnala la fine della stringa
- '\0' è anch'esso una sequenza di escape ed ha codice ASCII 0

Memorizzazione delle stringhe letterali

- La stringa letterale **`"abc"`** è memorizzata in un array di 4 caratteri:



- La stringa vuota "" è memorizzata in un array di 1 carattere:

Stringhe letterali e caratteri costanti

- Una stringa letterale che contiene un solo carattere è diversa da una costante carattere
  - **`"a"`** è rappresentata da un array di due **`char`**
  - **`'a'`** è rappresentata da un **`char`**

Stringhe variabili

- Una variabile stringa è realizzata come un array di **`char`**.
- La sequenza dei caratteri utili è terminata dal carattere '\0'.
- Se una stringa può contenere 80 caratteri, deve essere quindi dichiarata come array di 81 **`char`**:

```
#define STR_LEN 80
…
char str[STR_LEN+1];
```

- L'elemento in più serve per ospitare il carattere di fine stringa.

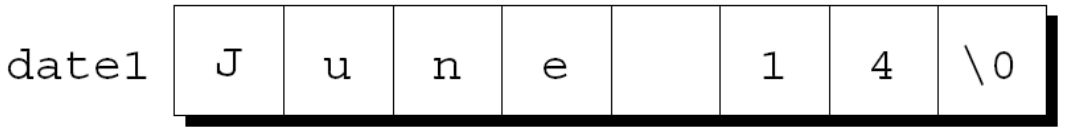
Inizializzazione di una stringa

- Una stringa variabile può essere inizializzata al momento della definizione:

```
char date1[8] = "June 14";
```

- L'inizializzazione equivale a

```
char date1[] = {'J','u','n','e',' ','1','4','\0'};
```

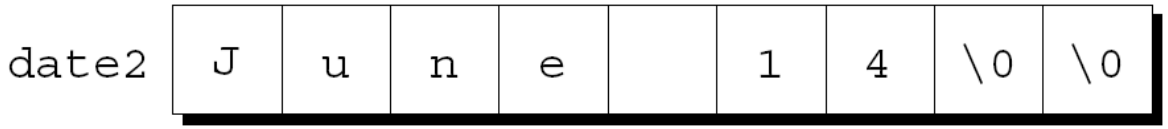


- In questo caso saranno allocati 8 caratteri

Inizializzazione di una stringa

- Se la stringa costante di inizializzazione ha meno caratteri della dimensione dell'array, gli elementi in più saranno inizializzati con '\0':

```
char date2[9] = "June 14";
```



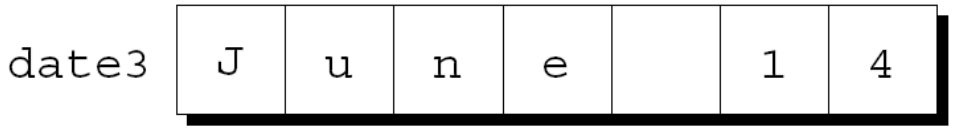
- Attenzione quindi a distinguere
  - Lunghezza della stringa: numero di caratteri utili (i.e. prima del '\0') presenti nella stringa
  - Dimensione dell'array che ospita la stringa: è il numero di caratteri con cui si è definito l'array

Inizializzazione di una stringa

- Se la stringa costante di inizializzazione ha più caratteri della dimensione dell'array, possono succedere guai seri.
- Che cosa succede con la seguente definizione?

```
char date3[7] = "June 14";
```

- Non c'è spazio per il carattere di fine stringa che non sarà quindi memorizzato:



- **In questo caso l'array non è utilizzabile come una stringa**

Inizializzazione di una stringa

- È più sicuro omettere la lunghezza dell'array nella definizione con inizializzazione di una stringa variabile.
- In tale caso l'array conterrà il numero necessario di **`char`**

```
char date4[] = "June 14";
```

- ACHTUNG! Questo va bene se e solo se la stringa variabile dovrà contenere al massimo 8 caratteri.
- Altrimenti sarà bene precisare una dimensione adeguata per l'array. Es:

```
char date4[21] = "June 14";
```

I/O di stringhe

- Per stampare una stringa, è possibile usare **`printf`** con lo specificatore **`%s`**

```
include <stdio.h>
#define LENGTH 20
int main(void)
{
   char st[LENGTH+1] = "Paperino";
   printf("Stringa: %s\n", st);

   return 0;
}
```

Quanti caratteri saranno stampati?

I/O di stringhe

- Per leggere una stringa si può usare **`scanf`** con lo specificatore **`%s`**, **ma …**

```
include <stdio.h>
#define LENGTH 20
int main(void)
{
    char st[LENGTH+1];
    printf("Inserire una stringa: ");
    scanf("%s", st);
    printf("Stringa letta: %s\n", st);

    return 0;
}
```

I/O di stringhe

- Nel leggere una stringa, scanf acquisisce qualunque carattere diverso da un blank (spazio, tab, newline), fermandosi al primo carattere blank trovato.
- Il carattere di terminazione '\0' viene poi aggiunto automaticamente alla fine della sequenza memorizzata.

I/O di stringhe

```
include <stdio.h>
#define LENGTH 20
int main(void)
{
   char st[LENGTH+1];
    printf("Inserire una stringa: ");
    scanf("%s", st);
    printf("Stringa letta: %s\n", st);

   return 0;
}
```

```
Inserire una stringa: pippo
Stringa letta: pippo

Inserire una stringa: paperino
Stringa letta: paperino

Inserire una stringa: qui quo
Stringa letta: qui
```

Lettura alternativa di stringa

```
include <stdio.h>
#define LENGTH 20
int main(void)
{
   char ch, st[LENGTH+1];
   int i=0;
   printf("Inserire una stringa: ");
   while ((ch = getchar()) !='\n') {
      st[i] = ch;
      i++;
   }
   st[i]='\0';

   printf("Stringa letta: %s\n", st);
   return 0;
}
```

Lettura alternativa di stringa

```
include <stdio.h>
#define LENGTH 20
int main(void)
{
   char ch, st[LENGTH+1];
   int i=0;
   printf("Inserire una stringa: ");
   while (i < LENGTH &&(ch = getchar()) !='\n') {
      st[i] = ch;
      i++;
   }
   st[i]='\0';

   /* Svuota il buffer dai caratteri rimanenti */
   while(getchar() != '\n')
     ;

   printf("Stringa letta: %s\n", st);
   return 0;
}
```

Questa modifica alla condizione del ciclo WHILE evita che si inserisca un numero di caratteri superiori alla dimensione dell'array di caratteri. In questo caso solo parte dei caratteri forniti in input sarà memorizzata nell'array. La restante parte rimarrà nel buffer di input (e sarà eliminata dal ciclo successivo).

Funzioni per la gestione delle stringhe

- Molte funzioni per la elaborazione e manipolazione di stringhe sono disponibili nella libreria C e dichiarate in **`string.h`**.
- Per utilizzarle, va quindi inserito
  `#include <string.h>`

Funzioni per la gestione delle stringhe

strcpy() `char* strcpy(char dest[], char sorg[]);`
Copia la stringa `sorg` nella stringa `dest`. Restituisce `dest`.

strlen() `size_t strlen (char s[]);`
Restituisce la lunghezza della stringa `s`.

strcat() `char* strcat(char dest[], char sorg[]);`
Aggiunge una copia della stringa `sorg` alla fine di `dest`. Restituisce `dest`.

strcmp() `int strcmp(char s[], char t[]);`
Confronta le stringhe `s` e `t` e restituisce:

0	se le due stringhe sono uguali
<0	se `s` precede `t` in ordine alfabetico
>0	se `s` segue `t` in ordine alfabetico

Assegnazione tra stringhe

- Come per gli array non è possibile fare assegnazioni dirette tra variabili stringa.
- Una possibilità per assegnare i caratteri di una stringa ad un'altra stringa è quella di fare una serie di assegnazioni tra elementi corrispondenti.
- Tuttavia, vista la frequenza di un'operazione del genere, si usa la funzione **`strcpy`** che la realizza direttamente:

```
char nome1[6]="Pippo";
char nome2[6];

strcpy(nome2,nome1);
```