

Stringhe

Le stringhe

- Le stringhe sono sequenze di caratteri,
 - in C le stringhe costanti vengono denotate da una successione di caratteri racchiusa fra apici Es:

`"ciccio"`

`"n = %d"`

`"Ciao Mondo"`

- La rappresentazione interna è come un array di caratteri non modificabile terminato dal carattere `'\0'`



Le stringhe



- Quindi una stringa occupa un array con un carattere in più riservato al carattere terminatore
- Le stringhe variabili sono rappresentate come array di caratteri:
 - **char parola[M], frase[N];**
 - Per quanto detto prima bisogna sempre ricordarsi di allocare un carattere in più dei caratteri contenuti nelle stringhe che intendiamo scriverci dentro

Le stringhe

- Le costanti di tipo stringa sono di tipo puntatore a carattere:
 - `char * msg = "Errore di conversione";`
 - E non sono modificabili, per dichiarare una stringa modificabile bisogna utilizzare un array di char
 - `char msg[N] = "Errore di conversione"` oppure
`char msg[] = "Errore di conversione";`
 - La stampa delle stringhe si può effettuare direttamente con il modificatore `%s`

Le stringhe

Vediamo un esempio:

```
int main (void) {
    char * msgconst = "Errore di conversione";
    char msg[N] = "Errore di conversione";
    printf( "%s, %s", msgconst, msg);
    /* stampa "Errore di conversione, Errore di conversione" su stdout
    */
    msg[0]='Z';
    printf( "%s", msg);
    /* stampa "Zrrore di conversione" */
    .....
    msgconst[0]='Z';
    /* da errore in esecuzione (Segmentation fault) */
```

Le stringhe

- La libreria **string.h** contiene un insieme di funzioni predefinite per lavorare con le stringhe, ad esempio
- **size_t strlen(char * s)** fornisce la lunghezza della stringa **senza il terminatore**
- **Ad esempio:**

```
int main (void) {  
    char a[5]="ciao";  
    int i;  
    i = strlen(a);  
    /* i vale 4 */  
    /* notare che a e' lungo 5 */  
..... }
```

Le stringhe

Altre funzioni interessanti:

- **char * strcpy(char* s, char* p)**
 - copia la stringa **p** nella stringa **s** e restituisce il puntatore **s**
- **int strcmp(char* s, char* p)**
 - che confronta lessicograficamente **p** ed **s** (restituisce 0 se sono uguali, $n < 0$ se $s < p$ e $n > 0$ altrimenti)
- **char * strcat(char* s, char* p)**
 - che concatena **p** ed **s** (modifica **s**) e restituisce il puntatore **s**
- **char * strstr(char* s, char* p)**
 - che cerca la prima occorrenza della stringa **p** in **s** e restituisce il puntatore a tale occorrenza se esiste NULL altrimenti

Le stringhe

- Vediamo alcuni esempi:

```
int main (void) {  
    char a[5]="ciao";  
    char b[20]="arrivederci";  
    int i;  
    strcpy (b,a);  
    printf( "%s\n", b); /* cosa stampa ???? */  
    ..... }  
}
```


Le stringhe

- Vediamo alcuni esempi:

```
int main (void) {  
    char a[5]="ciao", c[10];  
    char b[20]="arrivederci";  
    int i;  
    strcpy(c,a);  
    strcat(b,":");  
    strcat(b,c);  
    printf( "%s\n", b); /* cosa stampa ? */  
}
```

Le stringhe

- Vediamo alcuni esempi:

```
int main (void) {  
    char a[5]="ciao", char c[10];  
    char b[20]="arrivederci";  
    int i;  
    if (strcmp(a,b)<0)  
        printf( "%s", a);  
    else  
        printf( "%s", b); /* cosa stampa ? */  
}
```

Le stringhe

- Vediamo alcuni esempi:

```
int main (void) {
    char a[5]="ciao", c[10]="ve";
    char b[20]="arrivederci", *s;
    int i;
    s = strstr(a,c);
    printf( "%s\n", s);
    /* cosa stampa ? */
    s = strstr(b,c);
    printf( "%s\n", s);
    /* cosa stampa ? */
}
```

Le stringhe

- Se non gestite bene le stringhe sono pericolose e generano errori difficili da rilevare e catastrofici
 - Le funzioni di libreria si aspettano sempre di lavorare con stringhe correttamente terminate dal carattere nullo '`\0`',
 - Es: implementazione di **strcpy** (da K&R)

```
void strcpy (char*s, char*t){  
    while ( ( *s++ = *t++ ) != '\0' );  
}
```

Le stringhe

```
void strcpy (char*s, char*t){  
    while ( ( *s++ = *t++ ) != '\0');  
}
```

- Se la stringa non è terminata (o se **s** non ha abbastanza spazio) si continuano ad incrementare i puntatori andando avanti a leggere (e scrivere!) valori in memoria (*buffer overrun*)
- Si può sovrascrivere e danneggiare lo spazio di memoria di altre variabili, i frame sullo stack o la tabella di allocazione dello heap
- Si può raggiungere memoria non allocate o non accessibile ricevendo segnali di violazione di Segmento con conseguente terminazione del programma in esecuzione

Le stringhe

Morale :

- Assicuratevi sempre che le stringhe siano terminate e che ci sia abbastanza spazio nei buffer
- Se non siete sicuri della presenza del terminatore usate funzioni che non permettano l'overrun, perchè è possibile dire quanto è grande il buffer

Es: `strncpy(char* s, char*p, size_t n)`

in cui il terzo parametro serve per dire quanto è lungo il buffer **s**

- In questo caso dopo la copia bisogna controllare che il risultato contenga effettivamente il terminatore perchè la fine del buffer può essere stata raggiunta prima
- Usate strumenti come **valgrind** se avete dubbi sul comportamento del vostro programma (segnala scritture e letture fuori dai buffer - sullo heap)

Leggere le stringhe

Sono pericolose:

- **scanf** ("...%s...", p)
 - copia la stringa **p** nella stringa **s**
 - ma meglio evitarla, può causare overrun su **p**
- **char * gets(char * s)**
 - da non usare mai perchè può causare overrun su **s** utilizzeremo **fgets()** (vedi libreria `stdio.h`)

Safe:

- **char * fgets(char* p, int sz, FILE* stream)**

legge dallo stream al più **sz-1** caratteri fino al primo **'\n'** (compreso) ricopia tutto in **p** aggiungendo sempre il terminatore **'\0'**

Esempio: tutto maiuscolo

- Voglio leggere da stdin una serie di stringhe fino all'EOF e stamparle trasformando tutti i caratteri in maiuscoli

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define N 256
int main (void) {
    int i, n; char * p; char buf[N+2];
    p = fgets(buf, N+2, stdin);
    while (p != NULL ) {
        n = strlen(buf);
        for( i=0; i< n-1; i++)
            buf[i]=toupper(buf[i]);
        printf( "%s", buf);
        p = fgets(buf, N+2, stdin);
    } return 0; }
```


Esempio: tutto maiuscolo

```
$ ./maiuscolo
```

Se digito "pippo" e ritorno carrello ↓

```
$ ./maiuscolo  
pippo  
PIPPO
```

Se digito "ciccio" e ritorno carrello ↓

```
$ ./maiuscolo  
pippo  
PIPPO  
ciccio  
CICCIO
```

Esempio: tutto maiuscolo

```
$ ./maiuscolo  
pippo  
PIPPO  
ciccio  
CICCIO
```

```
$ ./maiuscolo  
pippo  
PIPPO  
ciccio  
CICCIO  
$
```

Esco con EOF
(Control + D)

Conversioni stringa numero

- Per convertire una stringa in un tipo numerico sono disponibili diverse funzioni

```
int sscanf ( const char * s, const char * format, ... ),  
int atoi(const char * str),  
long int atol ( const char * str ),  
double atof (const char* str);
```

ASCII to integer, long or double

- Da usare quando siamo sicuri che la stringa e' ben formata e la base di conversione è 10

- long int strtol (const char* str, char** endptr, int base), strtod()** e varianti

per convertire interi/reali quando è necessario gestire i possibili errori o scegliere una base diversa

Esempio: atoi

- Voglio leggere da standard input una serie di stringhe e convertirla in interi (base 10 con `atoi()`)

```
#include <stdio.h>
#include <stdlib.h>
#define N 256
int main (void) {
    int i; char * p; char buf[N+2];
    p = fgets(buf, N+2, stdin);
    while (p != NULL ) {
        i = atoi(buf);
        printf( "risultato = %d\n", i);
        p = fgets(buf, N+2, stdin);
    }
    return 0;
}
```

Esempio: atoi

```
$ ./converti
```

Se digito "56" e ritorno carrello ↓

```
$ ./converti  
56  
risultato = 56
```

Se digito "ciccio" e ritorno carrello ↓

```
$ ./converti  
56  
risultato = 56  
ciccio  
risultato = 0
```

Esempio: strtol

- Voglio leggere da standard input una serie di stringhe convertirla in interi (base 16 con `strtol()`)

```
#include <stdio.h>
#include <stdlib.h>
#define N 256
int main (void) {
    int i; char * p; char buf[N+2];
    p = fgets(buf, N+2, stdin);
    while (p != NULL ) {
        i = strtol(buf, NULL, 16);
        printf( "risultato = %d", i);
        p = fgets(buf, N+2, stdin);
    }
    return 0;
}
```

Esempio: strtol

- Voglio leggere da standard input una serie di stringhe convertirla in interi (base 16 con strtol()) controllando gli errori

```
#include <stdio.h>
#include <stdlib.h>
#define N 256
int main (void) {
    int i; char * p, *q, buf[N+2];
    p = fgets(buf, N+2, stdin);
    while ( p != NULL ) {
        i = strtol(buf,&q,16);
        if ( *q!='\n' ) printf("Errore!\n");
        else printf( "risultato = %d\n", i);
        p = fgets(buf, N+2, stdin);
    } return 0; }
```

Esempio: strtol

```
$ ./converti16
```

Se digito "a123" e ritorno carrello ↓

```
$ ./converti16  
a123  
risultato = 41251
```

Se digito "ciccio" e ritorno carrello ↓

```
$ ./converti16  
a123  
risultato = 41251  
ciccio  
Errore!
```


Memcpy: copiare aree di memoria

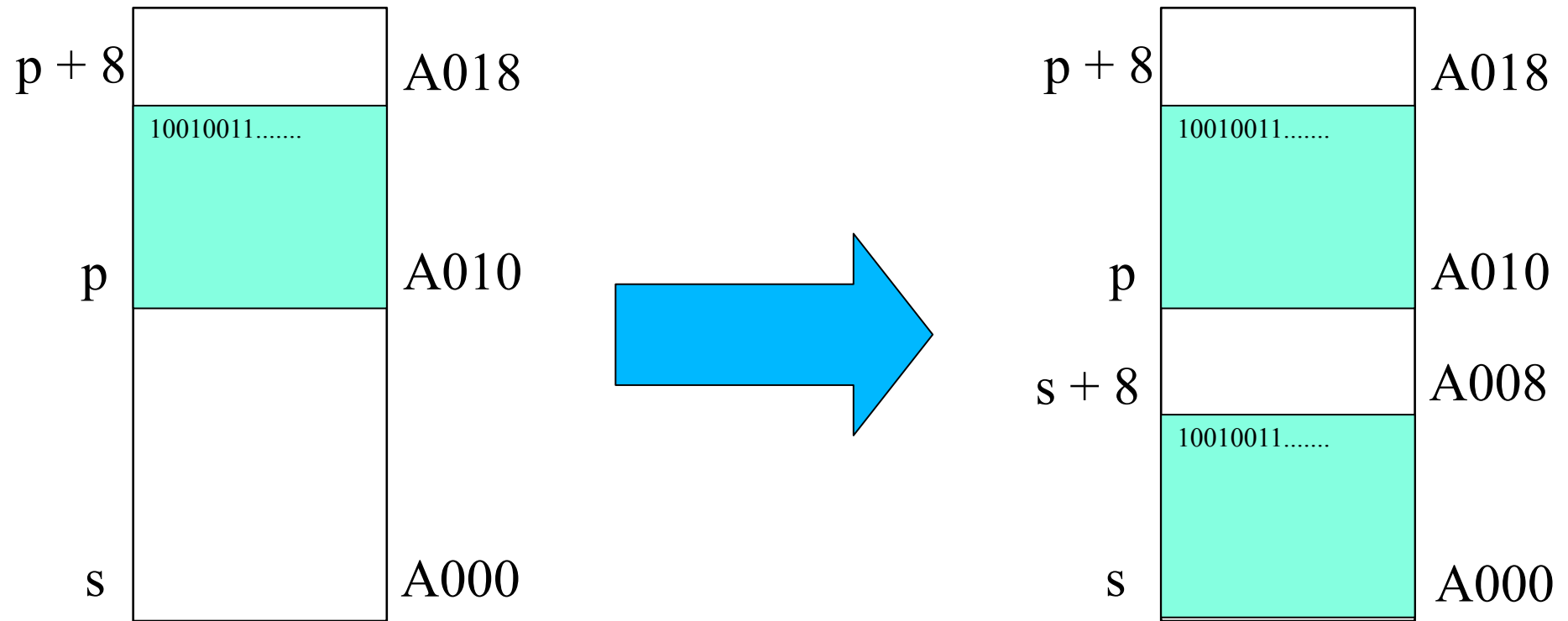
```
memcpy(void* s, const void*p, size_t n)
```

Copia il contenuto della memoria a partire dall'indirizzo **p** per **n** byte (fino a **p+n**) nell'area che va da **s** a **s+n**

Memcpy: copiare aree di memoria

```
memcpy(void* s, const void* p, size_t n)
```

Es:



Memcpy: copiare aree di memoria

```
memcpy(void* s, const void* p, size_t n)
```

Copia il contenuto della memoria a partire dall'indirizzo **p** per **n** byte (fino a **p+n**) nell'area che va da **s** a **s+n**

Es: assegnare un array ad un altro, abbiamo visto che l'assegnamento diretto non è possibile

```
int a[N], b[N];
```

```
....
```

```
memcpy(b, a, N*sizeof(int));
```

```
/* copia tutto a in b */
```