

LA PROGRAMMAZIONE

- Abbiamo visto come le informazione vengono codificate per potere essere elaborate dal computer.
- Ora passeremo a vedere gli strumenti che abbiamo a disposizione per programmare il computer, per fare in modo, cioè, che il computer elabori le informazione che gli forniamo in modo utile per noi.

UN PO' DI STORIA - GLI ANNI '40

- All'inizio, negli anni '40, l'unico metodo per programmare era il **linguaggio macchina**.
- Il lavoro del programmatore consisteva nel settare ogni singolo bit a 1 o 0 su enormi computer che occupavano stanze intere e pesavano decine di tonnellate.
- I monitor non esistevano; i dati e i programmi si fornivano al computer su schede perforate e il computer mandava i risultati su telescriventi.
- In questo modo gli scienziati riuscirono in pochi mesi a completare i calcoli per costruire la prima bomba atomica, calcoli che se fatti a mano, con calcolatrici meccaniche avrebbero richiesto anni.

GLI ANNI '50

- Viene progettato il **FORtrAN** (FORmula trANslator), il cui utilizzo era ed è prettamente quello di svolgere in maniera automatica calcoli matematici e scientifici,
- Viene progettato l'**ALGOL** (ALGOritmic Language), altro linguaggio per applicazioni scientifiche sviluppato da Backus (l'inventore del FORtrAN) e da Naur.
- Backus e Naur mettono a punto un metodo per rappresentare le regole dei vari linguaggi di programmazione che stavano nascendo.

GLI ANNI '60

- Nel **1960** venne presentato il **COBOL** (COmmon Business Oriented Language), ideato per applicazioni nei campi dell'amministrazione e del commercio, per l'organizzazione dei dati e la manipolazione dei file.
- Nel **1964** fa la sua comparsa il **BASIC**, il linguaggio di programmazione per i principianti, che ha come caratteristica fondamentale quella di essere molto semplice e, infatti, diventa in pochi anni uno dei linguaggi più utilizzati al mondo.

GLI ANNI '70

- Intorno al 1970, però, Niklus Wirth propone il **PASCAL** per andare incontro alle esigenze di apprendimento dei neo-programmatori, introducendo però la possibilità di creare programmi più leggeri e comprensibili di quelli sviluppati in basic.
- Pochi anni più tardi fa la sua comparsa il **C**.
- C e Pascal segnano una svolta importante in quanto sono linguaggi strutturati: struttura dei dati, struttura del codice.

GLI ANNI '80

- Negli anni ottanta comincia ad affermarsi la Object Oriented Programming; la programmazione orientata agli oggetti basata sul nuovo concetto di Classe.
- Il primo linguaggio a proporla ai programmatori professionisti è il C++ .
- C e Pascal segnano una svolta importante in quanto sono linguaggi strutturati: struttura dei dati, struttura del codice.

JAVA

- È il primo linguaggio progettato appositamente per la programmazione orientata agli oggetti (non è cioè l'adattamento di un linguaggio preesistente).
- Il programma ottenuto non è un programma destinato a *girare* in un ambiente specifico (Windows o Macintosh o Linux) ma gira su un computer virtuale, la *Java Virtual Machine*. Basterà installare la versione specifica della *Java Virtual Machine* per il sistema operativo specifico e lo stesso programma potrà girare in ambienti completamente diversi.

LA PANORAMA ATTUALE

- L'enorme diffusione dei personal computer ha influito anche sul mercato dei linguaggi di programmazione, una volta sicuramente un mercato marginale.
- I produttori di software hanno tentato di assecondare questo processo cercando di offrire prodotti sempre più semplici da usare (Visual Basic, Visual C ++, Delphi, Visual Java, ecc. sono prodotti che rendono enormemente più semplice di un tempo sviluppare programmi nei moderni ambienti grafici a finestre
- Gli strumenti per lo sviluppo di contenuti multimediali on-line e off-line (pagine web, animazioni interattive, ecc.) si sono dotati di linguaggi di programmazione sempre più potenti.

LA PANORAMA ATTUALE

- Oggi il mercato offre molti prodotti.
- I linguaggi di programmazione, per loro natura, possono fare benissimo alcune cose e male altre.
- Per scegliere cosa usare bisogna prima decidere cosa si vuole produrre, qual è il nostro obiettivo.
- Il nostro interesse è soprattutto rivolto allo sviluppo di contenuti multimediali. Può essere, comunque interessante dare un rapido sguardo agli strumenti che abbiamo a disposizioni per realizzare progetti orientati ad altri fini.

LA PANORAMA ATTUALE

- Per **Applicazioni Matematiche o di Ricerca** i linguaggi più adatti sono ancora i *vecchi Fortran e Algol*.
- Per i **grandi progetti software** (sviluppo di sistemi operativi, applicativi complessi, applicazioni lato server) normalmente viene impiegato il **C** o il **C++**.
- Per lo sviluppo di **programmi gestionali** in ambiente Windows lo strumento più utilizzato è **Visual Basic** prodotto da Microsoft. Un'ottima alternativa al Visual Basic è, da qualche anno **Delphi** un prodotto Borland basato sul Pascal che consente anche il porting dei programmi tra l'ambiente Windows e l'ambiente Unix/Linux grazie a *Kilyx* versione Unix di Delphi. Negli ambienti diversi da Windows si utilizzano C/C++ e ambienti di sviluppo dedicati (i cosiddetti **CASE**) basati su C/C++.

APPLICAZIONI MULTIMEDIALI

- Dal punto vista del nostro corso meritano una particolare attenzione gli strumenti per lo sviluppo di applicazioni multimediali.
- Possiamo distinguere due tipi principali di applicazioni multimediali: quelle **on-line** (pagine Web) e quelle **off-line**.

Sviluppo di Pagine Web

- L'attività di programmazione entra a due livelli nello sviluppo di contenuti da distribuire via Internet:
- **La formattazione delle pagine**
- **L'inserimento di oggetti programmabili**

HTML (HYPERTEXT MARKUP LANGUAGE)

- Descrive come una pagina web debba essere mostrata da un browser.
- È un linguaggio a marcatori: tutte le istruzioni proprie del linguaggio sono inserite tra due segni specifici (nel caso di HTML tra “<” e “>”) e costituiscono i cosiddetti **tag**.
- I browser cercano di interpretare i **tag** come istruzioni mentre il resto viene mostrato come testo.
- I **tag** che il browser non riesce ad interpretare vengono semplicemente ignorati.

HTML ESEMPIO DI CODICE...

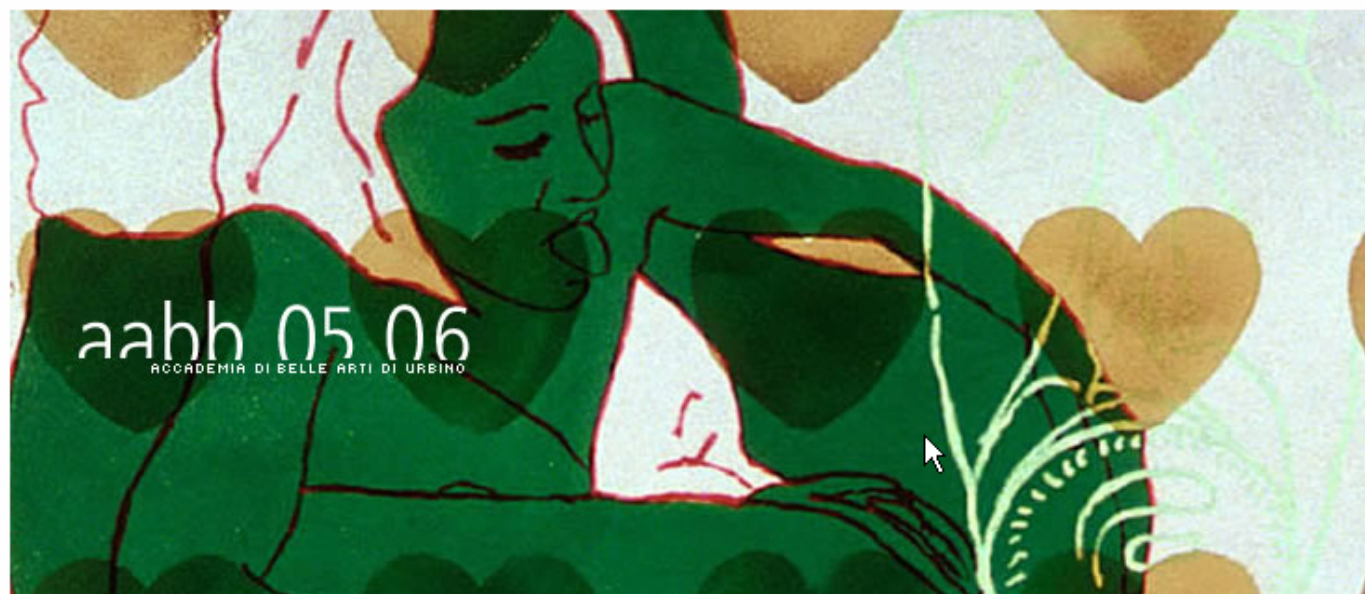
```
<body onLoad="MM_preloadImages('images/nm_r1_c1_f2.gif','images/nm_r1_c2_f2.gif','images/
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td align="left" valign="top" bgcolor="#2F5564"><br />
      <table width="753" height="301" border="0" align="center" cellpadding="10" cellspac
      <tr>
        <td><table width="474" height="272" border="0" align="center" cellpadding="0" cel
          <tr>
            <td align="center" valign="middle">
        </table></td>
      </tr>
    </table>
    <table width="757" border="0" align="center" cellpadding="0" cellspacing="0">
      <tr>
        <td><a href="accademia/" onMouseout="MM_swapImgRestore()" onMouseover="
"MM_swapImage('nm_r1_c1','images/nm_r1_c1_f2.gif',1);"></a></td>
        <td><a href="servizi/" onMouseout="MM_swapImgRestore()" onMouseover="MM_swapImage
"></a></td>
        <td><a href="didattica" onMouseout="MM_swapImgRestore()" onMouseover="
"MM_swapImage('nm_r1_c3','images/nm_r1_c3_f2.gif',1);"><img src="images/nm_r1_c3.gif"
```


aahh 07 08

ACCADEMIA DI BELLE ARTI DI URBINO

SISTEMI E LINGUAGGI DI PROGRAMMAZIONE EVOLUTI

...E RISULTATO



L'ACCADEMIA | SERVIZI | DIDATTICA | COBASLID

aahh news

20/10/2006

A.A. 2006/2007- COBASLID- CALENDARIO DELLA II^

OGGETTI PROGRAMMABILI

- Specifici **tag** consentono di inserire nella pagine web vari tipi di componenti programmabili. I più importanti sono:
- **Script**
- **Applet**
- L'accoppiata **Object - Embed**

SCRIPT

- Il tag SCRIPT consente di inserire dei nuclei di codice che il browser è in grado di eseguire.
- I linguaggi a disposizione dello sviluppatore sono due: **VBSCRIPT** (che deriva dal Visual Basic e viene riconosciuto però solo da Internet Explorer) e **JAVASCRIPT** che deriva invece da JAVA ed è riconosciuto più o meno da tutti i browser.

SCRIPT ESEMPIO

```
<script language="JavaScript" type="text/JavaScript">
<!--
function MM_preloadImages() { //v3.0
    var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
        var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
            if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
    }

function MM_swapImgRestore() { //v3.0
    var i,x,a=document.MM_sr; for(i=0;a&&i<a.length&&(x=a[i])&&x.oSrc;i++) x.src=x.oSrc;
    }

function MM_findObj(n, d) { //v4.01
    var p,i,x;  if(!d) d=document; if((p=n.indexOf("?"))>0&&parent.frames.length) {
        d=parent.frames[n.substring(p+1)].document; n=n.substring(0,p);}
    if(!(x=d[n])&&d.all) x=d.all[n]; for (i=0;!x&&i<d.forms.length;i++) x=d.forms[i][n];
    for(i=0;!x&&d.layers&&i<d.layers.length;i++) x=MM_findObj(n,d.layers[i].document);
    if(!x && d.getElementById) x=d.getElementById(n); return x;
    }

function MM_swapImage() { //v3.0
    var i,j=0,x,a=MM_swapImage.arguments; document.MM_sr=new Array; for(i=0;i<(a.length-2);i+=3)
        if ((x=MM_findObj(a[i]))!=null){document.MM_sr[j++]=x; if(!x.oSrc) x.oSrc=x.src; x.
src=a[i+2];}
    }
//-->
</script>
```

APPLET

- Il tag **APPLET** consente di inserire in una pagina web un programma scritto in JAVA.
- Il programma dovrà rispettare alcune condizioni di sicurezza.
- Per girare il programma utilizza la **Java Virtual Machine** che deve essere installata sul computer.

APPLET ESEMPIO

```
<applet code="CellularAutomata3" archive="media/CellularAutomata3.jar" width="200" height="200">  
</applet>
```

OBJECT-EMBED

- Il tag *OBJECT* (Internet Explorer su piattaforma Windows) insieme al tag *EMBED* (altri browser e Internet Explorer su Macintosh) consentono di inserire nelle pagine web oggetti di varia natura gestiti da estensioni dei browser o (in Windows) dai oggetti gestiti direttamente dal sistema operativo (i cosiddetti ActiveX).
- Qui segnaliamo due oggetti/plugin che negli ultimi anno hanno avuto una grande diffusione: **SHOCKWAVE FLASH** e **SHOCKWAVE DIRECTOR** entrambi orientati alla sviluppo di applicazioni multimediali interattive, entrambi dotati di un potente linguaggio di programmazione.

OBJECT-EMBED ESEMPIO

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"  
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=7,0,0,0"  
  WIDTH="100%" HEIGHT="100%" ALIGN="">  
  <PARAM NAME=movie VALUE="Loader.swf">  
  <PARAM NAME=menu VALUE=true>  
  <PARAM NAME=quality VALUE=high>  
  <PARAM NAME=bgcolor VALUE=#FFFFFF>  
  <EMBED src="Loader.swf" menu=false quality=high bgcolor=#FFFFFF WIDTH="100%" HEIGHT="100%"  
ALIGN="" TYPE="application/x-shockwave-flash" PLUGINSPAGE =  
"http://www.macromedia.com/go/getflashplayer"></EMBED>  
</OBJECT>
```

I

PAGINE STATICHE E DINAMICHE

- I tipi di pagine web di cui abbiamo fin qui parlato sono pagine statiche. Pagine, cioè, che vengono inviate al browser dal server web esattamente come sono state composte.
- Molti siti web oggi usano, invece, pagine dinamiche, pagine, cioè, il cui contenuto viene composto dal server al momento della richiesta sulla base dei parametri che gli vengono passati dal browser e del contenuto di database che il server può consultare.

STRUMENTI DI PROGRAMMAZIONE

- Questo tipo di programmazione, un tempo riservato agli specialisti, è oggi alla portata di qualsiasi programmatore.
- Gli strumenti più diffusi sono **PERL** e **PHP** per i server web che girano in ambiente Unix e Linux e **ASP** e la sua più recente evoluzione **ASP.NET** per Internet Information Server (il server web Microsoft).

APPLICAZIONI MULTIMEDIALI OFF-LINE

- Per quanto riguarda la Creazione di Giochi gli sviluppatori tendono a usare (per ottenere la massima efficienza) strumenti il più vicino possibile al linguaggio dei processori quindi **C++** e **Assembler**.
- Per altre applicazioni (didattiche, pubblicitarie, ecc) e giochi non troppo complessi vengono usati **FLASH**, **DIRECTOR** e **JAVA**.

ALGORITMI

FUNZIONAMENTO DEL CALCOLATORE

- Un computer per poter risolvere un qualsiasi problema ha bisogno di qualcosa che gli indichi esattamente cosa fare passo dopo passo
- Un computer si limita ad eseguire delle istruzioni “*macchina*” per l'esecuzione delle quali e' stato progettato
- Un'istruzione “*macchina*” e' un'istruzione “*primitiva*” comprensibile e direttamente eseguibile dal calcolatore senza bisogno di essere interpretata

SOFTWARE

- Qualsiasi compito per poter essere eseguito da un calcolatore deve essere tradotto in un insieme di istruzioni *macchina*
- Un software non e' altro che un insieme di istruzioni eseguibili dal calcolatore che nel loro insieme risolvono un problema o eseguono un determinato compito
- E' compito del programmatore "tradurre" un problema utilizzando solamente il set di istruzioni "primitive" comprensibili al calcolatore

ALGORITMO

Si può definire come un *procedimento* che consente di **ottenere** un dato **risultato** eseguendo, in un determinato ordine, un insieme di **passi semplici** corrispondenti ad azioni scelte solitamente da un insieme finito.

PROBLEMA

Problema:

Un contadino deve fare attraversare il fiume ad una capra, un cavolo, e un lupo, avendo a disposizione una barca in cui può trasportare solo uno dei tre alla volta. Se incustoditi, la capra mangerebbe il cavolo e il lupo mangerebbe la capra

SOLUZIONE

Algoritmo che descrive la soluzione:

1. Porta la capra sull'altra sponda
2. Porta il cavolo sull'altra sponda
3. Riporta la capra sulla sponda di partenza
4. Porta il lupo sull'altra sponda
5. Porta la capra sull'altra sponda

PROPRIETÀ FONDAMENTALI DEGLI ALGORITMI

- *Non-ambiguità*: il procedimento deve essere interpretabile in modo univoco da chi lo deve eseguire (l'ordine di esecuzione dei passi deve essere non ambiguo)
- *Eseguibilità*: ogni istruzione dell'algoritmo deve poter essere eseguita senza ambiguità da un esecutore reale o ideale
- *Finitezza*: sia il numero di istruzioni che compongono l'algoritmo che il tempo di esecuzione dello stesso devono essere finiti

ALGORITMO BASE DEI CALCOLATORI

Finché non trovi un'istruzione di *halt* fai:

- Leggi un'istruzione dalla memoria
- Decodifica l'istruzione appena letta
- Esegui l'istruzione

RAPPRESENTAZIONE DI UN ALGORITMO

- Rappresentazione mediante **pseudolinguaggio** o **pseudocodice** che possiamo definire come un *linguaggio informale (ma non ambiguo) per la descrizione delle istruzioni*. Ogni riga rappresenta un'istruzione. Se non diversamente specificato, le righe si leggono dall'alto verso il basso.
- rappresentazione mediante **diagramma di flusso** (flowchart) che è una rappresentazione grafica in cui ogni istruzione è descritta all'interno di un riquadro e l'ordine di esecuzione delle istruzioni è indicato da frecce di flusso tra i riquadri.

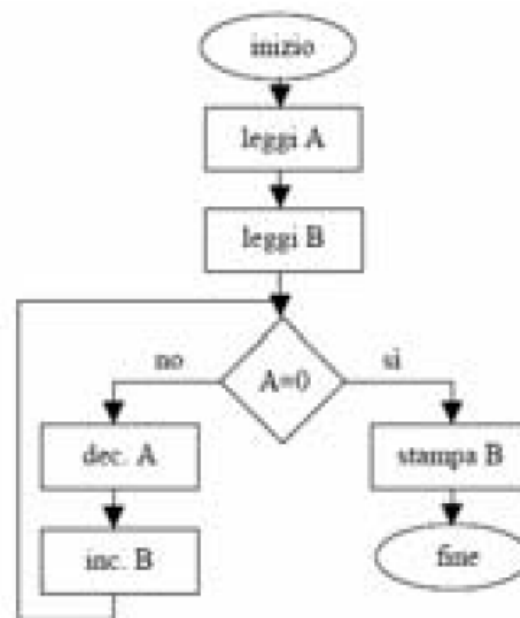
PROBLEMA:

sommare due numeri naturali utilizzando solo incrementi e decrementi

Pseudolinguaggio

1. Leggi A
2. Leggi B
3. Se $A=0$ vai all'istruzione 7
4. Decrementa A
5. Incrementa B
6. Vai all'istruzione 3
7. Stampa B

Diagramma di flusso



AZIONI PRIMITIVE

- La definizione di funzioni primitive permette di eliminare l'ambiguità
 - Es: "Andare a lezione può essere irritante"
 - "Le lezioni causano irritazione"
 - "Il tragitto per arrivare dove si svolgono le lezioni e' irritante"
- Il livello di astrazione delle primitive deve essere scelto in funzione dell'esecutore
- L'insieme delle primitive più un set di regole per metterle insieme costituisce un *Linguaggio di programmazione*

DEFINIZIONE DI UNO PSEUDOLINGUAGGIO

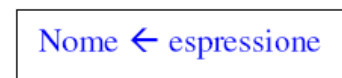
Istruzioni di inizio e di fine	start, end	Sono i punti d' ingresso e di uscita del flusso di esecuzione
Istruzione di assegnamento	nome ← espressione	L' esecuzione di un' istruzione di assegnamento avviene in due fasi: la valutazione dell' espressione (utilizzando i valori correnti delle eventuali variabili che in essa compaiono) e l' aggiornamento del valore della variabile a sinistra del segno
Scelta tra due possibili attività	if (condizione) then (attività 1) else (attività 2)	Biforcazione del flusso in due percorsi alternativi (mutuamente esclusivi) la cui esecuzione è condizionata al verificarsi di una condizione. L' esecuzione di un' istruzione condizionale comporta innanzitutto la valutazione della condizione, e quindi l' esecuzione delle istruzioni che compongono uno dei due cammini.
Strutture iterative	while (condizione) do (azione)	Esecuzione ripetuta di una o più istruzioni. La ripetizione dipende dall' esito di un controllo. È fondamentale che l' esito del controllo dipenda da variabili il cui valore può essere modificato dall'esecuzione delle istruzioni del ciclo. In caso contrario il ciclo verrebbe eseguito o mai (risultando inutile) o all' infinito (violando la definizione di algoritmo). Le variabili da cui dipende il controllo sono dette variabili di controllo del ciclo. Ogni ciclo è composto da 4 elementi: l' inizializzazione delle variabili di controllo, il controllo della condizione da cui dipende l' iterazione, il corpo del ciclo (sequenza delle istruzioni da eseguire ciclicamente) e l' aggiornamento delle variabili di controllo.
Istruzione di salto	goto (riga di destinazione)	L'istruzione di salto sposta il flusso di esecuzione in un particolare punto dell'algoritmo. La riga di destinazione rappresenta, appunto, il punto in cui l' esecuzione riprenderà dopo il goto.
Definizione di procedure	procedure nome (istruzione 1 istruzione 2 istruzione 3)	Unità di programma utilizzabili attraverso invocazione da un qualsiasi punto del codice. Tutti i linguaggi di programmazione utilizzano questa strategia per rendere più leggibile il codice. Sinonimi di procedura sono: funzione, metodo, subroutine, sottoprogramma ecc

DIAGRAMMA DI FLUSSO

Inizio e fine



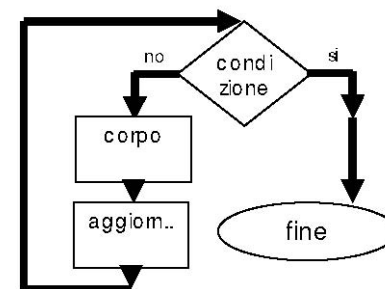
Assegnamento ed elaborazione



Scelta condizionata



Iterazione e salti



ESECUZIONE

Ecco come risulta l'algoritmo per sommare due numeri naturali utilizzando solo incrementi e decrementi:

1. **Start**
2. $A \leftarrow \text{input}$
3. $B \leftarrow \text{input}$
4. **If**($A=0$) **then** (
5. **goto**(9))
6. $A \leftarrow A-1$
7. $B \leftarrow B+1$
8. **goto**(4)
9. Stampa B
10. **end**

GRANDEZZE

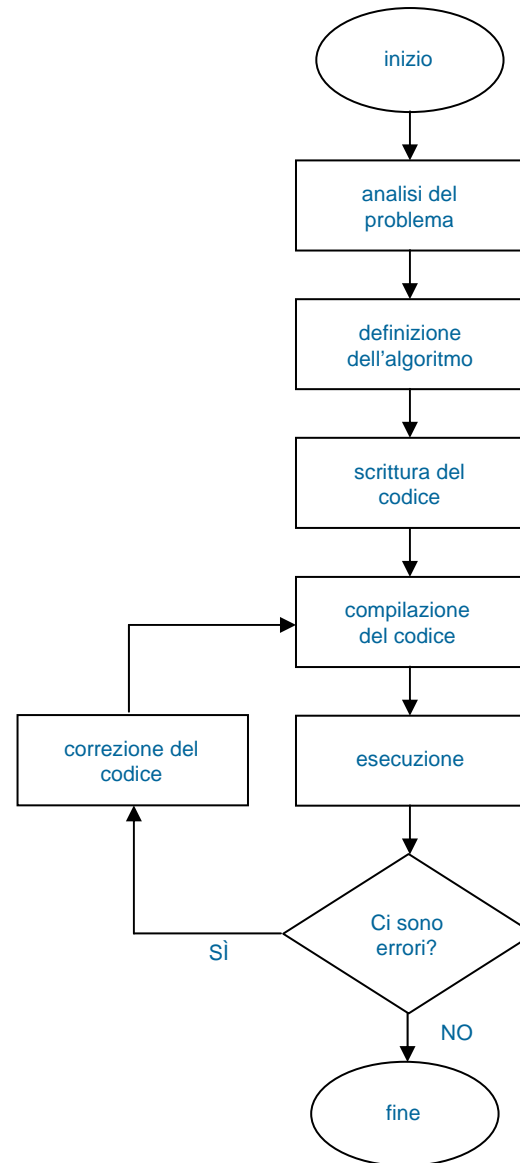
- *Costante*: quantità nota a priori il cui valore non dipende dai dati di ingresso e non cambia durante l'esecuzione (es: valore 0 nell'algoritmo precedente)
- *Variabile*: nome simbolico cui è assegnato un valore che può dipendere dai dati di ingresso e variare durante l'esecuzione (es: A e B)
- *Espressione*: sequenza di variabili, costanti o espressioni combinate tra loro mediante operatori (es: $a+b*4$)

COMPILAZIONE

Qualsiasi sia il linguaggio che abbiamo scelto il nostro codice dovrà essere tradotto, perché possa essere eseguito dal computer, in una serie di istruzioni che la CPU è in grado di interpretare ed eseguire. Questo processo, che chiameremo compilazione, a seconda dell'ambiente e del linguaggio scelto potrà avere diverse caratteristiche ma sarà comunque un passaggio necessario del nostro processo creativo.

IL PROCESSO

Se utilizziamo un diagramma di flusso per descrivere il processo di creazione di un'applicazione otterremo lo schema a fianco.



LA LEGGIBILITÀ DEL CODICE

Leggibilità

- Scrivere programmi *sensati* e *leggibili* è difficile, ma molto importante
- È essenziale per lavorare in gruppo
- Aiuto il debugging
- Aiuta a riutilizzare il codice e quindi ci risparmia fatica

Leggibilità significa:

- Progettare con chiarezza
- Scrivere codice con chiarezza

Progettare con chiarezza

- Dedicare il tempo necessario alla progettazione della nostra applicazione non è tempo perso.
- Ci aiuterà a chiarire la logica e la sintassi del nostro lavoro.
- Più avremo sviluppato l'algoritmo che sta alla base della nostra applicazione più il nostro programma sarà comprensibile

Scrivere con chiarezza

- La chiarezza della scrittura si ottiene attraverso due *tecniche* :
- L'***indentazione***: inserire spazi o tabulazioni per mettere subito in evidenza le gerarchie sintattiche del codice.
- I ***commenti***: inserire note e spiegazione nel corpo del codice.

Identazione: un esempio

- Prendiamo in esame questo brano di codice HTML :

```
<table> <tr> <td>a</td> <td>b</td> <td>c</td>
</tr> <tr> <td> <table> <tr> <td>a1</td> </tr>
<tr> <td>a2</td> </tr> </table> </td> <td>b1</td>

<td>c1</td> </tr> </table>
```

Identazione: un esempio

- E confrontiamolo con questo:

```
<table>
  <tr>
    <td>a</td>
    <td>b</td>
    <td>c</td>
  </tr>
  <tr>
    <td>
      <table>
        <tr>
          <td>a1</td>
        </tr>
        <tr>
          <td>a2</td>
        </tr>
      </table>
    </td>
    <td>b1</td>
    <td>c1</td>
  </tr>
</table>
```

Identazione

- Si tratta della stessa tabella, ma nel primo caso ci risulta molto difficile capire come è organizzata. Nel secondo la gerarchia degli elementi risulta molto più chiara.

Identazione

- L'identazione non ha nessun effetto sulla compilazione del programma
- Serve solo a rendere il nostro lavoro più leggibile.

Inserire commenti

- Rende il codice leggibile anche ad altri
- Quando decidiamo di apportare modifiche a cose che abbiamo scritto ci rende la vita più facile.

Delimitatori

- Delimitatori di riga: tutto ciò che segue il contrassegno di commento fino alla fine della riga non viene compilato.
Esempi:

// #

- Delimitatori di inizio e fine: tutto ciò compreso tra il contrassegno di inizio e il contrassegno di fine non viene compilato.

/* ... */ <!-- ... -->

INTRODUZIONE ALLA LOGICA

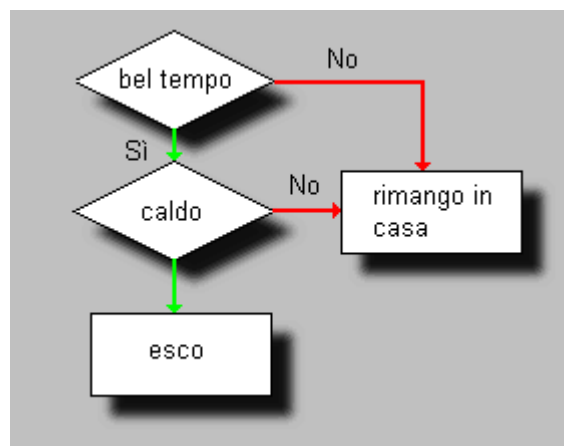
Introduzione

- Nella lezione precedente abbiamo visto che qualsiasi processo logico può essere ricondotto ad una sequenza di eventi elementari (**algoritmo**)
- Che tale sequenza può essere rappresentata con un diagramma di flusso (il quale a sua volta è facilmente traducibile in un particolare programma comprensibile dall'elaboratore).

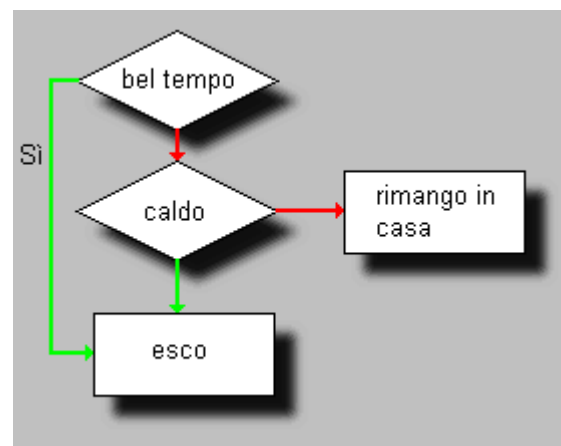
Problema

- Prendiamo questi due enunciati:
 - esco se è bel tempo ed è caldo
 - esco se è bel tempo o se è caldo

Diagrammi di flusso



esco se è bel tempo ed è caldo



esco se è bel tempo o è caldo

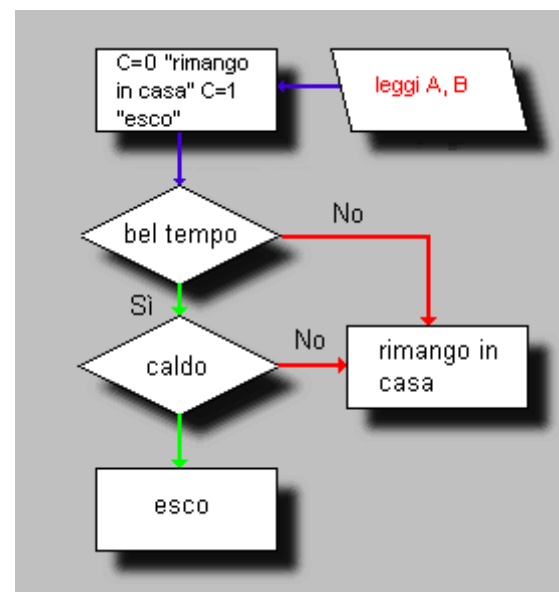
Gli operatori logici

- Come secondo passo, si tratta di convertire i diagramma di flusso in un linguaggio comprensibile dall'elaboratore. Ciò si ottiene con i cosiddetti operatori logici elementari.

operazione	istruzione	porta logica
controllo	IF (se...)	
azione	THEN (allora...)	
congiunzione	AND (...e...)	AND
separazione	OR (... oppure...)	OR
negazione	NOT (negazione)	NOT

Formalizzazione

- $A = 1$ corrisponde all'evento "bel tempo"
- $B = 1$ corrisponde all'evento "caldo"
- $C = 1$ corrisponde all'azione "esco"
- $A = 0$ corrisponde all'evento "non bel tempo"
- $B = 0$ corrisponde all'evento "non caldo"
- $C = 0$ corrisponde all'azione "resto in casa"



con queste condizioni, il primo diagramma di flusso risulta così formalizzato:

IF A AND B THEN C

Gli operatori logici

AND – Congiunzione

falso AND falso	risultato falso
falso AND vero	risultato falso
vero AND falso	risultato falso
vero AND vero	risultato vero

NOT - Negazione

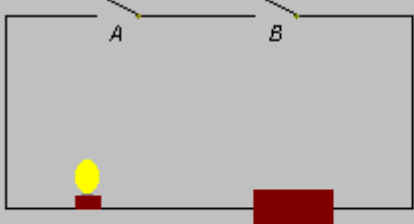
NOT falso	risultato vero
NOT vero	risultato falso

OR - Disgiunzione

falso OR falso	risultato falso
falso OR vero	risultato vero
vero OR falso	risultato vero
vero AND vero	risultato vero

Gli operatori logici

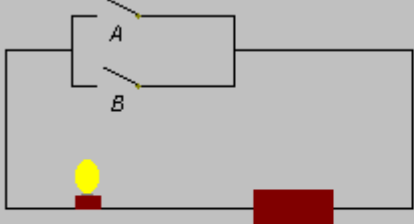
rappresentazione con i circuiti elettrici



se gli interruttori *A* e *B* sono entrambi abbassati, il circuito è chiuso e la lampadina si accende

<i>A</i>	<i>B</i>	<i>C</i>
0	0	0
0	1	0
1	1	1
1	0	0

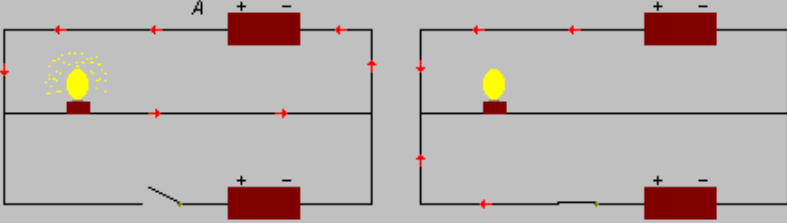
tavola di veridicità per AND



se l'interruttore *A* o *B* è abbassato, il circuito è chiuso e la lampadina si accende.

<i>A</i>	<i>B</i>	<i>C</i>
0	0	0
0	1	1
1	0	1
1	1	1

tavola di veridicità per OR



se l'interruttore è aperto, la batteria *A* alimenta il circuito e la lampadina è accesa

se l'interruttore è abbassato, entrambe le batterie alimentano il circuito e la lampadina è spenta

<i>A</i>	<i>C</i>
1	0
0	1

tavola di veridicità per NOT

GLI ELEMENTI DEL LINGUAGGIO

Introduzione

- Nella lezione precedente abbiamo visto che nell'esecuzione di un **algoritmo** entrano in gioco tre tipi di grandezze:
 - Costanti
 - Variabili
 - Espressioni
- In più abbiamo visto che vengono definite un numero finito di azioni elementari

Introduzione

- *Costante*: quantità nota a priori il cui valore non dipende dai dati di ingresso e non cambia durante l'esecuzione (es: valore 0 nell'algoritmo precedente)
- *Variabile*: nome simbolico cui è assegnato un valore che può dipendere dai dati di ingresso e variare durante l'esecuzione (es: A e B)
- *Espressione*: sequenza di variabili, costanti o espressioni combinate tra loro mediante operatori (es: $a+b*4$)

Sintassi

- Ora prenderemo in esame questi elementi in termini grammaticali.
- Come punto di riferimento prenderemo il linguaggio con cui avremo a che fare in questo corso: **ActionScript** il linguaggio utilizzato da FLASH e che deriva da **JavaScript**, ma la maggior parte delle cose di cui parleremo varranno, in generale, per tutti i linguaggi.

Elementi di un linguaggio

- Le unità semantiche di base di un linguaggio sono:
 - *Parole chiave*
 - *Operatori e separatori*
 - *Letterali* (o *Costanti*)
 - *Nomi* (o *Identificatori*)

Parole chiave

- Le parole chiave sono i termini (composti da caratteri alfanumerici), riservati al linguaggio di programmazione.
- Il creatore del linguaggio di programmazione stabilisce a priori quali termini riservare e quale sarà la loro funzione, il compito del programmatore è quello di impararle ed usarle in maniera appropriata.
- L'uso improprio di tali termini viene generalmente rilevato durante la fase di compilazione di un programma.

Parole chiave in ActionScript 2.0

add, and, break, case, catch, class,
continue, default, delete, do,
dynamic, else, eq, extends, finally,
for, function, ge, get, gt, if,
ifFrameLoaded, implements, import,
in, instanceof, interface, intrinsic,
le, lt, ne, new, not, on,
onClipEvent, or, private, public,
return, set, static, switch,
tellTarget, this, throw, try, typeof,
var, void, while, with

Parole chiave in JAVA

`abstract, assert, boolean, break,
byte, case, catch, char, class,
const, continue, default, do, double,
else, enum, extends, final, finally,
float, for, goto, if, implements,
import, instanceof, int, interface,
long, native, new, package, private,
protected, public, return, short,
static, strictfp, super, switch,
synchronized, this, throw, throws,
transient, try, void, volatile, while`

Operatori

- Gli operatori sono token composti di uno o più caratteri speciali che servono a controllare il flusso delle operazioni che dobbiamo eseguire o a costruire **espressioni**
- Operatori usati sia in **ActionScript** che in **JAVA**:

++ ! != !== % %= & && &= () -
* *= , . ? : / // /* /= [] ^ ^=
{ } | || |= ~ + += < << <<=
<= <> = -= == === > >= >>
>>= >>> >>>=

Proprietà degli operatori

- **Posizione**

Un operatore si dice **prefisso** se viene posto prima degli operandi, **postfisso** se viene posto dopo e **infisso** se si trova tra gli operandi.

- **Arietà**

L'arietà è il numero di argomenti di un operatore: 1, 2 o 3.

Proprietà degli operatori

- **Precedenza (o Priorità)**

Indica l'ordine con il quale verranno eseguite le operazioni. Ad esempio in **$4+7*5$** verrà prima eseguita la moltiplicazione poi l'addizione.

- **Associtività**

Un operatore può essere associativo a **sinistra** oppure associativo a **destra**. Indica quale operazione viene fatta prima a parità di priorità.

Separatori

- I separatori sono simboli di interpunzione che permettono di chiudere un'istruzione o di raggruppare degli elementi.
- Il separatore principale è lo *spazio* che separa i *termini* tra di loro quando non ci sono altri separatori. Gli altri separatori sono:

() { } , ;

Letterali (o costanti)

- Le *costanti* (o letterali) sono quantità note a priori il cui valore non dipende dai dati d'ingresso e non cambia durante l'esecuzione del programma.
- Le costanti possono essere numeri, stringe di caratteri, valori di verità (true, false), array, oggetti, ecc.

Costanti numeriche

- Le **costanti numeriche** iniziano sempre con un carattere numerico: il fatto che un *token* inizi con un numero basterà ad indicare al compilatore che si tratta di una costante numerica. Se il compilatore non potrà valutare quel *token* come numero segnerà un errore.
- Il segno che separa la parte intera di un numero dalla parte decimale è il punto.
- È possibile inserire numeri in formato decimale, binario, ottale o esadecimale.
- Per segnalare al compilatore che un numero non è decimale si fa precedere il numero da un prefisso. Per i numeri esadecimali questo prefisso è **0x**.
- Gli altri *termini* (*parole chiave* e *nomi*) NON possono iniziare con un numero.

Esempi di costanti numeriche

1

2433

1000000000

3.14

.333333333333

0.5

2345.675

0xFF0088

0x5500ff

0xff.00aa

Costanti stringa

- Una stringa è una sequenza di caratteri **UNICODE** ed permette di rappresentare testi. Un *costante* stringa è una sequenza (anche vuota) di caratteri racchiusi tra apici singoli o apici doppi.
- Per inserire ritorni a capo, tabulazioni, particolari caratteri o informazioni di formattazione si utilizzano speciali sequenze di caratteri dette *sequenze di escape*. Una sequenza di escape è formata da un carattere preceduto dal simbolo “\” (*backslash*). La sequenza di escape inserisce un carattere che non sarebbe altrimenti rappresentabile in un letterale stringa.

Principali sequenze di escape

- `\n` nuova riga;
- `\r` ritorno a capo;
- `\t` tabulazione orizzontale;
- `\'` apostrofo (o apice singolo);
- `\"` doppio apice;
- `\\` backslash(essendo un carattere speciale deve essere inserito con una sequenza di escape).

Esempi di costanti stringa

```
// Stringa racchiusa da apici singoli
'Ciao a tutti'

// Stringa racchiusa tra apici doppi
"Ciao"

/* La sequenza di escape risolve l'ambiguità tra l'apostrofo
inserito nella stringa e gli apici singoli che la
racchiudono */
'Questo è l\'esempio corretto'

/* In questo caso non c'è ambiguità perché la stringa è
racchiusa tra doppi apici */
"Anche questo è l'esempio corretto"

/* Per inserire un ritorno a capo si usano le sequenze
di escape */
"Questa è una stringa valida\r\n due righe"
```


Costanti booleane

- Le costanti booleane, poiché rappresentano valori logici, possono avere solo due valori: vero (rappresentato dal letterale *true*) e falso (rappresentato dal letterale *false*).

Costanti di tipo Array

- Il letterale **Array** è costituito da una serie di elementi separati da virgole compresa tra due parentesi quadre:

```
// array che contiene i mesi dell'anno  
[ "January", "February", "March", "April" ] ;
```

Costanti di tipo Object

- Il letterale ***Object*** è invece compreso tra parentesi graffe ed è costituito da una serie di coppie “**chiave:valore**” separate da virgole:

```
//record di una rubrica telefonica in formato Object  
{name: "Irving", age: 32, phone: "555-1234"};
```

Altre costanti

- I linguaggi normalmente definiscono anche altre costanti. Alcune rappresentano valori speciali che non possono essere rappresentati che da un valore simbolico (come abbiamo visto per **true** e **false**) altre sono valori rappresentati da un letterale per comodità, ma possono essere anche rappresentate, a seconda dei casi, come stringe o come valori numerici.
- Per quanto riguarda il primo gruppo *ActionScript* definisce **Infinity**, **-Infinity**, **undefined**, **null** e **NaN**.

Identificatori (o Nomi)

- Un identificatore è un nome definito dal programmatore. Gli identificatori si usano per dare nomi alle variabili, alle funzioni e ai tipi di dati (o classi).

Regole per gli Identificatori

- il primo carattere deve essere una lettera o il simbolo “_” (ricordiamo che nel caso la prima lettera fosse un numero il compilatore tenterebbe di interpretare il nome come costante numerica);
- i caratteri successivi possono essere lettere, numeri o “_”.
- Gli identificatori non possono inoltre coincidere con le parole riservate del linguaggio.

VARIABILI E TIPI DI DATI

Variabili

Pensiamo a quando salviamo un numero di telefono del nostro amico Mario sul cellulare; se vogliamo chiamare il nostro amico, basterà inserire il suo nome (Mario, nome della **variabile**) ed il cellulare comporrà automaticamente il numero di telefono (**valore** della variabile). Se per qualche ragione Mario cambierà numero di telefono, modificherò il contenuto della mia rubrica (cambierò il valore della variabile). In questa maniera senza modificare le mie abitudini (inserirò sempre Mario) il mio cellulare comporrà il nuovo numero.



Variabili

- Una variabile è composta da due elementi: il suo **nome** e il suo **valore**; come ho visto nell'esempio del cellulare in un programma posso usare i nomi delle variabili al posto dei valori che rappresentano.
- Ho la possibilità di usare simboli mnemonici al posto di numeri e stringhe di grande entità o difficili da ricordare.
- Ho la possibilità di usare il nome della variabile al posto del suo valore per eseguirvi sopra delle operazioni, e generalizzare l'elaborazione.

Esempio

- Questo un programma scritto in pseudolinguaggio calcola il quadrato di un numero inserito dall'utente e lo mostra sul video.

```
scrivi sullo schermo "Ciao Inserisci un numero";
```

```
A = -numero inserito da tastiera-;
```

```
B = A * A;
```

```
scrivi sullo schermo "Il quadrato di " A " è " B;
```

```
/* A e B sono variabili */
```

Tipi

- Le variabili possono contenere vari tipi di dati. Un tipo di dato o, più semplicemente un *tipo* definisce come le informazioni verranno codificate per essere elaborate o semplicemente memorizzate.
- La dichiarazione è un comando che comunica al compilatore che un determinato nome è il nome di una variabile e che quella variabile conterrà un determinato tipo di dati.

Tipi primitivi

- I tipi primitivi sono i tipi quelli fissati dalle specifiche del linguaggio.
- Posso manipolare i tipi primitivi utilizzando gli operatori.
- Le variabili contengono completamente un valore di un tipo primitivo.
- In ActionScript i tipi primitivi sono **Number**, **Boolean** e **String**.

Boolean

- Il tipo di dati Boolean può avere due valori: **true** e **false**. Nessun altro valore è consentito per le variabili di questo tipo.
- Il valore predefinito di una variabile booleana dichiarata ma non inizializzata è **false**.

Number

- Questo tipo di dati può rappresentare numeri interi, numeri interi senza segno e numeri a virgola mobile.
- Per memorizzare un numero a virgola mobile, includere una punto decimale nel numero; senza il punto il numero viene memorizzato come numero intero e quindi i risultati delle operazioni vengono arrotondate al numero intero più vicino.

String

- Il tipo di dati String rappresenta una sequenza di caratteri a 16 bit che può includere lettere, numeri e segni di punteggiatura.
- Le stringhe vengono memorizzate come caratteri Unicode, utilizzando il formato UTF-16.
- Un'operazione su un valore String restituisce una nuova istanza della stringa.

Dichiarazioni di variabili

- Per dichiarare una variabile in ActionScript si usa la parola riservata **var** seguita dal nome della variabile, dai due punti e dal tipo:

```
var pippo:String;
```

- Opzionalmente si può assegnare un valore alla variabile all'atto della dichiarazione (inizializzazione):

```
var pippo:String = "Hello World";
```


Dichiarazione di variabili di tipi primitivi

```
//dichiarazioni di variabili in actionscript
/* a può contenere solo un numero, s una
   stringa k true o false */
var a:Number;
var s:String;
var k:Boolean;
/* per b e messaggio oltre a dichiarare il
   tipo viene Impostato un valore iniziale */
var b:Number = 1;
var messaggio:String = "Ciao a tutti";
```

Tipi derivati o complessi

- Per rappresentare dati complessi (ad esempio un elenco di valori, i dati che compongono un indirizzo, una data, ecc.) ho a disposizione alcuni tipi complessi che il linguaggio mi offre oppure ne posso creare ad hoc.
- Per i tipi complessi la variabile contiene il *puntatore* cioè il numero della casella, l'indirizzo in cui il dato è memorizzato sul computer.
- Nei linguaggio orientati agli oggetti il concetto di **tipo** e il concetto di **classe** coincidono.

PROGRAMMAZIONE CONDIZIONALE

Sintassi dell'istruzione if

- L'istruzione if consente di tradurre in un linguaggio di programmazione i ragionamenti fatti parlando della logica Booleana.
- L'istruzione if può avere due forme:
 - if** (espressione) blocco di istruzioni
 - if** (espressione) blocco di istruzioni **else** blocco di istruzioni
- L'espressione che compare dopo la parola chiave **if** deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'**else** (senza condizioni) in posizione finale.

Esempio in pseudocodice

```
intero A = 50;
scrivi sullo schermo "Inserisci un numero";
intero B = -numero inserito da tastiera-;
if (B minore di A) {
    scrivi sullo schermo "Il numero inserito è minore di
                          cinquanta";
} else if (B maggiore di A) {
    scrivi sullo schermo "Il numero inserito è maggiore di
                          cinquanta";
} else if (B uguale a A) {
    scrivi sullo schermo "Il numero inserito è cinquanta";
} else {
    //poiché B non è minore, né maggiore né uguale a A
    // A non è un numero
    scrivi sullo schermo "Inserisci un numero";
}
```

Esempio in ActionScript

```
var A:Number = 50;
var B:Number = input_txt.text;
if (B < A) {
    messaggio_txt.text = "Il numero inserito è minore di
                           cinquanta";
} else if (B > A) {
    messaggio_txt.text = "Il numero inserito è maggiore di
                           cinquanta";
} else if (B == A)
    messaggio_txt.text = "Il numero inserito è cinquanta";
} else {    //B non è un numero
    messaggio_txt.text = "Inserisci un numero!!!";
}
```