# Università degli Studi di Ferrara

Corso di Laurea in Chimica - A.A. 2018 - 2019

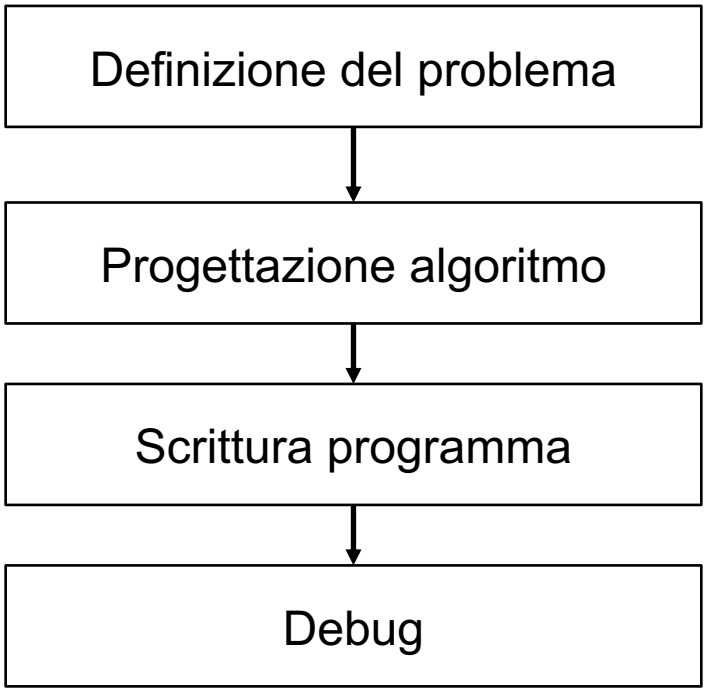
## Programmazione
## Lezione 4 – il Linguaggio di programmazione

Docente: Lorenzo Caruso – lorenzo.caruso@unife.it

# Nelle lezioni precedenti

- Problemi e Calcolatori: definire i passaggi risolutivi di un problema prima in linguaggio naturale e poi in una forma più vicina al calcolatore
- Algoritmi: definizione della risoluzione di un problema per passi successivi
- Costruire un Algoritmo: scegliere la rappresentazione di un algoritmo
- Programmazione strutturata: rappresentare l'algoritmo per mezzo di sequenze/selezioni/iterazioni
- Dal Problema al Programma: codificare l'algoritmo
- Linguaggi di programmazione: metodi di codifica di un algoritmo in modo che diventi eseguibile in un calcolatore

# Realizzare un programma: le fasi



# Codifica di un programma

- Editing
- Compilazione
  - Precompilazione
  - Compilazione
  - Assemblaggio
  - Link
- Esecuzione
  - Caricamento in memoria
  - Esecuzione

# Codifica di un programma

La codifica di un programma  comprende 4 fasi (inclusa l'esecuzione)

- Editing
- Compilazione
- Link
- Esecuzione

# Editing

È la fase in cui il programmatore scrive il codice necessario a codificare l'algoritmo.

La scrittura del codice, in genere avviene attraverso l'uso di editor specifici che aiutano il programmatore nella stesura del codice evidenziando con colori diversi le parole chiave del linguaggio.

Il file sul quale viene salvato il codice sorgente deve avere estensione adeguata

# Editing - Convenzioni

Ogni linguaggio di programmazione presenta delle **Convenzioni Stilistiche,** che è importante seguire:

- Posizione delle parentesi { } (per quelli che li prevedono)
- Stile dei nomi delle funzioni (o metodi)

TUTTI i linguaggi di programmazione prevedono una stessa “buona pratica”:

- Indentazione
- Commenti al codice

# Compilazione

Una volta terminata la scrittura del codice sorgente, il programmatore esegue la compilazione del codice sorgente per produrre il file eseguibile dal calcolatore.

La compilazione comprende diverse fasi:

- Compilazione
- Assemblaggio

# Compilazione

Il codice espanso, viene tradotto in codice assembly.
In questa fase vengono evidenziati gli errori di sintassi all'interno del codice prodotto.
Nell'ultima fase della compilazione, il codice scritto in linguaggio assembly, viene tradotto in una forma chiamata codice oggetto rilocabile

- Codice Oggetto: codice in linguaggio macchina nella maggior parte dei casi non eseguibile direttamente.
- Rilocabile: Non ancora allocato in locazioni fisiche della memoria centrale.

# Assemblaggio

Il codice assembly prodotto durante la fase di compilazione subisce un ulteriore trasformazione.
Il file viene assemblato e trasformato in codice macchina eseguibile dal calcolatore.

# Link

Nel caso il nostro codice sorgente sia diviso in più file, durante la fase di link, questi vengono uniti e viene incluso anche il codice delle librerie utilizzate.

Se il codice sorgente è incluso in un unico file, solitamente, questa fase è trasparente (cioè non si distingue questa fase dalla compilazione).

# Codifica di un Algoritmo

Lo scopo principale di scrivere programmi è quello di “far eseguire” gli algoritmi al calcolatore.

- Come un semplice algoritmo viene codificato ?

# Codifica di un Algoritmo

Riprendiamo il classico esempio di algoritmo per calcolare l'area di un rettangolo.
Per semplicità, per questo primo esempio faremo riferimento al calcolo dell'area di un rettangolo specifico.

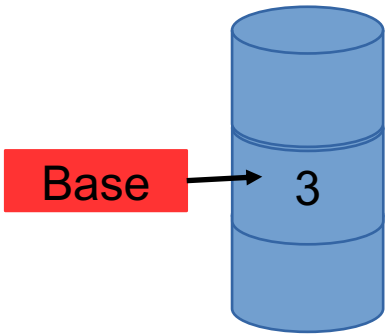
- Assegna a Base il valore 3;
- Assegna ad Altezza il valore 5;
- Assegna ad Area il valore Base * Altezza;
- Scrivi Area;

# Le Variabili

Le variabili sono delle porzioni di memoria a cui viene assegnato (per comodità) un nome.
Durante la vita del nostro programma alle variabili è possibile assegnare più valori e utilizzarle per svolgere i calcoli definiti dall'algoritmo.

Prendiamo ad esempio la prima riga del nostro algoritmo:

Assegna a Base il valore 3;




Base è il nome di una variabile che contiene il valore 3

# Variabili - Dichiarazione

Molti linguaggi sono tipizzati, per questo motivo, prima di utilizzare una variabile è necessario dichiarare che intendiamo utilizzarla attraverso un'operazione detta

**dichiarazione della variabile**

Con l'operazione di dichiarazione della variabile, definiamo il suo nome ed anche il suo tipo.

Nella maggior parte dei casi la sintassi per dichiarare una variabile è la seguente:

```
<tipo-variabile> <nome-variabile>;
```

# Variabili - Tipi di Dato

- Int (e.g. → 5)
- Float (e.g. → 3.4)
- Double (e.g. → 3.5678890)
- Char (e.g. → m)

Nella realtà queste dimensioni sono fortemente dipendenti dalla architettura, e sono definite da valori massimi e minimi per ogni tipo di dato.

# Variabili - MatLab

In Matlab non è necessario definire e dichiarare le variabili.
Tutte le variabili vengono trattate in doppia precisione (8 byte),
senza distinzione fra interi, reali e reali a doppia precisione

- tutte le variabili sono matrici
- non si dichiara il tipo di variabile

```
>> a=5            variabile scalare (1 × 1)
>> b=[4 6]              vettore riga (1 × 2)
>> c=[-5; 2]            vettore colonna (2 × 1)
>> d=[2 3; -1 7]        matrice quadrata (2 × 2)
```

# Variabili - La memoria

Una variabile rappresenta un'area di memoria alla quale il nostro programma accede per leggere o scrivere valori.
Il programma accede (in lettura e scrittura) alla cella desiderata per mezzo dell'indirizzo.

(per questo motivo la memoria accessibile a un programma si dice anche spazio di indirizzamento)

Ogni riferimento al nome della variabile è in realtà un riferimento al valore in essa contenuto.

# i Commenti

I commenti sono porzioni di testo che vengono ignorate dal compilatore in fase di compilazione.

I commenti sono importanti per descrivere cosa stiamo facendo all'interno del codice.

Inserire i commenti nel nostro codice è importante per spiegare ad altri e ricordare a noi stessi cosa stiamo facendo in quel punto del codice.

- Su una sola riga:

```
% Questo su una sola riga
```

- Su più righe:

```
% Questo commento
% è su più righe
```

# Assegnare un valore

Ad ogni variabile definita si può assegnare un valore (di un tipo coerente) con l'operatore di assegnamento (=)

Il valore può essere il risultato di una espressione:

```
area = base * altezza;
```

# Riassumendo

Formalmente, le variabili sono aree riservate di memoria identificate da un *nome* e da un *tipo*.

- **Nome** utilizzato per accedere alle variabili in lettura e scrittura

(lettera o underscore seguito da un numero qualsiasi di lettere, cifre o underscore)

- **Tipo** ne determina la dimensione in memoria, i valori ammissibili e le operazioni possibili

(es. double: Double-precision arrays, single: Single-precision arrays, int8/16/32/64 8/16/32/64-bit signed integer arrays, uint8/16/32/648/16/32/64-bit unsigned integer arrays, etc.)

➔ Il contenuto di una variabile in un dato momento è detto valore della variabile.

# La funzione fprintf()

| Codice di formato | Significato |
|---|---|
| %s | formato stringa |
| %d | formato decimale |
| %g | seleziona il formato per numeri interi, fixed point o esponenziali |
| %f | formato fixed point del numero (esempio 1343.675432) |
| %e | formato esponenziale del numero (esempio 1.34376e+003) |
| \n | inserisce carattere di ritorno a capo |
| \t | inserisce carattere di tabulazione |

Grazie per l’attenzione

# Università degli Studi di Ferrara

Corso di Laurea in Chimica - A.A. 2018 - 20189

## Programmazione
## Lezione 5 – Controllo di Flusso

Docente: Lorenzo Caruso – lorenzo.caruso@unife.it

# In questa lezione

- Costanti
- Operatori
- Operatori Logici
- Selezione
- Iterazione

# Ripasso: Variabili - Tipi di Dato

- Int (e.g. → 5)
- Float (e.g. → 3.4)
- Double (e.g. → 3.5678890)
- Char (e.g. → m)

Nella realtà queste dimensioni sono fortemente dipendenti dalla architettura

# Costanti

Una costante è del tutto simile ad una variabile, ovvero è uno spazio di memoria identificato da un tipo e da un nome, come però suggerisce il nome la costante è immutabile: il suo valore non cambia.

In Matlab sono a disposizione dell'utente alcune variabili speciali e costanti predefinite:
• ans è la variabile a cui viene assegnata la most recent ANSwer, ovvero il risultato di espressioni
valutate al prompt senza che siano precedute da un'istruzione di assegnazione;

• eps contiene l'epsilon di macchina; nel caso di arrotondamento la precisione di macchina `e 0.5*eps;

• realmax contiene il più grande numero di macchina positivo;

• realmin contiene il più piccolo numero di macchina positivo rappresentabile in forma normalizzata
(con denormalizzazione si possono rappresentare anche valori fino a realmin$*2^{-52}$);

• pi contiene il valore approssimato di π;

# Costanti

• i,j rappresentano entrambi l'unità immaginaria;

• inf rappresenta il valore speciale "infinito"; `e il risultato ad esempio della divisione 1/0;

• NaN rappresenta il valore speciale Not a Number; `e il risultato, ad esempio, della divisione 0/0.

N.B. Le costanti predefinite possono essere usate dall'utente come variabili, ignorando il loro valore e
significato, e usandole in una qualsiasi istruzione di assegnazione (ad esempio, le variabili i e j possono
essere tranquillamente usate in un ciclo for): il contenuto predefinito sar`a recuperato riavviando una
nuova sessione di lavoro.

# Gli operatori

In molti linguaggi vengono comunemente utilizzati simboli per richiamare alcune operazioni fondamentali, questi vengono detti operatori.

# Operatori

| Simbolo | Operazione |
|---|---|
| + | addizione |
| - | sottrazione |
| * | moltiplicazione |
| / | divisione intera |
| mod | modulo |

→ esempio operatori

# Operatore di incremento

Ricordiamo che:
**Gli accumulatori**, nel linguaggio della programmazione sono variabili nelle quali il nuovo valore non sostituisce il valore precedente, ma si somma (accumula) a quello già presente.

Se all'accumulatore viene "passata" la seguente sequenza:

1, 2, 3, 4

Allora l'accumulatore assumerà i seguenti valori* :

1, 3, 6, 10

Se il valore di incremento dell'accumulatore resta costante di una unità, allora otteniamo un **Contatore**

* Supponendo che il contatore sia inizializzato a 0

# Operatore di confronto

L’operatore == è l’operatore di **confronto**

- Restituisce il valore logico TRUE (1) se il valore dell’espressione che sta alla destra dell’operatore è uguale al valore dell’espressione che sta alla sua sinistra
- Restituisce il valore logico FALSE (0) se il valore dell’espressione che sta alla destra dell’operatore è diverso dal valore dell’espressione che sta alla sua sinistra

```
espressione1 == espressione2
```

```
x==y
```

```
(5+3)==(6+2)
```

# Operatori Logici

| Simbolo | Sintassi | Significato |
|---|---|---|
| ! | !a | NOT logico |
| > | a>b | maggiore |
| < | a<b | minore |
| >= | a>=b | maggiore/uguale |
| <= | a<=b | minore/uguale |
| == | a==b | uguale |
| != | a!=b | diverso |

# Prodotto logico (AND)

| a | b | a•b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Somma logica (OR)

| a | b | a+b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Selezione

Ricordiamo:
Secondo i dettami della programmazione strutturata un buon algoritmo viene espresso in forma di

sequenza – **selezione** – iterazione

**Selezione:** permette di scegliere fra due alternative la sequenza di esecuzione

# Selezione: costrutto if()

La selezione in viene implementata dal costrutto if()

Sintassi:

if <espressione logica è verificata>
end

La condizione deve essere una **espressione logica** (restituire vero o falso)
e.g.
a == b
a < b
b > 5 ecc..

# Selezione: costrutto if()

Nel caso volessimo distinguere due casi **mutuamente esclusivi** possiamo integrare il costrutto if() con il ramo **else**
Sintassi:

```
if <espressione logica e ’ verificata>
  <esegui processo 1>
  else
      <esegui processo 2>
end
```

In questo caso i due blocchi di istruzioni vengono eseguiti in maniera alternativa l’uno all’altro dipendentemente dalla condizione

# Selezione multipla - else if

Se volessimo implementare una scelta multipla possiamo sfruttare la possibilità di concatenare diversi costrutti “else if”

Sintassi

```
if <espressione logica 1 e ’ verificata>
        <esegui processo 1>
        else
            if <espressione logica 2 e ’ verificata>
            <esegui processo 2>
            else
                <esegui processo 3>
            end
end
```

# OPERATORI LOGICI

E’ possibile utilizzare gli operatori logici per mettere in relazione più condizioni

- && and
- || or
- ~ not
- & and (componente per componente)
- | or (componente per componente)

Operatore **AND** ( && )

Sintassi

if(condizione1 && condizione2)
<esegui processo 1>

Operatore **OR** ( || )

Sintassi

if(condizione1 || condizione2)
<esegui processo 1>

# Iterazione

Ricordiamo:
Secondo i dettami della programmazione strutturata un buon algoritmo viene espresso in forma di

sequenza – selezione – **iterazione**

**Iterazione:** permette di ripetere più volte la stessa sequenza fino al verificarsi di una condizione

# Iterazione: i cicli

Sono strutture fondamentali utilizzate **per ripetere istruzioni su insiemi di dati diversi.**

I linguaggi di programmazione mettono a disposizione vari tipi di cicli per adattarsi alle varie situazioni, in MATLAB esistono:

- ciclo “**while**”
- ciclo “**for**”

# Il Ciclo For

In Matlab la ripetizione di blocchi di istruzioni per un numero di volte specificato e in modo incondizionato viene eseguita tramite l'istruzione di ciclo FOR ...END la cui sintassi è:

```
for indice = espressione
        blocco di istruzioni
    end
```

Esempio:

```
for i=1:10;
  fprintf (i)
end
```

# Il Ciclo While

Se si ha la necessità di ripetere una o piu` istruzioni fintanto che una condizione saraà verificata non sapendo a priori il numero di ripetizioni, è necessario usare l'istruzione  WHILE ...END la cui sintassi e`:

```
while <<condizione 1>>
        istruzioni eseguite finché c <<condizione 1>> 1 è vera
end
```

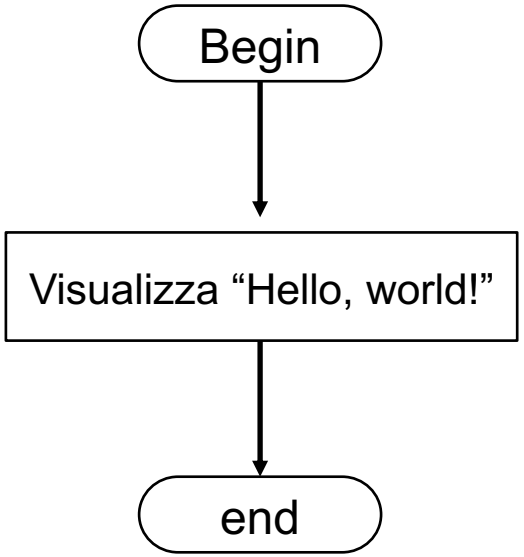
Esempio:

```
ii=1;
while ii<10
   disp(['Numero ',num2str(ii)])
   ii=ii+1;
end
```

1. Viene valutata la condizione
2. Se la condizione risulta essere vera, vengono eseguite le istruzioni contenute nel ciclo
3. Si ritorna al punto 1

L’istruzione viene eseguita se e finché la condizione risulta essere vera

# Hello, world!



Grazie per l’attenzione