Operazioni elementari sulle stringhe

Confronto di stringhe

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

```
const int 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 ;
```

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[M
int ugual
int i ;
...

ugual i = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        ugual i = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    ugual i = 0 ;
```

Flag: ricerca di universalità della condizione `sa[i]==sb[i]`

Confronto di uguaglianza

```
const int 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 ;
```

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

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

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

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

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

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

Confronto di ordine (1/2)

```
const int 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 ;
}
```

Confronto di ordine (2/2)

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

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

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore
{
    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)
    ...
```

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

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore
    if(sa[i]>sb
        minore
}

if(minore==0 &&
    minore=1 ;

if(minore==1)
    ...
```

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.

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0
            && mino
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}

                & sa
if(minore==1)
    ...
```

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

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

Se flag `minore==0` continua a ciclare

Commenti

```
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)
    ...
```

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

La funzione `strcmp`

- 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

Confronti vari

```
const int 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);
```

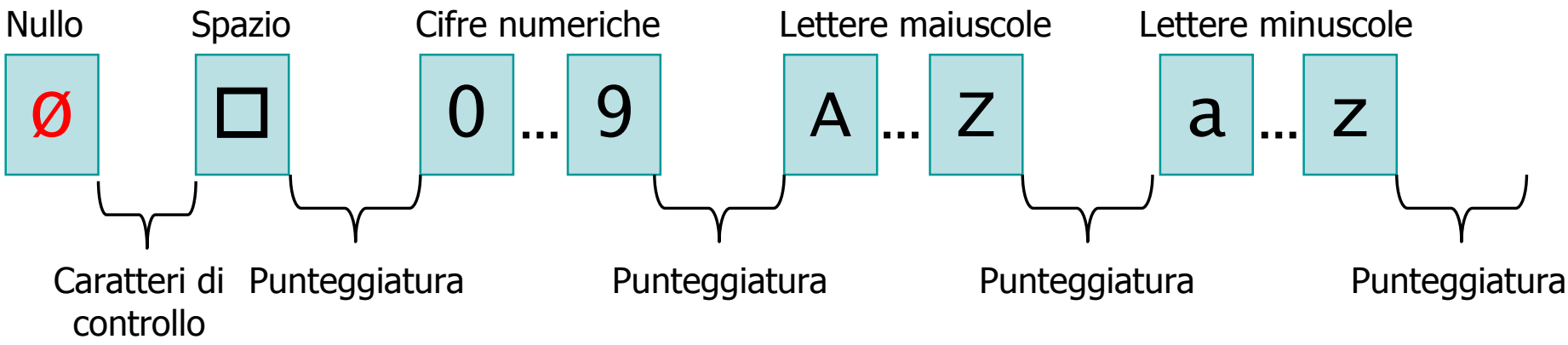
Suggerimento

- 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

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int ris ;
...
ris = strcmp(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



Conseguenze

- 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

Operazioni elementari sulle stringhe

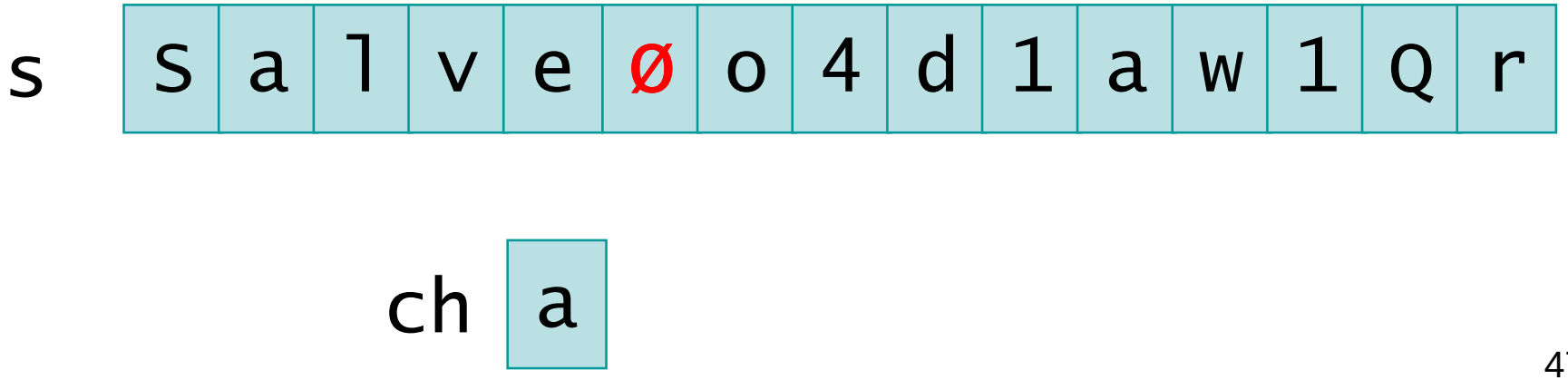
Ricerca di sotto-stringhe

Ricerca in una stringa

- È possibile concepire diversi tipi di ricerche da compiersi all'interno di una stringa:
  - Determinare se un determinato carattere compare all'interno di una stringa data
  - Determinare se una determinata stringa compare integralmente all'interno di un'altra stringa data, in una posizione arbitraria

Ricerca di un carattere (1/2)

- 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



Ricerca di un carattere (2/2)

```
const int 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 ;
}
```

La funzione `strchr` (1/2)

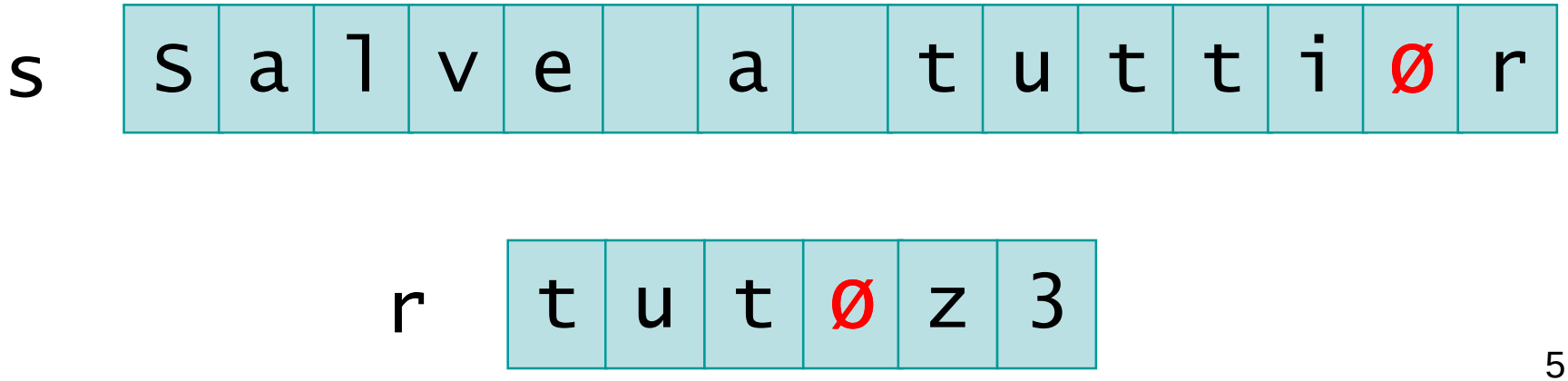
- Nella libreria standard C, includendo `<string.h>`, è disponibile la funzione `strchr`, che effettua la ricerca di un carattere
  - Primo parametro: stringa in cui cercare
  - Secondo parametro: carattere da cercare
  - Valore restituito:
    - `!=NULL` se il carattere c'è
    - `==NULL` se il carattere non c'è

La funzione `strchr` (2/2)

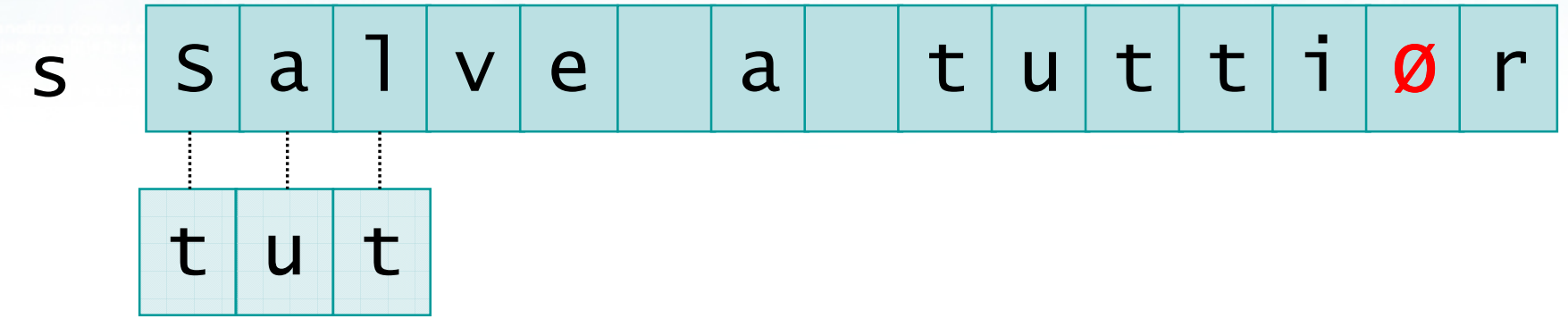
```
const int MAX = 20 ;
char s[MAX] ;
char ch ;

...

if(strchr(s, ch)!=NULL)
   printf("%s contiene %c\n", s, ch) ;
```

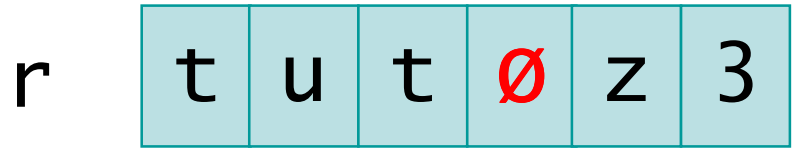
Ricerca di una sotto-stringa

- 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

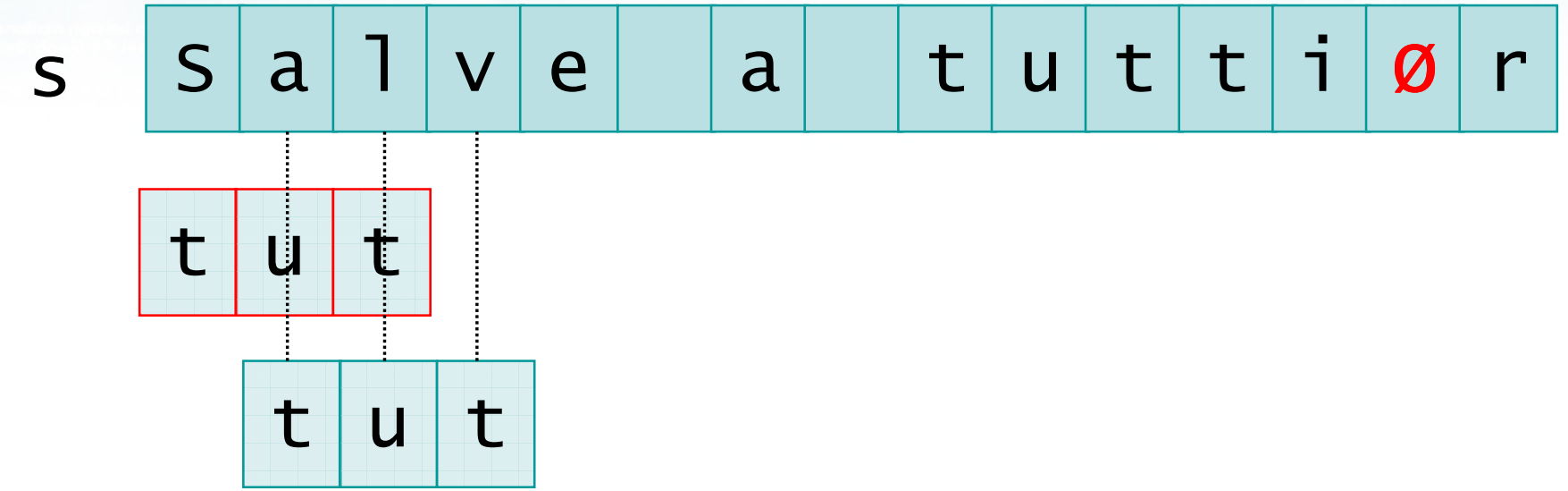


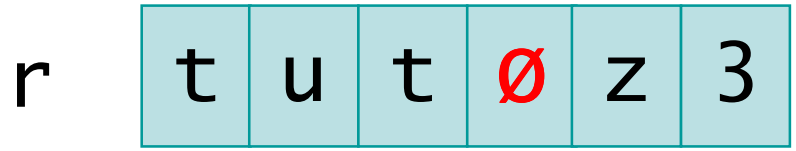
Esempio



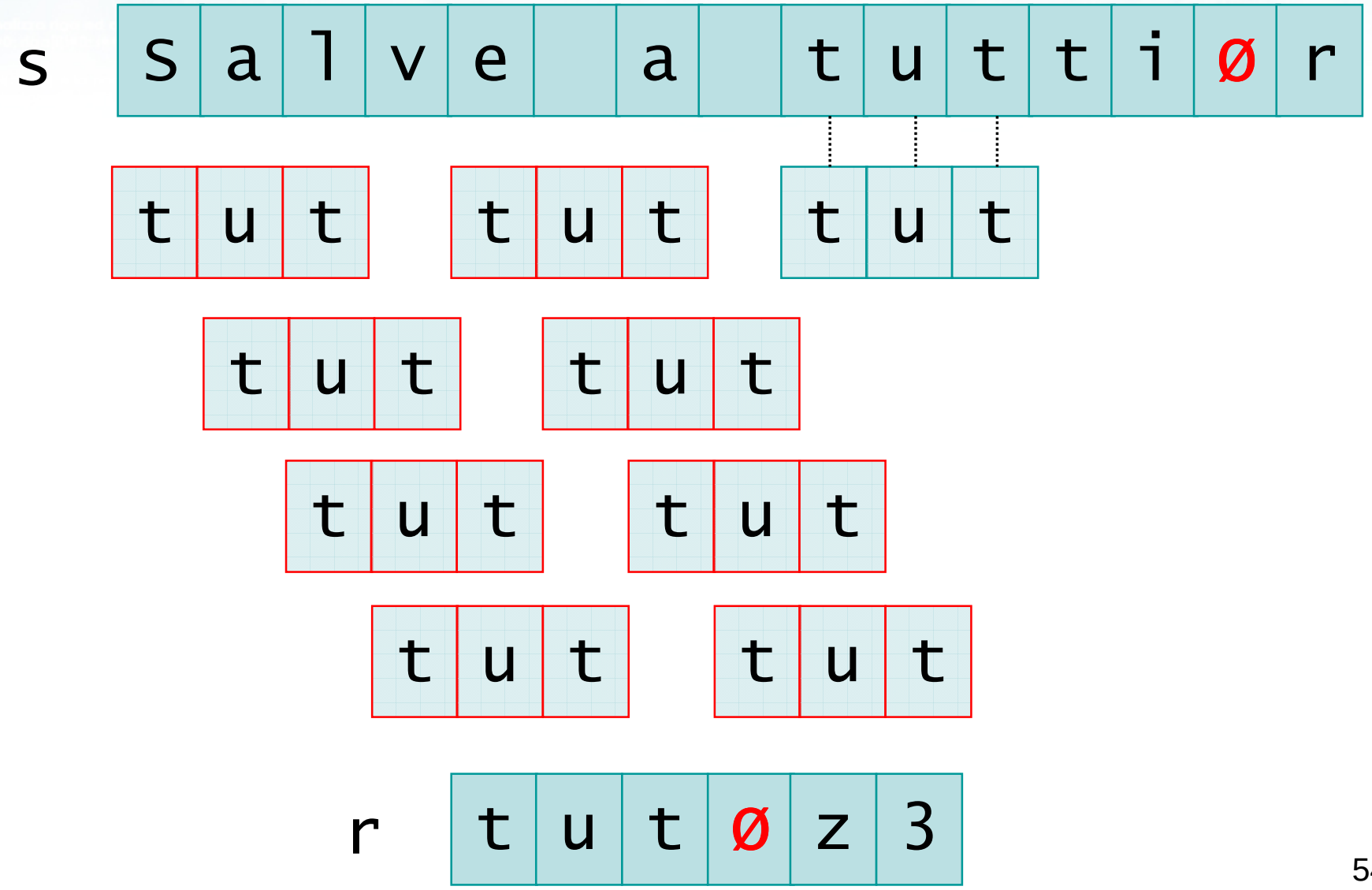


Esempio

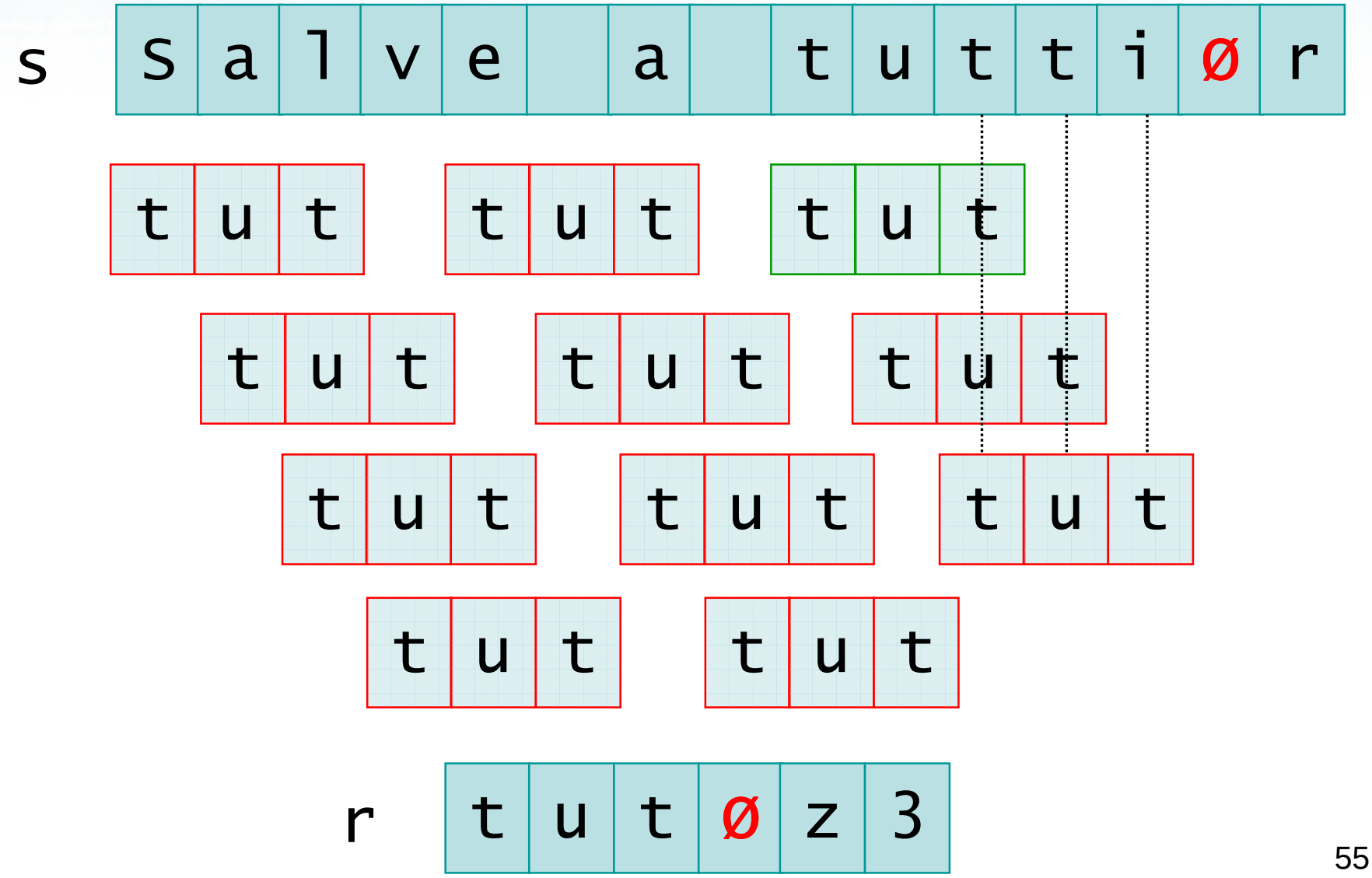




Esempio

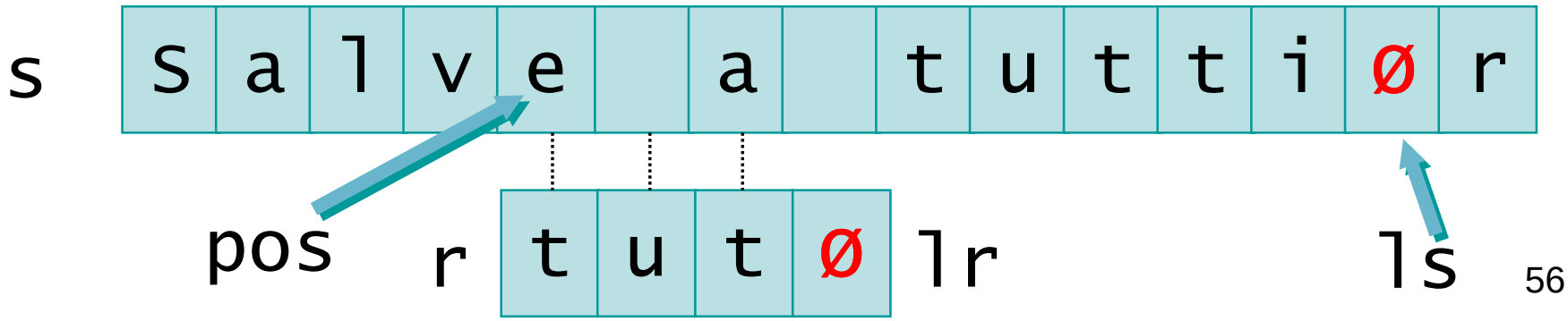


Esempio


s
S a l v e a t u t t i ø r
t u t t u t t u t
t u t t u t t u t
t u t t u t t u t
t u t t u t
r
t u t ø z 3

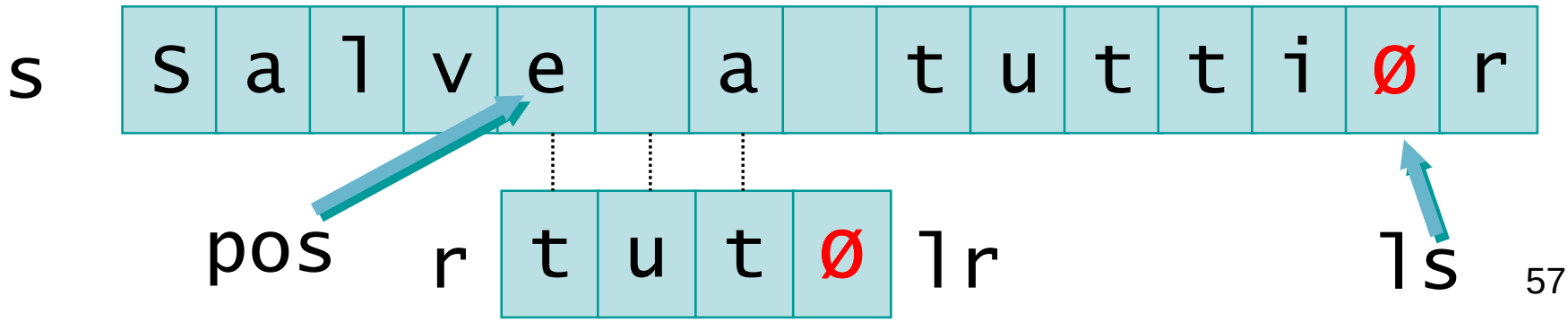
Algoritmo di ricerca

- `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

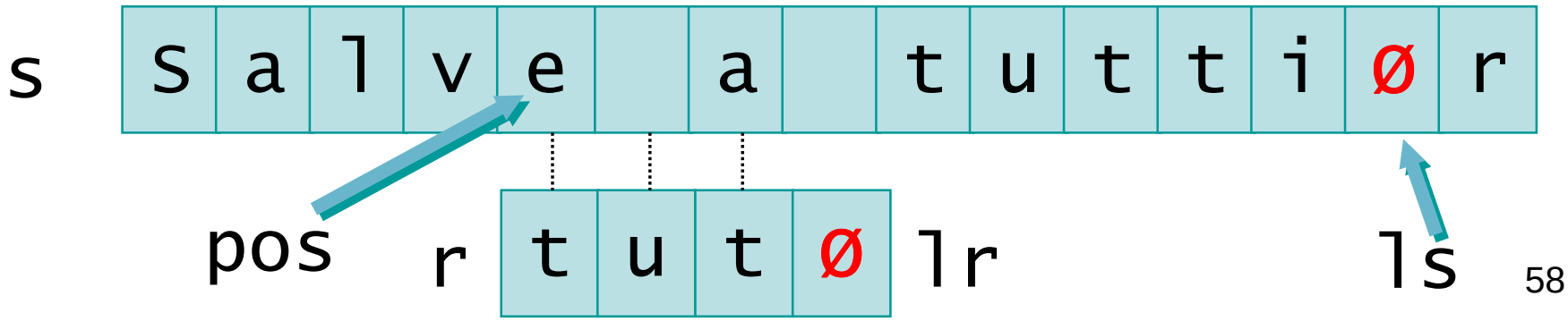
- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione possibile di `r` all'interno di `s`: `pos = 0...ls-lr` (compresi)
  - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`, tra `pos` e `pos+lr-1`
  - Se sì, `trovato = 1`



Algoritmo di ricerca

- `lr = strlen(r);`
- `trovato = 0`
- Per ogni posizione po
  `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`

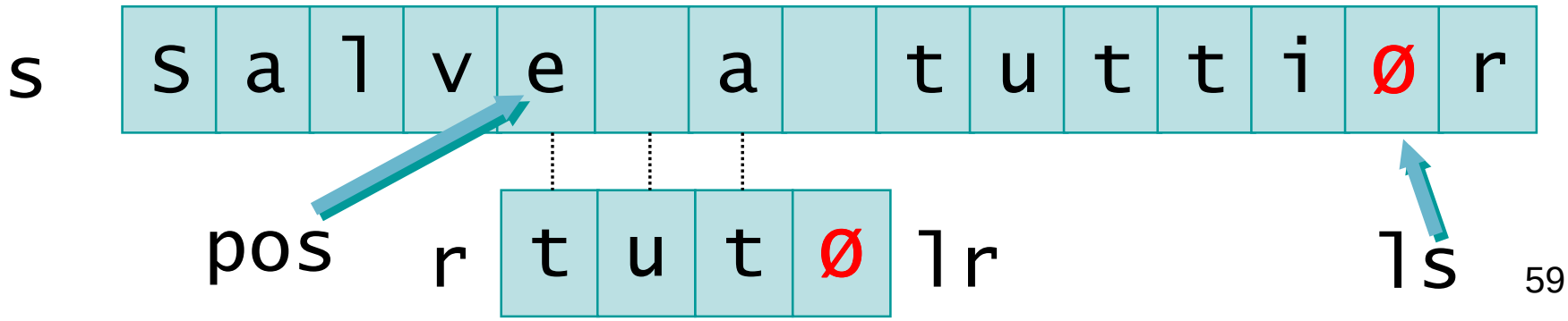
```
diversi = 0 ;
for(i=0; i<lr; i++)
    if(r[i]!=s[pos+i])
        diversi = 1 ;
```



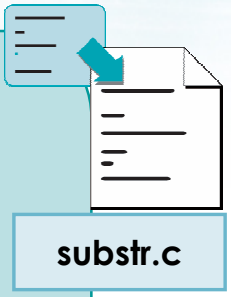
Algoritmo di ricerca

- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione po
  `pos = 0...ls-lr`
  - Controlla se i caratt
    coincidono con i ca
    `pos+lr-1`
  - Se sì, `trovato = 1`

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

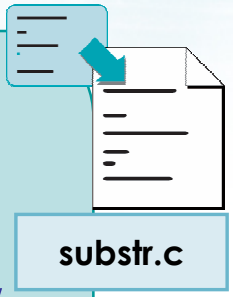


Ricerca di una sotto-stringa (1/2)




```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;
int lr, ls, pos ;
int i ;
int trovato, diversi ;

...

ls = strlen(s);
lr = strlen(r);
```

Ricerca di una sotto-stringa (2/2)




```
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");
```

La funzione `strstr` (1/2)

- 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

La funzione `strstr` (2/2)

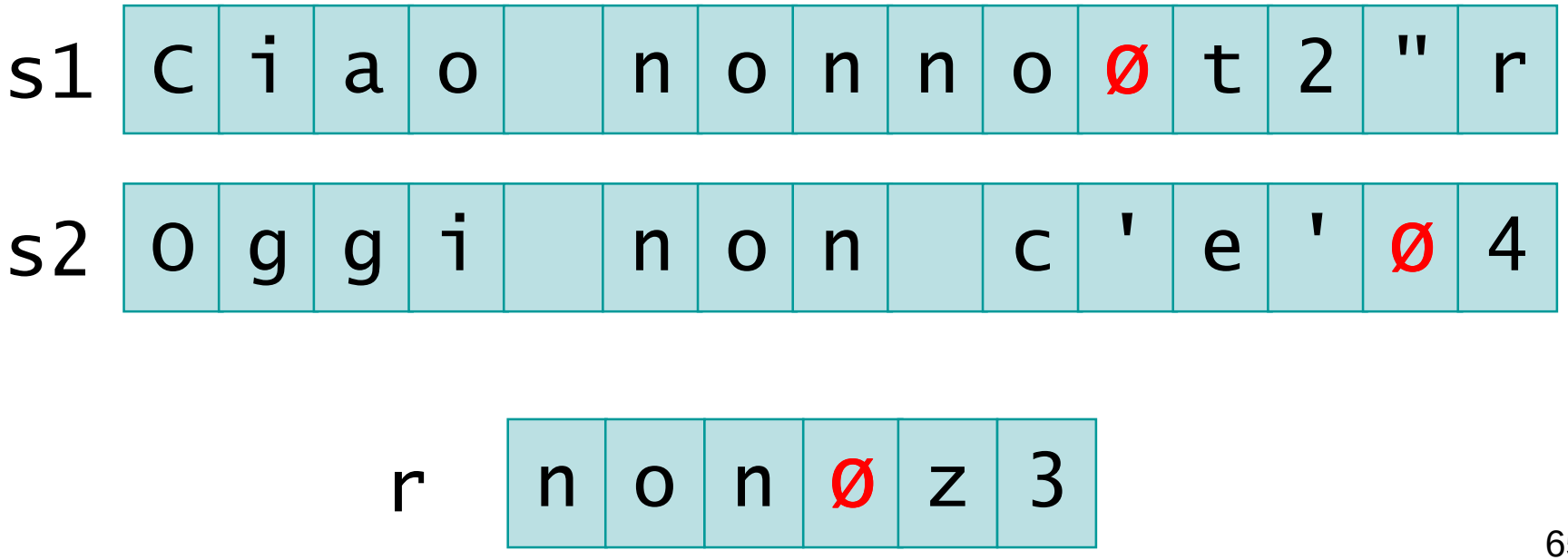
```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;

...

if(strstr(s, r)!=NULL)
    printf("Trovato!\n");
```

Operazioni elementari sulle stringhe

Ricerca di parole

Ricerca di parole

- Talvolta non interessa trovare una qualsiasi sotto-stringa, ma solamente verificare se una **parola completa** è presente in una stringa



Definizioni (1/2)

- **Lettera**: carattere ASCII facente parte dell'alfabeto maiuscolo ('A'...'Z') o minuscolo ('a'...'z')
- **Parola**: insieme consecutivo di lettere, separato da altre parole mediante spazi, numeri o simboli di punteggiatura

Definizioni (2/2)

- **Inizio di parola**: lettera, prima della quale non vi è un'altra lettera
  - Non vi è un altro carattere (inizio stringa)
  - Vi è un altro carattere, ma non è una lettera
- **Fine di parola**: lettera, dopo la quale non vi è un'altra lettera
  - Non vi è un altro carattere (fine stringa)
  - Vi è un altro carattere, ma non è una lettera

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra `0` e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra `0` e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```

```
if( pos == 0 ||
    !( (s[pos-1]>='a' && s[pos-1]<='z') ||
       (s[pos-1]>='A' && s[pos-1]<='Z') )
  )
```

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in inizio di parola
    - Controlla se i caratteri di `r`, tra `0` e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in
    inizio di parola

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```

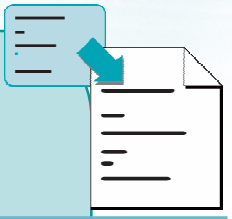
```
if( pos == ls-lr ||
    !( (s[pos+lr]>='a' && s[pos+lr]<='z') ||
       (s[pos+lr]>='A' && s[pos+lr]<='Z') )
  )
```

incidono

ola

    - Se entrambi i controlli sono ok, allora `trovato=1`

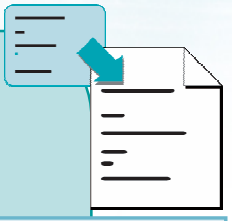
Ricerca di una parola (1/2)



parola.c

```
trovato = 0 ;
for(pos=0; pos<=ls-lr; pos++)
{
    if( pos==0 ||
        !( (s[pos-1]>='a' &&
            s[pos-1]<='z') ||
           (s[pos-1]>='A' &&
            s[pos-1]<='Z') ) )
    {
        diversi = 0 ;
        for(i=0; i<lr; i++)
            if(r[i]!=s[pos+i])
                diversi = 1 ;
```

Ricerca di una parola (2/2)



parola.c

```
            if( diversi==0 &&
                ( pos == ls-lr ||
                  !( (s[pos+lr]>='a' &&
                      s[pos+lr]<='z') ||
                     (s[pos+lr]>='A' &&
                      s[pos+lr]<='Z') )
                )  )
            {
                trovato=1 ;
            }
        }
}
```

La funzione `strparola`

- Nella libreria standard C non esiste alcuna funzione che svolga automaticamente la ricerca di una parola intera!!!
- Occorre identificare, ogni volta, se il compito da svolgere è riconducibile ad una o più funzioni di libreria
- Eventualmente si combinano tra loro più funzioni di libreria diverse
- In alcuni casi occorre però ricorrere all'analisi carattere per carattere