Istruzioni

- Le ***istruzioni*** esprimono ***azioni*** che, una volta eseguite, comportano una ***modifica permanente dello stato interno*** del programma o del mondo circostante
- • Le ***strutture di controllo*** permettono di aggregare istruzioni semplici in istruzioni più complesse

- Una ***istruzione semplice*** è qualsiasi ***espressione*** seguita da un punto e virgola

- **`Esempio`**
- **`x = 0; y = 1;/* due istruzioni */`**

Istruzioni di Controllo

- Le istruzioni di controllo sono alla base della **programmazione strutturata**

- Una istruzione di controllo può essere:
  - una istruzione ***composta*** (blocco)
  - una istruzione ***condizionale*** (selezione: ramifica il flusso di controllo in base al valore vero o falso di una espressione "*condizione di scelta*")
  - una istruzione di ***iterazione*** (ciclo: esegue ripetutamente un'istruzione finché rimane vera una espressione "*condizione di iterazione"*)

Strutture di controllo

- **Obiettivo:** rendere più facile la lettura dei programmi (e quindi la loro modifica e manutenzione)
- Abolizione di **salti incondizionati** (go to) nel flusso di controllo
- La parte di esecuzione di un programma viene vista come un comando ottenuto tramite **istruzioni elementari**, mediante alcune regole di composizione (**strutture di controllo**)

Blocco

[<dichiarazioni e definizioni>]

{ <istruzione> }

- Il ***campo di visibilità*** dei simboli del blocco è ristretto al blocco stesso
- dopo un blocco non occorre il punto e virgola (esso *termina* le istruzioni semplici, non *separa* istruzioni)

Esempio di Blocco

```
/* programma che letti due numeri da
  input ne stampi la somma*/
#include <stdio.h>
main()
{/* INIZIO BLOCCO */
   int X,Y;
   printf(“Inserisci due numeri ”);
   scanf("%d %d",&X,&Y);
   printf("%d",X+Y);
} /* FINE BLOCCO */
```

Visibilità

Esistono delle ***regole di visibilità*** per gli identificatori (nomi di variabili, di funzioni, costanti): definiscono in ***quali parti*** del programma tali identificatori possono essere usati

In un programma esistono diversi ***ambienti***:

- area globale
- **il main**
- ogni singola funzione
- **ogni blocco**

Regole di Visibilità

1. *Un identificatore **NON** è visibile **prima** della sua dichiarazione*

SCORRETTO:
y non dichiarata

```
main(){
   int x = y*2;
   int y = 5;
   … …
```

CORRETTO:

```
main(){
   int y = 5;
   int x = y*2;
   … …
```

Regole di Visibilità

2. Se in un ambiente sono visibili due dichiarazioni dello stesso identificatore, la dichiarazione valida è quella dell'ambiente più vicino al punto di utilizzo

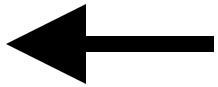
3 . *In ambienti diversi si può dichiarare lo stesso identificatore per denotare due oggetti diversi*

```
main(){
   float x = 3.5;
   {
      int y, x = 5;
      y = x; /* y vale 5 */
   }
   … …
```

Regole di Visibilità

4. In ciascun ambiente un identificatore può essere dichiarato una sola volta

```
main(){
   float x = 3.5;
   char x;
   … …
```



ERRORE

Regole di Visibilità

5. Un identificatore dichiarato in un ambiente è visibile in tutti gli ambienti in esso contenuti

SCORRETTO:
y non visibile!!

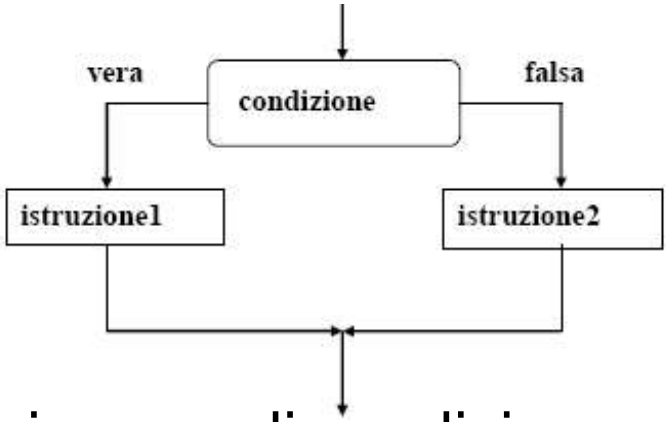
```
main(){
   int x;
   {
      int y = 5;
   }
   x=y;
   … …
```

CORRETTO:

```
main(){
   int x;
   {
      int y = 5;
      x=y;
   }
  … …
```

Istruzioni condizionali

```
if (<cond>)
  <istruzione1>
[ else <istruzione2> ]
```



La parte **else** è *opzionale*: se omessa, in caso di condizione falsa si passa subito all'istruzione che segue **if**

La condizione viene valutata al momento dell'esecuzione di if

<istruzione1> e <istruzione2> sono ciascuna una *singola istruzione.* Qualora occorra specificare più istruzioni, si deve quindi utilizzare **un *blocco***

Espressione condizionale ternaria (.. **?**…**:**…) fornisce *già* un mezzo per fare scelte, ma è *poco leggibile* in situazioni di medio/alta complessità. L'istruzione condizionale fornisce un altro modo per esprimere alternative

Esempio di istruzione if

```
/* determina il maggiore tra due numeri */

#include <stdio.h>
main()
{
 int a,b;
 scanf("%d %d",&a,&b);
 if (a > b)
   printf("%d", a);
 else printf("%d", b);
}
```

Esempio di utilizzo di un blocco

```
if (n > 0) { /* inizio blocco */
  a = b + 5;
  c = a;
} /* fine blocco */
else c = b;
```

Istruzioni IF annidate

Si verificano quando l'istruzione1 o l'istruzione2 sono rappresentate da un altro if.

La parte else (che è opzionale) si riferisce sempre alla istruzione if più interna.

```
if (n > 0)
   if (a>b) n = a;
   else n = b; /* riferito a if(a>b) */
```

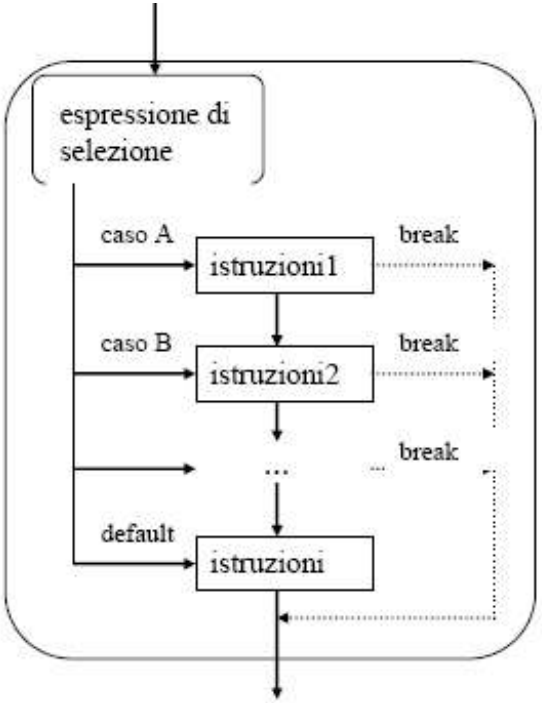
Per cambiare questa semantica, si deve inserire un blocco

```
if (n > 0)
{ if (a>b) n = a; }
else n = b; /* riferito a if(n>0) */
```

Istruzione di scelta multipla

Consente di scegliere fra ***molte istruzioni (alternative o meno)*** in base al valore di una ***espressione di selezione***

L'espressione di selezione deve ***denotare un valore numerabile*** (intero, carattere,…)



I vari rami *NON sono mutuamente esclusivi:* imboccato un ramo, si e*seguono anche tutti i rami successivi* a meno che non ci sia il comando **`break`** a *forzare esplicitamente* l'uscita

Istruzione switch

```
switch (selettore) {
   case <label1>: {case <labeli>:} <istruzioni>
    [break;]
   case <labelj>: {case <labeln>:} <istruzioni>
    [break;]
   …
   [ default : <istruzioni>]
}
```

Il valore di *selettore* viene confrontato con le etichette (costanti dello stesso tipo del selettore): si *eseguono tutti i rami corrispondenti* (se ne esistono), a meno che non si trovi break su un ramo.

Se nessuna etichetta corrisponde, si prosegue con il ramo default se esiste, altrimenti non si fa niente

Nella sequenza di istruzioni NON serve il blocco

Esempi di istruzione switch

```
switch (mese) {
  case 2: if (bisestile) giorni = 29;
             else giorni = 28; break  ;
   case 4: case 6: case 9: case 11: giorni = 30; break;
   default: giorni = 31;
}
```

```
printf("Vuoi eseguire il programma (s-n)?");
scanf("%c",&carattere);
switch (carattere) {
  case 's': case 'S': esegui=1; break ;
   case 'n': case 'N': esegui=0; break;
   default: printf("carattere scorretto\n");;
}
```

Istruzioni di Iterazione

Le istruzioni di iterazione ripetono l'esecuzione di una istruzione (o di un blocco di istruzioni).

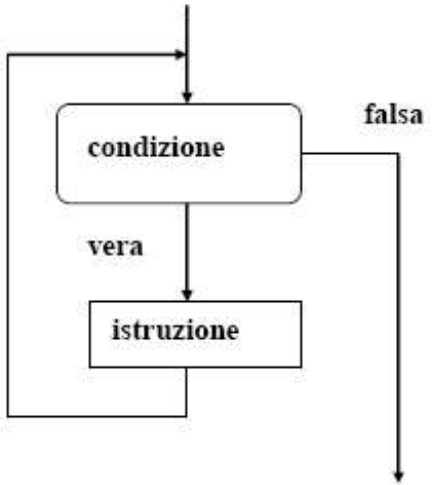
while **for** **do-while**

Le istruzioni di iterazione:

- hanno *un solo punto di ingresso* e *un solo punto di uscita* nel flusso del programma
- perciò possono essere interpretate *come una singola azione* in una computazione sequenziale

Istruzione while

```
while(<condizione>) <istruzione>
```



- L'istruzione viene ripetuta *per* ***tutto il tempo in cui la condizione rimane vera***

- Se la condizione è falsa, l'iterazione non viene eseguita ***neppure una volta***

- In generale, ***NON*** *è noto* ***quante volte*** l'istruzione sarà ripetuta

Esempio di istruzione while

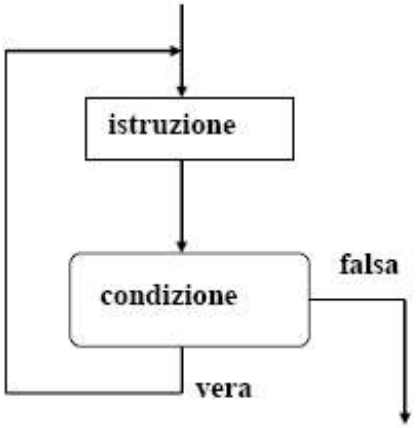
```
int a,somma=0,fine=0; char c;
printf("inserire gli addendi: ");
while (!fine){
   scanf("%d",&a); somma+=a;
   printf("sono finiti? s per finire:");
   scanf("%c",&c);
   switch (c) {
         case 's': case 'S': fine=1; break;
         default: fine=0;
   }
}
```

Prima o poi, *direttamente o indirettamente*, l'istruzione deve *modificare la condizione:a*ltrimenti, *CICLO INFINITO*

Quasi sempre *istruzione è un blocco*, in cui si m*odifica qualche variabile che compare nella condizione*

Istruzione do-while

```
do <istruzione> while(<condi
```



È una variante della precedente:
la condizione viene verificata
dopo aver eseguito l'istruzione
Anche se la condizione è falsa, l'iterazione
viene comunque eseguita
almeno una volta

Esempio di istruzione do-while

```
/* Calcolo del fattoriale di un numero N */

#include <stdio.h>
main()
{
int f, n, i;
f=1; /* inizializzazione del fattoriale*/
i=1;
printf("inserire il numero:");
scanf("%d",&n);
do {
     f = i*f;
     i = i+1;
}
while (i <= n);
printf("Il fattoriale è %d", f);
}
```

Esempio di istruzione do-while

```
int a,somma=0,fine; char c;
printf("inserire gli addendi: ");
do{
   scanf("%d",&a); somma+=a;
   printf("sono finiti? s per finire:");
   scanf("%c",&c);
   switch (c) {
         case 's': case 'S': fine=1; break;
         default: fine=0;
   }
} while (!fine);
```

Qual'è la differenza con l'esempio visto prima che usa l'istruzione while?

Istruzione for

```
for(<espr-iniz>;<cond>;<espr-modifica>)
        <istruzione>
```

Espressione di inizializzazione:
`<espr-iniz>` ***valutata una e una sola volta*** *prima di iniziare l'iterazione*

Condizione: `<cond>`
valutata *a* ***ogni interazione****,* per decidere se proseguire (come in un while). Se manca si assume *vera*

Espressione di modifica: `<espr-modifica>`
valutata *a* ***ogni interazione****,* ***dopo*** aver eseguito l'istruzione

for vs. while

L'istruzione for è una evoluzione del **`while`** che mira a eliminare alcune frequenti sorgenti di errore:

- mancanza delle *inizializzazioni delle variabili*
- mancanza della *fase di modifica del ciclo* (rischio di ciclo senza fine)

Si usa quando è noto quante volte il ciclo dovrà essere eseguito

```
for (e1; e2; e3)
  <istruzione>
```

```
e1;
while (e2) {
  <istruzione>
  e3;}
```

Esempio di istruzione for

```
int a,somma=0,n;
printf("quanti sono gli addendi? ");
scanf("%d",&n);
for(i=0;i<n;i++){
   printf("inserire addendo %d-esimo:",i);
   scanf("%d",&a); somma+=a;
}
printf("\nla somma vale: %d",somma);
printf("\nla media vale: %d",(float)somma/n);
```

Esempio di istruzione for

```
/* Calcolo del fattoriale di un numero N */
#include <stdio.h>
main()
{
     int n,f,i;
     printf(“Inserire il numero:”);
     scanf("%d",&n);
     f=1;     /*inizializzazione del fattoriale*/
     for (i=1; i <= n; i++)
     f=f*i;
     printf("Fattoriale: %d", f);
}
```