

Tecniche della Programmazione, lez.6

Qualche ultima cosa sui tipi base: formati di conversione e “*conversioni di tipo*”

E poi programmazione con istruzioni strutturate:

- istruzioni CONDIZIONALI (if ...)
- Istruzioni ripetitive, istruzioni iterative, istruzioni di ciclo... (while, for, do_while)



... sul TIPO e sul VALORE di un'espressione

(determinazione statica e dinamica dell'uno e dell'altro)

L'op. / tra interi produce un valore intero; quello tra double (è un altro /) un valore double ...

```
int n, m, q, p;
```

```
double a, b;
```

```
... lettura di n(3), m(4),  
p(6), a(3.0), b(4.0)
```

```
q = n/m;
```

```
printf ("bla bla %d\n", m/n);
```

```
printf ("bla bla %g\n", a/b);
```

```
printf ("bla bla %g\n", b/a);
```

0

1

0.75

1.33

DI CHE TIPO è l'espressione a/b?
double

Il TIPO di un'espressione è
DETERMINATO STATICAMENTE
(cioè a tempo di compilazione).

CHE VALORE HA l'espressione a/b?
Boh!

Che ne so? Dipende dai valori che assumono le sottoespressioni **durante l'esecuzione del programma!**

Il VALORE dell'espressione è
DETERMINATO DINAMICAMENTE

Ancora sui Tipi Base Formati di conversione per reali

`%f` formato standard (decimale)

`%e` notazione scientifica (esponente)

`%g` la forma piu` breve tra `%f` e `%e`, eliminando 0 superflui

```
double f1=3.14, f2=0.00000313;
```

```
printf ("bla bla ... %f\n", f1);  
printf ("bla bla ... %e\n", f1);  
printf ("bla bla ... %f\n", f2);  
printf ("bla bla ... %e\n", f2);  
printf ("bla bla ... %g\n", f2);  
printf ("bla bla ... %g\n", f1);
```

```
NB  
3.130000e-006  
e`  
3.130000 x 10-6  
cioe`  
0.00000313
```

```
bla bla ... 3.140000  
bla bla ... 3.140000e+000  
bla bla ... 0.000003  
bla bla ... 3.130000e-006  
bla bla ... 3.13e-006  
bla bla ... 3.14
```

Cosa viene stampato se

```
f2=0.000313;
```



Ancora sui Tipi Base

TIPO = INSIEME DI VALORI e
INSIEME DI OPERAZIONI
ammissibili su quei valori

"Formati di conversione"



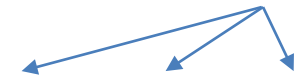
int interi in 32 bit
long int (64 bit)
short int (16 bit)

%d
%ld
%hd

float *Floating Point* in 32 bit
double come sopra ma 64 bit
Long double (128 bit)

%f, %e, %g
%f, %e, %g, %lf, %le, %lg
%Lf, %Le, %Lg

lettura



char caratteri ASCII (8bit) %c
(ASCII = American Standard Code for Information Interchange

[https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118033296.oth#:~:text=ASCII-ASCII%2C%20\(American%20Standard%20Code%20for%20Information%20Interchange\)%2C%20is%20assigned%20a%20unique%20binary%20string](https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118033296.oth#:~:text=ASCII-ASCII%2C%20(American%20Standard%20Code%20for%20Information%20Interchange)%2C%20is%20assigned%20a%20unique%20binary%20string)

<https://en.wikipedia.org/wiki/ASCII> https://en.wikipedia.org/wiki/Extended_ASCII)

NB Dimensioni in bit (dipendono dal sistema di programmazione)

Conversioni di tipo

La valutazione di un'espressione restituisce un VALORE, di un certo tipo. Il tipo dipende dalle componenti dell'espressione:

```
int i, j  
double a, b
```

$i*j$ è un'espressione **INTERA**, il risultato della cui valutazione è di tipo `int` (rappresentato in *compl. a due* in 32 bit...)

$a+b$ è un'espressione **REALE**: in questo caso il risultato è di tipo `double` (numero reale rappresentato in Floating Point su 64 bit) ...

Se vogliamo che il valore di un'espressione, di TIPO `T`, sia utilizzabile in un contesto dove ci si aspetta un valore di TIPO `T'` allora bisogna eseguire una conversione di tipo

valore di tipo `T` → valore equivalente ma di tipo `T'`

Conversioni di tipo

La valutazione di un'espressione restituisce un VALORE, di un certo tipo. Il tipo dipende dalle componenti dell'espressione:

```
int i, j  
double a, b
```

$i*j$ è un'espressione **INTERA**, il risultato della cui valutazione è di tipo `int` (rappresentato in *compl. a due* in 32 bit...)

$a+b$ è un'espressione **REALE**: il risultato è di tipo `double` (numero reale rappresentato in Floating Point su 64 bit) ...

Convertire il tipo di un'espressione vuol dire fare in modo che il valore calcolato durante la valutazione sia scritto (rappresentato) in un modo diverso dal normale, cioè sia di un tipo diverso da quello che sarebbe naturale.

La conversione può essere comandata dal programmatore (**esplicita**) oppure fatta automaticamente (**implicita**) durante l'esecuzione di un'istruzione.

Conversione Esplicita (casting)

```
int n=10, m=4;
```

```
... n ...
```

```
... (double)n ...
```

è un'espressione intera il cui valore è 10

è un'espressione double, il cui valore è 10.0 (FP/64bit)

Es. calcolo della media di tre numeri

```
int n1=10, n2=4, n3=12, somma;
```

```
    somma = n1+n2+n3;
```

```
    ... somma/3 ...
```

/* n1+n2+n3 è un'espressione ???; somma ora contiene un valore di tipo ??? (?) */

/* somma/3 è un'espressione ???, il cui valore è ??? */

/* NB la media esatta sarebbe ??? ...*/



Conversione Esplicita (casting)

Vedi Esercizi

```
int n=10, m=4;
```

```
... n ...
```

```
... (double)n ...
```

è un'espressione intera il cui valore è 10

è un'espressione double, il cui valore è 10.0 (FP/64bit)

Es. calcolo della media di tre numeri

```
int n1=10, n2=4, n3=12, somma;
```

```
somma = n1+n2+n3;
```

```
... somma/3 ...
```

/* n1+n2+n3 è un'espressione intera; somma ora
contiene un valore intero (26) */

/* somma/3 è un'espressione intera il cui valore è 8 */

/* NB la media esatta sarebbe 8.666667 ... circa */

Quindi?

Come si scrive un'espressione double che rappresenti il valore esatto della media?

? ... (double) (somma/3) ...

? ... (double) somma/3 ...

? ... somma / (double) 3 ...

? ... somma/3 (double) ...

? ... (double) somma / (double) 3



queste sono tutte conversioni esplicite ...
magari verificate con un piccolo programma ...

Conversione Implicita

Vedi
Approfondimenti

Avviene, in certi casi,

quando un'espressione che produce un valore di tipo T1, appare in un contesto in cui è atteso (voluto) un valore di tipo T2:

allora il valore di tipo T1 viene **convertito** al tipo T2 (**riscritto ... rappresentato come valore T2**) e come tale viene poi usato.

```
int i=137;
double a;
a = 12;      /* a ora contiene 12.0 */
a = i;       /* a ora contiene 137.0 */
a = i/2;     /* a ora contiene 68.0 */

sqrt(i) /* sqrt si aspetta un float ... e usa 137.0 */
```

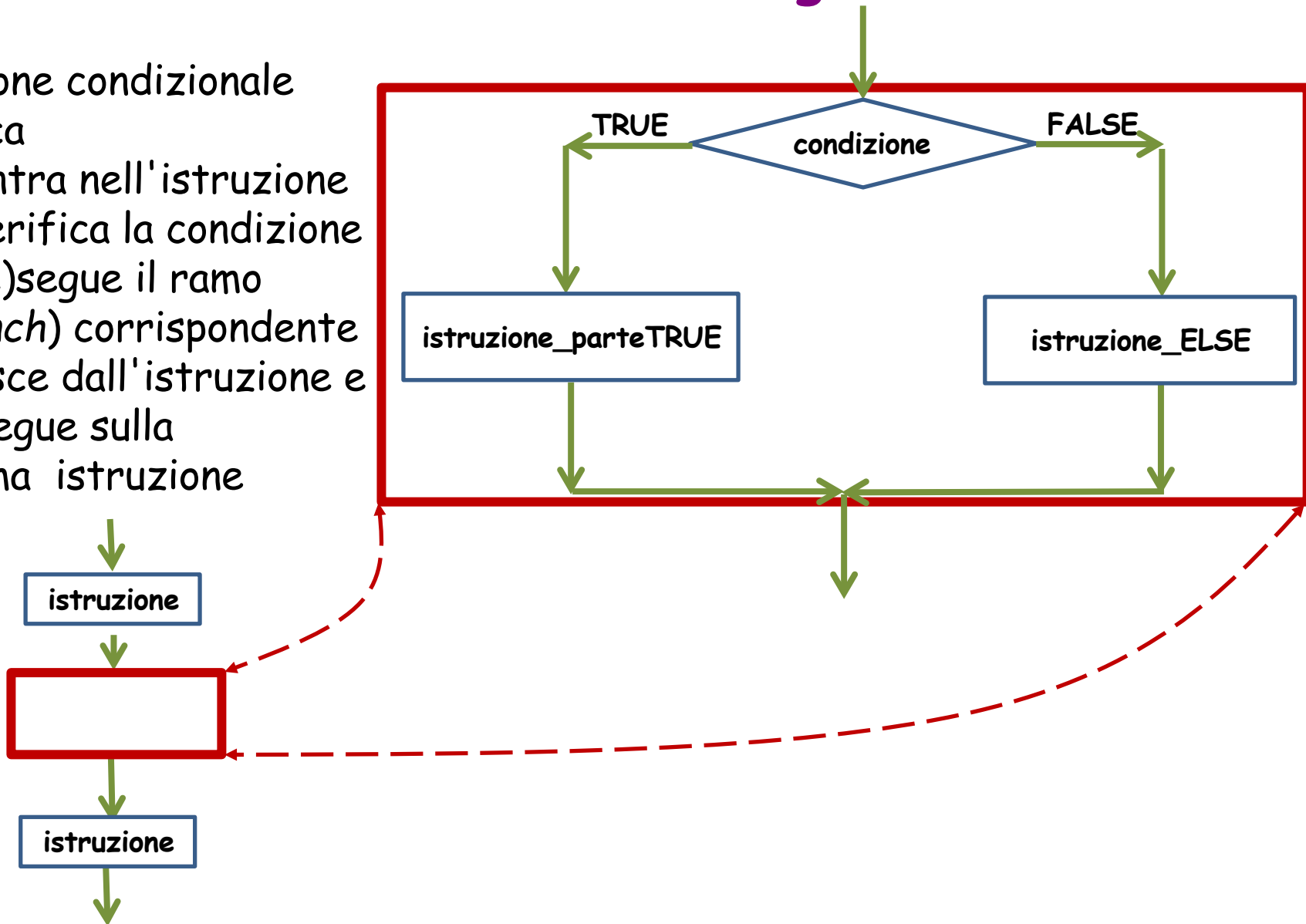
in ognuno di questi casi c'è stata una
conversione **implicita** da int a double/float



Istruzione condizionale: versione generica

istruzione condizionale
generica

- si entra nell'istruzione
- si verifica la condizione
- si (e)segue il ramo
(branch) corrispondente
- si esce dall'istruzione e
si prosegue sulla
prossima istruzione

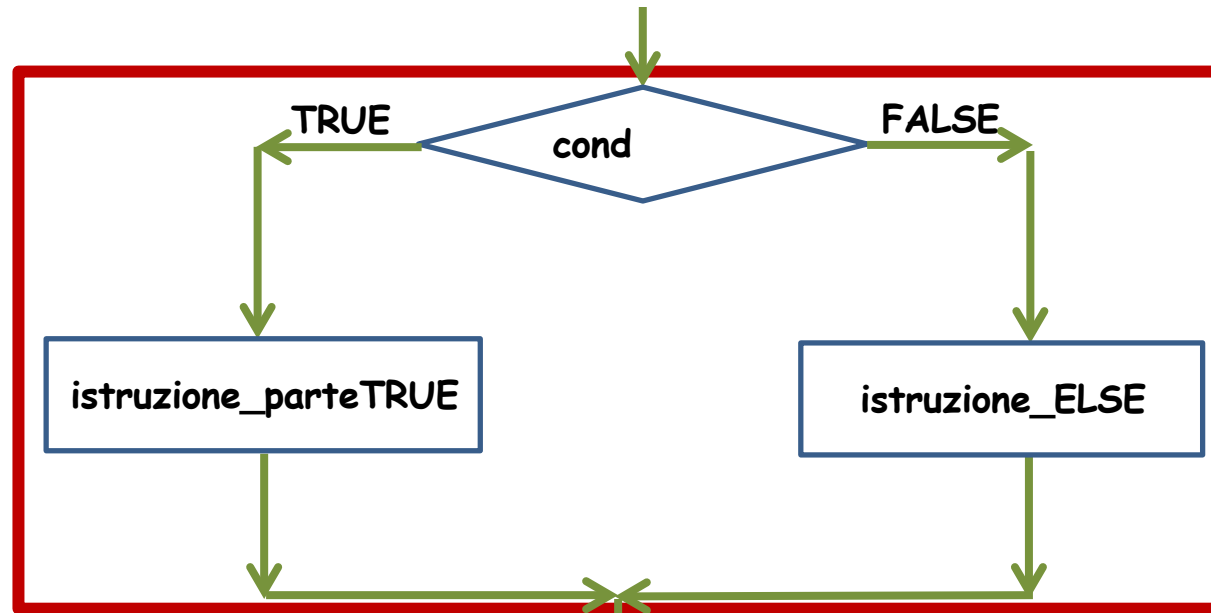


Istruzione condizionale generica: if_else



istruzione condizionale generica

- si entra nell'istruzione
- si verifica la condizione
- si esegue il ramo (branch) corrispondente
- si esce dall'istruzione e si prosegue sulla prossima istruzione

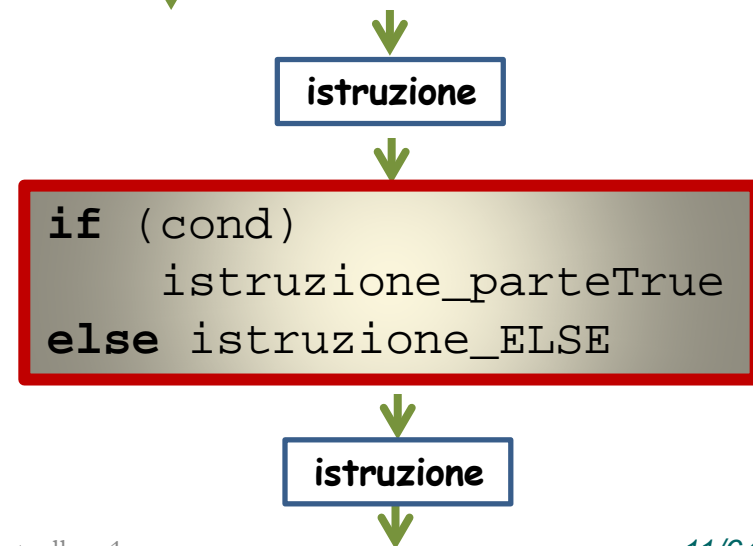


è realizzata in C mediante l'istruzione IF_ELSE (if_else)

la **condizione** (cond) è una **espressione logica**:

se la sua valutazione è 1 (o comunque un valore diverso da zero) → TRUE

se la sua valutazione è 0 → FALSE



Istruzione condizionale generica: `if_else` (esempio)

```
/* programma che legge un numero intero da input e stampa un
commento dicendo se il numero è negativo o non negativo */
#include <stdio.h>

int main () {
    int num;                /* il numero letto da input */

    printf ("Caro/a utente, dammi un numero: ");
    scanf ("%d", &num);    /* lettura */

    if (num < 0)
        printf ("beh, %c un numero negativo\n", 138);
    else {
        printf ("beh, %c un numero NON negativo\n", 138);
    }

    printf ("\nFINE programma\n");
    return 0;
}
```

istruzione_parte_True è una singola istruzione, quindi {} non servono.
Idem per Istruzione_parte_Else, ma lì le abbiamo messe perché a volte siamo ridondanti

Istruzione condizionale generica: if_else

Programma che legge un numero da INPUT e stampa un commento dicendo se il numero è negativo o non negativo

```
/* programma che legge un numero intero da input e stampa un  
commento dicendo se il
```

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

```
int main () {  
    int num;
```

```
    printf ("Caro/a
```

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

```
    if (num < 0)
```

```
        printf ("beh, %c un numero negativo\n", 138);
```

```
    else {
```

```
        printf ("beh, %c un numero NON negativo\n", 138);
```

```
    }
```

```
    printf ("\nFINE programma\n");
```

```
    return 0;
```

```
}
```

```
Caro/a utente, dammi un numero: -12  
beh, è un numero negativo  
  
FINE programma
```

```
/* lettura */
```

Istruzione condizionale generica: if_else

Programma che legge un numero da INPUT e stampa un commento dicendo se il numero è negativo

```
/* programma che legge un numero  
commento dicendo se il numero è  
#include <stdio.h>
```

```
Caro/a utente, dammi un numero: -12  
beh, è un numero negativo  
  
FINE programma
```

```
int main () {  
    int num;
```

```
    printf ("Caro/a ut  
    scanf ("%d", &num)
```

```
    if (num < 0)
```

```
        printf ("beh, %c un numero negativo\n", 138);
```

```
    else {
```

```
        printf ("beh, %c un numero NON negativo\n", 138);
```

```
    }
```

```
    printf ("\nFINE programma\n");
```

```
    return 0;
```

```
}
```

NB In questo caso l'istruzione `_ELSE` (come anche l'altra) è una singola istruzione semplice, quindi le `{` e `}` non servono (ma se si mettono non succede nulla di male)

Istruzione condizionale generica: if_else

Programma che legge un numero da INPUT e stampa un commento dicendo se il numero è pari o dispari

```
#include <stdio.h>
int main () {
    int num;          /* il numero letto da input */

    printf ("Caro/a utente, dammi un numero: ");
    scanf ("%d", &num);          /* lettura */

    if (    😊    )
        printf ("beh, è un numero pari\n");
    else
        printf ("beh, è un numero dispari\n");

    printf ("\nFINE programma\n");
    return 0;
}
```

Istruzione condizionale generica: if_else

Programma che legge un numero da INPUT e stampa un commento dicendo se il numero è pari o dispari

```
#include <stdio.h>
int main () {
    int num;          /* il numero letto da input */

    printf ("Caro/a utente, dammi un numero: ");
    scanf ("%d", &num);          /* lettura */

    if (num%2 == 0)
        printf ("beh, è un numero pari\n");
    else
        printf ("beh, è un numero dispari\n");

    printf ("\nFINE programma\n");
    return 0;
}
```


Istruzione condizionale generica: if_else (altro esempio)

Programma che legge due numeri interi da INPUT e stampa un commento indicando il più grande tra i due numeri

ALG

- 0) i due numeri e una variabile per contenere quello che sarà il massimo tra i due
- 1) input num1, num2
- 2) se num1 > num2
 ALLORA max = num1
 ALTRIMENTI max=num2
- 3) output max

```
#include <stdio.h>
int main () {
    int num1, num2,          /* yawn, i due numeri in input */
        max;                /* per contenere il mas ... yawn */

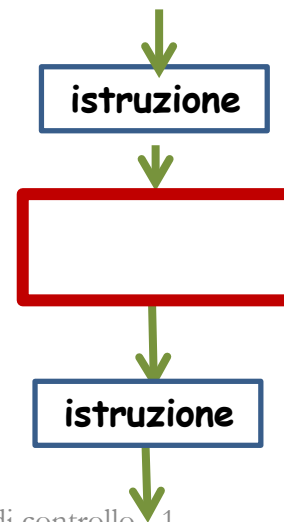
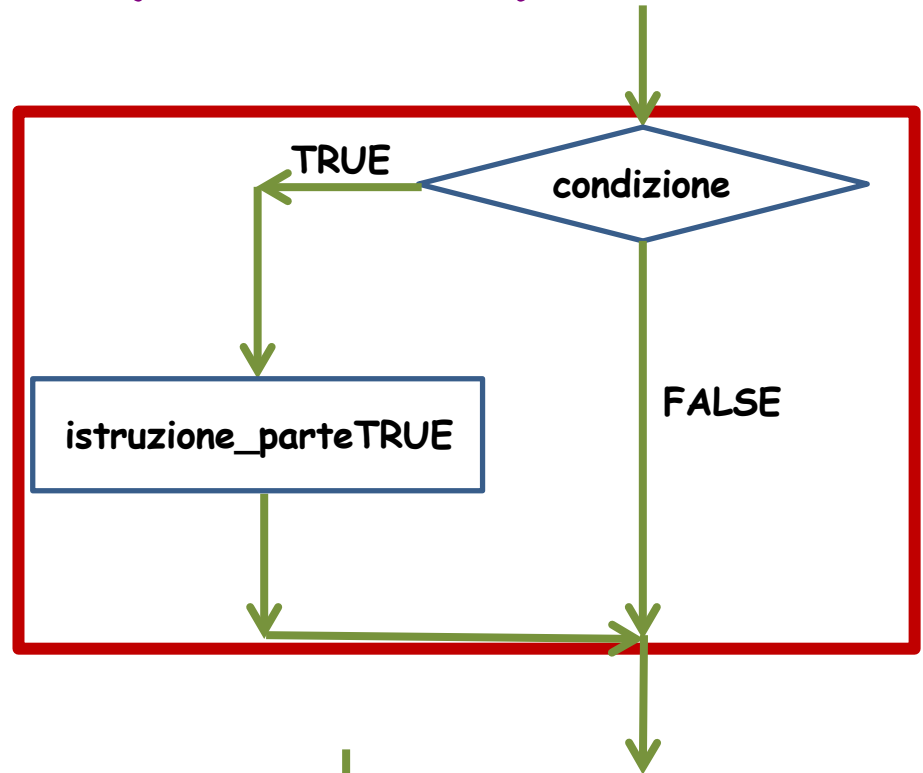
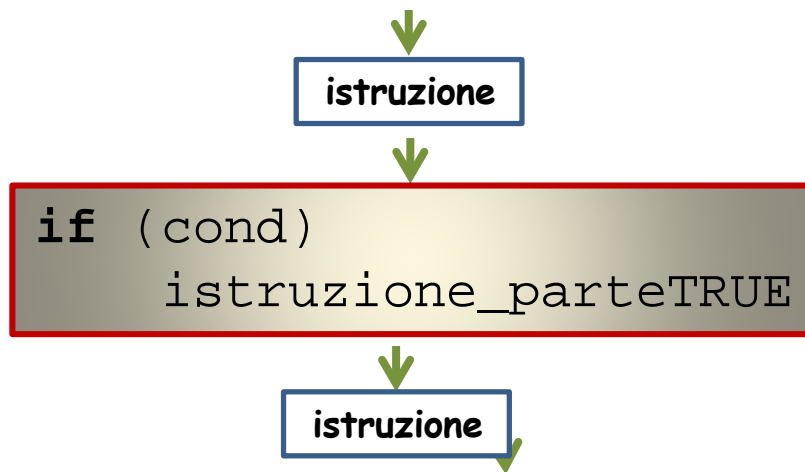
    printf ("yawn, dammi due numeri sulla stessa riga (separati ...): ");
    scanf ("%d %d", &num1, &num2);    /* lett...yawn...ura */

    if (num1>num2)          /* passo 2) assegnazione a max del massimo tra num1 e num2 */
        max = num1;
    else
        max=num2;

    printf ("il max tra %d e %d %c %d \n\n", num1, num2, 138, max);    /* pass0 3) output */
    printf ("\nFINE programma\n");
    return 0;
}
```

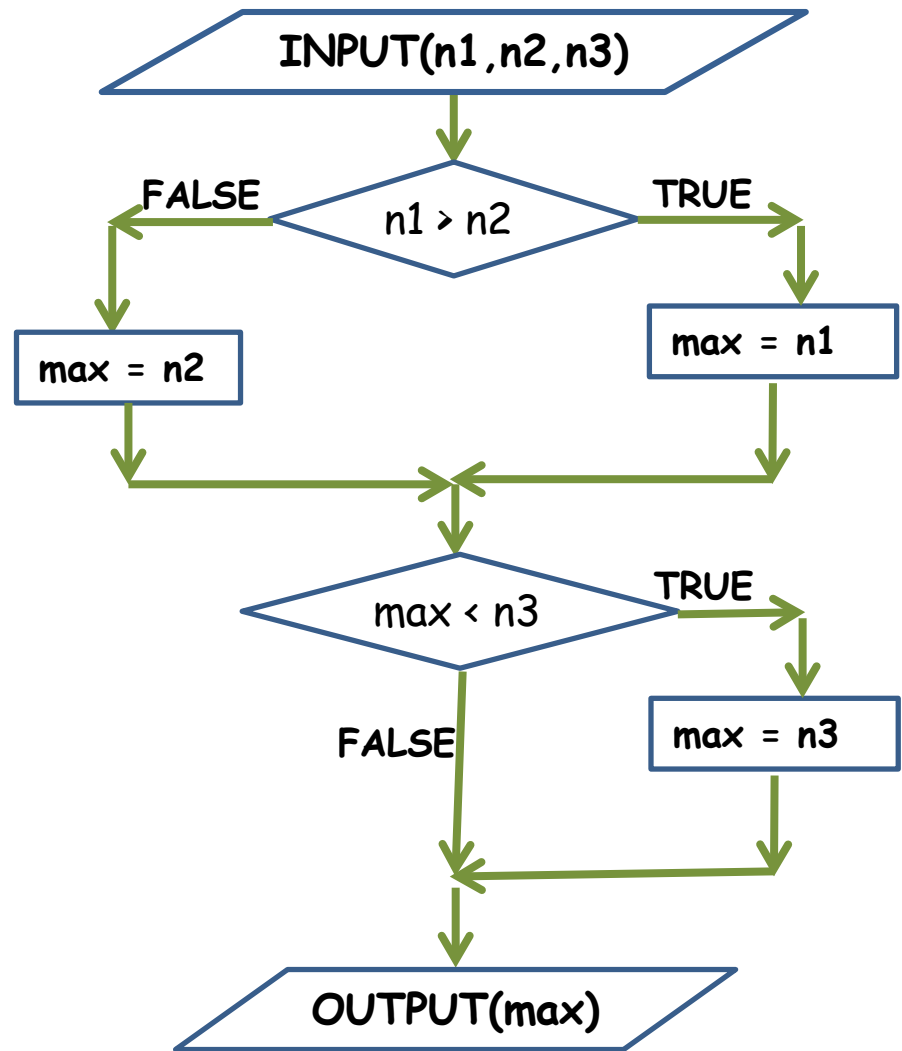
Istruzione condizionale: if (senza else)

" altrimenti
non fare nulla e prosegui ... "



Massimo tra tre numeri

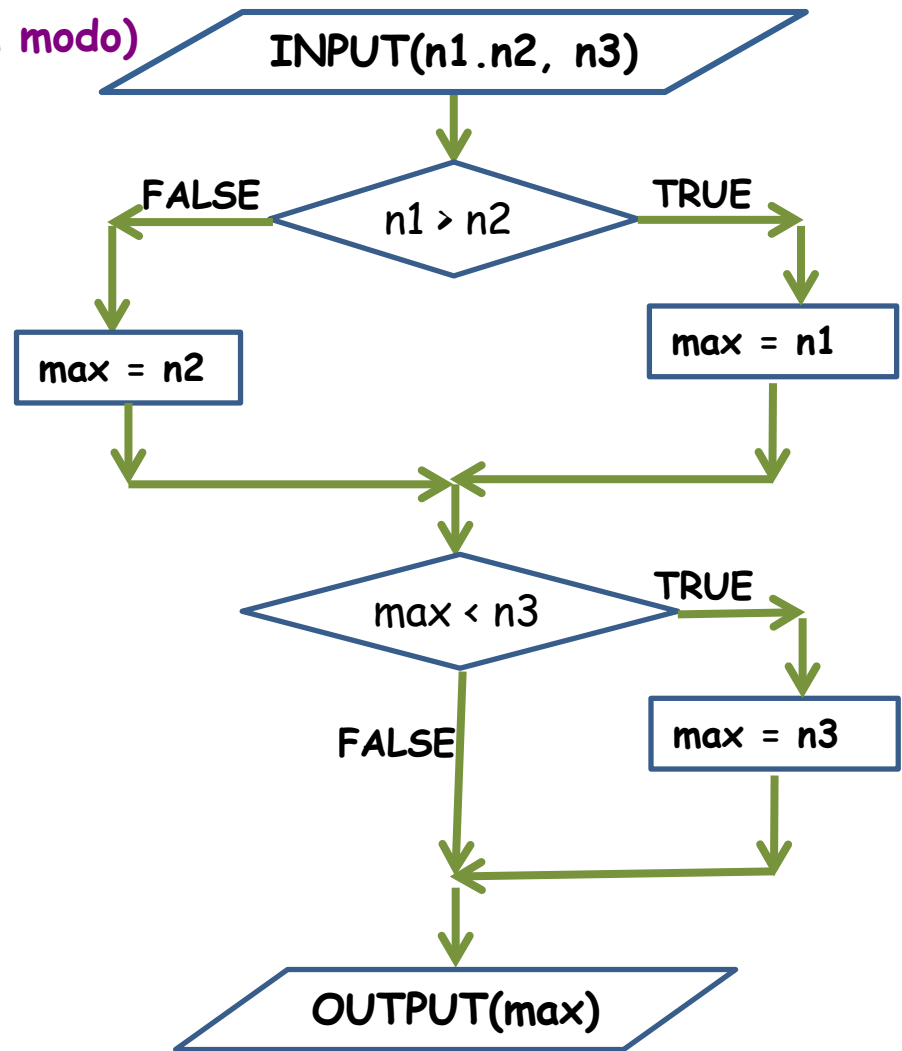
- 0) i dati: $n1, n2, n3$ (input), e max
- 1) INPUT ($n1, n2, n3$)
- 2) Inizializzazione di max :
SE $n1 > n2$
 $max = n1$
- 3) SE $n3 > max$
 $max = n3$
- 4) OUTPUT(max)



Problems?

Massimo tra tre numeri (un modo)

- 0) i dati: $n1, n2, n3$ (input), e max
- 1) INPUT ($n1, n2, n3$)
- 2) Inizializzazione di max :
SE $n1 > n2$
 $max = n1$
ALTRIMENTI $max = n2$
- 3) SE $n3 > max$
 $max = n3$
- 4) OUTPUT(max)



Massimo tra tre numeri

```
int main () {
    int n1, n2, n3,          /* i tre numeri */
        max;                /* per rappresentare il massimo
                             tra i numeri controllati */

    printf ("Caro/a utente, dammi tre numeri: ");
    scanf ("%d %d %d", &n1, &n2, &n3);          /* lettura */

    if ( 😊 )
        max = n1;          /* per ora vince n1 */
    else max = n2;        /* per ora vince n2 */

    /* ora se n3 è più grande del massimo parziale, ... */
    😊

    printf ("beh, il massimo %c %d\n", 138, max);

    printf ("\nFINE programma\n");
    return 0;
}
```

Massimo tra tre numeri

```
int main () {
    int n1, n2, n3,          /* i tre numeri */
        max;                /* per rappresentare il massimo
                             tra i numeri controllati */

    printf ("Caro/a utente, dammi tre numeri: ");
    scanf ("%d %d %d", &n1, &n2, &n3);          /* lettura */

    if (n1 > n2)
        max = n1;          /* per ora vince n1 */
    else max = n2;        /* per ora vince n2 */

    /* ora se n3 è più grande del massimo parziale, ... */
    if (max < n3)          /* vince n3 */
        max = n3;

    /* se non vince n3, max non cambia */

    printf ("beh, il massimo %c %d\n", 138, max);

    printf ("\nFINE programma\n");
    return 0;
}
```

Massimo tra quattro numeri (alg.2)



Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) i dati: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

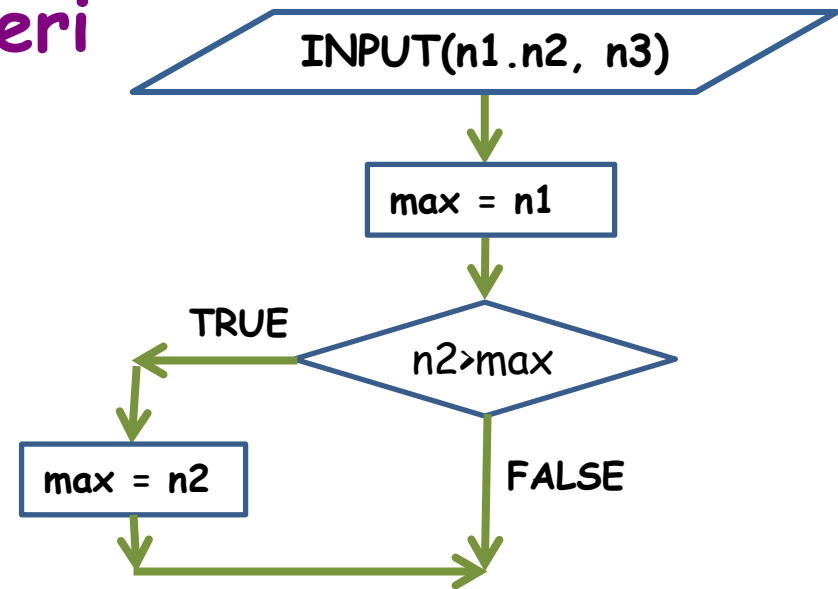
prima di continuare, verificare che l'algorithm funziona, eseguendolo

vedi Approfondimentii

Massimo tra quattro numeri

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) i dati: $n1, n2, n3, n4$ (input), e max
- 1) INPUT ($n1, n2, n3, n4$)
- 2) Inizializzazione di max: $max = n1$
- 3) SE $n2 > max$
 $max = n2$
- 4) SE $n3 > max$... fai $max = n3$
- 5) SE $n4 > max$... fai $max = n4$
- 6) OUTPUT(max)



```
max = n1; /* inizializzazione max. parz */
```

```
if (n2 > max)
```

```
max = n2; /* max superato da n2 */
```

```
printf ("beh, il massimo %c %d\n", 138, max);
```


Massimo tra quattro numeri

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) i dati: $n1, n2, n3, n4$ (input), e max
- 1) INPUT ($n1, n2, n3, n4$)
- 2) Inizializzazione di max: $max = n1$
- 3) SE $n2 > max$
 $max = n2$
- 4) SE $n3 > max$... fai $max = n3$
- 5) SE $n4 > max$... fai $max = n4$
- 6) OUTPUT(max)

```
max = n1; /* inizializzazione max. parz */
```

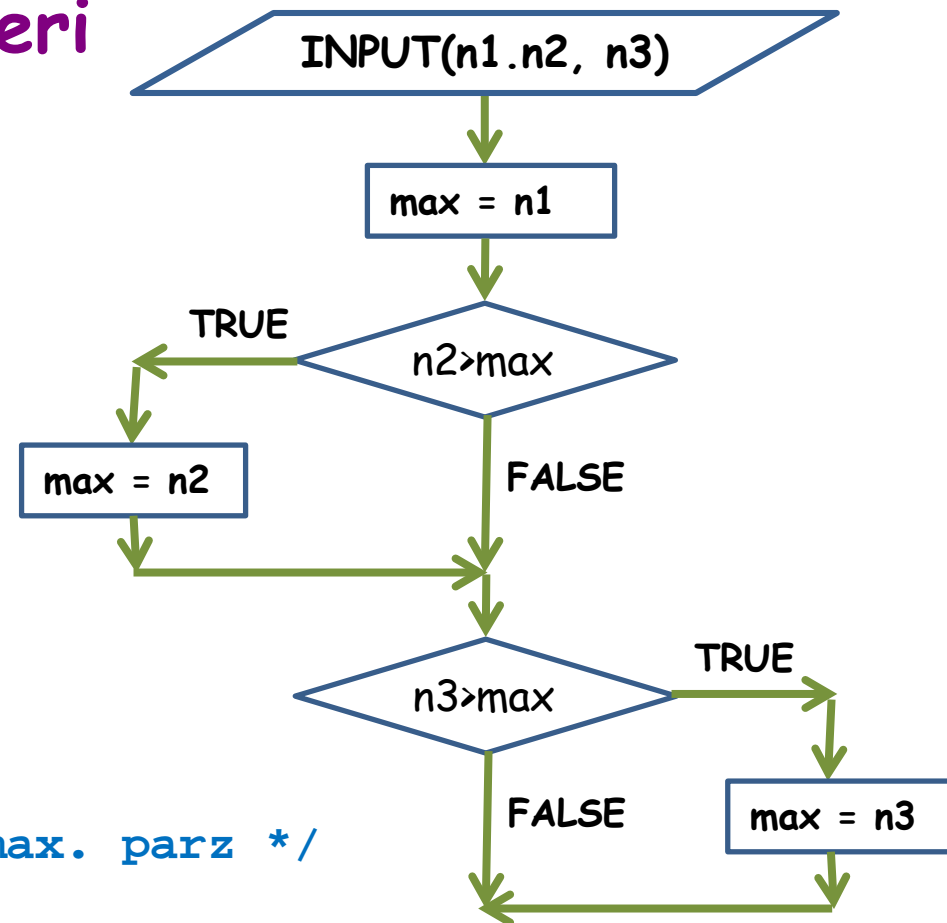
```
if (n2 > max)
```

```
    max = n2;      /* max superato da n2 */
```

```
if (max < n3)     /* se vince n3 */
```

```
    max = n3;
```

```
printf ("beh, il massimo %c %d\n", 138, max);
```



Massimo tra quattro numeri

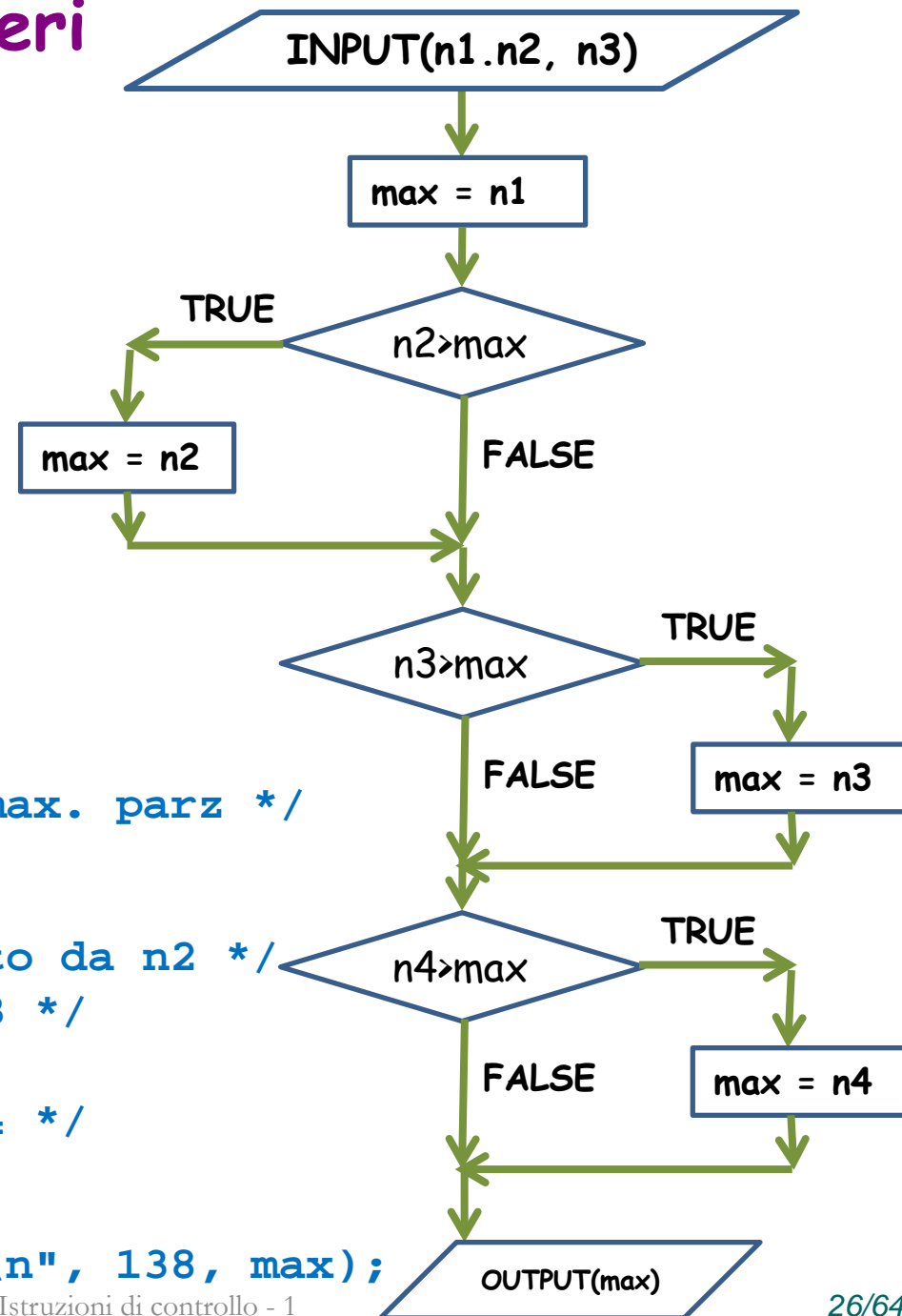
Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) i dati: $n1, n2, n3, n4$ (input), e max
- 1) INPUT ($n1, n2, n3, n4$)
- 2) Inizializzazione di max: $max = n1$
- 3) SE $n2 > max$
 $max = n2$
- 4) SE $n3 > max$... fai $max = n3$
- 5) SE $n4 > max$... fai $max = n4$
- 6) OUTPUT(max)

```
max = n1; /* inizializzazione max. parz */
```

```
if (n2 > max)
    max = n2; /* max superato da n2 */
if (max < n3)
    max = n3; /* se vince n3 */
if (max < n4)
    max = n4; /* se vince n4 */
```

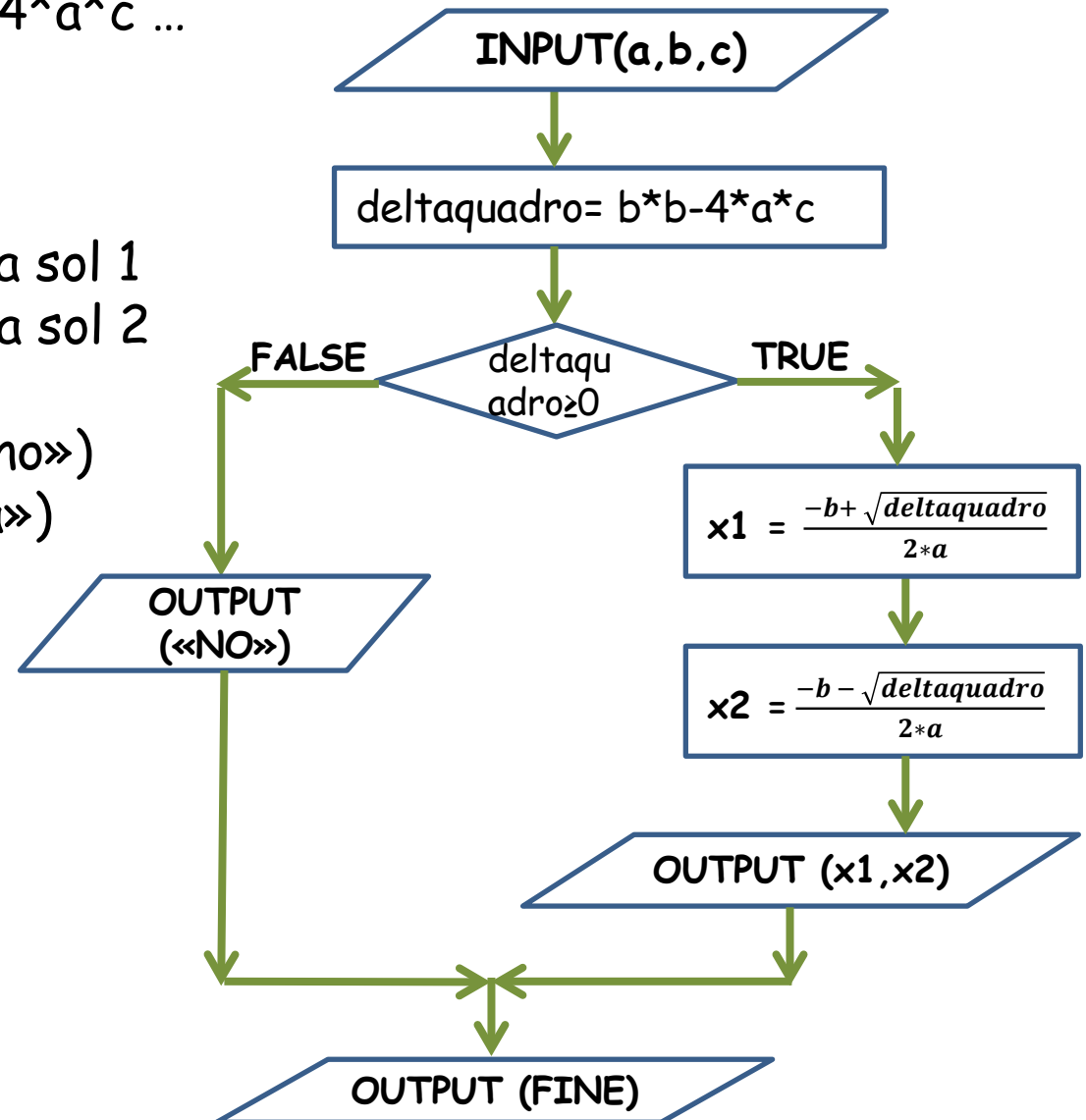
```
printf ("beh, il massimo %c %d\n", 138, max);
```



Algoritmo strutturato per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi Δ , ... $2*a$, $4*a*c$...

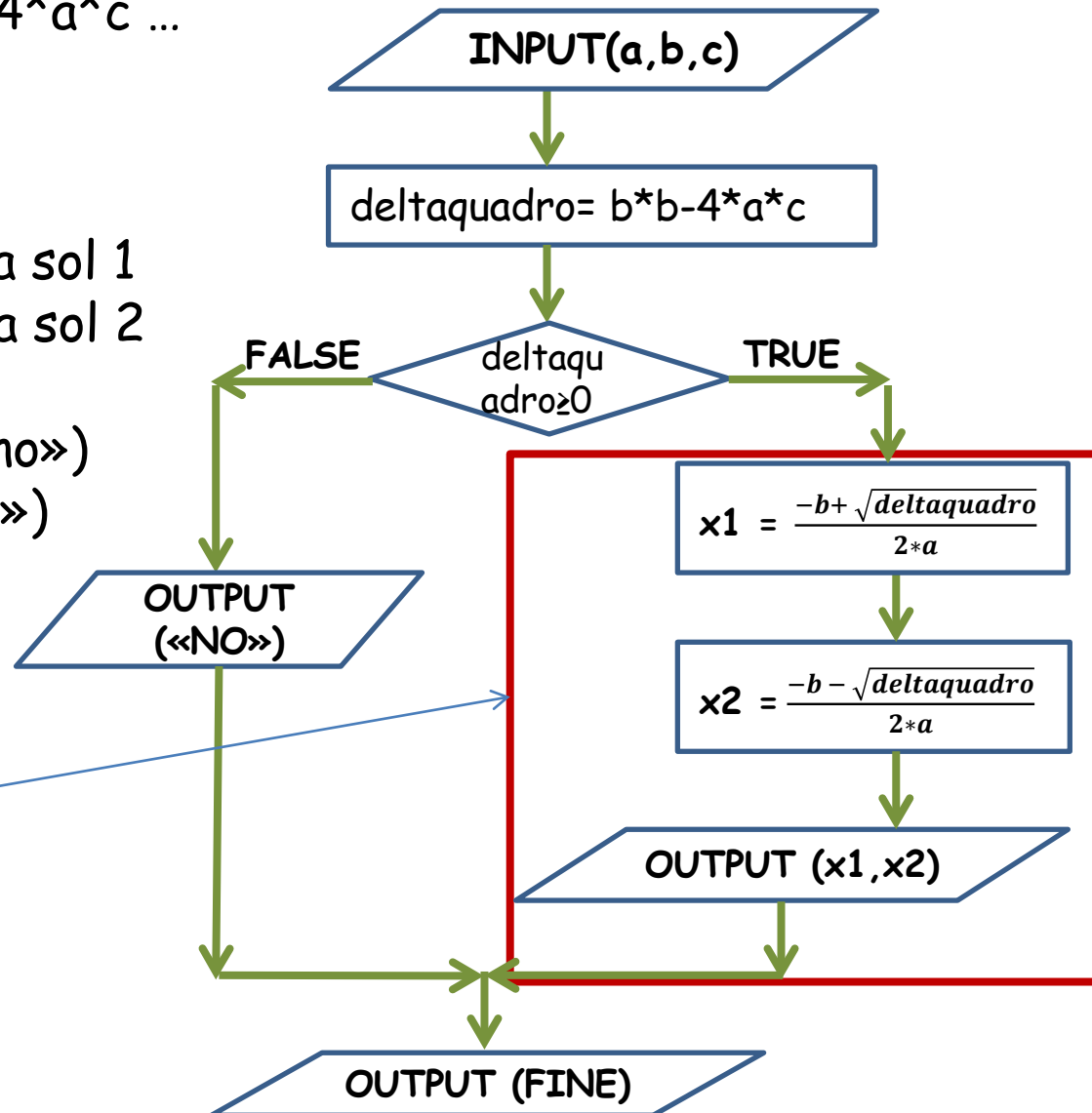
- 1) INPUT (a, b, c)
- 2) $\Delta = b*b - 4*a*c$
- 3) SE ($\Delta \geq 0$)
 - 3.1) x1 = formula per la sol 1
 - 3.2) x2 = formula per la sol 2
 - 3.3) OUTPUT(x1, x2)
- ALTRIMENTI OUTPUT(«no»)
- 4) OUTPUT(«fine programma»)



ancora sulle { }

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, ... $2*a$, $4*a*c$...

- 1) INPUT (a, b, c)
- 2) $\text{deltaquadro} = b*b - 4*a*c$
- 3) SE ($\text{deltaquadro} \geq 0$)
 - 3.1) x1 = formula per la sol 1
 - 3.2) x2 = formula per la sol 2
 - 3.3) OUTPUT(x1, x2)
- ALTRIMENTI OUTPUT(«no»)
- 4) OUTPUT(«fine programma»)



Blocco
(istruzione composta)

ancora sulle { }

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, ... $2*a$, $4*a*c$...

- 1) INPUT (a, b, c)
- 2) deltaquadro = $b*b - 4*a*c$
- 3) SE (deltaquadro \geq 0)
 - 3.1) x1 = formula per la sol 1
 - 3.2) x2 = formula per la sol 2
 - 3.3) OUTPUT(x1, x2)
- ALTRIMENTI OUTPUT(«no»)
- 4) OUTPUT(«fine programma»)

parentesi { } qui indispensabili per delimitare il blocco (perché è composto da più istruzioni)

parentesi { } qui NON indispensabili

```
deltaq = b*b - 4*a*c;          /* 2) */
if (deltaq>=0) {              /* 3 */
    x1 = ( -b + sqrt(deltaq) )/(2*a);
    x2 = ( -b - sqrt(deltaq) )/(2*a);

    printf ("l'equazione . . .(%g, %g, %g) ha\n", a, b, c);
    printf ("prima soluzione: %g\n", x1);
    printf ("seconda soluzione: %g\n", x2);
}
else printf("tsk, tsk... soluzioni complesse coniugate");
...
```

Blocco
(istruzione
composta)

... parentesi indispensabili per delimitare un ...

BLOCCO di istruzioni

che succede senza { }? (cioè senza delimitatori del blocco)

```
deltaq = b*b - 4*a*c;          /* 4) */
if (deltaq >= 0) {
    x1 = ( -b + sqrt(deltaq) ) / (2*a);
    x2 = ( -b - sqrt(deltaq) ) / (2*a);
    printf ("l'equazione ...(%g, %g, %g) ha ...);
    printf ("prima soluzione: %g\n", x1);
    printf ("seconda soluzione: %g\n", x2);
}
else
    printf("tsk, tsk... soluzioni complesse ...
```

... parentesi indispensabili per delimitare un ...

BLOCCO di istruzioni
che succede senza ?

(ora solo questa è la istruzione_parteTRUE)

```
deltaq = b*b - 4*a*c;          /* 4) */
if (deltaq >= 0) {
    x1 = ( -b + sqrt(deltaq) ) / (2*a);

    x2 = ( -b - sqrt(deltaq) ) / (2*a);
    printf ("l'equazione ...(%g, %g, %g) ha ...);
    printf ("prima soluzione: %g\n", x1);
    printf ("seconda soluzione: %g\n", x2);
}
else
    printf("tsk, tsk... soluzioni complesse ...
```

istruzioni dopo
istruzione_parte
e TRUE
MA
indipendenti
dall'if

INOLTRE!!!!

il compilatore ha interpretato l'if come un "if senza else" e adesso non ci capisce più niente
else non associato ad un if ... eeeekkkk!!!!

Massimo tra tre numeri altra tecnica (alg. 3)



Se n_1 è più grande di n_2 , allora il massimo è lui oppure n_3 ...

0) i dati: n_1, n_2, n_3 (input), e max

1) INPUT (n_1, n_2, n_3)

2) SE $n_1 > n_2$

2.1) SE $n_1 > n_3$

2.1.1) max = n_1

ALTRIMENTI

2.2.2) max = n_3

ALTRIMENTI

2.2) SE $n_2 > n_3$

2.2.1) max = n_2

ALTRIMENTI

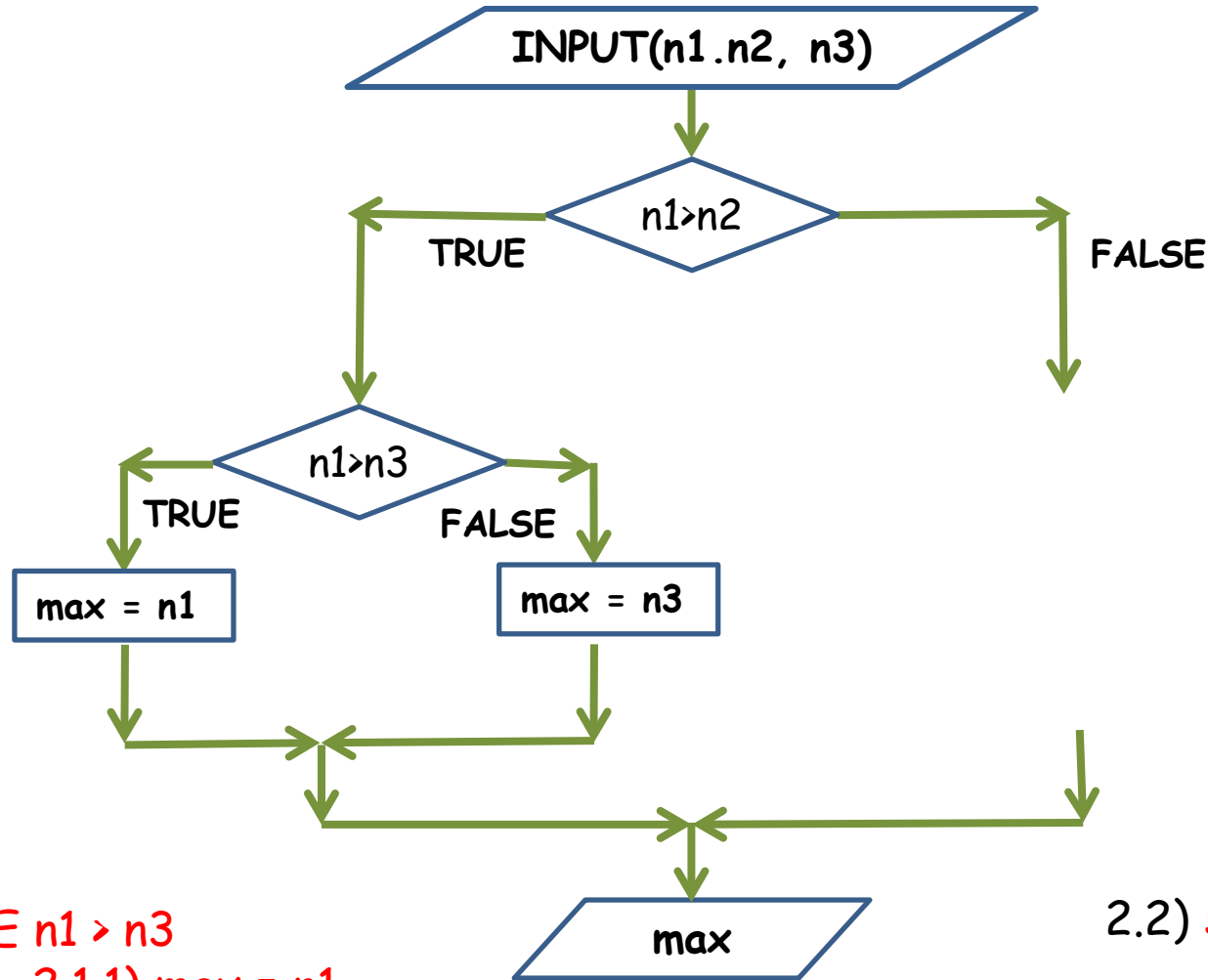
2.2.2) max = n_3

3) OUTPUT(max)

disegnare il diagramma di flusso



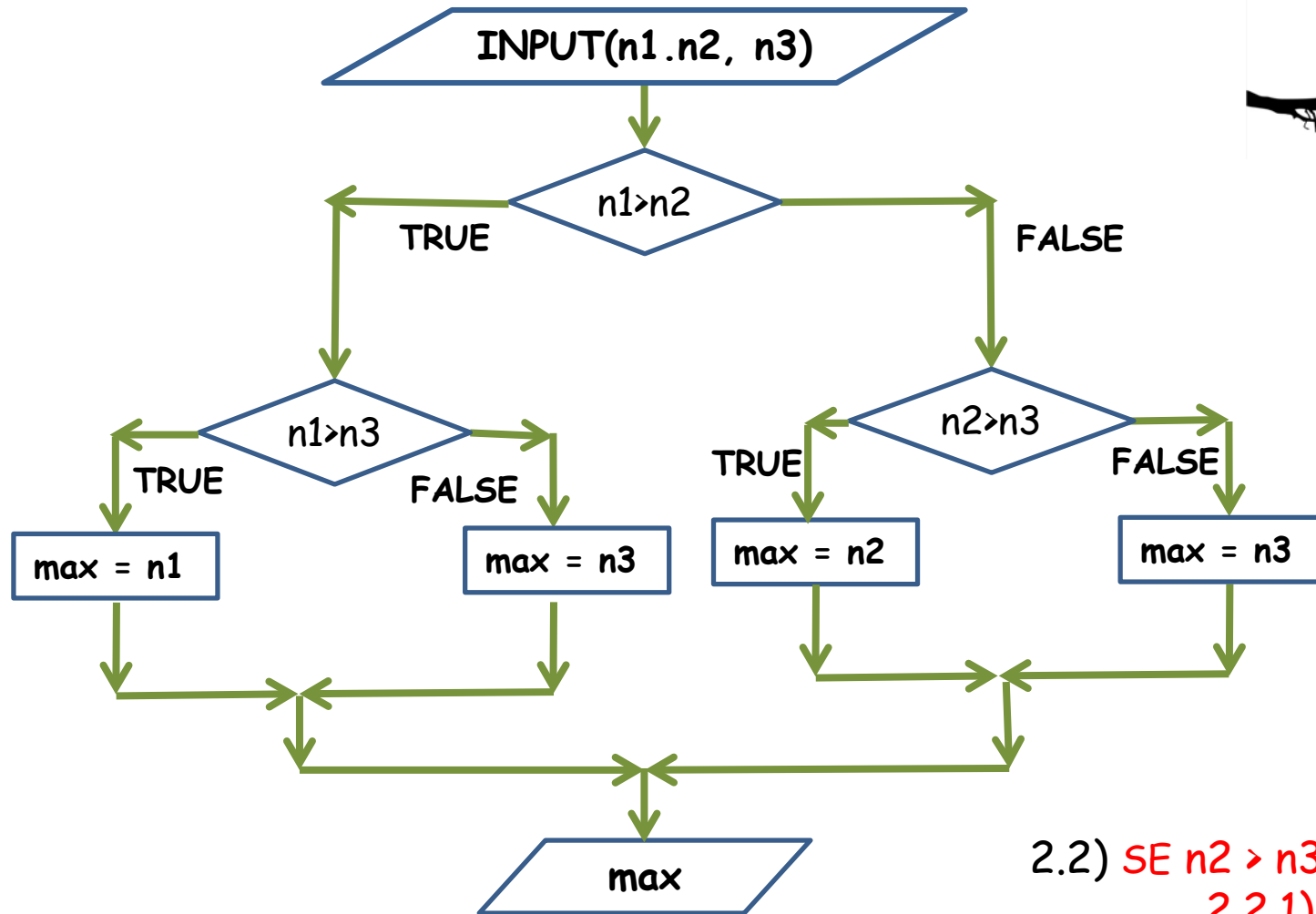
Massimo tra tre numeri altra tecnica (alg. 3)



2.1) SE $n1 > n3$
2.1.1) $max = n1$
ALTRIMENTI
2.2.2) $max = n3$

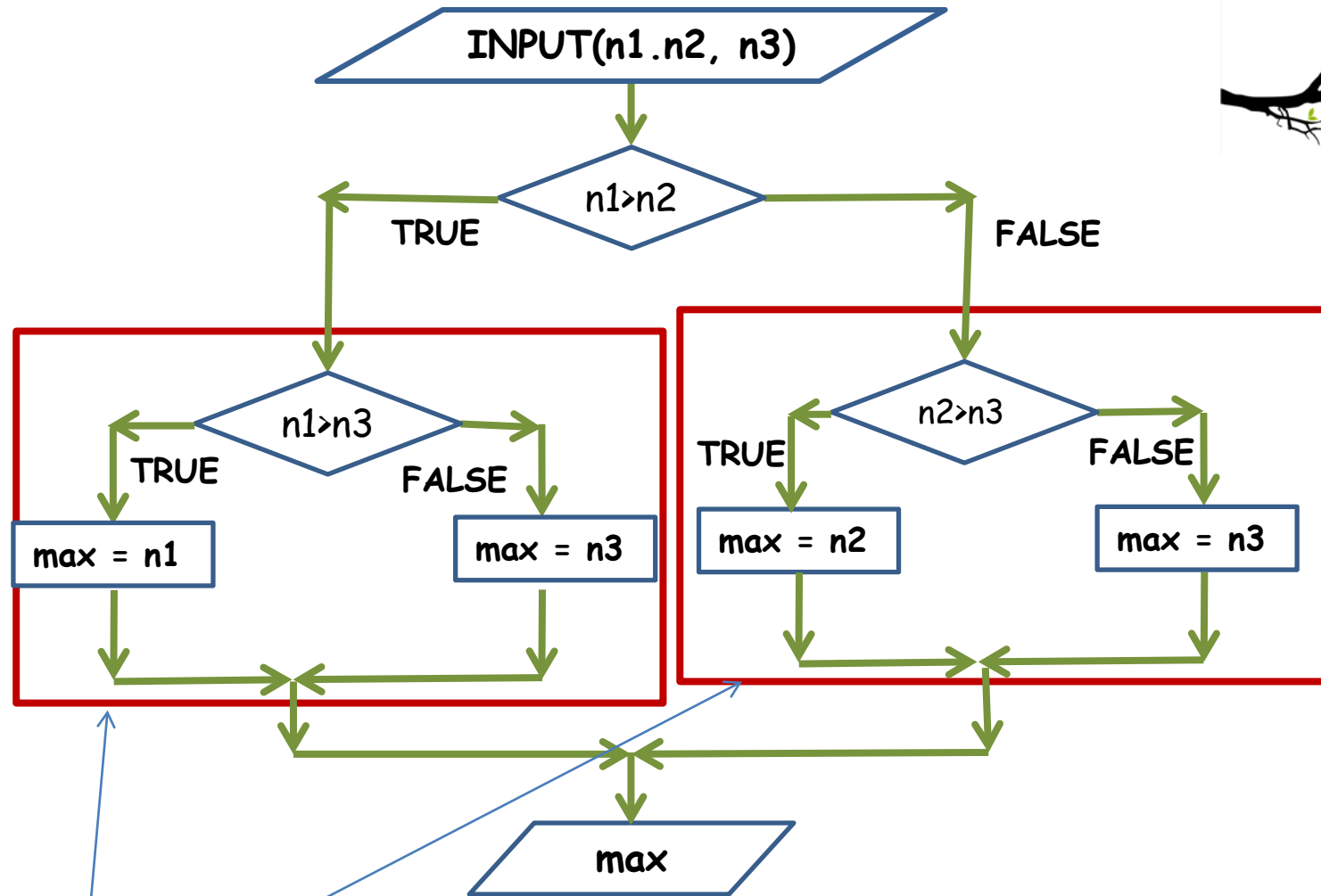
2.2) SE $n2 > n3$
2.2.1) $max = n2$
ALTRIMENTI
2.2.2) $max = n3$

Massimo tra tre numeri altra tecnica (alg. 3)



2.2) SE $n2 > n3$
2.2.1) $\max = n2$
ALTRIMENTI
2.2.2) $\max = n3$

Massimo tra tre numeri altra tecnica (alg. 3)



istruzioni strutturate (condizionali) ANNIDATE
dentro alla prima istruzione condizionale

Scrivere il programma corrispondente
(vedi Esercizi)

Massimo tra tre numeri (alg. 4)

Se n_1 è più grande di n_2 e di n_3 , allora il massimo è lui
altrimenti ... il massimo è n_2 o n_3 ...

0) i dati: n_1, n_2, n_3 (input), e max

1) INPUT (n_1, n_2, n_3)

2) SE ☺

2.1) max = n_1

ALTRIMENTI

2.2) ☺

3) OUTPUT(max)

Massimo tra tre numeri (alg. 4)

Se n_1 è più grande di n_2 e di n_3 , allora il massimo è lui
altrimenti ... il massimo è n_2 o n_3 ...

0) i dati: n_1, n_2, n_3 (input), e max

1) INPUT (n_1, n_2, n_3)

2) SE ($n_1 > n_2$) && ($n_1 > n_3$)

2.1) max = n_1

ALTRIMENTI

2.2) SE ($n_2 > n_3$)

2.2.1) max = n_2

ALTRIMENTI

2.2.2) max = n_3

3) OUTPUT(max)

disegnare il diagramma di flusso
fermandosi prima di 2.1)



Massimo tra tre numeri (alg. 4)

0) i dati: n_1, n_2, n_3 (input), e max

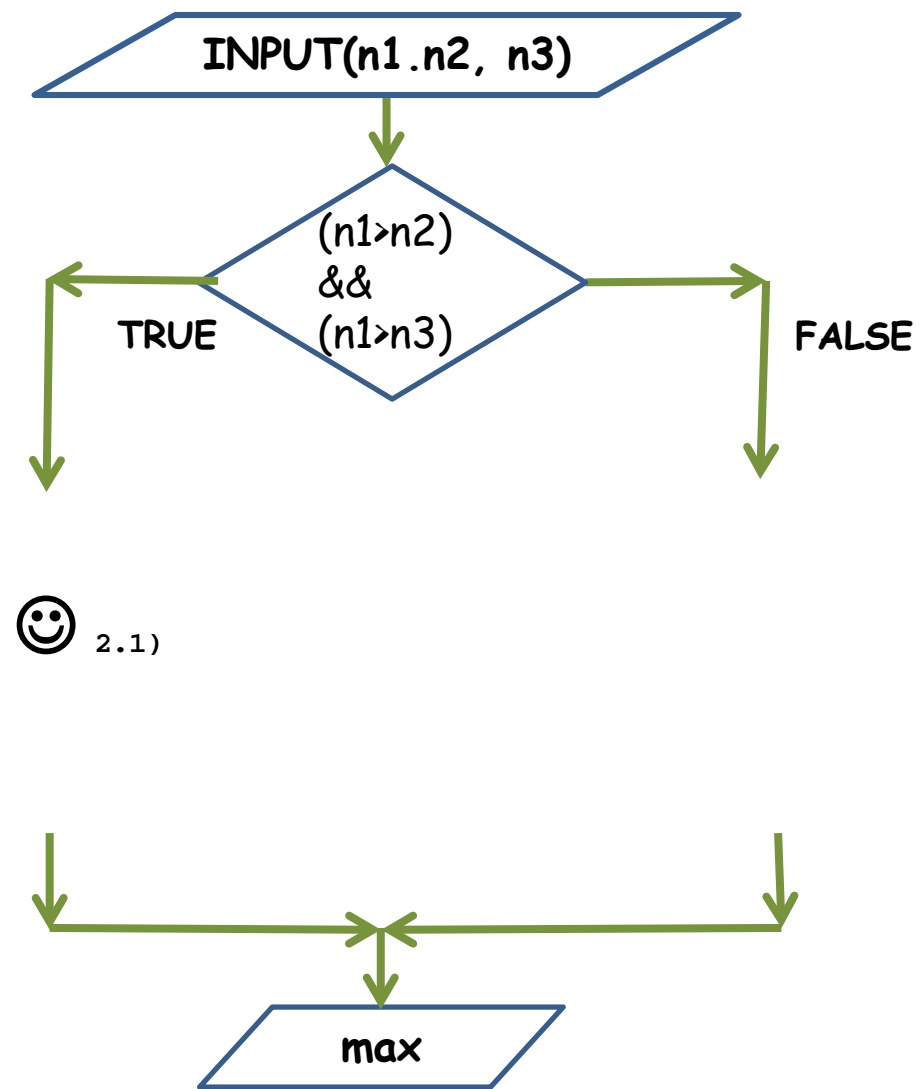
1) INPUT (n_1, n_2, n_3)

2) SE ($n_1 > n_2$) && ($n_1 > n_3$)
2.1) max = n_1

ALTRIMENTI

2.2) SE ($n_2 > n_3$)
2.2.1) max = n_2
ALTRIMENTI
2.2.2) max = n_3

3) OUTPUT(max)



Massimo tra tre numeri (alg. 4)

0) i dati: n_1, n_2, n_3 (input), e max

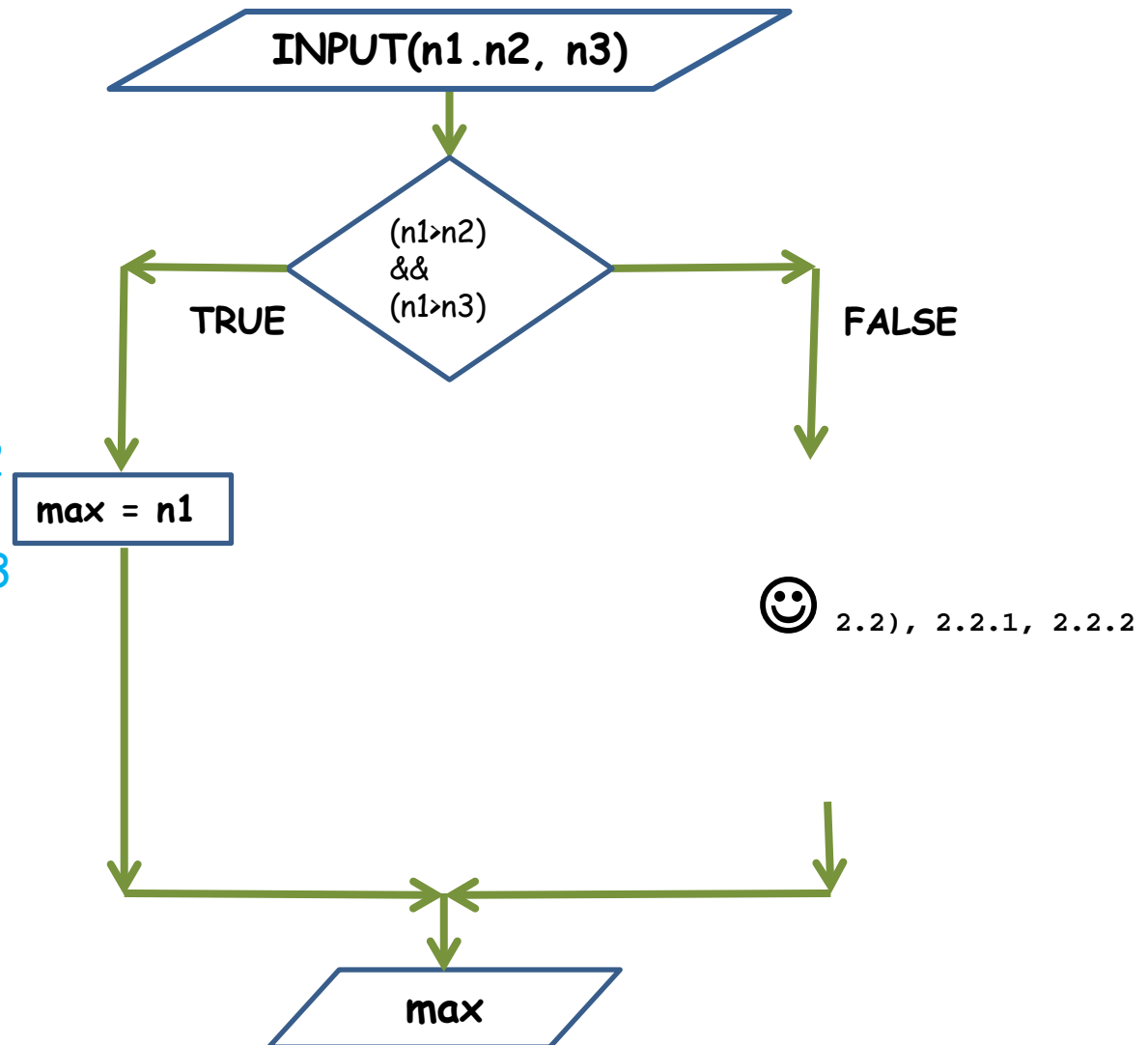
1) INPUT (n_1, n_2, n_3)

2) SE $n_1 > n_2 \ \&\& \ n_1 > n_3$
2.1) max = n_1

ALTRIMENTI

2.2) SE $n_2 > n_3$
2.2.1) max = n_2
ALTRIMENTI
2.2.2) max = n_3

3) OUTPUT(max)



Massimo tra tre numeri (alg. 4)

0) i dati: n_1, n_2, n_3 (input), e max

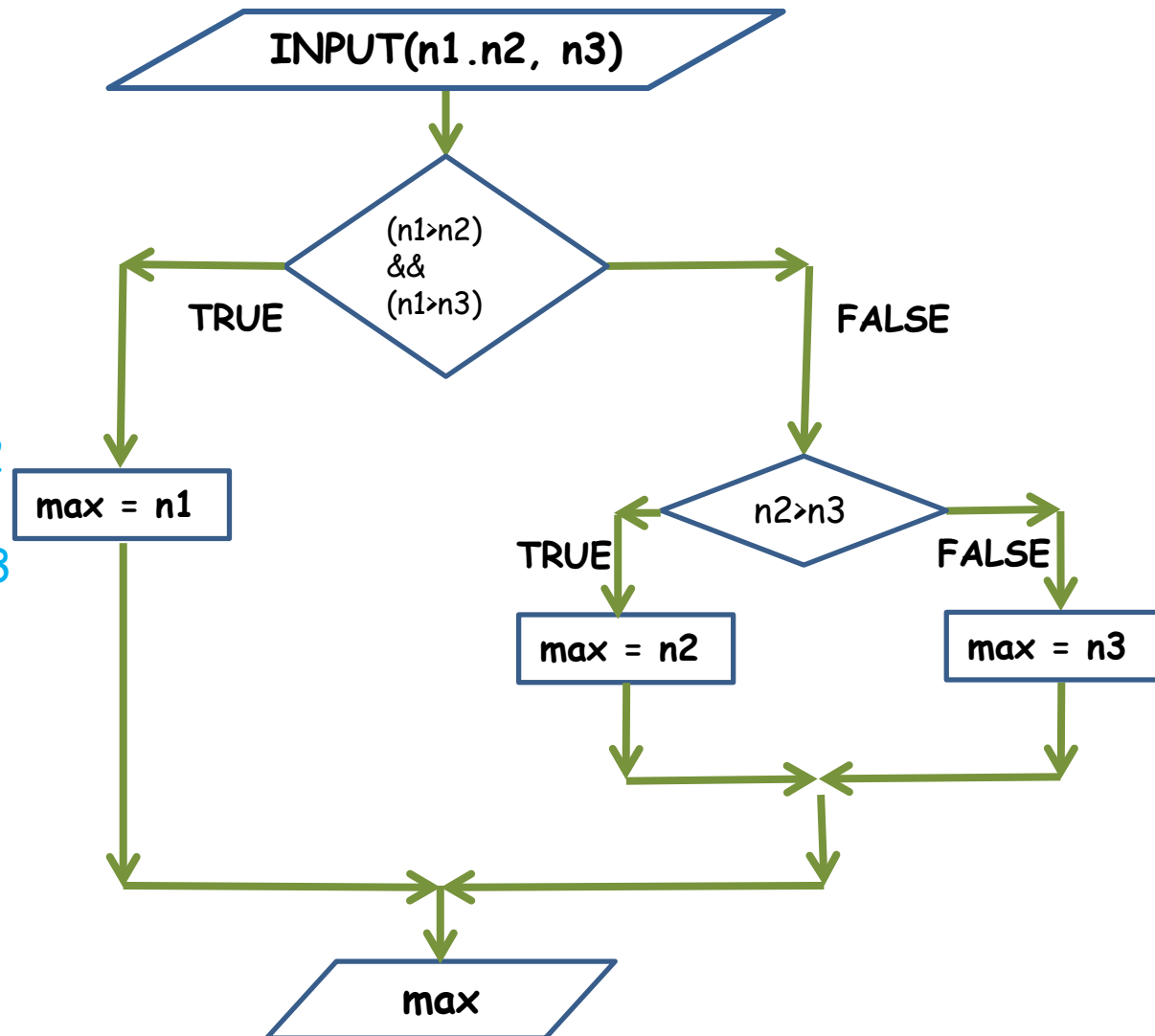
1) INPUT (n_1, n_2, n_3)

2) SE $n_1 > n_2 \ \&\& \ n_1 > n_3$
2.1) max = n_1

ALTRIMENTI

2.2) SE $n_2 > n_3$
2.2.1) max = n_2
ALTRIMENTI
2.2.2) max = n_3

3) OUTPUT(max)



Massimo tra tre numeri (alg. 4)

```
int main () {
    int n1, n2, n3,          /* i tre numeri */
        max;                /* per rappresentare il massimo ...

    printf ("Caro/a utente, dammi tre numeri: ");
    scanf ("%d %d %d", &n1, &n2, &n3);          /* lettura */

    if

    else                      😊

    printf ("beh, il massimo %c %d\n", 138, max);

    printf ("\nFINE programma\n");
    return 0;
}
```

0) i dati: n1, n2, n3 (input), e max
1) INPUT (n1, n2, n3)
2) SE $n1 > n2 \ \&\& \ n1 > n3$
 2.1) max = n1
 ALTRIMENTI
 2.2) SE $n2 > n3$
 2.2.1) max = n2
 ALTRIMENTI
 2.2.2) max = n3
3) OUTPUT(max)

Massimo tra tre numeri (alg. 4)

```
int main () {
    int n1, n2, n3,          /* i tre numeri */
        max;                /* per rappresentare il massimo ...

    printf ("Caro/a utente, dammi tre numeri: ");
    scanf ("%d %d %d", &n1, &n2, &n3);          /* lettura */

    if ((n1>n2) && (n1>n3))
        max = n1;
    else
        if (n2 > n3)
            max = n2;
        else max = n3;

    printf ("beh, il massimo è %d\n", max);

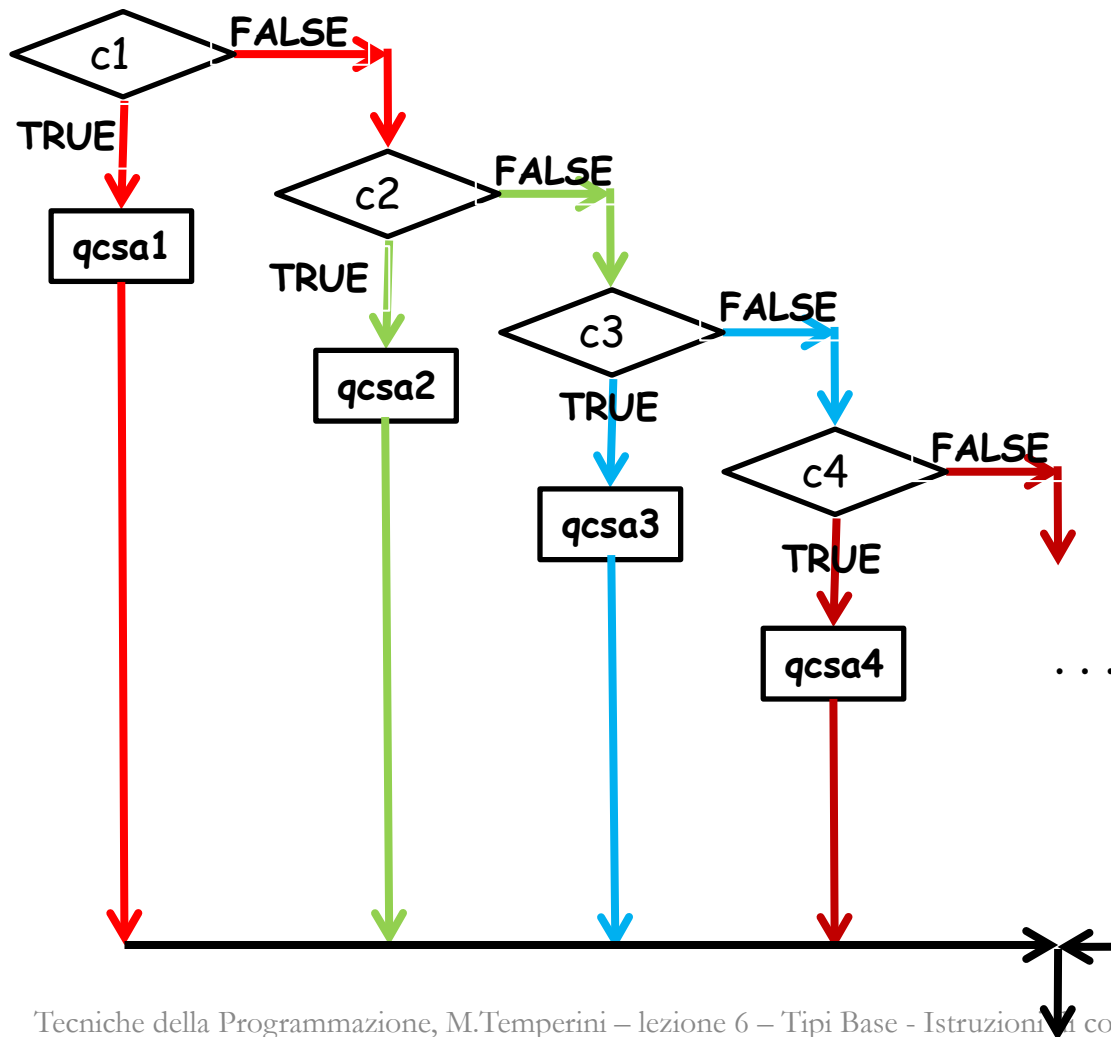
    printf ("\nFINE programma\n");
    return 0;
}
```

0) i dati: n1, n2, n3 (input), e max
1) INPUT (n1, n2, n3)
2) SE $n1 > n2$ && $n1 > n3$
 2.1) max = n1
 ALTRIMENTI
 2.2) SE $n2 > n3$
 2.2.1) max = n2
 ALTRIMENTI
 2.2.2) max = n3
3) OUTPUT(max)

Cascata di if

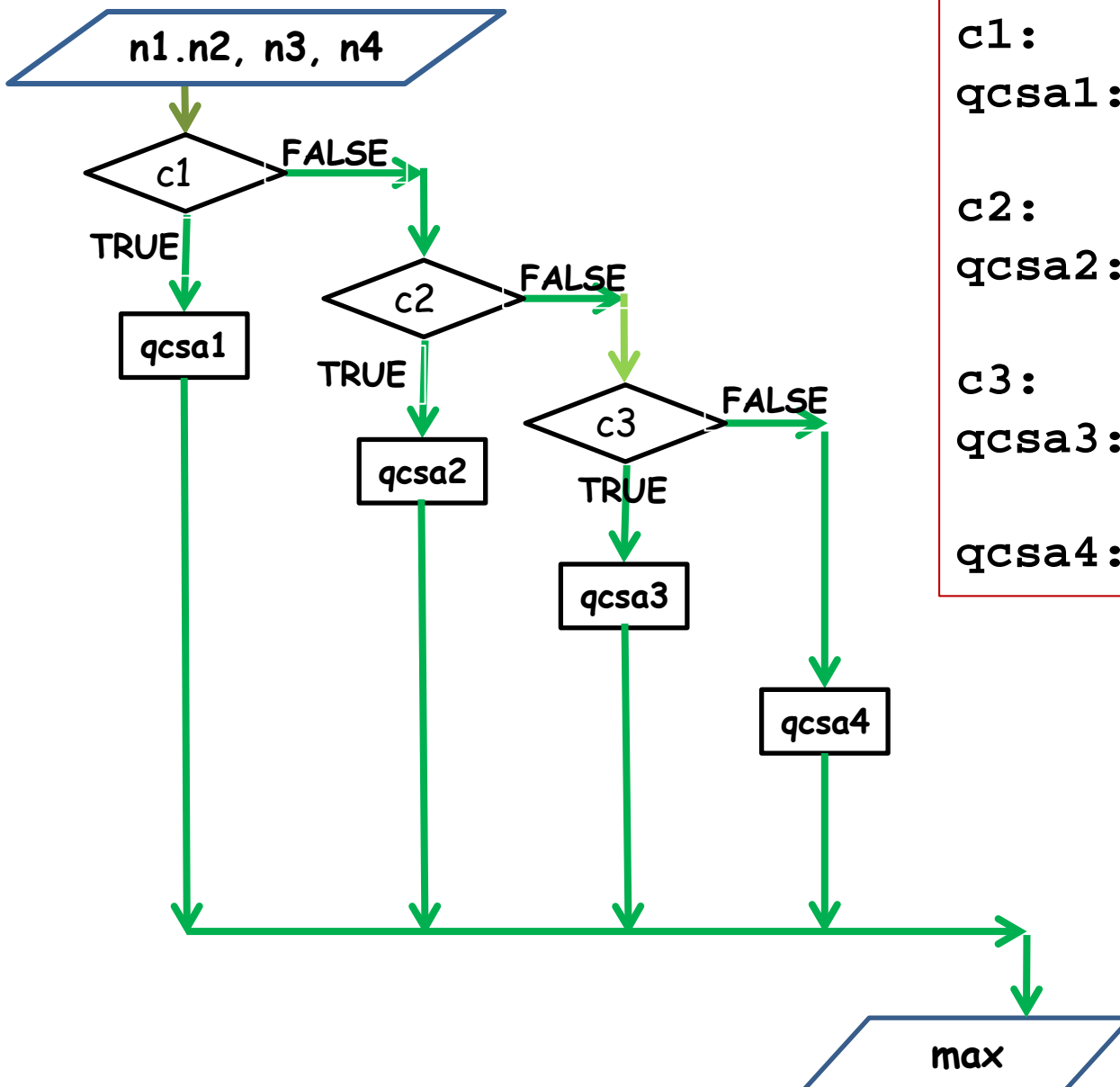
(annidati negli else ...)

l'esempio precedente mostrava un if annidato nella parte else. A volte la vita rende necessaria una CASCATA DI IF ... **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, ...



```
if (c1)
    qcsa1;
else
    if (c2)
        qcsa2;
    else
        if (c3)
            qcsa3;
        else
            if (c4)
                qcsa4;
            else
                ...
    ...
```

Massimo tra QUATTRO numeri (alg. 4 e cascata)



c1: ???

qcsa1: ☺

c2: ???

qcsa2: ☺

c3: ???

qcsa3: ☺

qcsa4: max=n4

suggerimento segue

Massimo tra QUATTRO numeri (alg. 4 e cascata)

PRIMA, scrivere l'algorithm4 nel caso di 4 numeri

nel caso 3 numeri era

0) i dati: $n1, n2, n3$ (input), e max

1) INPUT ($n1, n2, n3$)

2) SE $n1 > n2 \ \&\& \ n1 > n3$

2.1) max = $n1$

ALTRIMENTI

2.2) SE $n2 > n3$

2.2.1) max = $n2$

ALTRIMENTI

2.2.2) max = $n3$

3) OUTPUT(max)

c1: ???

qcsa1: ☺

c2: ???

qcsa2: ☺



0) i dati: $n1, n2, n3, n4$ (input), e max

...

2) condizione per assegnare $n1$ a max? Se $n1$ e' piu' grande di tutti ...

2.1) max = $n1$

ALTRIMENTI

...

Massimo tra QUATTRO numeri (alg. 4 e cascata)

PRIMA, scrivere l' algoritmo4 nel caso di 4 numeri

nel caso 3 numeri era

```
0) i dati: n1, n2, n3 (input), e max
1)  INPUT (n1, n2, n3)
2)  SE n1 > n2 && n1 > n3
    2.1) max = n1
    ALTRIMENTI
    2.2) SE n2 > n3
        2.2.1) max = n2
        ALTRIMENTI
        2.2.2) max = n3
3)  OUTPUT(max)
```

c1: ???

qcsa1: ☺

c2: ???

```
0) i dati: n1, n2, n3, n4 (input), e max
1)  INPUT (n1, n2, n3, n4)
```

```
2)  SE n1 > n2 && n1 > n3 && n1 > n4
    2.1) max = n1
    ALTRIMENTI
```

2.2) il massimo non è n1 ... quindi siamo a dover cercare il massimo tra n2, n3, n4, cioè tra tre numeri come abbiamo fatto prima ... SE ... che cosa?

```
2.2.1) max = n2
```

```
ALTRIMENTI
```

```
2.2.2) Il massimo non è n2 ...
quindi ... sarà n3 o n4
```

```
SE ☺
```

```
2.2.2.1) ...
```

```
ALTRIMENTI
```

```
2.2.2.2) ...
```

```
3)  OUTPUT(max)
```

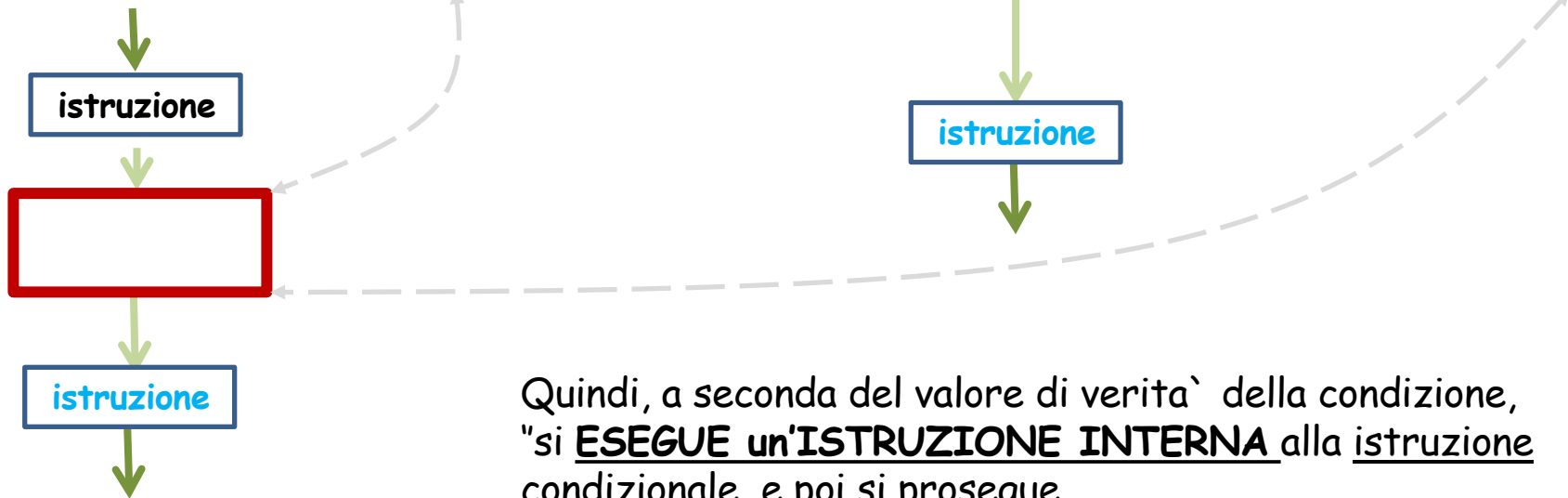
vedi Approfondimenti, per completare questo esercizio sulla cascata di if E per capire la potenziale ambiguità dell'istruzione if, quando nella parte positiva di un if ne annidiamo un altro (Ambiguif)

**And now,
istruzione ripetitiva**

Istruzione condizionale ... e istruzione ripetitiva

L'istruzione condizionale ...
l'abbiamo già vista:

- Si valuta la condizione (un'espressione logica)
- Si esegue l'istruzione (strutturata) nel ramo corrispondente al valore di verità dell'espressione
- E poi si prosegue con la successiva istruzione (strutturata)

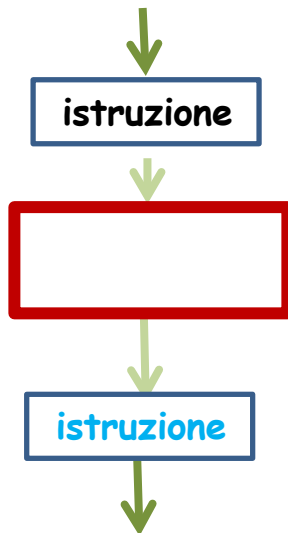
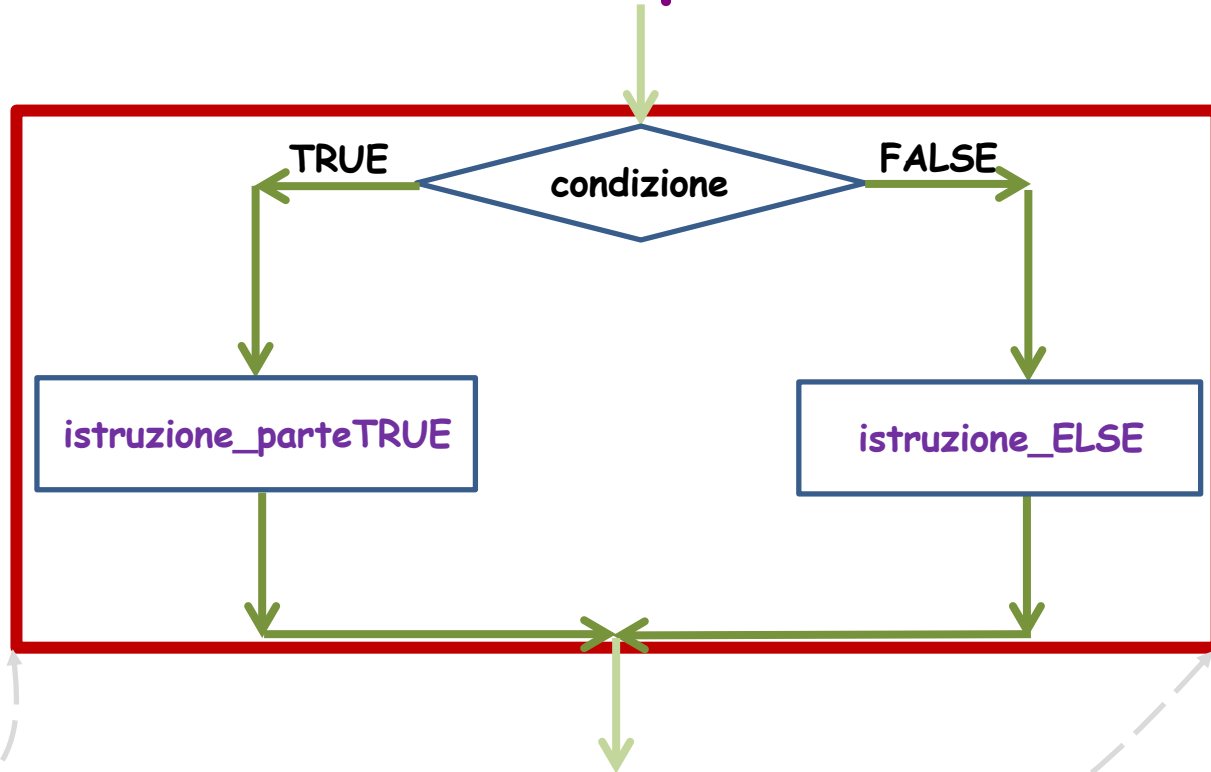


Quindi, a seconda del valore di verità della condizione, "si ESEGUE un'ISTRUZIONE INTERNA alla istruzione condizionale, e poi si prosegue.

Istruzione condizionale ... e istruzione ripetitiva

L'espressione condizionale ...
l'abbiamo già vista:

- Si valuta la condizione (un'espressione logica)
- Si esegue l'istruzione (strutturata) nel ramo corrispondente al valore di verità dell'espressione
- E poi si prosegue con la successiva istruzione (strutturata)



Quindi "si ESEGUE un'ISTRUZIONE INTERNA alla istruzione condizionale, e poi si prosegue.

Invece, nell'istruzione ripetitiva,

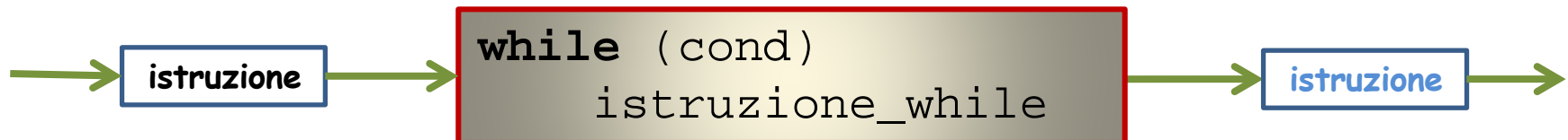
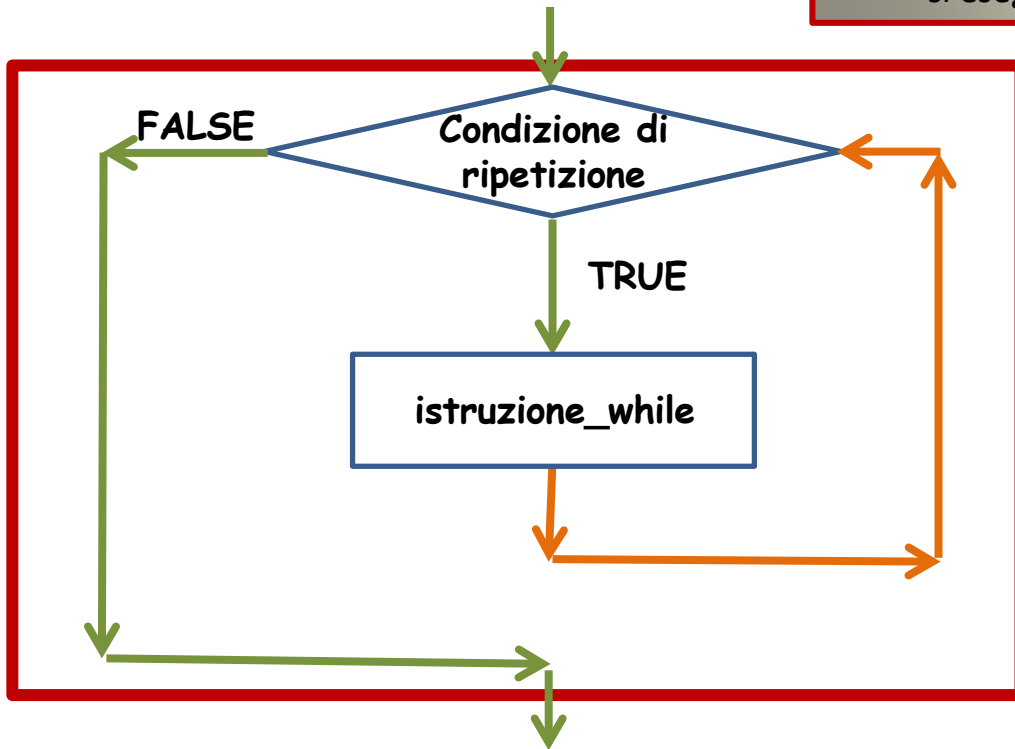
- c'è una sola istruzione interna (strutturata)
- e prima di proseguire con l'istruzione successiva, si può **'RIPETERE L'ISTRUZIONE INTERNA'** tante volte quante serve

Istruzione ripetitiva (generica)

Quindi anche qui si valuta una condizione (espressione logica);
il valore di verità della condizione decide se l'istruzione_while

- va eseguita (*iterata*)
- oppure no (e in tal caso si prosegue con la prossima **istruzione**).

MENTRE vale la `condizione_di_ripetizione`
si esegue `istruzione_while`

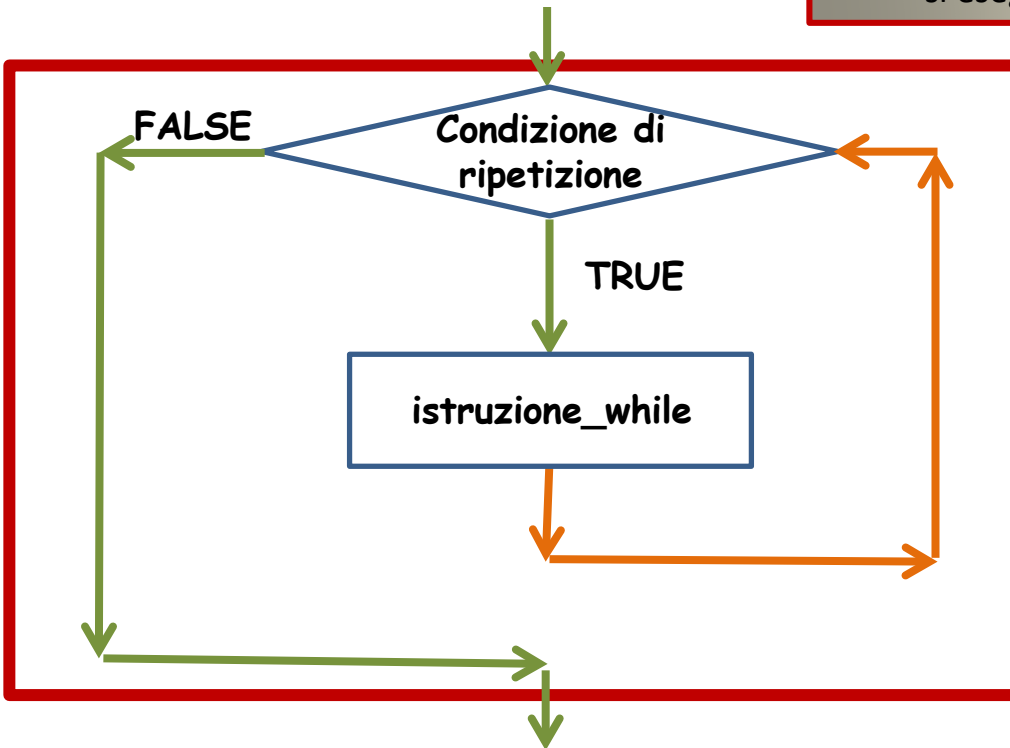


Istruzione ripetitiva (generica)

Quindi anche qui si valuta una condizione (espressione logica);
il valore di verità della condizione decide se l'istruzione_while

- va eseguita (*iterata*)
- oppure no (e in tal caso si prosegue con la prossima **istruzione**).

MENTRE vale la `condizione_di_ripetizione`
si esegue `istruzione_while`

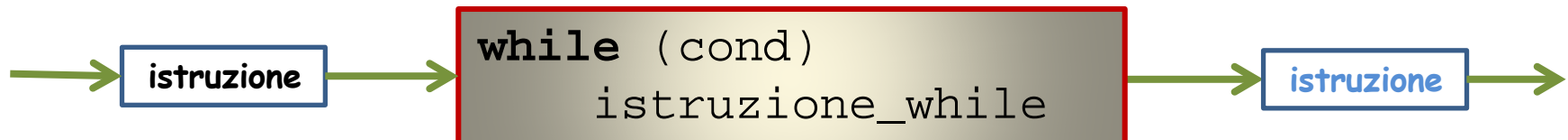


Durante l'esecuzione dell'istruzione_while, i fattori che concorrono nella valutazione dell'espressione

`Condizione_di_ripetizione`

possono cambiare, perciò

- dopo ogni iterazione si torna a valutare la condizione, per decidere se fare un'altra iterazione o no
- se la condizione non è verificata, "il ciclo si interrompe e si prosegue con la prossima **istruzione** strutturata.



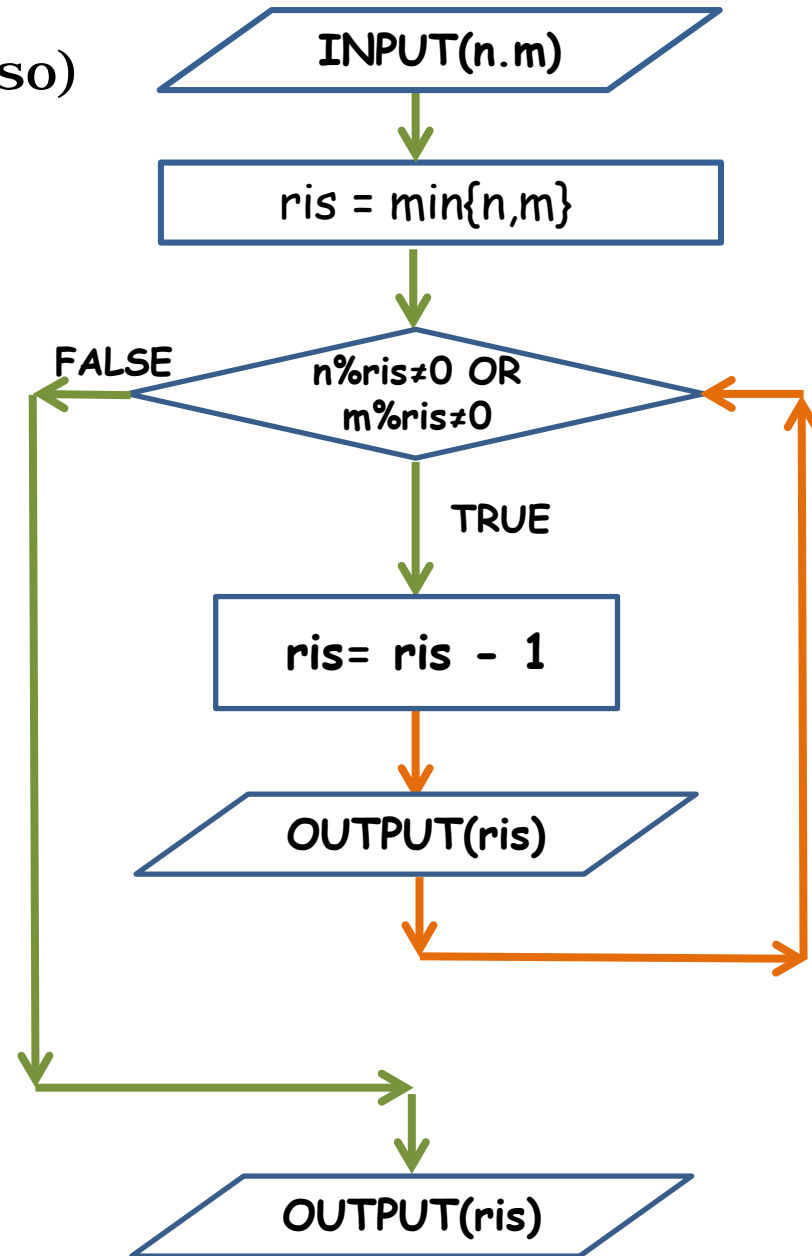
Istruzione ripetitiva: while

Usiamo ancora MCD (ma un pò di verso)

Body

Numero di esecuzioni

Terminazione del ciclo

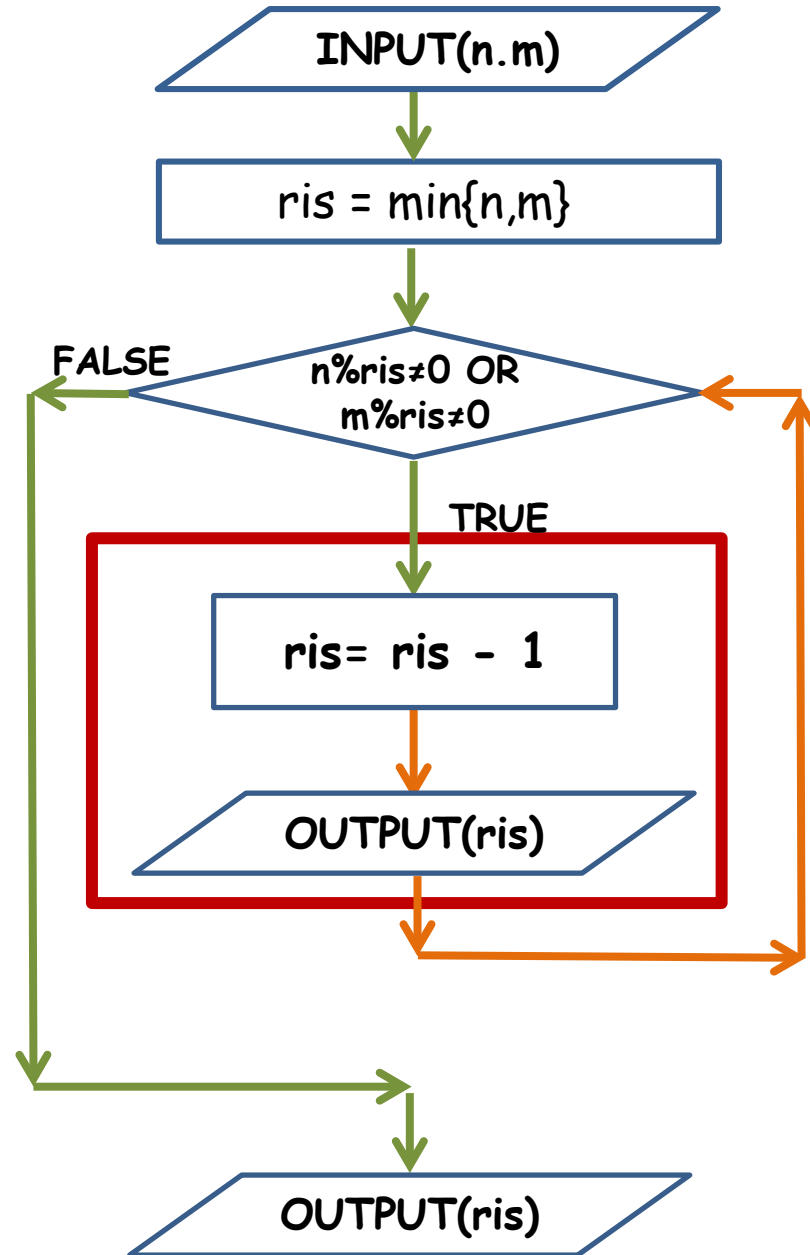


Istruzione ripetitiva: body

Body del ciclo

Numero di esecuzioni

Terminazione del ciclo



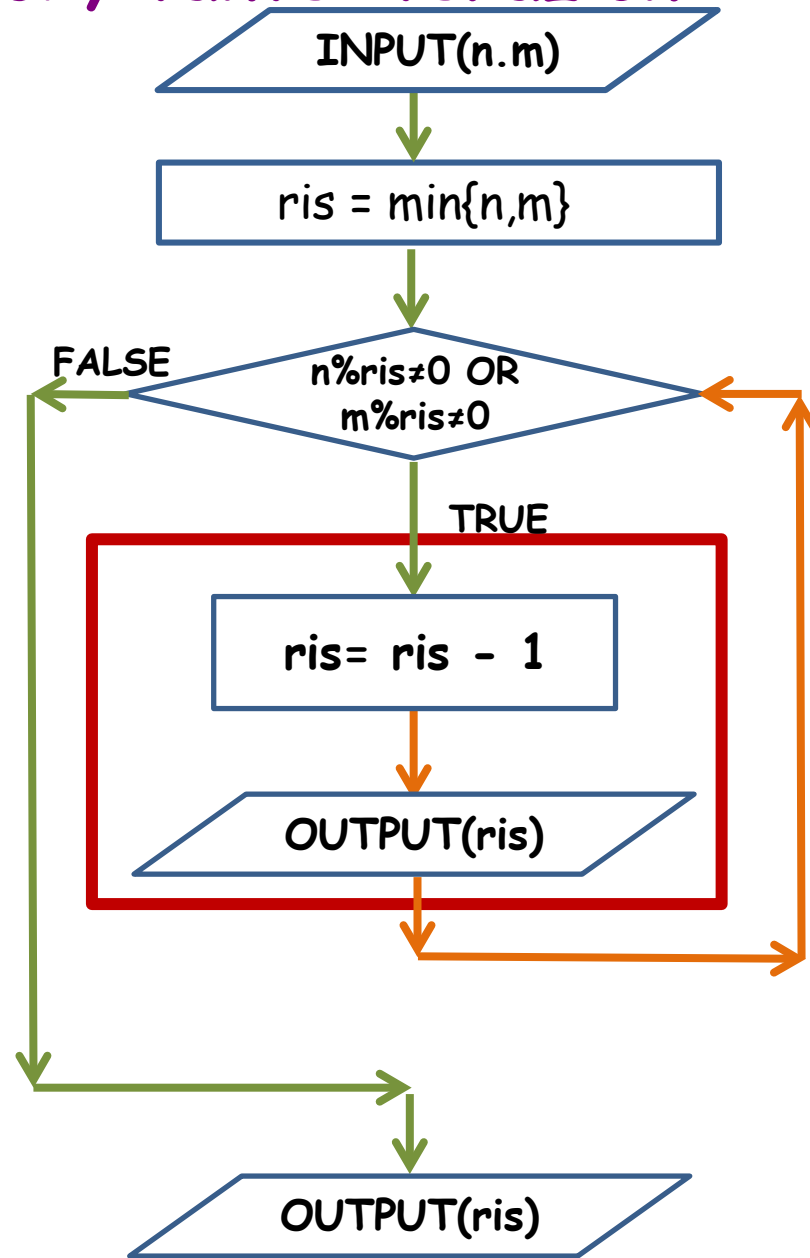
Istruzione ripetitiva: tanti cicli, tante iterazioni

Numero di esecuzioni:

In linea di principio, un'istruzione while può vedere eseguito il body qualsiasi numero di volte

In questo caso particolare,

- 0 ?
- 1 ?
- 2 ?
- infinite ?



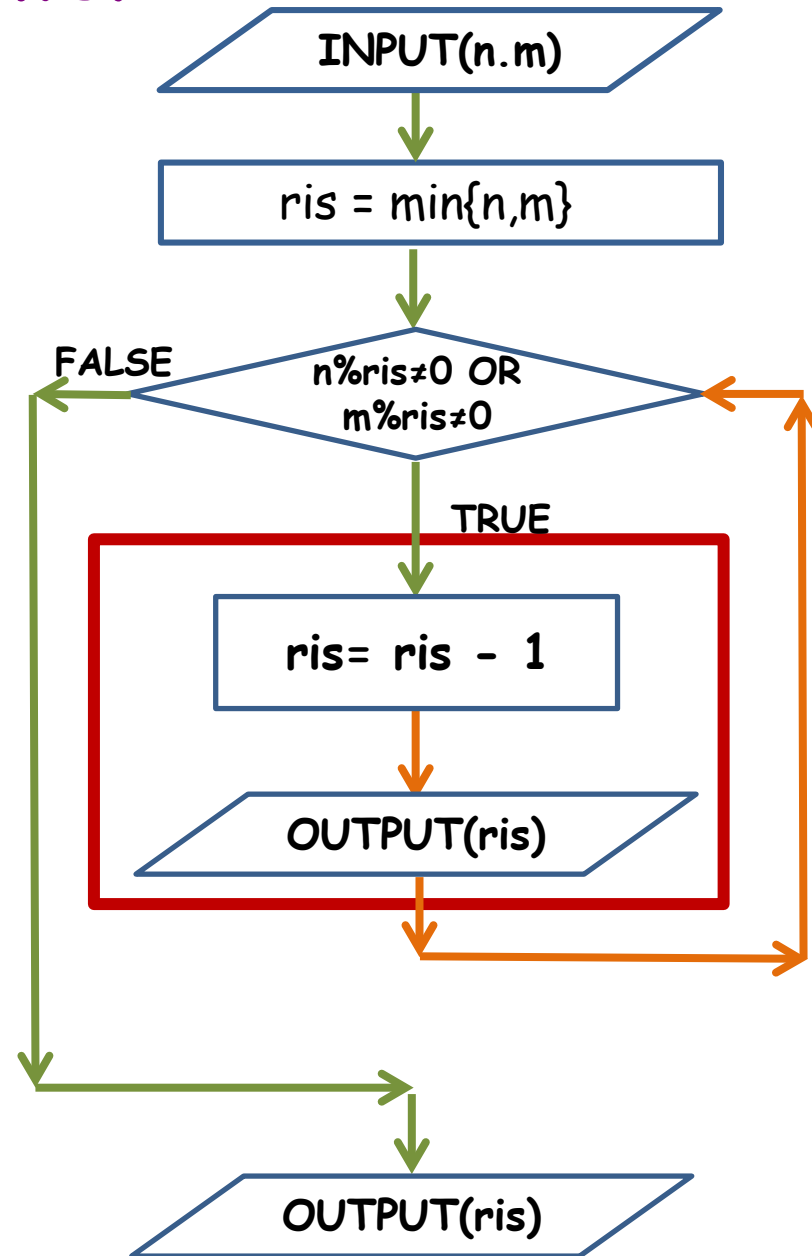
Istruzione ripetitiva: ma quante?

Numero di esecuzioni:

In linea di principio, un'istruzione while può vedere eseguito il body qualsiasi numero di volte

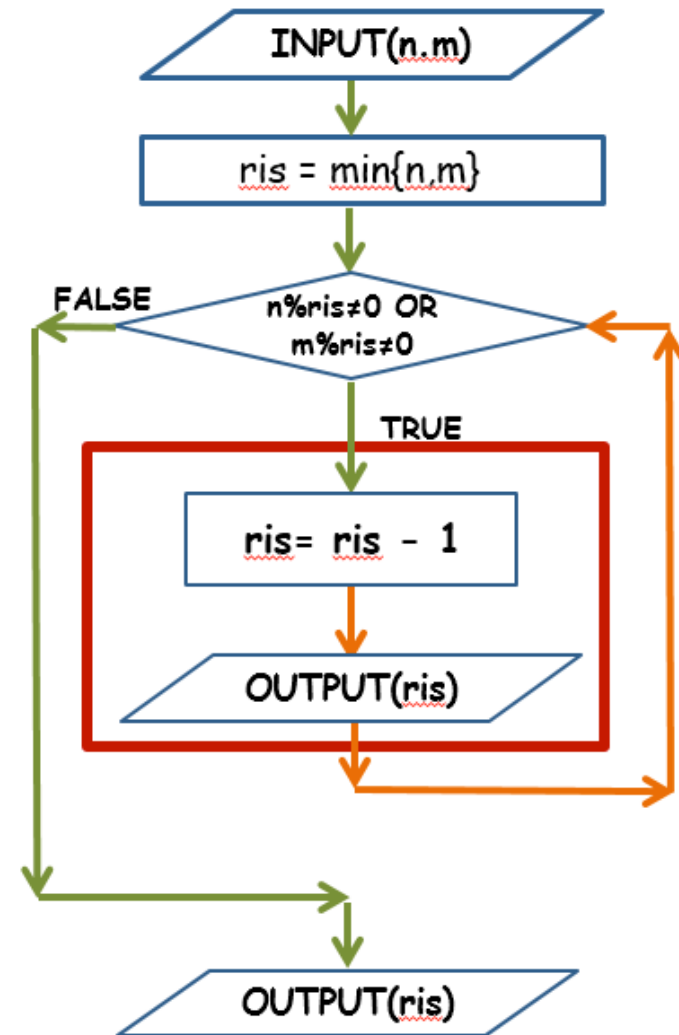
In questo caso particolare,

- 0 possibile
- 1 possibile
- 2 possibile
- infinite ?



Istruzione ripetitiva: intanto programmiamo

```
/* programma che riceve due interi e
 stampa il loro MCD */
#include <stdio.h>
int main () {
```



Istruzione ripetitiva: program MCD

```
/* programma che riceve due interi e stampa il loro MCD */
```

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

```
int main () {
```

```
    int n, m,                /* i due interi */  
        ris;                /* usata per .....*/
```

```
    printf ("Caro/a utente: ... separati da  
            uno ...: ");
```

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

```
/* calcolo del minimo tra n ed m,  
   che sarà l'inizializzazione di ris */
```

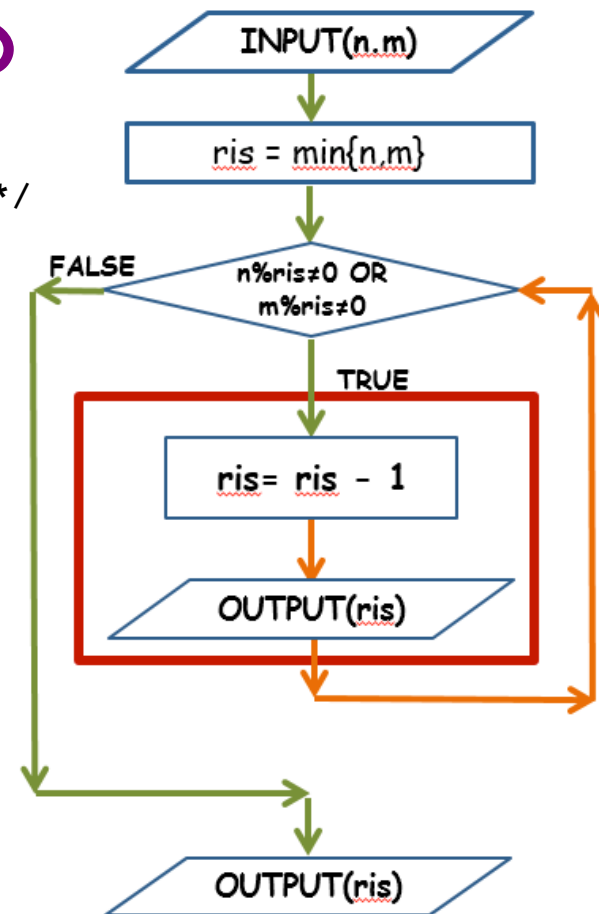
```
if (n<m)
```

```
    ris = n;
```

```
else ris = m;
```

```
/* mentre ris non divide n o non divide m, non può essere  
   l'MCD, quindi si passa a verificare il suo valore  
   decrementato */
```

```
while ( ..... ) {
```



Istruzione ripetitiva: program MCD

```
/* programma che riceve due interi e stampa il loro MCD */
```

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

```
int main () {  
    int n, m, ris;
```

```
    printf ("Car/a utente: ... separati da uno spazio: ");
```

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

```
    if (n<m) ris = n;
```

```
    else     ris = m;
```

```
    while ( (n%ris != 0) || (m%ris !=0) ) {  
        ris = ris - 1;  
        printf("ris attuale: %d\n", ris);  
    }
```

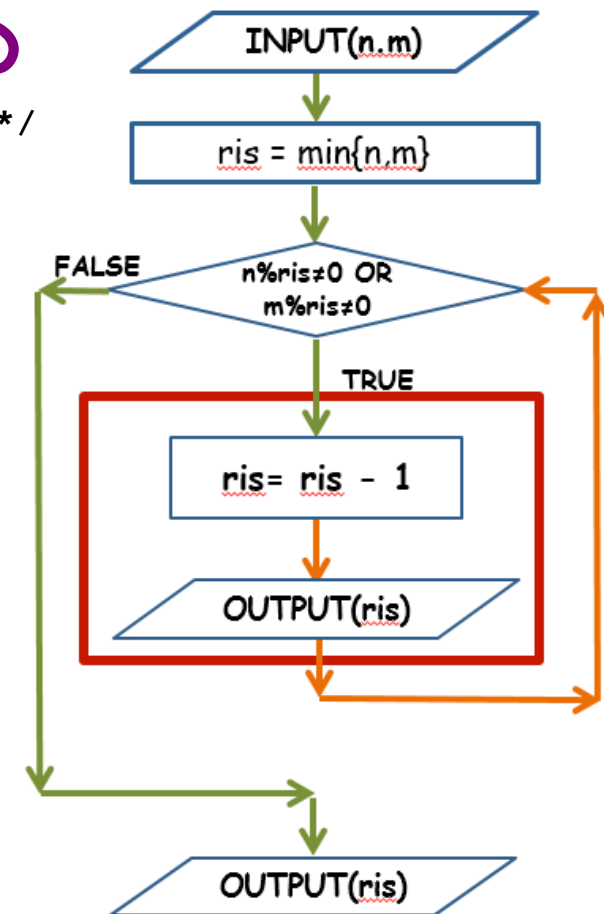
```
    /* ora siamo usciti dal ciclo,  
       quindi ris divide sia n che m  
       (al meno, è 1) */
```

```
    printf ("il MCD tra %d e %d è %d\n", n, m, ris);
```

```
    printf("FINE\n");
```

```
    return 0;
```

```
}
```



uhm, veramente ho qualche dubbio ... ma vabbè, andiamo avanti per ora

Istruzione ripetitiva: numero di esecuzioni 1/2

```
if (n<m) ris = n;  
    else    ris = m;
```

```
while ( (n%ris != 0) || (m%ris !=0) ) {  
    ris = ris - 1;  
    printf("ris attuale: %d\n", ris);  
}
```

Numero di esecuzioni:

In linea di principio, un'istruzione while può vedere eseguito il body qualsiasi numero di volte

In questo caso particolare,

- 0 possibile
- 1 possibile
- 2 possibile
- infinite quindi? E` possibile o no?

Istruzione ripetitiva: numero di esecuzioni 2/2

```
if (n<m) ris = n;  
    else    ris = m;
```

```
while ( (n%ris != 0) || (m%ris !=0) ) {  
    ris = ris - 1;  
    printf("ris attuale: %d\n", ris);  
}
```

Numero di esecuzioni:

In linea di principio, un'istruzione while può vedere eseguito il body qualsiasi numero di volte

In questo caso particolare,

- 0 possibile
- 1 possibile
- 2 possibile
- infinite possibile con input negativi es n=5, m=-100)
non possibile con input positivi

QUINDI se la specifica del problema viene (giustamente) cambiata in
"... riceve due numeri interi **POSITIVI**" il programma va bene

Istruzione ripetitiva: numero di esecuzioni in un caso diverso

In questo caso?

```
n=0;
while (12) {
    printf( 'n = %d ', n);
    printf( '... ciao!\n' );
}
```

Numero di esecuzioni:

In linea di principio, un'istruzione **while** può vedere eseguito il body qualsiasi numero di volte

In questo caso particolare,

- infinite yes

Questo è un caso in cui la condizione di ripetizione è sempre vera e quindi l'**istruzione_while** viene ripetuta per sempre cioè non si esce mai dall'istruzione ripetitiva

di solito, questa "mancanza di terminazione" non va bene!

Istruzione ripetitiva: Terminazione del ciclo

Body del ciclo

Numero di esecuzioni

Terminazione del ciclo

Deve essere assicurata!!
è un aspetto importante dell'algoritmo implementato dal programma!!

Nel body ci deve essere qualche istruzione che assicuri la modifica della condizione di ripetizione, in direzione della terminazione

```
n=0;
while (12) {
    printf('\n = %d ', n);
    printf('\n... ciao!\n');
}
```

Qui non c'è codice che possa cambiare la condizione di ripetizione, che quindi è sempre verificata e comporta sempre la iterazione del body

Istruzione ripetitiva: Terminazione del ciclo

Terminazione del ciclo:

Deve essere assicurata!!

è un aspetto importante dell'algoritmo realizzato dal programma!!

Nel body ci deve essere qualche istruzione che assicuri la modifica della condizione di ripetizione, **verso la terminazione**

```
n=0;
while (12) {
    printf('\n = %d ', n);
    printf('\n... ciao!\n');
}
```

Qui non c'è codice che possa cambiare la condizione di ripetizione

```
n=0;
while (n<12) {
    printf('\n = %d ', n);
    printf('\n... ciao!\n');
    n -= 1
}
```

Qui c'è codice che modifica la `variabile di test`,
Ma ... non c'è terminazione ...

Istruzione ripetitiva: while

Terminazione del ciclo:

Deve essere assicurata!!

è un aspetto importante dell'algoritmo realizzato dal programma!!

Nel body ci deve essere qualche istruzione che assicuri la modifica della condizione di ripetizione, **verso la terminazione**

```
while ( (n%ris != 0) || (m%ris !=0) ) {  
    ris = ris - 1;  
    printf("ris attuale: %d\n", ris);  
}
```

Qui (ALMENO nel caso di numeri positivi ...) il codice compreso nel body modifica la variabile di test e fa "convergere" il ciclo verso la terminazione

```
n=0;  
while (12) {  
    printf('\n = %d ', n);  
    printf('\n... ciao!\n');  
}
```

Qui non c'è codice che possa cambiare la condizione di ripetizione

```
n=0;  
while (n<12) {  
    printf('\n = %d ', n);  
    printf('\n... ciao!\n');  
    n += 1  
}
```

Qui c'è codice che modifica la 'variabile di test', e, a differenza del caso nella slide precedente, c'è terminazione ...

Anticipazione prossima lezione

Istruzione ripetitiva: while (1/5)

Cosa fa?

```
#include <stdio.h>

int main () {
    int n=4;

    while (n>0) {
        printf ("stampiamo n ... %d\n", n);
        n = n-5;
        printf ("stampiamo n ... %d\n", n);
        n = n+2;
    }
    printf ("\nFINE programma\n");
    return 0;
}
```



cosa stampa?

Istruzione iterativa: while (2/5)

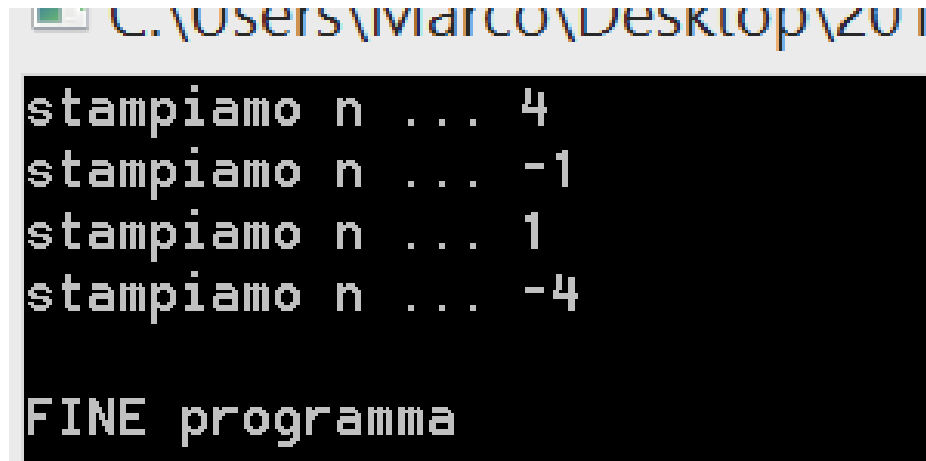
Cosa fa?

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

```
int main () {  
    int n=4;
```

```
    while (n>0) {  
        printf ("stampiamo n ... %d\n", n);  
        n = n-5;  
        printf ("stampiamo n ... %d\n", n);  
        n = n+2;  
    }
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```



```
C:\Users\marco\Desktop\zou  
stampiamo n ... 4  
stampiamo n ... -1  
stampiamo n ... 1  
stampiamo n ... -4  
  
FINE programma
```

Algoritmo?

Fai il Diagramma di Flusso, e l'algoritmo per passi

Istruzione ripetitiva: while (3/5)

diagramma a blocchi corrispondente

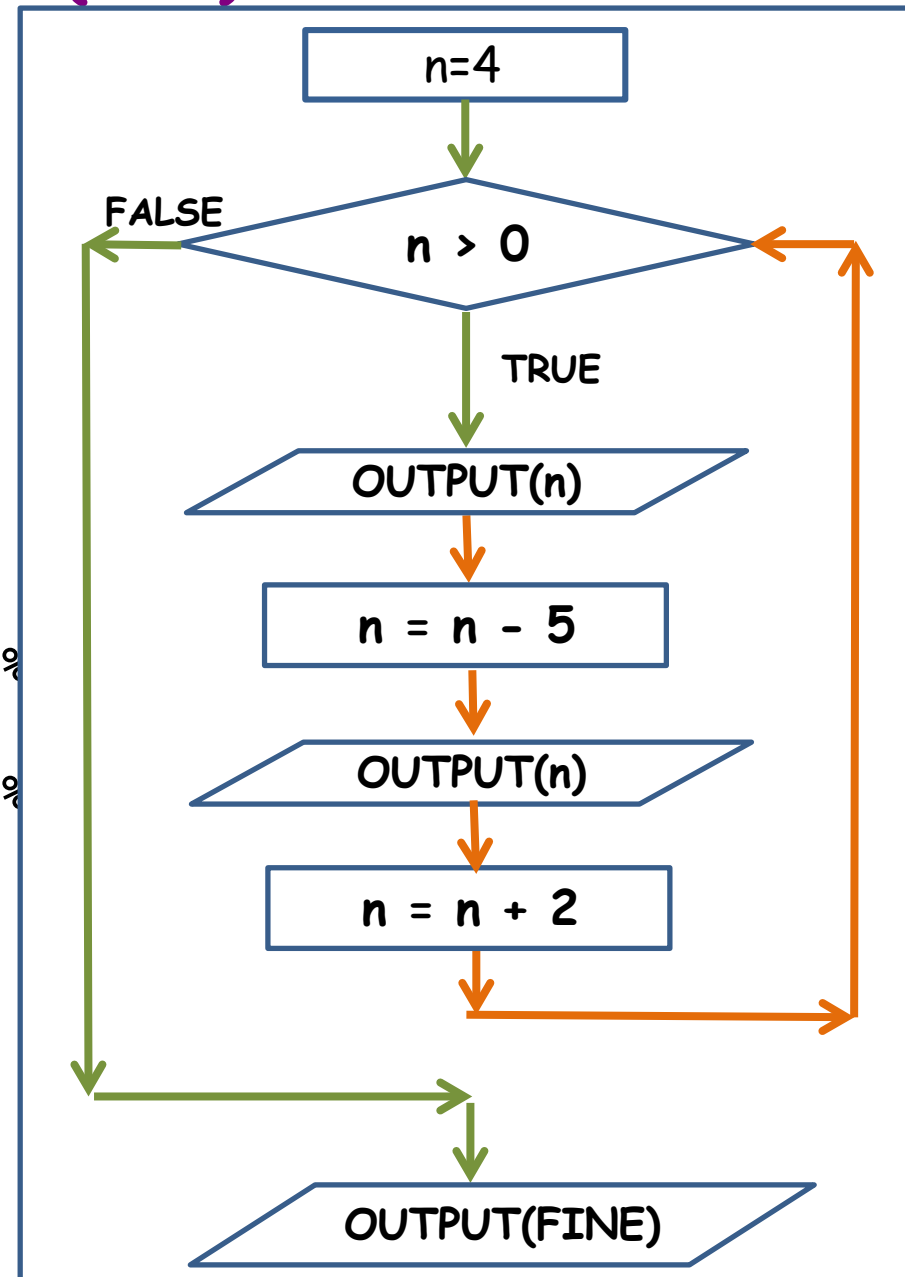
```
#include <stdio.h>
```

```
int main () {  
    int n;
```

```
    n=4
```

```
    while (n>0) {  
        printf ("stampiamo n ... %d\n", n);  
        n = n-5;  
        printf ("stampiamo n ... %d\n", n);  
        n = n+2;  
    }
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```



Istruzione iterativa: while (4/5)

Algoritmo corrispondente

```
#include <stdio.h>

int main () {
    int n;

    n=4

    while (n>0) {
        printf ("stampiamo n ... %d\n", n);
        n = n-5;
        printf ("stampiamo n ... %d\n", n);
        n = n+2;
    }
    printf ("\nFINE programma\n");
    return 0;
}
```

Algoritmo in pseudocodice

- 0) i dati: n, ma nemmeno letto da input
- 1) Inizializzazione n=4
- 2) MENTRE n>0
 - 2.1) stampa n
 - 2.2) cambia n: n=n-5
 - 2.3) stampa n
 - 2.4) cambia n: n+=2
- 3) Vabbè, abbiamo finito
(stampa FINE PROGRAMMA e poi end)

Istruzione iterativa: while (5/5)

esecuzione simulata



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

```
int main () {  
    int n;
```

```
    n=4
```

```
    while (n>0) {  
        printf ("stampiamo n ... %d\n", n);  
        n = n-5;  
        printf ("stampiamo n ... %d\n", n);  
        n = n+2;
```

```
    }
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```

init

4

n

n>0: iterazione 1

Algoritmo in pseudocodice

0) i dati: n, ma nemmeno letto da input

1) Inizializzazione n=4

2) MENTRE n>0

2.1) stampa n

2.2) cambia n: n=n-5

2.3) stampa n

2.4) cambia n: n+=2

3) Vabbè, abbiamo finito
(stampa FINE PROGRAMMA e poi
end)

Istruzione iterativa: while (5/5)



esecuzione simulata

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

```
int main () {  
    int n;  
  
    n=4  
  
    while (n>0) {  
        printf ("stampiamo n ... %d\n", n);  
        n = n-5;  
        printf ("stampiamo n ... %d\n", n);  
        n = n+2;  
    }  
    printf ("\nFINE programma\n");  
    return 0;  
}
```

Algoritmo in pseudocodice

- 0) i dati: n, ma nemmeno letto da input
- 1) Inizializzazione n=4
- 2) MENTRE n>0
 - 2.1) stampa n
 - 2.2) cambia n: n=n-5
 - 2.3) stampa n
 - 2.4) cambia n: n+=2
- 3) Vabbè, abbiamo finito
(stampa FINE PROGRAMMA e poi end)

init

4

n

n>0: iterazione 1

..... 4

Istruzione iterativa: while (5/5)



esecuzione simulata

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

```
int main () {  
    int n;  
  
    n=4  
  
    while (n>0) {  
        printf ("stampiamo n ... %d\n", n);  
        n = n-5;  
        printf ("stampiamo n ... %d\n", n);  
        n = n+2;  
    }  
    printf ("\nFINE programma\n");  
    return 0;  
}
```

Algoritmo in pseudocodice

- 0) i dati: n, ma nemmeno letto da input
- 1) Inizializzazione n=4
- 2) MENTRE n>0
 - 2.1) stampa n
 - 2.2) cambia n: n=n-5
 - 2.3) stampa n
 - 2.4) cambia n: n+=2
- 3) Vabbè, abbiamo finito (stampa FINE PROGRAMMA e poi end)

init

n>0: iterazione 1

~~4 -1~~

n

..... 4

Istruzione ripetitiva: while

Ciclo solo parzialmente visibile:

supponendo che termini, cosa stampa la `printf()`?

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

```
int main () {  
    int n=21;
```

```
    while (n!=7) {  
        n=n*12;  
        printf ("n ... %d\n", n);  
        n =
```



```
    }  
    printf ("stampiamo n ... %d\n", n);  
    printf ("\nFINE programma\n");  
    return 0;  
}
```

Tecniche della Programmazione, lez. 6

- Approfondimenti

Conversione Implicita

Avviene, in certi casi, quando un'espressione che produce un valore di tipo T1, appare in un contesto in cui è atteso un valore di tipo T2: allora il valore di tipo T1 viene convertito al tipo T2 e come tale viene poi usato.

```
int i=137;
double a;
    a = 12;      /* a ora contiene 12.0 */
    a = i;      /* a ora contiene 137.0 */
    a = i/2;    /* a ora contiene 68.0 */
```

(#include math.h da fare prima)

```
sqrt(i)      /* sqrt si aspetta un float ... e usa 137.0 */
```

in ognuno di questi casi c'è stata una conversione implicita da int a double/float

NB la conversione implicita è uno strumento efficace solo se avviene senza perdita di informazione; cioè quando la conversione è

da "tipo meno capiente" a "tipo più capiente",

ad esempio da char a short, da short ad int,

da int a float ... a double ...

Il percorso **inverso può invece produrre errori** (es. da int a short, o da double a float), che possono essere anche molto significativi.

Massimo tra quattro numeri (alg.2)



Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati**: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

dobbiame eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



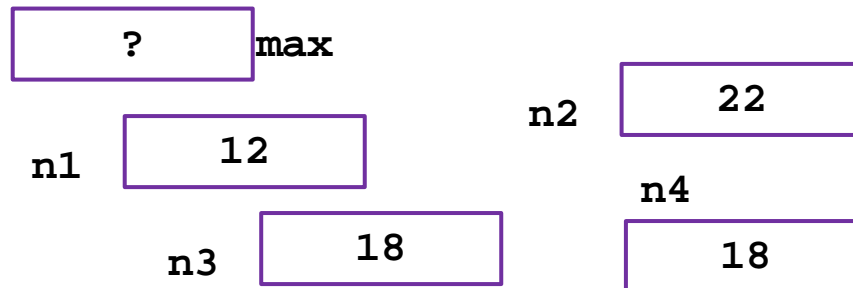
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati**: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18



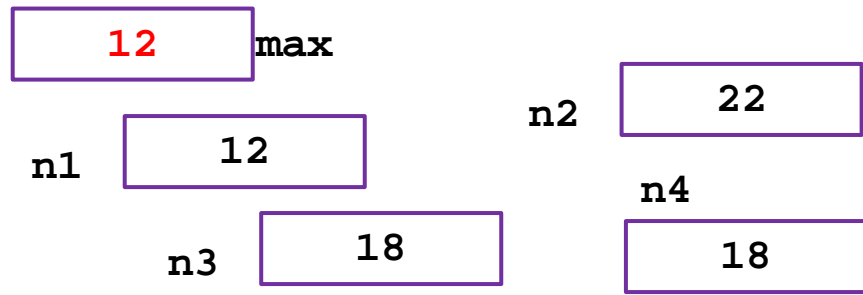
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) i dati: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18

passo 2



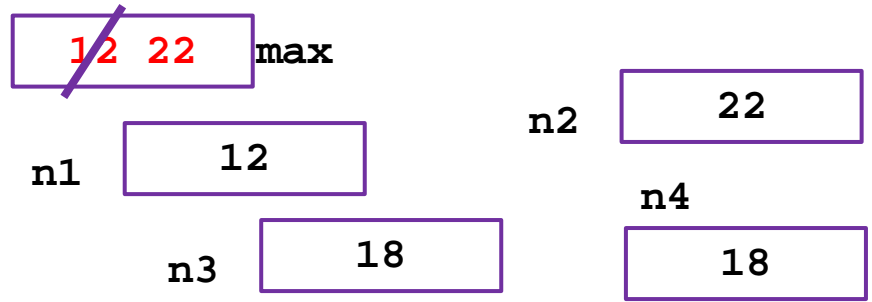
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati:** n1, n2, n3, n4 (input), e max
- 1) INPUT (n1, n2, n3, n4)
- 2) Inizializzazione di max: max = n1
- 3) SE n2 > max
 max = n2
- 4) SE n3 > max ... fai max = n3
- 5) SE n4 > max ... fai max = n4
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18

passo 2

passo 3



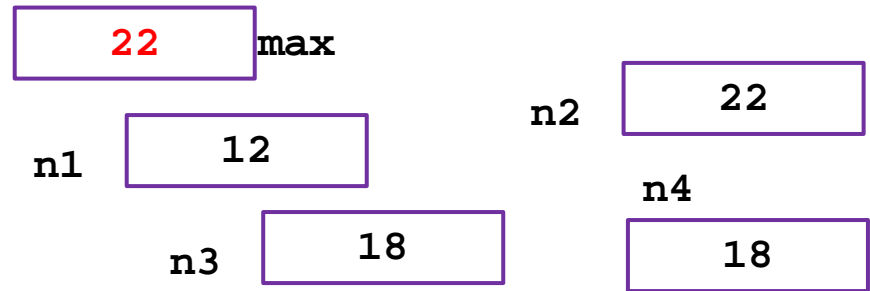
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati**: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18

passo 2

passo 3

passo 4 n3 perde ... max rimane 22



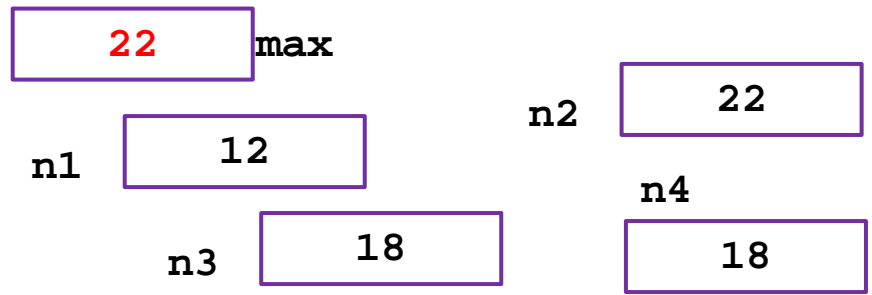
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati:** n1, n2, n3, n4 (input), e max
- 1) INPUT (n1, n2, n3, n4)
- 2) Inizializzazione di max: max = n1
- 3) SE n2 > max
 max = n2
- 4) SE n3 > max ... fai max = n3
- 5) SE n4 > max ... fai max = n4
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18

passo 2

passo 3

passo 4 n3 perde ... max rimane 22

passo 5 n4 perde ... max rimane 22



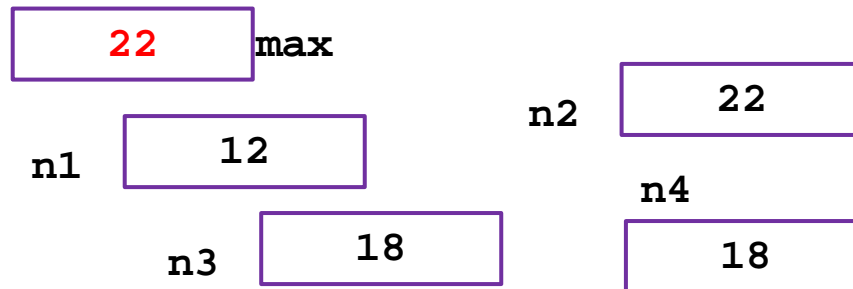
Massimo tra quattro numeri (alg.2)

Tecnica del **MASSIMO PARZIALE**, per trovare il max tra i numeri di una sequenza (in questo caso di 4 numeri):

- 0) **i dati**: n_1, n_2, n_3, n_4 (input), e max
- 1) INPUT (n_1, n_2, n_3, n_4)
- 2) Inizializzazione di max: $\text{max} = n_1$
- 3) SE $n_2 > \text{max}$
 $\text{max} = n_2$
- 4) SE $n_3 > \text{max}$... fai $\text{max} = n_3$
- 5) SE $n_4 > \text{max}$... fai $\text{max} = n_4$
- 6) OUTPUT(max)

dobbiamo eseguire questo algoritmo, per convincerci che funge ...

prima cosa, disegniamo le variabili (le locazioni di memoria associate alle variabili farlo e poi proseguire



passo 1 ... in input abbiamo 12 22 18 18

passo 2

passo 3

passo 4 n_3 perde ... max rimane 22

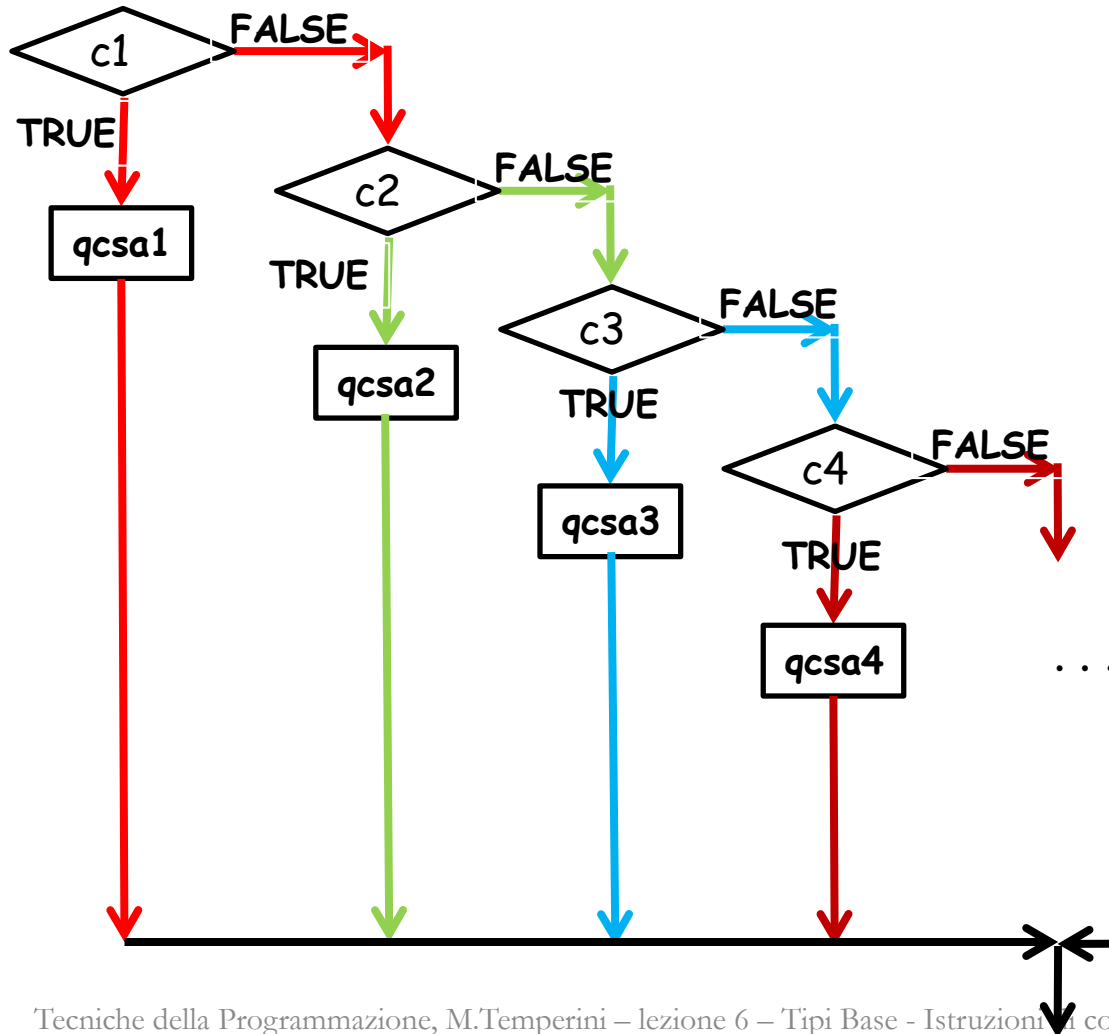
passo 5 n_4 perde ... max rimane 22

passo 6 viene stampato 22 e il programma finisce

Cascata di if

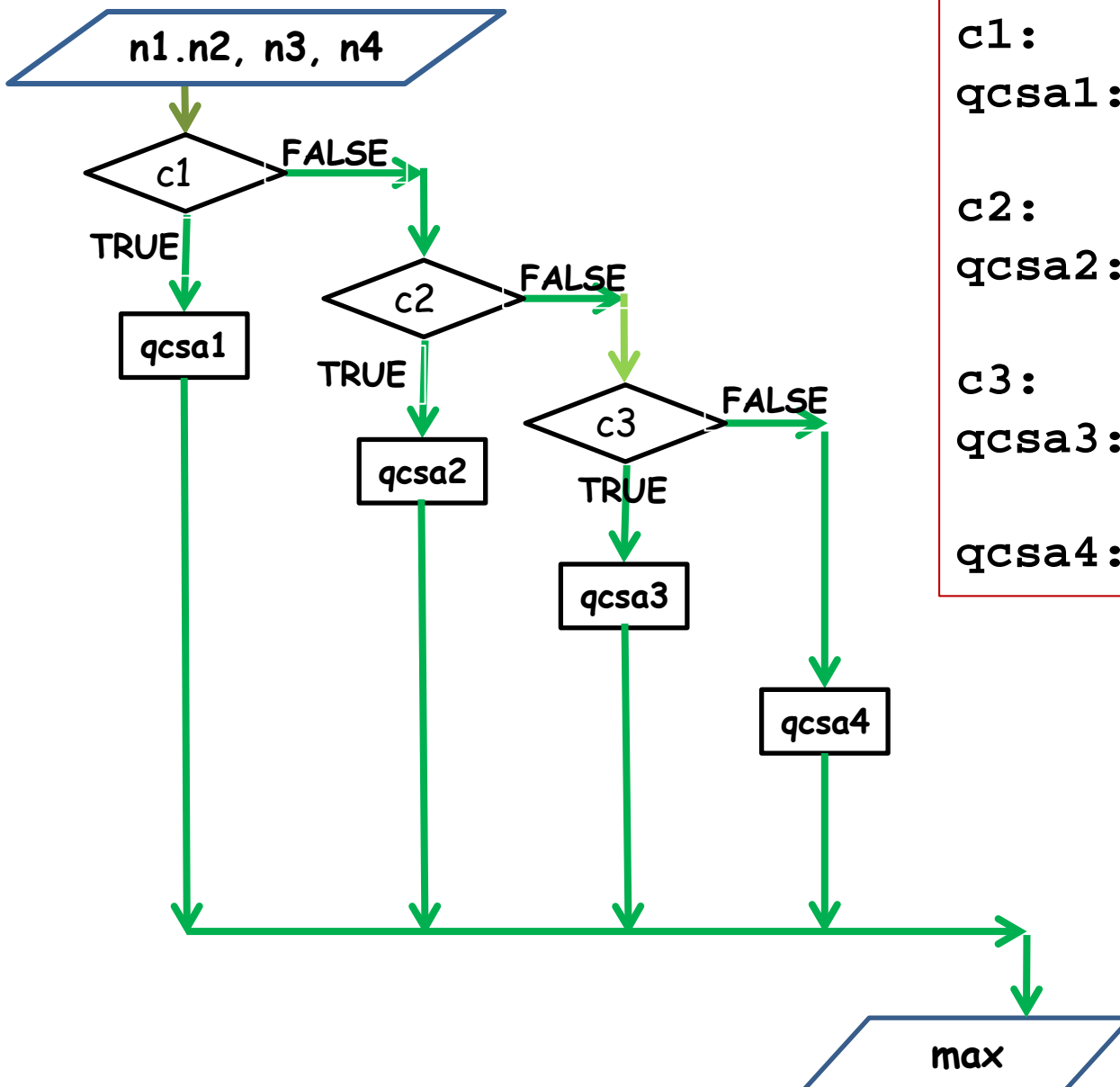
(annidati negli else ...)

l'esempio precedente mostrava un if annidato nella parte else. A volte la vita rende necessaria una CASCATA DI IF ... **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, **se vale una condizione, fai qualcosa, senno**, ...



```
if (c1)
    qcsa1;
else
    if (c2)
        qcsa2;
    else
        if (c3)
            qcsa3;
        else
            if (c4)
                qcsa4;
            else
                ...
    ...
```

Massimo tra QUATTRO numeri (alg. 4 e cascata)



c1:	???
qcsa1:	☺
c2:	???
qcsa2:	☺
c3:	???
qcsa3:	☺
qcsa4:	max=n4

suggerimento segue

Massimo tra QUATTRO numeri (alg. 4 e cascata)

PRIMA, scrivere l'algorithm4 nel caso di 4 numeri

nel caso 3 numeri era

0) i dati: $n1, n2, n3$ (input), e max

1) INPUT ($n1, n2, n3$)

2) SE $n1 > n2 \ \&\& \ n1 > n3$

2.1) max = $n1$

ALTRIMENTI

2.2) SE $n2 > n3$

2.2.1) max = $n2$

ALTRIMENTI

2.2.2) max = $n3$

3) OUTPUT(max)

c1: ???

qcsa1: ☺

c2: ???

qcsa2: ☺



0) i dati: $n1, n2, n3, n4$ (input), e max

...

2) condizione per assegnare $n1$ a max? Se $n1$ e' piu' grande di tutti ...

2.1) max = $n1$

ALTRIMENTI

...

Massimo tra QUATTRO numeri (alg. 4 e cascata)

PRIMA, scrivere l' algoritmo4 nel caso di 4 numeri

nel caso 3 numeri era

0) i dati: n_1, n_2, n_3 (input), e max
1) INPUT (n_1, n_2, n_3)
2) SE $n_1 > n_2 \ \&\& \ n_1 > n_3$
 2.1) max = n_1
 ALTRIMENTI
 2.2) SE $n_2 > n_3$
 2.2.1) max = n_2
 ALTRIMENTI
 2.2.2) max = n_3
3) OUTPUT(max)

c1: ???

qcsa1: ☺

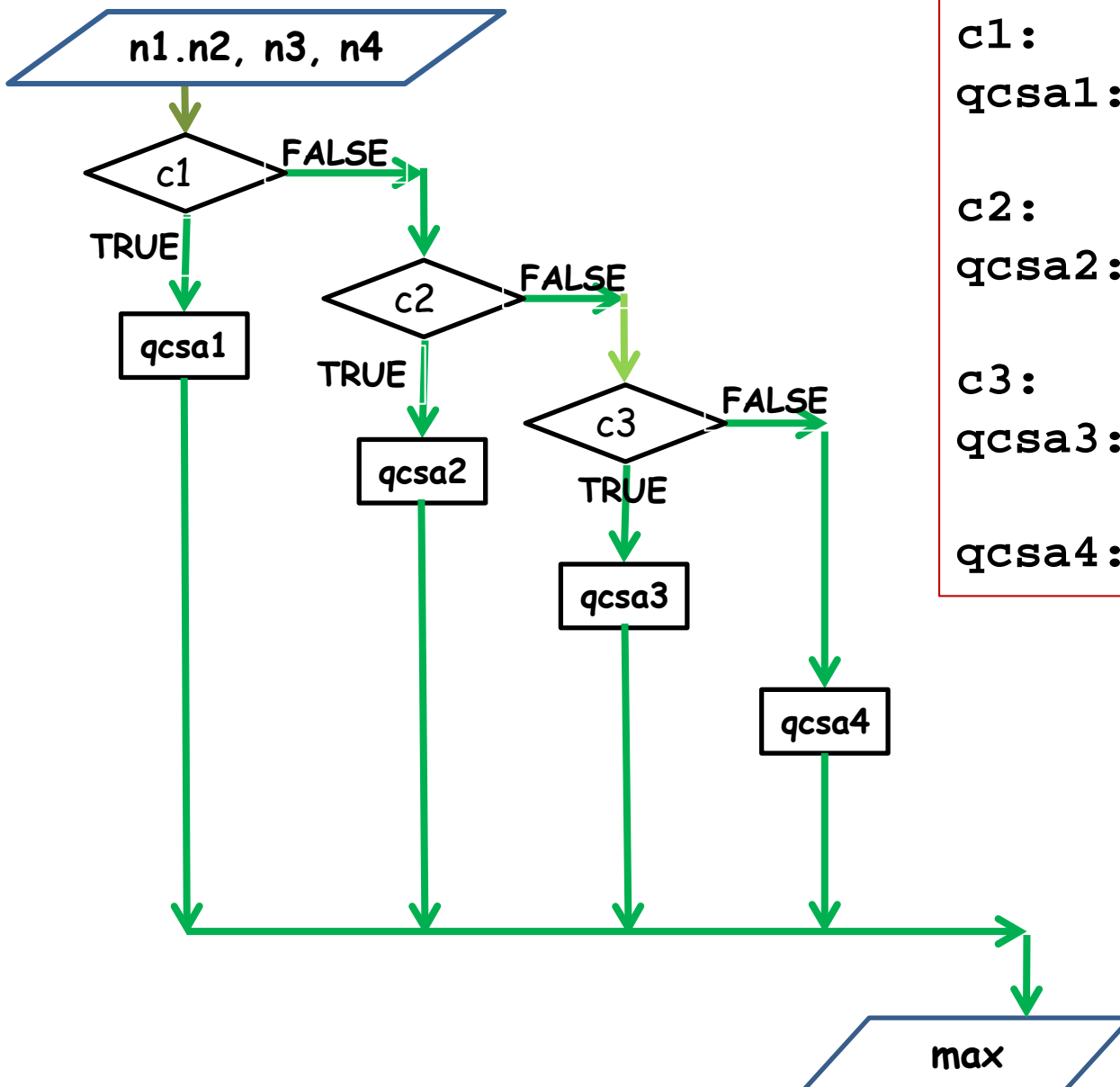
c2: ???

qcsa2: ☺

0) i dati: n_1, n_2, n_3, n_4 (input), e max
1) INPUT (n_1, n_2, n_3, n_4)
2) SE $n_1 > n_2 \ \&\& \ n_1 > n_3 \ \&\& \ n_1 > n_4$
 2.1) max = n_1
 ALTRIMENTI
 2.2) SE $n_2 > n_3 \ \&\& \ n_2 > n_4$
 2.2.1) max = n_2
 ALTRIMENTI
 2.2.2) SE $n_3 > n_4$
 2.2.2.1) max = n_3
 ALTRIMENTI
 2.2.2.2) max = n_4
3) OUTPUT(max)

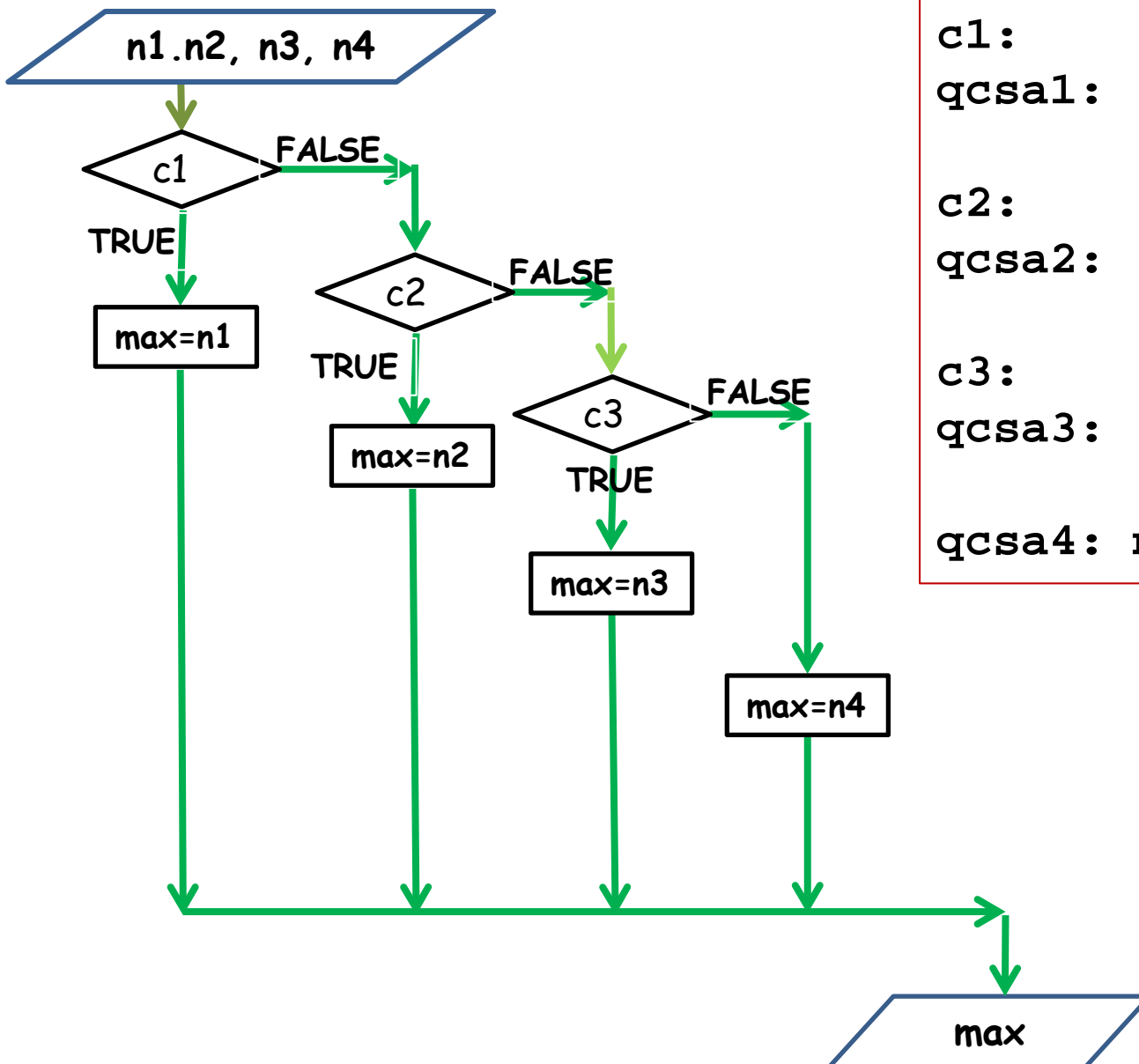
☺ ora il diagramma di flusso

Massimo tra QUATTRO numeri (alg. 4 e cascata)



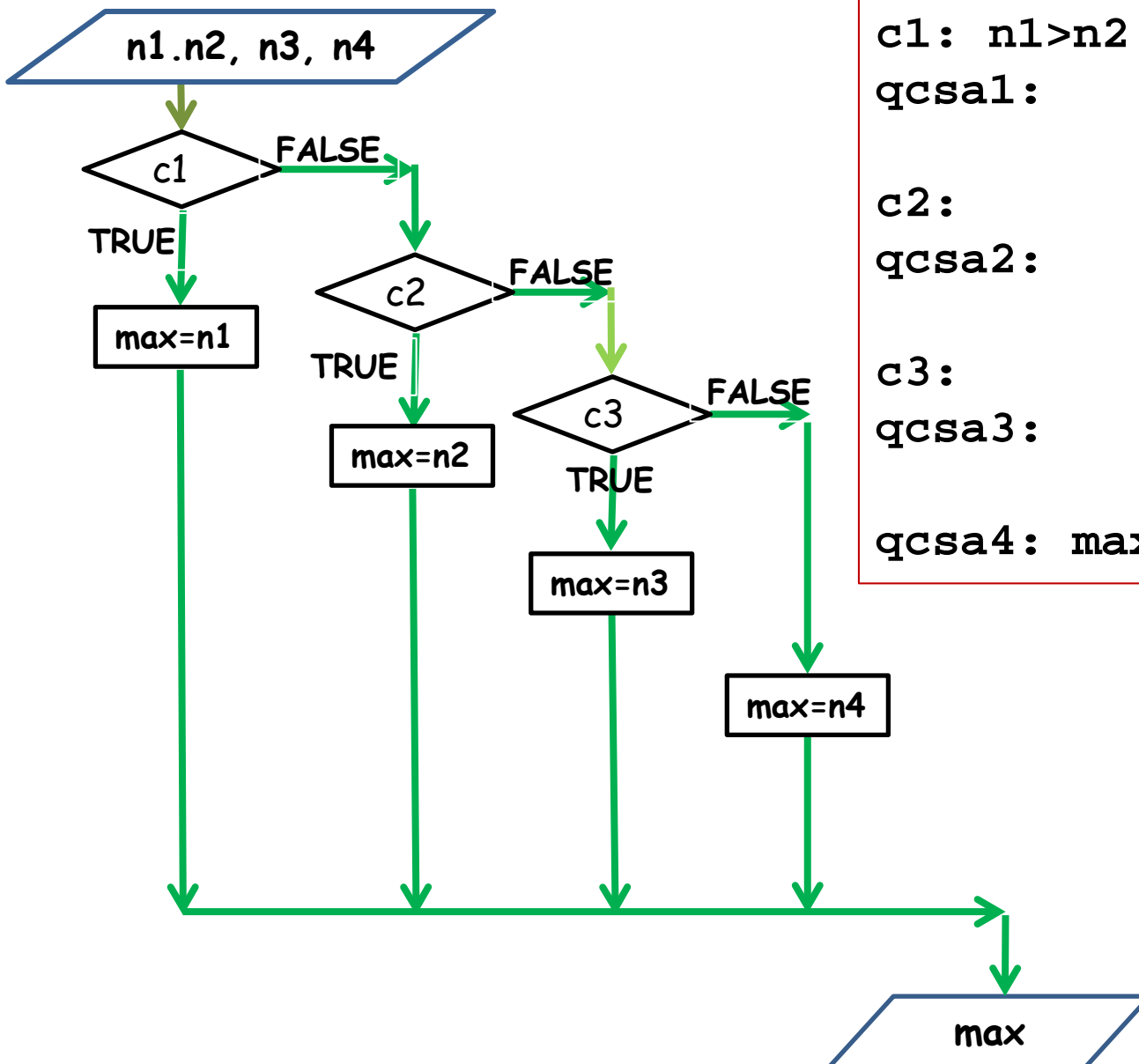
c1:	???
qcsa1:	☺
c2:	???
qcsa2:	☺
c3:	???
qcsa3:	☺
qcsa4:	max=n4

Massimo tra QUATTRO numeri (alg. 4 e cascata)



c1:	☺
qcsa1:	max=n1
c2:	???
qcsa2:	max=n2
c3:	???
qcsa3:	max=n3
qcsa4:	max=n4

Massimo tra QUATTRO numeri (alg. 4 e cascata)



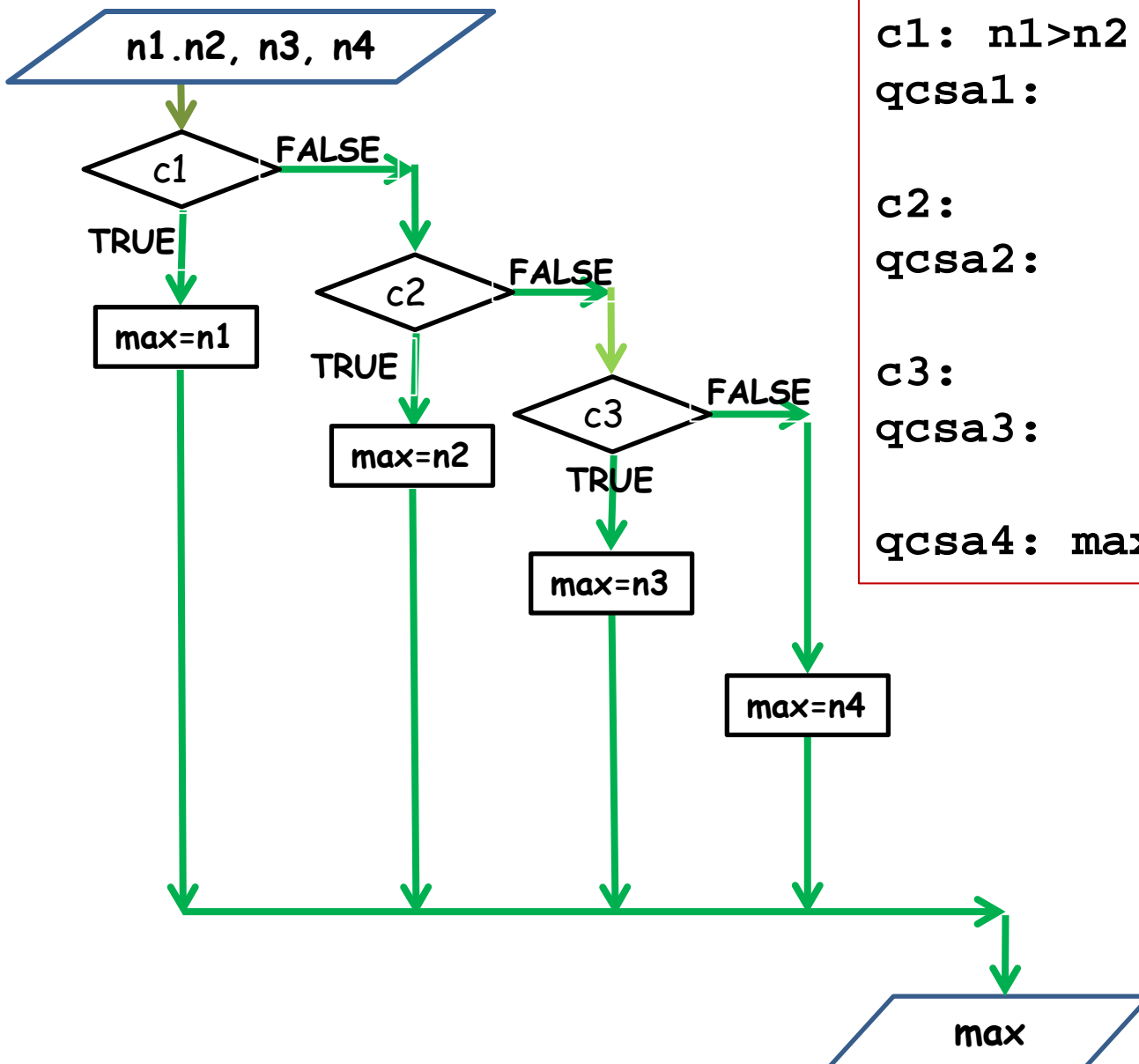
```
c1: n1>n2 && n1>n3 && n1>n4
qcsa1:      max=n1

c2:      ???
qcsa2:      max=n2

c3:      ???
qcsa3:      max=n3

qcsa4: max=n4
```

Massimo tra QUATTRO numeri (alg. 4 e cascata)



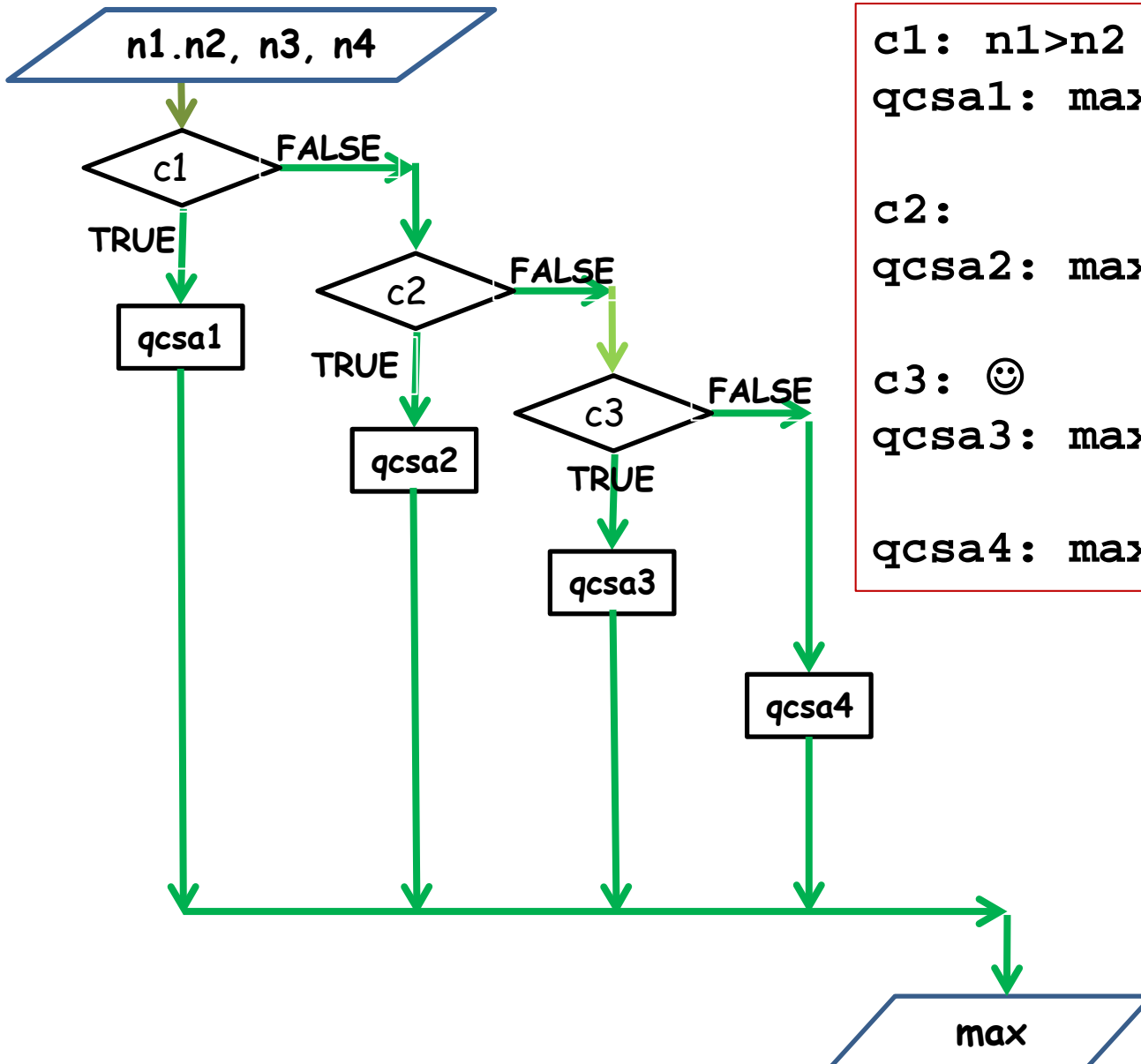
c1: $n1 > n2 \ \&\& \ n1 > n3 \ \&\& \ n1 > n4$
qcsa1: max=n1

c2: ☺
qcsa2: max=n2

c3: ???
qcsa3: max=n3

qcsa4: max=n4

Massimo tra QUATTRO numeri (alg. 4 e cascata)



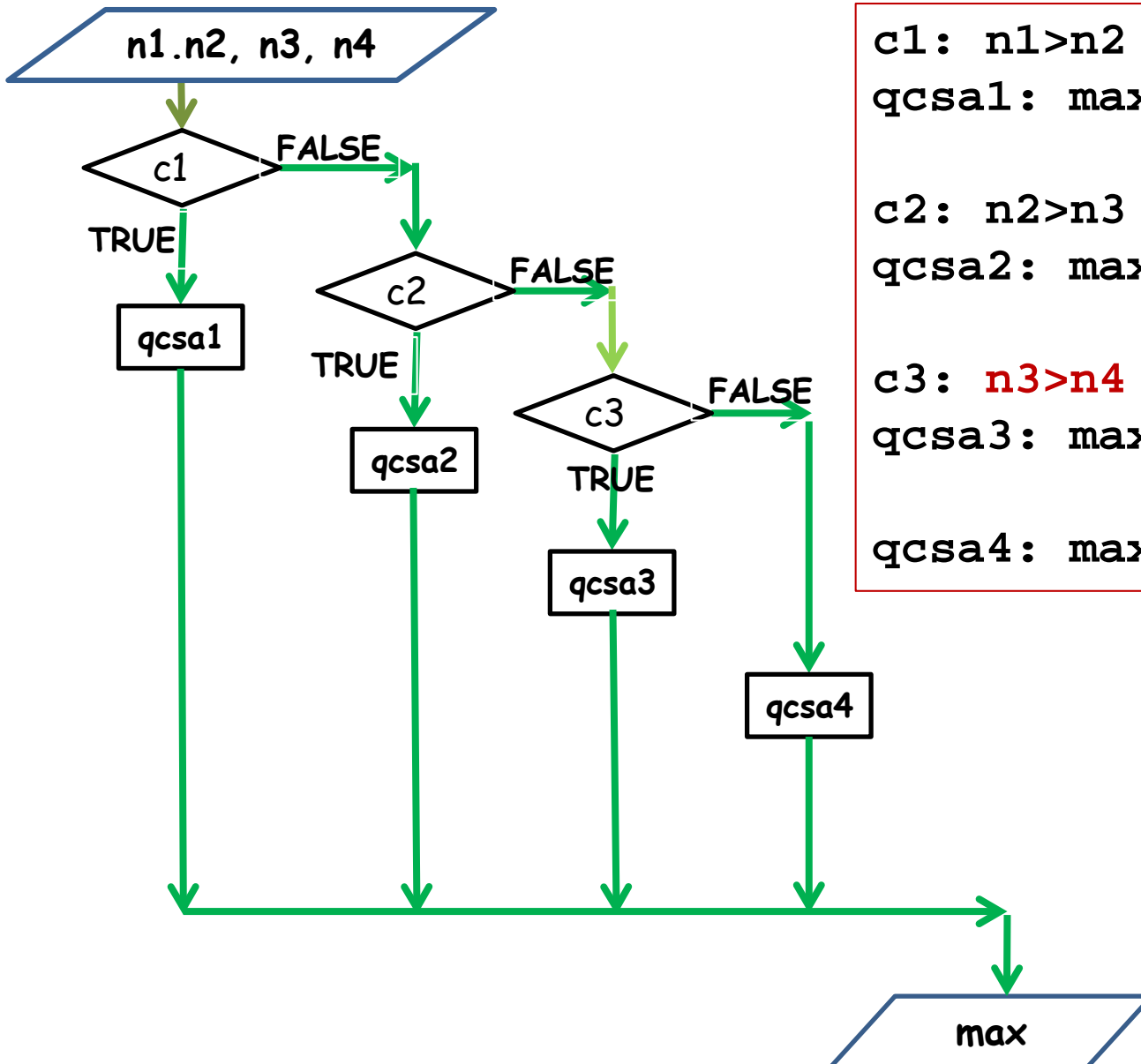
`c1: n1>n2 && n1>n3 && n1>n4`
`qcsa1: max=n1`

`c2: n2>n3 && n2>n4`
`qcsa2: max=n2`

`c3: ☺`
`qcsa3: max=n3`

`qcsa4: max=n4`

Massimo tra QUATTRO numeri (alg. 4 e cascata)



`c1: n1>n2 && n1>n3 && n1>n4`
`qcsa1: max=n1`

`c2: n2>n3 && n2>n4`
`qcsa2: max=n2`

`c3: n3>n4`
`qcsa3: max=n3`

`qcsa4: max=n4`

abbiamo visto
l'annidamento di
un if nella parte
else di un altro if

Ambigui

L'annidamento nella parte "TRUE" dell'istruzione condizionale è meno frequente, e più foriero di problemi...

(esempio: programma che riceve un intero e stampa "ma è grande" se è maggiore di 100 e stampa "ma è negativo" se è <0, e poi stampa "bella giornata")

```
printf ("Caro/a utente, dammi un numero: ");
scanf ("%d", &n);

if (n>=0)
    if (n>100)
        printf ("ma %c grande!\n", 138);
else
    printf ("ma %c negativo!\n", 138);
printf("bella giornata ...\n");
```

? cosa produce con n = 1000, 100, -10 ?

☺ provare il programma

Ambigui

L'annidamento nella parte "TRUE" dell'istruzione condizionale è meno frequente, e più foriero di problemi...

```
printf ("Caro/a utente, dammi un numero: ");  
scanf ("%d", &n);
```

```
if (n>=0)  
    if (n>100)  
        printf ("ma %c grande!\n", 138);  
else  
    printf ("ma %c negativo!\n", 138);  
printf("bella giornata ...\n");
```

cosa produce con
n == 1000, 100, -10

n==1000

```
Caro/a utente, dammi un numero: 1000  
ma è grande!  
bella giornata ...
```

n==100

```
Caro/a utente, dammi un numero: 100  
ma è negativo!  
bella giornata ...
```

-10
Caro/a utente, dammi un numero: -10
bella giornata ...

Ambigui

L'annidamento nella parte "if" dell'istruzione condizionale è meno frequente, e più foriero di problemi...

```
printf ("Caro/a utente, dammi un numero: ");  
scanf ("%d", &n);
```

```
if (n>=0)
```

```
    if (n>100)
```

```
        printf ("ma %c grande!\n", 138);
```

```
else
```

```
    printf ("ma %c negativo!\n", 138);
```

```
printf("bella giornata ...\n");
```

questa è la parte "TRUE" del primo if

questa è la parte TRUE del secondo if

questa è la parte ELSE del secondo if

questa istruzione è al di fuori degli if

```
Caro/a utente, dammi un numero: 100  
ma è negativo!  
bella giornata ...
```

Ambigui

L'annidamento nella parte "if" dell'istruzione condizionale è meno frequente, e più foriero di problemi...

```
printf ("Caro/a utente, dammi un numero: ");
scanf ("%d", &n);
if (n>=0)
    if (n>100)
        printf ("ma %c grande!\n", 138);
else
    printf ("ma %c negativo!\n", 138);
printf("bella giornata ... \n");
```

Il compilatore interpreta l'occorrenza della *keyword* "else" come introduttiva della parte else del più vicino if precedente. A dispetto dell'indentazione dell'esempio, che è stata concepita qui solo per ingannare, qui ci sono un *if_else* e un *if senza else*: il primo annidato nel secondo (vedi il codice qui sotto)

```
if (n>=0)
    if (n>100)
        printf ("ma è grande!\n");
    else
        printf ("ma è negativo!\n");
```

Come dovrebbe essere il programma per ottenere le stampe che ci aspettiamo? Perché non parlarne durante la prossima EG? Nella prossima slide c'è una frase che specifica "quel che volevamo". Nella successiva il codice ... da guardare dopo aver scritto e provato il programma che "fa quel che volevamo".

Ecco quello che volevamo veramente: il programma stampa «grande» se il numero è maggiore di 100, «negativo» se è negativo, e poi stampa «bella giornata»

Ecco quello che volevamo veramente: il programma stampa «grande» se il numero è maggiore di 100, «negativo» se è negativo, e poi stampa «bella giornata»

```
printf ("Caro/a utente, dammi un numero: ");
scanf ("%d", &n);
if (n>=0) {
    if (n>100)
        printf ("ma %c grande!\n", 138);
}
else printf ("ma %c negativo!\n", 138);
printf("bella giornata ... \n");
```

Tecniche della Programmazione, lez. 6

- Esercizi

Conversione Esplicita (casting)

Vedi Esercizi

```
int n=10, m=4;
```

```
... n ...
```

```
... (double)n ...
```

è un'espressione intera il cui valore è 10

è un'espressione double, il cui valore è 10.0 (FP/64bit)

Es. calcolo della media di tre numeri

```
int n1=10, n2=4, n3=12, somma;
```

```
somma = n1+n2+n3;
```

```
... somma/3 ...
```

/* n1+n2+n3 è un'espressione intera; somma ora
contiene un valore intero (26) */

/* somma/3 è un'espressione intera il cui valore è 8 */

/* NB la media esatta sarebbe 8.666667 ... circa */

Quindi?

Come si scrive un'espressione double che rappresenti il valore esatto della media?

? ... (double) (somma/3) ...

? ... (double) somma/3 ...

? ... somma / (double) 3 ...

? ... somma/3 (double) ...

? ... (double) somma / (double) 3



queste sono tutte conversioni esplicite ...
magari verificate con un piccolo programma e
poi guardate dopo

Conversione Esplicita (casting)

```
int n=10, m=4;
```

... n ...

è un'espressione intera il cui valore è 10

... (double)n ...

è un'espressione double, il cui valore è 10.0 (FP/64bit)

Es. calcolo

Tutte le printf sono con formato %g

Somma/3 (double) è scritta male ... che dice il compilatore?

```
int n1=10
```

```
somma
```

(double)(somma/3) produce un valore double, ma partendo da (somma/3), che è 8 ...

... Le altre sono espressioni di divisione in cui almeno uno degli operandi è convertito correttamente a double, quindi si usa la divisione double e il risultato è (più) corretto.

Quindi?

Come si scrive un'espressione double che rappresenti il valore giusto di somma/3

```
no ... (double) (somma/3) ...
```

```
yes ... (double) somma/3 ...
```

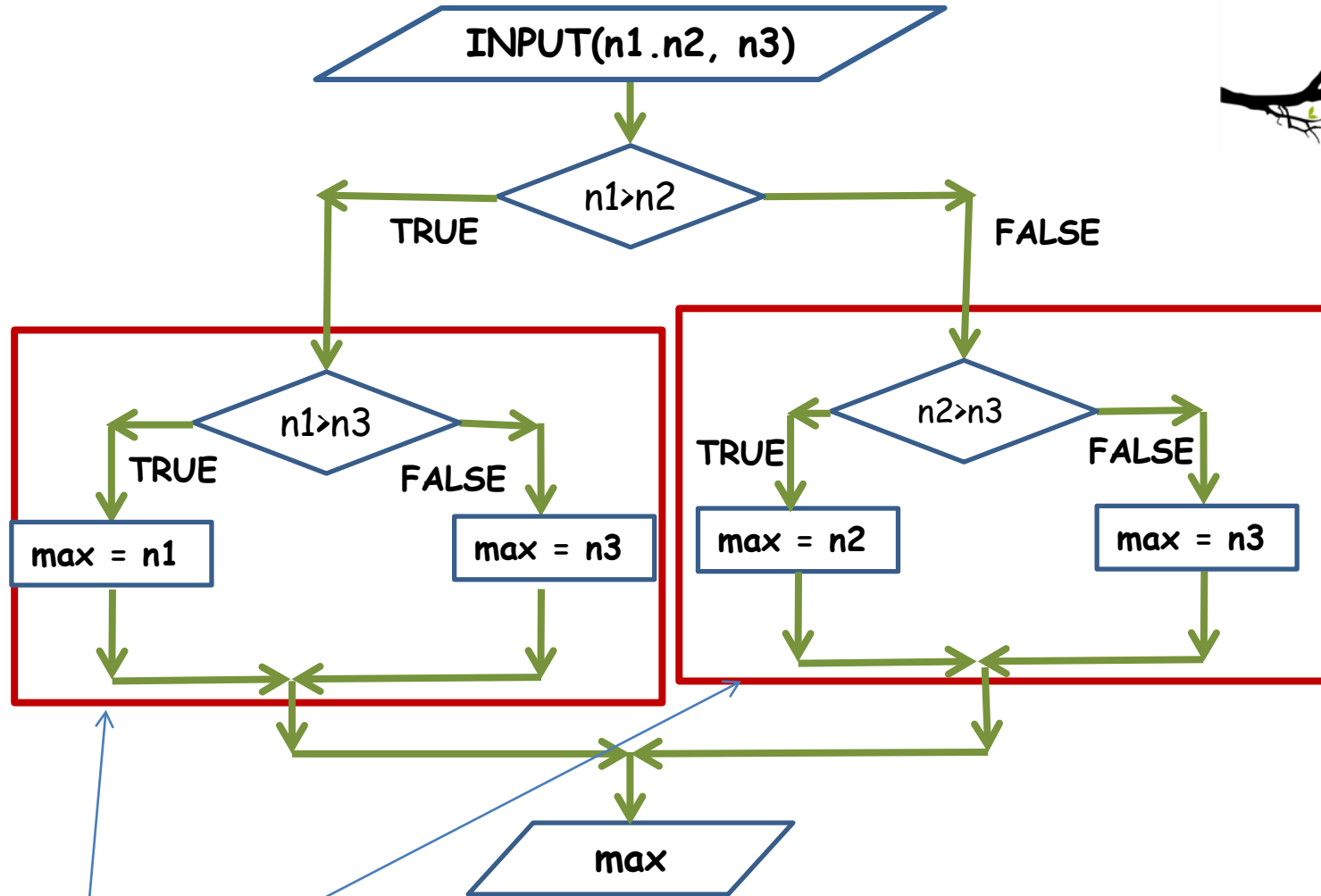
```
yes ... somma / (double) 3 ...
```

```
? ... somma/3 (double) ...
```

```
yes ... (double) somma / (double) 3
```

```
C:\Users\marcotemperini\Desktop\n  
stampiamo ... 8  
stampiamo ... 8.66667  
stampiamo ... 8.66667  
stampiamo ... 8.66667
```

Massimo tra tre numeri altra tecnica (alg. 3)



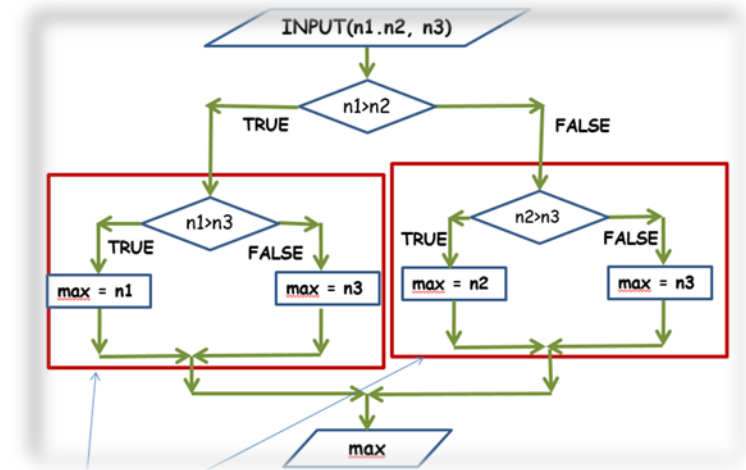
istruzioni strutturate (condizionali) ANNIDATE
dentro alla prima istruzione condizionale

Scrivere il programma corrispondente
(vedi Esercizi)

Massimo tra tre numeri - alg.3

```
int main () {  
    int n1, n2, n3,          /* i tre numeri */  
        max;                /* per rappresentare il massimo ...  
  
    printf ("Caro/a utente, dammi tre numeri: ");  
    scanf ("%d %d %d", &n1, &n2, &n3);          /* lettura */
```

```
    if (n1>n2)  
        if (n1>n3)  
            max = n1;  
        else max =n3;  
    else  
        if (n2 > n3)  
            max = n2;  
        else max = n3;
```



```
    printf ("beh, il massimo %c %d\n", 138, max);
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```