PUNTATORI

Introduzione
Dichiarazione ed inizializzazione delle variabili di tipo puntatore
L'operatore puntatore
Chiamata di funzioni per indirizzo
Espressioni ed aritmetica dei puntatori
Puntatori ed array
Puntatori a funzioni

INTRODUZIONE - I puntatori

Una *variabile* è un'area di memoria a cui viene dato un nome:

```
int x;
```

La dichiarazione precedente riserva un'area di memoria che viene individuata dal nome x. Il vantaggio di questo approccio è che *è possibile accedere al valore memorizzato mediante il suo nome*. La seguente istruzione salva il valore 10 nell'area di memoria identificata dal nome x:

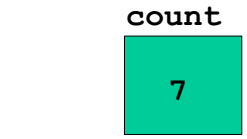
```
x =10;
```

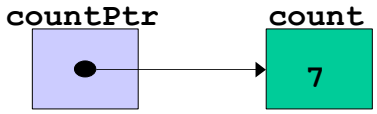
Il calcolatore fa accesso alla propria memoria non utilizzando i nomi delle variabili ma utilizzando una mappa della memoria in cui *ogni locazione viene individuata univocamente da un numero, chiamato indirizzo della locazione di memoria*.

Dichiarazione ed inizializzazione dei puntatori

Le variabili di tipo puntatore:

- Contengono come valore un indirizzo di memoria
- Le variabili normali (count) contengono uno specifico valore (indirizzamento diretto)
- I puntatori (countPtr) contengono l'indirizzo di una variabile (count) che a sua volta contiene un valore specifico (indirizzamento indiretto)





Puntatore

Un **puntatore** è una variabile che memorizza l'*indirizzo* di una locazione di memoria, cioè l'indirizzo di una variabile.

Un puntatore deve essere dichiarato come qualsiasi altra variabile, in quanto anch'esso è una variabile. Per esempio:

int *p;

la dichiarazione specifica una variabile di tipo: *puntatore ad un intero*.

L'introduzione del carattere * davanti al nome della variabile indica che si tratta di un puntatore del tipo dichiarato. Si noti che l'asterisco si applica alla singola variabile, non al tipo. Quindi:

int *p , q;

dichiara un puntatore ad un intero (variabile p) ed un intero (variabile q).

Dichiarazione ed inizializzazione di puntatori (cont.)

- Dichiarazione di puntatori
  - **Per i puntatori si usa il simbolo ***

    int *myPtr;
  - Questo dichiara myPtr come puntatore ad **int** (puntatore di tipo **int ***)
  - Se ci sono più dichiarazioni di puntatori, si usano più *

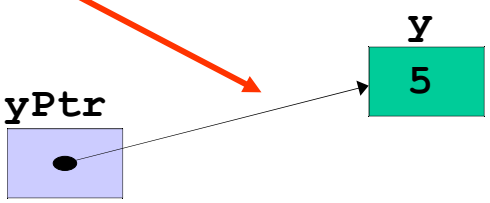
    int *myPtr1, *myPtr2;
  - Si possono dichiarare puntatori a qualsiasi tipo
  - I puntatori si inizializzano a **0**, **NULL**, o ad un indirizzo
    - **0** o **NULL** – puntano a nessun indirizzo (**NULL** è preferibile)

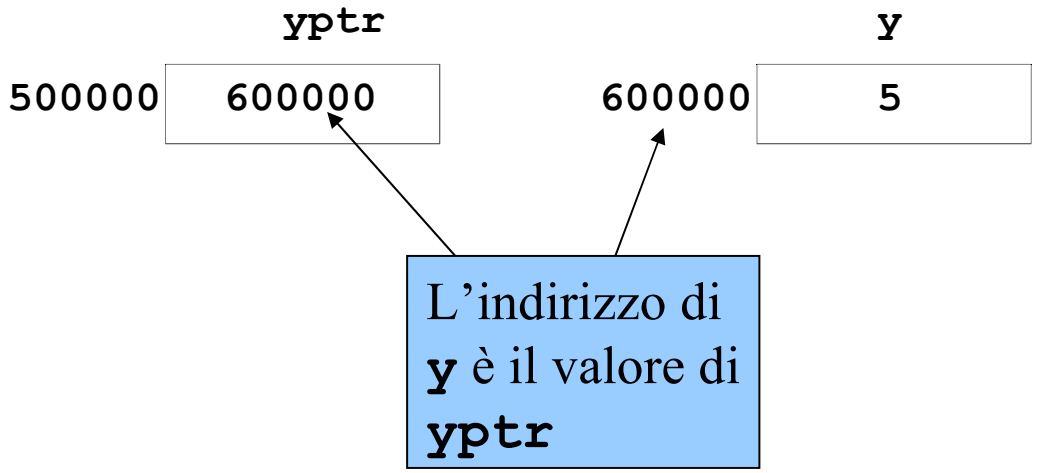
Operatori & e * e puntatori

- **& (operatore indirizzo)**
  - Restituisce l'indirizzo dell'operando

```
int y = 5;                    /*variabile intera y*/
int *yPtr;          /*variabile puntatore ad intero yPtr*/
yPtr = &y;   /*yPtr restituisce l'indirizzo di y */
```

  - `yPtr` "punta a" `y`





Operatori & e *

L'operatore & restituisce l'indirizzo di una variabile. Si consideri lo stralcio di codice seguente:

```
int *p , q, n;

...
p = &q;                 /* -1- */
n = q;                  /* -2- */
```

L'effetto dell'istruzione 1 è memorizzare l'indirizzo della variabile q nella variabile p. Dopo questa operazione, p *punta* a q.

L'operatore * ha il seguente effetto: se applicato ad una variabile puntatore restituisce il valore memorizzato nella variabile a cui punta:

p memorizza l'*indirizzo*, o *punta*, ad una variabile

*p restituisce il *valore* memorizzato nella variabile a cui punta p.

L'operatore * viene chiamato operatore di *dereferenziazione*.

Operatori e puntatori (cont.)

* (operatore di dereferenziazione)
  - Restituisce un sinonimo/alias di quello a cui *punta* il suo operando

    ***yptr** restituisce **y** (perchè **yptr** punta a **y**)
  - * Può essere usato nelle assegnazioni
    - Restituisce l'alias di un oggetto

    ***yptr = 7; /* cambia y in 7 */**

Per riassumere

- Per **dichiarare un puntatore**, mettere * davanti al nome.
- Per **ottenere l'indirizzo di una variabile**, utilizzare & davanti al nome.
- Per **ottenere il valore di una variabile**, utilizzare * di fronte al nome del puntatore.

Si consideri lo stralcio di codice seguente:

```
int *a , b , c;
b = 10;
a = &b;          /* a 'punta' a b */
c = *a;          /*c contiene il valore della variabile puntata da a*/
```

Delle tre variabili dichiarate, a è un puntatore ad intero, mentre b e c sono interi.

La **prima istruzione** salva il valore 10 nella variabile b.

La **seconda istruzione** salva in a il valore dell'indirizzo della variabile b. Dopo questa istruzione a *punta* a b.

Infine, la **terza istruzione** memorizza il valore della variabile puntata da a (ossia b) in c, quindi viene memorizzato in c il valore di b (10).

OSSERVAZIONI

•Si noti che **b è un int** e **a è un puntatore ad un int,** quindi

b = a;

è un errore perchè cerca di memorizzare un *indirizzo* in un int.

•In modo analogo:

b = &a;

cerca di memorizzare l'*indirizzo* di un puntatore in un int ed è altrettanto errato.

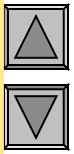
•L'unico assegnamento sensato tra un int ed un puntatore ad un int è:

b = *a;

Cioè a b si assegna il valore della variabile puntata da a

•* e **&** sono uno l'inverso dell'altro, quindi si possono "eliminare" a vicenda:

- ***&yptr -> * (&yptr) -> * (indirizzo di yptr)-> restituisce un alias di ciò a cui *punta* l'operando -> yptr**
- **&*yptr -> &(*yptr) -> &(y) -> restituisce l'indirizzo di y, che *è* yptr -> yptr**



Outline

```c
/* USO DEGLI OPERATORI & E * */

#include <stdio.h>

int main()
{
   int a;        /* a è un intero */
   int *aPtr;    /* aPtr è un puntatore ad intero */

   a = 7;
   aPtr = &a;    /* aPtr contiene l'indirizzo di a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are inverses of "
           "each other.\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   return 0;
}
```

Dichiarazione e inizializzazione delle variabili

L'indirizzo di **a** è il valore di **aPtr.**

L'operatore ***** restituisce un alias di ciò a cui punta il suo operando. **aPtr** punta ad **a**, quindi ***aPtr** restituisce **a**.

Stampa

Si noti che ***** e **&** sono uno l'inverso dell'altro

```
The address of a is 0012FF88
The value of aPtr is 0012FF88

The value of a is 7
The value of *aPtr is 7
Proving that * and & are complements of each other.
&*aPtr = 0012FF88
*&aPtr = 0012FF88
```

Output del programma

Chiamata delle funzioni per indirizzo

Chiamata per indirizzo tramite l'uso dei puntatori

Si passa l'indirizzo del parametro con l'operatore **&**

Consente di modificare la posizione di memoria

Gli array non si passano con il simbolo **&** perchè il nome dell'array è già un indirizzo

Operatore *

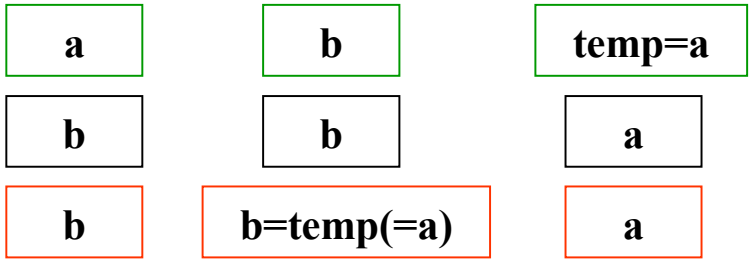
Si usa come alias per le variabili all'interno di una funzione

```
void double(int *number) /*a double si passa un puntatore a
                           interi*/
 {
*number = 2 * (*number); /*il valore puntato viene
                           raddoppiato*/
 }
```


Passaggio di una variabile o del puntatore alla variabile

Si consideri a titolo d'esempio il seguente problema: si scriva una funzione che scambia il contenuto di due variabili. In linea di massima l'operazione dovrebbe essere semplice: scrivere una funzione scambia(a,b) che scambia il contenuto di a e b.

```
void scambia(int a , int b)
{
int temp;
temp = a;
a = b;
b = temp;
printf("funz: var1 = %d var2 = %d\n", a, b);
}
```



Questo non scambia a con b all'esterno della funzione. Infatti, il passaggio dei parametri in C per valore, fa una copia del valore della variabile che viene passata e su questa copia agisce la funzione.

Quindi se si considera lo stralcio di codice seguente che effettua la **chiamata** alla funzione scritta, otterremmo un risultato diverso da quello desiderato:

...

```
int x, y;
x = 18;
y = 22;
printf("prima: var1 = %d var2 = %d\n", x, y);
scambia(x, y);
printf("dopo: var1 = %d var2 = %d\n", x, y);
...
```

A video si ottiene:

Prima della chiamata: var1 = 18 var2 = 22
All'interno della funzione: var1 = 22 var2 = 18
Dopo la chiamata: var1 = 18 var2 = 22

La soluzione al problema è passare (sempre per valore) il *riferimento* alle variabili, ossia il loro *indirizzo* cosicché la funzione possa accedere direttamente alla memoria (tramite appunto l'indirizzo) e modificarne il valore.

Alla funzione vengono quindi passati gli indirizzi delle variabili; la funzione corretta è dunque:

```
scambia (int *a , int *b);
{
int temp;
temp = *a;          /*temp contiene il valore della variabile puntata da a*/
*a = *b;            /*la variabile puntata da a contiene la variabile
                    puntata da b*/
*b = temp;          /*la variabile puntata da b contiene il valore di temp
                                (cioè a)*/
printf("funz2: var1 = %d var2 = %d\n", *a, *b);
}
```

Si noti che i due parametri a e b sono puntatori e quindi per scambiare il valore è necessario utilizzare gli operatori di *dereferenziazione* per far sì che i *valori* delle *variabili* a cui puntano vengano scambiati.

Infatti *a è il contenuto della variabile a cui *punta* a.

Naturalmente anche la chiamata della funzione deve essere adattata: è necessario passare due indirizzi e non più due interi. Quindi, il codice è il seguente:

```
...
int x, y;
x = 18;
y = 22;
printf("prima: var1 = %d var2 = %d\n", x, y);
scambia(&x, &y);          /*passo gli indirizzi degli operandi*/
printf("dopo: var1 = %d var2 = %d\n", x, y);
...
```

A video si ottiene:

```
prima: var1 = 18 var2 = 22
funz: var1 = 22 var2 = 18
dopo: var1 = 22 var2 = 18
```

che è ciò che si desidera.

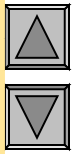
La regola è che ogniqualvolta si passa ad una funzione una variabile il cui contenuto deve essere modificato, è necessario passare l'indirizzo della variabile.

Si faccia attenzione che chiamare la funzione passando il valore della variabile invece del suo indirizzo quando il parametro dichiarato è l'indirizzo causa lo scambio di contenuto di due aree casuali della memoria, provocando danni eventualmente all'intero sistema e non solo al programma in esecuzione!

Nota:

La necessità di passare l'indirizzo ad una funzione spiega anche il perchè le due funzioni di I/O printf e scanf sono diverse.

La funzione printf non modifica il valore dei suoi parametri, quindi viene chiamata con printf("%d", a), ma *la funzione scanf modifica il valore della variabile, per memorizzarci quello appena acquisito*, quindi viene chiamata con scanf("%d", &a).



Outline

```
1
2     /* Cubo di un numero con chiamata alla funzione per indirizzo
3     con l'uso di un puntatore */
4
5  #include <stdio.h>
6
7  void cubeByReference( int * );    /* pr
8  ).
9  int main()
10 {
11    int number = 5;
12
13    printf( "The original value of number is %d", number );
14    cubeByReference( &number );
15    printf( "\nThe new value of number is %d\n", number );
16
17    return 0;
18 }
19
20 void cubeByReference( int *nPtr )
21 {
22    *nPtr = *nPtr * *nPtr * *nPtr;  /* cube number in main */
23 }
```

cubeByReference ha in ingresso un puntatore (l'indirizzo di una variabile

Prototipo – ha in ingresso un puntatore a `int`.

Inizializzazione delle variabili

Chiamata della funzione

All' interno di **cubeByReference**, si usa ***nPtr** (***nPtr** è **number**).

Definizione della funzione

```
The original value of number is 5
The new value of number is 125
```

Output

Puntatori e array

In C c'è uno stretto legame tra puntatori e array: nella dichiarazione di un array di fatto si dichiara un puntatore a al primo elemento dell'array:

Dichiarare int a[10];

equivale a individuare &a[0]; (=indirizzo del primo elemento di a)

L'unica differenza tra a e una *variabile puntatore* è che **il nome dell'array è un puntatore costante**: non si modifica la posizione a cui punta (altrimenti si perde una parte dell'array).

Quando si scrive un'espressione come a[i] questa viene convertita in un'espressione a puntatori che restituisce il valore dell'elemento appropriato.

Più precisamente, **a[i] è equivalente a *(a + i)** ossia il valore a cui punta a + i. In modo analogo *(a + i) è uguale ad a[i] e così via.

Per riassumere

Il nome di un array è un ***puntatore costante*** al primo elemento dell'array, ossia a == &a[0] e *a == a[0].

L'accesso agli array mediante indici è equivalente all'aritmetica dei puntatori, ossia a+i = &a[i] e *(a+i) == a[i].

Inoltre, per il passaggio di un array ad una funzione si passa il puntatore all'array.

Questo consente di scrivere funzioni che possono accedere all'intero array senza dover passare ogni singolo valore contenuto nell'array: si passa il puntatore al primo elemento (e, in linea di massima, il numero degli elementi presenti).

Esempio

Si consideri il seguente esempio: scrivere una funzione che riempie un array di interi con numeri casuali (mediante la funzione di libreria rand()).

```
void main()       /* stralcio di programma chiamante */
{
int numeri[NMAX], i;
 ...
riempirandom(numeri, NMAX);  /*chiamata alla funzione
riempirandom*/
...
}
void riempirandom(int a[] , int n)       /*definizione di riempirandom*/
{
 int i;
for (i = 0; i< n ; i++)
a[i] = rand()%n + 1;          /* rand()%n + 1 genera il valore casuale */
}
```


Espressioni ed aritmetica dei puntatori

Sui puntatori si possono effettuare operazioni aritmetiche

- Incremento /decremento (++ o --)
- Somma o sottrazione di un intero ad un puntatore(+ o += , - o -=)
- I puntatori possono essere sottratti fra di loro

Sommare uno ad un puntatore ad array significa farlo passare al successivo elemento

Ad esempio, in generale un int occupa due byte, un float quattro. Quindi se si dichiarare un array di int e si somma uno al puntatore all'array, il puntatore si muoverà di due byte. D'altra parte, se si dichiara un array di float e si somma uno al puntatore all'array, questo si muove di quattro byte.

Ciò che è sufficiente ricordare è che l'aritmetica dei puntatori si basa su unità del tipo di dati trattato.

È possibile utilizzare gli operatori di incremento e decremento (++ e --) con i puntatori, ma *non con il nome dell'array*, perchè quello è un puntatore costante e non può essere modificato

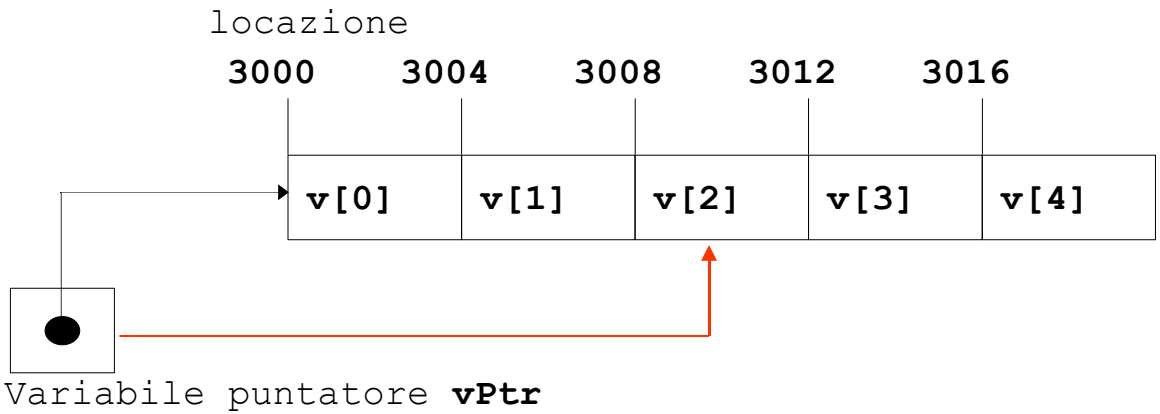
Espressioni ed aritmetica dei puntatori (cont.)

- Array di 5 elementi di tipo **`int`** su una macchina con interi di 4 byte

  `vPtr` punta al primo elemento **`v[0]`**

  Alla locazione di memoria **`3000`**. (**`vPtr = 3000`**)

  - **`vPtr += 2;`** pone **`vPtr`** pari a **`3008`**
    - **`vPtr`** punta a **`v[2]`** (incrementato di 2), ma su una macchina con interi a 4 byte.



Espressioni ed aritmetica dei puntatori (cont.)

- Sottrazione di puntatori
  - Restituisce il numero di elementi fra i due puntatori.

```
vPtr2 = v[2];
vPtr = v[0];
vPtr2 - vPtr == 2.
```

- Puntatori dello stesso tipo possono essere assegnati uno all'altro
  - Tranne il puntatore a **`void`** (tipo **`void *`**)
    - Infatti questo è un puntatore generico e rappresenta tutti i tipi
    - **`void`** non può essere dereferenziato

Scansione e visualizzazione di un vettore di interi

```
#include <stdio.h>
int a[5] = { 1, 2, 3, 4, 5 };

main()
{
  int i, *p;
  p = a;
  printf("Gli elementi del vettore sono:\n\n");

  for (i = 0; i <= 4; i++)
    printf("a[%d] = %d\n", i, *p++);
}
```


Puntatori ed array

Il nome dell'array è un puntatore costante

- Per dichiarare un puntatore **bPtr** all'array **b[5]:**

  bPtr = b;

  Poichè il nome dell'array è l'indirizzo del suo primo elemento

  Oppure: **bPtr = &b[0]**

  Assegna in modo esplicito **bPtr** all'indirizzo del primo elemento dell'array

I puntatori possono indirizzare i singoli elementi dell'array

- Per accedere all'elemento **b[n]**

  Si può usare ***(bPtr + n)**

  Il nome stesso dell'array può essere usato per l'aritmetica dei puntatori:

  b[3] corrisponde a ***(b + 3)**

  Si può fare riferimento ai singoli elementi del vettore con

  bPtr[3] che equivale a **b[3]**

Immissione sequenza numeri

La funzione non restituisce alcun valore, ma ha in ingresso il puntatore alla variabile intera n, che può essere modificata

```
void immissione(int *pn, int *vet)
{
int i, n;
char invio;
do {
  printf("\nNumero elementi: ");
  scanf("%d", &n);
}
while (n < 1 || n > MAX_ELE);

for(i = 0; i < n; i++)
{
  printf("\nImmettere un intero n.%d: ",i);
  scanf("%d", &vet[i]);
}

*pn = n;
}
```


Puntatori a funzioni

- Puntatori a funzioni
  - Contengono l'indirizzo di memoria della funzione
  - Analogamente ai vettori, il nome dlla funzione è l'indirizzo di memoria iniziale del codice della funzione
- I puntatori a funzioni possono
  - Essere passati a funzioni
  - Essere memorizzati in vettori
  - Essere assegnati ad altri puntatori a funzioni

CARATTERI E STRINGHE

Caratteri e stringhe
Funzioni della libreria standard I/O
Funzioni della libreria di gestione delle stringhe

Caratteri e stringhe

- Caratteri
  - Sono i blocchi costitutivi di ogni programma: Ogni programma è una successione di caratteri raggruppati opportunamente
  - Si indicano fra apici: **'z'** rappresenta il carattere **z**
- Stringhe
  - Serie di caratteri trattati come una singola unità
    - Possono comprendere lettere, numeri ed alcuni simboli speciali (*, /, $)
  - Le stringhe letterali (costanti) – si scrivono fra doppi apici

    "Hello"
  - Le stringhe sono array di caratteri
    - Il nome della stringa è il puntatore al suo primo carattere
    - Il valore di una stringa è l'indirizzo del suo primo carattere

Le stringhe

È possibile definire array per gestire qualsiasi tipo di dato semplice, int, float, char, ...

In genere una collezione di numeri interi è comoda per tenere uniti tutti i dati, che però hanno un significato proprio.

Quando si parla di caratteri, invece, può essere interessante poter manipolare l'intero insieme di caratteri appartenenti ad un array, in quanto costituiscono nell'insieme un vocabolo o un'intera frase (con i debiti spazi).

Il C consente quindi di interpretare una sequenza di caratteri come una singola unità, per una più efficiente manipolazione e vi è una apposita libreria standard - string.h - con funzioni di utilità di frequente utilizzo.

STRINGHE

Dichiarazione di stringhe

- Si dichiara come un array di caratteri o come una variabile di tipo **char ***

  char color[] = "blue";

  char *colorPtr = "blue";

- Le stringhe rappresentate come array di caratteri **devono terminare con '\0'**

  color ha 5 elementi

Si consideri la seguente dichiarazione di un array di caratteri di 20 elementi:

char Vocabolo[20];

È possibile accedere ad ogni elemento dell'array singolarmente, come si fa per ogni altro tipo di array.

È inoltre possibile manipolare l'intero array come un'unica entità purchè esista un carattere terminatore '\0' in uno degli elementi dell'array, che sia stato inserito dalle funzioni di manipolazione della stringa, oppure direttamente da programma durante l'esecuzione.

È importante ricordarsi di dimensionare opportunamente l'array includendo un elemento anche per contenere il terminatore.

Ad esempio, se un algoritmo prevede che si debba gestire vocaboli "di al più 20 caratteri" è necessario dichiarare un array di 21 elementi.

```
#define DIM 20
char Vocabolo[DIM+1];
```


```
/* Visualizzazione caratteri di una stringa */
#include <stdio.h>

char frase[] = "Analisi, requisiti ";

main()
{
int i=0;
while(frase[i]!='\0') {
  printf("%c\n", frase[i]);
  i++;
}
}
```


INPUT DA TASTIERA DI STRINGHE

- Introdurre una stringa da tastiera
  - Si usa **`scanf`**

    `scanf("%s", word);`
    - Copia l'input in **`word[]`**, che non richiede **`&`** (perchè una stringa è un puntatore)
  - Bisogna ricordarsi di lasciare spazio per **`'\0'`**

ASSEGNARE UNA STRINGA AD UN'ALTRA

Si potrebbe pensare che fosse possibile assegnare una stringa ad un'altra, direttamente, mediante un'espressione del tipo:

```
char a[l0], b[10];
b = a;
```

La parte di codice *non copia ordinatamente i caratteri presenti nell'array a nei caratteri dell'array b.*

Ciò che viene effettivamente ottenuto è che ***b punti allo stesso insieme di caratteri di a*** senza farne una copia.

Quindi, modificando poi i valori di b si modificano quelli di a.

Questo fatto ha effetto anche sulla inzializzazione degli array.

Non è possibile scrivere:

```
a = "prova";
```

perchè a indica l'inizio dell'array (è un puntatore) mentre "prova" è una stringa costante.

Copia e confronto di stringhe

Per copiare il contenuto di una stringa in un'altra è necessaria la funzione (di libreria)

char * strcopy(char[],char[])

che effettivamente effettua la copia elemento ad elemento dell'array a nell'array b fino al carattere terminatore. Ci sono numerose altre funzioni, tra cui citiamo solo l'importate funzione di confronto tra stringhe.

Infatti il confronto a == b darebbe esito positivo solamente se i due array puntassero allo stesso insieme di caratteri, e non se il loro contenuto fosse identico.

La funzione

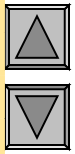
int strcmp(char[],char[])

confronta due stringhe e restituisce 0 se il loro contenuto è identico.

Alcune funzioni di libreria standard Input/Output

- Funzioni in **<stdio.h>**

Prototipo	Descrizione
`int getchar( void );`	Acquisisce un carattere da tastiera.
`char *gets( char *s );`	Inserisce caratteri nell'array `s` fino ad un "a capo" o "fine linea". Viene inserito il carattere di fine stringa.
`int putchar( int c );`	Stampa il carattere memorizzato in `c`.
`int puts( const char *s );`	Stampa la stringa `s` seguita da un "a capo"
`int sprintf( char *s, const char *format, ... );`	**Equivale a `printf`, ma l'output è memorizzato nella stringa `s` invece di essere visualizzato sul monitor**
`int sscanf( char *s, const char *format, ... );`	**Equivale a `scanf`, ma l'input è letto dalla stringa `s` invece che dalla tastiera**



Outline

Stampa di una stringa in ordine inverso

Input

Definizione della ne ricorsiva

```
1  /*USO DELLE FUNZIONI gets e putchar */

3  #include <stdio.h>
4
5  int main()
6  {
7     char sentence[ 80 ];
8     void reverse( const char * const );
9
10    printf( "Enter a line of text:\n" );
11    gets( sentence );
12
13    printf( "\nThe line printed backwards is:\n" );
14    reverse( sentence );
15
16    return 0;
17 }
18
19 void reverse( const char * const sPtr )
20 {
21    if ( sPtr[ 0 ] == '\0' )
22       return;
23    else {
24       reverse( &sPtr[ 1 ] );
25       putchar( sPtr[ 0 ] );
26    }
27 }
```

reverse chiama se stessa con sotto-stringhe di quella iniziale. Quando trova il carattere **'\0'** stampa con **putchar**

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC
```

Copia di una stringa su un'altra (senza funzioni di libreria)

```
/* versione base */
#include <stdio.h>
char frase[] = "Analisi, requisiti ";
main()
{
int i;
char discorso[80];
for(i=0; (discorso[i]=frase[i])!='\0'; i++)  ;
printf(" originale: %s \n copia:     %s \n",
     frase, discorso);
}
```

```
/* versione 1  che usa gli array*/
strcop( char s[], char t[])
{
  int i=0;
  while ((s[i] = t[i]) != '\0') i++;
}
```

```
/*versione 2.1 che usa i puntatori*/
strcop(char *s, char *t)
{
  while ((*s = *t) != '\0') {s++; t++;}
}
```

```
/* versione 2.2 */
strcop(char *s, char *t)
{
  while ((*s++ = *t++) != '\0') ;
}
```

```
/* versione 2.3 */
strcop(char *s, char *t)
{
  while (*s++ = *t++);
}
```


/* Concatenazione di due stringhe (senza funzioni di libreria)*/

```
#include <stdio.h>

char frase[160] = "Analisi, requisiti ";

main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
scanf("%s", dimmi);
for(i=0; (frase[i])!='\0'; i++)  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)  ;
printf("frase: %s \n", frase);
}
```


/* Concatenazione di due stringhe
Introduzione della seconda stringa con getchar */

```
#include <stdio.h>

char frase[160] = "Analisi, requisiti ";

main()
{
char dimmi[80];
int i, j;

printf("Inserisci una parola: ");
for(i=0; (dimmi[i]=getchar())!='\n'; i++)  ;
dimmi[i]='\0';
for(i=0; frase[i]!='\0'; i++)  ;
for(j=0; (frase[i]=dimmi[j])!='\0'; i++,j++)  ;
printf(" frase: %s \n", frase);
}
```


```
/* Confronto fra due stringhe (senza funzioni di libreria)*/
#include <stdio.h>

char prima[160] = "mareggiata";

main()
{
char seconda[80];
int i;

printf("Inserisci una parola: ");
for(i=0; ((seconda[i]=getchar()) != '\n') && (i<80) ;i++) ;
seconda[i]='\0';
for(i=0; (prima[i] == seconda[i]) && (prima[i] != '\0') && (seconda[i] != '\0'); i++);
if(prima[i]==seconda[i])
  printf("Sono uguali\n");
else
  if(prima[i]>seconda[i])
    printf("La prima e' maggiore della seconda\n");
  else
    printf("La seconda e' maggiore della prima\n");
}
```


Funzioni di libreria per la gestione di stringhe

Prototipo	Descrizione
`char *strcpy( char *s1, const char *s2 )`	Copia la stringa `s2` in `s1`. Restituisce `s1`
`char *strncpy( char *s1, const char *s2, size_t n )`	Copia fino a `n` caratteri della stringa `s2` in `s1`. Restituisce `s1`.
`char *strcat( char *s1, const char *s2 )`	Concatena `s2` a `s1`. Il primo carattere di `s2` è sovrascritto su carattere di fine stringa di `s1`. Restituisce `s1`.
`char *strncat( char *s1, const char *s2, size_t n )`	Concatena fino a `n` caratteri della stringa `s2` a `s1`. Il primo carattere di `s2` è sovrascritto su carattere di fine stringa di `s1`. Restituisce `s1`.
`int strcmp( const char *s1, const char *s2 );`	Confronta la stringa s1 con s2 Restituisce un numero negativo (s1 < s2), zero (s1 == s2), o positivo (s1 > s2)
`int strncmp( const char *s1, const char *s2, size_t n );`	Confronta fino a n caratteri di s1 con s2 Restituisce un numero negativo (s1 < s2), zero (s1 == s2), o positivo (s1 > s2)

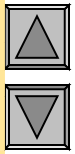
Lunghezza di una stringa con strlen

```
#include <stdio.h>

char str[] = “INFORMATICA";
int strlen(char *);

main()
{
printf("la stringa %s ha lunghezza %d\n", str, strlen(str));
}

int strlen(char *p)
{
  int i = 0;
  while (*p++) i++;
  return i;
}
```

Outline

```c
/* Fig. 8.19: fig08_19.c
   Using strcat and strncat */
#include <stdio.h>
#include <string.h>

int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
   return 0;
}
```

Concatenazione di stringhe

Chiamata delle funzioni di libreria

Stampa dei risultati

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Output

/* Concatenazione di dei primi n caratteri di una stringa su di un'altra con strcat */

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

char frase[160] = "Analisi, requisiti";

main()
{
char dimmi[80];
int i;

for(i=0; ((dimmi[i]=getchar())!='\n') && (i<80); i++)
  ;
dimmi[i] = '\0';
strncat(frase, dimmi, 5);
printf("%s \n", frase);
}
```


```
/* Confronto tra due stringhe con strcmp */
#include <stdio.h>
#include <string.h>

char prima[160] = "mareggiata";

main()
{
char seconda[80];
int i, x;

printf("Inserisci una parola: ");
for(i=0; ((seconda[i]=getchar())!='\n') && (i<80); i++) ;
seconda[i] = '\0';

if( (x = (strcmp(prima, seconda))) == 0)
  printf("Sono uguali\n");
else
  if(x>0)
    printf("la prima e' maggiore della seconda\n");
  else
    printf("la seconda e' maggiore della prima\n");
}
```


/* Confronto dei primi n caratteri di due stringhe con strncmp */

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

char prima[160] = "Analisi, requisiti";

main()
{
char seconda[80];
int i, x;

for(i=0; ((seconda[i]=getchar())!='\n') && (i<80); i++)
  ;
seconda[i]='\0';
if((x=(strncmp(prima, seconda, 5)))==0)
  printf("Sono uguali\n");
else
  if(x>0)
    printf("La prima e' maggiore della seconda\n");
  else
    printf("La seconda e' maggiore della prima\n");
}
```


Input/Output formattato

Output formattato con `printf`
Stampa di interi
Stampa di numeri floating point
Stampa di caratteri, stringhe, puntatori
Definizione della ampiezza del campo e della precisione di stampa
Input formattato con `scanf`

Stampa di interi

Specificatore di formato	Descrizione
d	Intero decimale con segno.
i	Intero decimale con segno. (*Nota:* **i** e **d** sono diversi se usati con **scanf**.)
o	Intero ottale senza segno.
u	Intero decimale senza segno.
x or X	Intero esadecimale senza segno
h or **l** (letter **l**)	Si pone prima di uno specificatore di intero a indicare un intero **short** oppure **long**

Stampa di numeri in virgola mobile

Numeri in virgola mobile

- Sono i numeri con il punto decimale (**33.5**)
- Notazione esponenziale (notazione scientifica)
  - **150.3** corrisp. a **1.503 x 10²** in notazione scientifica
  - **150.3** corrisp. a **1.503E+02** in notazione esponenziale (**E** indica esponente)
  - Si usa **e** o **E**
- **f** – Stampa il numero con almeno una cifra dopo il punto decimanle
- **g** (o **G**) – stampa in formato **f** o **e**(**E**) eliminando gli zeri superflui (**1.2300** becomes **1.23**)
  - Si usa la notazione esponenziale se l'esponente è minore di **-4**, o maggiore o uguale alla precisione (6 cifre di default)

Stampa di caratteri, stringhe e puntatori

c

- Stampa una variabile di tipo **char**
- Non si può usare per stampare il primo elemento di una stringa

s

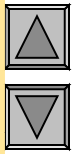
- Ha come argomento un puntatore a **char**
- Stampa i caratteri finchè non incontra il carattere **NULL** (**'\0'**)
- Non può stampare un **char**

NOTA

- Apici singoli per i char (**'z'**)
- Apici doppi per le stringhe **"z"** (che in effetti contiene due caratteri, **'z'** e **'\0'**)

p

- Stampa il valore del puntatore (indirizzo di memoria)



Outline

```
1
2  /* STAMPA DI CARATTERI E STRINGHE */
3  #include <stdio.h>
4
5  int main()
6  {
7     char character = 'A';
8     char string[] = "This is a string";
9     const char *stringPtr = "This is also a string";
10
11    printf( "%c\n", character );
12    printf( "%s\n", "This is a string" );
13    printf( "%s\n", string );
14    printf( "%s\n", stringPtr );
15
16    return 0;
17 }
```

Inizializzazione delle variabili

Stampa

```
A
This is a string
This is a string
This is also a string
```

Output

FORMATTAZIONE DELL'OUTPUT CON PRINTF

printf

- **Formattazione precisa dell'output**

  Specificatori di conversione: ampiezza, precisione,ecc.

- Effettua arrotondamenti, allineamenti, giustificazione a destra/sinistra, inserimento di caratteri, formato esponenziale, esadecimale, ecc.

Formato

- **printf(** stringa-controllo-formato**,** argomenti**);**
- Stringa-controllo-formato: descrive il formato di uscita
- Argomenti: corrispondono ad ogni specificazione nella stringa di controllo del formato
  - Ogni specificazione inizia con il simbolo %, e termina con lo specificatore di formato

Ampiezza del campo e precisione

Ampiezza del campo

- Dimensione (intera) del campo in cui stampare l'uscita. Si pone fra % e lo specificatore di conversione:
- **`%4d`** – indica un campo di ampiezza 4
- Se l'ampiezza indicata è maggiore del dato, questo viene "giustificato" a destra, se è troppo piccola, viene incrementata

Precisione

- Per gli interi (default 1) – è il minimo numero di cifre da stampare
- Per i floating point – è il numero di cifre che devono apparire dopo il punto decimale (per **`e`** ed **`f; per g`** è il massimo numero di cifre significative
- Per le stringhe – è il massimo numero di caratteri della stringa che devono essere scritti

Ampiezza del campo e precisione (cont.)

Formato

- Precisione: si usa un punto (.) seguito dalla precisione, dopo %

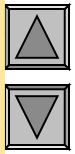
  `%.3f`

- Si può combinare con l'ampiezza del campo

  `%5.3f`

- Si possono usare espressioni intere per calcolare ampiezza e precisione
  - Si può usare * per indicare il default
  - Ampiezza negativa – giustifica a sinistra
  - Ampiezza positiva – giustifica a destra
  - La precisione deve essere positiva

```
printf( "%*.*f", 7, 2, 98.736 );
```

Outline

```
1
/* USO DELLA PRECISIONE NEL PRINTF PER INTERI, FLOAT E STRINGHE */
3
4  #include <stdio.h>
5
6  int main()
7  {
8     int i = 873;
9     double f = 123.94536;
10    char s[] = "Happy Birthday";
11
12    printf( "Using precision for integers\n" );
13    printf( "\t%.4d\n\t%.9d\n\n", i, i );
14    printf( "Using precision for floating-point numbers\n" );
15    printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
16    printf( "Using precision for strings\n" );
17    printf( "\t%.11s\n", s );
18
19    return 0;
20 }
```

Inizializzazione delle variabili

Stampa

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+02
        124

Using precision for strings
        Happy Birth
```

Output

Formattazione dell'input con Scanf

scanf

- Formattazione dell'input
- Caratteristiche
  - Consente l'input di qualsiasi tipo di dati
  - Immette o salta specifici caratteri

Formato

scanf(stringa-controllo-formato, argomenti);

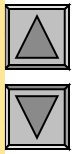
- Stringa-controllo-formato : descrive il formato degli ingressi
- Argomenti : puntatori a variabili in cui vengono salvati i dati letti
- Può includere l'ampiezza del campo per leggere un numero specifico di caratteri

SCANF

SPECIFICATORE DI CONVERSIONE	DESCRIZIONE
Interi	
d	Intero decimale con eventuale segno. L'argomento corrispondente è un puntatore a intero.
i	Intero decimale, ottale o esadecimale con eventuale segno. L'argomento corrispondente è un puntatore a intero.
o	Legge un intero ottale. L'argomento corrispondente è un puntatore a intero senza segno.
u	Legge un intero decimale senza segno. L'argomento corrispondente è un puntatore a intero senza segno.
x or **X**	Legge un intero esadecimale. L'argomento corrispondente è un puntatore a intero senza segno.
h or **l**	Posto prima di una delle conversioni di interi, specifica se l'input è un intero **short** o **long**.
Floating-point s	
e, **E**, **f**, **g** or **G**	Numero floating point (reale). L'argomento corrispondente è un puntatore a floating point.
l or **L**	Posto prima di una delle conversioni di floating-point conversion specifica se l'input è **double** o **long double**.
Caratteri e stringhe	
c	Legge un carattere. L'argomento corrispondente è un puntatore a **char**, non viene aggiunto (**'\0'**).
s	Legge una stringa. L'argomento corrispondente è un puntatore a array di tipo **char** di dimensioni sufficienti per contenere la stringa e il carattere (**'\0'**) che viene aggiunto automaticamente.
Varia	
p	Legge un indirizzo della stessa forma di quello che si ottiene con **%p** in un **printf**.
n	Memorizza il numero di caratteri immessi fino a questo punto con **scanf**. L'argomento corrispondente è un puntatore a intero

```
1
2  /* LETTURA DI CARATTERI E STRINGHE */
3  #include <stdio.h>
4
5  int main()
6  {
7     char x, v[ 9 ];
8
9     printf( "Enter a string: " );
10    scanf( "%c%s", &x, v );
11
12    printf( "The input was:\n" );
13    printf( "the character \"%c\" ", x );
14    printf( "and the string \"%s\"\n", v );
15
16    return 0;
17 }
```

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```



Outline

Inizializzazione delle variabili

Input con scanf

Stampa con printf

Output