Programmazione I

Il Linguaggio C

Le stringhe

Daniel Riccio
Università di Napoli, Federico II

25 ottobre 2021

Sommario

- Argomenti
  - Stringhe
  - Operazioni sulle stringhe
  - Libreria string.h

Sgtringhe in C

Nel linguaggio C non è supportato esplicitamente alcun tipo di dato “**stringa**”

Le informazioni di tipo stringa vengono memorizzate ed elaborate ricorrendo a semplici vettori di caratteri

```
char saluto[10] ;
```

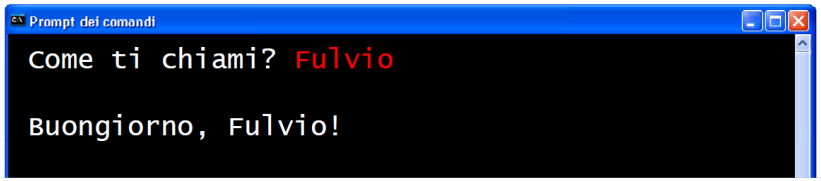
B	u	o	n	g	i	o	r	n	o

Sgtringhe in C

Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max **20** caratteri), e stampi a video un saluto per l'utente stesso

```
#define MAX  20
char nome[MAX];
int N;
char ch;
int i;

printf("Come ti chiami? ");
N = 0;

ch = getchar();
while( ch != '\n' && N<MAX ) {
  nome[N] = ch;
  N++;
  ch = getchar();
}
```

```
printf("Buongiorno, ");
for(i=0; i<N; i++)
  putchar( nome[i] );

printf("!\n");
```



Sgtringhe in C

Qualsiasi operazione sulle stringhe si può realizzare agendo opportunamente su vettori di caratteri, gestiti con **occupazione variabile**

Così facendo, però vi sono alcuni svantaggi.

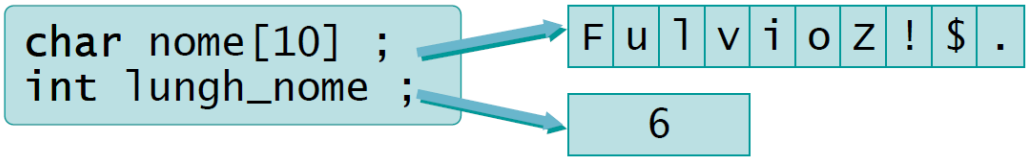
Per ogni vettore di caratteri, occorre definire un'opportuna variabile che ne indichi la lunghezza

Ogni operazione, anche elementare, richiede l'uso di cicli **for**/**while**

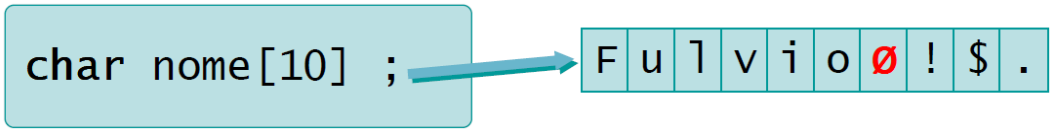
Sgtringhe in C

Vi sono due tecniche per determinare la lunghezza di una stringa

1. utilizzare una variabile intera che memorizzi il numero di caratteri validi



2. utilizzare un carattere “speciale”, con funzione di **terminatore**, dopo l’ultimo carattere valido



Carattere terminatore

Il carattere “**terminatore**” deve avere le seguenti Caratteristiche

- Fare parte della tabella dei codici ASCII
  - Deve essere rappresentabile in un char
- Non comparire mai nelle stringhe utilizzate dal programma
  - Non deve confondersi con i caratteri “normali”

Inoltre, il vettore di caratteri deve avere una posizione libera in più, per memorizzare il terminatore stesso

Carattere terminatore

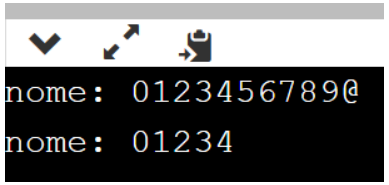
Per convenzione, in C si sceglie che tutte le stringhe siano rappresentate mediante un carattere terminatore

Il terminatore corrisponde al carattere di codice ASCII pari a zero

```
nome[6] = 0;
nome[6] = '\0';
```

F	u	l	v	i	o	ø	!	$	.

```c
#include <stdio.h>
#define MAX 10

int main(){
  int i=0;
  char nome[MAX];

  for(i=0; i<10; i++)
    nome[i] = '0'+i;

  printf("nome: %s\n", nome);
  nome[5] = '\0';

  printf("nome: %s\n", nome);

  return 0;
}
```



Vantaggi

Non è necessaria un'ulteriore variabile intera per ciascuna stringa

L'informazione sulla lunghezza della stringa è interna al vettore stesso

Tutte le funzioni della libreria standard C rispettano questa convenzione

- Si aspettano che la stringa sia terminata
- Restituiscono sempre stringhe terminate

Svantaggi

Necessario 1 byte in più

- Per una stringa di **N** caratteri, serve un vettore di **N**+1 elementi

Necessario ricordare di aggiungere sempre il terminatore

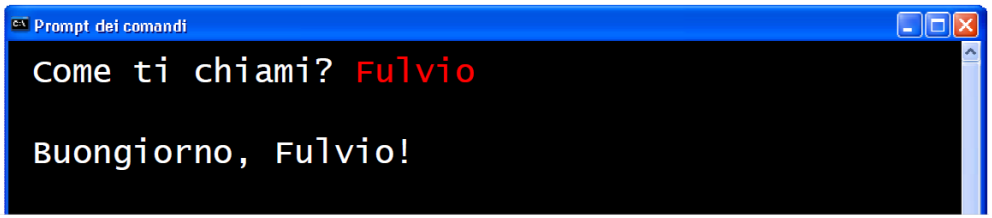
Impossibile rappresentare stringhe contenenti il carattere ASCII 0

Esempio

Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max 20 caratteri), e stampi a video un saluto per l'utente stesso

```
#define MAX 10

char nome[MAX];
char ch;
int i;
printf("Come ti chiami? ");
i = 0 ;

ch = getchar();
while( ch != '\n' && i<MAX ) {
        nome[i] = ch ;
        i++ ;
         ch = getchar();
}
```

```
/* aggiunge terminatore nullo */
nome[i] = '\0' ;

printf("Buongiorno, ");
for(i=0; nome[i]!='\0'; i++)
   putchar( nome[i] ) ;

printf("!\n") ;
```



I/O di stringhe

Diamo per scontato di utilizzare la convenzione del terminatore nullo

Si possono utilizzare

- Funzioni di lettura e scrittura carattere per carattere
  - Come nell'esercizio precedente

Funzioni di lettura e scrittura di stringhe intere

- **scanf** e **printf**
- **gets** e **puts**

I/O di stringhe

Utilizzare la funzione **scanf** con lo specificatore di formato "**%s**"

La variabile da leggere deve essere il nome di un vettore di caratteri

- **Non utilizzare le parentesi quadre**
- **Non utilizzare la &**

Legge ciò che viene immesso da tastiera, fino al primo spazio o fine linea (esclusi)

- **Non adatta a leggere nomi composti** (es. "Pier Paolo")

Esempio:

```
#define MAX 20
char nome[MAX+1] ;
printf("Come ti chiami? ") ;
scanf("%s", nome) ;
```

Lettura di stringhe con gets

La funzione **gets** è pensata appositamente per acquisire una stringa

Accetta un parametro, che corrisponde al nome di un vettore di caratteri

Non utilizzare le parentesi quadre

Legge ciò che viene immesso da tastiera, fino al fine linea (escluso), e compresi eventuali spazi

Possibile leggere nomi composti (es. "Pier Paolo")

Esempio:

```
#define MAX 20
char nome[MAX+1] ;
printf("Come ti chiami? ") ;
gets(nome) ;
```

Scrittura di stringhe con printf

Utilizzare la funzione **printf** con lo specificatore di formato "**%s**"

La variabile da stampare deve essere il nome di un vettore di caratteri

Non utilizzare le parentesi quadre

È possibile combinare la stringa con altre variabili nella stessa istruzione

Esempio:

```
printf("Buongiorno, ") ;
printf("%s", nome) ;
printf("!\n") ;
printf("Buongiorno, %s!\n", nome) ;
```

Scrittura di stringhe con puts

La funzione **puts** è pensata appositamente per stampare una stringa

La variabile da stampare deve essere il nome di un vettore di caratteri

Non utilizzare le parentesi quadre

Va a capo automaticamente

Non è possibile stampare altre informazioni sulla stessa riga

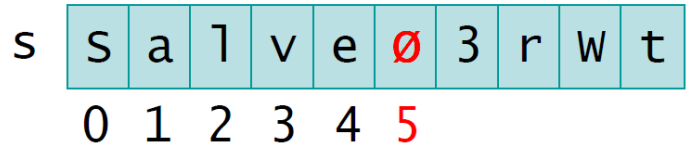
Esempio:

```
printf("Buongiorno, ") ;
puts(nome) ;
/* No!! printf("!\n") ; */
```

Lunghezza di una stringa

La lunghezza di una stringa si può determinare ricercando la posizione del terminatore nullo

```
char s[MAX+1] ;
int lun ;
```



Esempio:

```
#define MAX 10
char s[MAX+1] ;
int lun ;
int i ;
... /* lettura stringa */
for( i=0 ; s[i] != 0 ; i++ )
  /* Niente */ ;
lun = i ;
```

Nella libreria standard C è disponibile la funzione **strlen**, che calcola la lunghezza della stringa passata come parametro

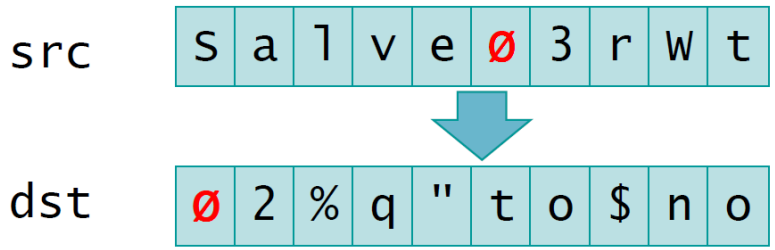
Necessario includere <**string.h**>

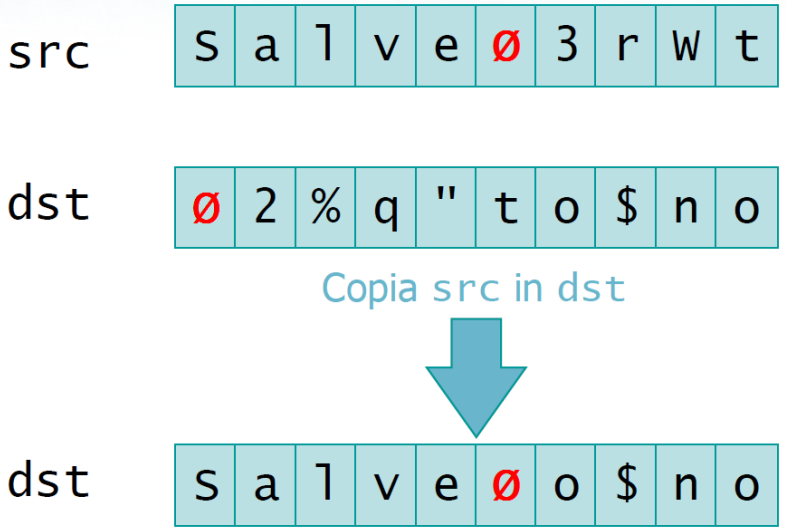
```
#define MAX 10
char s[MAX+1] ;
int lun ;
... /* lettura stringa */
lun = strlen(s) ;
```

Copia di una stringa

L'operazione di copia prevede di ricopiare il contenuto di una prima stringa "**sorgente**", in una seconda stringa "**destinazione**"

```
char src[MAXS+1] ;
char dst[MAXD+1] ;
```





Copia di una stringa

Nella libreria standard C, includendo <**string.h**>, è disponibile la funzione **strcpy**, che effettua la copia di stringhe

Primo parametro: stringa destinazione

Secondo parametro: stringa sorgente

```
#define MAXS  20
#define MAXD  30

char src[MAXS+1];
char dst[MAXD+1];
int i ;
... /* lettura stringa src */
for( i=0 ; src[i] != 0 ; i++ )
   dst[i] = src[i]; /* copia */

dst[i] = '\0'; /* aggiunge terminatore */
```

```
#define MAXS  20
#define MAXD  30

char src[MAXS+1] ;
char dst[MAXD+1] ;
... /* lettura stringa src */
strcpy(dst, src) ;
```

Avvertenze

Nella stringa destinazione vi deve essere un numero sufficiente di locazioni libere

MAXD+1 >= **strlen**(src)+1

Il contenuto precedente della stringa destinazione viene perso

La stringa sorgente non viene modificata

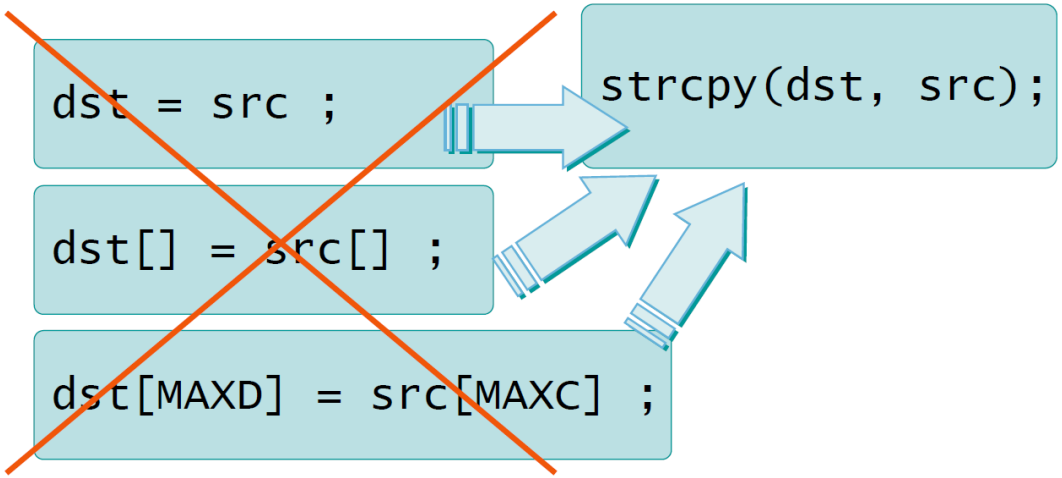
Il terminatore nullo

- Deve essere aggiunto **in coda** a **dst**
- La **strcpy** pensa già autonomamente a farlo

Errore frequente

Per effettuare una copia di stringhe non si può assolutamente utilizzare l'operatore =

Necessario usare **strcpy**



Concatenazione di stringhe

L'operazione di concatenazione corrisponde a creare una nuova stringa composta dai caratteri di una prima stringa, seguiti dai caratteri di una seconda stringa

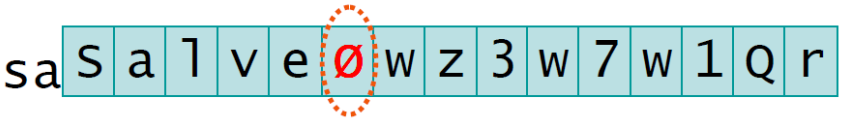
Per maggior semplicità, in C l'operazione di concatenazione scrive il risultato nello stesso vettore della **prima stringa**

Il valore precedente della prima stringa viene così perso

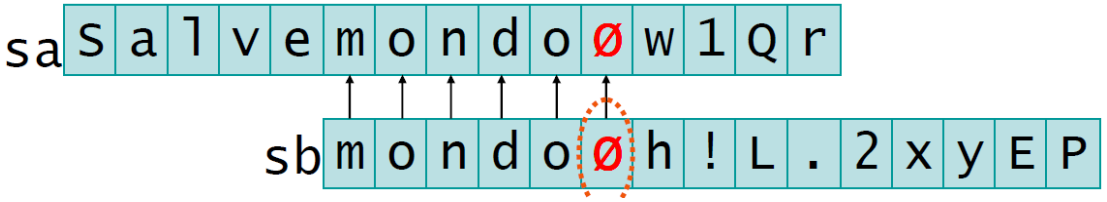
Per memorizzare altrove il risultato, o per non perdere la prima stringa, è possibile ricorrere a stringhe temporanee ed alla funzione **strcpy**

Algoritmo di concatenazione

Trova la fine della prima stringa



Copia la seconda stringa nel vettore della prima, a partire della posizione del terminatore nullo (sovrascrivendolo)




```
#define MAX  20
char sa[MAX];
char sb[MAX];
int la;
int i;
... /* lettura stringhe */
la = strlen(sa);
for( i=0 ; sb[i] != 0 ; i++ )
   sa[la+i] = sb[i]; /* copia */

sa[la+i] = 0; /* terminatore */
```

Algoritmo di concatenazione

Nella libreria standard C, includendo <**string.h**>, è disponibile la funzione **strcat**, che effettua la concatenazione di stringhe

Primo parametro: prima stringa (destinazione)

Secondo parametro: seconda stringa

Esempio:

```
#define MAX  20
char sa[MAX] ;
char sb[MAX] ;
... /* lettura stringhe */
strcat(sa, sb) ;
```

Nella prima stringa vi deve essere un numero sufficiente di locazioni libere

MAX+1 >= **strlen**(sa)+**strlen**(sb)+1

Il contenuto precedente della prima stringa viene perso

La seconda stringa non viene modificata

Il terminatore nullo

Deve essere aggiunto in coda alla prima stringa

La **strcat** pensa già autonomamente a farlo

Per concatenare 3 o più stringhe, occorre farlo due a due:

```
strcat(sa, sb);
strcat(sa, sc);
```

È possibile concatenare anche stringhe costanti

```
strcat(sa, "!");
```

Confronto di stringhe

Il confronto di due stringhe (es.: **sa** e **sb**), mira a determinare se:

- Le due stringhe sono uguali
  - **hanno uguale lunghezza e sono composte dagli stessi caratteri nello stesso ordine**
- Le due stringhe sono diverse
- La stringa **sa** precede la stringa **sb**
  - secondo l'ordine **lessicografico** imposto dal codice ASCII
  - parzialmente compatibile con l'ordine alfabetico
- La stringa **sa** segue la stringa **sb**

Confronto di uguaglianza

Ogni carattere di **sa** deve essere uguale al carattere corrispondente di **sb**

Il terminatore nullo deve essere nella stessa posizione

I caratteri successivi al terminatore vanno ignorati

sa	S	a	l	v	e	Ø	o	4	d	1	Ø	w	1	Q	r
sb	S	a	l	v	e	Ø	h	!	L	.	2	x	y	E	P

Confronto di uguaglianza

```
#define MAX 20
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...
uguali = 1;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ ){
   if(sa[i]!=sb[i])
      uguali = 0;
}
if(sa[i]!=0 || sb[i]!=0)
   uguali = 0;
```

ricerca della condizione **sa**[i]==**sb**[i]

Cicla fino al terminatore di **sa** o di **sb** (il primo che si incontra)

Verifica che tutti i caratteri incontrati siano uguali. Se no, poni a **0** il flag uguali

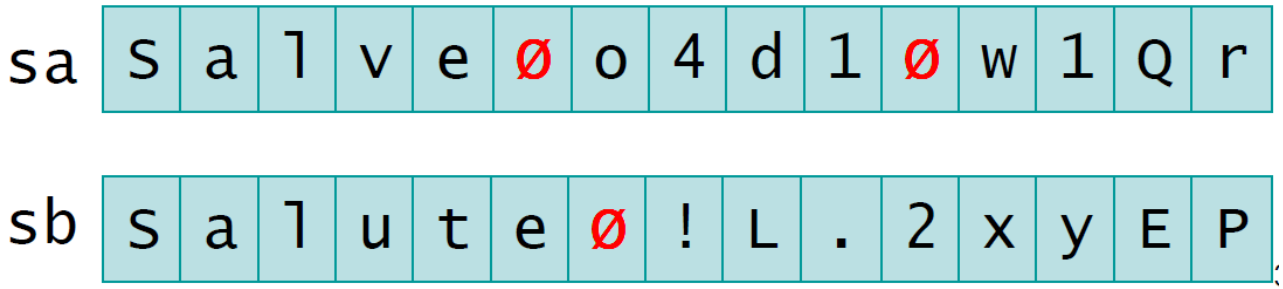
In questo punto sicuramente una delle due stringhe è arrivata al terminatore. Se non lo è anche l'altra, allora non sono uguali!

Confronto di ordine

Verifichiamo se **sa** “è minore di” **sb**. Partiamo con i=0

- Se **sa**[i]<**sb**[i], allora **sa** è minore
- Se **sa**[i]>**sb**[i], allora **sa** non è minore
- Se **sa**[i]=**sb**[i], allora bisogna controllare i caratteri successivi (i++)

Il terminatore nullo conta come “minore” di tutti



Confronto di ordine

```
#define MAX 20
char sa[MAX];
char sb[MAX];
int minore;
int i;
...
minore = 0 ;
for( i=0; sa[i]!=0 && sb[i]!=0 && minore==0; i++ ){
  if(sa[i]<sb[i])
    minore = 1;

  if(sa[i]>sb[i])
    minore = -1;
}

if(minore==0 && sa[i]==0 && sb[i]!=0)
  minore=1 ;

if(minore==1)
  printf("%s e' minore di %s\n", sa, sb ) ;
```

Ricerca di esistenza della condizione sa[i]<sb[i]

Cicla fino al primo terminatore nullo, oppure fino a che non si "scopre" chi è minore.
In altre parole, continua a ciclare solo finché le stringhe "sembrano" uguali

Sicuramente **sa** è minore di **sb**
Flag: minore = 1

Sicuramente **sa** non è minore di **sb**
Flag: minore = -1

Se finora erano uguali, ma **sa** è più corta di **sb**, allora **sa** è minore

Se flag minore==0 continua a ciclare

Confronto di ordine

Nella libreria standard C, includendo <**string.h**>, è disponibile la funzione **strcmp**, che effettua il confronto di stringhe

Primo parametro: prima stringa

Secondo parametro: seconda stringa

Valore restituito:

- **<0** se la prima stringa è minore della seconda
- **==0** se le stringhe sono uguali
- **>0** se la prima stringa è maggiore della seconda

Per ricordare il significato del valore calcolato da **strcmp**, immaginare che la funzione faccia una "sottrazione" tra le due stringhe

sa – sb

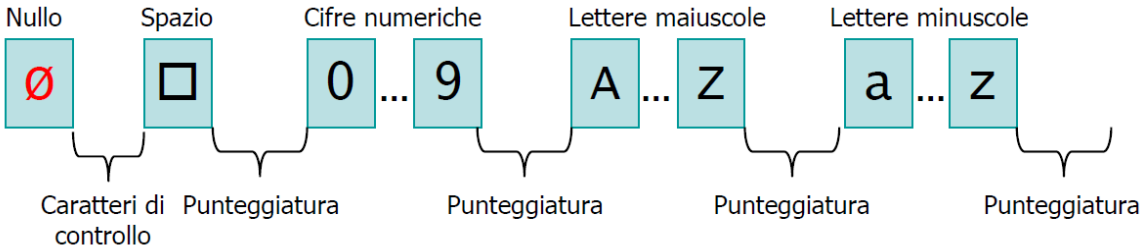
Negativo: **sa** minore
Positivo: **sa** maggiore
Nullo: uguali

```
#define MAX 20
char sa[MAX] ;
char sb[MAX] ;
int ris ;
...
ris = strcmp(sa, sb) ;

if(ris<0)
  printf("%s minore di %s\n", sa, sb);
if(ris==0)
  printf("%s uguale a %s\n", sa, sb);
if(ris>0)
  printf("%s maggiore di %s\n", sa, sb);
```

Ordinamento delle stringhe

La funzione **strcmp** lavora confrontando tra loro i codici ASCII dei caratteri

Il criterio di ordinamento è quindi dato dalla posizione dei caratteri nella tabella ASCII



Ogni lettera maiuscola precede ogni lettera minuscola

Ciao precede **ciao**

Zulu precede **apache**

Gli spazi contano, e precedono le lettere

Qui Quo Qua precede **QuiQuoQua**

I simboli di punteggiatura contano, ma non vi è una regola intuitiva
L'ordinamento che si ottiene è lievemente diverso da quello "standard" alfabetico

Ricerca in una stringa

È possibile concepire diversi tipi di ricerche da compiersi all'interno di una stringa:

1) Verificare se un determinato carattere compare all'interno di una stringa data

2) Determinare se una determinata stringa compare integralmente all'interno di un'altra stringa data, in una posizione arbitraria

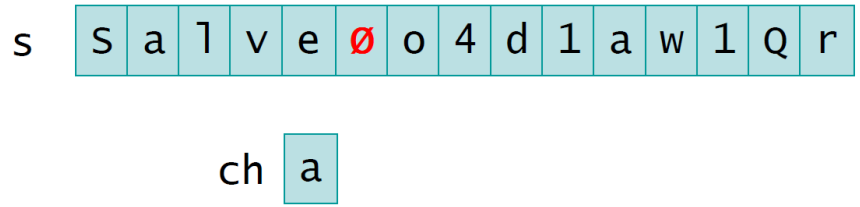
Ricerca di un carattere

Detti:

s una stringa arbitraria

ch un carattere qualsiasi

Determinare se la stringa **s** contiene (una o più volte) il carattere **ch** al suo interno, in qualsiasi posizione




```
#define  MAX 20
char s[MAX];
char ch;
int trovato;
int i;
...
trovato = 0;
for(i=0; s[i]!=0 && trovato==0; i++){
  if(s[i]==ch)
    trovato = 1;
}
```

Ricerca di un carattere

Nella libreria standard C, includendo <**string.h**>, è disponibile la funzione **strchr**, che effettua la ricerca di un carattere

Primo parametro: stringa in cui cercare

Secondo parametro: carattere da cercare

Valore restituito:

!=NULL se il carattere c'è

==NULL se il carattere non c'è

```
#define MAX 10
char s[MAX] ;
char ch ;
...
if(strchr(s, ch)!=NULL)
   printf("%s contiene %c\n", s, ch) ;
```

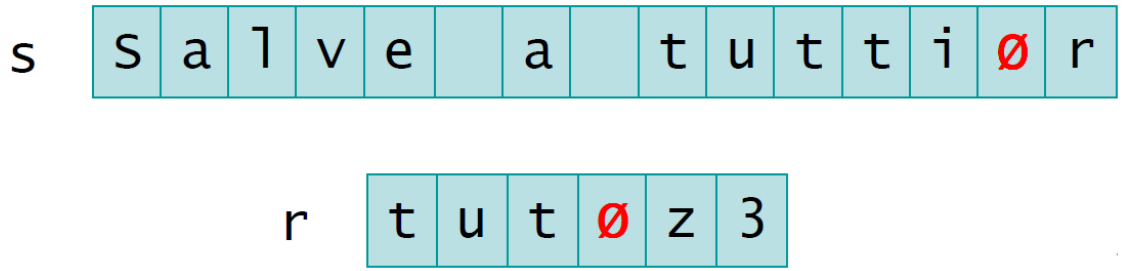
Ricerca di una sotto-stringa

Detti:

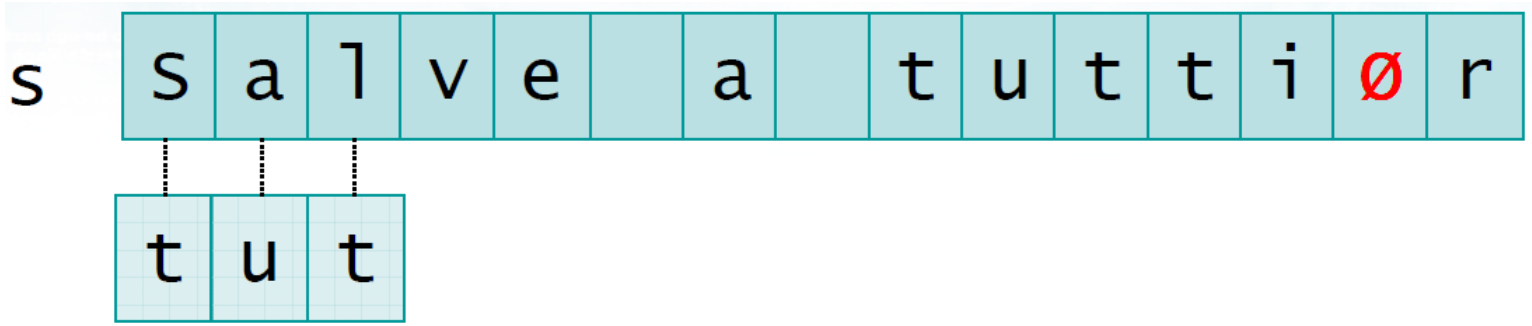
s una stringa arbitraria

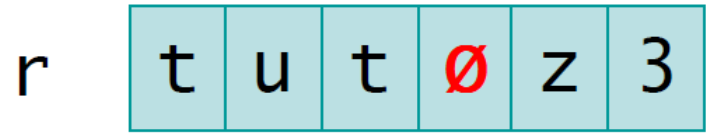
r una stringa da ricercare

Determinare se la stringa **s** contiene (una o più volte) la stringa **r** al suo interno, in qualsiasi posizione

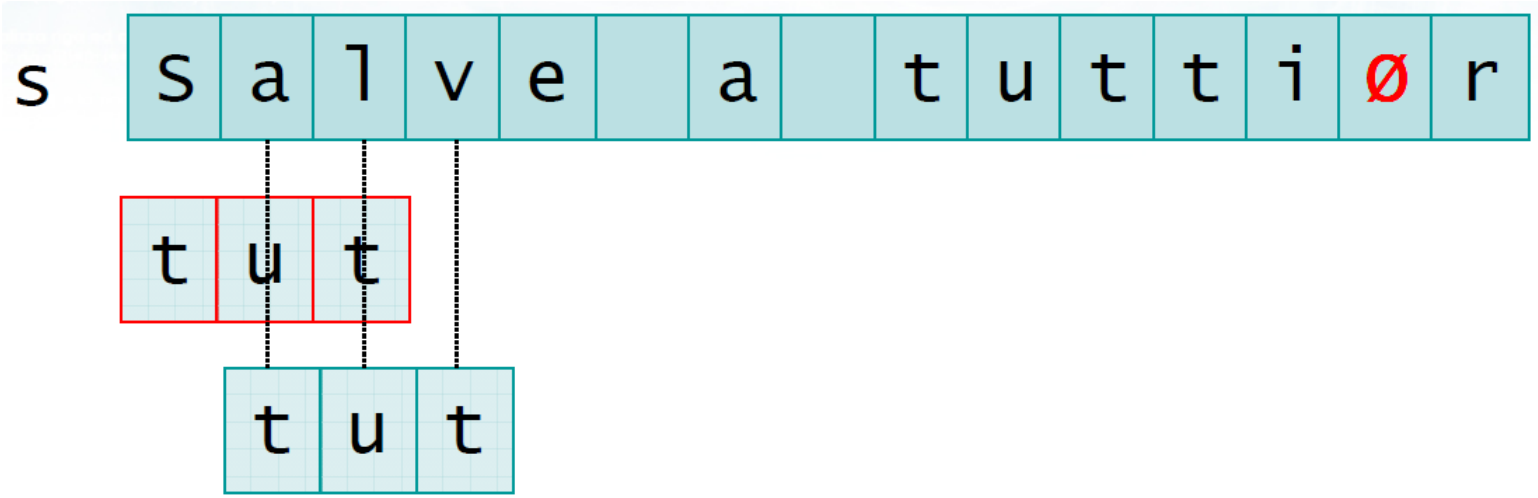


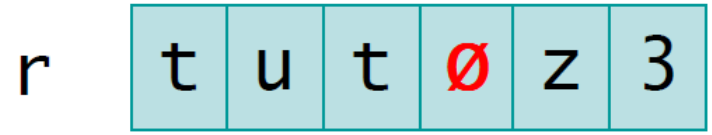
Ricerca di una sotto-stringa



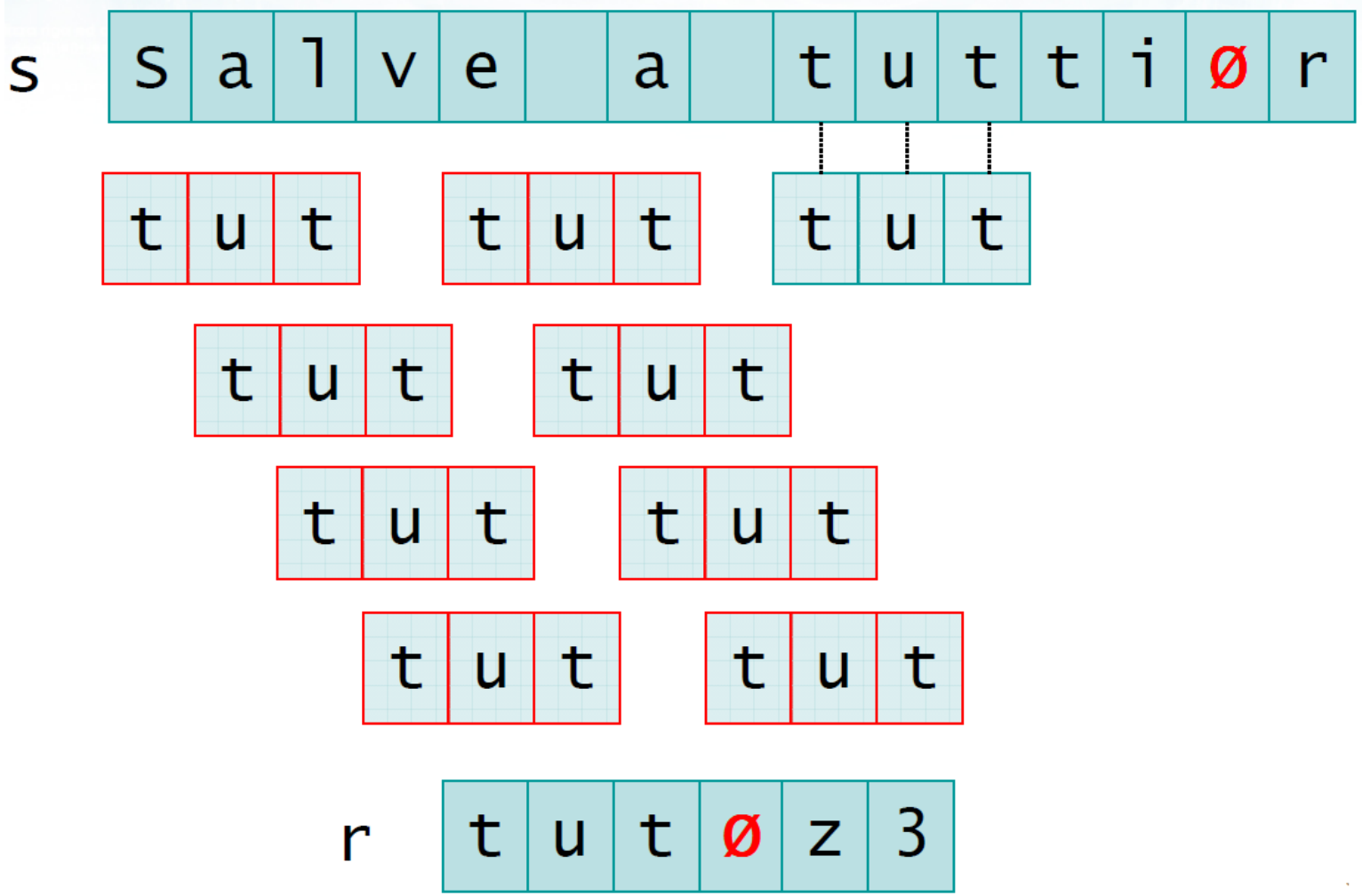


Ricerca di una sotto-stringa

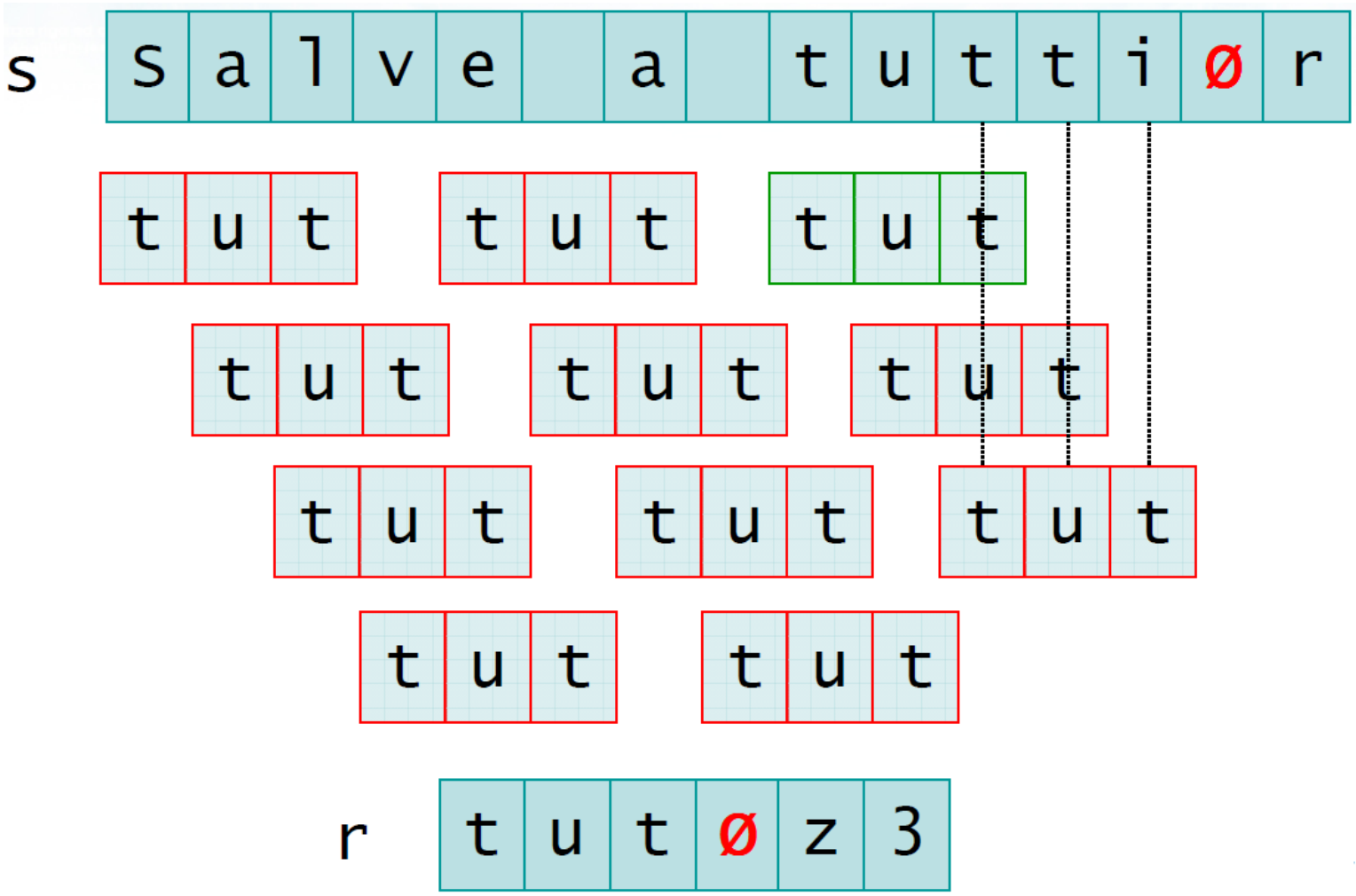




Ricerca di una sotto-stringa



Ricerca di una sotto-stringa



Algoritmo di ricerca

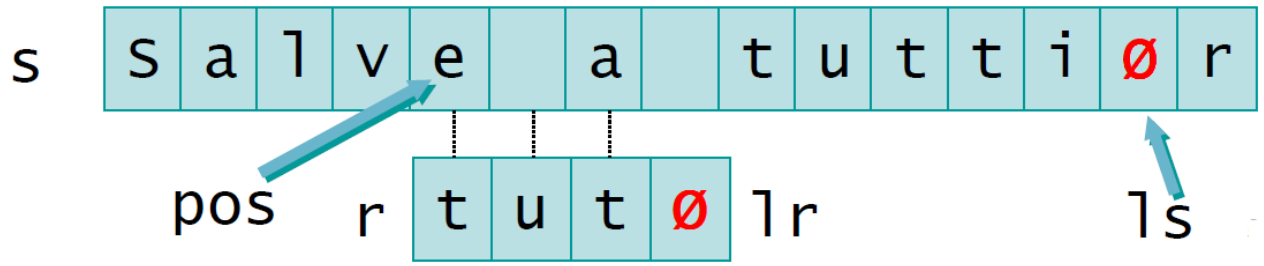
lr = strlen(r);

ls = strlen(s);

trovato = 0

Per ogni posizione possibile di **r** all'interno di **s**:

pos = 0...ls-lr (compresi)



Algoritmo di ricerca

lr = strlen(r);

ls = strlen(s);

trovato = 0

Per ogni posizione possibile di r all'interno di s:

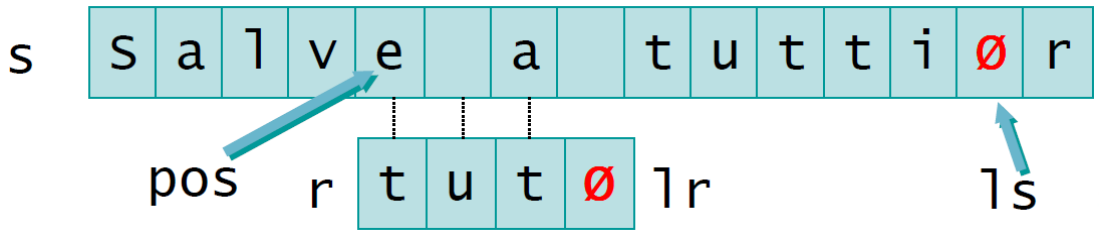
pos = 0...ls-lr (compresi)

Controlla se i caratteri di **r**, tra 0 e lr-1, coincidono con i caratteri di **s**, tra pos e pos+lr-1

Se sì, trovato = 1

```
diversi = 0 ;
for(i=0; i<lr; i++)
   if(r[i] != s[pos+i])
      diversi = 1 ;
```

```
if(diversi==0)
   trovato=1
```



Algoritmo di ricerca

```
#define MAX 20
char s[MAX];
char r[MAX];
int lr, ls, pos;
int i;
int trovato, diversi ;
...
ls = strlen(s);
lr = strlen(r);

trovato = 0 ;
```

```
for(pos=0; pos<=ls-lr; pos++){
  /* confronta r[0...lr-1] con s[pos...pos+lr-1] */
  diversi = 0 ;
  for(i=0; i<lr; i++)
    if(r[i]!=s[pos+i])
      diversi = 1 ;
    if(diversi==0)
      trovato=1 ;
}

if(trovato==1)
  printf("Trovato!\n");
```

Algoritmo di ricerca

Nella libreria standard C, includendo <**string.h**>, è disponibile la funzione **strstr**, che effettua la ricerca di una sottostringa

Primo parametro: stringa in cui cercare
Secondo parametro: sotto-stringa da cercare
Valore restituito:

!=NULL se la sotto-stringa c'è
==NULL se la sotto-stringa non c'è

```
#define MAX 10
char s[MAX] ;
char r[MAX] ;
...
if(strstr(s, r)!=NULL)
  printf("Trovato!\n");
```