

Costrutti di controllo

Scegliere: istruzione **if-else**

Sintassi:

if (*exp*) *istr1*

else *istr2*

- *exp* è un'espressione con un valore booleano
- *istr1*, *istr2* sono un'unica istruzione o un blocco {...}

Semantica:

1. viene prima valutata *exp*
2. se è vera viene eseguita *istr1*
3. altrimenti (ovvero se è falsa) viene eseguita *istr2*

Scegliere: istruzione **if**

Sintassi:

if (*exp*) *istr1*

~~**else** *istr2*~~

Questa parte può essere omessa se non dobbiamo fare niente nel caso che la condizione exp sia falsa

Semantica:

1. viene prima valutata *exp*
2. se è vera viene eseguita *istr1*
3. Altrimenti non si esegue niente

if/if-else: esempio

```
/* calcolo il massimo fra tre reali  
Leggiamo 3 reali da stdininput stabiliamo  
il piu' grande e stampiamolo su stdout  
...
```

Algoritmo ?

Varibili ?

Codifica ?

```
*/  
#include <stdio.h>  
int main(void) {  
.....  
}
```

if: max fra 3

```
/* una possibile soluzione */
#include <stdio.h>
int main(void) {
    double max, tmp;
    printf("Inserisci il primo valore:");
    scanf("%lf", &max);
    printf("Inserisci il secondo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Inserisci il terzo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

Max fra 3 : output

Esecuzione:

Inserisci la il primo valore:

Se digitiamo 15.1 e ↓ (invio) poi 7.2 e ↓ (invio)

Inserisci il primo valore: 15.1

Inserisci il secondo : 7.2

Inserisci il terzo :

Se infine inseriamo 0 e ↓ (invio)

Inserisci il terzo: 0

Il massimo è: 15.1

if: max fra tre

```
/* una possibile soluzione */
#include <stdio.h>
int main(void) {
    double max, tmp;
    printf("Inserisci il primo valore:");
    scanf("%lf", &max);
    printf("Inserisci il secondo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Inserisci il terzo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

If con una
singola
istruzione

if: max fra tre

```
/* una possibile soluzione */
#include <stdio.h>
int main(void) {
    double max, tmp;
    printf("Inserisci il primo valore:");
    scanf("%lf", &max);
    printf("Inserisci il secondo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Inserisci il terzo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

Operatori di
Confronto



Operatori di confronto

- Dipendono dal tipo, es
 - Reali/interi: `==`, `!=`, `>`, `<`, `>=`, `<=`,
 - Stringhe: `strcmp()`, `strncmp()`
- Gli operatori per reali/interi:
 - Restituiscono $k \neq 0$ (true) se la relazione è vera e 0 (false) altrimenti
 - `strcmp()`, `strncmp()` si comportano in modo diverso
- Lo vedremo in dettaglio per ogni tipo di dato

if/if-else: esempio

```
/* calcolo il massimo fra tre reali
   ed il suo indice
   Leggiamo 3 reali da stdin, stabiliamo
   il piu' grande e stampiamolo su stdout
   ...
```

voglio però anche sapere se il massimo
è il primo, secondo o terzo numero
letto...

```
*/
#include <stdio.h>
int main(void) {
    .....
}
```

```
/* calcoliamo massimo di tre ed indice */
```

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

```
int main(void) {
```

```
double max, tmp;
```

```
int imax = 1, i = 1;
```

```
printf("Inserisci il primo valore:");
```

```
scanf("%lf", &max);
```

```
printf("Inserisci il secondo:");
```

```
scanf("%lf", &tmp); i++;
```

```
if ( max < tmp ) {
```

```
max = tmp;
```

```
imax = i;
```

```
}
```

```
printf("Inserisci il terzo:");
```

```
scanf("%lf", &tmp); i++;
```

```
if ( max < tmp ) {
```

```
max = tmp;
```

```
imax = i;
```

```
}
```

```
printf("Il massimo e' :%f indice:%d \n", max, imax);
```

```
return 0;
```

```
}
```

Contatore per l'indice
Inizializzato ad 1 ed
incrementato ogni
volta che leggo
un nuovo valore


Contiene l'indice
del massimo

```

/* calcoliamo anche l'indice */
#include <stdio.h>
int main(void) {
    double max,tmp;
    int imax = 1, i = 1;
    printf("Inserisci il primo valore:");
    scanf("%lf",&max);
    printf("Inserisci il secondo:");
    scanf("%lf",&tmp); i++;
    if ( max < tmp ) {
        max = tmp;
        imax = i;
    }
    printf("Inserisci il terzo:");
    scanf("%lf",&tmp); i++;
    if ( max < tmp ) {
        max = tmp;
        imax = i;
    }
    printf("Il massimo e' :%f indice:%d \n",max,imax);
    return 0;
}

```

If con più istruzioni
(è necessario il blocco)



if annidati (cascata)

- Servono a programmare una serie di casi mutuamente esclusivi
- Es voglio effettuare una azione diversa per valori x della temperatura nei seguenti intervalli

Valore di x	Stringa stampata
$x > 30$	Molto caldo
$20 < x \leq 30$	Caldo
$10 < x \leq 20$	Gradevole
$x \leq 10$	Freddo

```
/* codifica usando if else in cascata */
```

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

```
int main(void) {
```

```
    double x;
```

```
    printf("Inserisci la temperatura:");
```

```
    scanf("%lf", &x);
```

```
    if ( x > 30 )
```

```
        printf("Molto caldo!\n");
```

```
    else if ( ( 20 < x ) && ( x <= 30 ) )
```

```
        printf("Caldo!\n");
```

```
    else if ( ( 10 < x ) && ( x <= 20 ) )
```

```
        printf("Gradevole!\n");
```

```
    else if ( x <= 10 )
```

```
        printf("Freddo!\n");
```

```
    return 0;
```

```
}
```

```
/* codifica usando if else in cascata */
```

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

```
int main(void) {
```

```
    double x;
```

```
    printf("Inserisci la temperatura:");
```

```
    scanf("%lf", &x);
```

```
    if ( x > 30 )
```

```
        printf("Molto caldo!\n");
```

```
    else if ( ( 20 < x ) && ( x <= 30 ) )
```

```
        printf("Caldo!\n");
```

```
    else if ( ( 10 < x ) && ( x <= 20 ) )
```

```
        printf("Gradevole!\n");
```

```
    else if ( x <= 10 )
```

```
        printf("Freddo!\n");
```

```
    return 0;
```

```
}
```

E' una singola istruzione!

```
/* codifica usando if else in cascata */
```

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

```
int main(void) {
```

```
    double x;
```

```
    printf("Inserisci la temperatura:");
```

```
    scanf("%lf", &x);
```

```
    if ( x > 30 )
```

```
        printf("Molto caldo!\n");
```

```
    else if ( ( 20 < x ) && ( x <= 30 ) )
```

```
        printf("Caldo!\n");
```

```
    else if ( ( 10 < x ) && ( x <= 20 ) )
```

```
        printf("Gradevole!\n");
```

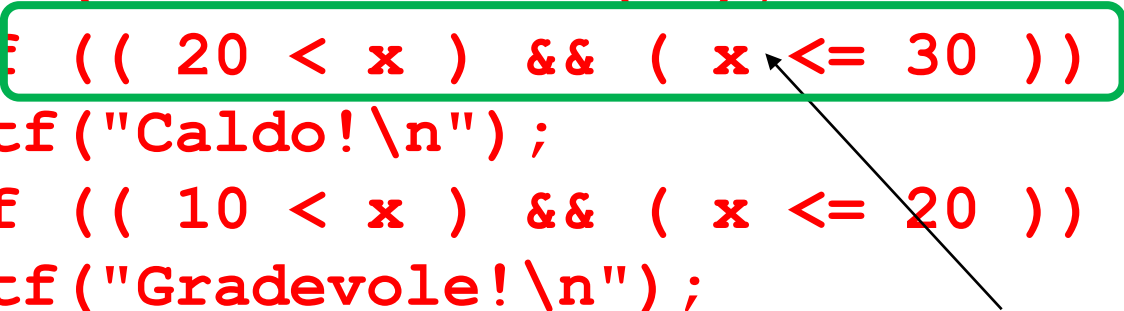
```
    else if ( x <= 10 )
```

```
        printf("Freddo!\n");
```

```
    return 0;
```

```
}
```

Si possono combinare
più confronti nella
Stessa condizione



Ambiguità di if annidati

Cosa viene stampato per $x = 22$?

```
if ( x < 20 )
    if ( x > 15 )
        printf("Calduccio!\n");
else
    printf("Caldino!\n");
```

Interpretazione 1

Cosa viene stampato per $x = 22$?

```
if ( x < 20 )  
    if ( x > 15 )  
        printf("Calduccio!\n");  
else  
    printf("Caldino!\n");
```

Interpretazione 2

Cosa viene stampato per $x = 22$?

```
if ( x < 20 )  
    if ( x > 15 )  
        printf("Calduccio!\n");  
    else  
        printf("Caldino!\n");
```

....in realtà....

- Il ramo else lega sempre con l'**if** più vicino
- Se si riferisce ad uno precedente va inserito in un blocco

Ancora su: Interpretazione 1

Per avere questa semantica dobbiamo inserire il blocco

```
if ( x < 20 )  
    { if ( x > 15 )  
        printf("Calduccio!\n"); }  
else  
    printf("Caldino!\n");
```

istruzione **switch**

- Serve per selezionare vie alternative, sintassi :

```
switch (esp) {  
case valore-1: istruzioni-1  
                break;  
                ...  
case valore-n: istruzioni-n  
                break;  
default : istr-default  
}
```

istruzione `switch`: semantica

- Serve per selezionare vie alternative, semantica:
 1. viene valutata *esp*
 2. viene cercato il primo *valore-i* uguale al valore di *esp*
 3. Se tale *valore-i* esiste, allora vengono eseguite le corrispondenti *istruzioni-i* , e tutte le istruzioni dei case successivi fino alla fine dello switch o fino a che si incontra un comando di `break`
 4. Altrimenti, vengono eseguite *istruzioni-default*

```
int giorno;
...
switch (giorno) {
case 1: printf("Lunedì '\n");
        break;
case 2: printf("Martedì '\n");
        break;
case 3: printf("Mercoledì '\n");
        break;
case 4: printf("Giovedì '\n");
        break;
case 5: printf("Venerdì '\n");
        break;
default : printf("Week end\n");
}
```


istruzione switch

- Se abbiamo piu valori a cui corrispondono le stesse istruzioni, possiamo raggrupparli come segue:

```
case valore-1:
```

```
...
```

```
case valore-n:
```

```
    istruzioni
```

```
    break;
```

```
int giorno;
...
switch (giorno) {
case 1: case 2:
case 3: case 4:
case 5: printf("Giorno lavorativo\n");
        break;
case 6:
case 7: printf("Week end\n");
        break;
default : printf("Giorno non valido\n");
}
```

istruzione **switch**: osservazioni

- L'espressione usata per la selezione (*exp*) può essere una qualsiasi espressione C che restituisce un valore intero.
- I valori specificati nei vari **case** devono invece essere costanti intere
 - valori noti a tempo di compilazione
 - In particolare, non possono essere espressioni in cui compaiono variabili, ad esempio **è sbagliato scrivere:**

```
int a; ...
```

```
case a<0: printf("negativo\n");
```

istruzione **switch**: osservazioni

- Il C non richiede che nei case di un'istruzione switch l'ultima istruzione sia **break**, ma è buona norma inserirla per avere una semantica più chiara
- Quindi, in generale la sintassi di un'istruzione switch è:

```
switch (esp) {  
case valore-1: istruzioni-1  
...  
case valore-n: istruzioni-n  
default : istruzioni-default  
}
```

Es. di switch senza break: corretto ...

```
int lati;
printf("Immetti il massimo numero di lati del poligono
      (al piu` 6): ");
scanf("%d", &lati);
printf("Poligoni con al piu` %d lati: ", lati);
switch (lati) {
case 6: printf("esagono, ");
case 5: printf("pentagono, ");
case 4: printf("rettangolo, ");
case 3: printf("triangolo\n");
break;
case 2: case 1: printf("nessuno\n");
break;
default : printf("\nErrore: valore immesso > 6.\n");
}
```

Es. di switch senza break: corretto ...

- Quando si omettono i break, diventa rilevante l'**ordine** in cui vengono scritti i vari case
- Questo puo essere facile causa di errori
- È quindi buona norma **mettere break** come ultima istruzione di ogni case

Istruzioni iterative

- Riprendiamo il programma di stampa del massimo fra tre numeri

if: max fra 3

```
/* una possibile soluzione */
#include <stdio.h>
int main(void) {
    double max, tmp;
    printf("Inserisci il primo valore:");
    scanf("%lf", &max);
    printf("Inserisci il secondo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Inserisci il terzo:");
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```


Istruzioni iterative

- Riprendiamo il programma di stampa del massimo fra tre numeri
- E se i numeri diventano 5, 10, 100 ?
 - Abbiamo 5x 10x 100x **if**, **printf()**, **scanf()**,
- *Le istruzioni iterative* servono a ripetere un comando o più comandi (in un blocco) più volte

Istruzioni iterative

- In C le istruzioni iterative sono 3:
 - `for`
 - `while`
 - `do..while`

istruzione **while**

- sintassi:

```
while (esp)  
    istruzione
```

- semantica:

1. Si valuta il valore di *esp*
2. se è vera si esegue *istruzione*, e poi si riesegue tutto il **while**
3. altrimenti si termina l'esecuzione del **while**

Nota che se *esp* è falsa all'inizio il **while** non fa niente

while: max fra 3

```
/* soluzione iterativa con while */
#include <stdio.h>
#include <float.h>

int main(void) {
    double max=DBL_MIN, tmp;
    int i = 0;
    while (i < 3){
        printf("Inserisci valore %d:", i+1);
        scanf("%lf", &tmp);
        if ( max < tmp ) max = tmp;
        i++;
    }
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

Minimo reale rappresentabile
dal tipo **double**

```
int main(void) {  
    double max=DBL_MIN, tmp;  
    int i = 0;  
    while (i < 3) {  
        printf("Inserisci valore %d:", i+1);  
        scanf("%lf", &tmp);  
        if ( max < tmp ) max = tmp;  
        i++;  
    }  
    printf("Il massimo e' :%f\n", max);  
    return 0;  
}
```

while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

Variabile che tiene conto

Del numero di valori inseriti

```
int main(void) {  
    double max=DBL_MIN, tmp;  
    int i = 0;  
    while (i < 3) {  
        printf("Inserisci valore %d:", i+1);  
        scanf("%lf", &tmp);  
        if ( max < tmp ) max = tmp;  
        i++;  
    }  
    printf("Il massimo e' :%f\n", max);  
    return 0;  
}
```

while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

Prima di eseguire il while
si valuta la condizione
(detta **guardia**)

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i = 0;
```

```
    while (i < 3){
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
        i++;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

```
int main(void) {  
    double max=DBL_MIN, tmp;
```

```
    int i = 0;
```

```
    while (i < 3) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
        i++;
```

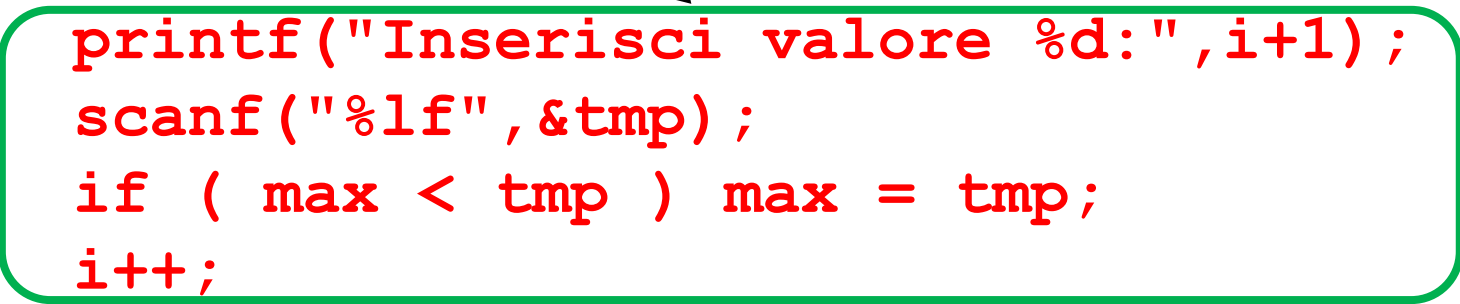
```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

Viene eseguita l'istruzione
In questo caso un blocco
(notare che le variabili
mantengono il valore fra una
esecuzione e l'altra del blocco)



while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

Si ripete l'esecuzione finchè
La variabile **i** non
raggiunge il valore 3

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i = 0;
```

```
    while (i < 3) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
        i++;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

while: max fra 3

```
/* soluzione iterativa con while */
```

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

```
#include <float.h>
```

Quando **i** vale 3

(la guardia non è più verificata)

Si passa alla istruzione

successiva al **while**

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i = 0;
```

```
    while (i < 3) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
        i++;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

Max fra 3 con **while** : output

Esecuzione:

Inserisci il valore 1:

Se digitiamo 15.1 e ↓ (invio) poi 7.2 e ↓ (invio)

Inserisci il valore 1: 15.1

Inserisci il valore 2: 7.2

Inserisci il valore 3:

Se infine inseriamo 0 e ↓ (invio)

Inserisci il valore 3: 0

Il massimo è: 15.1

while: *variabile di controllo*

```
/* soluzione iterativa con while */
#include <stdio.h>
#include <float.h>

int main(void) {
    double max=DBL_MIN, tmp;
    int i = 0;
    while (i < 3) {
        printf("Inserisci valore %d:", i+1);
        scanf("%lf", &tmp);
        if ( max < tmp ) max = tmp;
        i++;
    }
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

Costanti come macro

- È sempre meglio non inserire costanti esplicite nel codice
- Tipicamente in C si usa il meccanismo delle macro (senza parametri)
- Una macro permette di dare un nome simbolico ad una costante

Costanti come macro: esempio

```
/* soluzione iterativa con while */
#include <stdio.h>
#include <float.h>
#define N 3
int main(void) {
    double max=DBL_MIN, tmp;
    int i = 0;
    while (i < N) {
        printf("Inserisci valore %d:", i+1);
        scanf("%lf", &tmp);
        if ( max < tmp ) max = tmp;
        i++;
    }
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```

Costanti come macro: vantaggi

- **Leggibilità del codice**
 - si possono usare le macro nelle espressioni, legando fra loro tutti i valori del programma
 - es. N , $N+1$, $N*2$
- **Modificabilità:**
 - Si può usare un nuovo valore per la macro modificando una singola riga senza dover reinterpretare tutto il codice

Costanti come macro: esempio

```
/* soluzione iterativa con while */
#include <stdio.h>
#include <float.h>
#define N 100          /* passo a 100 valori */
int main(void) {
    double max=DBL_MIN, tmp;
    int i = 0;
    while (i < N) {
        printf("Inserisci valore %d:", i+1);
        scanf("%lf", &tmp);
        if ( max < tmp ) max = tmp;
        i++;
    }
    printf("Il massimo e' :%f\n", max);
    return 0;
}
```


while: primi esercizi ...

- Scrivere un programma C che stampa 100 asterischi ("*")
- Scrivere un programma C che somma 5 double inseriti da standard output e stampa il valore totale su standard output
- Modificare il programma precedente in modo da leggere N da standard input e poi sommare N double ...
- Li vediamo in laboratorio....

while: terminazione...

- Come si comporta il seguente ciclo se N vale 5 ?

```
int i = 0;
while (i < N) {
    printf("Inserisci valore%d:", i+1);
    scanf("%lf", &tmp);
    if ( max < tmp ) max = tmp;
    i--;
}
```

while: terminazione...

- Con i costrutti iterativi è possibile scrivere programmi che non terminano ...
 - Occorre fare attenzione
 - È possibile uccidere i programmi che *vanno in ciclo occupando il processore* (lo vediamo in laboratorio)
- In generale non è possibile capire se un programma termina o no in modo automatico!
 - È un esempio di problema indecidibile

Ciclo **for** ...

Analizziamo la struttura del ciclo **while** visto:

- Utilizza una variabile di controllo
- La guardia verifica se la variabile di controllo ha raggiunto un limite
- Ad ogni iterazione si esegue una azione
- Alla fine di ogni iterazione si modifica la variabile di controllo

Questa struttura si ritrova in molti programmi

- Facciamo un altro esempio

Ciclo for ...

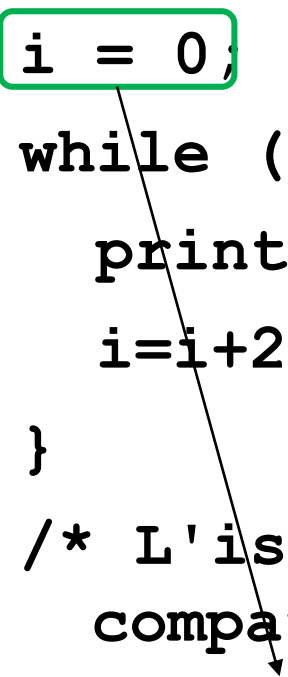
```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}
```

Ciclo for ...

```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}  
/* L'istruzione for permette di esprimere in modo  
compatto questi passi */  
for (i = 0; i <= N; i=i+2)  
    printf("%d", i);
```

Ciclo for ...

```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}  
/* L'istruzione for permette di esprimere in modo  
compatto questi passi */  
for (i = 0; i <= N; i=i+2)  
    printf("%d", i);
```

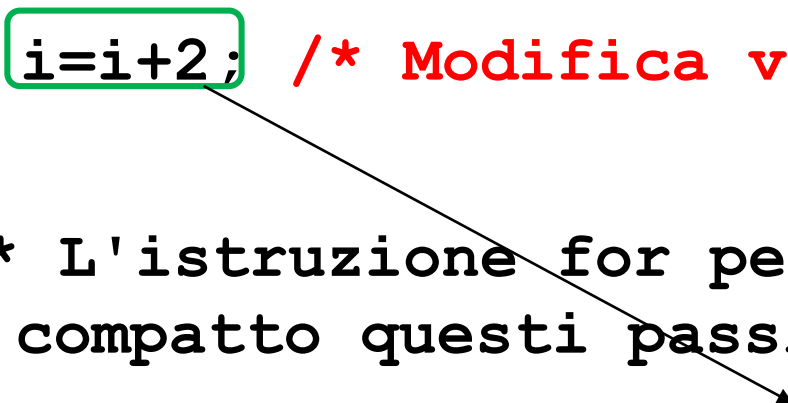


Ciclo for ...

```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}  
/* L'istruzione for permette di esprimere in modo  
   compatto questi passi */  
for (i = 0; i <= N; i=i+2)  
    printf("%d", i);
```

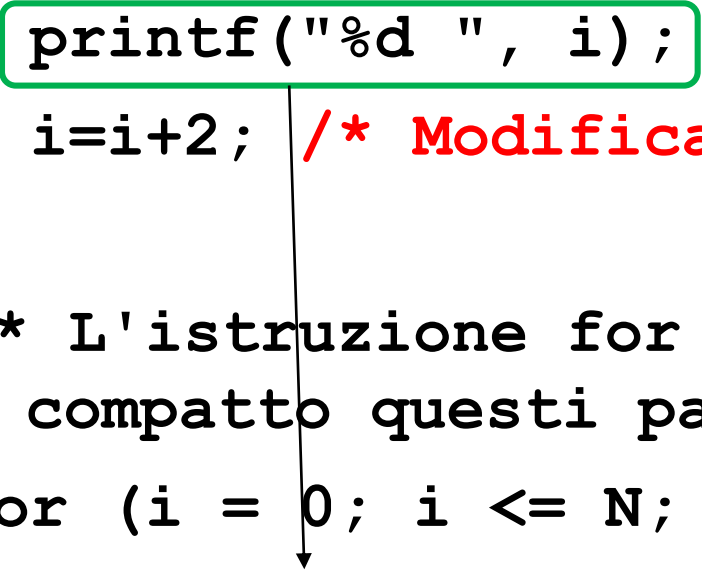

Ciclo for ...

```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}  
/* L'istruzione for permette di esprimere in modo  
compatto questi passi */  
for (i = 0; i <= N; i=i+2)  
    printf("%d", i);
```



Ciclo for ...

```
/* Stampare i numeri pari da 0 a N. */  
i = 0; /* Inizializza la var. Di controllo */  
while (i <= N) { /* guardia */  
    printf("%d ", i); /* Azione da ripetere */  
    i=i+2; /* Modifica var. di controllo */  
}  
/* L'istruzione for permette di esprimere in modo  
   compatto questi passi */  
for (i = 0; i <= N; i=i+2)  
    printf("%d", i);
```



Istruzione **for**

Sintassi:

```
for (istr-1; espr-2; istr-3)  
    istruzione
```

- **istr-1** inizializza la variabile di controllo
- **espr-2** è la guardia da verificare
- **istr-3** modifica la variabile di controllo
- **istruzione** è il corpo del ciclo

Semantica: l'istruzione è equivalente a

```
istr-1;  
while (espr-2) {  
    istruzione  
    istr-3;  
}
```

for: max fra 3

```
/* soluzione iterativa con for */
```

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

Minimo reale rappresentabile

```
#include <float.h>
```

dal tipo **double**

```
int main(void) {
```

```
double max=DBL_MIN, tmp;
```

```
int i;
```

```
for (i = 0; i < 3; i++) ) {
```

```
    printf("Inserisci valore %d:", i+1);
```

```
    scanf("%lf", &tmp);
```

```
    if ( max < tmp ) max = tmp;
```

```
}
```

```
printf("Il massimo e' :%f\n", max);
```

```
return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

Viene ripetuto tre volte,

Con valori di *i* uguale

a 0, 1 e 2

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) ) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

La variabile di

controllo va

dichiarata prima del **for**

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) ) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

Prima di iniziare ad eseguire
il primo blocco si esegue
la inizializzazione **i=0**

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) ) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

Poi si controlla la guardia se $i < 3$ se è vero si eseguono tutte le istruzioni del blocco ...

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```


for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

Poi si controlla la guardia se $i < 3$ se è vero si eseguono tutte le istruzioni del blocco ...

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) ) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

All fine del blocco si esegue la
terza espressione...

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

for: max fra 3

```
/* soluzione iterativa */
```

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

```
#include <float.h>
```

All fine del blocco si esegue la terza espressione `i++` e si ricomincia dal controllo della guardia ...

```
int main(void) {
```

```
    double max=DBL_MIN, tmp;
```

```
    int i;
```

```
    for (i = 0; i < 3; i++) ) {
```

```
        printf("Inserisci valore %d:", i+1);
```

```
        scanf("%lf", &tmp);
```

```
        if ( max < tmp ) max = tmp;
```

```
    }
```

```
    printf("Il massimo e' :%f\n", max);
```

```
    return 0;
```

```
}
```

Ancora su **for**

In realtà, la sintassi del **for** è

```
for (espr-1; espr-2; espr-3)
```

```
istruzione
```

- dove **espr-1**, **espr-2** e **espr-3** sono delle espressioni qualsiasi (in C anche l'assegnamento e un'espressione ...)

Vediamo altri esempi:

Ancora su **for**

```
for (i = 1; i <= 10; i=i+1)
```

```
    /* => i: 1, 2, 3, . . . , 10 */
```

```
for (i = 10; i >= 1; i=i-1)
```

```
    /* => i: 10, 9, 8, . . . , 2, 1 */
```

```
for (i = -4; i <= 4; i = i+2)
```

```
    /* => i: -4, -2, 0, 2, 4 */
```

```
for (i = 0; i >= -10; i = i-3)
```

```
    /* => i: 0, -3, -6, -9 */
```

```
for (i = 0, j = 0; i+j < 10; i++, j+=2)
```

```
    /* => (i,j): (0,0), (1,2), (2,4), (3,6) */
```

Ancora su **for**

È buona prassi cercare di essere ordinati

```
for (espr-1; espr-2; espr-3)  
  istruzione
```

- usare ciascuna **espr-i** in base al significato descritto prima
- non modificare la variabile di controllo nel corpo del ciclo (aggiornarla solo in **espr-3**)

Ancora su **for**

```
for (espr-1; espr-2; espr-3)
```

```
    istruzione
```

- Ciascuna delle tre **espr-i** puo anche mancare:
 - Ma i " ;" vanno messi lo stesso
- se manca **espr-2** viene assunto il valore vero
- Se manca una delle tre **espr-i** è meglio usare un'istruzione **while** a meno di non sapere bene cosa si sta facendo

Ancora su **for**

```
for (i = 1; ; i=i+1) { ciclo }
```

```
/* => i: ????? */
```

```
for (i = 10; i >= 1;) { ciclo }
```

```
/* => i: ????? */
```

```
for (;;) { ciclo }
```

```
/* => ????? */
```

```
for (;;) ;
```

```
/* => ????? */
```


Ancora su **for**

```
for (i = 1; ; i=i+1) { ciclo }
```

```
/* => i: 1, 2, 3, . . . , 10, .....
```

```
  cicla all'infinito (a meno non sia  
  presente un break all' interno del  
  ciclo)*/
```

```
for (i = 10; i >= 1;) { ciclo }
```

```
/* => i: 10, 10, 10, .....
```

```
  cicla all'infinito (a meno non sia  
  presente un assegnamento a i o un break all'  
  interno del  ciclo)*/
```

do while: guardia alla fine del ciclo

Iterazione indefinita:

- In alcuni casi il numero di iterazioni da effettuare non è noto prima di iniziare il ciclo, perchè dipende dal verificarsi di una condizione

Vediamo un esempio:

Leggere una sequenza di double che termina con 0 e calcolarne la somma.

do while: guardia alla fine del ciclo

```
#include <stdio.h>
int main (void) {
double dato, somma = 0;
scanf("%lf", &dato);
while (dato != 0) {
    somma = somma + dato;
    scanf("%lf", &dato);
}
printf("somma: %f\n", somma);
return 0;
}
```

do while : guardia alla fine del ciclo

Sintassi:

do

istruzione

while (*espressione*);

Semantica: è equivalente a

istruzione

while (*espressione*)

istruzione

=> una iterazione viene eseguita comunque.

do while: somma

```
#include <stdio.h>
int main (void) {
double dato, somma = 0;
do {
    scanf ("%lf", &dato);
    somma = somma + dato;
} while (dato != 0) ;

printf ("somma: %f\n", somma);
return 0;
}
```

Esempio: MCD

- Vogliamo leggere due interi positivi e calcolarne il Massimo Comun Divisore:

$$\text{MCD}(12, 8) = 4$$

$$\text{MCD}(12, 6) = 6$$

$$\text{MCD}(12, 7) = 1$$

sfruttando direttamente la definizione di MCD

Esempio: MCD

- **Algoritmo:**
 1. Leggiamo m ed n da input
 2. Generiamo tutti i numeri compresi tra 1 e $\min(m,n)$, in ordine decrescente
 3. Per ogni numero x verifichiamo che m ed n siano divisibili
 - Verifichiamo che il resto della divisione per x sia uguale a 0
 4. Il primo che divide entrambi è l'MCD e viene stampato a video

Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mcd == 0 ) && ( n % mcd == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```


MCD: output

Esecuzione:

Inserisci m ed n:

Se digitiamo 12 e ↓ (invio) poi 8 e ↓ (invio)

12

8

MCD di 12 e 8 = 4

Esempio: MCD

```
int main (void) {  
    int m, n, mcd;  
    printf("inserisci m ed n\n");  
    scanf ("%d", &m);  
    scanf ("%d", &n);  
  
    mcd = m > n ? n : m;  
    for ( ; mcd > 1; mcd-- )  
        if ( ( m % mcd == 0 ) && ( n % mcd == 0 ) )  
            break ;  
    printf("MCD di %d e %d = %d\n", n, m, mcd);  
    return 0;  
}
```

Acquisisce i valori
di n ed m

Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

Espressione condizionale (poteva essere inserita come prima espressione del **for**)

Esempio: MCD


```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mcd == 0 ) && ( n % mcd == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

Decremento **mcd** fino ad arrivare ad 1

Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

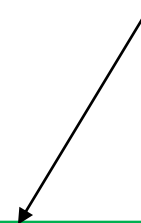
Calcola il resto della divisione di **m** per **mcd** e controlla che sia 0



Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mcd == 0 ) && ( n % mcd == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

Calcola il resto della divisione di **n** per **mcd** e controlla che sia 0

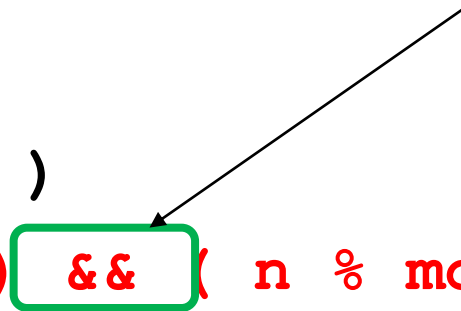


`n % mdc == 0`

Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

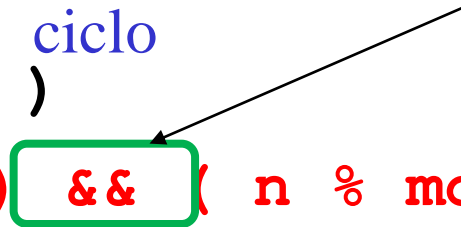
Operatore AND, è vero se entrambi i valori sono veri



Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

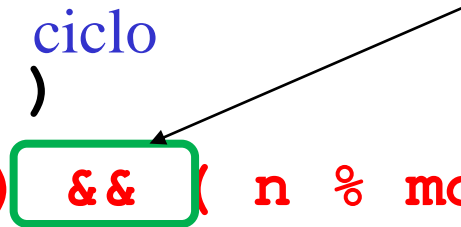
Quindi se entrambi sono divisibili l' **if** è verificato, si esegue il **break** e si esce dal ciclo



Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

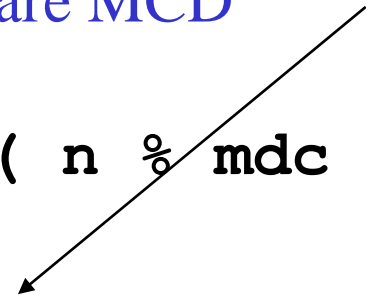
Quindi se entrambi sono divisibili l' **if** è verificato, si esegue il **break** e si esce dal ciclo



Esempio: MCD

```
int main (void) {
    int m, n, mcd;
    printf("inserisci m ed n\n");
    scanf ("%d", &m);
    scanf ("%d", &n);
    mcd = m > n ? n : m;
    for ( ; mcd > 1; mcd-- )
        if ( ( m % mdc == 0 ) && ( n % mdc == 0 ) )
            break ;
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;
}
```

Se siamo usciti dal for o abbiamo
arrivato un divisore o siamo
arrivati ad uno, quindi possiamo
stampare MCD



MCD : altra implementazione

```
int main (void) {
    int m, n, mcd, trovato = 0;
    printf("inserisci m ed n\n");
    scanf("%d", &m); scanf("%d", &n);
    mcd = m > n ? n : m;
    while ( mcd > 1 && ! trovato ) {
        if ( ( m % mcd == 0 ) && ( n % mcd == 0 ) )
            trovato = 1 ;
        else
            mcd = mcd -1;
    }
    printf("MCD di %d e %d = %d\n", n, m, mcd);
    return 0;}
```

MCD: considerazioni di stile

- Ogni algoritmo può avere diverse codifiche C corrette
- Ogni linguaggio ha degli stili di programmazione propri
 - La prima versione è quella che per me è più chiara
 - Molti programmi C usano `for + break` per uscire appena una condizione è verificata o qualcosa è stato trovato

MCD: complessità

- La complessità di un algoritmo è data da una misura del numero di istruzioni eseguite
- Nel nostro caso il ciclo esegue
 - 1 istruzione se m divide n o viceversa
 - $\min(n,m)$ istruzioni se MCD è 1
- Potrebbe essere un numero elevato se n, m sono interi grandi
 - Per abbassare la complessità possiamo cercare un altro algoritmo....

MCD: algoritmo di Euclide ...

- Sfrutta la seguente proprietà:

$$MCD(x, x) = x$$

$$MCD(x, y) = MCD(x-y, y) \text{ se } x > y$$

$$MCD(x, y) = MCD(x, y-x) \text{ se } y > x$$

- cioè i divisori comuni di m ed n , con $m > n$, sono anche divisori di $m-n$
- Questo può essere usato per ridurre drasticamente il numero di iterazioni quando il MCD è molto piccolo rispetto ad n ed m

MCD: algoritmo di Euclide ...

- Vediamo un esempio:
 1. $\text{MCD}(12, 8) = \text{MCD}(12-8, 8)$
 2. $\text{MCD}(4, 8) = \text{MCD}(4, 8-4)$
 3. $\text{MCD}(4, 4) = 4$
- In questo caso, ho fatto 3 soli passi invece di 5
- Inoltre ogni passo costa molto meno perchè richiede una sola sottrazione invece che due divisioni...

MCD: algoritmo di Euclide ...

- In generale guadagnamo molto di più:

m	n	m>n?m-n:n-m
210	63	147
147	63	84
84	63	21
21	63	42
21	42	21
21	21	

Esempio: MCD con Euclide

```
int main (void) {
    int m, n, tmp;
    printf("inserisci m ed n\n");
    scanf("%d", &m); scanf("%d", &n);
    printf("MCD di %d e %d = ", n, m);
    while ( m != n ) {
        if ( m > n ) m = m - n;
        else n = n - m;
    }
    printf("%d\n", tmp);
    return 0;
}
```

Algoritmo di Euclide con i resti...

- Cosa succede se $m \gg n$?

Esempio: $\text{MCD}(1000, 2)$

1000	2
998	2
996	2
994	2
....
2	2

- Come possiamo comprimere questa lunga sequenza di sottrazioni?

Algoritmo di Euclide con i resti...

Si può osservare che se

$$m = nk + r \text{ (con } 0 < r < m \text{)}$$

Cioè conosco quoziente n e resto r della divisione di m per n ...

Allora

$$MCD(m, n) = n \text{ se } r=0$$

$$MCD(m, n) = MCD(r, n) \text{ se } r>0$$

- In pratica posso trasformare la serie di sottrazioni in una unica divisione di cui controllo il resto
- Lo implementeremo in laboratorio (e controlleremo i tempi ...)