

FONDAMENTI DI INFORMATICA

Prof. PIER LUCA MONTESSORO
Università degli Studi di Udine

Linguaggio C
Strutture di controllo



Strutture di controllo

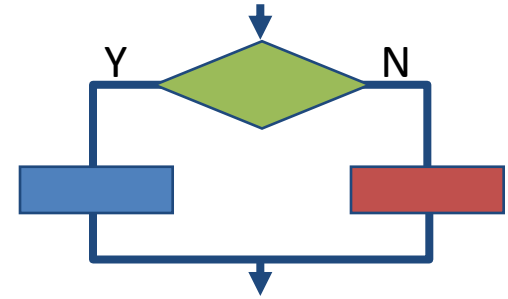
- In ogni linguaggio le **strutture di controllo** permettono di scrivere programmi nei quali il flusso di esecuzione non sia strettamente sequenziale
- I costrutti possono essere:
 - **condizionali:** `if-else, switch`
 - **iterativi:** `while, for, do ... while`



if-else

- L'istruzione condizionale **if-else** viene usata per esprimere una decisione
- La sintassi più generale è:

```
if (espressione)  
    istruzione_1;  
else  
    istruzione_2;
```



- **if** valuta l'espressione, e se risulta vera (\rightarrow valore non zero) viene eseguita l'**istruzione_1**; in caso contrario viene eseguita l'**istruzione_2**

if-else

- Esempio

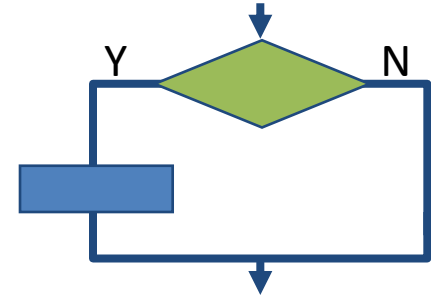
```
if (i != 0)
    printf ("%d", a / (double) i);
else
    printf ("Divisione impossibile");
```



if

- Nell'istruzione condizionale **if-else** la parte relativa a **else** è opzionale
- La sintassi più generale è:

```
if (espressione)  
    istruzione_1;
```



- **if** valuta l'espressione, e se risulta vera (→ valore non zero) viene eseguita l'**istruzione_1**; in caso contrario il programma prosegue

Istruzioni `if` annidate

- In assenza di un **`else`** all'interno di una sequenza di **`if`** annidati, ogni **`else`** viene associato all'**`if`** più vicino

```
if (max_iniz == VERO)
    if (dato > max)
        max = dato;
else
    max = dato;
```

*Il compilatore lo
interpreta così*

```
if (max_iniz == VERO)
{
    if (dato > max)
        max = dato;
else
    max = dato;
}
```

!?

- L'indentazione non è discriminante per il compilatore; con gli **`if`** annidati, ogni ambiguità va risolta utilizzando le parentesi

Istruzioni `if` annidate

- Quindi:

```
if (max_iniz == VERO)
{
    if (dato > max)
        max = dato;
}
else
    max = dato;
```



else if

- Un blocco di istruzioni costituito da una sola istruzione può essere scritto senza le parentesi graffe. Quindi, se da un'istruzione **if** dipende una sola istruzione, questa può essere scritta direttamente:

```
if (n % 2 == 0)
    printf ("n e` pari");
else
    printf ("n e` dispari");
```

- Questo vale anche se l'istruzione è un altro **if-else**:

```
if (n % 2 == 0)
    printf ("n e` divisibile per due");
else
    if (n % 3 == 0)
        printf ("n e` divisibile per tre");
    else
        if (n % 5 == 0)
            printf ("n e` divisibile per cinque");
    ...
```



else if

- Poiché l'indentazione non è significativa per il compilatore, la sequenza di **if-else** può essere riscritta così:

```
if (n % 2 == 0)
    printf ("n e` divisibile per due");
else if (n % 3 == 0)
    printf ("n e` divisibile per tre");
else if (n % 5 == 0)
    printf ("n e` divisibile per cinque");
...
```

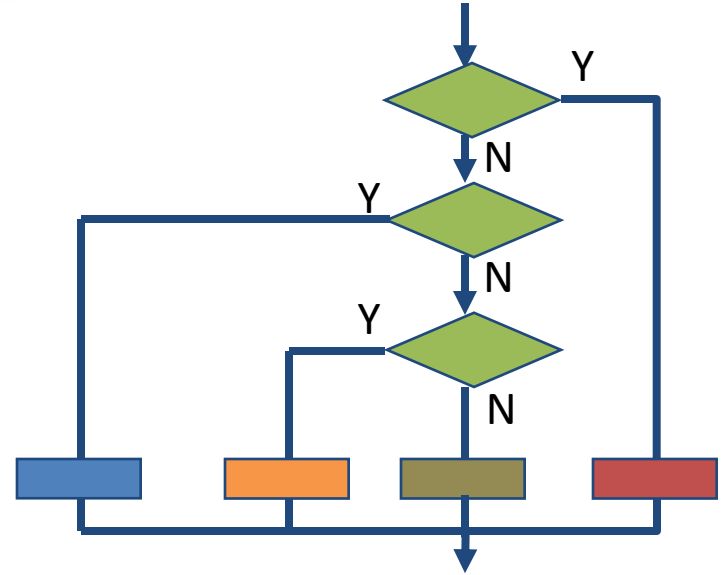
- Come si può osservare, **else if** non è una nuova istruzione, ma una forma di scrittura di if ann



else if

- La sintassi:

```
if (espressione_1)
    istruzione_1;
else if (espressione_2)
    istruzione_2;
else if (espressione_3)
    istruzione_3;
else
    istruzione_4;
```



è il modo più generale per realizzare una **scelta tra più opzioni**

- Le espressioni vengono analizzate nell'ordine in cui si presentano; se una di esse è vera, viene eseguita l'istruzione corrispondente e l'intera catena termina
- L'ultimo else rappresenta il caso di default

switch

- L'istruzione **switch** effettua una scelta multipla controllando se un'espressione assume uno dei valori in un insieme di costanti intere

```
switch (espressione)
{
    case costante_1:  istruzione_1;
                      break;
    case costante_2:  istruzione_2;
                      break;
    ...
    default:          istruzione_default;
                      break;
}
```



switch

- Ogni **case** è etichettato da un valore intero costante o da un'espressione intera costante
- I valori nei diversi **case** devono essere tra loro differenti
- Il caso **default**, opzionale, viene eseguito solo se nessuno dei casi precedenti si è verificato
- L'esecuzione delle istruzioni relative a un caso è seguita dall'esecuzione sequenziale di tutte le istruzioni successive (incluse quelle appartenenti ad altri **case**) fino ad una istruzione **break** o alla fine dello **switch** (parentesi graffa chiusa)

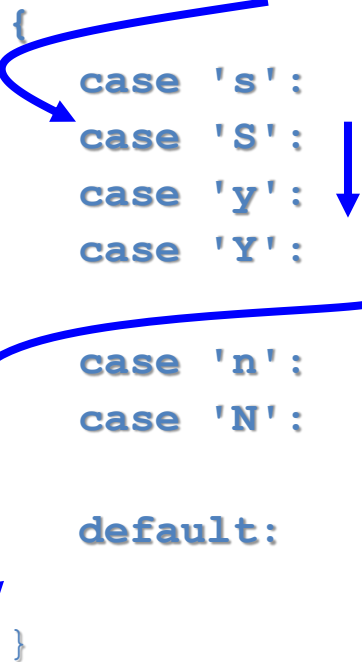


switch

```
char risposta;  
int favorevoli, contrari;  
...  
printf ("Sei favorevole?");  
scanf ("%c", &risposta);  
switch (risposta)
```

Es.: risposta = 'S'

```
{  
    case 's':  
    case 'S':  
    case 'y':  
    case 'Y':    favorevoli++;  
                break;  
    case 'n':  
    case 'N':    contrari++;  
                break;  
    default:    printf("Risposta errata!");  
                break;  
}
```



NOTA: si possono anche associare più etichette allo stesso blocco di istruzioni



while

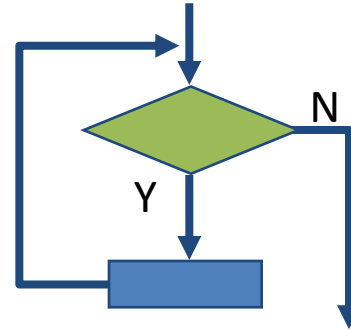
- Il costrutto **while** ha la sintassi:

```
while (espressione)  
    istruzione;
```

oppure

```
while (espressione)  
{  
    blocco di istruzioni  
}
```

- Se l'espressione risulta vera, allora viene eseguito il blocco di istruzioni, al termine del quale l'espressione viene rivalutata



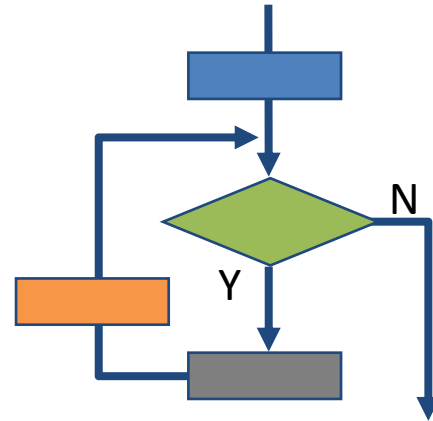
for

- Il costrutto **for** permette di scrivere in forma sintetica un **while** contenente:
 - una o più **istruzioni di inizializzazione**
 - la **condizione** da verificare prima di ogni iterazione
 - una o più **istruzioni di aggiornamento**, eseguite al termine di ogni ciclo ha la sintassi:

```
for (istr_1; espr_2; istr_3)  
    istruzione;
```

oppure

```
for (istr_1; espr_2; istr_3)  
{  
    blocco di istruzioni  
}
```



Operatore virgola

- L'operatore , (virgola) consente di raggruppare più istruzioni (per esempio assegnazioni e incrementi)
- Esempio:

```
for (i = 0, j = 5; funz_1(i) < funz_2(j) ; i++, j--)  
{  
    istruzioni  
}
```

inizializza **i** a 0 e **j** a 5, valuta l'espressione e, dopo ogni esecuzione delle istruzioni del ciclo, incrementa **i** e decrementa **j**



for

- Il frammento di codice relativo al ciclo **while**

```
contatore = 0;  
while (contatore < esponente)  
{  
    potenza = potenza * base;  
    contatore++;  
}
```

può essere riscritto come

```
for (contatore = 0; contatore < esponente; contatore++)  
    potenza = potenza * base;
```



do ... while

- La sintassi del costrutto **do ... while** è la seguente:

do

istruzione;

while (**espressione**);

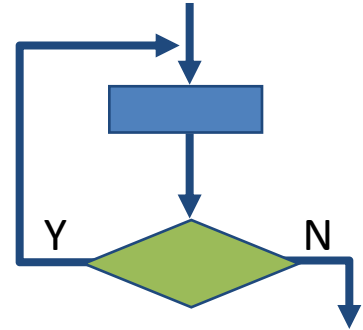
oppure:

do

{

blocco di istruzioni

} while (**espressione**);



- L'istruzione (o il blocco di istruzioni) viene sempre eseguita almeno una volta; al termine viene valutata l'espressione e, se questa è vera, l'istruzione viene ripetuta e poi l'espressione rivalutata

break e continue

- L'istruzione **break** provoca l'uscita incondizionata da un ciclo **for**, **while** o **do ... while**, senza eseguire le istruzioni successive né valutare la condizione di controllo

SI TRATTA DI ISTRUZIONI CHE VIOLANO IL PARADIGMA DELLA PROGRAMMAZIONE STRUTTURATA, QUINDI NON VANNO UTILIZZATE

(Naturalmente fa eccezione l'uso nello **switch**)



Attenzione al punto e virgola

```
contatore = 0;  
while (contatore < esponente) ;  
{  
    potenza = potenza * base;  
    contatore++;  
}
```

*Il ciclo esegue
soltanto l'istruzione
vuota, e non termina*

```
if (dato > max) ;  
    max = dato;
```

*Se la condizione è falsa non
viene eseguita l'istruzione
vuota, poi viene sempre
eseguita l'assegnazione*

