Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Introduzione al linguaggio C

Corso di Fondamenti di Informatica

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Prof. Paolo Romano

Si ringrazia il Prof. Alberto Finzi per aver reso
disponibile il materiale didattico sul quale si basano queste slides

Introduzione al C

Linguaggio di alto livello (in realtà di *medio livello*, infatti ha costrutti di alto livello ma consente un buon controllo della macchina).

Sviluppato nel 1972 come evoluzione del BCLP e B, come linguaggio per scrivere Compilatori e Sistemi Operativi (linguaggio per lo sviluppo del sistema UNIX). Dal ’70 ad oggi diventa uno dei linguaggi di alto livello più diffusi. Standardizzato nel 1984 dall’*American National Standard Institute (ANSI C)*, con aggiunte nel 1994.

Noi facciamo riferimento all’*ANSI C*.

Primo Programma in C

Programma Ciao Mondo:

```
#include<stdio.h>
int main() {
   printf("ciao mondo\n");
   return 0;
}
```

Il programma stampa sullo schermo una riga di testo:

```
ciao mondo
```

Di seguito il programma viene analizzato.

Analisi del Programma

```
#include<stdio.h>
int main() {
    printf("ciao mondo\n");
    return 0;
}
```

`int main()`: intestazione della funzione principale (main) di un programma **C**.

- I programmi **C** contengono una o più funzioni, main deve essere presente: l'esecuzione del programma inizia da quella del main.

- la parentesi graffa "`{`" apre il corpo della funzione, "`}`" lo chiude. La coppia di parentesi e la parte da esse racchiusa definiscono un blocco.

- Le funzioni C (come quelle matematiche) prendono in ingresso un insieme (anche vuoto) di argomenti e restituiscono un valore (anche nullo): in questo caso le parentesi **`()`** dopo **`main`** indicano il fatto che **`main`** è una funzione priva di argomenti (in input non prende nulla), che restituisce un valore di tipo intero (**`int`**).

Analisi del Programma

```
#include<stdio.h>
int main() {
  printf("ciao mondo\n");
   return 0;
}
```

printf(“ciao mondo\n”);

- istruzione semplice, in questo caso permette di visualizzare sullo schermo (canale di output standard) la frase scritta tra apici.
- ogni istruzione semplice deve terminare con “;”.
- oltre alle istruzione semplici esistono le istruzione composte (non tutte le composte terminano con “;”).
- la parte racchiusa tra apici è una stringa (di caratteri).
- “\n” non viene visualizzata sullo schermo e indica la nuova riga (a capo).
- “\” è un carattere di escape (fuga) e, insieme al carattere che segue ha un significato particolare (sequenza di escape).

Analisi del Programma

```
#include<stdio.h>
int main() {
        printf("ciao mondo\n");
        return 0;
}
```

`return 0;`

- se usato nella funzione main determina la fine dell'esecuzione del programma. In questo caso il programma termina restituendo il valore 0 ed indica solo che il programma è terminato correttamente.

`#include<stdio.h>`

- È una direttiva di compilazione (valutata in fase di compilazione):
- `#include` dice al compilatore di includere il contenuto del file indicato nel punto corrente.
- nel nel file `stdio.h` è contenuta la libreria standard di istruzioni per gestire le operazioni di *input/output* contenuto, ad esempio, l'istruzione `printf` è definita in `stdio.h`

Nota: l'indentazione evidenzia la struttura del programma.

Riassumendo

1. Un programma in C è un insieme di funzioni;
2. Ogni funzione prende in ingresso un insieme di argomenti e restituisce un valore;
3. Ci deve sempre essere una funzione principale `main();`
4. Le istruzioni di una funzione sono comprese tra una parentesi graffa iniziale ed una finale;
5. Ogni istruzione termina con un punto e virgola.

Variazioni sul Primo Programma

```
#include<stdio.h>
/* secondo programma */
int main() {
   printf("ciao ”);
   printf(“mondo\n");
   return 0;
}
```

Questo programma produce lo stesso effetto del precedente.

`/* secondo programma */` è un commento:

il testo contenuto tra “`/*`” e “`*/`” è ignorato dal compilatore e serve solo per aumentare la leggibilità del programma.

`printf("ciao”); printf(“ mondo\n");`
produce lo stesso effetto di **`printf("ciao mondo\n");`**

Esercizio: Cosa viene stampato con **`printf("ciao\n mondo\n");`** ?

Un altro semplice programma

Area del triangolo:

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;
   area = base * altezza / 2;

   printf(“Area: %d”, area);
   return 0;
}
```

L’esecuzione determina la seguente stampa sullo schermo:

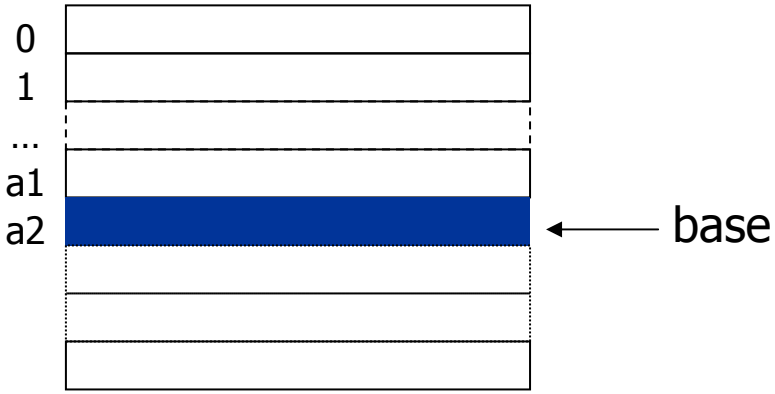
```
Area: 10
```

Variabili

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;
   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```



`int base; int altezza; int area;`

È una dichiarazione. **`base`**, **`altezza`**, **`area`** sono nomi di variabili. Le variabili rappresentano simbolicamente i dati all'interno dei programmi.

Una variabile identifica una locazione (posizione) della memoria in cui può essere memorizzato un dato a cui il programma può accedere.

Proprietà delle Variabili

Una variabile è caratterizzata dalle seguenti proprietà:

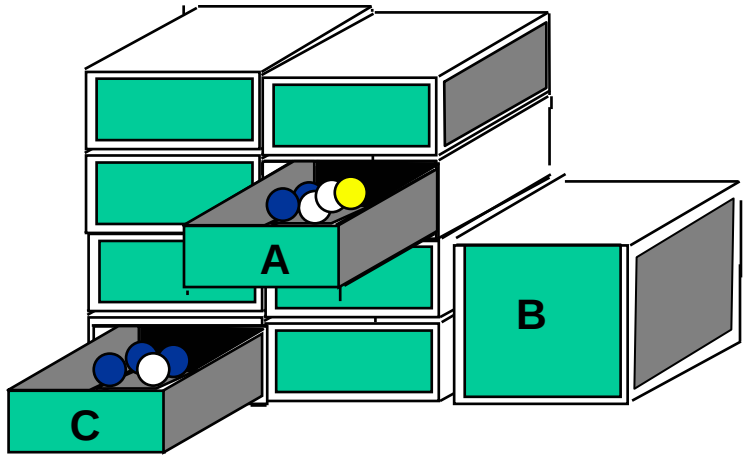
- **Nome:** identifica la variabile. Esempio: area.
  E' un *identificatore C*: sequenza di lettere, cifre, "_" che non inizia con una cifra (es. a123b e _as_231 lo sono, 1ab no).
  - solo i primi 31 caratteri sono considerati.
  - lettere maiuscole e minuscole sono distinte.
- **Tipo:** specifica il tipo del dato. Esempio: `int area` specifica il fatto che `area` rappresenta un valore intero.
- **Indirizzo:** della cella di memoria che contiene il dato. Se il dato occupa più celle, questo è memorizzato in celle consecutive e l'indirizzo è quello della prima cella.
- **Valore:** dato rappresentato dalla variabile in certo momento dell'esecuzione. Può cambiare (variabile) durante l'esecuzione.

Nome, Tipo, Indirizzo non variano a tempo di esecuzione

Proprietà delle Variabili

Le variabili come contenitori etichettati e posizionati su scaffali.

Nome		Etichetta
Tipo	⇨	Forma del contenitore (es. capienza)
Indirizzo		Posizione sullo scaffale
Valore		Contenuto



Note: non tutte le variabili hanno un identificatore, esistono identificatori che non rappresentano variabili (funzioni, tipi etc.).

Dichiarazione delle Variabili

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;
   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

`int base;` È una dichiarazione di variabile

- **Tipo:** `int` ⇒ intero
- **Nome:** `base`
- **Indirizzo:** allocata memoria
- **Valore:** quello iniziale è casuale

Dichiarazione delle Variabili

Variabili intere.

-Per dichiarare variabili intere si può usare il tipo `int`
-valori di tipo int sono rapp. in C con almeno 16 bit
-il numero effettivo di bit dipende dal compilatore (es. gcc 32 bit)
-in C esistono altri tipi di variabili intere (`short, long`).

Variabili reali.

-Per dichiarare variabili reali si può usare il tipo `float`
-specificatore di formato %g

Vedremo meglio più avanti

Assegnamento

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;
   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

`base = 5;` è un'istruzione di assegnamento

- "=" è l'operatore di assegnamento
- il valore a destra di "=" viene assegnato alla variabile di sinistra (nelle celle di memoria allocate per la variabile)

`area = base * altezza / 2;`

assegnamento con espressione alla destra

- Viene calcolato il valore indicato dall'espressione;
- viene assegnato tale valore alla variabile

Assegnamento

Nota bene:

Nell'istruzione **x = e**

1. viene prima valutata l'espressione e a destra di =;
2. Quindi si assegna il valore alla variabile **x** a sinistra di =.

Esempio:

```
int a; int b;
 a = 1;
 b = 2;
 a = b;
 b = a + b;
```

Valore var:

a	b
?	?
1	?
1	2
2	2
2	4

Esercizio: scambio di valore variabili

Stampa

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;
   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

`printf("Area: %d", area);`

- Il primo argomento è *stringa di formato*, la stringa di formato puo' contenere *specificatori di formato.*
- *Specificatore* `%d` indica la stampa di un intero in modalità decimale ("d" come decimale)
- Ogni specificatore deve corrispondere ad un valore che compare come argomento di `printf`:

$$\texttt{printf("\%d,\ldots,\%d",}a_1,\ldots,a_n\texttt{);}$$

Ingresso dati (da tastiera)

Area Triangolo con dati letti da tastiera (interazione):

```
#include<stdio.h>
int main() {
   int base, altezza, area;

   printf(“digita base \n:”);
   scanf(“%d”,&base);

   printf(“digita altezza \n:”);
   scanf(“%d”,&altezza);

   area = base * altezza / 2;

   printf(“Area: %d”, area);
   return 0;
}
```

Confronto programmi triangolo

```
#include<stdio.h>
int main() {
   int base, altezza, area;

   printf("digita base \n:");
   scanf("%d",&base);

   printf("digita altezza \n:");
   scanf("%d",&altezza);

   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;

   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

`scanf("%d",&base);`

- legge da tastiera un valore intero (%d) e lo assegna alla variabile `base`
- "%d" è una stringa di controllo del formato: intero in forma decimale;
- "&" è l'operatore di indirizzo "`&base`" indica l'indirizzo di mem. associato alla variabile `base`.

Confronto programmi triangolo

```
#include<stdio.h>
int main() {
   int base, altezza, area;

   printf("digita base \n:");
   scanf("%d",&base);

   printf("digita altezza \n:");
   scanf("%d",&altezza);

   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

```
#include<stdio.h>
int main() {
   int base; int altezza; int area;

   base = 5;
   altezza = 4;

   area = base * altezza / 2;

   printf("Area: %d", area);
   return 0;
}
```

Nota inoltre: **`int base, altezza, area;`**

cioè, si possono dichiarare piu' variabili dello stesso tipo con la seguente *sintassi*:

tipo variabile-1, variabile-2, ..., variabile-n;

Ingresso dati (da tastiera): esecuzione

```
#include<stdio.h>
int main() {
   int base, altezza, area;

   printf(“digita base: \n”);
   scanf(“%d”,&base);

   printf(“digita altezza: \n:”);
   scanf(“%d”,&altezza);

   area = base * altezza / 2;

   printf(“Area: %d”, area);
   return 0;
}
```

Cosa appare sullo schermo

```
digita base:
# 5 ^
digita altezza
# 6 ^
Area: 15
#
```

- Quando viene eseguita una `scanf` il prog. rimane in attesa di un input da tastiera.
- L’utente digita una sequenza di caratteri, quindi digita invio (^)
- La sequenza di caratteri viene convertita in intero (%d) quindi viene assegnata alla vaiabile (&base)

Operatori Aritmetici del C

In ordine di priorità:

1. - unario (priorità più alta)
2. * (moltiplicazione) / (divisione) % (modulo)
3. + (somma) – (sottrazione) – priorità bassa.

La / tra due interi indica la divisione intera:
Es. 24/4 = 6 25/6 = 4 -24/6 = -4 -25/6= -4

% può essere usato solo tra interi

-Le espressioni vengono valutate da sinistra a destra tenendo conto della priorità degli operatori (come in algebra) (es. `2 + 3 * 4` vale 14)

-Si possono utilizzare le parentesi per imporre un certo ordine di valutazione (es. `(2+3) *4`)

Riassunto

Abbiamo visto:

1. Struttura programma C
   (primo programma C)

2. Variabili:
   - Proprietà
   - Dichiarazione
   - Assegnamento

3. Istruzioni di output (schermo), input (tastiera):
   - `printf(“aaa %d”, &b);`
   - `scanf(“%d”, &b);`

```
#include<stdio.h>
int main() {
   int base, altezza, area;

   printf(“digita base \n:”);
   scanf(“%d”,&base);

   printf(“digita altezza \n:”);
   scanf(“%d”,&altezza);

   area = base * altezza / 2;

   printf(“Area: %d”, area);
   return 0;
}
```

Strutture di Controllo

Costrutti di Controllo

I programmi visti fino ad ora hanno una struttura sequenziale

Diagramma di flusso

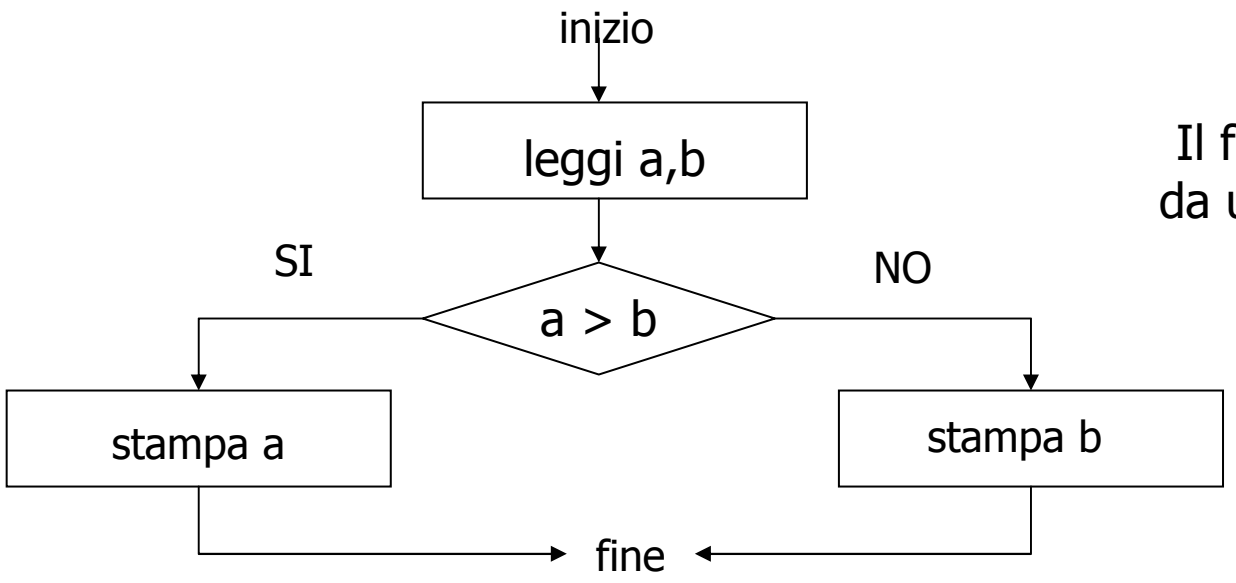
Gli algoritmi possono essere più complicati: occorre controllare il flusso delle istruzioni.

Costrutti di Controllo: condizionale

Consideriamo l'algoritmo:

1. Leggi due interi a,b
2. Se a > b allora stampa a
   altrimenti stampa b

Vediamo il flusso



Il flusso dipende da una condizione

Per descrivere questo algoritmo occorre una istruzione condizionale.

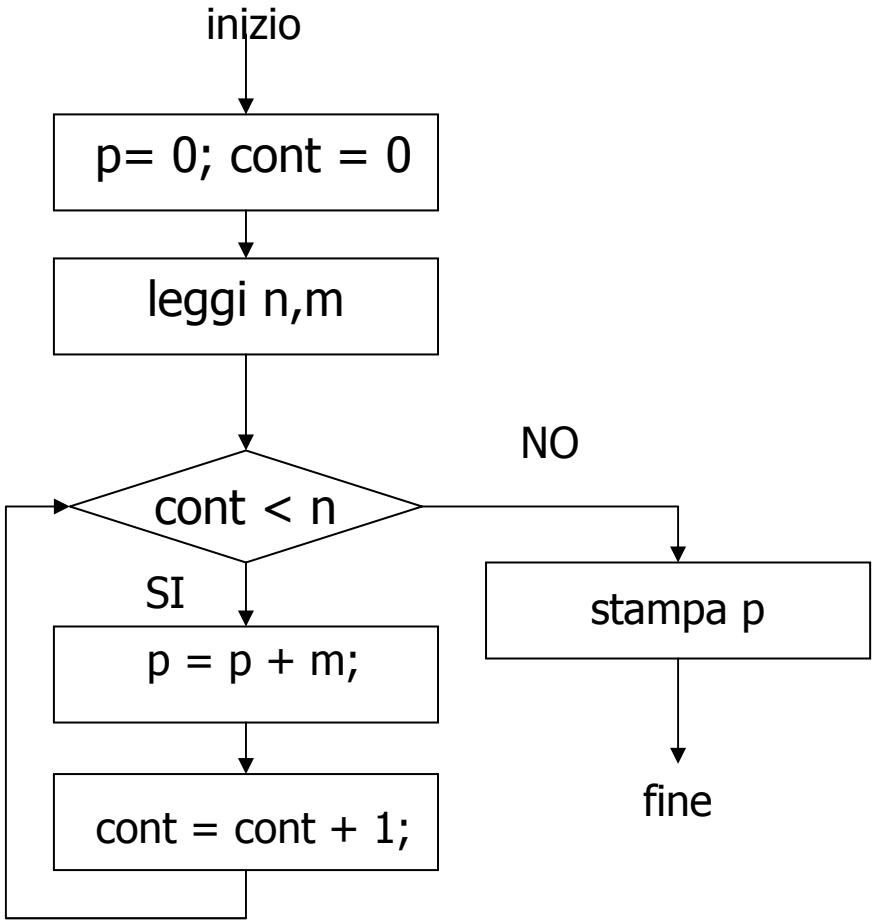
Costrutti di Controllo: ciclo

Consideriamo l'algoritmo:

1. Leggi due interi n, m;
2. Assegna p = 0;
3. Ripeti n volte l'istruzione (a):

   (a) p = p +m;
4. Stampa il valore di p

Vediamo il flusso



Per descrivere questo algoritmo occorre un'istruzione per l'esecuzione ciclica.

Istruzioni condizionali in C

Fino ad ora abbiamo visto come dichiarare variabili e modificare il loro valore:

```
int a, b;

a = 10;
b = 20;
```

Vediamo ora come il programma può valutarne il valore e decidere in base ad esso cosa fare:

istruzioni condizionali

Istruzioni condizionali in C

Istruzione `if-else`

Sintassi:

```
if (espressione)
   istruzione1
else
   istruzione2
```

Semantica (significato):

se l'`espressione` è vera allora
   esegui `istruzione1`
altrimenti
   esegui `istruzione2`

`espressione` è un'espressione condizionale il cui valore può essere vero o falso.

Istruzioni condizionali in C

Esempio istruzione `if-else`

```
#include<stdio.h>
int main() {
   int numero;

   printf("digita numero?");
   scanf("%d",&numero);

   if(numero > 0)
     printf("%d è positivo \n",numero);
   else
     printf("%d non è positivo \n",numero);

   return 0;
}
```

ramo then

ramo else

Espressione condizionale: `numero > 0`

Istruzione1: `printf("%d è positivo \n",numero);`

Istruzione2: `printf("%d non è positivo \n",numero);`

Espressioni condizionali

Un'espressione condizionale è una espressione, che opportunamente valutata, restituisce un *valore di verità*: vero o falso.

espressione matematica ⟹ numero (es. 3+2*4 ha valore 11)
espressione condizionale ⟹ vero/falso (es. 10 > 1 è vero: ha valore vero)

Per definire condizioni occorrono operatori opportuni
Es. L'espressione a + 1 > b * 3 contiene l'operatore relazionale >.

Operatori standard	Operatori C	Esempio condizione C
=, ≠	==, !=	a == b, 10 != c
>, <, ≥, ≤	>,<,>=,<=	a >= b, 10 < 5

Attenzione!!
Non confondere assegnazione = (es. a=10;)
con uguaglianza == (es. 10 == 10)

Espressioni condizionali

In C il valore vero/falso (booleano) si rappresenta come un intero (`int`):

vero ⟶ **`1`** (in realtà ogni valore diverso da `0`)
falso ⟶ **`0`**

Esempio:

`10 == 1` ha valore `0`
(con `printf("valore: %d", 10 == 1);` si ottiene `valore: 0`)

`10 > 5 + 2` ha valore `1`
`2 != 1` ha valore `1`

In pratica op. logici e matematici sono tutti operatori su numeri

Sintassi: *espr_matematica* *operatore_condizionale* *espr_matematica*

In realtà anche: `2+3*4-(2+1) < 12 == 1`

Precedenze: op. matematico, op. relazionale, op. uguaglianza

Istruzione if

Istruzione `if`: è una istruzione `if-else` in cui manca la parte `else`.

Sintassi:

if (espressione)
istruzione

Semantica:

1. Viene valutata l'espressione
2. Se l'espressione è vera si esegue l'istruzione1
3. Altrimenti si esegue l'istruzione successiva (senza eseguire istruzione1)

Esempio:

```
int temperatura;

scanf("%d",&temperatura);

if(temperatura >= 25)
  printf("fa caldo \n");
printf("ciao \n");
```

Blocco istruzioni

Fino ad ora una sola una sola istruzione nel ramo then (o nel ramo else) dell'istruzione if-else (o if). Per eseguire più istruzioni occorre un blocco istruzioni.

Sintassi

```
{ dichiarazione-variabili
  istruzione-1
     . . .
  istruzione-n
}
```

Le variabili nel blocco vengono dette locali al blocco.

Esempio blocco istruzioni

Esempio: dati mese ed anno, calcolare mese e anno del mese successivo

```
int mese, anno, annosucc, mesesucc;

scanf("%d%d",&mese,&anno);

if (mese == 12) {
    mesesucc = 1;
    annosucc = anno + 1;
}
else {
   mesesucc = mese + 1;
   annosucc = anno;

}
```

If annidati

L'istruzione nel ramo-then (ramo-else) di un if può essere un'altra istruzione if, es.

```
if (espressione1)
   if (espressione1)
      istruzione;
```

Possibili ambiguità:

```
if (espressione1)
if (espressione2)
istruzione1;
else
istruzione2;
```

a quale `if` si riferisce `else`??

Occorre una regola:

in `C else` si riferisce all'`if` più vicino

Quindi:

```
if (espressione1)
   if (espressione2)
      istruzione1;
   else
       istruzione2;
```

Esempio if annidati

```
int lato1, lato2, lato3;

scanf("%d%d%d", &lato1, &lato2, &lato3);

if (lato1 == lato2)
   if (lato2 == lato3)
      printf("Equilatero\n");
   else
      printf("Isoscele\n");
else
   if (lato2 == lato3)
      printf("Isoscele\n");
   else
      if (lato1 == lato3)
         printf("Isoscele\n");
      else
         printf("Scaleno\n");
```

If annidati

```
if (espressione1)
if (espressione2)
istruzione1;
else
istruzione2;
```

Per ottenere l'altra interpretazione?

Occorre introdurre un blocco:

```
if(espressione1) {
   if(espressione2)
      istruzione1;
}
else
   istruzione2;
```

Caso più complicato:

```
if (espressione1)
if (espressione2)
if (espressione3)
istruzione1;
else
istruzione2;
else
istruzione3;
```



```
if (espressione1)
   if (espressione2)
      if (espressione3)
         istruzione1;
      else
         istruzione2;
   else
      istruzione3;
```

Espressioni Booleane

Fino ad ora:

espr_matematica *operatore_condizionale* *espr_matematica*

Per combinare espressioni logiche occorrono operatori booleani

espr_booleana *operatore_booleano* *espr_booleano*

Operatori in ordine di priorità:
- ! (not logico): es. `!(10 == 2)`
- && (and logico): es. `(10 != 2)&&(1 > 0)`
- || (or logico): es. `(10 == 2)||(3 != 0)`

Semantica

a	b	!a	a&&b	a||b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Espressioni Booleane

Le espressioni booleane sono valutate da sinistra a destra:
- per && appena un operando è falso restituito 0 senza valutare il secondo argomento
- per || appena un operando è vero restituito 1 senza valutare il secondo argomento

Priorità tra operatori:
- not logico
- operatori aritmetici
- relazionali
- booleani &&, ||

Esempio:

```
     a + 2 == 3*b || !trovato && c < a / 3
((a + 2) == (3*b)) || (!trovato && (c < (a / 3)))
```

Esempio Espressioni Booleane

```
int altezza,anni;

printf(“digita altezza e anni quindi invia?”);
scanf(“%d %d”,&altezza,&anni);

printf(“altezza . . .”);
if(altezza >= 250 || altezza < 80 && anni >= 6 )
   printf(“non ”);
printf(“nella norma \n”);
```

Esempio Espressioni Booleane

```
int lato1, lato2, lato3;

scanf("%d%d%d", &lato1, &lato2, &lato3);

if (lato1 == lato2 && lato1 == lato3)
   printf("Equilatero\n");
else
   if (lato1 == lato2 || lato1 == lato3 || lato2 == lato3)
      printf(“Isoscele\n”);
   else
      printf(“Scaleno\n”);
```

Switch-case

Selezione a più vie: istruzione `switch-case`

Sintassi:

```
switch (espressione) {
        case valore-1: istruzione
        break;
        . . .
        case valore-n: istruzione
        break;
        default: istruzione-default
    }
```

Semantica:

1. Valutata `espressione`
2. Cercato primo `i` tale che `espressione == valore-i`
3. Se non trovato allora eseguita `istruzione-default`
4. L'esecuzione procede con l'istruzione che segue `switch`

Attenzione, se non c'è `break;` l'esecuzione procede con il `case` successivo

Esempio Switch-case

```
#include <stdio.h>

int main()

{     int a;

      scanf("%d",&a);
      switch (a){
         case 1: printf("uno \n"); break;
         case 2: printf("due \n"); break;
         case 3: printf("tre \n"); break;
         case 4: printf("quattro \n"); break;

         default: printf("altro \n");

      }

return 0;

}
```