

# Laboratorio di Informatica

## Linguaggio C

(Stringhe e Caratteri)

**docente: Cataldo Musto**

[cataldo.musto@uniba.it](mailto:cataldo.musto@uniba.it)

# Stringhe

# Stringhe

- «Stringa» è il nome che identifica un insieme di caratteri chiusi tra virgolette
  - `char string1[] = "first";`

# Stringhe

- «Stringa» è il nome che identifica un insieme di caratteri chiusi tra virgolette
  - `char string1[] = "first";`
  - Equivalente a un array di caratteri
    - `char string1[] = {'f','i','r','s','t','\0'};`
    - Ha sei elementi (Stringhe di N caratteri hanno N+1 elementi), memorizzati **in locazioni contigue**
    - Il carattere **'\0'** termina le stringhe ed è detto terminatore.
      - Nell'inizializzazione delle stringhe è inserito in automatico, altrimenti deve essere inserito esplicitamente

# Stringhe

- Trattandosi di array, è possibile accedere ai caratteri individuali

`string1[3]` è il carattere 's'

- Nel caso degli array di stringhe, & non è richiesto nella scanf

- `scanf( "%s", string2 );`
- Perché?

# Stringhe

- **Trattandosi di array, è possibile accedere ai caratteri individuali**  
`string1[3]` è il carattere 's'
- **Nel caso degli array di stringhe, & non è richiesto nella scanf**
  - `scanf( "%s", string2 );`
  - Perché?
  - Il nome dell'array è un puntatore all'indirizzo del primo elemento

# Stringhe - Esempio

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

# Stringhe - Esempio

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

**Dichiaro una stringa di  
dimensione 10**



# Stringhe - Esempio

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

**Leggo l'input**

**Importante: %s non %c**

# Stringhe - Esempio

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

## Domanda

**Cosa stampa se  
inserisco una stringa la  
cui dimensione è  
maggiore a 10?**

# Stringhe - Esempio

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: %s\n", string);
8      puts(string);
9  }
```

## Domanda

**Cosa stampa se  
inserisco una stringa la  
cui dimensione è  
maggiore a 10?**

```
gcc version 4.6.3
```

```
>
```

```
Inserisci Stringa: stringamoltolunga
Stringa letta: stringamoltolunga
```

```
>
```

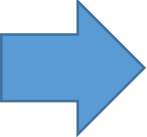
# Stringhe - Esempio

- **Il Linguaggio C non effettua nessun controllo sulla lunghezza delle stringhe**
  - Il realtà non effettua nessun controllo sugli array!
- Stringhe anche più lunghe di LENGTH saranno correttamente memorizzate
- **Non abbiamo però la garanzia che quelle locazioni di memoria non vengano occupate in futuro**
  - Possibili comportamenti inattesi!
- **Accorgimento:** limitare il numero di caratteri letti in input, sempre!

# Stringhe

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%9s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

# Stringhe

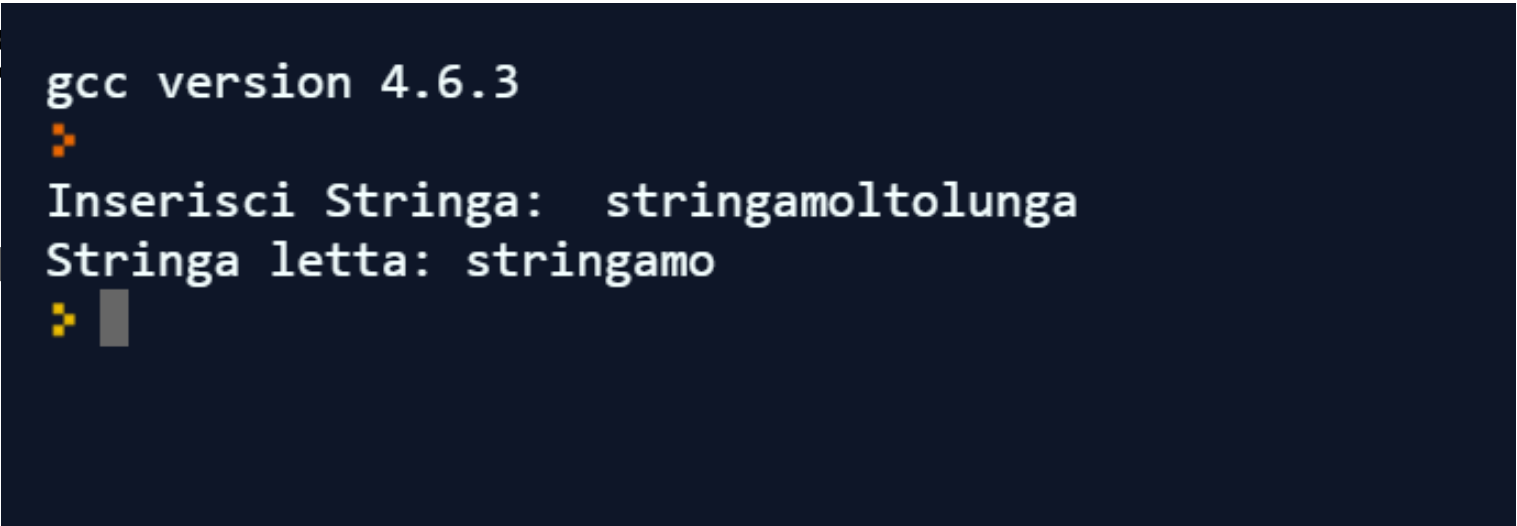
```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6   scanf("%9s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

**Limite la lettura ai primi  
N-1 caratteri (l'ultimo è il  
terminatore)**

# Stringhe

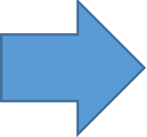
```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

**Limite la lettura ai primi  
N-1 caratteri (l'ultimo è il  
terminatore)**



```
gcc version 4.6.3
Inserisci Stringa: stringamoltolunga
Stringa letta: stringamo
```

# Stringhe

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6   scanf("%9s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```

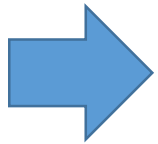
**Importante: la scanf( )  
si ferma se trova un  
terminatore o uno spazio**



# Stringhe

**Importante: la `scanf( )`  
si ferma se trova un  
terminatore o uno spazio**

```
1  #include "stdio.h"
2  #define LENGTH 10
3  int main(void) {
4      char string[LENGTH];
5      printf("Inserisci Stringa: ");
6      scanf("%9s", string);
7      printf("Stringa letta: ");
8      puts(string);
9  }
```



```
gcc version 4.6.3
```

```
Inserisci Stringa: stringa molto lunga
Stringa letta: stringa
```

# Stringhe – Funzioni di Elaborazione

- **C include numerose funzioni per l'elaborazione di stringhe e caratteri**
  - **Disponibili nella libreria `<ctype.h>`**
  - **Funzioni per l'elaborazione dei caratteri**
    - Controllo che un dato carattere sia **un numero o una lettera**
    - Controllo che un dato carattere sia **maiuscolo o minuscolo**
    - Controllo che un dato carattere sia un *simbolo di punteggiatura, uno spazio, etc.*

# Stringhe – Funzioni di Elaborazione

- **C include numerose funzioni per l'elaborazione di stringhe e caratteri**
  - **Disponibili nella libreria `<ctype.h>`**
  - **Funzioni per l'elaborazione dei caratteri**
    - Controllo che un dato carattere sia **un numero o una lettera**
    - Controllo che un dato carattere sia **maiuscolo o minuscolo**
    - Controllo che un dato carattere sia un *simbolo di punteggiatura, uno spazio, etc.*
  - **Funzioni per l'elaborazione delle stringhe**
    - **Confronto** tra due stringhe (verifica se sono uguali o meno)
    - Conversione stringhe → valori numerici (e viceversa)
    - Copia (parziale o totale) di una stringa in un'altra stringa o concatenazione tra stringhe
    - Suddivisione di una frase in singoli termini (*tokenizzazione*)

# Funzioni per l'elaborazione dei caratteri

Prototipo	Descrizione
<code>int isdigit( int c );</code>	Returns true if c is a digit and false otherwise.
<code>int isalpha( int c );</code>	Returns true if c is a letter and false otherwise.
<code>int isalnum( int c );</code>	Returns true if c is a digit or a letter and false otherwise.
<code>int isxdigit( int c );</code>	Returns true if c is a hexadecimal digit character and false otherwise.
<code>int islower( int c );</code>	Returns true if c is a lowercase letter and false otherwise.
<code>int isupper( int c );</code>	Returns true if c is an uppercase letter; false otherwise.
<code>int tolower( int c );</code>	If c is an uppercase letter, tolower returns c as a lowercase letter. Otherwise, tolower returns the argument unchanged.

In C i valori booleani sono codificati come interi:  
**True = 1 False = 0**

# Funzioni per l'elaborazione dei caratteri

Prototipo	Descrizione
<code>int toupper( int c );</code>	If c is a lowercase letter, toupper returns c as an uppercase letter. Otherwise, toupper returns the argument unchanged.
<code>int isspace( int c );</code>	Returns true if c is a white-space character—newline ('\n'), space (' '), form feed ('\f'), carriage return ('\r'), horizontal tab ('\t'), or vertical tab ('\v')—and false otherwise
<code>int iscntrl( int c );</code>	Returns true if c is a control character and false otherwise.
<code>int ispunct( int c );</code>	Returns true if c is a printing character other than a space, a digit, or a letter and false otherwise.
<code>int isprint( int c );</code>	Returns true value if c is a printing character including space (' ') and false otherwise.
<code>int isgraph( int c );</code>	Returns true if c is a printing character other than space (' ') and false otherwise.

In C i valori booleani sono codificati come interi:  
**True = 1 False = 0**

# Funzioni per l'elaborazione dei caratteri

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char a = 'a'; // IMPORTANTE
6      char n = '9'; // 'virgolette singole' per i char
7
8      // Funzione isdigit()
9      if(isdigit(a)) printf("\n%c is a digit", a);
10     else printf("\n%c is not a digit", a);
11
12     if(isdigit(n)) printf("\n%c is a digit", n);
13     else printf("\n%c is not a digit", n);
14 }
```

**isdigit(char)** restituisce a 1 se il carattere è un **numero**, altrimenti sarà uguale a 0

# Funzioni per l'elaborazione dei caratteri

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char a = 'a'; // IMPORTANTE
6      char n = '9'; // 'virgolette singole' per i
7
8      // Funzione isdigit()
9      if(isdigit(a)) printf("\n%c is a digit", a);
10     else printf("\n%c is not a digit", a);
11
12     if(isdigit(n)) printf("\n%c is a digit", n);
13     else printf("\n%c is not a digit", n);
14 }
```

**isdigit(char)** restituisce a 1 se il carattere è un **numero**, altrimenti sarà uguale a 0

gcc version 4.6.3

```
a is not a digit
9 is a digit
```

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5
6 #include <stdio.h>
7 #include <ctype.h>
8
9 int main() {
10     char a = 'a'; // IMPORTANTE
11     char up = 'A'; // virgolette 'singole'
12     char n = '1'; // per le variabili
13     char s = "#"; // char
14
15     // funzione 'isalpha()'
16     if(isalpha(a)) printf("\n%c is a digit", a);
17     else printf("\n%c is not a digit", a);
18
19     if(isalpha(n)) printf("\n%c is a digit", n);
20     else printf("\n%c is not a digit", n);
21 }
```

**isalpha(char)** restituisce a 1 se il carattere è alfabetico, altrimenti sarà uguale a 0



# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5
6 #include <stdio.h>
7 #include <ctype.h>
8
9 int main() {
10     char a = 'a'; // IMPORTANTE
11     char up = 'A'; // virgolette 'singole'
12     char n = '1'; // per le variabili
13     char s = "#"; // char
14
15     // funzione 'isalpha()'
16     if(isalpha(a)) printf("\n%c is alphabetic", a);
17     else printf("\n%c is not alphabetic", a);
18
19     if(isalpha(n)) printf("\n%c is alphabetic", a);
20     else printf("\n%c is not alphabetic", a);
21 }
```

**isalpha(char)** restituisce a 1 se il carattere è alfabetico, altrimenti sarà uguale a 0

gcc version 4.6.3

```
a is alphabetic
1 is not alphabetic
```

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5 #include <stdio.h>
6 #include <ctype.h>
7
8 int main() {
9     char a = 'a'; // IMPORTANTE
10    char up = 'A'; // virgolette 'singole'
11    char n = '1'; // per le variabili
12    char s = "#"; // char
13
14    // funzione 'isalnum()'
15    if(isalnum(a)) printf("\n%c is alpha-numeric", a);
16    else printf("\n%c is not alpha-numeric", a);
17
18    if(isalnum(n)) printf("\n%c is alpha-numeric", n);
19    else printf("\n%c is not alpha-numeric", n);
20
21    if(isalnum(s)) printf("\n%c is alpha-numeric", s);
22    else printf("\n%c is not alpha-numeric", s);
23 }
```

**isalnum(char)** restituisce a 1 se il carattere è alfanumerico, altrimenti sarà uguale a 0

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5 #include <stdio.h>
6 #include <ctype.h>
7
8 int main() {
9     char a = 'a'; // IMPORTANTE
10    char up = 'A'; // virgolette 'singole'
11    char n = '1'; // per le variabili
12    char s = "#"; // char
13
14    // funzione 'isalnum()'
15    if(isalnum(a)) printf("\n%c is alpha-numeric", a);
16    else printf("\n%c is not alpha-numeric", a);
17
18    if(isalnum(n)) printf("\n%c is alpha-numeric", n);
19    else printf("\n%c is not alpha-numeric", n);
20
21    if(isalnum(s)) printf("\n%c is alpha-numeric", s);
22    else printf("\n%c is not alpha-numeric", s);
23 }
```

**isalnum(char)** restituisce a 1 se il carattere è alfanumerico, altrimenti sarà uguale a 0.

```
gcc version 4.6.3
```

```
✧
```

```
a is alpha-numeric
```

```
1 is alpha-numeric
```

```
# is not alpha-numeric ✧
```

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5
6 #include <stdio.h>
7 #include <ctype.h>
8
9 int main() {
10     char a = 'a'; // IMPORTANTE
11     char up = 'A'; // virgolette 'singole'
12     char n = '1'; // per le variabili
13     char s = '#'; // char
14
15
16     if(islower(a)) printf("\n%c is lower-case", a);
17     else printf("\n%c is upper-case", a);
18
19     if(islower(up)) printf("\n%c is lower-case", up);
20     else printf("\n%c is upper-case", up);
21 }
```

**isupper(char)** restituisce a 1 se il carattere è **maiuscolo**, altrimenti sarà uguale a 0.

**islower(char)** restituisce a 1 se il carattere è **minuscolo**, altrimenti sarà uguale a 0.

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5
6 #include <stdio.h>
7 #include <ctype.h>
8
9 int main() {
10     char a = 'a'; // IMPORTANTE
11     char up = 'A'; // virgolette 'singole'
12     char n = '1'; // per le variabili
13     char s = '#'; // char
14
15
16     if(islower(a)) printf("\n%c is lower-case", a);
17     else printf("\n%c is upper-case", a);
18
19     if(islower(up)) printf("\n%c is lower-case", up);
20     else printf("\n%c is upper-case", up);
21 }
```

**isupper(char)** restituisce a 1 se il carattere è **maiuscolo**, altrimenti sarà uguale a 0.

**islower(char)** restituisce a 1 se il carattere è **minuscolo**, altrimenti sarà uguale a 0.

```
gcc version 4.6.3
```

```
✶
```

```
a is lower-case
```

```
A is not lower-case ✶
```

# Funzioni per l'elaborazione dei caratteri

```
1 // LABORATORIO DI INFORMATICA
2 // a.a. 2017/2018
3 // Corso di Laurea in Informatica T.P.S.
4 // docente: Cataldo Musto
5
6 #include <stdio.h>
7 #include <ctype.h>
8
9 int main() {
10     char a = 'a'; // IMPORTANTE
11     char up = 'A'; // virgolette 'singole'
12     char n = '1'; // per le variabili
13     char s = '#'; // char
14
15     // funzione 'toupper()'
16     printf("\n%c to upper case: %c", a, toupper(a));
17 }
```

**toupper(char)** converte un carattere in maiuscolo.

gcc version 4.6.3



a to upper case: A

# Recap

Prototipo	Descrizione
<code>int isdigit( int c );</code>	Returns true if c is a digit and false otherwise.
<code>int isalpha( int c );</code>	Returns true if c is a letter and false otherwise.
<code>int isalnum( int c );</code>	Returns true if c is a digit or a letter and false otherwise.
<code>int isxdigit( int c );</code>	Returns true if c is a hexadecimal digit character and false otherwise.
<code>int islower( int c );</code>	Returns true if c is a lowercase letter and false otherwise.
<code>int isupper( int c );</code>	Returns true if c is an uppercase letter; false otherwise.
<code>int tolower( int c );</code>	If c is an uppercase letter, tolower returns c as a lowercase letter. Otherwise, tolower returns the argument unchanged.

In C i valori booleani sono codificati come interi:  
**True = 1 False = 0**

# Recap (cont.)

Prototipo	Descrizione
<code>int toupper( int c );</code>	If <code>c</code> is a lowercase letter, <code>toupper</code> returns <code>c</code> as an uppercase letter. Otherwise, <code>toupper</code> returns the argument unchanged.
<code>int isspace( int c );</code>	Returns true if <code>c</code> is a white-space character—newline ( <code>'\n'</code> ), space ( <code>' '</code> ), form feed ( <code>'\f'</code> ), carriage return ( <code>'\r'</code> ), horizontal tab ( <code>'\t'</code> ), or vertical tab ( <code>'\v'</code> )—and false otherwise
<code>int iscntrl( int c );</code>	Returns true if <code>c</code> is a control character and false otherwise.
<code>int ispunct( int c );</code>	Returns true if <code>c</code> is a printing character other than a space, a digit, or a letter and false otherwise.
<code>int isprint( int c );</code>	Returns true value if <code>c</code> is a printing character including space ( <code>' '</code> ) and false otherwise.
<code>int isgraph( int c );</code>	Returns true if <code>c</code> is a printing character other than space ( <code>' '</code> ) and false otherwise.

In C i valori booleani sono codificati come interi:  
**True = 1 False = 0**



# Problema 3.1

Scrivere un programma che acquisisca in input la password inserita da un utente.  
Verificare che la password contenga almeno una lettera maiuscola ed almeno un numero  
Stampare in input un messaggio di conferma se la password è corretta. In alternativa,  
stampare un messaggio di errore.

**Input?**

**Output?**

**Quale tipologia di istruzioni ci serve?**

# Problema 3.1

Scrivere un programma che acquisisca in input la password inserita da un utente. Verificare che la password contenga almeno una lettera maiuscola ed almeno un numero. Stampare in input un messaggio di conferma se la password è corretta. In alternativa, stampare un messaggio di errore.

Input?

Output?

Quale tipologia di istruzioni ci serve?

## Esempio

**Input:** cat4ldo

**Output:** «Password non corretta, inserire almeno una lettera maiuscola»

**Input:** Password

**Output:** «Password non corretta, inserire almeno un numero»

**Input:** Pa55word

**Output:** «Password impostata correttamente.»

# Problema 3.1

Scrivere un programma che acquisisca in input la password inserita da un utente. Verificare che la password contenga almeno una lettera maiuscola ed almeno un numero. Stampare in input un messaggio di conferma se la password è corretta. In alternativa, stampare un messaggio di errore.

## Input?

Stringa, inserita dall'utente

## Output?

Messaggio di conferma o messaggio di errore

## Quale tipologia di istruzioni ci serve?

- Istruzioni per la manipolazione dei caratteri
- (che altro?)

## Esempio

**Input:** cat4ldo

**Output:** «Password non corretta, inserire almeno una lettera maiuscola»

**Input:** Password

**Output:** «Password non corretta, inserire almeno un numero»

**Input:** Pa55word

**Output:** «Password impostata correttamente.»

# Problema 3.1

Scrivere un programma che acquisisca in input la password inserita da un utente. Verificare che la password contenga almeno una lettera maiuscola ed almeno un numero. Stampare in input un messaggio di conferma se la password è corretta. In alternativa, stampare un messaggio di errore.

## Input?

Stringa, inserita dall'utente.

## Output?

Messaggio di conferma o messaggio di errore.

## Quale tipologia di istruzioni ci serve?

- Istruzioni per la manipolazione dei caratteri
- Istruzioni di iterazione
  - **Suggerimento: una stringa è un array di caratteri**

## Codificare la soluzione su Repl.it

### Esempio

**Input:** cat4ldo

**Output:** «Password non corretta, inserire almeno una lettera maiuscola»

**Input:** Password

**Output:** «Password non corretta, inserire almeno un numero»


**Input:** Pa55word

**Output:** «Password impostata correttamente.»

# Soluzione 3.1 (parte 1)

```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                      funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                      // è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```


# Soluzione 3.1 (parte 1)



```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                      funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                      // è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

**Rigo 7** – dichiaro una variabile per uscire dal ciclo

# Soluzione 3.1 (parte 1)



```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                      funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                      // è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

**Rigo 7** – dichiaro una variabile per uscire dal ciclo

**Rigo 14** – il programma viene eseguito finché la variabile **sentinella** è **uguale a zero**

# Soluzione 3.1 (parte 1)

```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                      funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                      // è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

**Rigo 7** – dichiaro una variabile per uscire dal ciclo

**Rigo 14** – il programma viene eseguito finché la variabile **sentinella** è **uguale a zero**


**Rigo 19-20** – perché inizializzo nuovamente queste variabili?



# Soluzione 3.1 (parte 2)

```
21 // ELABORAZIONE INPUT
22 for ( unsigned i=0; i<PASSWORD_LENGTH ; i++) {
23     digit += isdigit(password[i]);
24     upper += isupper(password[i]);
25
26     if ((digit > 0) && (upper > 0))
27         i = PASSWORD_LENGTH; // a che serve?
28 }
29
30 // VISUALIZZAZIONE OUTPUT
31 if ( (digit>0) && (upper>0) ) {
32     puts("Password impostata correttamente");
33     correct++;
34 }
35 else if ( digit == 0)
36     puts("Inserire almeno un carattere numerico");
37     if (upper == 0)
38         puts("Inserire almeno un carattere maiuscolo");
39 }}
```

# Soluzione 3.1 (parte 2)




```
21 // ELABORAZIONE INPUT
22 for ( unsigned i=0; i<PASSWORD_LENGTH ; i++) {
23     digit += isdigit(password[i]);
24     upper += isupper(password[i]);
25
26     if ((digit > 0) && (upper > 0))
27         i = PASSWORD_LENGTH; // a che serve?
28 }
29
30 // VISUALIZZAZIONE OUTPUT
31 if ( (digit>0) && (upper>0) ) {
32     puts("Password impostata correttamente");
33     correct++;
34 }
35 else if ( digit == 0)
36     puts("Inserire almeno un carattere numerico");
37     if (upper == 0)
38         puts("Inserire almeno un carattere maiuscolo");
39 }}
```

## Riga 22-28

scorro tutti i caratteri della password, conteggiando il numero di caratteri maiuscoli e di cifre numeriche

# Soluzione 3.1 (parte 2)



```
21 // ELABORAZIONE INPUT
22 for ( unsigned i=0; i<PASSWORD_LENGTH ; i++) {
23     digit += isdigit(password[i]);
24     upper += isupper(password[i]);
25
26     if ((digit > 0) && (upper > 0))
27         i = PASSWORD_LENGTH; // a che serve?
28 }
29
30 // VISUALIZZAZIONE OUTPUT
31 if ( (digit>0) && (upper>0) ) {
32     puts("Password impostata correttamente");
33     correct++;
34 }
35 else if ( digit == 0)
36     puts("Inserire almeno un carattere numerico");
37     if (upper == 0)
38         puts("Inserire almeno un carattere maiuscolo");
39 }}
```


## Riga 22-28

scorro tutti i caratteri della password, conteggiando il numero di caratteri maiuscoli e di cifre numeriche

## Riga 26-27

Non ho bisogno di eseguire tutti i cicli! E' sufficiente farlo finché si trova un carattere e una maiuscola. **In questo modo si riducono gli accessi in memoria e il programma è più efficiente.**

# Soluzione 3.1 (parte 2)



```
21 // ELABORAZIONE INPUT
22 for ( unsigned i=0; i<PASSWORD_LENGTH ; i++) {
23     digit += isdigit(password[i]);
24     upper += isupper(password[i]);
25
26     if ((digit > 0) && (upper > 0))
27         i = PASSWORD_LENGTH; // a che serve?
28 }
29
30 // VISUALIZZAZIONE OUTPUT
31 if ( (digit>0) && (upper>0) ) {
32     puts("Password impostata correttamente");
33     correct++;
34 }
35 else if ( digit == 0)
36     puts("Inserire almeno un carattere numerico");
37     if (upper == 0)
38         puts("Inserire almeno un carattere maiuscolo");
39 }}
```

## Riga 30-38

Stampa dell'output.

Stampa un messaggio di avvenuta impostazione se l'utente ha inserito almeno un numero e almeno una maiuscola, altrimenti stampa un messaggio di errore.

# Soluzione 3.1 (parte 2)

```
21 // ELABORAZIONE INPUT
22 for ( unsigned i=0; i<PASSWORD_LENGTH ; i++) {
23     digit += isdigit(password[i]);
24     upper += isupper(password[i]);
25
26     if ((digit > 0) && (upper > 0))
27         i = PASSWORD_LENGTH; // a che serve?
28 }
29
30 // VISUALIZZAZIONE OUTPUT
31 if ( (digit>0) && (upper>0) ) {
32     puts("Password impostata correttamente");
33     correct++;
34 }
35 else if ( digit == 0)
36     puts("Inserire almeno un carattere numerico");
37     if (upper == 0)
38         puts("Inserire almeno un carattere maiuscolo");
39 }}
```

## Riga 30-38

Stampa dell'output.

Stampa un messaggio di avvenuta impostazione se l'utente ha inserito almeno un numero e almeno una maiuscola, altrimenti stampa un messaggio di errore.

**Torniamo alla domanda precedente.**

# Soluzione 3.1 (parte 1)

```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                        funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                        è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

**Rigo 7** – dichiaro una variabile per uscire dal ciclo

**Rigo 14** – il programma viene eseguito finché la variabile **sentinella è uguale a zero**

**Rigo 19-20** – perché inizializzo nuovamente queste variabili?

# Soluzione 3.1 (parte 1)

```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                      funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                      // è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

**Rigo 19-20** – perché inizializzo nuovamente queste variabili?

Commentare le righe 19 e 20 ed eseguire il programma inserendo queste password:

- password
- passw0rd
- Password

La terza password viene accettata, **nonostante non sia corretta**

# Soluzione 3.1 (parte 1)

Se non inizializzassimo le variabili ad ogni ciclo, rimarrebbero memorizzati i valori dei cicli precedenti, e password non corrette potrebbero essere accettate

```
1  #include <stdio.h>
2  #include <ctype.h> // includo la libreria per gestire le
3                        funzioni sui caratteri
4  #define PASSWORD_LENGTH 10 // lunghezza massima della password
5
6  int main() {
7      int correct = 0; // variabile di controllo, diventa = 1 quando
8                        è stata inserita una password corretta
9      char password[PASSWORD_LENGTH]; // vettore contenente la password
10
11     int upper = 0; // variabili di controllo del numero di maiuscole
12     int digit = 0; // e del numero di cifre numeriche
13
14     while ( correct == 0 ) {
15         //INSERIMENTO INPUT
16         printf("Inserisci la tua password (max. 10 caratteri): ");
17         scanf("%9s", &password); // leggo esattamente nove caratteri
18
19         upper = 0; // NOTA: perchè inizializzo queste variabili a zero?
20         digit = 0;
```

*Rigo 19-20 – perché inizializzo nuovamente queste variabili?*

Commentare le righe 19 e 20 ed eseguire il programma inserendo queste password:

- password
- passw0rd
- Password

La terza password viene accettata, **nonostante non sia corretta**



# Stringhe – Funzioni di Elaborazione

- **Confrontare le stringhe**

- Il computer confronta i codici numerici ASCII dei caratteri delle stringhe

- ```
int strcmp( const char *s1, const char *s2 );
```

- Confronta la stringa s1 con s2

- Restituisce zero se sono uguali, un numero negativo se  $s1 < s2$ , o un numero positivo se  $s1 > s2$  (es. Roma > Bari, Albero < Bari)

# Stringhe – Funzioni di Elaborazione

- **Confrontare le stringhe**

- Il computer confronta i codici numerici ASCII dei caratteri delle stringhe

```
int strcmp( const char *s1, const char *s2 );
```

- Confronta la stringa s1 con s2

- Restituisce zero se sono uguali, un numero negativo se  $s1 < s2$ , o un numero positivo se  $s1 > s2$  (es. Roma > Bari, Albero < Bari)

```
int strncmp( const char *s1, const char *s2, size_t n );
```

- Confronta **N caratteri** della stringa s1 con s2

- Restituisce gli stessi valori come sopra
- Restituisce zero **se i primi N caratteri sono uguali.**

# Stringhe – Confronto tra Stringhe

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char string1[] = "test-Ok";      // dichiaro
6      char string2[] = "test-Ok";      // tre varibili
7      char string3[] = "test-Non-Ok"; // di tipo stringa
8
9      if(!strcmp(string1,string2)) // string1 == string2
10         printf("Stringhe %s e %s uguali\n", string1, string2);
11     else printf("Stringhe %s e %s non uguali\n", string1, string2);
12
13     if(!strcmp(string1,string3)) // string1 != string3
14         printf("Stringhe %s e %s uguali\n", string1, string3);
15     else printf("Stringhe %s e %s non uguali\n", string1, string3);
16
17     int n = 5; // caratteri da confrontare
18     if(!strncmp(string1,string3,n)) // string1 == string3
19         printf("Stringhe %s e %s uguali nei primi %d caratteri\n", string1, string3, n);
20     else printf("Stringhe %s e %s NON uguali nei primi %d caratteri\n", string1, string3, n);
21 }
```

# Stringhe – Confronto tra Stringhe

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char string1[] = "test-Ok";      // dichiaro
6      char string2[] = "test-Ok";      // tre varibili
7      char string3[] = "test-Non-Ok"; // di tipo stringa
8
9      if(!strcmp(string1,string2)) // string1 == string2
10         printf("Stringhe %s e %s uguali\n", string1, string2);
11     else printf("Stringhe %s e %s non uguali\n", string1, string2);
12
13     if(!strcmp(string1,string3)) // string1 != string3
14         printf("Stringhe %s e %s uguali\n", string1, string3);
15     else printf("Stringhe %s e %s non uguali\n", string1, string3);
16
17     int n = 5; // caratteri da confrontare
18     if(!strncmp(string1,string3,n)) // string1 == string3
19         printf("Stringhe %s e %s uguali nei primi %d caratteri\n", string1, string3, n);
20     else printf("Stringhe %s e %s NON uguali nei primi %d caratteri\n", string1, string3, n);
21 }
```

Perché si usa il «!» (not) ?

# Stringhe – Confronto tra Stringhe

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char string1[] = "test-Ok";      // dichiaro
6      char string2[] = "test-Ok";      // tre varibili
7      char string3[] = "test-Non-Ok"; // di tipo stringa
8
9      if(!strcmp(string1,string2)) // string1 == string2
10         printf("Stringhe %s e %s uguali\n", string1, string2);
11     else printf("Stringhe %s e %s non uguali\n", string1, string2);
12
13     if(!strcmp(string1,string3)) // string1 != string3
14         printf("Stringhe %s e %s uguali\n", string1, string3);
15     else printf("Stringhe %s e %s non uguali\n", string1, string3);
16
17     int n = 5; // caratteri da confrontare
18     if(!strncmp(string1,string3,n)) // string1 == string3
19         printf("Stringhe %s e %s uguali nei primi %d caratteri\n", string1, string3, n);
20     else printf("Stringhe %s e %s NON uguali nei primi %d caratteri\n", string1, string3, n);
21 }
```

## Perché si usa il «!» (not) ?

Perché `strcmp` restituisce 0 se le due stringhe sono uguali e lo 0 viene identificato in C come «negazione», quindi bisogna utilizzare l'operatore di negazione «!»

# Stringhe – Confronto tra Stringhe

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char string1[] = "test-Ok";      // dichiaro
6      char string2[] = "test-Ok";      // tre variabili
7      char string3[] = "test-Non-Ok"; // di tipo stringa
8
9      if(!strcmp(string1,string2)) // string1 == string2
10         printf("Stringhe %s e %s uguali\n", string1, string2);
11     else printf("Stringhe %s e %s non uguali\n", string1, string2);
12
13     if(!strcmp(string1,string3)) // string1 != string3
14         printf("Stringhe %s e %s uguali\n", string1, string3);
15     else printf("Stringhe %s e %s non uguali\n", string1, string3);
16
17     int n = 5; // caratteri da confrontare
18     if(!strncmp(string1,string3,n)) // string1 == string3
19         printf("Stringhe %s e %s uguali nei primi %d caratteri\n", string1, string3, n);
20     else printf("Stringhe %s e %s NON uguali nei primi %d caratteri\n", string1, string3, n);
21 }
```

gcc version 4.6.3

```
Stringhe test-Ok e test-Ok uguali
Stringhe test-Ok e test-Non-Ok non uguali
Stringhe test-Ok e test-Non-Ok uguali nei primi 5 caratteri
```

# Funzioni per la ricerca nelle stringhe

| Prototipo                                                      | Descrizione                                                                                                                                                                                                                                      |
|----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>char *strchr( const char *s, int c );</code>             | Locates the first occurrence of character <b>c</b> in string <b>s</b> . If <b>c</b> is found, a pointer to <b>c</b> in <b>s</b> is returned. Otherwise, a <b>NULL</b> pointer is returned.                                                       |
| <code>size_t strcspn( const char *s1, const char *s2 );</code> | Determines and returns the length of the initial segment of string <b>s1</b> consisting of characters not contained in string <b>s2</b> .                                                                                                        |
| <code>size_t strspn( const char *s1, const char *s2 );</code>  | Determines and returns the length of the initial segment of string <b>s1</b> consisting only of characters contained in string <b>s2</b> .                                                                                                       |
| <code>char *strpbrk( const char *s1, const char *s2 );</code>  | Locates the first occurrence in string <b>s1</b> of any character in string <b>s2</b> . If a character from string <b>s2</b> is found, a pointer to the character in string <b>s1</b> is returned. Otherwise, a <b>NULL</b> pointer is returned. |

# Funzioni per la ricerca nelle stringhe

| Prototipo                                                    | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>char *strrchr( const char *s, int c );</code>          | Locates the last occurrence of <b>c</b> in string <b>s</b> . If <b>c</b> is found, a pointer to <b>c</b> in string <b>s</b> is returned. Otherwise, a <b>NULL</b> pointer is returned.                                                                                                                                                                                                                                                                                                       |
| <code>char *strstr( const char *s1, const char *s2 );</code> | Locates the first occurrence in string <b>s1</b> of string <b>s2</b> . If the string is found, a pointer to the string in <b>s1</b> is returned. Otherwise, a <b>NULL</b> pointer is returned.                                                                                                                                                                                                                                                                                               |
| <code>char *strtok( char *s1, const char *s2 );</code>       | A sequence of calls to <b>strtok</b> breaks string <b>s1</b> into “tokens”—logical pieces such as words in a line of text—separated by characters contained in string <b>s2</b> . The first call contains <b>s1</b> as the first argument, and subsequent calls to continue tokenizing the same string contain <b>NULL</b> as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, <b>NULL</b> is returned. |
| <code>int strlen( char *s1 );</code>                         | Returns the length of the string <b>s1</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



# Funzioni per la **ricerca** nelle stringhe - Esempi

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main() {
5      char string[] = "prova_stringa";
6
7      // Trova la prima occorrenza della stringa, e restituisce la parte rimanente
8      printf("Parte rimanente dopo la s: %s\n", strchr(string, 's')); // stampa 'stringa'
9
10     // Restituisce il numero di caratteri prima della prima occorrenza di quel carattere
11     printf("Lunghezza Stringa prima della S: %d\n", strcspn(string, "s")); // stampa '6' - nb: parametro è una stringa non un char
12
13     int length = strlen(string); // lunghezza stringa
14     printf("%d", length); // stampa la lunghezza della stringa = 13
15
16 }
```

```
gcc version 4.6.3
Parte rimanente dopo la s: stringa
Lunghezza Stringa prima della S: 6
13
```

# Funzioni per la conversione delle stringhe

| Prototype                                                                        | Description                                             |
|----------------------------------------------------------------------------------|---------------------------------------------------------|
| <code>double atof( const char *nPtr );</code>                                    | Converts the string <code>nPtr</code> to double.        |
| <code>int atoi( const char *nPtr );</code>                                       | Converts the string <code>nPtr</code> to int.           |
| <code>long atol( const char *nPtr );</code>                                      | Converts the string <code>nPtr</code> to long int.      |
| <code>double strtod( const char *nPtr, char **endPtr );</code>                   | Converts the string <code>nPtr</code> to double.        |
| <code>long strtol( const char *nPtr, char **endPtr, int base );</code>           | Converts the string <code>nPtr</code> to long.          |
| <code>unsigned long strtoul( const char *nPtr, char **endPtr, int base );</code> | Converts the string <code>nPtr</code> to unsigned long. |

**Prendono in input una stringa e restituiscono in output la stringa convertita in un numero.**

# Funzioni per la conversione delle stringhe - Esempi

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      double d; /* variable to hold converted string */
7
8      d = atof( "99.0" );
9
10     printf( "%s%.3f\n%s%.3f\n",
11             "The string \"99.0\" converted to double is ", d,
12             "The converted value divided by 2 is ",
13             d / 2.0 );
14
15     return 0;
16
17 }
```

The string "99.0" converted to double is 99.000  
The converted value divided by 2 is 49.500

# Funzioni per la **conversione** delle stringhe - Esempi

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      double d; /* variable to hold converted string */
7
8      d = atof( "99.0" );
9
10     printf( "%s%.3f\n%s%.3f\n",
11             "The string \"99.0\" converted to double is ", d,
12             "The converted value divided by 2 is ",
13             d / 2.0 );
14
15     return 0;
16
17 }
```

The string "99.0" converted to double is 99.000  
The converted value divided by 2 is 49.500

La funzione prende in input un valore **in formato stringa** e lo **converte in un float** (o in un altro formato, a seconda della funzione)

# Funzioni per la conversione delle stringhe - Esempi

```
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8      /* initialize string pointer */
9      const char *string = "51.2% are admitted";
10
11     double d;          /* variable to hold converted sequence */
12     char *stringPtr; /* create char pointer */
13
14     d = strtod( string, &stringPtr );
15
16     printf( "The string \"%s\" is converted to the\n", string );
17     printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );
18
19     return 0;
20
21 } /* end main */
```

The string "51.2% are admitted" is converted to the double value 51.20 and the string "% are admitted"

La funzione prende in input un valore **in formato stringa** e lo **converte in un float** (o in un altro formato, a seconda della funzione). **La parte rimanente viene memorizzata in una ulteriore stringa.**

# Funzioni per la manipolazione delle stringhe

| Prototype                                                        | Description                                                                                                                                                                                                                                    |
|------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>char *strcpy( char *s1, const char *s2 )</code>            | Copies string <code>s2</code> into array <code>s1</code> . The value of <code>s1</code> is returned.                                                                                                                                           |
| <code>char *strncpy( char *s1, const char *s2, size_t n )</code> | Copies at most <code>n</code> characters of string <code>s2</code> into array <code>s1</code> . The value of <code>s1</code> is returned.                                                                                                      |
| <code>char *strcat( char *s1, const char *s2 )</code>            | Appends string <code>s2</code> to array <code>s1</code> . The first character of <code>s2</code> overwrites the terminating null character of <code>s1</code> . The value of <code>s1</code> is returned.                                      |
| <code>char *strncat( char *s1, const char *s2, size_t n )</code> | Appends at most <code>n</code> characters of string <code>s2</code> to array <code>s1</code> . The first character of <code>s2</code> overwrites the terminating null character of <code>s1</code> . The value of <code>s1</code> is returned. |

**Permettono di copiare una parte dei caratteri di una stringa in un'altra stringa oppure di concatenare due stringa in una più grande**

# Funzioni per la manipolazione delle stringhe

```
4  #include <stdio.h>
5  #include <string.h>
6
7  int main() {
8
9  char stringa_a[30] = "Provo ";
10 char stringa_b[30] = "Concatenazione";
11 char stringa_c[30] = "";
12 char stringa_d[30] = "Copia Stringa";
13 char stringa_e[30] = "";
14
15 // Concateno la stringa B alla stringa A
16 printf("%s", strcat(stringa_a,stringa_b));
17 // Concateno i primi 15 caratteri della stringa A a C
18 printf("\n%s", strncat(stringa_c,stringa_a, 15));
19
20 // Copio una stringa in un'altra
21 printf("\n%s", stringa_d);
22 strcpy(stringa_e, stringa_d);
23 printf("\n%s", stringa_e);
24
25 // Lunghezza della stringa
26 printf("\nLunghezza Stringa A: %d", strlen(stringa_a));
27 }
```

# Funzioni per la manipolazione delle stringhe

```
4  #include <stdio.h>
5  #include <string.h>
6
7  int main() {
8
9  char stringa_a[30] = "Provo ";
10 char stringa_b[30] = "Concatenazione";
11 char stringa_c[30] = "";
12 char stringa_d[30] = "Copia Stringa";
13 char stringa_e[30] = "";
14
15 // Concateno la stringa B alla stringa A
16 printf("%s", strcat(stringa_a,stringa_b));
17 // Concateno i primi 15 caratteri della stringa A a C
18 printf("\n%s", strncat(stringa_c,stringa_a, 15));
19
20 // Copio una stringa in un'altra
21 printf("\n%s", stringa_d);
22 strcpy(stringa_e, stringa_d);
23 printf("\n%s", stringa_e);
24
25 // Lunghezza della stringa
26 printf("\nLunghezza Stringa A: %d", strlen(stringa_a));
27 }
```

```
gcc version 4.6.3
>
Provo Concatenazione
Provo Concatena
Copia Stringa
Copia Stringa
Lunghezza Stringa A: 20
```



# Elaborazione di Stringhe

- **Approfondimenti:**
  - Capitolo 8 del libro Deitel & Deitel

# Domande?