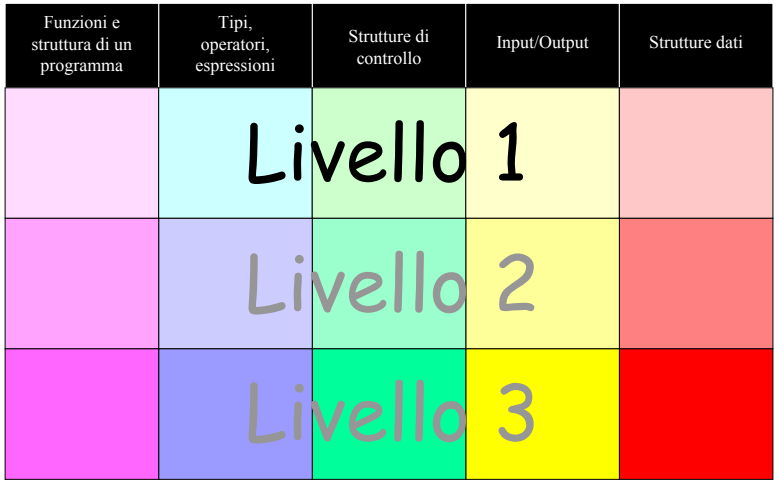
ROADMAP



FUNZIONI E STRUTTURA DI UN PROGRAMMA

- struttura di un programma "C"
- direttive per il compilatore
- dichiarazioni

STRUTTURA DI UN PROGRAMMA 'C'

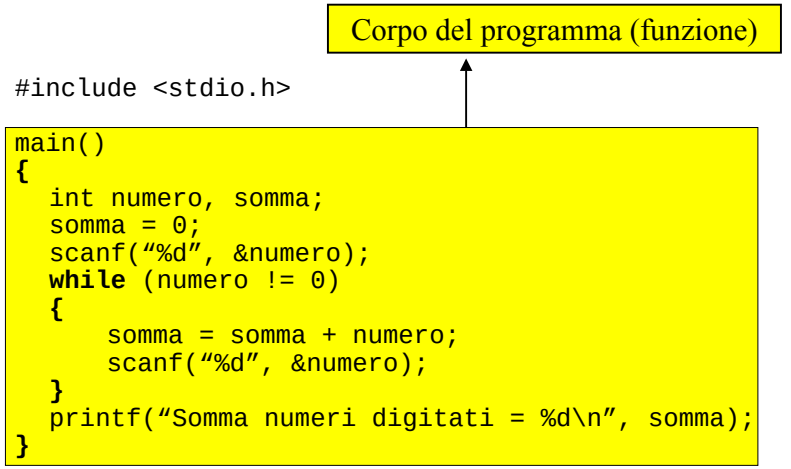
```
                    /* Programma SommaSequenza */

#include <stdio.h>

main()
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```


STRUTTURA DI UN PROGRAMMA 'C'

```
                    /* Programma SommaSequenza */

#include <stdio.h>

main()
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```

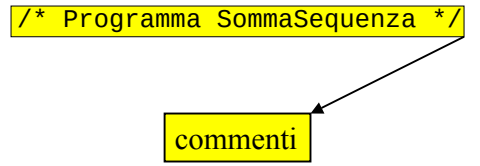
Direttive per il compilatore

STRUTTURA DI UN PROGRAMMA 'C'




```
#include <stdio.h>

main()
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```

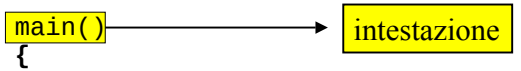

STRUTTURA DI UN PROGRAMMA 'C'




```
#include <stdio.h>

main()
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```


STRUTTURA DI UN PROGRAMMA 'C'

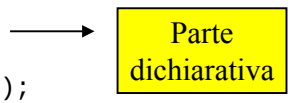
```
/* Programma SommaSequenza */

#include <stdio.h>

main()
```




```
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```


STRUTTURA DI UN PROGRAMMA 'C'

```
/* Programma SommaSequenza */

#include <stdio.h>

main()
{
   int numero, somma;
   somma = 0;
   scanf("%d", &numero);
   while (numero != 0)
   {
       somma = somma + numero;
       scanf("%d", &numero);
   }
   printf("Somma numeri digitati = %d\n", somma);
}
```



STRUTTURA DI UN PROGRAMMA 'C'

```
                                    /* Programma SommaSequenza */
#include <stdio.h>                                  Parte
                                                    esecutiva
main()
{
  int numero, somma;
  somma = 0;
  scanf("%d", &numero);
  while (numero != 0)
  {
      somma = somma + numero;
      scanf("%d", &numero);
  }
  printf("Somma numeri digitati = %d\n", somma);
}
```


STRUTTURA DI UN PROGRAMMA 'C'

```
                                    /* Programma SommaSequenza */
#include <stdio.h>                                  istruzioni

main()
{
  int numero, somma;
  somma = 0;
  scanf("%d", &numero);
  while (numero != 0)
  {
      somma = somma + numero;
      scanf("%d", &numero);
  }
  printf("Somma numeri digitati = %d\n", somma);
}
```


DIRETTIVE PER IL COMPILATORE

- Indicano al compilatore di includere le informazioni relative ad altri programmi gia` scritti dall'utente o a **funzioni di libreria** (funzioni predefinite - sottoprogrammi - specializzate per la realizzazione di particolari operazioni)
- Es. l'istruzione `#include <stdio.h>` copia il contenuto del file `stdio.h` nel programma
  - `stdio.h` contiene, tra gli altri, riferimenti a funzioni di gestione dell'I/O
- Il significato di queste istruzioni e il loro uso sara` approfondito nel seguito

CORPO DEL PROGRAMMA

- Un programma "C" e` composto da un'**intestazione** e da una **sequenza di istruzioni** racchiusa tra i simboli **{** e **}**
  - L'intestazione e` costituita dall'identificare predefinito **`main`** seguito da una coppia di parentesi **( )**
    - **`main()`** rappresenta la **funzione principale** del programma
    - l'esecuzione inizia a partire dalla prima istruzione della parte esecutiva della funzione **`main`**
  - Le istruzioni terminano con il simbolo **`;`**

CORPO DEL PROGRAMMA

- Il blocco di istruzioni racchiuse tra **`{ }`** comprende:
  - la **parte dichiarativa**

    elenca tutti gli elementi che fanno parte del programma, con le loro principali caratteristiche
  - la **parte esecutiva**

    consiste in una successione di istruzioni
    - di assegnamento
    - di ingresso/uscita
    - condizionali
    - iterative

DICHIARAZIONE DI VARIABILI

- Ha lo scopo di:
  - elencare tutte le variabili che saranno usate nella parte esecutiva e di attribuire a ognuna di esse un **tipo**

    ☞ specificarne le caratteristiche che ne regolano l'uso
- Consiste in:
  - uno **specificatore di tipo**
  - seguito da una lista di uno o più **identificatori di variabili** separati da una virgola e terminata con **`;`**

TIPI, OPERATORI ED ESPRESSIONI

- tipi semplici predefiniti
- conversione tra tipi
- costanti
- macro

TIPI SEMPLICI PREDEFINITI

- Per il momento faremo uso solo di un numero ristretto di tipi di dati:

`int i,j;`	un intero (riflette, in genere) l'ampiezza degli interi sulla macchina utilizzata
`float x,y;`	approssimazione di un numero reale in singola precisione
`double var1;`	approssimazione di un numero reale in doppia precisione
`char simb;`	un singolo byte, in grado di rappresentare uno qualsiasi dei caratteri disponibili

TIPI SEMPLICI PREDEFINITI

Tipo	Allocazione memoria	Accuratezza	Insieme di rappresentazione
int	2 byte	-	$\{-2^{15}, 2^{15}-1\}$
float	4 byte	6 cifre dec.	$\pm\{10^{-38}, 10^{+38}\}$
double	8 byte	15 cifre dec.	$\pm\{10^{-308}, 10^{+308}\}$
char	1 byte	-	{0, 255}*

*** codice numerico dell'insieme dei caratteri ASCII**

N.B. i valori nella tabella sono indicativi e possono variare da macchina a macchina

INSIEME DEI CARATTERI ASCII 1/2

Non-Printing Characters					Printing Characters								
Name	Ctrl char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
null	ctrl-@	0	00	NUL	32	20	Space	64	40	@	96	60	`
start of heading	ctrl-A	1	01	SOH	33	21	!	65	41	A	97	61	a
start of text	ctrl-B	2	02	STX	34	22	"	66	42	B	98	62	b
end of text	ctrl-C	3	03	ETX	35	23	#	67	43	C	99	63	c
end of xmit	ctrl-D	4	04	EOT	36	24	$	68	44	D	100	64	d
enquiry	ctrl-E	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	ctrl-F	6	06	ACK	38	26	&	70	46	F	102	66	f
bell	ctrl-G	7	07	BEL	39	27	'	71	47	G	103	67	g
backspace	ctrl-H	8	08	BS	40	28	(	72	48	H	104	68	h
horizontal tab	ctrl-I	9	09	HT	41	29	)	73	49	I	105	69	i
line feed	ctrl-J	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
vertical tab	ctrl-K	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
form feed	ctrl-L	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
carriage feed	ctrl-M	13	0D	CR	45	2D	-	77	4D	M	109	6D	m
shift out	ctrl-N	14	0E	SO	46	2E	.	78	4E	N	110	6E	n
shift in	ctrl-O	15	0F	SI	47	2F	/	79	4F	O	111	6F	o

INSIEME DEI CARATTERI ASCII 2/2

Non-Printing Characters					Printing Characters								
Name	Ctrl char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
data line escape	ctrl-P	16	10	DLE	48	30	0	80	50	P	112	70	p
device control 1	ctrl-Q	17	11	DC1	49	31	1	81	51	Q	113	71	q
device control 2	ctrl-R	18	12	DC2	50	32	2	82	52	R	114	72	r
device control 3	ctrl-S	19	13	DC3	51	33	3	83	53	S	115	73	s
device control 4	ctrl-T	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	ctrl-U	21	15	NAK	53	35	5	85	55	U	117	75	u
synchronous idel	ctrl-V	22	16	SYN	54	36	6	86	56	V	118	76	v
end of xmit block	ctrl-W	23	17	ETB	55	37	7	87	57	W	119	77	w
cancel	ctrl-X	24	18	CAN	56	38	8	88	58	X	120	78	x
end of medium	ctrl-Y	25	19	EM	57	39	9	89	59	Y	121	79	y
substitute	ctrl-Z	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
escape	ctrl-[	27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
file separator	ctrl-\	28	1C	FS	60	3C	<	92	5C	\	124	7C	|
group separator	ctrl-]	29	1D	GS	61	3D	=	93	5D	]	125	7D	}
record separator	ctrl-^	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
unit separator	ctrl-_	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

OPERAZIONI SU "float" e "double"

=	Assegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione (a risultato reale)
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Relazione "minore di"
>	Relazione "maggiore di"
<=	Relazione "minore o uguale a"
>=	Relazione "maggiore o uguale a"

OPERAZIONI SU "int"

`=`	Assegnamento di un valore **`int`** a una variabile **`int`**
`+`	Somma (tra **`int`** ha come risultato un **`int`**)
`-`	Sottrazione (tra **`int`** ha come risultato un **`int`**)
`*`	Moltiplicazione (tra **`int`** ha come risultato un **`int`**)
`/`	Divisione con troncamento parte non intera (risul. **`int`**)
`%`	Resto della divisione intera
`==`	Relazione di uguaglianza
`!=`	Relazione di diversità
`<`	Relazione "minore di"
`>`	Relazione "maggiore di"
`<=`	Relazione "minore o uguale a"
`>=`	Relazione "maggiore o uguale a"

OPERAZIONI SU "char"

`=`	Assegnamento di un valore **`char`** a una variabile **`char`**
`+`	Somma (tra **`char`** ha come risultato un **`char`**)
`-`	Sottrazione (tra **`char`** ha come risultato un **`char`**)
`*`	Moltiplicazione (tra **`char`** ha come risultato un **`char`**)
`/`	Divisione con troncamento parte non intera (risul. **`char`**)
`%`	Resto della divisione intera
`==`	Relazione di uguaglianza
`!=`	Relazione di diversità
`<`	Relazione "minore di"
`>`	Relazione "maggiore di"
`<=`	Relazione "minore o uguale a"
`>=`	Relazione "maggiore o uguale a"

OSSERVAZIONI

- La comunanza di operazioni tra **`char`** e **`int`** e` una naturale conseguenza della rappresentazione dei caratteri tramite numero intero
- Operazioni di relazione su due valori di tipo **`float`** o **`double`** produce come risultato un valore intero pari a 0 se la relazione non e` verificata, e un valore intero diverso da 0 se la relazione e` verificata

COMPATIBILITA` TRA TIPI

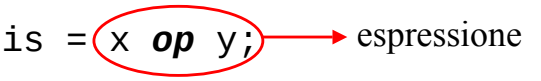
- Ha senso l'assegnamento del valore di una variabile dichiarata come **`int`** a una variabile dichiarata come **`float`**?
- Che cosa succede in fase di esecuzione?
- Come viene trattato il risultato ottenuto?

☞ **Il "C" risponde con la tipizzazione forte e precise regole per la conversione implicita ed esplicita tra tipi**

TIPIZZAZIONE FORTE

- Le regole per la gestione del tipo delle variabili sono tutte verificabili a tempo della compilazione senza richiedere l'esecuzione del programma
  - Grazie alle dichiarazioni di tipi e variabili, e` normalmente possibile controllare durante la compilazione se un'istruzione e` compatibile con le variabili coinvolte
  - Eventuali violazioni alle regole possono essere individuate preliminarmente all'esecuzione del programma, aumentandone la sicurezza e risparmiando sui costi per la sua verifica

REGOLE DI CONVERSIONE IMPLICITA



- Il tipo del risultato dell'espressione viene convertito in quello della variabile a cui è assegnato
- L'espressione viene valutata
  - convertendo ogni variabile di tipo **`char`** in una variabile di tipo **`int`**
  - usando le operazioni definite sull'insieme del tipo superiore nella gerarchia:

    `int < float < double`
  - Il risultato avrà tipo uguale a quello di più alto livello gerarchico

REGOLE DI CONVERSIONE IMPLICITA

- Esempio:

```
double d, step;
float res;
int i,k;
```

```
d=i;
```

- provoca una temporanea conversione del valore dell'intero `i` a **`double`** e successivamente l'assegnamento di tale valore **`double`** a `d`

REGOLE DI CONVERSIONE IMPLICITA

- Esempio:

```
double d, step;
float res;
int i,k;
```

```
i=d;
```

- comporta una perdita di informazione. Il valore di `d` subisce infatti un troncamento alla parte intera con perdita della parte decimale.

REGOLE DI CONVERSIONE IMPLICITA

- Esempio:

```
double d, step;
float res;
int i,k;
```

Attenzione al "punto decimale" !! divisione tra numeri reali !!

```
res=(1.0/2.0)*(i-k)*step;
```

- Il risultato della sottrazione tra interi e` temporaneamente convertito in **`double`**

CONVERSIONE ESPLICITA

- In qualsiasi espressione e` possibile forzare particolari conversioni tramite l'operatore **cast**:

  `(nome-del-tipo) espressione`

- l'*espressione* viene convertita nel tipo specificato con le conseguenze descritte
- Un **cast** equivale ad assegnare l'*espressione* ad una variabile del tipo specificato che viene poi utilizzata al posto dell'intera costruzione

Es. `A = 1/(float) n;`

DICHIARAZIONI DI COSTANTI

- Associano permanentemente un valore a un identificatore
- Consiste in:
  - la parola chiave **`const`**
  - lo **specificatore di tipo** della costante
  - l'**identificatore** della costante
  - il simbolo **=**
  - il **valore** della costante (intero, num. reale, carattere, ...)
  - il "terminatore" **;**

DICHIARAZIONI DI COSTANTI

- Esempi di dichiarazioni di costanti sono i seguenti:

```
const  float     PiGreco=3.14;
const  float     PiGreco=3.1415,e=2.718;
const  int       N=100, M=1000;
const  char      car1='A', car2='B';
```

- Permettono:
  - di trattare in modo simbolico l'informazione che rappresentano → aiutano la lettura del programma
  - una buona *parametrizzazione* dei programmi, → li rendono facilmente riutilizzabili anche al cambiare di certe circostanze esterne

DEFINIZIONE DI MACRO

- Le **macro** consentono l'associazione di un identificatore simbolico ad un valore costante o, piu` in generale, ad un'espressione
- La definizione di una macro ha la forma

  `#define` *`nome testo_da_sostituire`*
  - tutte le successive occorrenze di *`nome`* sono sostituite con il *`testo_da_sostituire`*
- Come le costanti, le macro:
  - aiutano la lettura del programma
  - lo rendono facilmente riutilizzabile anche al cambiare delle circostanze esterne

DEFINIZIONE DI COSTANTI MEDIANTE `MACRO'

- A differenza delle dichiarazioni di costanti le **macro** rappresentano delle direttive per il preprocessore C
  - non viene riservato spazio di memoria
  - la sostituzione del valore costante al posto dell'identificatore viene eseguita dal preprocessore nel tempo della compilazione
    - ad ogni occorrenza, la macro viene espansa in codice di linea

ISTRUZIONI DI INPUT/OUTPUT

- `printf()`
- `scanf()`

ISTRUZIONI DI INPUT/OUTPUT

- Si sfruttano un insieme di **funzioni predefinite**, appartenenti alla *Standard Library* del linguaggio "C"

  `printf()` `scanf()`
- Tali funzioni realizzano, quando richiamate all'interno di un programma, rispettivamente la scrittura sullo Standard Output (`stdout`) e la lettura dallo Standard Input (`stdin`)
- Occorre includere la direttiva:

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

LA FUNZIONE printf

- Funzione di uso generale per la stampa di output formattato
- Richiede la specifica di una **stringa** di caratteri da stampare (compresi quelli speciali di tabulazione)

```
printf("stringa");
```

- Piu` in generale, puo` essere necessario passare alla funzione una **lista di elementi** da inserire nella stringa di stampa (valore di variabili, costanti, o espressioni composte con variabili e costanti).

LA FUNZIONE printf

```
printf("stringa", el1,. . .,elN);
```

- La stringa contiene **caratteri di conversione** (o **di formato**) preceduti dal simbolo **%**

```
%d   %f   %c    %s  ...
```

  - % indica il punto in cui devono essere sostituiti, in ordine corrispondente, gli elementi elencati)
  - i caratteri immediatamente successivi ad un % indicano la forma nella quale l'argomento deve essere stampato

%d → intero	%f → numero reale
%c → carattere	%s → stringa di caratteri

ESEMPIO 1

```
                        /* Primo programma C */
#include <stdio.h>
main()
{
  printf("Ciao mondo!");
}
```

```
> Ciao mondo!
```


ESEMPIO 1bis

```
                        /* Primo programma C */
#include <stdio.h>
main()
{
  printf("Ciao mondo!\n");
}
```

```
> Ciao mondo!
>
```

ESEMPIO 2

```
#include <stdio.h>
main()
{
   . . .
   printf("Questo programma e` stato scritto da
   %s\n",nome);
   printf("in data %d/%d/%d\n",gg,mm,aa);
}
```

> Questo programma e` stato scritto da Giorgio
>
> in data 12/5/2000
>
> ▯

LA FUNZIONE scanf

- Funzione di uso generale per la lettura di input formattato

  `scanf("stringa", &el1,…,&elN);`
- Richiede la specifica di
  - una stringa di controllo (**caratteri di formato** preceduti dal simbolo **%**)
  - una lista di elementi da leggere (nomi delle variabili a cui si desidera associare un valore, preceduti dall'operatore **&**)

LA FUNZIONE scanf

- La stringa di controllo consente di specificare come devono essere interpretati i caratteri letti da tastiera

  `scanf("%c%d%f", &alfa, &i, &num);`

`%d` → intero	`%f` → numero reale
`%c` → carattere	`%s` → stringa di caratteri

- `&num` va letto come indirizzo della variabile `num`

LA FUNZIONE scanf

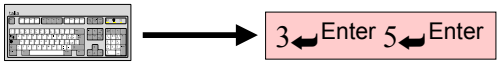
`scanf("%c%c%d%f", &c1, &c2, &i, &x);`

Inserendo i dati: AB 3 7.345

- la variabile `c1` (di tipo **`char`**) assume il valore A
  - il carattere A viene posizionato da `scanf` nella cella di memoria il cui indirizzo e` quello della variabile `c1`
- la variabile `c2` (di tipo **`char`**) assume il valore B
- la variabile `i` (di tipo **`int`**) assume valore 3
- la variabile `x` (di tipo **`float`**) assume valore 7.345

ESEMPIO 3

```
#include <stdio.h>
main()
{
   int a, b, somma;
   scanf("%d%d", &a, &b);
   somma = a+b;
   printf("La somma di a+b e`:\n%d", somma);
}
```




```
> La somma di a+b e`:
> 8
```

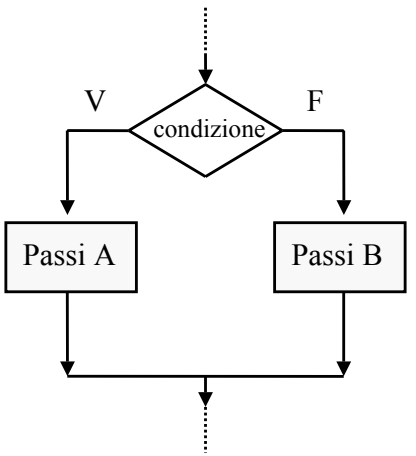

STRUTTURE DI CONTROLLO

- if-else
- for
- while
- do-while

L'ISTRUZIONE DI SELEZIONE if-else

- Consente di eseguire in alternativa due diverse sequenze di istruzioni sulla base del valore di verita` di una condizione
- E' costituita da:
  - la parola chiave **if**
  - una condizione racchiusa tra **( )**
  - una prima sequenza di istruzioni racchiuse tra **{ }**
  - la parola chiave **else**
  - una seconda sequenza di istruzioni racchiuse tra **{ }**

L'ISTRUZIONE DI SELEZIONE if-else




```
if (condizione)
{
 Passi A;
}
else
{
 Passi B;
}
```

L'ISTRUZIONE DI SELEZIONE if-else

- La parte di **`else`** è opzionale
- Più selezioni possono essere innestate una dentro l'altra (*alternative multiple*)
- La mancanza di un **`else`** all'interno di una sequenza di **`if`** innestati comporta un'ambiguità

```
if (n > 0)
  if (a > b)
      z = a;
else
      z = b;
```

- l'ambiguita` e` risolta associando ogni **`else`** all'**`if`** più interno che ne è privo

ESEMPIO 4

```
if  (a >= PrimoLimite)
     sum = 1 + (b - PrimoLimite)/100;
else if (a >= SecondoLimite)
     sum = 2 + (b - SecondoLimite)/100;
else if (a >= TerzoLimite)
     sum = 3 + (b - TerzoLimite)/100;
else
     sum = 0;
```

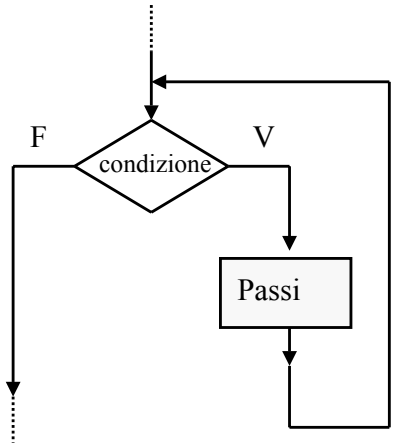

NOTE

- E' il modo più generale per realizzare una scelta plurima
- Le espressioni vengono valutate nell'ordine in cui si presentano
  - se una di esse risulta vera, l'istruzione ad essa associata viene eseguita e ciò termina l'intera catena
  - l'ultimo **`else`** gestisce il caso "di default", eseguito quando nessuna delle espressioni precedenti risulta vera

L'ISTRUZIONE CICLICA while

- **Ciclo condizionato**: ripetizione dell'esecuzione di una sequenza di istruzioni **finche`** una certa condizione e` verificata
- E' costituita da:
  - la parola chiave **`while`**
  - una condizione racchiusa tra **`( )`**
  - una sequenza di istruzioni tra **`{ }`** *(corpo del ciclo)*
- L'esecuzione di un'istruzione **`while`** consiste in primo luogo nella valutazione della condizione

L'ISTRUZIONE CICLICA while




```
while (condizione)
{
 Passi;
}
```

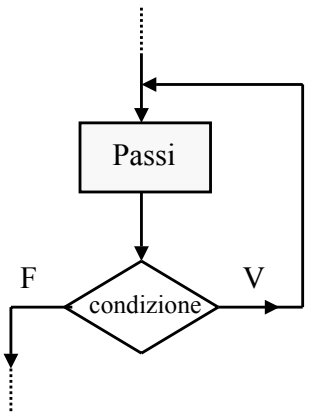

ESEMPIO 5

```
#include <stdio.h>
main()
{
  int somma, numero;
  somma = 0;
  scanf("%d",&numero);
  while (numero != 0)
 {
   somma=somma+numero;
   scanf("%d",&numero);
 }
 printf("Somma --> %d\n\n", somma);
}
```


L'ISTRUZIONE CICLICA do-while

- **Ciclo condizionato**: alternativa al ciclo **`while`**
- E' costituita da:
  - la parola chiave **`do`**
  - una sequenza di istruzioni tra **`{ }`** *(corpo del ciclo)*
  - la parola chiave **`while`**
  - una condizione racchiusa tra **`( )`** e terminata con **`;`**
- **Il corpo del ciclo viene comunque eseguito almeno una volta** e ripetuto finche` la condizione indicata dopo **`while`** diventa falsa

L'ISTRUZIONE CICLICA do-while



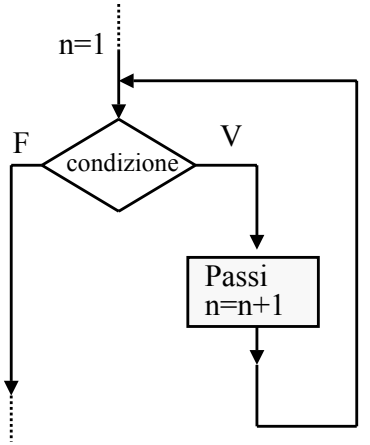

```
do
{
 Passi;
}
while (condizione);
```

In alcuni casi e` preferibile a quella del ciclo **`while`**

L'ISTRUZIONE CICLICA for

- **Ciclo "a conteggio"**: ripetizione dell'esecuzione di una sequenza di istruzioni **un numero precisato di volte**
- E' costituita da:
  - la parola chiave **`for`**
  - una coppia **`( )`** comprendente *tre* espressioni separate una dall'altra da **`;`** (*`expr1; expr2; expr3`*)
    - *`expr1`* : inizializzazione della variabile di conteggio
    - *`expr2`* : condizione che determina l'esecuzione del corpo del ciclo
    - *`expr3`* : incremento o decremento della variabile di conteggio. Viene eseguita come **ultima istruzione del corpo del ciclo**
  - una sequenza di istruzioni tra **`{ }`** *(corpo del ciclo)*

L'ISTRUZIONE CICLICA for




```
for (n=1;n<MAX;n=n+1)
{
 Passi;
}
```


IL CONFRONTO TRA for E while

while

```
VarDiConteggio = ValoreIniziale;
while (VarDiConteggio <= ValoreFinale)
{
  [sequenza di istruzioni da ripetere];
  VarDiConteggio = VarDiConteggio +1;
}                /* costrutto while */
```

for

```
for (VarDiConteggio = ValoreIniziale;
    VarDiConteggio <= ValoreFinale;
    VarDiConteggio = VarDiConteggio +1)
{
  [sequenza di istruzioni da ripetere];
}                /* costrutto for */
```


IL CONFRONTO TRA for E while

- Strutture quali **`while`** e **`do-while`** sono concepite per cicli condizionali nei quali il termine del ciclo non e` determinato da un semplice conteggio, bensi` da una condizione piu` generale

NOTA

- Operatore di autoincremento **++**

```
VarDiConteggio = VarDiConteggio +1;
```

puo` essere espressa piu` sinteticamente nel modo seguente:

```
VarDiConteggio++;
```

- Operatore di autodecremento **--**

(uso analogo)

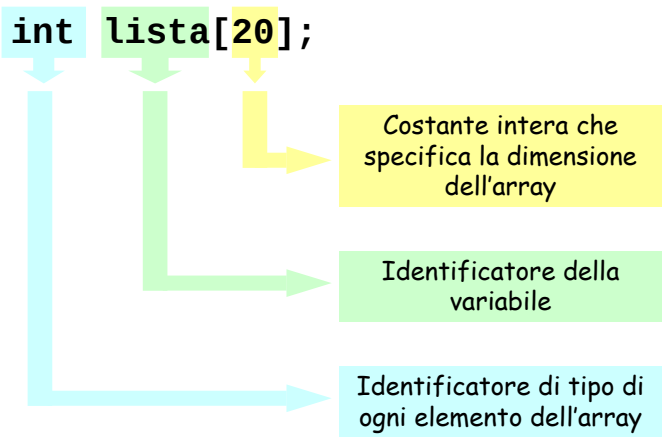
STRUTTURE DATI

- vettori (array)
- matrici

VETTORI

- Il **vettore** o **array** rappresenta una sequenza di celle di memoria
  - **consecutive**
  - **omogenee** (contenenti dati dello stesso tipo)
- La sua dichiarazione deve specificare:
  - il **tipo** degli elementi del vettore
  - il **nome della variabile** associata al vettore
  - la **dimensione** del vettore, ovvero il numero dei suoi elementi

VETTORI



VETTORI

- Il singolo elemento di un vettore viene identificato mediante il nome della variabile e un **indice**
  - numero **intero** ∈ [0, *dim* - 1]
    dove *dim* = dimensione del vettore
  - indica il numero d'ordine della singola cella all'interno della sequenza
  - può essere denotato mediante il contenuto di un'altra variabile o il valore di un'espressione
- Il nome della variabile associata a un vettore rappresenta l'indirizzo della cella di memoria che contiene il primo elemento del vettore

ESEMPIO 6

```
#include <stdio.h>
main()
{
  int a[5], b[5], i;
  for (i=0;i<5;i++)
     a[i]=i+1;
  for (i=0;i<5;i++)
     b[i]=a[4-i];
}
```


ESEMPIO 7

```
#include <stdio.h>
main()
{
  int a[5], i;
  char b[5];
  a[0]=1; a[1]=3; a[2]=2; a[3]=0; a[4]=4;
  b[0]='o'; b[1]='c'; b[2]='a'; b[3]='i';
  b[4]='!';
  for (i=0;i<5;i++)
     printf("%c",b[a[i]]);
}
```


NOTE

- In alternativa a:

```
a[0]=1; a[1]=3; a[2]=2; a[3]=0; a[4]=4;
b[0]='o'; b[1]='c'; b[2]='a'; b[3]='i'; b[4]='!';
```

- L'**inizializzazione** dei vettori può anche avvenire al momento della loro dichiarazione:

```
int a[5] = {1, 3, 2, 0, 4};
char b[5] = "ocai!";
```

- L'inizializzazione di un vettore è una **lista di inizializzatori** racchiusa tra parentesi { } (o una **stringa**)

NOTE

- Se la dimensione del vettore non è indicata, ciò che la determina è il numero degli inizializzatori (o dei caratteri della stringa)

```
int a[] = {1, 3, 2, 0, 4};
char b[] = "ocai!";
```

- Se il vettore ha ampiezza fissata, il numero degli inizializzatori (o dei caratteri della stringa) non deve superare quello degli elementi del vettore stesso

VETTORI

- Gli elementi di un vettore occupano in memoria un numero di parole che dipende dal loro tipo
- E' possibile utilizzare l'operatore **`sizeof`** per ottenere il numero di byte occupati da
  - ciascun elemento di un vettore
  - da un vettore nel suo complesso
- Se ad es., `int a[5];`
  allora `sizeof(a[2])` → 2 *(byte)*
  `sizeof(a)` → 5x2=10 *(byte)*

VETTORI BIDIMENSIONALI

- Un vettore bidimensionale è un vettore ad una dimensione in cui **ogni elemento è un vettore**
- Gli indici devono quindi essere scritti come:

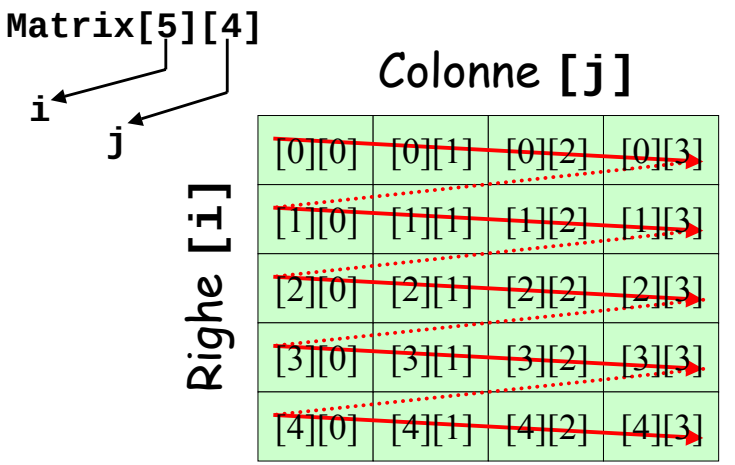
```
mat[i][j]   /*[riga] [colonna]*/
```

e non come

```
mat[i,j]    /*SBAGLIATO*/
```

- Gli elementi vengono **memorizzati per righe**
  - l'indice a destra (la colonna) varia più velocemente mano a mano che si accede agli elementi nell'ordine in cui sono memorizzati

VETTORI BIDIMENSIONALI



ESERCIZIO

- *Si supponga di dover leggere una sequenza di caratteri, tralasciando gli "a capo" e terminata dal carattere % e di doverla memorizzare, ultimo carattere incluso, nell'array* `Testo` *di lunghezza finita predefinita:*

```
char Testo[LunghMax]
```


RISOLUZIONE

```
#include <stdio.h>
#define LunghMax 500
main()
{
  int Contatore;
  char dato, Testo[LunghMax];
  . . . parte esecutiva . . .
}
```

Definizione di una **macro** per `LunghMax`

- Vediamo due soluzioni:
  - mediante il ciclo `while`
  - mediante il ciclo `do-while`

RISOLUZIONE (ciclo while)

```
Contatore = 0;
scanf("%c", &dato);
while ( (dato == '\n') || (dato ==' \r') )
  scanf("%c", &dato);
Testo[Contatore] = dato;
while ( (dato != '%') && (Contatore < LunghMax) ){
  Contatore = Contatore +1;
  scanf("%c", &dato);
  while ( (dato == '\n') || (dato == '\r') )
    scanf("%c", &dato);
  Testo[Contatore] = dato; }
if ( (Contatore == LunghMax) && (dato != '%') )
   printf("La sequenza e` troppo lunga\n");
```


RISOLUZIONE (ciclo do-while)

```
Contatore = 0;
do
{
  do
    scanf("%c", &dato);
  while ( (dato=='\n') || (dato=='\r') );
  Testo[Contatore] = dato;
  Contatore = Contatore +1;
}
while ( (dato != '%') && (Contatore < LunghMax) );
if ( (Contatore == LunghMax) && (dato != '%') )
   printf("La sequenza e` troppo lunga\n");
```

NOTE

- Il “valore carattere” viene indicato tra apici **‘ ’**
- **Caratteri di controllo**
  - non sono legati ad un vero e proprio simbolo grafico, ma all’esecuzione di un’operazione correlata alla visualizzazione dei dati

  Es.:

  \n = “newline”
  \b = “backspace”
  \t = “horizontal tab”
  \r = “carriage return”