Capitolo 18

Schede riassuntive

18.1 Primo programma in C

Struttura di un file sorgente in C

```
/* programma: NomeFile.c
 * autore: NomeAutoreDelProgramma
 * BreveDescrizioneDelProgramma
 */

/* Inclusione delle librerie */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
   /* Definizione delle variabili */
   . . . .

   /* Istruzioni eseguibili */
   . . . .

   exit(0) ;
}
```

Nota

Quando il programma riceve degli argomenti sulla linea di comando, allora la definzione della funzione `main` deve essere modificata come:

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

Librerie principali

Lettura/scrittura su terminale e su file	`#include <stdio.h>`
Interazione con sistema operativo	`#include <stdlib.h>`
Funzioni matematiche	`#include <math.h>`
Elaborazione di testi e stringhe	`#include <string.h>`
Analisi del tipo di caratteri	`#include <ctype.h>`
Valori minimi e massimi	`#include <limits.h>`

Definizione delle variabili

Definizione di variabili intere	`int i, j ;`
Definizione di variabili reali	`float r ;`

Istruzioni eseguibili

Assegnazione a variabile	`a = 0 ;`
	`b = 17 ;`
	`c = a + b ;`
	`b = b + 1 ;`
	`d = b * b - 4 * a * c ;`
	`e = b * (b - 4 * a) * c ;`
	`x1 = ( -b + sqrt(d) ) / ( 2*a ) ;`
	`nomevariabile = espressione ;`

Lettura (input) di numeri interi	`scanf("%d", &i) ;` ⇐ ricordare il '`&`'
Lettura (input) di numeri reali	`scanf("%f", &r) ;` ⇐ ricordare il '`&`'
Stampa (output) di messaggi e numeri	`printf("Numero␣%d,␣valore␣%f\n", i, r) ;`
Vai a capo nella stampa	`printf("\n") ;`

Espressioni aritmetiche

Le 4 operazioni	`+ - * /`
Le parentesi	`( (a + b) * (c / (d - e) ) )`
Resto della divisione	`%`

Funzioni definite in `math.h`

Valore assoluto	$y \leftarrow \lvert x \rvert$	`y = fabs(x) ;`
Radice quadrata	$y \leftarrow \sqrt{x}$	`y = sqrt(x) ;`
Radice cubica	$y \leftarrow \sqrt[3]{x}$	`y = cbrt (x) ;`
Elevamento a potenza	$y \leftarrow x^z$	`y = pow(x, z) ;`
Ipotenusa	$y \leftarrow \sqrt{x^2 + z^2}$	`y = hypot(x, z) ;`
Ceiling	$y \leftarrow \lceil x \rceil$	`y = ceil(x) ;`
Floor	$y \leftarrow \lfloor x \rfloor$	`y = floor(x) ;`
Arrotondamento	$y \leftarrow \lfloor x + 1/2 \rfloor$	`y = round(x) ;`
Troncamento verso 0	$y \leftarrow sign(x)\, \lfloor \lvert x \rvert \rfloor$	`y = trunc(x) ;`
Resto della divisione	$y \leftarrow Resto(x/z)$	`y = fmod(x, z) ;`
Esponenziale	$y \leftarrow e^x$	`y = exp(x) ;`
	$y \leftarrow 2^x$	`y = exp2(x) ;`
Logaritmo	$y \leftarrow \ln x$	`y = log(x) ;`
	$y \leftarrow \log_2 x$	`y = log2(x) ;`
	$y \leftarrow \log_{10} x$	`y = log10(x) ;`
Funzioni trigonometriche	$y \leftarrow \sin x$	`y = sin(x) ;`
	$y \leftarrow \cos x$	`y = cos(x) ;`
	$y \leftarrow \tan x$	`y = tan(x) ;`
Funzioni trigonometriche inverse	$y \leftarrow \arcsin x$	`y = asin(x) ;`
	$y \leftarrow \arccos x$	`y = acos(x) ;`
	$y \leftarrow \arctan x$	`y = atan(x) ;`
	$y \leftarrow \arctan(x/z)$	`y = atan2(x, z) ;`
Funzioni iperboliche	$y \leftarrow \sinh x$	`y = sinh(x) ;`
	$y \leftarrow \cosh x$	`y = cosh(x) ;`
	$y \leftarrow \tanh x$	`y = tanh(x) ;`
Funzioni iperboliche inverse	$y \leftarrow \sinh^{-1} x$	`y = asinh(x) ;`
	$y \leftarrow \cosh^{-1} x$	`y = acosh(x) ;`
	$y \leftarrow \tanh^{-1} x$	`y = atanh(x) ;`

18.2 Istruzioni di scelta in C

Espressioni condizionali

Confronto di uguaglianza	`==`	⇐ mai usare `=`
Confronto di disuguaglianza	`!=`	
Confronto di ordine	`< <= > >=`	
Congiunzione AND	`(a>0) && (b>0)`	
Disgiunzione OR	`(a>0) || (b>0)`	
Negazione NOT	`!(a+b<c)`	
Appartenenza ad intervalli $x \in [a, b]$	`a<=x && x<=b`	⇐ mai usare `a<=x<=b`
Esclusione da intervalli $x \notin [a, b]$	`x<a || x>b`	⇐ oppure `!(a<=x && x<=b)`

Costrutto `if-else`

Costrutto condizionale semplice	`if (condizione)`
	`{`
	`  istruzioni 1 ;`
	`}`
	`else`
	`{`
	`  istruzioni 2 ;`
	`}`

Costrutto condizionale senza alternativa	`if (condizione)`
	`{`
	`  istruzioni 1 ;`
	`}`

Costrutti `if-else` multipli

Costrutti condizionali sequenziali

```
if ( condizione1 )
{
    istruzioni 1 ;
}
else
{
    istruzioni 2 ;
}

if( condizione2 )
{
    istruzioni 3 ;
}
else
{
    istruzioni 4 ;
}
```

Costrutti condizionali annidati

```
if ( condizione1 )
{
    istruzioni 1 ;
    if( condizione2 )
    {
        istruzioni 2 ;
    }
    else
    {
        istruzioni 3 ;
    }
    istruzioni 4 ;
}
else
{
    istruzioni 5 ;
    if( condizione3 )
    {
        istruzioni 6 ;
    }
    else
    {
        istruzioni 7 ;
    }
    istruzioni 8 ;
}
```

Costrutto condizionale con più alternative

```
if (condizione 1)
{
    istruzioni 1 ;
}
else if (condizione 2)
{
    istruzioni 2 ;
}
else
{
    istruzioni 3 ;
}
```

Costrutto `switch`

Espressione di tipo intero

```
switch (espressione)
{
    case 2:
        istruzioni 1 ;
    break ;

    case 20:
        istruzioni 2 ;
    break ;

    case 210:
        istruzioni 3 ;
    break ;

    default:
        istruzioni di default ;
    break ;
}
```

Espressione di tipo carattere

```
switch (carattere)
{
    case 'a':
    case 'A':
            istruzioni 1 ;
    break ;

    case 'b':
    case 'B':
        istruzioni 2 ;
    break ;

    case 'c':
    case 'C':
        istruzioni 3 ;
    break ;

    case '␣':
        istruzioni 4 ;
    break ;

    case '*':
        istruzioni 5 ;
    break ;

    default:
            istruzioni di default ;
    break ;
}
```

18.3 Cicli ed iterazioni in C

Struttura di un ciclo

1. **Inizializzazione**. Assegnazione del valore iniziale a tutte le variabili che vengono lette durante il ciclo (nella condizione o nel corpo).
2. **Condizione di ripetizione**. Condizione, di solito inizialmente vera, che al termine del ciclo diventerà falsa. Deve dipendere da variabili che saranno modificate all'interno del ciclo (nel corpo o nell'aggiornamento).
3. **Corpo del ciclo**. Le istruzioni che effettivamente occorre ripetere: sono lo scopo per cui il ciclo viene realizzato. Si possono usare e modificare le variabili inizializzate.
4. **Aggiornamento**. Modifica di una o più variabili in grado di aggiornare il valore della condizione di ripetizione (rendendola, prima o poi, falsa). Tengono "traccia" del progresso dell'iterazione.

Operatori di auto-incremento/decremento

Auto-incremento	`i++ ;` `++i ;`	equivale a `i = i + 1 ;`
Auto-decremento	`i-- ;` `--i ;`	equivale a `i = i - 1 ;`

Costrutti iterativi

Costrutto **`while`**	`while(condizione)`
	`{`
	`  corpo ;`
	`}`

Costrutto **`do-while`**	`do`
	`{`
	`  corpo ;`
	`}`
	`while(condizione) ;`

Costrutto **`for`**	`for( inizializzazione; condizione; incremento )`
	`{`
	`  corpo ;`
	`}`

Equivalenza `for-while`

```
for ( inizializz; condiz; aggiornamento )
{
    corpo ;
}
```

```
inizializz ;
while ( condiz )
{
    corpo ;
    aggiornamento ;
}
```

Ciclo infinito

```
for( ; ; )
{
    corpo ;
}
```

```
while( 1 )
{
    corpo ;
}
```

Numero di iterazioni noto a priori

Da 0 a $N-1$, crescente	`for( i=0 ; i<N ; i++)`
	`{`
	`  corpo ;`
	`}`
		`i = 0 ;`
		`while( i<N )`
		`{`
		`  corpo ;`
		`  i++ ;`
		`}`

Da 1 a N, crescente	`for( i=1 ; i<=N ; i++)`
	`{`
	`  corpo ;`
	`}`
		`i = 1 ;`
		`while( i<=N )`
		`{`
		`  corpo ;`
		`  i++ ;`
		`}`

Da $N-1$ a 0, decrescente	`for( i=N-1 ; i>=0 ; i--)`
	`{`
	`  corpo ;`
	`}`
		`i = N-1 ;`
		`while( i>=0 )`
		`{`
		`  corpo ;`
		`  i-- ;`
		`}`

Da N a 1, decrescente	`for( i=N ; i>0 ; i--)`
	`{`
	`  corpo ;`
	`}`
		`i = N ;`
		`while( i>0 )`
		`{`
		`  corpo ;`
		`  i-- ;`
		`}`

Numero di iterazioni non noto a priori

Finché l'utente non inserisce un dato speciale

```
scanf("%d", &dato) ;

while( dato != DATOSPECIALE )
{
    elabora_dato ;
    scanf("%d", &dato) ;
}
```

```
do
{
    scanf("%d", &dato) ;
    if( dato != DATOSPECIALE )
    {
        elabora_dato ;
    }
}
while( dato != DATOSPECIALE ) ;
```

Finché non si verifica una condizione particolare

```
fine = 0 ; /* inizializzazione "flag" */
while( fine == 0 )
{
    elabora1 ;

    if( condizione_particolare )
       fine = 1 ;

    elabora2 ;
}
```

Contatori

Conta le iterazioni

```
conta = 0 ;

while( condizione )
{
    istruzioni ;

    conta ++ ;
}
```

Conta quante volte si verifica una condizione particolare

```
conta = 0 ;

while( condizione )
{
    istruzioni ;

    if (condizione_particolare)
        conta ++ ;

    altre_istruzioni ;
}
```

Accumulatori

Somma valori

```
somma = 0 ;

for( i=0 ; i<N; i++ )
{
    istruzioni ; /* calcola "valore" */

    somma = somma + valore ;
}
```

Massimo

```
max = INT_MIN ;
/* inizializzato ad un valore minore dei
   numeri di cui si vuole calcolare
   il massimo */

for( i=0 ; i<N; i++ )
{
    istruzioni ; /* calcola "numero" */

    if( numero > max )
        max = numero ;
}
```

Minimo

```
min = INT_MAX ;
/* inizializzato ad un valore maggiore dei
   numeri di cui si vuole calcolare
   il massimo */

for( i=0 ; i<N; i++ )
{
    istruzioni ; /* calcola "numero" */

    if( numero < min )
        min = numero ;
}
```

Flag

```
trovato = 0 ; /* flag per la ricerca */
/* inizializzo a "NO" = falso */

for( i=0 ; i<N; i++ )
{
    istruzioni ;

    if(condizione_particolare)
        trovato = 1 ;

    altre_istruzioni ;
}

/* al termine del ciclo, verifico */
if( trovato == 1 )
{
    printf("SI") ;
}
else
{
    printf("NO") ;
}
```

Esistenza e Universalità

	Esistenza	**Universalità**
P è vero	Esiste almeno un caso in cui P sia vero	In tutti i casi, P è vero

	`esiste = 0 ;`
	`while(condizione)`
	`{`
	`  if( P è vero )`
	`  esiste = 1 ;`
	`}`
	
	`if ( esiste==1 ) ...`
		`sempre = 1 ;`
		`while(condizione)`
		`{`
		`  if( P non è vero )`
		`  sempre = 0 ;`
		`}`
		
		`if ( sempre==1 ) ...`

P è falso	Esiste almeno un caso in cui P sia falso	In tutti i casi, P è falso

	`esiste = 0 ;`
	`while(condizione)`
	`{`
	`  if( P non è vero )`
	`  esiste = 1 ;`
	`}`
	
	`if ( esiste==1 ) ...`
		`sempre = 1 ;`
		`while(condizione)`
		`{`
		`  if( P è vero )`
		`  sempre = 0 ;`
		`}`
		
		`if ( sempre==1 ) ...`

Cicli Annidati

```
for( i=0; i<10; i++ )
{
    for( j=0; j<10; j++ )
    {
        printf("i=%d␣-␣j=%d\n", i, j);
    }
}
```

```
i=0 - j=0
i=0 - j=1
i=0 - j=2
...
i=0 - j=8
i=0 - j=9
i=1 - j=0
i=1 - j=1
i=1 - j=2
...
i=1 - j=8
i=1 - j=9
i=2 - j=0
i=2 - j=1
i=2 - j=2
...
i=2 - j=8
i=2 - j=9
...
...
i=9 - j=0
i=9 - j=1
i=9 - j=2
...
i=9 - j=8
i=9 - j=9
```

Istruzioni `break` e `continue`

```
while ( C )
{
  B1 ;
  if ( U ) /* condizione uscita */
     break ;
  B2 ;
}
/* se U e' vera, salta
immediatamente qui,
ed interrompe il ciclo
anche se C e' ancora vera.
In tal caso, B2 non
viene eseguita. */
```

```
while ( C )
{
  B1 ;
  if ( U )
     continue ;
  B2 ;
  /* se U e' vera, salta
  immediatamente qui,
  poi riprende la prossima
  iterazione. In tal caso,
  B2 non viene eseguita. */
}
```

18.4 Vettori in C

Definizione di costanti

`#define MAX 100`	`const int MAX = 100 ;`
Prima del `main()`	Solitamente dentro al `main()`
Senza `;`	Necessario il `;`
Senza `=`	Necessario il `=`
Senza tipo di dato	Necessario il tipo: **`int`**, **`float`**, ...

Definizione di vettori

```
int vet[100] ;
```

`int`	Tipo del vettore: **`int`**, **`float`**, **`char`**, **`double`**
`vet`	Nome del vettore (arbitrario)
`100`	Numero di elementi, deve essere **costante** (numerica o simbolica)

Numero di elementi con costante simbolica

```
#define N 10

int main(void)
{
   int dato[N] ;
   . . .
}
```

```
int main(void)
{
   const int N = 10 ;

   int dato[N] ;

   . . .
}
```

Vettori con occupazione variabile

`MAX`	Costante, la massima dimensione del vettore
`N`	Variabile **`int`**, pari al numero effettivo di elementi usati
da `0` a `N-1`	Posizioni del vettore effettivamente occupate
da `N` a `MAX-1`	Posizioni del vettore non utilizzate

```
const int MAXN = 100 ; /* dimensione massima */

int v[MAXN] ; /* vettore di dim. max. */

int N ; /* occupazione effettiva del vettore */

N = 0 ; /* inizialmente "vuoto" */
```

```
/* aggiunta in coda */

v[N] = nuovo_elemento ;

N++ ;
```

Operazioni elementari sui vettori

Stampa

```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    printf("%d\n", v[i]) ;
}
```

Lettura

```
printf("Lettura di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    scanf("%d", &v[i]) ;
}
```

Copia

```
/* copia il contenuto di v[] in w[] */

for( i=0; i<N; i++ )
{
    w[i] = v[i] ;
}
```

Ricerca di `dato`

```
trovato = 0 ;
pos = -1 ;

for( i=0 ; i<N ; i++ )
{
    if( v[i] == dato )
    {
        trovato = 1 ;
        pos = i ;
    }
}

if(trovato==1) ...
```

Ricerca del massimo

```
float max ;  /* valore del massimo */
int posmax ; /* posizione del max */

max = r[0] ;
posmax = 0 ;

for( i=1 ; i<N ; i++ )
{
    if( r[i]>max )
    {
        max = r[i] ;
        posmax = i ;
    }
}
```

18.5 Caratteri e stringhe in C

Codice ASCII a 7 bit

		0 0x	16 1x	32 2x	48 3x	64 4x	80 5x	96 6x	112 7x
0	x0	NUL			0	@	P	`	p
1	x1			!	1	A	Q	a	q
2	x2			"	2	B	R	b	r
3	x3			#	3	C	S	c	s
4	x4			$	4	D	T	d	t
5	x5			%	5	E	U	e	u
6	x6			&	6	F	V	f	v
7	x7	BEL		'	7	G	W	g	w
8	x8	BS		(	8	H	X	h	x
9	x9	TAB		)	9	I	Y	i	y
10	xA	LF		*	:	J	Z	j	z
11	xB		ESC	+	;	K	[	k	{
12	xC	FF		,	<	L	\	l	|
13	xD	CR		-	=	M	]	m	}
14	xE			.	>	N	^	n	~
15	xF			/	?	O	_	o	DEL

Variabili di tipo `char`

Definizione	`char ch ;`
Assegnazione	`ch = 'K';` `ch = 75 ;`
Lettura	`scanf("%c", &ch) ;` `ch = getchar() ;`
Stampa	`printf("%c", ch) ;` `putchar(ch) ;`

Sequenze di escape

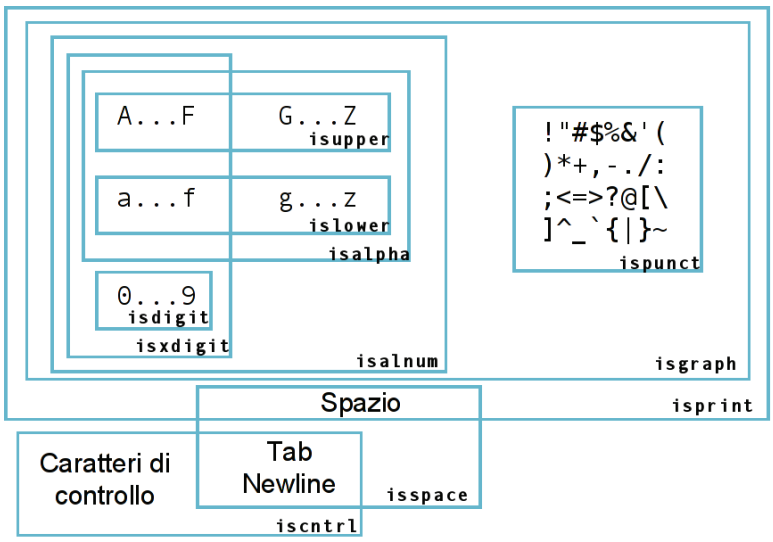
`'\n'`	A capo
`'\t'`	Tabulazione
`'\b'`	Backspace (cancella ultimo carattere)
`'\a'`	Campanello (*alert*)
`'\r'`	Ritorno carrello sulla stessa riga
`'\\'`	Carattere di *backslash* `\`
`'\''`	Carattere di singolo apice `'`
`'\"'`	Carattere di doppio apice `"`
`'\xNN'`	Carattere il cui codice ASCII vale `NN` (in base 16)

Variabili di tipo stringa

Definizione	`char s[LUN+1] ;`
Assegnazione	`strcpy(s, "ciao") ;` `strcpy(s, s2) ;`
Lettura	`scanf("%s", s) ;` `gets(s) ;`
Stampa	`printf("%s", s) ;` `puts(s) ;`

Funzioni della libreria `<ctype.h>`

Nome	Parametri	Restituisce	Descrizione	Esempi
`isalpha`	**char** ch	vero/falso	Lettera maiuscola o minuscola (`A...Z`, `a...z`)	`if(isalpha(ch)) { ... }`
`isupper`	**char** ch	vero/falso	Lettera maiuscola (`A...Z`)	`if(isupper(ch)) { ... }`
`islower`	**char** ch	vero/falso	Lettera minuscola (`a...z`)	`if(islower(ch)) { ... }`
`isdigit`	**char** ch	vero/falso	Cifra numerica (`0...9`)	`if(isdigit(ch)) { ... }`
`isalnum`	**char** ch	vero/falso	Lettera oppure cifra numerica: `isalpha(ch)||isdigit(ch)`	`if(isalnum(ch)) { ... }`
`isxdigit`	**char** ch	vero/falso	Cifra numerica oppure lettera valida in base 16 (`a...f`, `A...F`)	`if(isxdigit(ch)) { ... }`
`ispunct`	**char** ch	vero/falso	Simbolo di punteggiatura (`!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`)	`if(ispunct(ch)) { ... }`
`isgraph`	**char** ch	vero/falso	Qualsiasi simbolo visibile (lettera, cifra, punteggiatura)	`if(isgraph(ch)) { ... }`
`isprint`	**char** ch	vero/falso	Qualsiasi simbolo visibile o spazio	`if(isprint(ch)) { ... }`
`isspace`	**char** ch	vero/falso	Spazio, tab o a capo	`if(isspace(ch)) { ... }`
`iscntrl`	**char** ch	vero/falso	Qualsiasi carattere di controllo	`if(iscntrl(ch)) { ... }`
`toupper`	**char** ch	**char**	Ritorna la versione maiuscola di `ch`	`for(i=0; s[i]!=0; i++) s[i] = toupper(s[i]) ;`
`iscntrl`	**char** ch	**char**	Ritorna la versione minuscola di `ch`	`for(i=0; s[i]!=0; i++) s[i] = tolower(s[i]) ;`



Funzioni della libreria `<string.h>`

Nome	Parametri	Restituisce	Descrizione	Esempi
`strlen`	**char** `s[N]`	**int**	Lunghezza della stringa	`lun = strlen(s) ;`
`strcpy`	**char** `dst[N]`, **char** `src[M]`		Copia il contenuto di `src` all'interno di `dst`	`strcpy(s1, s2) ;` `strcpy(s, "") ;` `strcpy(s1, "ciao") ;`
`strncpy`	**char** `dst[N]`, **char** `src[M]`, **int** `nc`		Copia il contenuto di `src` (max `nc` caratteri) all'interno di `dst`	`strncpy(s1, s2, 20) ;` `strncpy(s1, s2, MAX) ;`
`strcat`	**char** `dst[N]`, **char** `src[N]`		Accoda il contenuto di `src` alla fine di `dst`	`strcat(s1, s2) ;` `strcat(s1, "␣") ;`
`strncat`	**char** `dst[N]`, **char** `src[M]`, **int** `nc`		Accoda il contenuto di `src` (max `nc` caratteri) alla fine di `dst`	`strncat(s1, s2, 50) ;`
`strcmp`	**char** `s1[N]`, **char** `s2[M]`	**int**	Risultato `<0` se `s1` precede `s2`, `==0` se `s1` è uguale a `s2`, `>0` se `s1` segue `s2`	**if**`(strcmp(s, r)==0)` **while**`(strcmp(r,"*")!=0)`
`strncmp`	**char** `s1[N]`, **char** `s2[M]`, **int** `n`	**int**	Come `strcmp`, ma confronta solo i primi `n` caratteri	**if**`(strncmp(r, "buon", 4)==0)`
`strchr`	**char** `s[N]`, **char** `ch`	`==NULL` o `!=NULL`	Risultato `!=NULL` se il carattere `ch` compare nella stringa, `==NULL` se non compare.	**if**`(strchr(s, '.')!=NULL)` **if**`(strchr(s, ch)==NULL)`
`strstr`	**char** `s[N]`, **char** `r[N]`	`==NULL` o `!=NULL`	Risultato `!=NULL` se la sotto-stringa `r` compare nella stringa `s`, `==NULL` se non compare.	**if**`(strstr(s, "xy")!=NULL)` **if**`(strstr(s, s1)==NULL)`
`strspn`	**char** `s[N]`, **char** `r[N]`	**int**	Restituisce la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri presenti in `r` (in qualsiasi ordine).	`lun = strspn(s, "␣") ;` `lun = strspn(s, ".,") ;`
`strcspn`	**char** `s[N]`, **char** `r[N]`	**int**	Restituisce la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri *non* presenti in `r`.	`lun = strspn(s, "␣") ;` `lun = strspn(s, ".,") ;`

18.6 Matrici e Vettori di stringhe in C

Definizione di matrici (vettori multidimensionali)

int `pitagora[10][20] ;`

int	Tipo della matrice: **int**, **float**, **char**, **double**
`pitagora`	Nome della matrice (arbitrario)
`10`	Numero di righe, deve essere **costante** (numerica o simbolica)
`20`	Numero di colonne, deve essere **costante** (numerica o simbolica)

Operazioni elementari sulle matrici

Stampa (per righe)

```
printf("Matrice: %d x %d\n", N, M);

for(i=0; i<N; i++)
{
    for(j=0; j<M; j++)
    {
        printf("%f ", mat[i][j]) ;
    }
    printf("\n");
}
```

Stampa (per colonne)

```
printf("Matrice: %d x %d\n", N, M);

for(j=0; j<M; j++)
{
    for(i=0; i<N; i++)
    {
        printf("%f ", mat[i][j]) ;
    }
    printf("\n");
}
```

Lettura

```
printf("Immetti matrice %d x %d\n", N, M) ;

for(i=0; i<N; i++)
{
    printf("Riga %d:\n", i+1) ;
    for(j=0; j<M; j++)
    {
        printf("Elemento (%d,%d): ", i+1, j+1) ;
        scanf("%f", &mat[i][j]) ;
    }
}
```

Copia

```
/* copia il contenuto di mat[][] in mat2[][] */
for(i=0; i<N; i++)
  for(j=0; j<M; j++)
     mat2[i][j] = mat[i][j] ;
```

Somma delle righe

```
for(i=0 ; i<N ; i++)
{
    somma = 0.0 ;
    for(j=0; j<M; j++)
        somma = somma + mat[i][j] ;
    sommarighe[i] = somma ;
}

for(i=0; i<N; i++)
    printf("Somma riga %d = %f\n", i+1, sr[i]) ;
```

Somma delle colonne

```
for(j=0 ; j<M ; j++)
{
    somma = 0.0 ;
    for(i=0; i<N; i++)
        somma = somma + mat[i][j] ;
    sommacolonne[j] = somma ;
}

for(j=0; j<M; j++)
    printf("Somma colonna %d = %f\n", j+1, sc[j]) ;
```

Somma di tutto

```
somma = 0.0 ;
for(i=0 ; i<N ; i++)
{
    for(j=0; j<M; j++)
        somma = somma + mat[i][j] ;
}

printf("Somma complessiva = %f\n", somma) ;
```

Ricerca di un elemento

```
trovato = 0 ;
riga = -1 ;
col  = -1 ;

for(i=0; i<N && trovato==0; i++)
    for(j=0; j<M && trovato==0; j++)
        if( mat[i][j]==dato )
        {
            trovato=1 ;
            riga = i ;
            col = j ;
        }

if(trovato==1)
  printf("Dato %f presente: (%d,%d)\n",
    dato, riga+1, col+1) ;
else
  printf("Dato %f non presente\n", dato) ;
```

Definizione di vettori di stringhe (matrici di caratteri)

```
char vett[MAX][LUN+1] ;
```

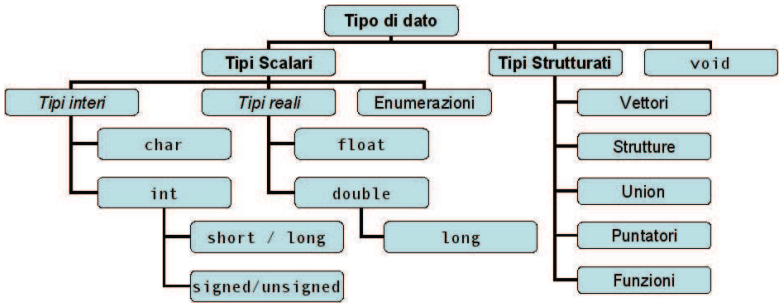
`char`	Tipo base `char`
`vett`	Nome del vettore (arbitrario)
`MAX`	Numero di righe, ossia numero di *stringhe* diverse da memorizzare. Deve essere **costante** (numerica o simbolica)
`LUN+1`	Numero di colonne, ossia lunghezza massima delle stringhe (max `LUN` caratteri più il terminatore nullo). Deve essere **costante** (numerica o simbolica)

Confronto tra stringa e vettore di stringhe

`char s[LUN+1] ;`	`char v[MAX][LUN+1] ;`
`s[i]` è un singolo carattere	`v[i][j]` è un singolo carattere
`s` è l'intera stringa	`v[i]` è un'intera stringa
	`v` è l'intera matrice

18.7 Tipi di dato in C

Il sistema dei tipi di dato in C



I tipi interi

	<limits.h>		compilatore gcc		
Tipo	**Min**	**Max**	**N.bit**	**Min**	**Max**
`char`	CHAR_MIN	CHAR_MAX	8	-128	127
`int`	INT_MIN	INT_MAX	32	$-2\,147\,483\,648$	$2\,147\,483\,647$
`short int`	SHRT_MIN	SHRT_MAX	16	$-32\,768$	$32\,767$
`long int`	LONG_MIN	LONG_MAX	32	$-2\,147\,483\,648$	$2\,147\,483\,647$
`unsigned int`	0	UINT_MAX	32	0	$4\,294\,967\,295$
`unsigned short int`	0	USHRT_MAX	16	0	$65\,535$
`unsigned long int`	0	ULONG_MAX	32	0	$4\,294\,967\,295$

I tipi reali

Tipo	**N.bit**	**Mantissa**	**Esponente**	**Min/Max**	**Epsilon**
`float`	32	23 bit	8 bit	$\pm 3.402 \cdot 10^{+38}$	$\pm 1.175 \cdot 10^{-38}$
`double`	64	53 bit	10 bit	$\pm 1.797 \cdot 10^{+308}$	$\pm 2.225 \cdot 10^{-308}$

Specificatori di formato

Tipo	scanf	printf
`char`	`"%c"`	`"%c", "%d"`
`int`	`"%d"`	`"%d"`
`short int`	`"%hd"`	`"%hd", "%d"`
`long int`	`"%ld"`	`"%ld"`
`unsigned int`	`"%u", "%o", "%x"`	`"%u", "%o", "%x"`
`unsigned short int`	`"%hu"`	`"%hu"`
`unsigned long int`	`"%lu"`	`"%lu"`
`float`	`"%f"`	`"%f", "%g"`
`double`	`"%lf"`	`"%f", "%g"`

Conversioni di tipo automatiche

Promozione automatica	da	...	a
	`char`	...	`int`
	`short int`	...	`int`
	`int`	...	`long int`
	`long int`	...	`double`
	`float`	...	`double`

Conversioni di tipo esplicite

Tra tipi scalari	`(nuovotipo)espressione`

Da stringa a numero	`gets(line) ;`
	`x = atoi(line) ; /* int */`
	`x = atol(line) ; /* long */`
	`x = atof(line) ; /* float o double */`

18.8 Funzioni in C

Definizione di funzioni

- **Prototipo**: dichiarazione del nome, dell'interfaccia e del tipo delle variabili. Ricorda: finisce con un punto-e-virgola!

```
int leggi(int min, int max) ;
```

- **Definizione**: dichiarazione dell'interfaccia e definizione del corpo effettivo della funzione. Nessun punto-e-virgola! Ricorda: necessaria l'istruzione **`return`**.

```
int leggi(int min, int max)
{
   int val ;

   /* ... codice del corpo della funzione ... */

   return val ;
}
```

- **Chiamata**: utilizzo della funzione all'interno di un altra funzione

```
int main(void)
{
  int x, a, b ;
    . . .
  x = leggi(a, b) ;
    . . .
}
```

Parametri delle funzioni

- Tipi scalari, passati *by value*

```
int funz(int x, double f) ;
```

- Vettori, passati *by reference*

```
int funz(int v[]) ;
```

- Tipi scalari, passati *by reference*

```
int funz(int *x, double *f)
{
   *x = 1 ;
   *f = 2.3 ;
}

. . .

funz( &i, &w ) ;
```

La funzione `main`

```
int main(int argc, char *argv[])
{
   printf("%d argomenti ricevuti\n", argc ) ;

   printf("Nome del programma: %s\n", argv[0] ) ;

   printf("Primo argomento: %s\n", argv[1] ) ;

   exit(0) ; /* termina con successo */

   exit(1) ; /* termina con errore */
}
```

Vediamo gli argomenti ricevuti

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i ;

    printf("argc = %d\n", argc) ;
    for(i=0; i<argc; i++)
    {
        printf("argv[%d] = \"%s\"\n", i, argv[i]) ;
    }
}
```

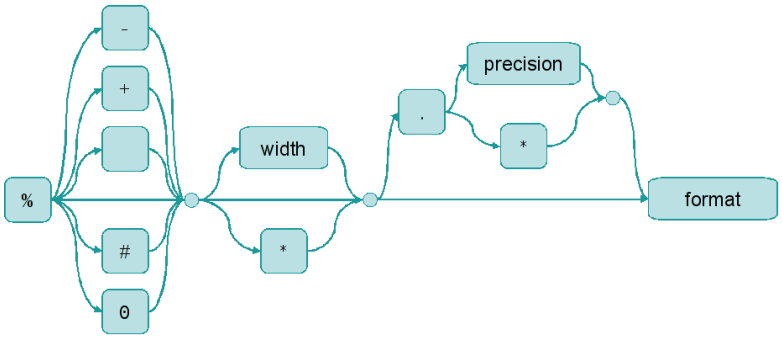
18.9 I/O Avanzato in C

Funzione `sscanf`

```
r = sscanf(str, "formato", &variabili ) ;
```

`str`	Stringa da cui acquisire i dati
`"formato"`	Sequenza dei campi da leggere (`"%d"`, `"%s"`, ...)
`&variabili`	Variabili nelle quali depositare i valori estratti dalla stringa
`r`	Valore di ritorno: numero di `&variabili` lette con successo

Formato di output avanzato



- format: carattere `d`, `f`, `c`, `s`
- width: lunghezza totale, numero minimo di caratteri stampati
- .precision: dipende dal formato:
  - `"%d"`: numero minimo numero di caratteri totali (eventualmente aggiunge 0 a sinistra)
  - `"%f"`: numero massimo di cifre dopo la virgola
  - `"%s"`: massimo numero di caratteri (stringhe più lunghe vengono troncate)

- modificatori iniziali di riempimento ed allineamento:
  - `"-"`: allinea a sinistra anziché a destra
  - `"+"`: aggiungi il segno anche davanti ai numeri positivi
  - `"␣"`: aggiungi uno spazio davanti ai numeri positivi
  - `"0"`: aggiungi 0 iniziali fino a width
  - `"#"`: formato "alternativo" (dipende dai casi, vedere documentazione)

Formato di output avanzato – esempi

`printf("%d", 13) ;`	`13`
`printf("%1d", 13) ;`	`13`
`printf("%3d", 13) ;`	`␣13`
`printf("%f", 13.14) ;`	`13.140000`
`printf("%6f", 13.14) ;`	`13.140000`
`printf("%12f", 13.14) ;`	`␣␣␣13.140000`
`printf("%6s", "ciao") ;`	`␣␣ciao`
`printf("%.1d", 13) ;`	`13`
`printf("%.4d", 13) ;`	`0013`
`printf("%6.4d", 13) ;`	`␣␣0013`
`printf("%4.6d", 13) ;`	`000013`
`printf("%.2s", "ciao") ;`	`ci`
`printf("%.6s", "ciao") ;`	`ciao`
`printf("%6.3s", "ciao") ;`	`␣␣␣cia`
`printf("%.2f", 13.14) ;`	`13.14`
`printf("%.4f", 13.14) ;`	`13.1400`
`printf("%6.4f", 13.14) ;`	`13.1400`
`printf("%9.4f", 13.14) ;`	`␣␣13.1400`
`printf("%6d", 13) ;`	`␣␣␣␣13`
`printf("%-6d", 13) ;`	`13␣␣␣␣`
`printf("%06d", 13) ;`	`000013`
`printf("%6s", "ciao") ;`	`␣␣ciao`
`printf("%-6s", "ciao") ;`	`ciao␣␣`
`printf("%d", 13) ;`	`13`
`printf("%d", -13) ;`	`-13`
`printf("%+d", 13) ;`	`+13`
`printf("%+d", -13) ;`	`-13`
`printf("%␣d", 13) ;`	`␣13`
`printf("%␣d", -13) ;`	`-13`

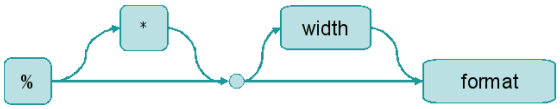
Come ragiona `scanf`

- Controlla un carattere alla volta nella stringa di formato, "consumando" via via i caratteri che trova nell'input (tastiera per `scanf`, stringa per `sscanf`, file per `fscanf`).
- Se l'input è vuoto (file in condizione di end-of-file per `fscanf`, oppure stringa vuota per `sscanf`), la funzione ritorna `-1`.
- Se nella stringa di formato vi è:
  - un qualsiasi carattere di spaziatura (secondo la definizione di `isspace()`: spazio, tab `\t`, a capo `\n`), allora `scanf` "consuma" tutti gli eventuali caratteri di spaziatura che incontra, fino al primo carattere non-di-spaziatura, il quale non viene ancora consumato.
  - un carattere non di spaziatura, diverso da `%`, allora `scanf` si aspetta che ci sia esattamente *quel* carattere nell'input. Se c'è, esso viene consumato, altrimenti `scanf` si ferma.

  - il carattere `%`, che viene interpretato in funzione del tipo di dato che deve essere letto.
    * Se il comando *non* è `%c`, allora vengono innanzitutto "consumati" e scartati eventuali caratteri di spaziatura (come se vi fosse uno spazio nella stringa di formato).
    * Vengono quindi consumati i caratteri successivi, fino a che non si incontra un primo carattere di spaziatura oppure un primo carattere che non è compatibile con il formato specificato.
    * I caratteri validi incontrati vengono convertiti nel tipo di dato opportuno, e memorizzati nella variabile specificata.
    * Se il primo carattere che si era incontrato era già un carattere non compatibile, allora in dato non può essere letto e `scanf` si ferma.

- Viene ritornato al chiamante il numero di variabili lette con successo.

Formato di input avanzato

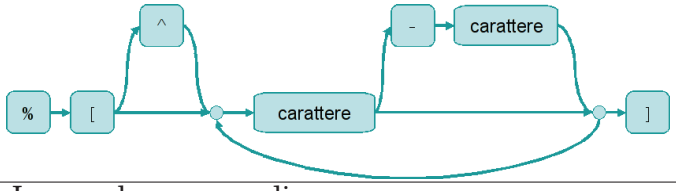


- width: numero massimo di caratteri letti in questa conversione
- `*`: legge un dato del formato specificato, ma non lo memorizza in alcuna variabile

Formato di input avanzato – esempi

Istruzione	Input	Risultato
`scanf("%d", &x) ;`	`134xyz`	`x = 134`
`scanf("%2d", &x) ;`	`134xyz`	`x = 13`
`scanf("%s", v) ;`	`134xyz`	`v = "134xyz"`
`scanf("%2s", v) ;`	`134xyz`	`v = "13"`
`scanf("%d␣%s", &x, v) ;`	`10␣Pippo`	`x = 10`, `v = "Pippo"`
`scanf("%s", v) ;`	`10␣Pippo`	`x` invariato, `v = "10"`
`scanf("%*d␣%s", v) ;`	`10␣Pippo`	`x` invariato, `v = "Pippo"`

Pattern `%[...]`



`"%[r]"`	Legge solo sequenze di `'r'`
`"%[abcABC]"`	Legge sequenze composte da `'a'`, `'b'`, `'c'`, `'A'`, `'B'`, `'C'`, in qualsiasi ordine e di qualsiasi lunghezza
`"%[a-cA-C]"`	Idem come sopra
`"%[a-zA-Z]"`	Sequenze di lettere alfabetiche
`"%[0-9]"`	Sequenze di cifre numeriche
`"%[a-zA-Z0-9]"`	Sequenze alfanumeriche
`"%[^x]"`	Qualunque sequenza che non contiene `'x'`
`"%[^\n]"`	Legge fino a file riga (escluso)
`"%[^,;.!?␣]"`	Si ferma alla punteggiatura o spazio
`"%[^␣\t\n]"`	Equivalente al "classico" `"%s"`

Stampa messaggi di errore

```
int myerror(const char *message)
{
    fputs( message, stderr ) ;
    exit(1) ;
}
```

18.10 Gestione dei file in C

Funzioni principali

`f = fopen(nome, modo);`	Apertura di un file
`fclose(f);`	Chiusura di un file
`if ( feof(f) ) ...`	Verifica se è stata raggiunta la fine del file (in lettura)
`ch = fgetc(f) ;`	Leggi un singolo carattere
`fgets(s, LUN, f) ;`	Leggi un'intera riga
`fputc(ch, f) ;`	Stampa un singolo carattere
`fputs(s, f) ;`	Stampa un'intera riga
`fprintf(f, "%d\n", i) ;`	Stampa in modo formattato
`fscanf(f, "%d\n", &i) ;`	NON USARE MAI. Usare `fgets`+`sscanf`

Modi di accesso al file

`"r"`	Lettura di un file esistente
`"w"`	Scrittura da zero di un file nuovo o ri-scrittura di un file esistente
`"a"`	Scrittura da zero di un file nuovo o aggiunta in coda ad un file esistente

Lettura di un file di testo per caratteri

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

int main(int argc, char *argv[])
{
  FILE * f ;
  int ch ;

  f = fopen( "nomefile", "r" ) ;
  if( f == NULL )
  {
    printf("Impossibile␣aprire␣il␣file\n");
    exit(1) ;
  }

  while( ( ch = fgetc(f) ) != EOF )
  {
     /* elabora il carattere ch */
  }

  fclose(f) ;

  exit(0) ;
}
```

Lettura di un file di testo per righe

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

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

```
{
  const int LUN = 200 ;

  FILE * f ;
  int riga[LUN+1] ;

  f = fopen( "nomefile", "r" ) ;
  if( f == NULL )
  {
    printf("Impossibile␣aprire␣il␣file\n");
    exit(1) ;
  }

  while( fgets(riga, MAX, f) != NULL )
  {
     /* elabora la riga contenuta in riga[] */
  }

  fclose(f) ;

  exit(0) ;
}
```