Stringhe
e
Array multidimensionali

Stringhe

- In C non esiste il tipo “string”
- Le stringhe vengono trattate come array di caratteri:
  – una stringa è un array di caratteri terminato dal carattere ASCII ‘\0’ (marcatore di fine stringa)
- Come per gli array, è compito del programmatore assicurarsi che la dimensione della stringa (incluso il carattere ASCII ‘\0’) non superi lo spazio allocato per l’array

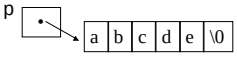
Costanti stringhe

- Sequenze di caratteri delimitate da doppi apici.
  – Esempio: “abc”
- La costante carattere ‘a’ è diversa dalla costante stringa “a”
  – la costante carattere occupa 1 byte mentre la stringa “a” occupa 2 bytes: uno per la ‘a‘ e uno per ‘\0’
- Una costante stringa viene trattata, come accade per i nomi di array, come un puntatore
  – Esempio:

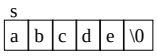
```
char *p = “abcd”;
printf (“%s %s”, p, p+1);    /*stampa abcd  bcd */
```

Costanti stringa in dichiarazioni di di array e puntatori

- Dichiarazione di puntatore: `char *p = “abcde”;`
  – il compilatore alloca spazio per p e per la stringa “abcde”
  – inizializza p con l’indirizzo di base della costante stringa



- Dichiarazione di array di caratteri: `char s[ ] = “abcde”;`
  – e` equivalente a `char s[ ] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘\0’ };`
  – il compilatore riserva 6 byte per l’array s



Esempio dell’uso di costanti stringhe

```
#include <ctype.h>
int word_count(char *s)
{
int cnt = 0;
while (*s != ‘\0’) {
while ( isspace (*s) )
   ++s;
if (*s != ‘\0’) {
  ++ cnt;
while ( ! isspace (*s) && *s != ‘\0’ )
  ++s;
 }                  /* end if */
  }            /* end while */
return (cnt);
}
```

- *La funzione word_count()*
  – riceve una stringa come argomento e restituisce il numero di parole contenute nella stringa
  – usa una macro *isspace(c)* definita in *ctype.h* che verifica se un carattere e` uno spazio, una tabulazione o un newline

Un esercizio

- Vogliamo scrivere una funzione *conv_in_maiuscole()* che prende in input una stringa e la sua lunghezza e restituisce una stringa uguale a quella in input in cui però tutte le lettere dell’alfabeto sono maiuscole
- Esempio:
  – la funzione *conv_in_maiuscole(“Ciao mondo!”, 11)* restituisce la stringa *“CIAO MONDO!”*

La funzione *conv_in_maiuscole*

```
char * conv_in_maiuscole(char *s, int lung)
{
char *ris, *p;
ris = (char *) malloc (lung +1);
p = ris;
while (*s != '\0')
 if ( *s >= 'a' && *s < = 'z')
   *p++ = *s++ –'a' + 'A';
else
   *p ++ = *s++;
*p = '\0';
return (ris);
}
```

- Il puntatore *ris* al termine punterà alla stringa risultante
- Il puntatore *p* serve per creare la nuova stringa:
  - alloca spazio per la nuova stringa
  - se il carattere puntato da s è una lettera minuscola allora inserisce la corrispondente lettera maiuscola, altrimenti inserisce il carattere stesso

Funzioni di libreria per la gestione delle stringhe

- La libreria standard mette a disposizione funzioni per effettuare le più comuni operazioni su stringhe
  - copia di una stringa
  - concatenazione di stringhe
  - lunghezza di una stringa
  - confronto tra stringhe
  - ecc…
- I prototipi sono contenuti nel file *string.h*

Alcune funzioni per le stringhe

*char *strcat (char *s1, const char*s2)*
- concatena *s2* a *s1* e restituisce s1.
- Il programmatore deve assicurarsi che *s1* punti ad uno spazio sufficiente a contenere la stringa concatenata

*int strcmp (const char *s1, const char *s2)*
- effettua il confronto lessicografico tra *s1* e *s2* e restituisce un intero minore di 0 se *s1* precede *s2*, maggiore di 0 se *s1* segue *s2* e *0* se le due stringhe sono uguali.

*char *strcpy (char *s1, const char *s2)*
- copia *s2* in *s1* e restituisce *s1*
- il contenuto precedente di *s1* viene perso
- lo spazio a cui punta *s1* deve essere sufficientemente grande

- *int strlen(const char *s1)*
  – restituisce il numero di caratteri (escluso \0) di *s1*

Due implementazioni di *strlen()*

```
unsigned strlen_1 (char *s)
{
register int n;
for (n = 0; *s != '\0'; ++s)
   n++;
return n;
}
```

- *n* è una variabile *register* per rendere più veloce l'esecuzione

```
unsigned strlen_2(char *s)
{
register int n=0;
while (*s++)
n++;
return n;
}
```

- si esce dal while quando si arriva al marcatore di fine stringa '\0' (carattere nullo il cui valore ASCII è zero)

Un'implementazione di *strcpy()*

```
char *strcpy (char *s1, const char *s2)
{
while (*s1++ = *s2++) ;
return s1;
}
```

- assegnazione, incremento e test contemporaneamente

Un'implementazione (**errata!**) di *strcat()*

```
char *strcat_err (char *s1, const char *s2)
{
register char *p = s1;
while (*p++) ;
while (*p++ = *s2++) ;
}
```

- sposta il puntatore locale *p* a fine stringa *s1*
- copia da quella posizione tutti i caratteri (compreso il carattere '\0') di *s2*

Un'implementazione (corretta) di *strcat()*

```
char *strcat (char *s1, const char *s2)
{
register char *p = s1;
while (*p)
  ++p;
while (*p++ = *s2++) ;
}
```

• sposta il puntatore locale *p* a fine stringa *s1*
• copia da quella posizione tutti i caratteri (compreso il carattere '\0') di *s2*

Array multidimensionali

• In C si possono creare array a più dimensioni

• Il numero di dimensioni di un array è dato dal numero di coppie di parentesi quadrate [ ] utilizzate nella sua dichiarazione

– l'i-esima coppia [ ] racchiude la lunghezza dell'i-esima dimensione dell'array.

–Esempi:

int a[2][4]; dichiara un array bidimensionale

char b[3][2][9]; dichiara un array tridimensionale

• La quantità di memoria allocata per un array multidimensionale è data dal prodotto delle dimensioni

• Gli elementi sono allocati in uno spazio contiguo

Array bidimensionali

•Un array bidimensionale è un array i cui elementi sono a loro volta degli array

– L'array *int a[3][4]* è un array di tre elementi ciascuno dei quali è a sua volta un array di quattro elementi di tipo *int*

• Gli elementi di un array bidimensionale possono essere immaginati come disposti sulle *righe* e le *colonne di un rettangolo*

– L'array *int a[3][4] consiste di 3 righe e 4 colonne*

a[0][0]	**a[0][1]**	**a[0][2]**	**a[0][3]**
a[1][0]	**a[1][1]**	**a[1][2]**	**a[1][3]**
a[2][0]	**a[2][1]**	**a[2][2]**	**a[2][3]**

Array bidimensionali

• Gli elementi di un array bidimensionale vengono memorizzati in maniera contigua riga per riga a partire dall'indirizzo base dell'array

– l'indirizzo base dell'array *int a[3][4]* è *&a[0][0]*

– il nome dell'array *a* è equivalente a &a[0] ed è quindi un puntatore alla prima riga dell'array

• Il *mapping* tra indirizzi di memoria e indici dell'array è detta *mappa di memorizzazione*

– per l'array *int a[3][4]* la mappa di memorizzazione indica che

a[i][j] è equivalente a **(&a[0][0] + 4*i + j)*

Gli array bidimensionali

• L'array *int a[3][4]*

– memorizzato per righe

• Per accedere a *a[i][j]:*

– ci si sposta di *i* righe (4 elementi per riga)

– ci si sposta di *j* posizioni all'interno della riga i-esima

• La *posizione di a[i][j] a partire dall'indirizzo di base è quindi*

*4*i + j*

1021 1022 1023 1024	a[0][0]
	a[0][1]
1025 1026 1027 1028	a[0][2]
	a[0][3]
1029 1030 1031 1032	a[1][0]
	a[1][1]
1033 1034 1035 1036	a[1][2]
	a[1][3]
1037 1038 1039 1040	a[2][0]
	a[2][1]
1041 1042 1043 1044	a[2][2]
	a[2][3]
1045	

Espressioni per accedere a *a[i][j]*

• **(a[i] +j)*

• *(*(a+i)) [j]*

• **((*(a+i)) + j)*

• **(&a[0][0]+4*i+j)*

	a[0][0]	**a[0][1]**	**a[0][2]**	**a[0][3]**
&a[1]	**a[1][0]**	**a[1][1]**	**a[1][2]**	**a[1][3]**
	a[2][0]	**a[2][1]**	**a[2][2]**	**a[2][3]**

Array tridimensionali

• Esempio: *int a[5][8][4]*

– l'indirizzo base dell'array è *&a[0][0][0]*

– si tratta di un array di 5 elementi ognuno dei quali è un array bidimensionale con 8 righe e 4 colonne

– la mappa di memorizzazione viene specificata osservando che

a[i][j][k] è equivalente a *(&a[0][0][0] + 4 *8*i + 4*j + k)*

&a[i][0][0]

&a[i][j][0]

Array multidimensionali come parametri di funzione

• Necessario specificare le lunghezze di tutte le dimensioni eccetto quella della prima

– necessario per determinare la mappa di memorizzazione

• Esempio

```
int sum (int a [ ] [4])
{
int i,j, sum = 0;
for (i=0 ; i<3; ++i)
 for (j = 0; j < 4; j++)
   sum += a[i][j];
return sum;
}
```

Array multidimensionali come parametri di funzione

• Esempio

```
int sum (int a [ ] [8][4])
{
int i,j, sum = 0;
for (i=0 ; i<5; ++i)
 for (i=0 ; i<8; ++i)
   for (k = 0; k < 4; k++)
     sum += a[i][j][k];
return sum;
}
```

Inizializzazione

• Alcune forme di inizializzazione

int a[2][3] = {4, 5, 2, 8, 9, 11};

int a[2][3] = {{4,5, 2}, {8 ,9 ,11}};

int a[][3] = {{4,5, 2}, {8 ,9 ,11}};

Esercizi

• Scrivere una funzione *int cerca(float a[ ])* che legge un numero reale x e lo cerca nell'array *a[ ]*. Se x si trova in *a[ ]* restituisce 1 altrimenti restituisce 0.

•Scrivere una funzione *raddoppia()* che prende in input una stringa e la sua lunghezza e restituisce una stringa contenente gli stessi caratteri della stringa in input ciascuno ripetuto due volte

– Esempio: *raddoppia("ciao",4) restituisce "cciiaaoo"*

• Esercizi 13, 18, 19 del cap 6