Lezione 15

Laura Bozzelli

a.a. 2020/2021

Sommario - Lezione 15: Matrici e stringhe

- Array bidimensionali (matrici).
- Dichiarazione e inizializzazione di matrici.
- Accesso agli elementi di una matrice.
- Matrici e funzioni.
- Matrici verso array di puntatori.
- Stringhe.
- Funzioni di libreria per la manipolazione di caratteri.
- Funzioni di libreria per la manipolazione di stringhe.

Array monodimensionali (1/2)

- Nel linguaggio C, un **array**, detto anche **vettore**, è una sequenza ordinata di dati dello stesso tipo, raggruppati sotto un unico nome.

- I singoli dati in un array monodimensionale sono chiamati **elementi dell'array** e possono essere acceduti tramite l'**operatore di indicizzazione []**.

- Elementi consecutivi in un array hanno **locazioni di memoria** contigue.

- Il numero di elementi di un array monodimensionale è chiamato **lunghezza dell'array**. Gli array sono in generale dati **a lunghezza variabile** dal momento che la lunghezza di un array può essere specificata tramite espressioni numeriche intere il cui valore è noto a tempo di esecuzione.

Array monodimensionali (2/2)

- A differenza degli aggregati (strutture e unioni) e dei tipi enumerativi, per definire una variabile di tipo array non bisogna prima definire un tipo array.

- Nella dichiarazione di una variabile di tipo array monodimensionale si specifica il nome della variabile, il tipo comune degli elementi di un array, e la lunghezza dell'array.

- Il tipo di un elemento di un array monodimensionale può essere un tipo predefinito semplice, un tipo **struct**, un tipo **union**, o un tipo enumerativo.

- Una variabile array è una **variabile puntatore costante**: il suo valore rappresenta **sempre** l'indirizzo di memoria della locazione di memoria associata al primo elemento dell'array.

Array bidimensionali (1/5)

- Un **array bidimensionale** (detto anche **matrice**) o **array di dimensione 2** è un array monodimensionale i cui elementi sono array monodimensionali dello stesso tipo e aventi la stessa lunghezza L.

- I singoli elementi di tipo array di un array bidimensionale sono chiamati **righe** e hanno la stessa lunghezza. Per elemento di un array bidimensionale (o matrice) intendiamo un elemento di qualche riga della matrice.

- La lunghezza complessiva di una matrice corrisponde al numero complessivo dei sui elementi e, cioè, il numero di righe moltiplicato per la lunghezza di ciascuna riga.

Array bidimensionali (2/5)

- Un uso comune degli array bidimensionali (matrici) è quello di rappresentare **tabelle** di valori che contengono informazioni disposte in *righe* e *colonne*.

- I singoli elementi di un array bidimensionale vengono acceduti specificando tramite il duplice uso dell'operatore di indicizzazione **[]** due indici:
  - L'indice di riga che va da 0 al numero di righe -1.
  - L'indice di colonna che va da 0 a L-1, dove L è la lunghezza di ciascuna riga.

- Il **numero di colonne** di una matrice corrisponde alla lunghezza di ciascuna riga. Ad esempio, per una matrice **A** con N righe e M colonne per accedere all'elemento di indice di riga $0 \leq i \leq N-1$ e indice di colonna $0 \leq j \leq M-1$, si utilizza l'espressione A[i][j].

Array bidimensionali (3/5)

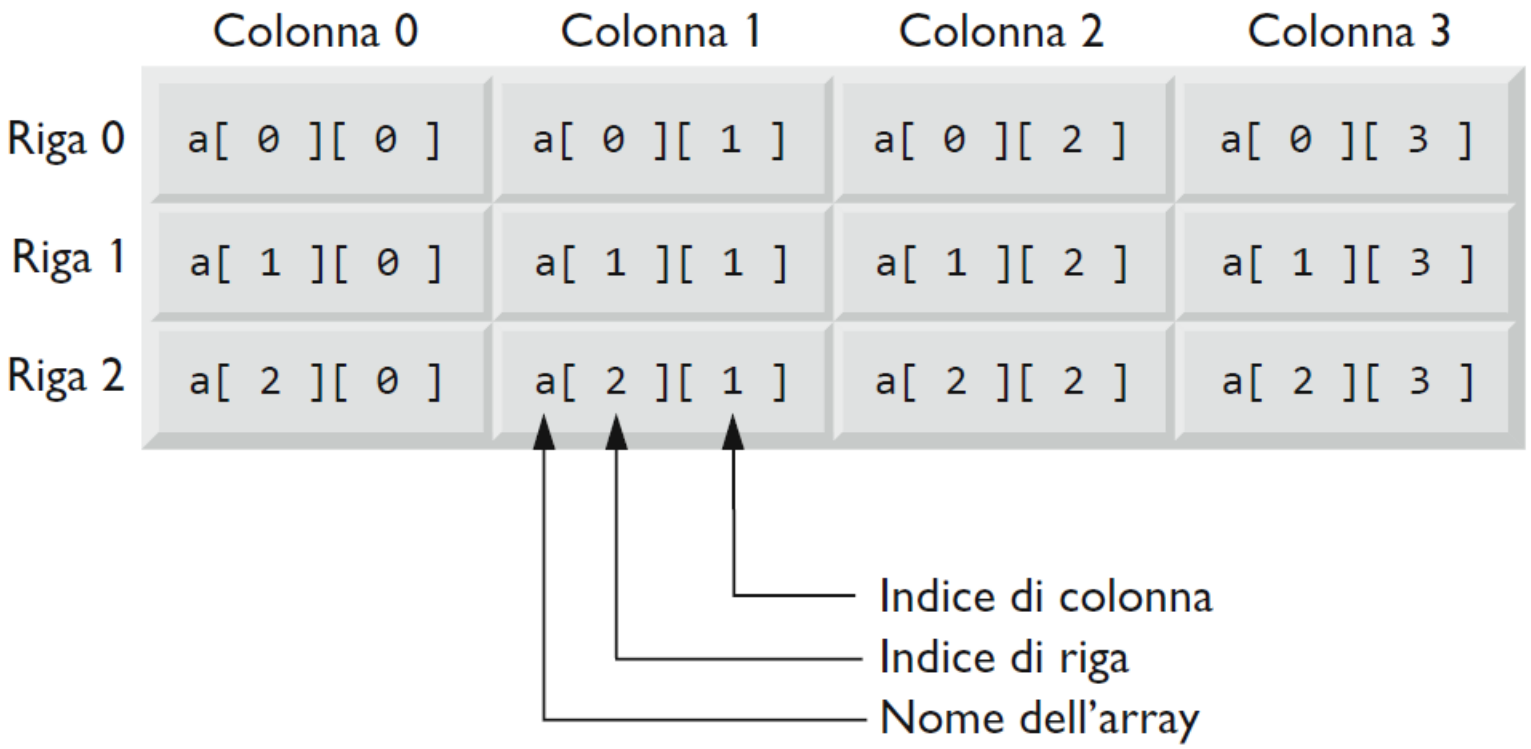
Dal punto di vista della memoria, gli array bidimensionali sono gestiti nel linguaggio C come array monodimensionali. Questo significa che tutti gli elementi della matrice sono disposti in memoria come un'unica sequenza ordinata (per indirizzi di memoria) di celle di memoria contigue:

- Gli elementi della prima riga precedono quelli della seconda riga, e così via. Le righe sono ordinate per valori crescenti dell'indice di riga.
- Elementi all'interno di una stessa riga sono ordinati per valori crescenti dell'indice di colonna.

Un array bidimensionale (matrice) con M righe e N colonne è chiamato anche **array o matrice M per N**.

Array bidimensionali (4/5)

Esempio: matrice **a** 3 per 4
Gli elementi della riga i con $0 \leq i \leq 2$ hanno tutti il primo indice (indice di riga) uguale a i.
Gli elementi della colonna j con $0 \leq j \leq 3$ hanno tutti il secondo indice (indice di colonna) uguale a j.



Array bidimensionali (5/5)

- Similmente agli array monodimensionali, per definire una variabile di tipo array bidimensionale non bisogna prima definire un tipo array bidimensionale.

- Nella dichiarazione di una variabile di tipo array bidimensionale si specifica il nome della variabile, il tipo comune degli elementi della matrice, il numero di righe, ed il numero di colonne.

- Il tipo di un elemento di un array bidimensionale può essere un tipo predefinito semplice, un tipo **struct**, un tipo **union**, o un tipo enumerativo.

Dichiarazione array bidimensionali (1/3)

Sintassi matrici in C:

<tipo elemento> <nome matrice> [<N_Righe>][<N_Colonne>];

- **<tipo elemento>**: rappresenta il tipo comune degli elementi. Per elementi semplici è l'insieme di parole chiave per identificare il tipo semplice (int, double, char, long double, ecc). Per i tipi struttura (risp., unione, risp., enumerazione) è la parola chiave **struct** (risp., **union**, risp., **enum**) seguita dal nome del tipo (come specificato nella definizione di tipo).
- **<nome matrice>**: nome della matrice (qualsiasi identificatore valido).
- **<N_Righe>**: numero di righe. Può essere una qualsiasi espressione numerica di tipo intero.
- **<N_Colonne>** : numero di colonne. Può essere una qualsiasi espressione numerica di tipo intero.

Dichiarazione array bidimensionali (2/3)

Esempio dichiarazione matrici in C: matrici di strutture

```
struct  Punto
{
    float x; //coordinata lungo l'asse X
    float y; //coordinata lungo l'asse Y
};

//Matrice di punti 10x20
//10 righe e 20 colonne
struct Punto GrigliaPunti[10][20];
```

Dichiarazione array bidimensionali (3/3)

Sintassi:

<tipo elemento> <nome matrice> [<N_Righe>][<N_Colonne>];

- Per dichiarazioni di (variabili) matrici globali o locali statiche (e, cioè, il cui campo di memoria è statico), le espressioni **<N_Righe>** e **<N_Colonne>** devono essere espressioni numeriche intere **costante** i cui valori, determinabili a tempo di compilazione, devono essere interi positivi. Altrimenti, il compilatore segnala un errore di sintassi. Per tali variabili, l'area di memoria viene allocata all'inizio dell'esecuzione del programma.
- Per dichiarazioni di matrici locali automatiche, è possibile utilizzare espressioni numeriche intere generiche per specificare il numero di righe ed il numero di colonne. La valutazione a tempo di esecuzione dell'espressione numerica per il numero di righe e numero di colonne può generare un errore irreversibile se il valore ottenuto non è positivo e eccede la capacità di memoria a disposizione.

Lista di inizializzazione per matrici (1/3)

Dichiarazione matrice con lista di inizializzazione:

<tipo elemento> <nome matrice> [<N_Righe>][<N_Colonne>]= <inizializzazione>;

<N_Righe> e **<N_Colonne>** devono essere **espressioni intere costanti.**

Sintassi <inizializzazione>: rappresenta la parte di inizializzazione.

{ <Init 1>, ….,<Init N>}

- N deve essere non superiore al numero di righe.
- **<Init i>** inizializza la righe di indice i-1: è una lista di inizializzazione completa o parziale per la riga di indice i che specifica i valori dei singoli elementi separati da virgole e racchiusi tra parentesi graffe
- Tutti gli elementi che **non** hanno un inizializzatore esplicito sono inizializzati automaticamente a zero.
- Per **dichiarazioni di variabili globali o statiche**, le espressioni numeriche utilizzate per l'inizializzazione dei singoli elementi devono essere costanti.

Lista di inizializzazione per matrici (2/3)

Esempio:

```
int matrix[4][4]={{1,0,0,0},{0,1,0,0},
                  {0,0,1,0}, {0,0,0,1}}
```

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Lista di inizializzazione per matrici (3/3)

Esempio:

```
//Array 2X3
//La prima sottolista inizializza la riga di indice 0 con i valori 1,2, e 3
//La seconda sottolista inizia la riga di indice 1 con i valori 5, 6, e 7
int A[2][3] ={{1,2,3},{4,5,6}};

//Array 2X3
//La prima sottolista inizializza i primi due elementi della prima riga
// con i valori 1 e 2.
//Il terzo elemento della prima riga è implicitamente inizializzato a zero.
//La seconda sottolista inizializza esplicitamente
//il primo elemento della seconda riga a 4
int B[2][3] = {{1,2},{4}};

//Array 2X3. Notazione alternativa nella lista di inizializzazione
//Le parentesi graffe vengono rimosse dalla lista degli inizializzatori
// delle singole righe. Il compilatore inizializza gli elementi della prima
// riga seguiti dagli elementi della seconda riga,ecc.
//Gli elementi che non hanno un inizializzatore esplicito sono inizializzati
//automaticamente a zero.
int C[2][3]= {1,2,3,4,5};
```

Accesso agli elementi di una matrice (1/2)

- Ad ogni elemento di un array bidimensionale è associato **un indice di riga** che va da 0 al numero di righe meno 1 ed un **indice di colonna** che va da 0 al numero di colonne meno 1.
- Si può far riferimento a uno qualunque degli elementi di un array bidimensionale fornendo il nome dell'array seguito dall'indice di riga racchiuso tra parentesi quadre a sua volta seguito dall'indice di colonna racchiuso tra parentesi quadre.

Accesso agli elementi di una matrice (2/2)

Sintassi accesso agli elementi di una matrice.

<nome matrice>[<indice riga>] [<indice colonna>]

- <indice riga> e <indice colonna> sono generiche espressioni numerica intere.
- L'espressione <nome matrice>[<indice riga>] [<indice colonna>] è un **left value**:può essere utilizzata alla stessa stregua di una variabile avente come tipo il tipo degli elementi della matrice. In particolare, può comparire come operando sinistro in un'istruzione di assegnazione e come parte di un'espressione numerica se il tipo degli elementi è numerico.

```
int D[5][8];

int i= 2;
D[i][i*i] = i+i;
```

Memorizzazione di una matrice (1/2)

Le matrici sono memorizzate per righe in celle contigue.

Matrice **A** con N righe e M colonne:

- Una variabile matrice **A** rappresenta una variabile puntatore all'array monodimensionale corrispondente alla prima riga della matrice.
- Per ogni indice di riga $0 \leq i \leq N-1$, A[i] è un puntatore costante il cui valore è l'indirizzo di memoria del primo elemento della riga i: **le espressioni A[i] e &A[i][0] sono equivalenti**.
- La variabile **A** è un puntatore costante a puntatori costanti: il suo valore coincide con l'indirizzo di memoria di A[0]: le espressioni A , &A[0], e &&A[0][0] sono equivalenti.

Memorizzazione di una matrice (2/2)

Le matrici sono memorizzate per righe in celle contigue.

Matrice **A** con N righe e M colonne:

- Nell'aritmetica dei puntatori, l'**offset** dell'elemento A[i][j] rispetto all'indirizzo di memoria di A[0][0] è M*i + j.

&A[i][j] ⟺ &A[0][0] + M*i +j ⟺ A[i]+j ⟺ *(A+i) + j

A[i][j] ⟺ *(&A[0][0] + M*i +j) ⟺ *(A[i]+j) ⟺ *(*(A+i) + j)

Array bidimensionali e funzioni (1/5)

- Le funzioni possono avere come parametri formali variabili di qualsiasi tipo e, dunque, anche variabili matrici (array bidimensionali).
- In una funzione, la dichiarazione di un parametro formale di tipo matrice corrisponde alla dichiarazione di una variabile matrice dove è necessario specificare il numero di colonne ma non il numero di righe.

Sintassi parametro formale array monodimensionale:

<tipo elemento> <nome parametro> [ ][<N_Colonne>]

- **<N_Colonne>**: deve essere un'espressione numerica costante. Va dichiarato necessariamente il numero di colonne dal momento che ciò che viene passato è un puntatore ad un array di righe ed il compilatore deve conoscere la lunghezza di ogni riga (e, cioè, il numero di colonne) per consentire l'accesso agli elementi della matrice.

Array bidimensionali e funzioni (2/5)

- **Sintassi parametro formale array monodimensionale:**

  <tipo elemento> <nome parametro> [ ][<N_Colonne>]

- Il numero di righe è irrilevante ai fini dell'aritmetica dei puntatori su **<nome parametro>** .
- L'argomento (parametro attuale) in una chiamata di funzione deve essere il **nome di una variabile matrice** i cui elementi hanno lo stesso tipo degli elementi del parametro formale.

Array bidimensionali e funzioni (3/5)

- Una variabile matrice **A** è una **variabile puntatore a puntatore**: il suo valore è l'indirizzo di memoria dell'array monodimensionale corrispondente alla prima riga di A. Il valore di A è &A[0].
- Quando in una chiamata di funzione viene passato come argomento una variabile matrice **A**, viene copiato nel corrispondente parametro formale l'indirizzo di memoria di A[0].
- Di conseguenza, quando la funzione chiamata modifica nel suo corpo gli elementi del parametro formale, essa modifica gli effettivi elementi della matrice passato come argomento nelle loro originarie locazioni di memoria.
- Dunque, le matrici come gli array monodimensionali **vengono passati per riferimento**.

Array bidimensionali e funzioni (4/5)

- **Sintassi parametro formale array monodimensionale:**

  <tipo elemento> <nome parametro> [ ][<N_Colonne>]

- **Sintassi alternativa come puntatore alla prima riga:**

  <tipo elemento> (* <nome parametro>) [<N_Colonne>]

BUONA NORMA: specificare nel prototipo della funzione un parametro intero che rappresenta il numero di righe della matrice.

Array bidimensionali e funzioni (5/5)

- **Sintassi alternativa come puntatore di puntatore:**

  <tipo elemento> ** <nome parametro>

In questo caso, nella chiamata di funzione, deve essere passato il nome di un array monodimensionale di puntatori di lunghezza pari al numero di righe della data matrice **A** e il cui elemento i-esimo ha come valore l'indirizzo della riga i-esima di **A** e, cioè, A[i].

BUONA NORMA: adottando la notazione di puntatore a puntatore, specificare nel prototipo della funzione un parametro intero che rappresenta il numero di righe della matrice ed un altro parametro intero rappresentante il numero di colonne.

Esempio: inizializzazione matrice (1/3)

```c
#define N_Colonne 4

void InizializzaMatrice(int A[][N_Colonne], unsigned int N_Righe)
{
    int i,j;
    //Ciclo esterno: itera sul
    //numero di righe
    for(i=0;i<N_Righe;i++)
    {
        //Ciclo interni: itera sul
        //numero di colonne
        for(j=0;j<N_Colonne;j++)
        {
            printf("\n Inserici elemento A[%u,%u]: ",i,j);
            scanf("%d",&A[i][j]);
        }
    }
}


int main()
{
    unsigned int N_Righe = 4;
    int A[N_Righe][N_Colonne];
    InizializzaMatrice(A,N_Righe);
 }
```

Esempio: inizializzazione matrice (2/3)

```c
#define N_Colonne 4

void InizializzaMatrice2(int (* A)[N_Colonne], unsigned int N_Righe)
{
    int i,j;
    //Ciclo esterno: itera sul
    //numero di righe
    for(i=0;i<N_Righe;i++)
    {
        //Ciclo interni: itera sul
        //numero di colonne
        for(j=0;j<N_Colonne;j++)
        {
            printf("\n Inserici elemento A[%u,%u]: ",i,j);
            scanf("%d",&A[i][j]);
        }
    }
}

int main()
{
    unsigned int N_Righe = 4;
    int A[N_Righe][N_Colonne];
    InizializzaMatrice2(A,N_Righe);
    return 0;
}
```

Esempio: inizializzazione matrice (3/3)

```
void InizializzaMatrice3(int ** A,  unsigned int N_Righe,
                         unsigned int N_Colonne)
{
    int i,j;
    for(i=0;i<N_Righe;i++)
    {
        for(j=0;j<N_Colonne;j++)
        {
            printf("\n Inserici elemento A[%u,%u]: ",i,j);
            scanf("%d", &A[i][j]);
        }
    }
}

int main()
{
    unsigned int N_Righe = 4,N_Colonne =4;
    int A[N_Righe][N_Colonne];
    //Array di puntatori ad interi
    int * B[N_Righe];
    int i;
    for(i=0;i<N_Righe;i++)
       B[i] = A[i];
    InizializzaMatrice3(B,N_Righe,N_Colonne);
    return 0;
}
```

Matrice verso array di puntatori (1/3)

Un array di puntatori è un array monodimensionale i cui elementi sono puntatori a dati di un certo tipo.

Sintassi dichiarazione array di puntatori:

<tipo elemento> * <nome array> [<lunghezza array>]

La definizione alloca un array di N puntatori ad elementi aventi tipo **<tipo elemento>** dove N è il valore di **<lunghezza array>**.

- Ogni puntatore deve essere esplicitamente istanziato in modo statico oppure tramite funzioni di libreria per l'allocazione dinamica della memoria (lo vedremo in seguito).

Matrice verso array di puntatori (2/3)

- Una matrice **A** N × M può essere vista come un'array di puntatori costanti di lunghezza pari al numero di righe. L'i-esimo elemento dell'array memorizza l'indirizzo della riga i-esima e, cioè, il valore di A[i].
- I vettori di puntatori sono più generali delle matrici dal momento che elementi distinti dell'array possono puntare ad array monodimensionali (righe) aventi lunghezza diverse (**array di array di lunghezze diversa**).
- Dunque, l'importante vantaggio offerto da un array di puntatori consiste nel fatto che esso consente di avere righe di lunghezza variabile.
- L'uso più frequente dei vettori di puntatori consiste nel memorizzare stringhe (sequenze di caratteri) di diversa lunghezza.

Matrice verso array di puntatori (3/3)

Esempio:

```
//Definizione matrice di interi 10X20
int A[10][20];
//Definizione array di puntatori ad interi
// di lunghezza 10
int * B[10];
```

- Ad esempio, **A[3][4] e B[3][4]** sono entrambi riferimenti ad un **int** sintatticamente corretti.
- Tuttavia, **A** è un vettore bidimensionale: per esso sono state riservate 200 locazioni di memoria contigue di ampiezza pari a quella di un **int**.
- Per **B**, invece, la definizione alloca soltanto 10 locazioni di memoria contigue di ampiezza pari a quella per memorizzare un indirizzo di memoria. Ognuno dei 10 puntatori nell'array deve essere inizializzato esplicitamente in modo statico o **dinamico**.

Esempio utilizzo matrici

Calcolo del massimo degli elementi di una matrice di interi.

```c
#define N_Colonne 4
int CalcolaMassimo(int A[][N_Colonne],  unsigned int N_Righe)
{
    int max = A[0][0];
    int i,j;
    for(i=0;i<N_Righe;i++)
    {
        for(j=0;j<N_Colonne;j++)
        {
            if(max<A[i][j])
               max=A[i][j];
        }
    }
    return max;
}

int main()
{
    unsigned int N_Righe = 4;
    int A[N_Righe][N_Colonne];
    InizializzaMatrice(A,N_Righe);
    int max= CalcolaMassimo(A,N_Righe);
    printf("Il massimo degli elementi della matrice e\' %d",max);
    return 0;
}
```

Tipi carattere

Ricordiamo che i tipi carattere in C sono tipi interi a 1 byte utilizzati per rappresentare i caratteri alfanumerici e alcuni caratteri speciali (**sequenze di escape**) in accordo allo standard di codifica dei caratteri ASCII (American Standard Code for Information Interchange) tramiti interi a 8 bits. Vi sono due tipi per caratteri in C, uno con segno e l'altro senza segno:

- char
- unsigned char

Costante carattere: simbolo di carattere racchiuso tra apici singoli come 'a' . Il valore di una costante carattere è il valore numerico di quel carattere all'interno del set di caratteri della macchina (usualmente il codice ASCII del carattere).

Stringhe

- Ricordiamo che nel linguaggio C, una **costante stringa** o **stringa letterale** è una arbitraria sequenza di caratteri racchiusa tra doppi apici, ad esempio *"Esame di Programmazione"*.
- Per stringa si intende un array (monodimensionale) di caratteri (elementi di tipo **char**) il cui ultimo carattere coincide con il **carattere nullo di terminazione** ('\0'). La lunghezza di una stringa coincide con la lunghezza dell'array meno 1 (il carattere di terminazione non viene considerato).
- Il linguaggio C fornisce alcuni costrutti per il trattamento specifico di stringhe insieme a funzioni di libreria per la manipolazione di stringhe.

Dichiarazione e inizializzazione di stringhe (1/2)

Una stringa può essere dichiarata come un normale array monodimensionale di elementi di tipo **char** assicurando nella fase di inizializzazione che l'ultimo carattere sia '\0'. Il linguaggio fornisce costrutti alternativi per la dichiarazione e l'inizializzazione di variabili stringhe.

Sintassi alternativa con lista di inizializzazione:

char <nome stringa> [ ] = {<c_1>,…,<c_N>,'\0'};

- **<c_1>,…,<c_N>** sono costanti carattere.
- Definisce un array di caratteri di lunghezza N+1 che termina con il carattere di terminazione nulla.

Esempio:

```
char color[] = {'b','l','u','\0'};
```

Dichiarazione e inizializzazione di stringhe (2/2)

Sintassi alternativa con inizializzazione basata su costante stringa:

char <nome stringa> [ ] = <costante stringa>;

oppure come un puntatore a carattere:

char * <nome stringa> = <costante stringa>;

Definisce un array di caratteri di lunghezza N+1 che termina con il carattere di terminazione nullo ‘\0’ dove N è la lunghezza della costante stringa.

Esempio:

```
char color[] = "blue";
char * colorPtr ="blue";
```

Stampa di una stringa di caratteri

Un array di caratteri che rappresenta una stringa può essere inviato in uscita con **printf** usando lo specificatore di conversione **%s**.

Esempio:

```
int main()
{
    char color[] = "blue";
    printf("s%\n",color);
}
```

La funzione **printf** *non* controlla quanto è grande l'array di caratteri. I caratteri della stringa sono stampati finchè non si incontra un carattere nullo di terminazione.

Inizializzazione di stringhe tramite scanf (1/2)

- È possibile utilizzare lo specificatore di conversione %s per inizializzare tramite la funzione **scanf** un array di caratteri ad una stringa di caratteri inseriti dall'utente.
- Quando **scanf** incontra lo specificatore %s, legge i caratteri e li memorizza nell'array di input finchè non incontra uno spazio, una tabulazione, un newline o un indicatore di fine file. A questo punto, la funzione inizializza l'elemento corrente dell'array di input con '\0'.

Inizializzazione di stringhe tramite scanf (2/2)

- È importante assicurarsi che il numero di caratteri processati + il carattere nullo di terminazione non superi la lunghezza del vettore di caratteri (altrimenti si genera un **overflow del buffer**). Ciò può essere garantito utilizzando lo specificatore di conversione per stringhe nel formato %Ns dove N è una costante intera non negativa che indica il numero massimo di caratteri che possono essere letti ed inseriti nell'array di input.

Esempio:

```
int main()
{
    char stringa[20];
    scanf("%19s\n",stringa);
    printf("Stringa = %s",stringa);
}
```

Libreria per il trattamento di caratteri (1/3)

La **libreria di funzioni per il trattamento di caratteri** (<ctype.h>) includono diverse funzioni che eseguono test e conversioni di dati di tipo carattere. L'argomento di ognuna di queste funzioni è un **int**, il cui valore dev'essere una quantità rappresentabile come **unsigned char**. Ognuna di queste funzioni restituiscono un **int**.

Funzioni di test: ritornano un valore diverso da zero (true) se l'argomento soddisfa la condizione descritta, zero altrimenti.

- **int isalnum (int c)**: testa se c è una cifra decimale o una lettera.
- **int isalpha (int c)**: testa se c è una lettera.
- **int isblank (int c)**: testa se c è un *carattere della classe blank*: uno spazio o tab orizzontale ('\t').

Libreria per il trattamento di caratteri (2/3)

- **int iscntrl (int c)**: testa se c è un *carattere di controllo*: tab orizzontale ('\t'), tab verticale ('\v'), avanzamento pagina ('\f'), avviso ('\a'), backspace ('\b'), ritorno a capo ('\r'), newline ('\n').
- **int isdigit (int c)**: testa se c è una cifra decimale.
- **int isgraph (int c)**: testa se c è un carattere stampabile diverso da uno spazio.
- **int islower (int c)**: testa se c è una lettera minuscola.
- **int isprint (int c)**: testa se c è un carattere stampabile incluso uno spazio.
- **int ispunct (int c)**: testa se c è un carattere stampabile diverso da uno spazio, da una cifra o da una lettera.

Libreria per il trattamento di caratteri (3/3)

- **int isspace (int c)**: testa se c è un *carattere di spaziatura*: newline ('\n'), spazio (‘ '), avanzamento pagina ('\f'), ritorno a capo ('\r'), tab orizzontale ('\t') o verticale ('\v').
- **int isupper (int c)**: testa se c è una lettera maiuscola.
- **int isxdigit (int c)**: testa se c è carattere esadecimale.

Funzioni di conversione:

- **int tolower (int c)**: se c è una lettera maiuscola, restituisce c come lettera minuscola. Altrimenti, restituisce l’argomento inalterato.
- **int toupper (int c)**: se c è una lettera minuscola, restituisce c come lettera maiuscola. Altrimenti, restituisce l’argomento inalterato.

Esempio gestione caratteri

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

//Converti una stringa in lettere maiuscole
void ConvertiLettereMaiuscole(char * sPtr)
{
    while(*sPtr != '\0')//Il carattere corrente non è '\0'
    {
        *sPtr = toupper(*sPtr);//Converti in maiuscolo
        ++sPtr;//fai puntare sPtr al carattere successivo
    }
}

//Stampa i singoli caratteri di una stringa
//Il parametro d'input è un puntatore a dati costanti
//dal momento che la stringa è solo acceduta in lettura
void StampaCaratteriStringa(const char *sPtr)
{
    for(;*sPtr != '\0'; ++sPtr)//nessuna inizializzazione
       printf("%c",*sPtr);
}

int main()
{
   char str[] = "StamPami In Lettere MaiuScole";
   ConvertiLettereMaiuscole(str);
   StampaCaratteriStringa(str);
}
```

Libreria per il trattamento di stringhe

La **libreria di funzioni per il trattamento di stringhe** (<string.h>) fornisce diverse funzioni per:

- Manipolazione di stringhe: copia e concatenazione.
- Confronto lessicografico tra stringhe.
- Ricerca di sotto-stringhe all'interno di stringhe.
- Suddivisione di stringhe in **token**.
- Determinazione della lunghezza di una stringa.
- Funzioni di gestione della memoria (lo vedremo in seguito).

Le funzioni della libreria utilizzano la macro **size_t** che denota un tipo intero senza segno (dipendente dal compilatore).

Copia e concatenazione di stringhe (1/2)

- **char * strcpy (char * s1, const char * s2)**: copia il suo secondo argomento **s2** (array di caratteri) nel suo primo argomento **s1** e restituisce il valore di **s1**. È necessario assicurarsi che l'array di caratteri **s1** sia grande abbastanza per memorizzare la stringa (incluso il carattere nullo di terminazione) riferita da **s2**.

- **char * strncpy (char * s1, const char * s2, size_t n)**: equivalente a **strcpy**, ma **strncpy** addizionalmente specifica il numero di caratteri da copiare nel primo argomento **s1**. Questo implica che la funzione non copia il carattere nullo di terminazione se il parametro **n** è minore della lunghezza dell'array **s2**.

Copia e concatenazione di stringhe (2/2)

- **char * strcat (char * s1, const char * s2)**: aggiunge il suo secondo argomento **s2** (stringa) in coda al suo primo argomento **s1** e restituisce il valore di **s1**. Il primo carattere del secondo argomento sostituisce il carattere ‘\0’ che termina la stringa nel primo argomento. È necessario assicurarsi che l’array di caratteri **s1** sia grande abbastanza per memorizzare la prima stringa concatenata con la seconda stringa.
- **char * strncat (char * s1, const char * s2, size_t n)**: aggiunge un numero specificato **n** di caratteri della seconda stringa in coda alla prima e restituisce il valore di **s1**. Un carattere nullo di terminazione è aggiunto in coda automaticamente al risultato.

Esempio manipolazione stringhe

```c
#include <string.h>

int main()
{
    char x[] = "Verra\' l'ora del giudizio e la verita\'  risplendera\!";
    char y[] = "la verita\' risplendera\'!";
    size_t SIZE = 100;
    char z[SIZE];
    strncpy(z,x,28);
    z[28] ='\0';
    printf("La stringa z e\': %s\n",z);
    printf("Ora la stringa z e\': %s\n",strcat(z,y));
}
```

```
La stringa z e': Verra' l'ora del giudizio e
Ora la stringa z e': Verra' l'ora del giudizio e la verita' risplendera'!
```

Confronto tra stringhe

- **char * strcmp (char * s1, const char * s2)**: confronta la stringa **s1** con la stringa **s2**, restituendo 0, un valore minore di 0 o maggiore di 0 se **s1** è, rispettivamente, uguale, minore o maggiore di **s2** rispetto all'ordinamento lessicografico basato sui codici numerici (ASCII o Unicode) dei singoli caratteri che compongono una stringa.

- **char * strncmp (char * s1, const char * s2, size_t n)**: confronta fino a **n** caratteri della stringa **s1** con la stringa **s2**. La funzione restituisce 0, un valore minore di 0 o maggiore di 0 se **s1** è, rispettivamente, uguale, minore o maggiore di **s2** per i primi **n** caratteri.

Ricerca di caratteri e sottostringhe (1/2)

- **char * strchr (const char * s1, int c)**: cerca *la prima occorrenza* del carattere **c** nella stringa **s1**. Se il carattere viene trovato, restituisce un puntatore al carattere in **s1**. Altrimenti, restituisce NULL.
- **char * strrchr (const char * s1, int c)**: cerca *l'ultima occorrenza* del carattere **c** nella stringa **s1**. Se il carattere viene trovato, restituisce un puntatore al carattere in **s1**. Altrimenti, restituisce NULL.
- **Size_t strncspn (const char * s1, const char * s2)**: restituisce la lunghezza della *parte iniziale* della stringa nel suo primo argomento che **non** contiene alcun carattere appartenente alla stringa nel suo secondo argomento.

Ricerca di caratteri e sottostringhe (2/2)

- **Size_t strnspn (const char * s1, const char * s2)**: restituisce la lunghezza della *parte iniziale* della stringa nel suo primo argomento che contiene solo caratteri della stringa nel suo secondo argomento.
- **char * strpbrk (const char * s1, const char * s2)**: cerca nel primo argomento **s1** (una stringa) *la prima occorrenza* di un qualsiasi carattere che fa parte del suo secondo argomento **s2** (un'altra stringa). Se un carattere viene trovato, restituisce un puntatore al carattere in **s1**. Altrimenti, restituisce NULL.
- **char * strpstr (const char * s1, const char * s2)**: se la prima stringa **s1** contiene una sottostringa uguale a **s2**, restituisce un puntatore all'inizio della prima occorrenza di tale sottostringa in **s1**. Altrimenti, restituisce NULL.

Ricerca di token in stringhe

char * strtok (char * s1, const char * s2)

- Suddivide la stringa **s1** in una collezione di token, sottostringhe della stringa **s1** separate da caratteri (delimitatori) contenuti nella stringa **s2**. Ad esempio, in una riga di testo, ogni parola può essere considerata un token e gli spazi e la punteggiatura che separano le parole possono essere considerati delimitatori.
- Sono necessarie più chiamate alla funzione per ottenere i token. Nella prima chiamata, con il primo argomento che rappresenta la stringa data, viene restituito un puntatore in **s1** al primo token ed il primo delimitatore in **s1** viene sostituito con '\0'. Nelle chiamate successive, vengono restituiti i puntatori agli altri token in **s1** fino a quando non viene restituito NULL (indica che non ci sono più token). In queste chiamate secondarie, il primo argomento deve essere NULL. La funzione tiene traccia internamente, tramite una variabile statica, di un puntatore al carattere successivo della stringa data.

Esempio suddivisione in token

```c
#include <string.h>

int main()
{
    char str[] = "Questa e\' una frase con 7 token!";
    //Usiamo lo spazio come delimitatore
    //Prima chiamata a strtok
    char * tokenPtr = strtok(str," ");

    while(tokenPtr != NULL)
    {
        printf("%s\n",tokenPtr);
        //Successive chiamate a strtok
        tokenPtr = strtok(NULL, " ");
    }
}
```

OUTPUT

```
Questa
e'
una
frase
con
7
token!
```

Lunghezza di una stringa

size_t strlen (const char * s): determina la lunghezza della stringa s restituendo il numero di caratteri in s che precedono il carattere nullo ‘\0’ di terminazione.

Esempio:

```c
#include <string.h>

int main()
{
    const char str[] = "Questa e\' una frase con 7 token!";
    printf("La lunghezza di \''%s\'' e\':\n    %u",str,strlen(str));
}
```

OUTPUT

```
La lunghezza di ''Questa e' una frase con 7 token!'' e':
    32
--------------------------------
```