Il linguaggio C
Gestione della memoria

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
http://www.moreno.marzolla.name/

Ringraziamenti

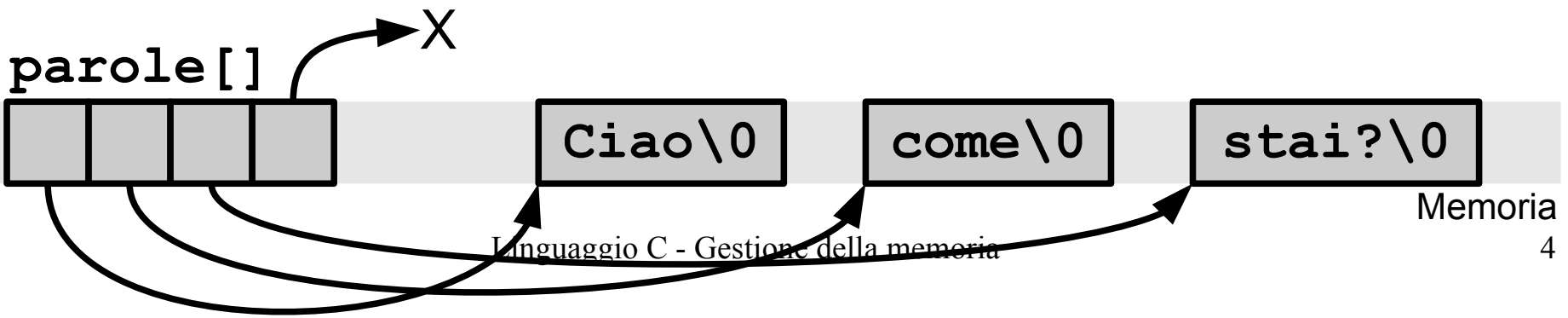
- Questi lucidi si basano su materiale fornito dal prof. Mirko Viroli, Università di Bologna

Puntatori a stringhe

- Sono utili per gestire array di stringhe di caratteri

```
/* es-array-str.c : Esempio di array di stringhe */
#include <stdio.h>

int main( void )
{
    char *parole[] = {"Ciao", "come", "stai?", NULL};
    int i;
    for (i=0; parole[i] != NULL; i++) {
        printf("%s ", parole[i]);
    }
    printf("\n");
    return 0;
}
```



Array di stringhe

- In realtà la funzione **`main()`** può accettare come parametri un intero e un array di stringhe
  - Il sistema operativo li inizializza al numero di argomenti sulla riga di comando, e alle stringhe relative

```
$ ./main-param uno due tre -3
parametro 0: ./main-param
parametro 1: uno
parametro 2: due
parametro 3: tre
parametro 4: -3
```

```
/* main-param.c : Esempio di array di stringhe */
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int i;
    for (i=0; i < argc; i++) {
       printf("parametro %d: %s\n", i, argv[i]);
    }
    return 0;
}
```

La funzione `malloc()`

- Definita in **`stdlib.h`**
- Serve per allocare (riservare) una porzione di memoria da usare per memorizzare qualunque dato
  - Si passa come parametro la dimensione (in byte) richiesta
  - **`malloc()`** ritorna un puntatore **`void *`** che deve essere convertito (con un cast) al puntatore di tipo richiesto
- La memoria va liberata quando non serve più con **`free()`**
- Con **`malloc()`** è possibile creare array la cui dimensione non è nota a priori
  - Consente anche di realizzare funzioni che creano array e li restituiscono come risultato

Esempio

```
/* reverse-fun.c : esempio di funzione che ritorna un array */
#include <stdio.h>
#include <stdlib.h>

/* Dato un array a[] di lunghezza n, restituisce un nuovo
array di lunghezza n il cui contenuto e' il contenuto di a[]
"rovesciato" */
int *reverse(int a[], int n)
{
    int *r = (int *)malloc(n*sizeof(int));
    int i;
    for (i=0; i<n; i++){
        r[i]=a[n-1-i];
    }
    return r;
}

int main( void )
{
   int in[] = {10, 20, 30, 40};
   int *out = reverse(in, 4);
   printf("%d %d %d %d\n", out[0], out[1], out[2], out[3]);
   free(out);
   return 0;
}
```

Contiene la dichiarazione di malloc()

Attenzione

- Il parametro della funzione **`free()`** deve puntare all'inizio di una regione di memoria riservata con **`malloc()`**

OK

```
int *out;
out = (int*)malloc(10*sizeof(int));
out[0] = out[1] = 5;
free(out); /* OK */
```

ERRORE

```
int *out;
out = (int*)malloc(10*sizeof(int));
out[0] = out[1] = 5;
out++;
free(out); /* NO! */
```

OK

```
int *out, *orig;
out = (int*)malloc(10*sizeof(int));
orig = out;
out[0] = out[1] = 5;
out++;
free(orig); /* OK */
```

Attenzione

- La memoria allocata con **`malloc()`** non viene liberata uscendo da una funzione

```
int *prova( void )
{
    int *r = (int *)malloc(10*sizeof(int));
    ...
    return r;  /* OK: la memoria puntata da r rimane allocata anche fuori
dalla funzione */
}
```

Attenzione

- La memoria occupata da un array locale viene invece liberata uscendo da una funzione

```
int *prova( void )
{
    int r[10];
    ...
    return r;  /* NO: la mem. di r[] viene liberata al termine della funzione */
}
```

```
prova.c:7:12: warning: function returns address of local
  variable [-Wreturn-local-addr]
     return r;
            ^
```

La libreria `<string.h>`

- Contiene varie funzioni per manipolare stringhe
- Si ricordi che la stringa "**`prova`**" è un array di 6 caratteri, **`'p' 'r' 'o' 'v' 'a' '\0'`**

Alcune funzioni di `<string.h>`

- `size_t strlen(const char *s);`
  - Restituisce la lunghezza (numero di caratteri escluso terminatore) di s
- `int strcmp(const char *s1, const char *s2);`
  - Ritorna un valore minore, maggiore, uguale a 0 se s1 è "minore", "maggiore", "uguale a" s2
  - Si usa l'ordinamento lessicografico tra stringhe
- `char *strcpy(char *dest, const char *src);`
  - Copia il contenuto di src in dest
  - dest deve puntare ad una area di memoria già allocata sufficientemente capiente
  - restituisce un puntatore a dest
- `char *strcat(char *dest, const char *src);`
  - Appende la stringa src alla fine di dest
  - dest deve puntare ad una area di memoria già allocata sufficientemente capiente
  - restituisce un puntatore a dest

Esempio riassuntivo

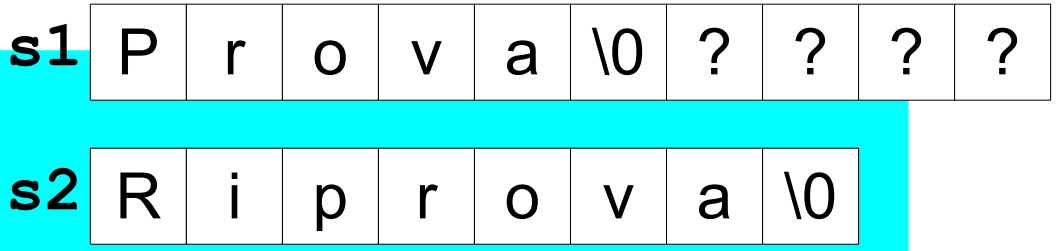
```
Lunghezza di "Prova" e' 5
Lunghezza di "Casa" e' 4
Copia di "Prova" su c3; c3 diventa: "Prova"
Concatena "Casa" a c3: "ProvaCasa"
Confronto "Prova" e "Prova": 0
Confronto "Prova" e "Casa": 13
Confronto "Casa" e "Prova": -13
```

```c
/* demo-string.c */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char c1[] = "Prova";
   char c2[] = "Casa";
   char c3[100];
   printf("Lunghezza di \"%s\" e' %d\n", c1, (int)strlen(c1));
   printf("Lunghezza di \"%s\" e' %d\n", c2, (int)strlen(c2));
   strcpy(c3,c1);
   printf("Copia di \"%s\" su c3; c3 diventa: \"%s\"\n", c1, c3);
   strcat(c3,c2);
   printf("Concatena \"%s\" a c3: \"%s\"\n", c2, c3);
   printf("Confronto \"%s\" e \"%s\": %d\n", c1, c1, strcmp(c1, c1));
   printf("Confronto \"%s\" e \"%s\": %d\n", c1, c2, strcmp(c1, c2));
   printf("Confronto \"%s\" e \"%s\": %d\n", c2, c1, strcmp(c2, c1));
   return 0;
}
```

Attenzione all'allocazione

- Le funzioni **`strcat()`** e **`strcpy()`** vanno a copiare dei caratteri da una stringa **`src`** ad una **`dest`**
- Bisogna fare attenzione ad accertarsi che nel fare ciò non si esca dai limiti consentiti per **`dest`**, rischiando di sovrascrivere memoria altrui




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

int main( void )
{
   char s1[10] = "Prova";
   char s2[] = "Riprova";
   strcat(s1, s2); /* No! Non c'e' abbastanza spazio in s1[] */
   printf("Risultato della concatenazione: \"%s\"\n", s1);
   return 0;
}
```