

**Unità didattica:** string matching su file

[2-AC]

**Titolo:** Ricerca di una stringa su un file testo

Argomenti trattati:

- ✓ Algoritmo di ricerca diretta di una stringa in un file.
- ✓ Algoritmo di ricerca diretta di un pattern in un testo dove il pattern è definito in input, il testo è memorizzato in un file e parzialmente caricato in memoria mediante un array-buffer.

**Prerequisiti richiesti:** file in C, puntatori, allocazione dinamica

## 07\_02.2

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la diritta via era smarrita

accede al file per ogni carattere (lento!)

**Laboratorio:** scrivere una *function* **C** per la ricerca diretta di un *pattern* in un testo dove:

- ✓ il testo è memorizzato in un file;
- ✓ la lettura del file avviene mediante un array-buffer;
- ✓ il **pattern** è definito in input.

**1...** **pattern tramite array di char ...**

```
char stringa[81]
```

dichiara la variabile **stringa** come array di **char**.  
La stringa può essere definita dinamicamente in input, ma è vincolata a non superare il massimo numero di caratteri dichiarati (81).

Come creare in fase di esecuzione una stringa di cui non si conosce a priori la lunghezza e gestirla tramite puntatore?

## 2... pattern tramite puntatore ad un char ...

```
char *p_stringa
```

dichiara la variabile **p\_stringa** come puntatore ad un **char**, ma il valore del puntatore è indefinito, cioè il puntatore non punta a nulla !!!

```
char *p_string;  
gets(p_string);
```

Si è già visto che  
è **sbagliato** in C!!!

```
char *p_stringa = "Stringa costante";
```

```
char *p_stringa;  
p_stringa = "Stringa costante";
```

dichiara il medesimo puntatore di prima, ma ora è inizializzato, in fase di compilazione, a puntare sempre alla stringa **"Stringa costante"** che non può essere modificata.

equivalenti

# Esempio: una stringa costante non può essere modificata!

The screenshot shows the Microsoft Visual C++ IDE with a project named "StringPointer1". The source file "StringPointer1.c" contains the following code:

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

void main()
{
    char *stringa = 'ciao';
    puts(stringa);
    *stringa = 'm';
    puts(stringa);
}
```

A red box highlights the line `*stringa = 'm';`, with a red arrow pointing to it from a text box that says "non funziona!".

The output window shows the program's execution, displaying "ciao" and "Press any key to continue\_".

The console window shows the build output:

```
Configuration: StringPointer1 - Win32 D
Linking...
StringPointer1.exe - 0 error(s), 0 warning(s)
```

An error dialog box titled "StringPointer1.exe" is displayed, indicating a runtime error:

**StringPointer1.exe**

Si è verificato un errore in StringPointer1.exe.  
L'applicazione verrà chiusa.

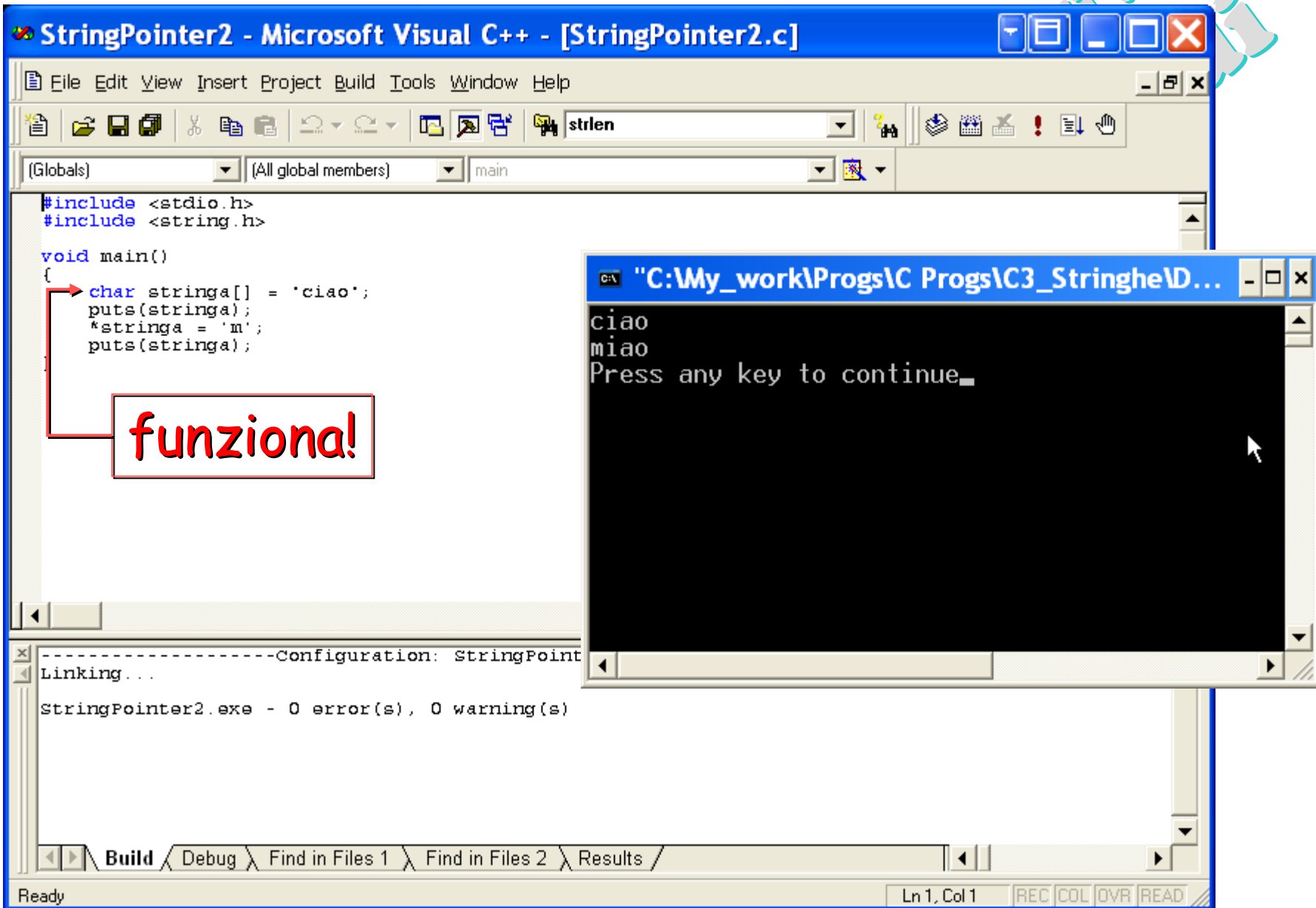
Potrebbe essersi verificata la perdita dei dati su cui si stava lavorando.

**Segnalazione del problema a Microsoft**  
È stata creata una segnalazione errori che è possibile inviare. Il contenuto della segnalazione sarà riservato e anonimo.

Per visualizzare i dati contenuti nella segnalazione errori [fare clic qui](#)

Buttons: Debug, Invia segnalazione errori, Non inviare

# Esempio: una stringa come array può essere modificata!



The screenshot displays the Microsoft Visual C++ IDE with the file `StringPointer2.c` open. The code defines a `main` function that declares a character array `stringa` initialized to `'ciao'`, prints it, changes its value to `'m'`, and prints it again. A red arrow points from the word **funziona!** to the line `*stringa = 'm';`. The console window shows the output: `ciao`, `miao`, and `Press any key to continue.` The status bar at the bottom indicates `StringPointer2.exe - 0 error(s), 0 warning(s)`.

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

void main()
{
    char stringa[] = 'ciao';
    puts(stringa);
    *stringa = 'm';
    puts(stringa);
}
```

**funziona!**

C:\My\_work\Progs\C Progs\C3\_Stringhe\ID...  
ciao  
miao  
Press any key to continue.

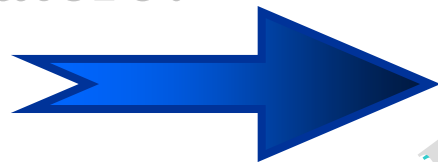
-----Configuration: StringPoint  
Linking...  
StringPointer2.exe - 0 error(s), 0 warning(s)

Build Debug Find in Files 1 Find in Files 2 Results

Ready Ln 1, Col 1 REC COL OVR READ

3...

Come creare in fase di esecuzione una stringa di cui non si conosce a priori la lunghezza e gestirla tramite puntatore?



**Allocazione dinamica**

- ▶ stabilire la lunghezza della stringa (**len\_str**)
- ▶ allocare dinamicamente lo spazio necessario per la stringa
- ▶ avere un puntatore per accedere alla stringa (**p\_str**)

**in C**

**malloc(...)**

# Esempio: *uso di* malloc(...)

```
#include <stdio.h>
#include <stdlib.h>
void main()
{short len_str; char *p_str, ch;
  puts("numero caratteri della stringa ");
  fflush(stdin);
  scanf("%d",&len_str); len_str++;
  p_str = malloc(len_str);
  if (p_str == NULL)
    {puts("memoria insufficiente");
     exit(1);
    }
  printf("Immetti la stringa, poi RETURN: ");
  fflush(stdin); gets(p_str);
  printf("\nstringa con punt=%s\n",p_str);
}
```

per inserire il carattere finale '\0'

svuota il buffer di **stdin**  
(unità standard di input)



## Esempio: provare al posto di fflush

```
#include <stdio.h>
#include <stdlib.h>
void main()
{short len_str; char *p_str, ch;
  puts("lunghezza stringa");
  scanf("%d",&len_str);

  ch=(char)getchar();
  printf("ch=%c decimale=%d",ch,ch);

  len_str++; /* per il char finale '\0' */
  p_str = malloc(len_str);
  if (p_str == NULL)
  {puts("memoria insufficiente");
   exit(1);
  }
  printf("Immetti la stringa, poi RETURN: ");
  gets(p_str);
  printf("\nstringa con punt=%s\n",p_str);
}
```

talvolta caratteri strani!!!



**Laboratorio:** scrivere una *function* *C* per la ricerca diretta di un *pattern* in un testo dove:

- ✓ il testo è memorizzato in un file;
- ✓ la lettura del file avviene mediante un array-buffer;
- ✓ il pattern è definito in input (allocato dinamicamente).

## idea algoritmo

finché non è finito il file

- legge dal file i caratteri nel buffer;
- cerca il pattern nel buffer;
- esamina l'eventualità che il pattern sia diviso su due buffer consecutivi

```
/*String matching: main */
```

```
#include ...
```

```
short cerca_patt_in_buff(int , char *, FILE *);
```

```
void main( )
```

```
{char *p_patt, *p_buff, nomefile[60]; FILE *fp;
```

```
int len_patt; short quanti;
```

```
puts("lunghezza pattern="); scanf("%d",&len_patt);
```

```
len_patt++; p_patt = malloc(len_patt);
```

```
if (p_patt == NULL)
```

```
    {puts("memoria insufficiente"); exit(1);};
```

```
else {printf("Immetti pattern, poi RETURN=");
```

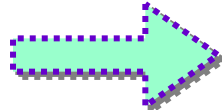
```
fflush(stdin); gets(p_patt);
```

```
puts("nome del file testo"); gets(nomefile);
```

```
if ((fp=fopen(nomefile,"r"))==NULL)
```

```
    {puts("Errore apertura file"); exit(1);};
```

```
else
```



```
{quanti=cerca_patt_in_buff(len_patt-1, p_patt, fp);
```

```
fclose(fp);
```

```
printf("\nnumero pattern trovati = %d\n",quanti);}
```

```
}} ...
```

```

#define BUFSIZE 30 ...
/* --- procedura di string matching --- */
short cerca_patt_in_buff (int len_patt, char *p_patt, FILE *fp)
{char *ch, *ultimi_jc, buffer[BUFSIZE], *p_null="";
 short num_trovati, j; num_trovati=0; j=0;
 while (!feof(fp))
     {*buffer=*p_null; fgets(buffer,BUFSIZE,fp);
      ch = strstr(buffer, p_patt+j);
      while (ch!=0) /* 0 sta per null! */
          {num_trovati++;
           ch = strstr(ch+len_patt-j, p_patt); j=0;}
 /* eventuale pattern disposto su due buffer successivi
    calcola l'eventuale numero j di caratteri finali del buffer
    che sono uguali a quelli iniziali del pattern */
    j = len_patt-1; ultimi_jc = buffer+BUFSIZE-1-j;
    while (strncmp(p_patt,ultimi_jc,j) !=0 && j>0)
        {j--; ultimi_jc = buffer+BUFSIZE-1-j;}
     }
 return num_trovati;
}

```

**Laboratorio:** come funziona `cerca_patt_in_buff`?

buffer

si supponga per il momento

`*p_patt="ita"`

BUFSIZE

N e l m e z z o d e l c a m m i n d i n o s t r

inizialmente  $j=0$

`ch=strstr(buffer, p_patt+j);`

$ch=0$ , quindi legge un altro buffer

a v i t a m i r i t r o v a i p e r u n a s e

ancora  $j=0$

`ch=strstr(buffer, p_patt+j);`

$ch$

cerca un'altra occorrenza di **patt**

a partire da

```
while (ch!=0)
{
    num_trovati++;
    ch = strstr(ch+len_patt-j, p_patt);
    j=0;
}
```

a v i t a m i r i t r o v a i p e r u n a s e

(in questo caso non ne trova!)

continuando, si supponga

`*p_patt="selva"`

eventuale pattern disposto a cavallo di 2 buffer successivi

a	v	i	t	a		m	i		r	i	t	r	o	v	a	i		p	e	r		u	n	a		s	e	'\0'
---	---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	------

```
j = len_patt-1;
```

```
ultimi_jc = buffer+BUFSIZE-1-j;
```

```
while (strncmp(p_patt, ultimi_jc, j) !=0 && j>0)
{
    j--;
    ultimi_jc = buffer+BUFSIZE-1-j;
}
```

j=4

ultimi\_jc

confronta i primi j(=4) caratteri tra il pattern ed il testo che inizia da `ultimi_jc`

buffer+BUFSIZE-1

A. Rizzardo

finché le due stringhe sono diverse e `j>0` ripete

```
j--;
ultimi_jc = buffer+BUFSIZE-1-j;
```

j=3

ultimi\_jc

a	v	i	t	a		m	i		r	i	t	r	o	v	a	i		p	e	r		u	n	a		s	e	'\0'
---	---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	------

```
*p_patt="selva"
```

**Laboratorio:** come funziona `cerca_patt_in_buff?`

P2\_07\_02.15

eventuale pattern disposto a cavallo di 2 buffer successivi

finché le due stringhe sono diverse e  $j > 0$  ripete

```
j--;  
ultimi_jc = buffer+BUFSIZE-1-j;
```

$j=2$

ultimi\_jc

a	v	i	t	a		m	i		r	i	t	r	o	v	a	i		p	e	r		u	n	a		s	e		0
---	---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	--	---

```
while (strncmp(p_patt, ultimi_jc, j) != 0 && j > 0)  
{  
  }  
}
```

ora il confronto le trova uguali! Si esce dal while

legge un altro buffer

`buffer+BUFSIZE-1`

l	v	a		o	s	c	u	r	a		c	h	e		l	a		d	i	r	i	t	t	a		v	i	a		0
---	---	---	--	---	---	---	---	---	---	--	---	---	---	--	---	---	--	---	---	---	---	---	---	---	--	---	---	---	--	---

ora  $j=2$

```
ch=strstr(buffer, p_patt+j);
```

cerca e trova `*(p_patt+2)="lva"` in `ch`

```
while (ch != 0)  
{  
  num_trovati++;  
  ch = strstr(ch+len_patt-j, p_patt);  
  j=0;  
}
```

incrementa `num_trovati`; continua la ricerca avanzando di `len_patt` (=3) caratteri; `j` ritorna a 0 ...

(prof. Rizzardi)

# Esercizio

1

Scrivere *function C* per la ricerca diretta di tutte le occorrenze di un *pattern* in un testo dove:

- ✓ il testo è memorizzato in un file;
- ✓ la lettura del file avviene mediante un array-buffer;
- ✓ il pattern è definito in input.