

Linguaggio C

strutture di controllo:
strutture iterative

Università degli Studi di Brescia

Docente: Massimiliano Giacomini

Strutture iterative

- Sono chiamate anche “cicli”
- Permettono di eseguire più volte un’istruzione, o un blocco di istruzioni, che costituisce il *corpo del ciclo*
- Prevedono di specificare una *condizione di permanenza* del ciclo: fintantoché la condizione è verificata viene eseguita una nuova *iterazione* del ciclo (ovvero, le istruzioni nel corpo del ciclo vengono eseguite)
- Tipologie di cicli:
 - cicli a condizione iniziale: **while** e **for**
 - cicli a condizione finale: **do-while**

Struttura iterativa while

Esempio

Input di un intero, continuando a ripetere l'input se l'utente inserisce un numero negativo

```
printf("Inserire un numero maggiore o uguale a 0\n");  
scanf("%d",&n);  
while(n<0)  
    scanf("%d",&n);  
...
```

Nota

Poiché la condizione è valutata all'inizio del ciclo, il corpo del ciclo potrebbe non essere mai eseguito (nell'esempio, se l'utente inserisce subito un valore positivo)

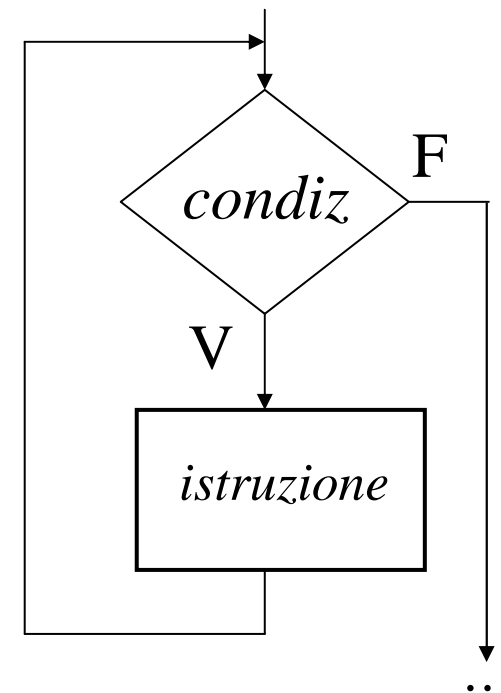
Struttura iterativa while

Sintassi

```
while (condiz)
    istruzione;
...
```

- *condiz* è una qualunque espressione aritmetica, ma tipicamente è costruita con operatori relazionali
p.es. $(i < 5)$
- *istruzione* può essere anche un blocco istruzioni

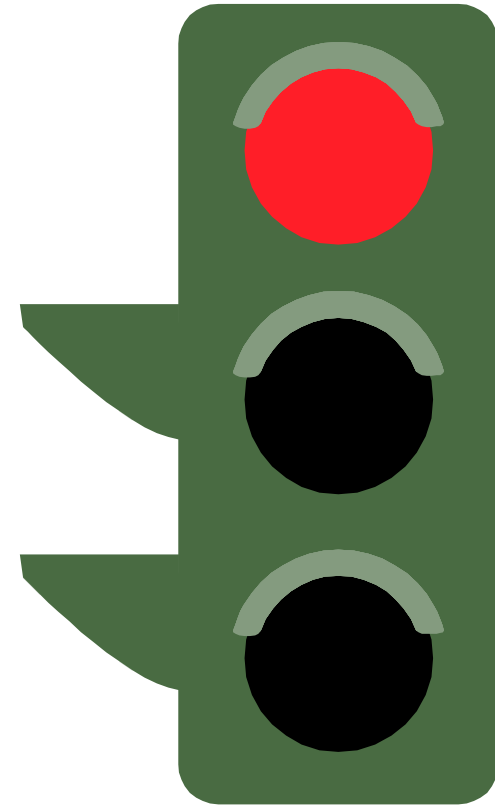
Semantica



- se *condiz* ha valore diverso da 0 (vera) viene eseguita *istruzione*, quindi si ritorna al controllo
- quando *condiz* diventa 0 (falsa) si passa alle istruzioni successive

CONSIGLIO

- Abituatevi subito a
“dare un significato preciso alle variabili”
- Anche se adesso sembra inutile,
è meglio imparare su esempi facili
(cfr. lezioni di chitarra)
- Se imparate adesso, gli esercizi d’esame
non saranno *tanto* più difficili di questi;
altrimenti, sembreranno più complicati di
quello che in realtà sono



Esempio 1: stampare i numeri da 1 a 5

```
int num=1;                \\ prossimo numero da stampare
while(num<=5){            \\ esci quando num > 5
    printf("Numero %d\n", num);
    num++;
}
```

Esempio 1: stampare i numeri da 1 a 5

```
int num=1;           \\ prossimo numero da stampare
while(num<=5){       \\ esci quando num > 5 (!!!)
    printf("Numero %d\n", num);
    num++;
}
```

Esempio 2: stampare i numeri da 1 a 5

```
int num=0;           \\ ultimo numero già stampato
while(num<5){        \\ esci quando num = 5 (!!!)
    num++;
    printf("Numero %d\n", num);
}
```

Errori tipici nell'uso di while

- Inserire una condizione di terminazione, non di permanenza

Es: per stampare i numeri da 1 a 5

```
int num=0;  
while(num==5){ //non è "finché" ma "mentre"...!  
    num++;  
    printf("Numero %d\n", num);  
}
```

- Usare l'operatore di assegnamento = al posto di quello di confronto ==

Es.

```
while(n=1){  
    scanf("%d", &n);  
    ...  
}
```

*In questo caso il ciclo è infinito
(la condizione $n=1$ è sempre vera!)*

Errori tipici nell'uso di while

- Mettere il punto e virgola dopo *while(cond)*: in questo caso il corpo del ciclo diventa l'istruzione vuota

Es.

```
while(n==1);  
{scanf("%d", &n);  
  ... }
```

In questo caso se $n==1$ il ciclo è infinito, altrimenti si esegue una sola volta il blocco di istruzioni

- Dimenticare di modificare le variabili che determinano la condizione

```
int num=0;  
while(num<5){  
    printf("Numero %d\n", num);  
}
```

Ciclo infinito: manca $n++$

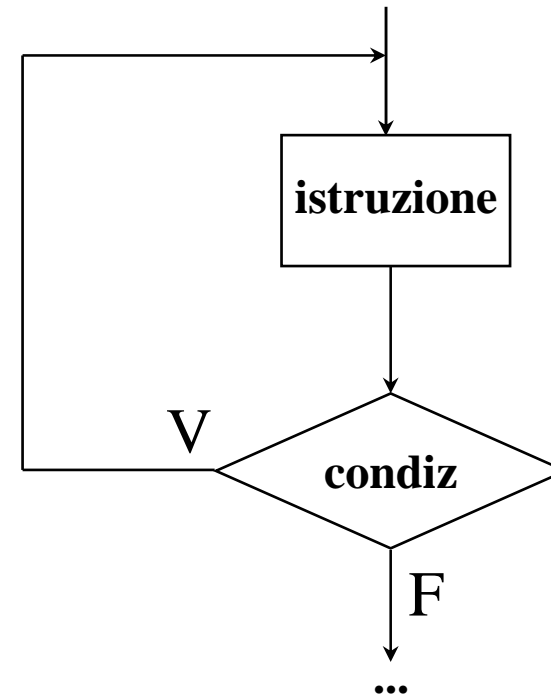
Struttura iterativa do-while: ciclo a condizione finale

Sintassi

```
do
    istruzione;
while (condiz);
...
```

- *condiz* è una qualunque espressione aritmetica, ma tipicamente è costruita con operatori relazionali
p.es. $(i < 5)$
- *istruzione* può essere anche un blocco istruzioni

Semantica



- Viene eseguita istruzione
- Se *condiz* ha valore diverso da 0 (vera) viene eseguita nuovamente istruzione e si prosegue; quando *condiz* diventa 0 (falsa) si passa alle istruzioni successive

Uso di do-while

- A differenza di *while* e *for*, la struttura *do-while* è un ciclo a condizione finale: garantisce che venga eseguita sempre almeno una iterazione del ciclo
- Dal punto di vista teorico tutto ciò che si può fare con *do-while* si può fare anche con *while* (o *for*); può essere però conveniente usare *do-while* nei casi in cui il corpo del ciclo debba in ogni caso essere eseguito almeno una volta
- Esempio già visto: acquisire un intero da tastiera, ripetendo l'operazione di input se l'utente inserisce un numero negativo

Versione con while

```
printf("Inserire positivo\n");  
scanf("%d",&n);  
while(n<0)  
    scanf("%d",&n);
```

Versione con do-while

```
printf("Inserire positivo\n");  
do  
    scanf("%d",&n);  
while(n<0);
```

Errori tipici nell'uso di do-while

- Inserire una condizione di terminazione al posto di quella di permanenza
- Usare l'operatore di assegnamento = al posto di quello di confronto ==
- Mettere il punto e virgola dopo *do*

Es.

```
do;  
    scanf("%d", &n);  
while(n<0);
```

Il compilatore segnala un errore (ci sono due istruzioni dopo *do*, non trova *while*)

- Dimenticarsi il punto e virgola dopo *while*

Es.

```
do  
    scanf("%d", &n);  
while(n<0)  
printf("Numero inserito=%d",n);
```

Il compilatore segnala un errore (la seconda istruzione dopo *scanf* è un costrutto *while*!)

Struttura iterativa for: un'alternativa al while

- Permette di scrivere un ciclo a condizione iniziale in modo spesso più compatto rispetto al while, incorporando:
 - l'assegnamento di una variabile indice con un valore iniziale
 - la condizione di permanenza
 - l'aggiornamento della variabile indice

Esempio: Vogliamo stampare i numeri da 1 a 10

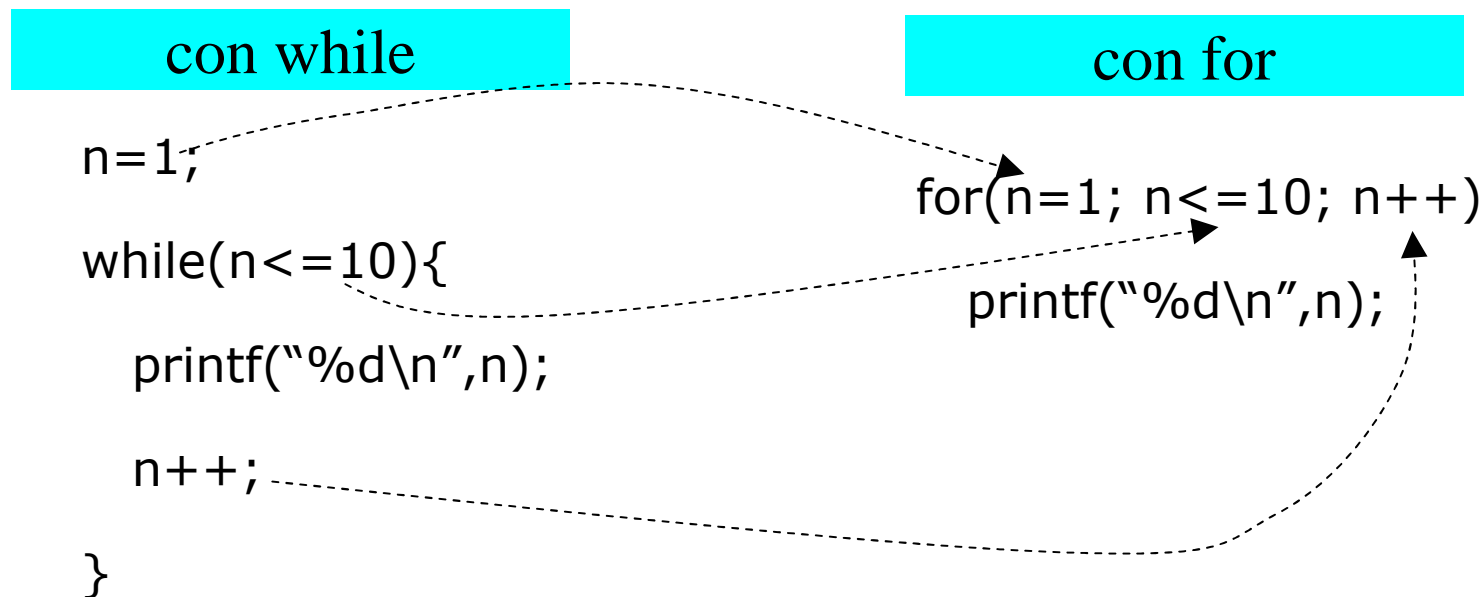
con while

```
n=1;
while(n<=10){
    printf("%d\n",n);
    n++;
}
```

Struttura iterativa for: un'alternativa al while

- Permette di scrivere un ciclo a condizione iniziale in modo spesso più compatto rispetto al while, incorporando:
 - l'assegnamento di una variabile indice con un valore iniziale
 - la condizione di permanenza
 - l'aggiornamento della variabile indice

Esempio: Vogliamo stampare i numeri da 1 a 10



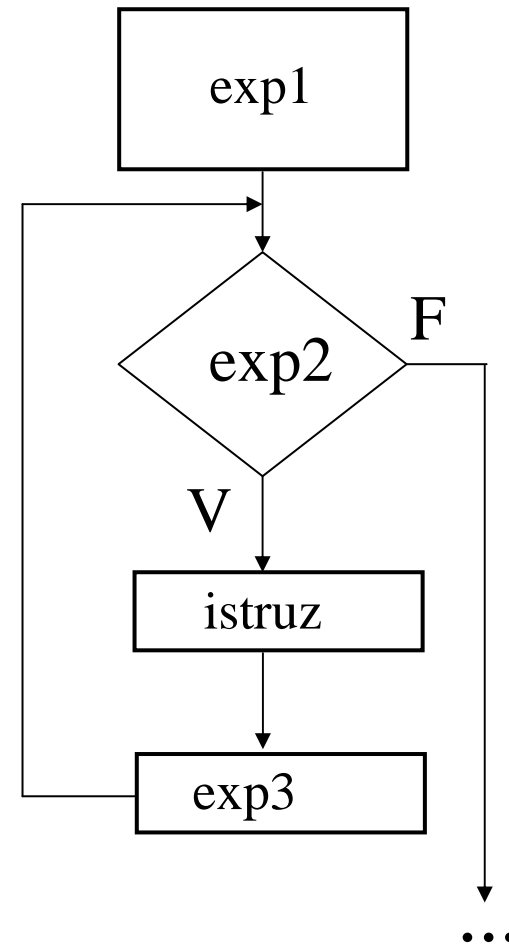
Struttura iterativa for

Sintassi

```
for(exp1; exp2; exp3)  
    istruz;  
...
```

- *exp1*, *exp2* ed *exp3* sono espressioni aritmetiche, ma:
 - tipicamente *exp1* è un assegnamento di una variabile, detta contatore, con un “valore iniziale” (ad esempio $i=1$)
 - *exp2* è la condizione di permanenza, tipicamente confronta il contatore con un valore finale (ad esempio $i \leq 5$)
 - *exp3* è tipicamente un assegnamento al contatore (ad esempio $i++$)
- *istruz* può essere anche un blocco istruzioni

Semantica



Altro esempio

Vogliamo stampare i numeri da 10 a 1

```
int n;
```

```
...
```

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

```
    printf("%d\n",n);
```

```
...
```


Equivalenza tra for e while

1) for per mezzo di while

*for(exp1; exp2; exp3)
istruzione;*



*exp1;
while(exp2){
 istruzione;
 exp3; }*

per esempio...

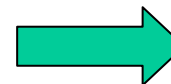
*for(n=10; n>=1; n--)
 printf("%d\n",n);*



*n=10;
while(n>=1){
 printf("%d\n",n);
 n--; }*

2) while per mezzo di for

*while(exp2)
 istruzione;*



*for(; exp2;)
 istruzione;*

Uso di for vs. while

- In generale, il *for* offre una sintassi che risulta più compatta rispetto al *while*
- Sicuramente, conviene usare il *for* ogni volta che viene usato una variabile “contatore” per scandire il numero di iterazioni del ciclo, tipicamente quando si conosce quante volte le istruzioni del corpo del ciclo devono essere ripetute
- Esempi:
 - acquisire 100 numeri da tastiera
 - acquisire n numeri da tastiera
 - stampare gli elementi di un vettore (vedi lezioni successive)
 - ecc. ecc.
- Non è comunque raro l'uso del *for* per qualsiasi tipologia di ciclo, al posto del *while* (dipende anche dal gusto personale)

Errori tipici nell'uso di for

- Inserire una condizione di terminazione al posto di quella di permanenza
- Usare l'operatore di assegnamento = al posto di quello di confronto ==
- Mettere il punto e virgola dopo *for(...;...;...)*: in questo caso il corpo del ciclo diventa l'istruzione vuota

Es.

```
for(i=1; i<=10;i++)  
    printf("numero %d\n", i);
```

In questo caso il ciclo viene ripetuto 10 volte, ma senza eseguire alcuna istruzione! Al termine del ciclo viene eseguita l'istruzione di stampa con *i=11* !

Esercizio 1

Scrivere un programma che acquisisce un intero positivo n e stampa tutti i numeri pari da 2 a n .

Esercizio 1

Scrivere un programma che acquisisce un intero positivo n e stampa tutti i numeri pari da 2 a n .

Soluzione

Uso un ciclo *for* con i che va da 2 a n (più o meno)
e che scandisce i numeri pari: deve essere incrementata di 2
[es: se $n = 11$ deve stampare 2, 4, 6, 8, 10]

Esercizio 1

Scrivere un programma che acquisisce un intero positivo n e stampa tutti i numeri pari da 2 a n .

Soluzione

Uso un ciclo *for* con i che va da 2 a n (più o meno) e che scandisce i numeri pari: deve essere incrementata di 2 [es: se $n = 11$ deve stampare 2, 4, 6, 8, 10]



```
for(i=2; i<=n; i=i+2)
```

```
    printf("%d\n",i);
```

← NB: i è il numero
“che devo stampare”

Il programma completo

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

main()
{
    int n, i;

    printf("Inserisci un numero positivo\n");
    scanf("%d", &n);

    for(i=2; i<=n; i=i+2)
        printf("%d\n", i);

    system("PAUSE");
}
```

Esercizio 2: while o do-while, questo è il dilemma

Scrivere un programma che continua ad acquisire un intero fino a quando l'utente non inserisce un numero positivo e multiplo di 100, dopodiché lo stampa a video.

Esempio:

L'utente inserisce

-100

34

49

200

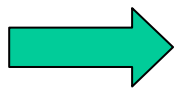


A questo punto il calcolatore stampa 200.

Un primo modo...

Ovviamente, mi serve un ciclo che continui ad acquisire numeri fino a quando il numero inserito non sia “corretto” (positivo e multiplo di 100).

Dato che devo acquisire almeno un numero, posso pensare di usare un ciclo do-while (a condizione finale)...



```
do{
```

```
    printf("Inserisci un numero positivo multiplo di 100\n");
```

```
    scanf("%d", &n); }
```

```
while (!(n>0 && n%100==0));
```

└ è lo stesso: while (n<=0 || n%100 != 0);

Il programma completo

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

main()
{
    int n;

    do{
        printf("Inserisci un numero positivo multiplo di 100\n");
        scanf("%d", &n); }
    while (n<0 || n%100!=0);

    printf("Numero inserito = %d\n", n);
    system("PAUSE");
}
```

Un'altra versione...

Supponiamo che:

- La prima volta che acquisisco un numero voglio stampare il messaggio "Inserisci un numero positivo multiplo di 100"
- Ogni volta che l'utente sbaglia, voglio inserire un messaggio diverso, per evidenziare che ha sbagliato

➡ *acquisisci numero n (con messaggio iniziale)*
while(n scorretto){
 stampa messaggio di errore;
 acquisisci n;
 }
stampa n; //corretto!

Il programma completo

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

main(){
    int n;
    printf("Inserisci un numero positivo multiplo di 100\n");
    scanf("%d", &n);

    while (n<0 || n%100!=0){
        printf("Forse non mi sono spiegato bene: ");
        printf("il numero deve essere positivo e multiplo di 100\n");
        printf("Riprova per favore\n");
        scanf("%d", &n);
    }
    printf("Numero inserito = %d\n", n);
    system("PAUSE");
}
```

Esercizi proposti

- Scrivere un programma per stampare l'alfabeto (da 'a' a 'z').

Suggerimento: ricordare che una variabile char rappresenta un numero e che 'a' indica il codice ASCII del carattere 'a' (che è un numero). Ricordare che la funzione printf consente di stampare una variabile char come carattere (%c) o numero (%d).

- Scrivere un programma che acquisisce un intero positivo n e stampa tutti i numeri multipli di 3 minori o uguali a n , in ordine inverso.

Per esempio con $n = 16$ stampa 15, 12, 9, 6, 3

Suggerimento: usare un ciclo for. Dato n intero, il numero multiplo di 3 da cui partire è dato da $(n/3)*3$: ad esempio, se $n = 20$ si ha $(n/3)*3 = 6*3 = 18$.

- Scrivere un programma che acquisisce un numero intero e continua l'acquisizione fino a quando l'utente inserisce un anno bisestile, quindi lo stampa.

- Scrivere un programma che continua ad acquisire un intero fino a quando l'utente non inserisce il numero 0, quindi stampa il minimo tra i valori inseriti (escludendo lo 0 che serve solo per segnalare la terminazione della sequenza di numeri).

Ad esempio, se l'utente inserisce 10 3 4 7 2 9 9 2 0 il programma stampa 2.

Suggerimento: rivedere l'esercizio in cui si calcolava il massimo tra quattro valori usando if “nel secondo modo” (lucido 30 dei lucidi sulle strutture condizionali)

- Scrivere un programma che acquisisce dall'utente due numeri, quindi chiede di inserire la somma. Fino a quando l'utente non inserisce la somma corretta, il programma stampa la frase “Errato: riprova” e ripete l'acquisizione; appena l'utente inserisce la somma corretta, il programma stampa la parola “Bravo” e termina.