Corso di Laboratorio di Sistemi Operativi
A.A. 2016–2017

Lezione 10

Ivan Scagnetto
ivan.scagnetto@uniud.it
Nicola Gigante
gigante.nicola@spes.uniud.it

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 4/11/2016

Manipolazione di stringhe

Gestione delle stringhe in C

Seguendo la sua impostazione a basso livello, il C non fornisce un tipo di dato primitivo per rappresentare stringhe di caratteri.

Array di caratteri

Per rappresentare una stringa ci si affida ad **array di caratteri**.

```
char message[] = "Ciao";
```

Il codice qui sopra è equivalente al seguente:

```
char message[] = { 'C', 'i', 'a', 'o', 0 };
```

- Le stringhe costanti (`"Ciao"`) sono delle scorciatoie per dichiarare array costanti contenenti i caratteri della stringa.
- Per **convenzione**, l'ultimo elemento dell'array è un carattere **nullo**, che segnala la fine della stringa.
- Ciò permette di comunicare la lunghezza della stringa senza utilizzare una variabile aggiuntiva.

Array di caratteri

Esempio

Con scorrimento indicizzato:

```c
#include <stdio.h>

int main() {
  char msg[] = "Ciao";

  printf("Stringa: \"%s\" \n", msg);
  printf("Caratteri: ");

  for(int i = 0; msg[i]; ++i) {
    printf("'%c' ", msg[i]);
  }
  putchar('\n');
}
```

Con scorrimento a puntatori:

```c
#include <stdio.h>

int main() {
  char msg[] = "Ciao";

  printf("Stringa: \"%s\" \n", msg);
  printf("Caratteri: ");

  for(char *p = msg; *p; ++p) {
    printf("'%c' ", *p);
  }
  putchar('\n');
}
```

Funzioni standard per le stringhe

La libreria standard contiene molte funzioni utili a manipolare stringhe, fornite dall'header `<string.h>`:

- Concatenazione di due stringhe, copia di una stringa in un altro arrays, ...
- Confrontare due stringhe
- Estrarre la lunghezza di una stringa, cercare caratteri o sottostringhe, separare una stringa in base a delimitatori, ...

Tutte queste operazioni vanno eseguite con **attenzione**:

- Il risultato deve sempre avere il terminatore nullo nella posizione corretta.
- Non si deve scrivere mai oltre la fine degli array contenenti le stringhe.

Funzioni standard per le stringhe

Lunghezza di una stringa

La funzione

```
int strlen(char *s);
```

restituisce la lunghezza di una stringa.

Possibile implementazione:

```
int strlen(char *s) {
  int n = 0;
  for(char *p = s; *p; ++s) {
    ++n;
  }
  return n;
}
```

Funzioni standard per le stringhe

Confrontare due stringhe

La funzione

```
int strncmp(char *s1, char *s2, unsigned len);
```

confronta lessicograficamente `s1` e `s2`.

- Restituisce **`0`** se sono **uguali**, `-1` o `1` altrimenti.
- Notare il terzo argomento: serve ad evitare che il confronto esca oltre i limiti di uno dei due array se per errore mancasse uno dei terminatori nulli.
- Esiste la versione "meno sicura", `strcmp`, senza terzo argomento.

Funzioni standard per le stringhe

Esempio: utilizzo di `strncmp()`

```c
#include <stdio.h>

#define MAXINPUT 100

int main() {
  char input[MAXINPUT] = "";

  do {
    printf("Password: ");
    scanf("%100s", input);
  } while(strncmp(input, "passw0rd", MAXINPUT) != 0);

  printf("Il segreto e': 42");

  return 0;
}
```

Funzioni standard per le stringhe

Copia di stringhe

La funzione

```
char *strncpy(char *dest, char *source, unsigned len);
```

copia al massimo `len` caratteri dalla stringa `source` nella stringa `dest`.

Possibile implementazione:

```
char *strncpy(char *dest, char *source, unsigned len)
{
  int i = 0;
  while(i < len && source[i]) {
    dest[i] = source[i];
    ++i;
  }

  if(i < len)
    dest[i] = 0;

  return dest;
}
```

Funzioni standard per le stringhe

Concatenazione di stringhe

La funzione

```
char *strncat(char *dest, char *source, unsigned len);
```

concatena al massimo `len` caratteri dalla stringa `source` in fondo a `dest`.

Possibile implementazione:

```
char *strncat(char *dest, char *source, unsigned len)
{
  int destlen = strlen(dest);

  strncpy(dest + destlen, source, len);

  return dest;
}
```

Funzioni standard per le stringhe

Esempio: uso di `strncpy` e `strncat`

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

#define MAXLEN 20

int main() {
  char msg[MAXLEN] = "";
  char msg1[MAXLEN] = "Ciao ";
  char msg2[MAXLEN] = "e Mandi";

  strncpy(msg, msg1, MAXLEN);
  strncat(msg, msg2, MAXLEN);

  printf("%s\n", msg);

  return 0;
}
```

Funzioni standard per le stringhe

Altre funzioni

È consigliabile prendere dimestichezza con la maggior parte delle funzioni esposte da `<string.h>`.

```
man string
```

Avvertenze

Attenzione: Per via della conversione automatica da `T[n]` a `T*`, il seguente codice compila:

```
char *stringa = "Ciao mondo";
```

Tuttavia, al 99% dei casi non fa quello che intendete.

- Non alloca un array per contenere la stringa `"Ciao mondo"`.
- La stringa `"Ciao mondo"` è già presente in memoria, nel **testo** del programma.
- Il puntatore viene initializzato a puntare alla lettera `'C'` nel testo del programma.
- Tale zona di memoria è **a sola lettura**, e quindi manipolare tale stringa potrebbe portare a undefined behavior.
  - Il tipo dei *literal* dovrebbe essere `const char[n]`, in modo da convertirsi solo in `const char *`.
    Non è così per retrocompatibilità.

Avvertenze

Attenzione: Per via della conversione automatica da `T[n]` a `T*`, il seguente codice compila:

```
char *stringa = "Ciao mondo";
```

Esempio di codice **errato**:

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

int main() {
  char *ciao = "Ciao mondo!";

  strncpy(ciao, "Mandi biel!", 11);

  printf("%s\n", ciao);

  return 0;
}
```

Argomenti da riga di comando

Array di puntatori

Essendo i puntatori delle variabili, possono essere a loro volta memorizzati in array; ad esempio la dichiarazione seguente:

```
char *line[42] = { };
```

dichiara un array di 42 elementi, ognuno dei quali è un `char *`.

- In particolare, se ogni `line[i]` punta al primo elemento di un ulteriore array di caratteri, il risultato complessivo sarà che l'array `line` potrà essere utilizzato per memorizzare delle righe di testo.

Array multidimensionali

In C è anche possibile dichiarare array multidimensionali:

```
float matrix[3][4] = {
  { 1, 0, 0 },
  { 0, 1, 0 },
  { 0, 0, 1 },
  { 1, 1, 1 }
};
```

dichiara una matrice di 4 righe e 3 colonne di elementi `float`.

- Per accedere all'elemento in posizione (i, j) si scrive

```
matrix[i][j] = x;
```

- **Attenzione**: La sintassi `matrix[i,j]` compila ma è scorretta (quiz: cos'è?).

Array di puntatori e array multidimensionali

Le dichiarazioni

```
int a[10][20];
int *b[10];
```

differiscono per i motivi seguenti:

- La prima dichiarazione alloca la memoria necessaria per contenere 200 interi (10×20), mentre nel caso della seconda vengono allocate le locazioni necessarie per contenere 10 puntatori ad interi;
- nonostante le espressioni `a[3][4]` e `b[3][4]` denotino entrambe un `int`, nel caso di `b` ogni elemento `b[i]` può puntare ad un array di lunghezza diversa.

Argomenti da riga di comando

Come ogni comando Unix, anche un programma C può ricevere argomenti da riga di comando:

- Vengono passati come argomenti alla funzione `main()`, che può avere questa dichiarazione:

```
int main(int argc, char **argv);
```

- Il primo parametro (`argc`) indica il numero di parametri presenti.
- Il secondo parametro (`argv`) punta ad un **array** di **puntatori a carattere**, ognuno dei quali punta una stringa contenente un argomento.

Argomenti da riga di comando

Come ogni comando Unix, anche un programma C può ricevere argomenti da riga di comando:

- Vengono passati come argomenti alla funzione `main()`, che può avere questa dichiarazione:

```
int main(int argc, char **argv);
```

- Il primo parametro è sempre il nome del programma
- I restanti sono disposti in ordine, così come passati dalla shell.
- L'ultimo elemento (`argv[argc]`) è `NULL`.

Argomenti da riga di comando

Esempio

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

int main(int argc, char **argv)
{
  int somma = 0;

  if(argc >= 2 && strcmp(argv[1], "-s") == 0)
    somma = 1;

  int x = 0, y = 0;

  printf("Inserire due numeri: ");
  int n = scanf("%d %d", &x, &y);

  if(somma)
    printf("%d + %d = %d\n", x, y, x + y);
  else
    printf("%d * %d = %d\n", x, y, x * y);

  return 0;
}
```

Qualche informazione in più su `scanf()`

Qualche informazione in più su `scanf()`

La funzione `scanf()` può essere usata in modi più complessi di quelli visti finora. Ricapitolando le cose già viste:

- La stringa di formato indica quanti input e di che tipo ci si aspetta di leggere.
- Gli argomenti successivi sono **puntatori** alle variabili dove tali valori verranno scritti.

```
int x = 0, y = 0;
scanf("%d %d", &x, &y);
```

- La funzione ritorna il numero di valori letti e convertiti.
  - Due nell'esempio qui sopra
  - Meno di due non viene letto un numero, o se termina l'input

Sintassi della stringa di formato

La funzione `scanf()` è più complessa di quanto visto finora.

- Per default, gli spazi bianchi tra due valori in input vengono ignorati.
- È possibile specificare altri caratteri che devono essere inseriti in input, e che verranno ignorati (ma richiesti), durante la lettura:

```
float real = 0, float imag = 0;
scanf("( %f , %f )", &real, &imag);
```

  In questo esempio, la funzione si aspetta in input due valori numerici, ma separati da una virgola e racchiusi tra parentesi.

- Qualsiasi specificatore, se arricchito con il carattere `'*'`, indica di leggere e convertire un valore secondo il tipo specificato, ma poi di ignorarlo:

```
scanf("(%f %*c %f)", &real, &imag);
```

Funzione `sscanf()`: leggere da una stringa

A volte è utile leggere dall'input un'intera riga di testo, e poi analizzarne il contenuto a parte:

- Per sapere in anticipo la lunghezza e quindi allocare sufficiente memoria
- Per velocità: leggere un blocco di dati in una sola volta è più veloce che leggere un carattere alla volta.

Funzione `sscanf()`: leggere da una stringa

La funzione `sscanf()` supporta questa necessità. Opera come `scanf()`, ma leggendo i dati da una stringa fornita come parametro invece che dallo standard input.

```
char valori="(3.14, 0)";
float real = 0, imag = 0;

sscanf(valori, "%f %f", &real, &imag);
printf("c = %f + i%f\n", real, imag);
```

Le funzioni `sprintf()` e `snprintf()`

Dualmente, esiste la funzione `sprintf()`, che opera come `printf()` ma scrivendo su una stringa invece che sullo standard output:

```
char valori[42] = "";
float real = 3.14, imag = 0;

sprintf(valori, "(%f, %f)", real, imag);
```

Una variante **raccomandata**, `snprintf()`, accetta un ulteriore argomento con la lunghezza massima della stringa di output:

```
snprintf(valori, 42, "(%f, %f)", real, imag);
```

Esercizi

Esercizi

Per venerdì 11 novembre

1. Scrivere le funzioni (ispirate alla libreria standard)

```
char *strchr(char *str, char c);
char *strstr(char *str, char *pattern);
```

che restituiscono il puntatore rispettivamente alla prima occorrenza del carattere `c` e alla prima occorrenza della stringa `pattern` all'interno della stringa puntata da `str`. Se un'occorrenza non viene trovata, si resituisca un puntatore nullo.

Esercizi (2)

Per venerdì 11 novembre

2. Scrivere una funzione

```
int readline(char *line, unsigned len);
```

che riempia la stringa puntata da `line` con caratteri letti dallo standard input, fino a che non viene letto `EOF`, il carattere di new line `'\n'`, o finchè non vengono letti `len` caratteri. Ci si assicuri che la stringa venga terminata correttamente. La funzione deve restituire il numero di caratteri letti, oppure `-1` se nessun carattere è stato letto perchè è stata trovata subito la costante `EOF`.

Esercizi (3)

Per venerdì 11 novembre

3. Scrivere una funzione:

```
void capitalize(char *str);
```

che sostituisca, nella stringa puntata da `str`, ogni occorrenza di una lettera latina minuscola all'inizio di una parola con la corrispondente maiuscola.

Esercizi (4)

Per venerdì 11 novembre

4. Scrivere un programma che legga una sequenza di numeri dallo standard input, e:
   - Se l'utente ha passato l'opzione `-r` sulla riga di comando, stampi i numeri in ordine inverso.
   - Se l'utente ha passato l'opzione `-s`, stampi i numeri ordinati in senso crescente.
   - Se l'utente ha passato l'opzione `-S`, stampi i numeri ordinati in senso decrescente.
   - Se l'utente non ha passato alcuna opzione, stampare un messaggio di errore e non fare nulla (ancora prima di leggere alcunchè).