

Introduzione al linguaggio C

Espressioni e operatori
Strutture di controllo

Operatori ed Espressioni

Espressioni

- Una espressione è un qualcosa che ha un valore numerico e un tipo

- Es.

a /* costante simbolica*/

20 /* costante letterale */

h /* variabile */

L'espressione a vale 2, l'espressione h vale -0.1, mentre la costante letterale vale 20.

Espressioni

- Le seguenti sono espressioni più complesse:
- `2+5; /*somma tra costanti*/`
- `x=a+10; /* assegnazione di una somma tra una costante e una variabile*/`
- Una espressione è una combinazione valida di costanti, variabili e *operatori*.

Operatori

- Un *operatore* è un simbolo che indica al C di eseguire una operazione su uno o più operandi.
- Esistono tre classi di operatori:
 - **aritmetici**
 - **logici e relazionali**
 - **unari.**

Operatori aritmetici unari

| Operatore | Simbolo | Azione | esempio |
|------------|---------|----------------------------|----------|
| Incremento | ++ | Incrementa l'operando di 1 | ++X, X++ |
| Decremento | -- | Decrementa l'operando di 1 | --X, X-- |

Esempio

x=10;

y=++x; /*incremento di x e sostituzione in y*/

(x=10 e y=11) è diverso da

x=10;

y=x++; /* y=x+1 e x viene incrementato DOPO*/

In cui (x=10 e y=10).

Esempio

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

```
int a, b;
```

```
main()
```

```
{
```

```
    a=b=5;
```

```
    printf("\n %d %d", a--, --b);
```

```
    printf("\n %d %d", a--, --b);
```

```
    printf("\n %d %d", a--, --b);
```

```
    printf("\n %d %d", a--, --b);
```

```
    printf("\n %d %d", a--, --b);
```

```
}
```

Decremento successivo

Decremento precedente

| | |
|---|---|
| 5 | 4 |
| 4 | 3 |
| 3 | 2 |
| 2 | 1 |
| 1 | 0 |

Operatori aritmetici binari

| | |
|--|--|
| <code><op1> + <op2></code> | Somma i due operandi |
| <code><op1> - <op2></code> | Sottrae dal primo il secondo operando |
| <code><op1> * <op2></code> | Moltiplica gli operandi |
| <code><op1> / <op2></code> | Divide il primo con il secondo operando |
| <code><op1> % <op2></code> | Resto della divisione tra il primo e il secondo operando |

L'operatore % non può essere applicato ai tipi `float`, `double`

Operatori binari (bit a bit)

- Possono essere usati solo con:
 - char, int
- Sono utilizzati per la maggior parte in programmi che interagiscono direttamente con l'hardware
- Operatori di Shift (di n posizioni) e logici
 - $x \ll n$, $x \gg n$
 - $\&$ (and), $|$ (or inclusivo), \wedge (or esclusivo)

- Shift:
 - $x \ll (\gg)n$: x con segno il 1° bit rimane (è aggiunto)
 x senza segno il 1° bit diventa 0
- L'operatore & viene utilizzato per azzerare particolari insiemi di bit:
- $n = n \& '\backslash 177';$ //azzerare tutti i bit di n , esclusi i 7 meno significativi
- L'operatore | viene utilizzato per forzare a 1 singoli bit:
- $x = x | SET_ON;$ //pone a 1, in x , i bit che sono =1 in SET_ON

ATTENZIONE:

Non confondere gli operatori bit a bit **&** e **|** con gli operatori logici **&&** e **||** perché si possono commettere errori!

se x vale 1 e y vale 2,

x & y vale 0,

x && y vale 1

Precedenze

| | |
|---------|------------|
| ++, -- | massima |
| *, /, % | intermedia |
| +, - | minima |

- Espressioni con operatori dello stesso livello sono valutate da sinistra verso destra

...continua

- Usare le parentesi per rendere chiaro l'ordine di valutazione dell'espressione
- Non sovraccaricare l'espressione
 - Spezzare un' espressione complessa in più espressioni
 - Soprattutto se si utilizzano gli operatori (++) o (--)

Assegnazione

- $x = (25 - 2 * (10 + (8 / 2))) ;$
- $i = j = k = 0 ;$
- $=$ ha priorità più bassa rispetto agli altri operatori ed associa da destra a sinistra
- Poiché anche l'assegnazione è un operatore si possono scrivere contrazioni:
- $z = (x = 1) + (y = 2) ;$

Conversioni di tipo

- Spesso nelle espressioni si mischiano tipi di dati diversi. Quale è il tipo dell'espressione risultante?
- Conversione automatica al tipo che comprende gli altri
 - Se f è float e i è intero:
 - f/i è convertito a float;
 - Se
 - $f = i/j$; con j intero?
- Conversione esplicita:
 - $f = (\text{float})i/j$;

Conversioni di tipo (assegnazione)

```
int i;  
char c;
```

```
i=c;  
c=i;
```

Oppure :

```
/*si potrebbe verificare una perdita di  
informazioni*/
```

```
c=i;  
i=c;
```


Assegnazione+operatore aritmetico

- $\langle op1 \rangle \ += \ \langle op2 \rangle$ $\langle op1 \rangle = \langle op1 \rangle + \langle op2 \rangle$
- $\langle op1 \rangle \ -= \ \langle op2 \rangle$ $\langle op1 \rangle = \langle op1 \rangle - \langle op2 \rangle$
- $\langle op1 \rangle \ *= \ \langle op2 \rangle$ $\langle op1 \rangle = \langle op1 \rangle * \langle op2 \rangle$
- $\langle op1 \rangle \ /= \ \langle op2 \rangle$ $\langle op1 \rangle = \langle op1 \rangle / \langle op2 \rangle$
- $\langle op1 \rangle \ \%= \ \langle op2 \rangle$ $\langle op1 \rangle = \langle op1 \rangle \% \langle op2 \rangle$

Operatori relazionali

| | |
|---|---|
| $\langle \text{op1} \rangle = = \langle \text{op2} \rangle$ | 1 se gli operatori sono uguali/ 0 se diversi |
| $\langle \text{op1} \rangle != \langle \text{op2} \rangle$ | 1 se gli operatori sono diversi/ 0 se uguali |
| $\langle \text{op1} \rangle < \langle \text{op2} \rangle$ | 1 se il 1 ^o è minore del 2 ^o operatore/ 0 altrimenti |
| $\langle \text{op1} \rangle > \langle \text{op2} \rangle$ | 1 se il 1 ^o è maggiore del 2 ^o operatore/ 0 altrimenti |
| $\langle \text{op1} \rangle <= \langle \text{op2} \rangle$ | 1 se il 1 ^o è minore o uguale del 2 ^o operatore/ 0 altrimenti |
| $\langle \text{op1} \rangle >= \langle \text{op2} \rangle$ | 1 se il 1 ^o è maggiore o uguale del 2 ^o operatore/ 0 altrimenti |

■Precedenze: <, <=, >, >= hanno precedenza su =, !=

Valutazione di espressioni relazionali (esercizio: output ??)

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

```
int a;
```

```
main()
```

```
{
```

```
    a=(5= =5);
```

```
    printf("\n a=(5= =5)\na=%d",a);
```

```
    a=(5! =5);
```

```
    printf("\n a=(5!=5)\na=%d",a);
```

```
    a=(12= =12)+(5!=1);
```

```
    printf("\n a=(12= =12)+(5!=1)\na=%d",a);
```

```
}
```

Operatori logici e precedenze

| Operatore | Valutazione | Precedenza |
|--------------------|---|------------|
| (!espr) | 1, se espr è falsa/0 se vera | massima |
| (espr1)&&(espr2) | 1, se espr1 e espr2 sono vere/ 0, altrimenti | intermedia |
| (espr1) (espr2) | 1, se espr1 o espr2 è vera/ 0, altrimenti | minima |

- Gli operatori logici binari hanno precedenza inferiore a tutti gli altri operatori binari

Esercizio: output??

```
#include <stdio.h>
int a = 5, b = 6, c = 5, d = 1;
int x;

main()
{
    x= a<b || a<c && c>d;
    printf("\n espressione senza parentesi vale %d",x);

    x= (a<b || a<c) && c>d;
    printf("\n espressione con parentesi vale %d",x);
}
```

Esercizio

- Esprimere un intero (esempio 14000) in ore, minuti e secondi.

Programmazione strutturata

Introduzione

- Prima di scrivere un programma è essenziale:
 - avere una piena conoscenza del problema da risolvere
 - un approccio pianificato con cura per risolverlo
- Per scrivere un programma è essenziale:
 - conoscere quali “componenti” sono disponibili
 - usare buoni principi di programmazione

Algoritmi

- Risoluzione di problemi
 - Esecuzione, in un ordine specifico, di una serie di azioni
- Algoritmo: procedura in termini di
 - *Azioni* da eseguire
 - *Ordine* in cui le azioni devono essere eseguite
- Controllo del programma
 - Specificazione dell'ordine in cui le istruzioni devono essere eseguite

Pseudocodice

- Pseudocodice
 - Linguaggio artificiale e informale che aiuta i programmatori a sviluppare gli algoritmi
 - Simile alla lingua parlata
 - Non è un vero linguaggio di programmazione
 - Aiuta a “riflettere” sul programma prima di scriverlo in un vero linguaggio di programmazione

Diagrammi di flusso

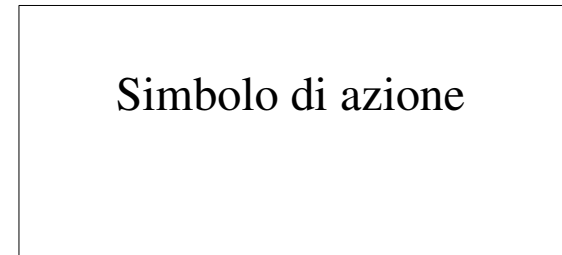
Simbolo di connessione



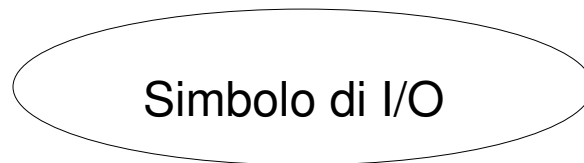
Simbolo di decisione



Simbolo di azione



Simbolo di I/O



Programmazione strutturata

- Esecuzione sequenziale
 - Le istruzioni sono eseguite una dopo l'altra nell'ordine in cui sono scritte
- Trasferimento di controllo
 - L'istruzione successiva da eseguire non è quella che segue nella sequenza
 - Un abuso dell'istruzione `goto` può far insorgere molti problemi.
- Bohm and Jacopini
 - Tutti i programmi possono essere scritti in termini di **3 strutture di controllo**
 - **Struttura di Sequenza**: è implicita in C. I programmi sono eseguiti sequenzialmente per default.
 - **Struttura di Selezione** : **if**, **if/else**, e **switch** .
 - **Struttura di iterazione**: **while**, **do/while**, e **for**.

Istruzioni di selezione

Fanno parte di questa classe di istruzioni:

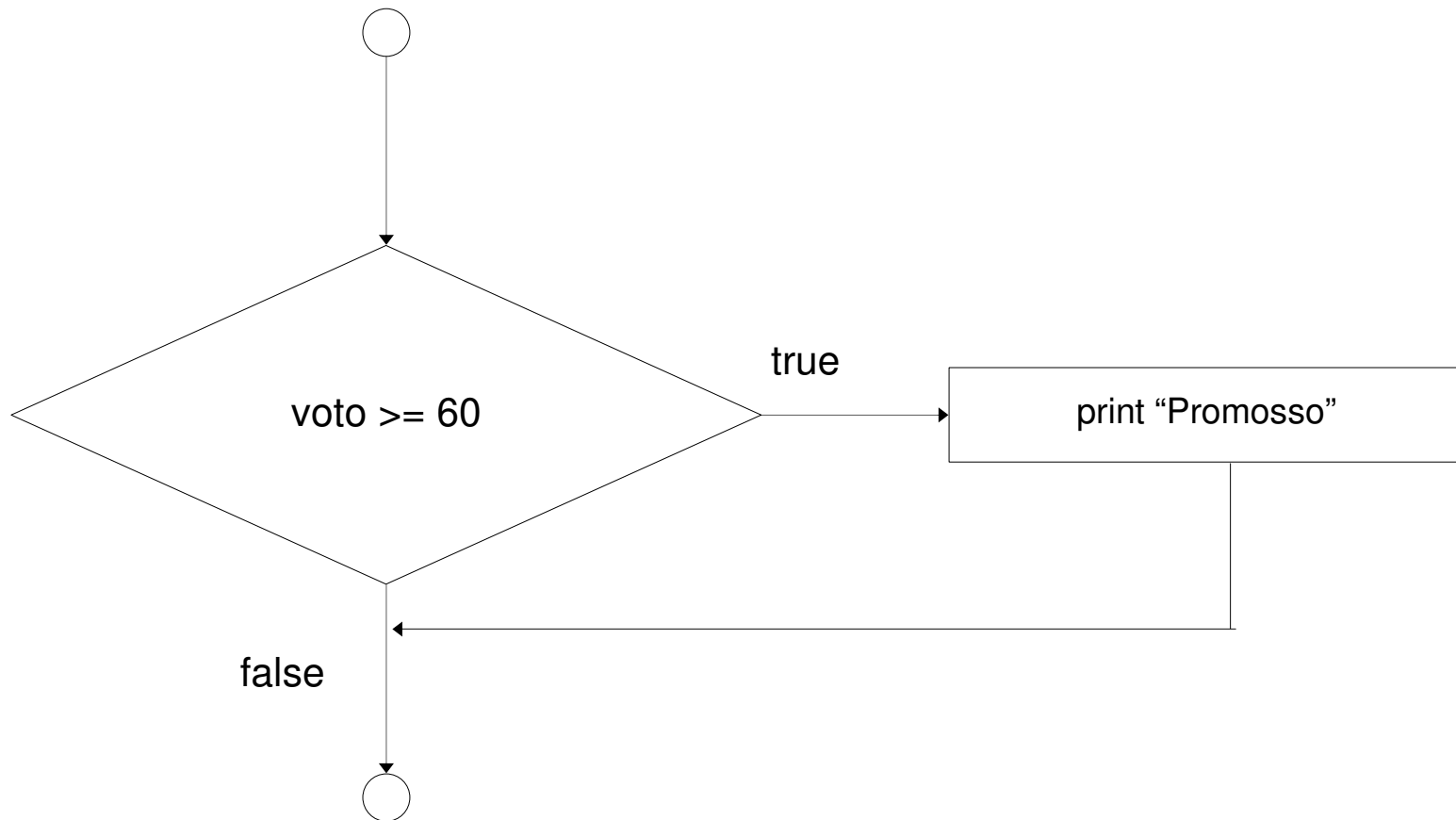
- `if`
 - `if - else`
 - `if - else - if`
- `switch`

Istruzione di selezione if

`if(espressione) istruzione;`

- Struttura di selezione
 - Usata per scegliere tra percorsi di azione alternativi
 - Pseudocodice: *Se il voto dello studente è maggiore o uguale a 60*
visualizza "Promosso"
- Se la condizione è **vera** (**true**)
 - viene eseguita l'istruzione di visualizzazione e il programma prosegue all'istruzione successiva
- Se la condizione è **falsa** (**false**)
 - l'istruzione di visualizzazione è ignorata e il programma prosegue all'istruzione successiva

...continua



Nota: operatore condizionale

- È un operatore ternario:
- `espr1 ? espr2 : epr3;`

```
if (espr1)
    espr2;
else
    espr3;
```

Ad esempio:

```
z = (x>y) ? x : y;
```


Struttura di selezione if-else

- **if**
 - Esegue l'azione indicata solo quando la condizione è **true**.
- **if/else**
 - Consente di specificare sia l'azione da eseguire quando la condizione è **true** sia quella da eseguire quando la condizione è **false**
- Psuedocodice: *Se il voto è maggiore o uguale a 60*
Visualizza "Promosso"
altrimenti
Visualizza "Bocciato"
- codice C :

```
if ( voto >= 60 )  
    printf( "Promosso\n" );  
else  
    printf( "Bocciato\n" );
```

...continua

- Le strutture **if/else** possono essere annidate
 - Verificare diversi casi ponendo strutture **if/else** all'interno di altre strutture **if/else**

Se il voto è maggiore o uguale a 90

Visualizza "A"

altrimenti

Se il voto è maggiore o uguale a 80

Visualizza "B"

altrimenti

Se il voto è maggiore o uguale a 70

Visualizza "C"

altrimenti

Se il voto è maggiore o uguale a 60

Visualizza "D"

altrimenti

Visualizza "F"

- Una sola condizione è soddisfatta, le altre istruzioni sono ignorate

...continua

```
if(espressione) istruzione;
```

```
else istruzione;
```

La forma generale di **if** quando è riferita a blocchi di istruzioni e' la seguente:

```
if(espressione)
{
    istruzione_1;
    .
    .
    istruzione_n;
}
else
{
    istruzione_1;
    .
    .
    istruzione_n;
}
```

- Istruzione composta:
 - Insieme di istruzioni all'interno di parentesi graffe

- Esempio:

```
if ( voto >= 60 )  
    printf( "Promosso.\n" );  
else {  
    printf( "Bocciato.\n" );  
    printf( "Devi seguire di nuovo questo  
corso.\n" );  
}
```

- Senza le parentesi,

```
printf( "Devi seguire di nuovo questo  
corso.\n" );
```

sarebbe eseguito in ogni caso

- **Blocco:** istruzione composta con dichiarazioni

Istruzioni `if` annidate

Un `if` *annidato* è un'istruzione `if` controllata da un altro `if` o `else`. Gli `if` annidati sono molto comuni in programmazione. La cosa più importante da ricordare è che un'istruzione `else` si riferisce sempre all'istruzione `if` più vicina che sia all'interno del medesimo blocco dell'`else` e che non sia già associata a un `else`.

Esempio in pseudo-C

```
if(i)
{
    if(j) istruzione1;
    if(k) istruzione2; /* questo if e' associato*/
    else                /* a questo else */
        istruzione3;
}
else istruzione 4;
/* questo if e' associato a if(i) */
```

Struttura di selezione

if-else-if

```
if(condizione)
    istruzione;
else if(condizione)
    istruzione;
else if(condizione)
    istruzione;
.
.
else
    istruzione;
```

- Alla prima condizione vera, viene eseguita l'istruzione ad essa associata ed il resto della scala **if-else-if** viene aggirato.
- Se nessuna delle condizioni è vera allora viene eseguita l'istruzione **else** finale. Se non c'è un **else** finale e tutte le altre condizioni sono false, allora non viene eseguita nessuna istruzione.

Esercizio: output??

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

```
int x,y;
```

```
main()
```

```
{
```

```
    x=100;
```

```
    y=10;
```

```
    if (x==y)
```

```
        printf("x=y");
```

```
    else if (x>y)
```

```
        printf("x>y");
```

```
    else
```

```
        printf("x<y");
```

```
    return 0;
```

```
}
```


Nota: operatori di uguaglianza (==) e di assegnamento (=)

- ```
if (payCode == 4)
 printf("You get a bonus!\n");
```

Controlla paycode, se è 4 hai vinto un bonus!

- ```
if ( payCode = 4 )  
    printf( "You get a bonus!\n" );
```

- **paycode** è inizializzato a 4
- 4 è diverso da zero, pertanto l'espressione è **true**, e il bonus è vinto, indipendentemente dal valore del codice di pagamento
- E' un errore logico, non sintattico

...continua

- Errore pericoloso
 - Solitamente non provoca un errore di sintassi
 - Nelle strutture di controllo può essere usata ogni espressione che produca un valore
 - I valori diversi da zero sono **true**, zero vale **false**

La struttura di selezione multipla

switch

- **switch**

- Utile quando una variabile od espressione debba essere confrontata distintamente con ognuno dei valori che può assumere e, a seconda del risultato del confronto, vengono intraprese azioni distinte.

- Struttura

- Serie di etichette **case** (caso) più un caso opzionale **default**

switch (value){

case '1':

actions

case '2':

actions

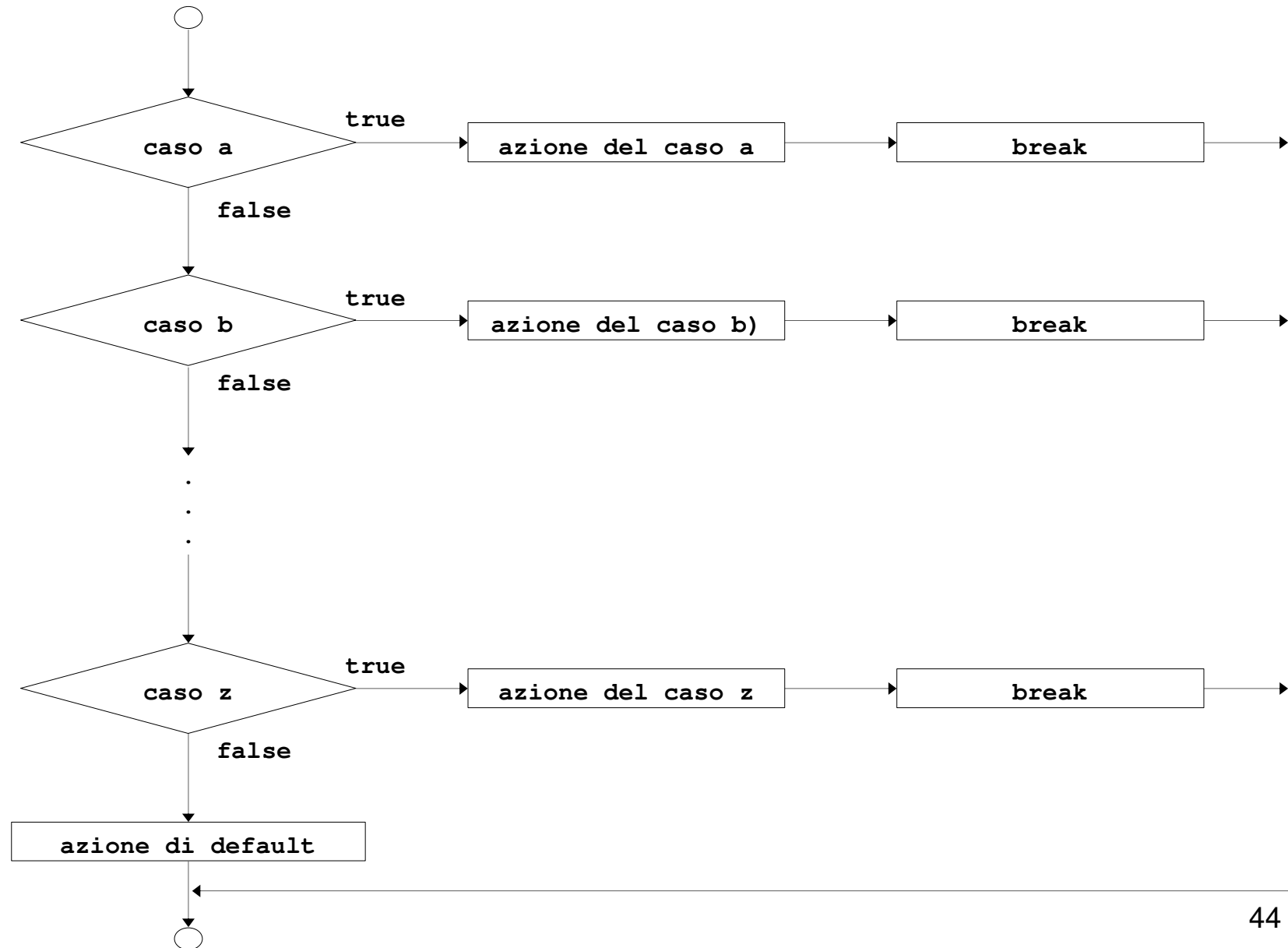
default:

actions

}

- **break**: causa l'uscita dalla struttura **switch**; l'esecuzione prosegue con la prima istruzione dopo la struttura **switch**.

...continua



```

1  /*
2      Counting letter grades */
3  #include <stdio.h>
4
5  int main()
6  {
7      int grade;
8      int aCount = 0, bCount = 0, cCount = 0,
9          dCount = 0, fCount = 0;
10
11     printf( "Enter the letter grades.\n" );
12     printf( "Enter the EOF character to end input.\n" );
13
14     while ( ( grade = getchar() ) != EOF ) {
15
16         switch ( grade ) {      /* switch nested in while */
17
18             case 'A': case 'a': /* grade was uppercase A */
19                 ++aCount;        /* or lowercase a */
20                 break;
21
22             case 'B': case 'b': /* grade was uppercase B */
23                 ++bCount;        /* or lowercase b */
24                 break;
25
26             case 'C': case 'c': /* grade was uppercase C */
27                 ++cCount;        /* or lowercase c */
28                 break;
29
30             case 'D': case 'd': /* grade was uppercase D */
31                 ++dCount;        /* or lowercase d */
32                 break;

```

```

33
34         case 'F': case 'f': /* grade was uppercase
35             ++fCount;        /* or lowercase f */
36             break;
37
38         case '\n': case ' ': /* ignore these in
39             break;
40
41         default: /* catch all other
42             printf( "Incorrect letter grade entered."
43             printf( " Enter a new grade.\n" );
44             break;
45     }
46 }
47
48 printf( "\nTotals for each letter grade are:\n" );
49 printf( "A: %d\n", aCount );
50 printf( "B: %d\n", bCount );
51 printf( "C: %d\n", cCount );
52 printf( "D: %d\n", dCount );
53 printf( "F: %d\n", fCount );
54
55 return 0;
56 }

```

Esecuzione



```
Enter the letter grades.  
Enter the EOF character to end input.  
A  
B  
C  
C  
A  
D  
F  
C  
E  
Incorrect letter grade entered. Enter a new grade.  
D  
A  
B  
  
Totals for each letter grade are:  
A: 3  
B: 2  
C: 3  
D: 2  
F: 1
```

Istruzioni di Iterazione

Fanno parte di questa classe di istruzioni i cicli:

- `while`
- `for`
- `do-while`

Elementi dell'iterazione

- Ciclo (Loop)
 - Insieme di istruzioni che il computer eseguirà ripetutamente finché una certa *condizione di continuazione del ciclo* rimarrà vera
- Ripetizioni controllate da un **contatore**
 - Iterazione definita - si conosce il numero delle ripetizioni
 - L'iterazione è controllata da una variabile contatore
- Ripetizioni controllate da un **valore sentinella**
 - Iterazione indefinita
 - Si usa quando il numero di ripetizioni non è conosciuto
 - Il valore sentinella indica "end of data"

Iterazione controllata da un contatore: formulazione degli algoritmi

- Il ciclo è ripetuto finché un contatore raggiunge un valore dato.
- Ciclo definito: il numero di ripetizioni è noto.
 - Esempio: *Una classe di dieci studenti sostiene un esame. Avete a disposizione le votazioni (degli interi da 0 a 100) per questo esame. Determinate la media della classe.*

- Pseudocodice:

Inizializzare il totale a zero

Inizializzare il contatore dei voti a uno

*Finché il **contatore dei voti** resta minore o uguale a **dieci***

Prendere dall'input il prossimo voto

Aggiungere il voto al totale

Aggiungere uno al contatore dei voti

Impostare il valore della media al totale diviso dieci

Visualizzare la media

...continua

- L'iterazione controllata da un contatore richiede:
 - il *nome* della variabile di controllo (o contatore di ciclo)
 - il *valore iniziale* della variabile di controllo
 - la condizione che verificherà il *valore finale* della variabile di controllo (cioè la condizione che determina se il ciclo deve continuare o meno)
 - l'*incremento* (o *decremento*) con cui la variabile di controllo sarà modificata ad ogni iterazione del ciclo.

..continua

- Esempio

```
int contatore=1;           /* inizializzazione */
while (contatore <= 10)    /*condizione di ripetizione
{
    printf( "%d\n", contatore );
    ++contatore;           /* incremento */
}
```

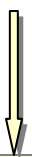
- `int contatore = 1;` definisce `contatore`, dichiara che è un intero, riserva spazio in memoria, e pone il suo valore iniziale 1

```

1  /*
2      Class average program with
3      counter-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8      int counter, grade, total, average;
9
10     /* initialization phase */
11     total = 0;
12     counter = 1;
13
14     /* processing phase */
15     while ( counter <= 10 ) {
16         printf( "Enter grade: " );
17         scanf( "%d", &grade );
18         total = total + grade;
19         counter = counter + 1;
20     }
21
22     /* termination phase */
23     average = total / 10;
24     printf( "Class average is %d\n", average );
25
26     return 0;    /* indicate program ended successfully */
27 }

```

Esecuzione



```

Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81

```

Iterazione controllata da un valore sentinella: formulazione degli algoritmi

- Consideriamo il seguente problema:

*Sviluppare un programma per il calcolo della media di una classe che elaborerà un **numero arbitrario** di votazioni ogni volta che il programma viene eseguito.*

- Il numero di studenti non è noto.
 - In quale modo il programma potrà sapere quando terminare l'immissione delle votazioni?
-
- Valore sentinella
 - Detto anche valore "segnale", "dummy" (fittizio) o "flag" (bandiera)
 - Indica "fine immissione dati".
 - Il ciclo termina quando viene immesso il valore sentinella.
 - Il valore sentinella deve essere scelto in modo da non essere confuso con gli altri dati in input (per esempio, -1 in questo caso)

- Pseudocodice:

Inizializzare il totale a zero

Inizializzare il contatore a zero

Prendere in input la prima votazione

Finché il dato immesso è diverso dal valore sentinella

Aggiungere la votazione al totale

Aggiungere uno al contatore

Prendere in input la prossima valutazione

Se il contatore non è uguale a zero

Impostare il valore della media al totale diviso per il contatore

Visualizzare la media

altrimenti

Visualizzare “Non sono state immesse valutazioni”


```

1  /*
2      Class average program with
3      sentinel-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8      float average;  /* new data type */
9      int counter, grade, total;
10
11     /* initialization phase */
12     total = 0;
13     counter = 0;
14
15     /* processing phase */
16     printf( "Enter grade,-1 to end: ");
17     scanf( "%d", &grade );
18
19     while ( grade != -1 ) {
20         total = total + grade;
21         counter = counter + 1;
22         printf( "Enter grade,-1 to end: ");
23         scanf( "%d", &grade );
24     }

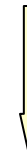
```

```

25
26     /* termination phase */
27     if ( counter != 0 ) {
28         average = ( float ) total / counter;
29         printf( "Class average is %.2f", average );
30     }
31     else
32         printf( "No grades were entered\n" );
33
34     return 0;  /* indicate program ended
35 }

```

Esecuzione



```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

```

Strutture di controllo nidificate

- **Problema**

- *Una scuola ha la lista dei risultati di un esame sostenuto da 10 studenti, (1 = promosso, 2 = bocciato).*
- *Scrivere un programma che analizzi i risultati*
 - *Se più di 8 studenti hanno superato l'esame, visualizzare il messaggio "Più di 8"*

- **Si noti che**

- Il programma deve elaborare 10 risultati
 - Sarà usato un ciclo controllato da un contatore
- Possono essere usati due contatori
 - Uno per il numero delle promozioni, uno per il numero delle bocciature
- Ciascun risultato è un numero (1 oppure 2)
 - Se il numero non è 1, si suppone che sia 2

- Pseudocodice:

Inizializzare le promozioni a zero

Inizializzare le bocciature a zero

Inizializzare gli studenti a zero

Finché il contatore degli studenti è minore o uguale a 10

Prendere in input il prossimo risultato di esame

Se lo studente è stato promosso

*Aggiungere uno ai promossi
altrimenti*

Aggiungere uno ai bocciati

Aggiungere uno al contatore degli studenti

Visualizzare il numero delle promozioni

Visualizzare il numero delle bocciature

Se più di 8 studenti sono stati promossi


Visualizzare il messaggio “Più di 8”

```

1
2  Analysis of examination results */
3 #include <stdio.h>
4
5 int main()
6 {
7     /* initializing variables in declarations */
8     int passes = 0, failures = 0, student = 1, result;
9
10    /* process 10 students; counter-controlled loop */
11    while ( student <= 10 ) {
12        printf( "Enter result ( 1=pass,2=fail ): " );
13        scanf( "%d", &result );
14
15        if ( result == 1 ) /* if/else nested in while */
16            passes = passes + 1;
17        else
18            failures = failures + 1;
19
20        student = student + 1;
21    }
22
23    printf( "Passed %d\n", passes );
24    printf( "Failed %d\n", failures );
25
26    if ( passes > 8 )
27        printf( "Più di 8\n" );
28
29    return 0;    /* successful termination */
30 }

```

Esecuzione



```

Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4

```

Struttura iterativa `while`

```
while (condizione)
{
    Istruzione1;
    :
    IstruzioneN;
}
```

Il programmatore specifica un'azione che deve essere ripetuta finché una condizione rimane **vera**

– Pseudocodice:

Finché ci sono ancora oggetti nella mia lista della spesa

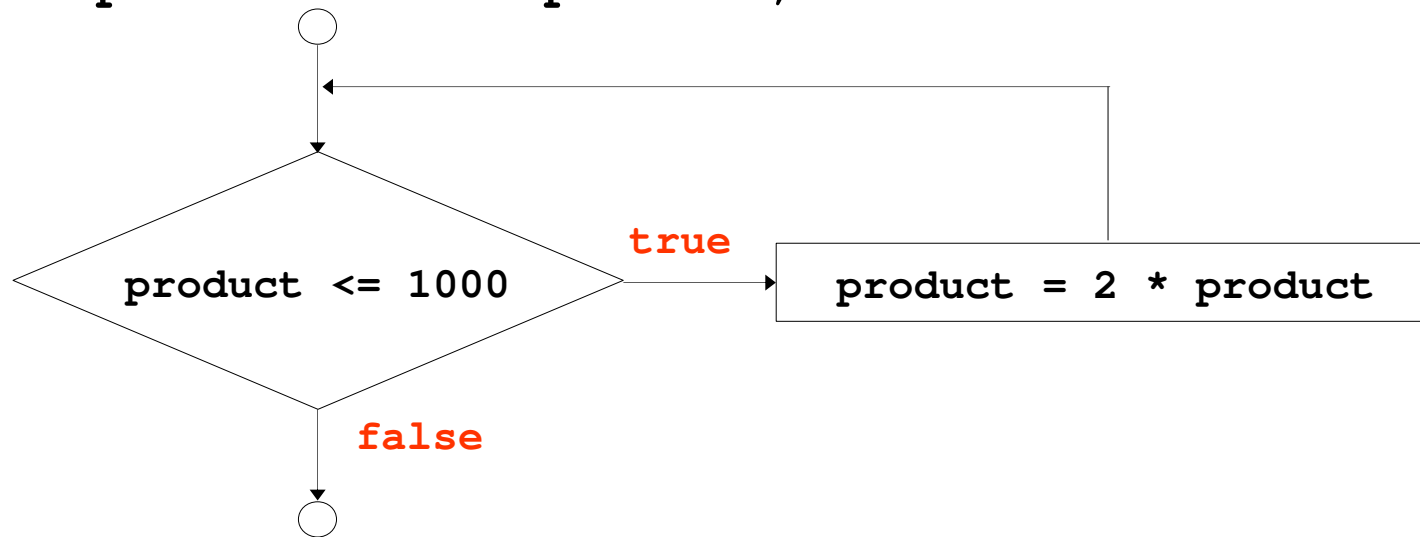
Acquista il prossimo oggetto e cancellalo dalla lista

- **while**: ciclo ripetuto finché la condizione rimane **true**

...continua

- Esempio

```
int product = 2;  
while ( product <= 1000 )  
    product = 2 * product;
```



Struttura iterativa **for**

Viene utilizzato per ripetere un'istruzione un numero specificato di volte.

```
for(inizializzazione;espressione;incremento) istruzione;
```

Per ripetere un blocco di istruzioni la forma generale è:

```
for(inizializzazione;espressione;incremento)
{
    istruzione1;
    .
    .
    istruzionen;
}
```

...continua

- Formato della struttura **for**

for (*inizializzazione*; *cond. continuaz.*; *incremento*)
istruzione

Esempio: Visualizzare gli interi da 1 a 10

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


...continua

- L'*inizializzazione*: di solito un'istruzione di assegnamento che imposta il valore iniziale della *variabile di controllo del ciclo*, che ha la funzione di contatore.
- Per *espressione* si intende un' espressione condizionale che determina se il ciclo continuerà (se vera) oppure no (se falsa).
- L'*incremento* definisce la quantità di cui la variabile di controllo cambierà ogni volta che il ciclo viene iterato.
- devono essere separate da punti e virgola.

...continua

- I cicli `for` possono essere riscritti come cicli `while`
`for (espressione1; espressione2; espressione3)`

`istruzione;`

`espressione1;`

`while (espressione2)`

`{`

`istruzione;`

`espressione3;`

`}`

- L'**inizializzazione** e l'**incremento** possono essere liste di istruzioni separate da virgola

```
for (int i = 0, j = 0; j + i <= 10; j++, i++)  
    printf( "%d\n", j + i );
```

Note

- L'inizializzazione, la condizione di continuazione del ciclo e l'incremento possono contenere delle espressioni aritmetiche.

Se **x = 2** e **y = 10**, l'istruzione

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

è equivalente all'istruzione

```
for ( j = 2; j <= 80; j += 5 )
```

- "Incremento" può essere negativo (decremento)
- Se la condizione di continuazione del ciclo è inizialmente **false**
 - il corpo del ciclo **for** non sarà eseguito
 - l'esecuzione procederà con l'istruzione successiva alla struttura **for**

ESERCIZIO

Programma che calcola la somma dei numeri pari
da 2 a 100

ESERCIZIO

Programma che calcola la somma dei numeri pari da 2 a 100

```
1  /*
2     Summation with for */
3  #include <stdio.h>
4
5  int main()
6  {
7     int sum = 0, number;
8
9     for ( number = 2; number <= 100; number += 2 )
10         sum += number;
11
12     printf( "Sum is %d\n", sum );
13
14     return 0;
15 }
```

Esecuzione



Sum is 2550

Struttura iterativa **do/while**

- Simile alla struttura **while**
- La condizione di continuazione del ciclo è controllata *dopo* che è stato eseguito il corpo del ciclo
- Tutte le azioni sono eseguite almeno una volta

- Struttura

```
do {  
    iterazione  
} while ( condizione );
```

...continua

- Esempio : Visualizza gli interi da 1 a 10.

(sia `counter = 1`)

```
do {  
    printf( "%d  ", counter );  
} while (++counter <= 10);
```

Equivalente a:

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

Esercizio

```
/* stampa i caratteri ascii dal 33 al 127 */
```

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

```
main()
```

```
{
```

```
    unsigned char ch=33;
```

```
    /* il ciclo do-while viene sempre eseguito almeno una  
    volta. Se ch=128 il ciclo verrebbe eseguito sempre e  
    comunque una volta soltanto perche' la condizione ch<128  
    viene controllata a fine ciclo */
```

```
    do{
```

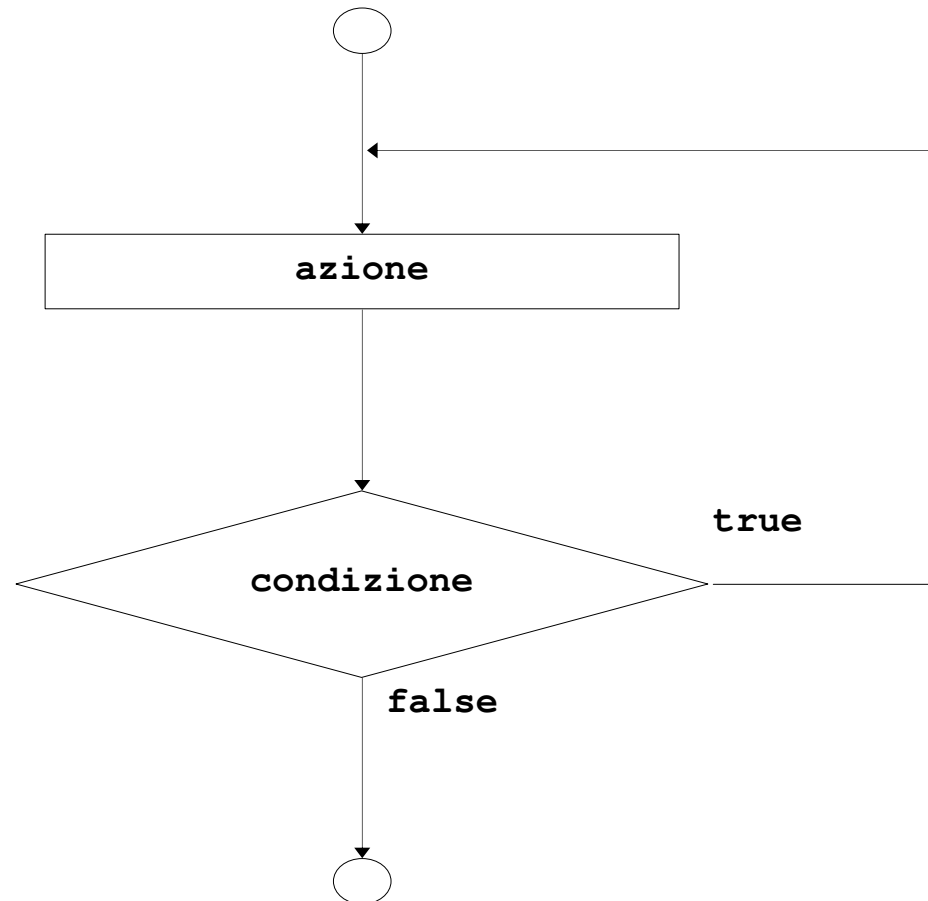
```
        printf("[ %c ]\n",ch);
```

```
        ch++;
```

```
    }while(ch<128);
```

```
}
```


...continua



Istruzioni di salto

Fanno parte di questa classe di istruzioni:

- break
- continue
- goto

Istruzione di salto **break**

- Causa l'uscita immediata da un ciclo **while**, **for**, **do/while** o **switch**
- L'esecuzione del programma continua con la prima istruzione successiva alla struttura iterativa
- Usi comuni dell'istruzione **break**
 - Uscita “anticipata” da un ciclo
 - Ignorare la parte rimanente di una struttura **switch**

Esempio di salto break

```
/* stampa i caratteri ASCII dal 33 al 127 */
#include <stdio.h>

main()
{
    unsigned char i; /* definisce i come char (-128 ≤ i ≤
127) */

    for(i=33; i<128; i++)
    { /* questo ciclo for continuerebbe all'infinito a
causa del "wrapping around" della variabile i */
        printf("[ %c ]\n", i);
        if(i==127) break; /* si forza il ciclo a
terminare quando il valore di i e' 127 */
    }
}
```

Istruzione di salto **continue**

- Fa ignorare le istruzioni rimanenti nel corpo di un ciclo **while**, **for** o **do/while**
 - Viene eseguita l'iterazione successiva del ciclo
- **while** e **do/while**
 - La condizione di continuazione del ciclo è valutata immediatamente **dopo** l'esecuzione dell'istruzione **continue**
- Ciclo **for**
 - Viene eseguita l'istruzione di incremento, poi è valutata la condizione di continuazione del ciclo

Esempio di salto continue

```
/* stampa i numeri pari da 0 a 20 */
#include <stdio.h>

main()
{
    unsigned int i;

    for(i=0; i<=20; i++) /* iteriamo da 0 a 20 compresi */
    {
        if(i%2) continue; /* se il numero e' dispari
        salta alla successiva iterazione del ciclo */
        printf(" %d \n", i); /* stampa i numeri */
    }
}
```

Istruzione `goto`

Permette di effettuare un salto **incondizionato** all'interno del codice. E' un'istruzione **DA NON USARE MAI PER NESSUN MOTIVO**.

Introduce disordine nel programma rendendolo illeggibile, un buon programmatore non utilizza **MAI** l'istruzione `goto`. In caso di necessita si possono utilizzare cicli `while` con istruzioni di salto `break` e/o `continue` ottenendo gli stessi risultati. `goto` non e' un elemento necessario per rendere completo un linguaggio di programmazione.

Nota

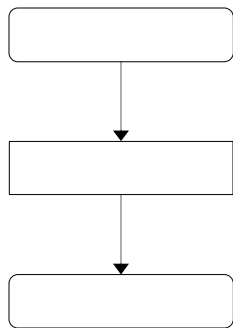
- Secondo il teorema di Jacopini-Boehm, qualsiasi algoritmo è riscrivibile utilizzando solo sequenza, selezione e iterazione, cioè eliminando i salti indietro e avanti:
 - Salto verso un'istruzione *precedente*:
 - Riscrivendo le istruzioni necessarie
 - Salto verso un'istruzione *successiva*:
 - Utilizzando una **variabile di appoggio** cui si assegna valori diversi per il flusso normale e per quello saltante
 - **Controllando il valore** della variabile di appoggio

Programmazione strutturata: riepilogo

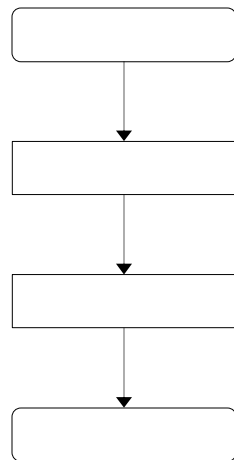
- Programmazione strutturata
 - Programmi più semplici (rispetto a quelli non strutturati) da comprendere, da collaudare, mettere a punto, modificare e anche da dimostrare corretti
- Regole per la formazione di programmi strutturati
 - Usare solo strutture di controllo “single-entry/single-exit”
 - Regole:
 - 1) Iniziare con il “diagramma di flusso elementare”.
 - 2) Ogni rettangolo (azione) può essere sostituito da due rettangoli (azioni) in sequenza.
 - 3) Ogni rettangolo (azione) può essere sostituito da una qualsiasi struttura di controllo (sequenza, **if**, **if/else**, **switch**, **while**, **do/while** o **for**).
 - 4) Le regole 2 e 3 possono essere applicate ripetutamente e in qualsiasi ordine.

...continua

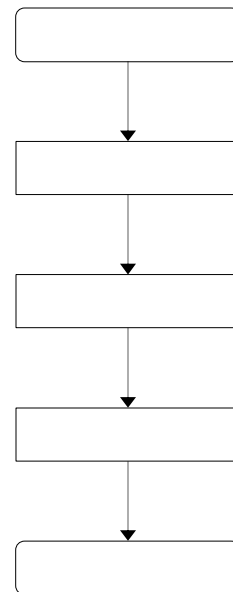
Regola 1 - Iniziare con il diagramma di flusso elementare



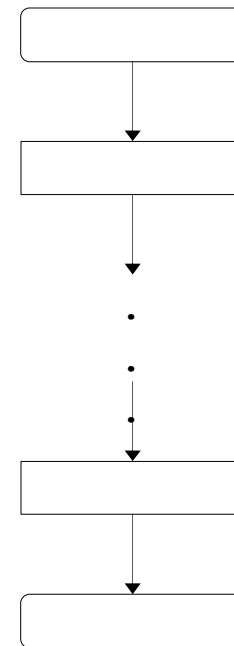
Regola 2 ↗



Regola 2 ↗

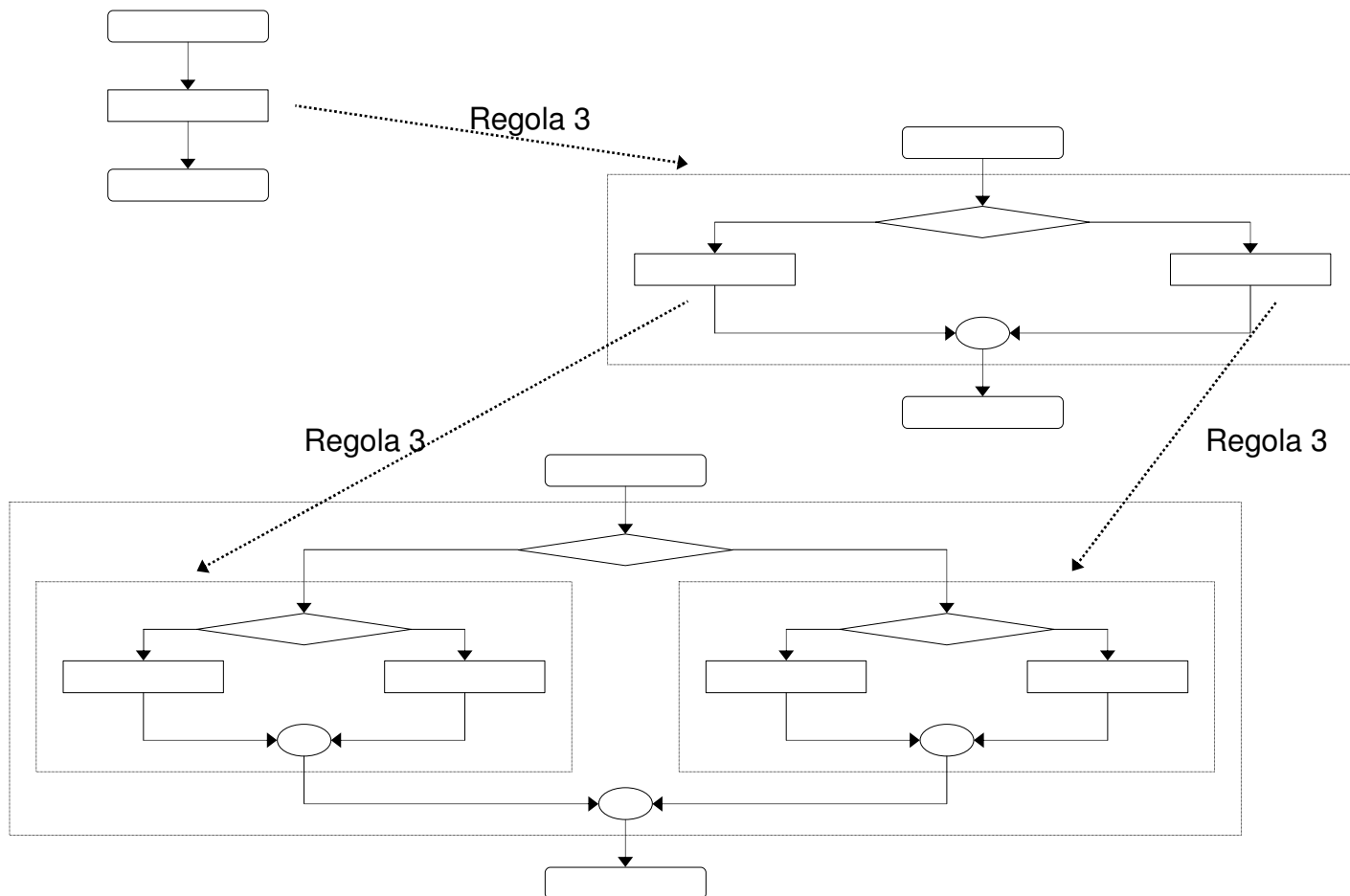


Regola 2 ↗



...continua

Regola 3 - Sostituire un rettangolo con una struttura di controllo

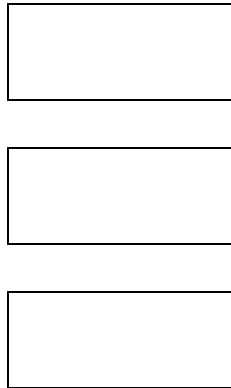


...continua

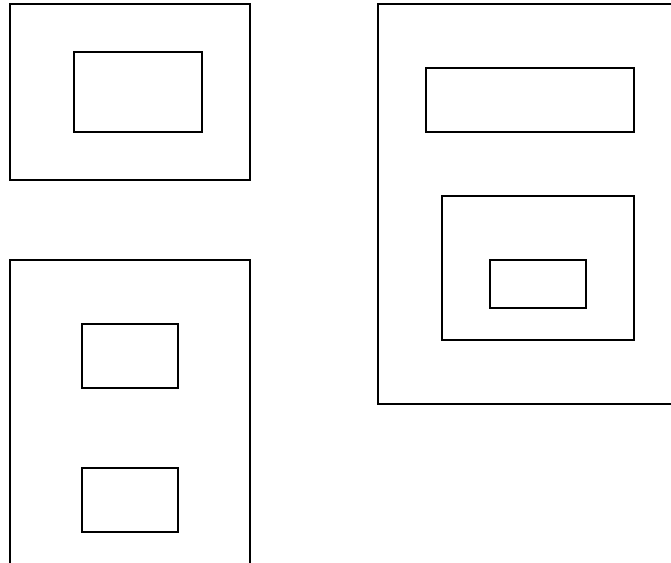
- Tutti i programmi possono essere costruiti con 3 blocchi
 - Sequenza -(banale)
 - Selezione - **if**, **if/else**, o **switch**
 - Iterazione - **while**, **do/while**, o **for**
 - Ogni selezione può essere tradotta in una istruzione **if**, ogni iterazione può essere tradotta in una istruzione **while**
- I programmi sono ridotti a
 - Sequenza
 - struttura **if** (selezione)
 - struttura **while** (iterazione)
 - Le strutture di controllo possono essere combinate solo in due modi: annidamento (regola3) e accatastamento (regola 2)

...continua

Blocchi
accatastati



Blocchi
nidificati



Esercizi

1. Scrivere un programma che stampa un rettangolo di 6 righe e 10 colonne la cui cornice sia costituita da caratteri asterisco e la parte interna da caratteri Q .
2. Scrivere un programma che calcola lo zero della funzione $f(x)=2x^3-4x+1$ nell'intervallo $[a,b]$, usando il metodo seguente:
 - Nell'ipotesi che $f(a)f(b)<0$:
 - Sia $m=(a+b)/2$;
 - si continua la ricerca in $[a,m]$, se $f(a)f(m)<0$;
 - si continua la ricerca in $[m,b]$, se $f(b)f(m)<0$;