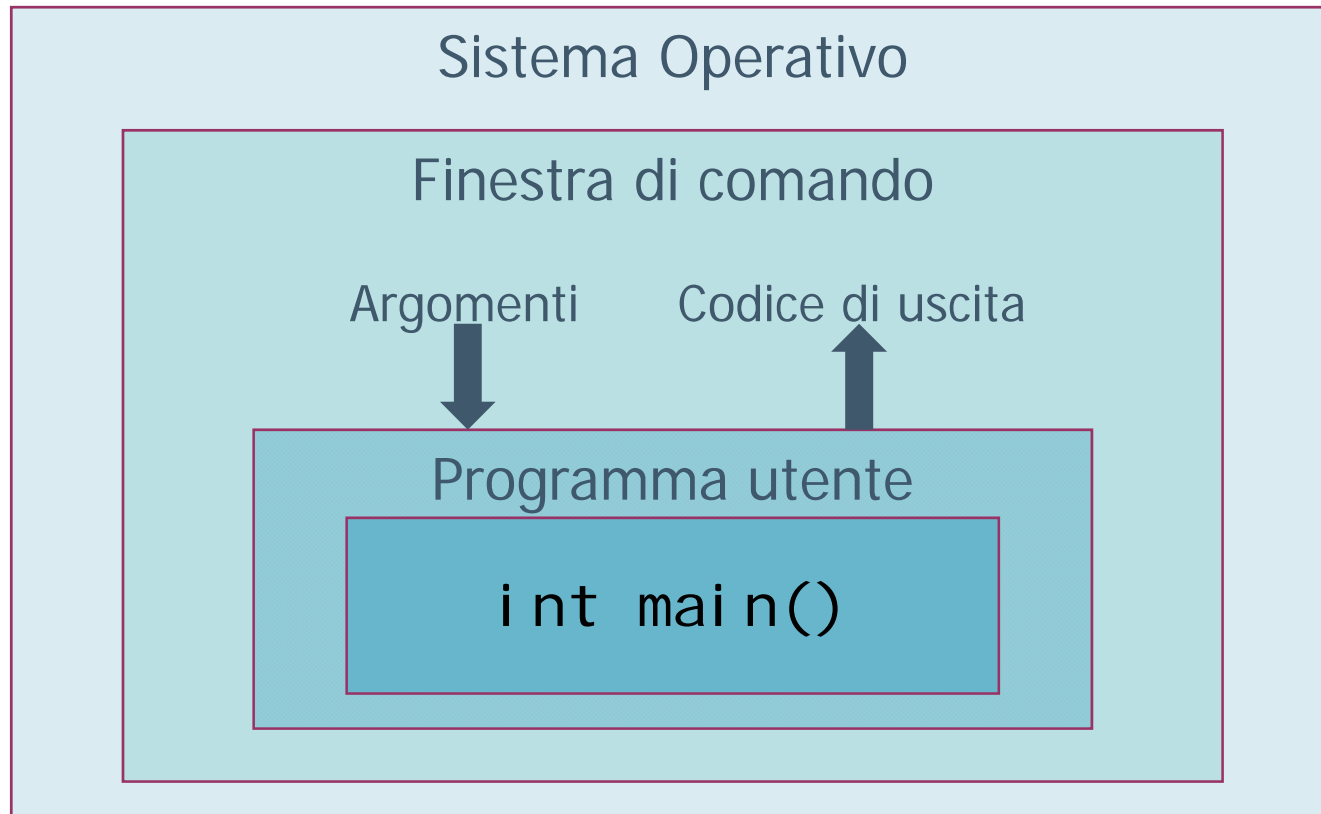


LINEA COMANDO



Argomenti sulla linea di comando

Il modello "console"



Argomenti sulla linea di comando

- In C, è possibile passare informazioni ad un programma specificando degli argomenti sulla linea di comando

- Esempio:

```
C:\> myprog <arg1> <arg2> ... <argN>
```

- Comuni in molti comandi "interattivi"

- Esempio: MS-DOS

```
C:\> copy file1.txt dest.txt
```

- Automaticamente memorizzati negli argomenti del `main()`

Argomenti del `main()`

- Prototipo:

```
main (int argc, char* argv[])
```

- `argc`: Numero di argomenti specificati
 - Esiste sempre almeno un argomento (il nome del programma)
- `argv`: Vettore di stringhe
 - `argv[0]` = primo argomento
 - `argv[i]` = generico argomento
 - `argv[argc-1]` = ultimo argomento

Esempi

```
C: \progr>quadrato
```

- Numero argomenti = 1
- (il nome del programma)

```
C: \progr>quadrato 5
```

- Numero argomenti = 2
- Argomento 2 = "5"

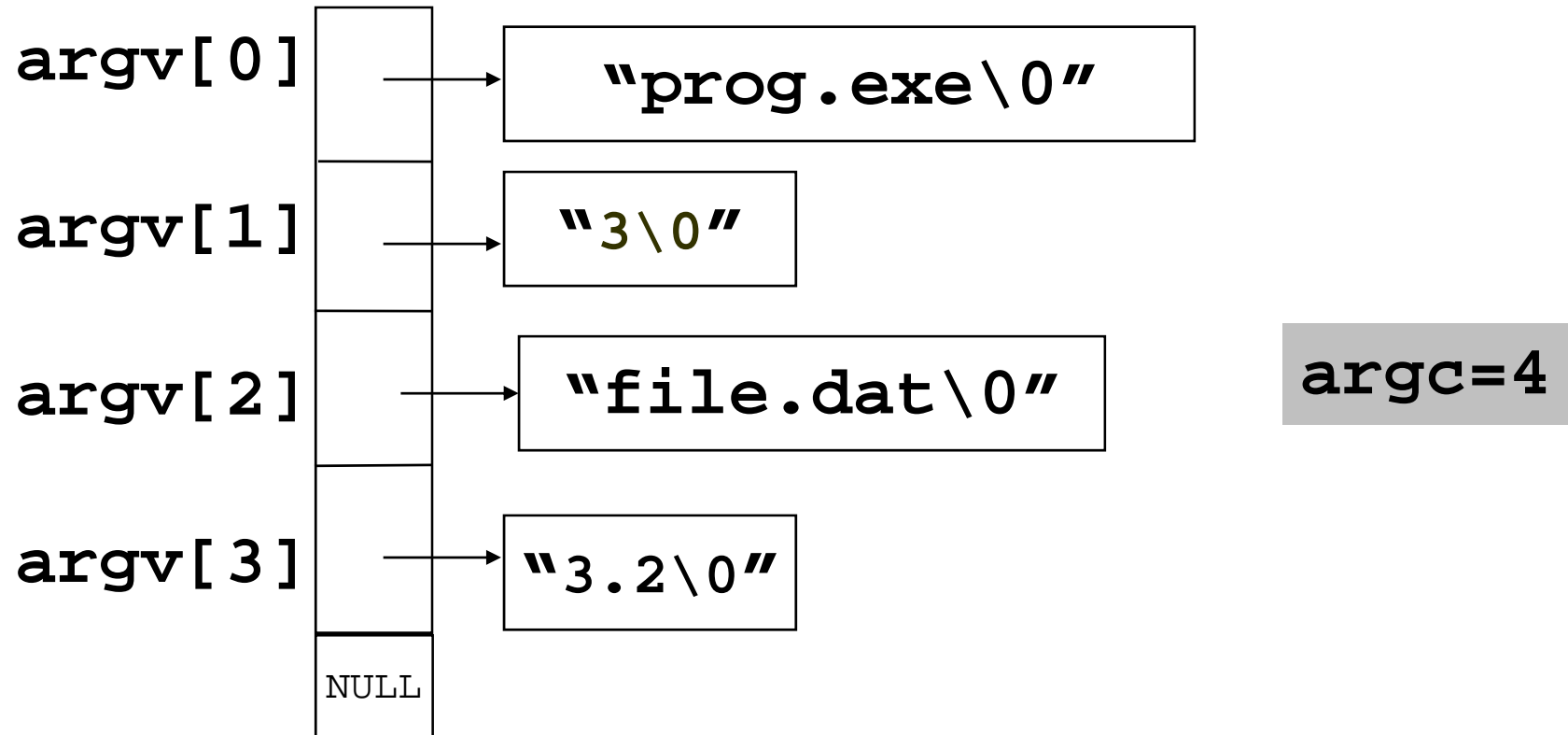
```
C: \progr>quadrato 5 K
```

- Numero argomenti = 3
- Argomento 2 = "5"
- Argomento 3 = "K"

argc e argv

- Struttura:

- Esempio: `c:\> prog.exe 3 file.dat 3.2`



argc e argv (Cont.)

- Ciclo per l'elaborazione degli argomenti

```
for (i=0; i<argc; i++) {  
    /*  
        elabora argv[i] come stringa  
    */  
}
```

- NOTA:
 - Qualunque sia la natura dell'argomento, è sempre una stringa
 - Necessario quindi uno strumento per "convertire" in modo efficiente stringhe in tipi numerici

Conversione degli argomenti

- Il C mette a disposizione tre funzioni per la conversione di una stringa in valori numerici

```
int atoi(char *s);
```

```
long atol(char *s);
```

```
double atof(char *s);
```

- Esempio:

```
int x = atoi("2") ;           // x=2
```

```
long y = atol("23L") ;        // y=23
```

```
double z = atof("2.35e-2") ; // z=0.0235
```

- Definite in `stdlib.h`

Conversione degli argomenti

- In alternativa ad atoi o atof si può usare la già nota sscanf:

```
x = atoi(argv[1])
```

Equivale a:

```
sscanf(argv[1], "%d", &x)
```

Conversione degli argomenti (Cont.)

- NOTA: Si assume che la stringa rappresenti l'equivalente di un valore numerico:
 - cifre, '+' , '-' per interi
 - cifre, '+' , '-' , 'l' , 'L' per long
 - cifre, '+' , '-' , 'e' , 'E' , '.' per reali
- In caso di conversione errata o non possibile le funzioni restituiscono il valore 0
 - Necessario in certi casi controllare il valore della conversione!
- NOTA: Importante controllare il valore di ogni `argv[i]`!

Conversione degli argomenti (Cont.)

- Esempio:
Programma C che prevede due argomenti sulla linea di comando:
 - Primo argomento: Un intero
 - Secondo argomento: Una stringa
- Schema:

```
int x;  
char s[80];  
x = atoi(argv[1]);  
strcpy(s,argv[2]);    /* s=argv[2] è errato! */
```

Programmi e opzioni


- Alcuni argomenti sulla linea di comando indicano tipicamente delle modalità di funzionamento “alternative” di un programma
- Queste “opzioni” (dette *flag* o *switch*) sono convenzionalmente specificate come

– <*carattere*>

per distinguerle dagli argomenti veri e propri

- Esempio

```
C:\> myprog -x -u file.txt
```


opzioni *argomento*

La funzione `exit`

- Esiste inoltre la funzione di libreria `exit`, dichiarata in `<stdlib.h>`, che:
 - Interrompe l'esecuzione del programma
 - Ritorna il valore specificato
- Il vantaggio rispetto all'istruzione `return` è che può essere usata all'interno di qualsiasi funzione, non solo del `main`

```
void exit(int value) ;
```

Esercizio 1

- Scrivere un programma che legga sulla linea di comando due interi N e D , e stampi tutti i numeri minori o uguali ad N divisibili per D

Esercizio 1: Soluzione

```
#include <stdio.h>
main(int argc, char* argv[]) {
    int N, D, i;
    if (argc != 3) {
        fprintf(stderr, "Numero argomenti non valido\n");
        return 1;
    }
    if (argv[1] != NULL) N = atoi(argv[1]);
    if (argv[2] != NULL) D = atoi(argv[2]);

    for (i=1; i<=N; i++) {
        if ((i % D) == 0) {
            printf("%d\n", i);
        }
    }
}
```

Altrimenti le operazioni
successive operano su
stringhe = NULL

Esercizio 2

- Scrivere un programma `m2m` che legga da input un testo e converta tutte le lettere maiuscole in minuscole e viceversa, a seconda dei flag specificati sulla linea di comando

`-l, -L` conversione in minuscolo

`-u, -U` conversione in maiuscolo

Un ulteriore flag `-h` permette di stampare un help

- Utilizzo:

`m2m -l`

`m2m -L`

`m2m -u`

`m2m -U`

`m2m -h`

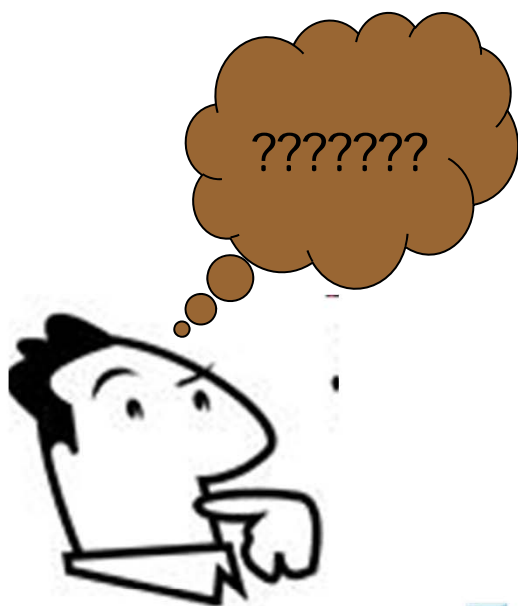
Esercizio 2: Soluzione

```
#include <stdio.h>

main(int argc, char* argv[]) {
    int lowercase = 0, uppercase = 0;

    for (i=1; i<argc; i++)
    {
        switch (argv[i][1]) {
            case 'l':
            case 'L':
                lowercase = 1;
                break;
            case 'u':
            case 'U':
                uppercase = 1;
                break;
            case 'h':
                printf("Uso: m2m [-luh]\n");
        }
    }
    ...
}
```

Esercizio "Bersagli" (1/2)



Esercizio "Bersagli"

Esercizio “Bersagli” (1/2)

- Si desidera creare un programma in grado di calcolare il numero di colpi andati a segno in un'esercitazione di tiro
- I bersagli sono descritti tramite le coordinate cartesiane del punto in cui sono posizionati all'interno di una griglia 100×100 . Le coordinate sono rappresentate solo da numeri interi, compresi tra 0 e 99. La posizione dei bersagli è contenuta nel file di testo bersagli.txt: ogni riga di tale file contiene le coordinate X e Y di un singolo bersaglio



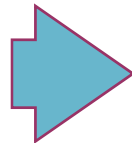
Esercizio "Bersagli" (2/2)

- I colpi sparati sono descritti anch'essi tramite le loro coordinate X e Y e sono memorizzati in un file di caratteri il cui nome è passato come primo parametro sulla linea di comando. Ogni riga di tale file contiene le coordinate X e Y del punto in cui è stato sparato un colpo
- Si scriva un programma che legga dai file succitati la posizione dei bersagli ed i colpi sparati e quindi calcoli il numero di colpi andati a segno, sia come valore assoluto sia come percentuale dei colpi sparati

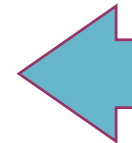
Analisi

bersagli i . txt

0	0
0	99
50	50
99	0
99	99



gi udi ce. exe



col pi . txt

49	49
50	50
51	51
52	52

```
Prompt dei comandi
C: \prog>gi udi ce col pi . txt
Col pi  sparati : 4
Col pi  andati a segno: 1 (25.0 %)
```

Algoritmo

- Acquisire dal file bersagli.txt tutte le coordinate, memorizzandole in due vettori paralleli Bx[] e By[]. Lunghezza dei vettori: Nb
- Acquisire dal file argv[1] le coordinate dei vari colpi Cx, Cy. Numero colpi: Nc
 - Per ciascun colpo, verificare se le coordinate coincidono con quelle di almeno un bersaglio
 - Se sì, incrementare Ncc
- Stampare Ncc e $Ncc/Nc * 100$

Soluzione (1/4)

```
int main(int argc, char *argv[])
{
    const int MAXB = 100 ;
        /* massimo numero di bersagli
x riga/colonna */
    const int MAX = 80 ;
        /* lunghezza riga del file */
    const char FILEB[] = "bersagli.txt";

    int Nb ; /* numero di bersagli */
    int Bx[MAXB], By[MAXB] ;
        /* coordinate dei bersagli */

    int Nc ; /* numero colpi sparati */
    int Ncc ; /* numero di colpi centrati */
}
```



Soluzione (2/4)

```
FILE *f ;
char riga[MAX] ;
int Cx, Cy ;
int i, r, trovato ;

/* 1: acquisizione coordinate bersagli */
f = myfopen( FILEB, "r" ) ;
Nb = 0 ;
while( fgets(riga, MAX, f) != NULL )
{
    r=sscanf(riga, "%d %d", &Bx[Nb], &By[Nb]);
    if( r!=2 )
        myerror("Formato errato\n") ;
    Nb ++ ;
}
myfclose(f);
```



Soluzione (3/4)

```
/* 2: analisi coordinate dei colpi */
if( argc != 2 )
    myerror("ERR: manca nome file\n");
f = myfopen( argv[1], "r" ) ;

Nc = 0 ;
Ncc = 0 ;
while( fgets(ri ga, MAX, f) != NULL )
{
    r = sscanf( ri ga, "%d %d", &Cx, &Cy );
    if(r!=2) myerror("Formato errato\n") ;
    Nc ++ ;

    /* Ri cerca del bersagli o */

}
myfclose(f);
```



Soluzione (3/4)

```
/* 2: analisi coordinate dei colpi */
```

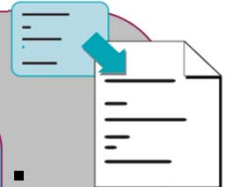
```
    trovato = 0 ;  
    for(i=0; i<Nb && trovato==0; i++)  
        if( Cx==Bx[i] && Cy==By[i] )  
            trovato = 1 ;
```

```
    if(trovato==1)  
        Ncc ++ ;
```

```
    Nc ++ ;
```

```
/* Ricerca del bersaglio */
```

```
}  
myfclose(f);
```



bersagli.c



Soluzione (4/4)



bersagli.c

```
/* 3: stampa risultati */  
printf("Colpi sparati: %d\n", Nc) ;  
printf("Colpi andati a segno: %d ", Ncc);  
if(Nc!=0)  
    printf("(%.2f%%)", Ncc*100.0/Nc) ;  
printf("\n");  
  
exit(0) ;  
}
```

Formattazione output



Formattazione dell'output

- L'output (su schermo o su file) viene formattato solitamente mediante la funzione `printf` (o `fprintf`)
- Ogni dato viene stampato attraverso un opportuno specificatore di formato (codici %)
- Ciascuno di questi codici dispone di ulteriori opzioni per meglio controllare la formattazione
 - Stampa incolonnata
 - Numero di cifre decimali
 - Spazi di riempimento
 - ...

Specificatori di formato

Tipo	printf
char	%c %d
int	%d
short int	%hd %d
long int	%ld
unsigned int	%u %o %x
unsigned short int	%hu
unsigned long int	%lu
float	%f %e %g
double	%f %e %g
char []	%s

Esempi

Istruzione	Risultato
<code>printf("%d", 13) ;</code>	13
<code>printf("%1d", 13) ;</code>	13
<code>printf("%3d", 13) ;</code>	13
<code>printf("%f", 13.14) ;</code>	13.140000
<code>printf("%6f", 13.14) ;</code>	13.140000
<code>printf("%12f", 13.14) ;</code>	13.140000
<code>printf("%6s", "ciao") ;</code>	ciao

Esempi (Cont.)

Istruzione	Risultato
<code>printf("%.1d", 13) ;</code>	13
<code>printf("%.4d", 13) ;</code>	0013
<code>printf("%6.4d", 13) ;</code>	__0013
<code>printf("%4.6d", 13) ;</code>	000013
<code>printf("%.2s", "ciao") ;</code>	ci
<code>printf("%.6s", "ciao") ;</code>	ci ao
<code>printf("%6.3s", "ciao") ;</code>	__ _ci a

Esempi (Cont.)

Istruzione	Risultato
<code>printf("%.2f", 13.14) ;</code>	13.14
<code>printf("%.4f", 13.14) ;</code>	13.1400
<code>printf("%6.4f", 13.14) ;</code>	13.1400
<code>printf("%9.4f", 13.14) ;</code>	13.1400

Esempi (Cont.)

Istruzione	Risultato
<code>printf("%6d", 13) ;</code>	13
<code>printf("%-6d", 13) ;</code>	13
<code>printf("%06d", 13) ;</code>	000013
<code>printf("%6s", "ci ao") ;</code>	ci ao
<code>printf("%-6s", "ci ao") ;</code>	ci ao

Esempi (Cont.)

Istruzione	Risultato
<code>printf("%d", 13) ;</code>	13
<code>printf("%d", -13) ;</code>	-13
<code>printf("%+d", 13) ;</code>	+13
<code>printf("%+d", -13) ;</code>	-13
<code>printf("% d", 13) ;</code>	13
<code>printf("% d", -13) ;</code>	-13

Approfondimenti su scanf

- Tipologie di caratteri nella stringa di formato
- Modificatori degli specificatori di formato
- Valore di ritorno
- Specificatore %[]

Stringa di formato (1/2)

- Caratteri stampabili:
 - scanf si aspetta che tali caratteri compaiano esattamente nell'input
 - Se no, interrompe la lettura
- Spaziatura ("whitespace"):
 - Spazio, tab, a capo
 - scanf "salta" ogni (eventuale) sequenza di caratteri di spaziatura
 - Si ferma al primo carattere non di spaziatura (o End-of-File)

Stringa di formato (2/2)

- Specificatori di formato (%-codice):
 - Se il codice non è %c, innanzitutto scanf “salta” ogni eventuale sequenza di caratteri di spaziatura
 - scanf legge i caratteri successivi e *cerca* di convertirli secondo il formato specificato
 - La lettura si interrompe al primo carattere che non può essere interpretato come parte del campo

Specificatori di formato

Tipo	scanf
char	%c
int	%d
short int	%hd
long int	%ld
unsigned int	%u %o %x
unsigned short int	%hu
unsigned long int	%lu
float	%f
double	%lf
char []	%s %[...]

Esempi

Istruzione	Input	Risultato
<code>scanf("%d", &x) ;</code>	134xyz	x = 134
<code>scanf("%2d", &x) ;</code>	134xyz	x = 13
<code>scanf("%s", v) ;</code>	134xyz	v = "134xyz"
<code>scanf("%2s", v) ;</code>	134xyz	v = "13"

Esempi (Cont.)

Istruzione	Input	Risultato
<code>scanf("%d %s", &x, v) ;</code>	10 Pi ppo	x = 10 v = "Pi ppo"
<code>scanf("%s", v) ;</code>	10 Pi ppo	x immutato v = "10"
<code>scanf("%*d %s", v) ;</code>	10 Pi ppo	x immutato v = "Pi ppo"

Valore di ritorno

- La funzione `scanf` ritorna un valore intero:
 - Numero di elementi (%) effettivamente letti
 - Non conta quelli "saltati" con %*
 - Non conta quelli non letti perché l'input non conteneva i caratteri desiderati
 - Non conta quelli non letti perché l'input è finito troppo presto
 - End-of-File per `fscanf`
 - Fine stringa per `sscanf`
 - EOF se l'input era già in condizione End-of-File all'inizio della lettura

Lettura di stringhe

- La lettura di stringhe avviene solitamente con lo specificatore di formato %s
 - Salta tutti i caratteri di spaziatura
 - Acquisisci tutti i caratteri seguenti, fermandosi al primo carattere di spaziatura (senza leggerlo)
- Qualora l'input dei separatori diversi da spazio, è possibile istruire scanf su quali siano i caratteri leciti, mediante lo specificatore %[*pattern*]

Esempi (Cont.)

Pattern	Effetto
<code>%[r]</code>	Legge solo sequenze di ' r '
<code>%[abcABC]</code>	Legge sequenze composte da a, b, c, A, B, C, in qualsiasi ordine e di qualsiasi lunghezza
<code>%[a-cA-C]</code>	Idem come sopra
<code>%[a-zA-Z]</code>	Sequenze di lettere alfabetiche
<code>%[0-9]</code>	Sequenze di cifre numeriche
<code>%[a-zA-Z0-9]</code>	Sequenze alfanumeriche
<code>%[^x]</code>	Qualunque sequenza che non contiene ' x '
<code>%[^\n]</code>	Legge fino a fine riga
<code>%[^, ; . ! ?]</code>	Si ferma alla punteggiatura o spazio