

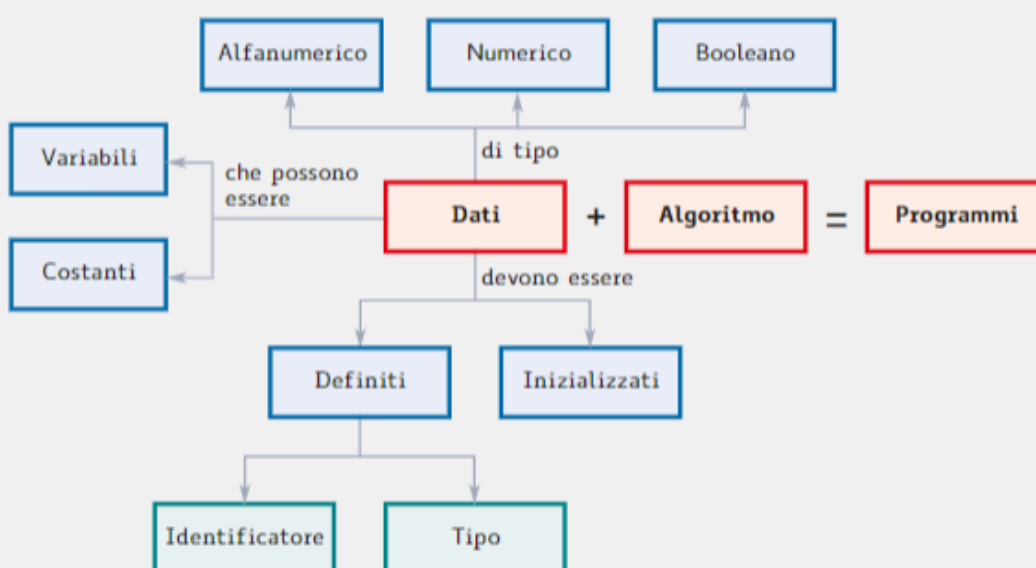
2

Il programma e le variabili

IN QUESTA LEZIONE IMPAREREMO...

- a definire e a utilizzare variabili e costanti
- a utilizzare l'istruzione di assegnazione

MAPPA CONCETTUALE



Niklaus Wirth

Già docente presso il Politecnico di Zurigo, Wirth è ricordato anche per aver definito, nel 1967, il linguaggio Pascal a cui fecero seguito il Modula-2 e Oberon, alla fine degli anni 80.



Struttura di un programma

Prima di affrontare la scrittura del nostro primo programma è doveroso citare il testo del prof. **Niklaus Wirth**, *Algorithms + Data Structures = Programs* (traduzione italiana: *Algoritmi + strutture di dati = programmi*), edito nel 1975, sul quale si sono formate intere generazioni di informatici.



Il libro è tutt'oggi utilizzato in quanto la sua proposta operativa, cioè la separazione tra strutture dati e parte algoritmica, rimane un classico dell'ingegneria del software.

In base al modello messo a punto da Wirth, un **programma** può essere visto come una **sequenza di istruzioni che operano su un insieme di dati**, cioè come “un manipolatore di dati”. Detto altrimenti, tutti i linguaggi di programmazione prevedono, oltre che a un insieme di istruzioni, la possibilità di memorizzare i dati nei propri programmi per poi elaborarli e comunicarli o farli acquisire dall’utente attraverso le **operazioni di I/O (input e output)**.



I dati e le variabili

Durante l'esecuzione di un algoritmo, le **istruzioni** eseguono **operazioni sui dati** a partire da una situazione iniziale (i **dati di partenza**): questi possono assumere dei valori temporanei durante le operazioni intermedie fino a che il programma arriva alla sua conclusione **producendo i risultati desiderati** (i **dati finali**), che devono essere comunicati all’utente.

! Durante l'esecuzione delle istruzioni è necessario **memorizzare** tutti i **valori che i dati assumono**.

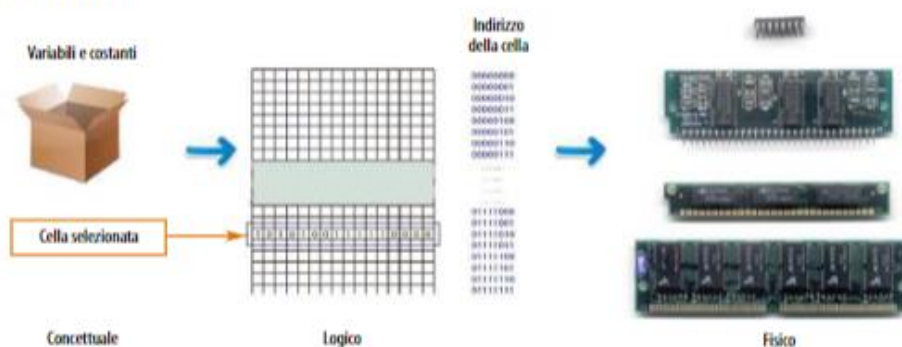
A tal fine, sono quindi necessari alcuni “**contenitori**” dove poter scrivere e leggere i valori di questi dati, collocare i valori risultanti dai calcoli e dalle elaborazioni intermedie e infine memorizzare i risultati finali: questi contenitori prendono il nome di **variabili**, con riferimento al concetto matematico di variabile, cioè di un “qualcosa” che **può assumere diversi valori** e, di conseguenza, cambiare.

! Ogni **variabile** può quindi essere considerata come il **contenitore di un dato**, cioè come una “scatola” dove viene inserito qualcosa che si può togliere o modificare in qualsiasi momento.

L’unica accortezza da adottare è assegnare un nome a ogni “scatola”, in modo da poterla distinguere dalle altre: tale nome viene scelto dal programmatore e, poiché serve per identificare univocamente le variabili, deve essere diverso per ciascuna di esse.

Una **variabile** è un’area della memoria dinamica del calcolatore (**RAM**) riservata per contenere un particolare dato; essa viene distinta dalle altre aree per mezzo di un nome (**identificatore**) che il programmatore stabilisce in modo univoco.

Se in memoria ci fossero due aree con lo stesso nome, esse non sarebbero distinguibili tra loro e quindi il calcolatore non saprebbe dove leggere o scrivere il valore che si vuole memorizzare nella variabile.



Identificatore della variabile

Nel progetto del programma il programmatore stabilisce "a piacere" che **identificatore** assegnare a una **variabile**: questo è costituito da un nome, che può essere formato da un insieme di caratteri alfanumerici minuscoli e maiuscoli e dal carattere "_" (underscore).

Non tutti i nomi sono ammessi come identificatori per le variabili: riportiamo in una tabella le limitazioni, alcune obbligatorie, altre di **buona norma**, con alcuni esempi per ciascun caso.

CARATTERISTICHE DELL'IDENTIFICATORE	ESEMPI
Non può iniziare con un numero	1nano, 2nano NON AMMESSO
Non può contenere spazi	valore medio, ora legale NON AMMESSO
Non può contenere caratteri speciali	valore%, anni/mes NON AMMESSO
Non può contenere lettere accentate	età, velocità, città NON AMMESSO
Deve indicare quello che contiene	temperatura, media, numero1
Può non avere senso compiuto	xyz, k123, kkzz
Può essere composto da due parole	valoreMedio, annoSolare, formaOcchi
Può essere lungo a piacere	variabiletemporanea, vt, vartempo
Può essere composto da un solo carattere	x, y, z, i, j
Può iniziare con l'underscore	_tempo, _var1, _pippo2

Anche se non è necessariamente obbligatorio, per attribuire il nome all'identificatore **sarebbe buona norma** seguire le regole riportate di seguito:

- scegliere un nome che indichi chiaramente il dato contenuto nella variabile;
- non utilizzare caratteri singoli, soprattutto quelli che potrebbero essere confusi per la loro somiglianza: per esempio "1" ("i" maiuscola) e "l" (uno), "O" ("o" maiuscola) e "0" (zero);
- non utilizzare nomi di fantasia come "pippo", "pluto", "paperino" ecc., che ostacolano la comprensibilità del codice;
- non utilizzare nomi troppo né lunghi né troppo corti.

Il linguaggio C++, come il linguaggio Java, differenzia per gli identificatori le lettere minuscole dalle lettere maiuscole (si dice che è **case sensitive**) mentre nel linguaggio Visual Basic questa differenziazione non viene effettuata.

Tipi di variabile

Il calcolatore può elaborare dati di natura diversa: per esempio, può effettuare calcoli matematici e quindi utilizzare numeri interi oppure decimali, memorizzare nomi e cognomi, quindi elaborare lettere dell'alfabeto, o ancora eseguire operazioni ed espressioni logiche e quindi lavorare su variabili che assumono solo due valori binari (per esempio VERO e FALSO).

In prima analisi, possiamo classificare le variabili in:

- **numeriche**: su tali variabili sarà possibile effettuare le operazioni algebriche; sono di due tipi:
 - **intero** (int): numeri senza virgola;
 - **reale** (float): numeri con la virgola.
- **alfanumeriche** (char): dette anche stringhe, sono variabili che contengono le lettere dell'alfabeto, che possono essere sia singole parole che intere frasi;
- **logiche** (bool): variabili che possono assumere sono i valori true e false (VERO o FALSO).

Prima di poter effettuare operazioni su di una **variabile** è necessario stabilire la sua **natura** (data type).

In un programma, bisogna per prima cosa individuare quali variabili sono necessarie e per ciascuna si devono effettuare due operazioni:

- definirne il **nome** (l'**identificatore**);
- definirne la **natura** (il **tipo**).

L'operazione che esegue queste due operazioni prende il nome di **dichiarazione di variabile**.

Con l'operazione di dichiarazione di una variabile si ottiene il triplice effetto di:

- indicare "alla macchina" di riservare in memoria un'area per la variabile;
- indicare "alla macchina" con che nome si vuole individuare quell'area di memoria;
- indicare "alla macchina" il tipo di dato che dovrà contenere e, quindi, di che dimensione dovrà essere.

AREA DIGITALE



Tipi e valori per le variabili

Nella tabella che segue sono riportati i tipi semplici di variabili più utilizzati.

Tipo	Parola Riservata C++	Parola Riservata Java	Valori ammessi
intero	int	int	interi positivi e negativi
reale	float	float	decimali positivi e negativi
alfanumerico	char	char	ABC, abc, 123, @, !, , (ASCII)
logico	bool	boolean	TRUE, FALSE



Le variabili (e, come vedremo in seguito, le **costanti**) sono, come si è detto, memorizzate nella memoria dinamica del calcolatore: quando viene a mancare l'alimentazione, cioè quando si spegne il calcolatore, viene perso tutto il contenuto della memoria RAM e, quindi, anche i valori di tutte le variabili.

La seguente tabella riporta alcuni esempi di definizione di variabile in C++ e in Java.

Linguaggio C++/C#	Linguaggio Java
<pre>int variabile1; int num1, num2; char inizialeDelCognome; char rosso, verde; bool avanti, finito; float reale;</pre>	<pre>int variabile1; int num1, num2; char inizialeDelCognome; char rosso, verde; boolean avanti, finito; float reale;</pre>

Assegnare un valore a una variabile

Dopo aver definito la variabile scegliendone il nome (identificatore) e il tipo, per poterla utilizzare è necessario assegnarle un **valore iniziale**: tale operazione prende il nome di **inizializzazione**.

Senza l'operazione di **inizializzazione** il valore contenuto nella variabile appena definita non è noto.

Generalmente, con l'inizializzazione si effettua l'azzeramento della variabile, in modo da essere sicuri che essa non contenga valori indesiderati.

L'**istruzione** che permette di inserire un valore in una variabile prende il nome di **assegnazione** (o assegnamento di un valore a una variabile).



Assegnare un valore a una variabile significa memorizzare un dato nella cella di memoria RAM corrispondente all'identificatore della variabile. L'operazione di assegnazione viene indicata nella pseudocodifica con il simbolo ← (freccia).

Vediamo alcuni esempi di casi generali che ritroviamo in ogni programma.

Inizializzazione (o azzeramento) di una variabile

Si è detto che l'operazione di inizializzazione è necessaria per completare la definizione di una variabile; essa è quindi l'operazione che viene eseguita per prima.

La sua struttura, nella forma generale, è la seguente.



Lo schema deve essere letto da destra verso sinistra: "il valore viene assegnato alla variabile".

Vediamo nel dettaglio le istruzioni per inizializzare ciascuno dei quattro tipi di variabile descritti.

1)

```
int variabile1; // definizione
variabile1 ← 0; // inizializzazione
```

che si legge, da destra verso sinistra: "il valore 0 viene assegnato alla variabile intera **variabile1**".

2)

```
float variabile2; // definizione
variabile2 ← 0.0; // inizializzazione
```

che si legge: "il valore 0 viene assegnato alla variabile **variabile2** di tipo float".

3)

```
char conferma; // definizione
conferma ← "S"; // inizializzazione
```

che si legge: "il valore 'S' viene assegnato alla variabile carattere **conferma**".

4)

```
bool finito; // definizione
finito ← FALSE; // inizializzazione
```

che si legge: "il valore FALSE viene assegnato alla variabile booleana **finito**".

Assegnazione di un valore a una variabile

Un valore viene assegnato a una variabile nel modo seguente.

```
variabl ← 10; // assegnazione
num1 ← 7.5;
```

Successivamente, il valore contenuto nelle variabili può essere modificato.

```
variabl ← 20;
num1 ← 1.5;
```

Il valore 20 viene memorizzato nella cella di memoria identificata con `variabl` in sostituzione del precedente valore 10, che viene irrimediabilmente perduto; il discorso è analogo per la cella `num1`.



Quando si assegna un valore a una variabile di tipo `char`, per esempio un carattere `ASCII`, il carattere viene indicato tra apici per evitare che possa essere confuso con l'identificatore della variabile stessa.

```
char rosso, verde, vuoto, at; // definizione
rosso ← "R"; // assegnazione
verde ← "V";
vuoto ← " ";
at ← "@";
```

Assegnazione di una variabile a una variabile

Oltre a un valore già definito, è possibile assegnare a una variabile un valore (che possiamo anche non conoscere!) presente in un'altra variabile. Esaminiamo il seguente segmento di codice.

```
int num1, num2; // definizione di due variabili
num1 ← 0; // inizializzazione
num2 ← 0;
num1 ← 20; // assegnazione valore alla prima variabile
num2 ← num1; // assegnazione del contenuto di num1 a num2
```

L'ultima istruzione permette di copiare il contenuto della variabile `num1` nella variabile `num2`.



Si potrebbe risparmiare una istruzione inizializzando direttamente la variabile `num1` al valore di 20.

Assegnazione di un'espressione a una variabile

È possibile assegnare a una variabile il valore risultante dal calcolo di un'espressione: basta scrivere l'espressione che produce tale valore nella parte destra dell'assegnazione, come nel diagramma seguente.

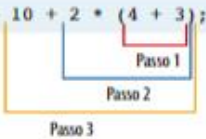


Vediamo alcuni esempi.

```
int num1, num2;           // definizione di due variabili
num1 ← 0;                 // inizializzazione
num2 ← 0;
num1 ← 10 + 2 * (4 + 3);  // a) espressione numerica
num2 ← (num1 * 2) + 10;  // b) espressione algebrica
```

La valutazione delle espressioni avviene nel rispetto delle precedenze algebriche.

```
num1 ← 10 + 2 * (4 + 3);
```



```
num1 ← 10 + 2 * (4 + 3);
num1 ← 10 + 2 * 7;           // passo 1
num1 ← 10 + 14;             // passo 2
num1 ← 24;                  // passo 3
num1 ← 24;                  // assegnazione di 24 alla variabile num1
```

Per la espressione b) lo svolgimento è analogo, quindi si ha:

```
num2 ← (24 * 2) + 10;       // in num1 ora è presente il valore 24
num2 ← 48 + 10;            // passo 1: calcolo 24 * 2
num2 ← 58;                  // passo 2: calcolo 48 + 10
num2 ← 58;                  // passo 3: assegnazione a num2
```

Assegnazione di una variabile a se stessa

Dato che la valutazione dell'istruzione di assegnazione avviene "da destra verso sinistra", è possibile seguire istruzioni come la seguente.

```
num1 ← num1 + 10;
```

che si legge: "10 viene sommato al valore contenuto nella variabile `num1` e il risultato viene assegnato alla variabile `num1`", cioè a se stessa!

Completiamo l'esempio assegnando a `num1` un valore iniziale.

```
num1 ← 2;                   // assegnazione
num1 ← num1 + 10;
```

Al termine della seconda istruzione, la variabile `num1` conterrà il valore 12, ottenuto dalla somma fra il valore della variabile `num1` (che è 2) e il valore 10.

Vediamo un esempio più articolato.

```
int num1, num2;           // definizione di due variabili
num1 ← 10;                 // inizializzazione
num2 ← 2;
num2 ← num2 * 3;           // in num2 viene messo 6 (= 2 * 3)
num1 ← num1 - num2;        // in num1 viene messo 4 (= 10 - 6)
```



Spesso una **variabile** viene utilizzata per **aggiornare il conteggio di un insieme di elementi** che si susseguono con una frequenza prestabilita.

ESEMPIO

Supponiamo di voler contare le vittorie (o le sconfitte) della nostra squadra settimanalmente: il calcolo del numero totale avviene incrementando progressivamente una variabile di un'unità per ogni partita vinta o persa.

Per effettuare questa operazione si utilizza una variabile che, inizializzata al valore 0, volta per volta è aggiornata con il nuovo valore, ottenuto sommando uno al valore precedente; tale variabile prende il nome di **contatore**.

Un programma che utilizza i dati

Scriviamo un primo programma che utilizza due dati per produrne altri due: calcoliamo il perimetro e l'area di un rettangolo conoscendo la sua base e la sua altezza.

Realizziamo dapprima il diagramma di flusso e quindi la codifica in **linguaggio C++** e **Java**, che come possiamo osservare è identica.

Flow-chart	Linguaggio C++	Linguaggio Java
<pre> graph TD A[base ← 10] --> B[altezza ← 5] B --> C[perimetro ← (base + altezza)*2] C --> D[area ← base * altezza] </pre>	<pre> int base, altezza; int area, perimetro; base = 10; altezza = 5; perimetro = (base + altezza)*2; area = base * altezza; </pre>	<pre> int base, altezza; int area, perimetro; base = 10; altezza = 5; perimetro = (base + altezza)*2; area = base * altezza; </pre>

Scambiare il contenuto di due variabili

Tra le operazioni che vengono eseguite sulle variabili, una in particolare assume una certa importanza: questa operazione consiste nell'effettuare lo **scambio del contenuto tra due variabili**.

L'operazione che effettua lo **scambio del contenuto tra due variabili** prende il nome di **swap**.

Di seguito, vediamo praticamente come ciò avviene.

Date due variabili di nome **variab1** e **variab2**, contenenti due valori numerici diversi, scriviamo un programma che esegue lo scambio dei numeri che tali variabili contengono. In altre parole, dobbiamo effettuare lo scambio del contenuto delle due variabili.

Per esempio, avendo le due variabili **variab1** e **variab2**:

5

variab1

100

variab2

si vuole ottenere:

100

variab1

5

variab2

A un primo approccio, una soluzione “immediata” e efficace potrebbe sembrare la seguente.

```
variab1 = variab2;  
variab2 = variab1;
```

Tuttavia, compiendo tali operazioni, si verifica una situazione indesiderata: in entrambe le variabili, infatti, viene a trovarsi il valore 100, quindi il risultato è errato!

Questo accade perché, dopo la prima istruzione, si ottiene la seguente situazione:



Il valore 100 è ora presente in entrambe le variabili: si è perso “irrimediabilmente” il valore 5, che era contenuto in `variab1` prima che “gli venisse sovrapposto” il contenuto di `variab2`.



Come spesso succede, la **soluzione immediata** porta a **risultati errati**: non sempre bisogna fidarsi dell'intuito, ma è necessario riflettere anche sulle operazioni che sembrano ovvie e aiutarsi con esempi e grafici.

Per risolvere il problema possiamo ricorrere a un'analogia “idraulico-gastronomica”: supponiamo di avere due bicchieri, uno colmo di acqua e l'altro colmo di vino, e di volerne scambiare il contenuto: come procediamo?

Non esiste di fatto soluzione, a meno di non utilizzare un terzo bicchiere in cui versare il contenuto di uno dei due per potervi travasare il contenuto dell'altro.

Questa è la **strategia** che verrà utilizzata anche per scambiare il contenuto delle variabili.

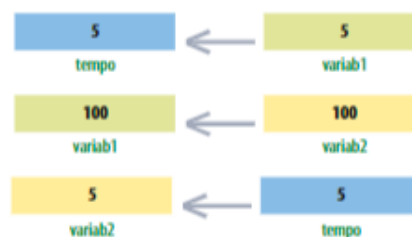


Con il termine **strategia** si indica l'**idea che porta alla soluzione di un problema** e in base alla quale vengono impostate ed eseguite le singole azioni necessarie per raggiungere tale scopo. La strategia è il punto centrale del lavoro di programmazione, in quanto consente di risolvere un problema e, di conseguenza, scrivere il programma.

Introduciamo pertanto una terza variabile utilizzata temporaneamente (che prende il nome di **variabile temporanea**) per effettuare un'operazione di salvataggio intermedio.

```
tempo = variab1;  
variab1 = variab2;  
variab2 = tempo;
```

Eseguito ora queste tre istruzioni possiamo osservare che il risultato dell'elaborazione è corretto: abbiamo realizzato il codice dell'operazione di swap.



Le costanti

In alcuni casi, il valore che viene scritto in una variabile (per esempio il numero di telefono dell'utente) può non subire modifiche durante l'esecuzione di un certo programma; in molti altri, invece, un valore può rimanere inalterato qualsiasi sia il programma che viene eseguito: per esempio il **numero dei nani di Biancaneve**, il valore della **costante di gravitazione universale** g , oppure il **valore di π** ($= 3,14$).

Queste "variabili che non variano" prendono il nome di **costanti**.



Una **costante** è un "oggetto" a cui sono associati un **identificatore** e un **valore che non può essere modificato** e che quindi resta fisso per tutta l'esecuzione del programma.

La definizione e la contemporanea inizializzazione della costante avvengono all'inizio del programma, nelle prime istruzioni. Anche se non è obbligatorio, è buona norma utilizzare caratteri maiuscoli per definire l'identificatore di una costante, come nei seguenti esempi.

```
FIGRECO ← 3.14;  
NUM_NANI ← 7;  
CAMBIO_EURO_LIRA ← 1936.27;
```

Una modalità di programmazione particolarmente efficiente, che rende flessibile il programma, è la **tecnica delle costanti manifeste**, che consiste nel raggruppare e definire come costanti tutti i numeri eventualmente utilizzati nel programma, anche senza averne un'apparente necessità; in questo modo, il codice delle istruzioni viene reso parametrico, cioè "libero" da numeri fissi.

```
NR_ALUNNI ← 22;  
STAGIONI ← 4;  
DITA_X_MANO ← 5;  
MESI_X_ANNO ← 12;  
GIORNI_X_ANNO ← 365;
```



Quando una **costante** viene **inserita** direttamente nel codice **in forma letterale attraverso un identificatore** prende anche il nome di **costante simbolica**.

Questi numeri costanti potrebbero invece essere impiegati all'interno del programma dalle strutture di controllo che verranno descritte in seguito senza necessariamente essere definiti come **costanti simboliche**: in tal caso, prendono il nome di **numeri magici** e possono essere utilizzati senza commettere errori.



Avendo raggruppati tutti i **valori numerici all'inizio del codice**, **gli interventi per effettuare le rettifiche saranno limitati e circoscritti**, in quanto non sarà necessario andarne a cercare tutte le occorrenze dei valori all'interno del programma. Così, oltre a mantenere la leggibilità del codice, si guadagna anche in flessibilità, come vedremo nelle lezioni successive.

Non è detto che alcuni valori, che oggi sembrano immutabili, in un prossimo futuro non possano essere modificati o sostituiti: ricordiamo, per esempio, la scomparsa della lira e l'introduzione dell'euro, certamente non prevista (e non prevedibile) dai programmatori della fine del secolo scorso e che ha comportato, per l'adeguamento dei codici scritti senza tecniche di parametrizzazione, interventi anche piuttosto radicali (come la completa riscrittura del codice).

Le costanti in C++

In C++ le costanti possono essere definite in due modi:

– con la parola riservata `const`:

```
const int NR_NANI = 7;
const char OK = 's';
const float PI_GRECO = 3,1415;
```

– con l'istruzione `#define`, come nel linguaggio C++:

```
#define MESI 12
#define GIORNI 7
```

Entrambe le modalità hanno come effetto quello di definire alcune **variabili immutabili**, cioè di definire aree di memoria e inizializzarle con un valore che non **può essere successivamente modificato**.



Preprocessore

Il preprocessore è un semplice programma che agisce in modo automatico sul testo del programma prima del compilatore: interviene cioè sulle righe del codice "senza conoscere il linguaggio C++", eseguendo semplicemente delle direttive contenute all'inizio del file per:

- includere altre porzioni di testo, prese da altri file;
- effettuare ricerche e sostituzioni (più o meno sofisticate) sul testo;
- inserire o sopprimere parti del testo a seconda del verificarsi di certe condizioni da noi specificate all'inizio del codice sorgente che sta processando.

PER SAPERNE DI PIÙ

L'ISTRUZIONE #define

L'istruzione `#define` viene utilizzata per definire quelle che nel preprocessore sono chiamate macro: il **preprocessore** legge la definizione della **macro** e, ogni volta che ne incontra il nome all'interno del file sorgente, **sostituisce al simbolo il corrispondente valore, senza verificare la correttezza sintattica dell'espressione risultante**.

Attraverso una **macro** è possibile definire una istruzione completa: il preprocessore la inserirà tutte le volte che la incontra nel programma.

Le costanti in Java

In Java le costanti vengono definite con la parola riservata `final`:

```
final int NR_NANI = 7;
final char OK = 's';
final double PI_GRECO = 3.1415;
final float PI_GRECOF = 3.1415f; // vuole f come terminatore
```

Inoltre le costanti di tipo `float` devono terminare con il carattere `f`.

Ogni tentativo di cambiare il valore di una costante produce una segnalazione di errore in quanto le aree di memoria così definite sono **immutabili**, cioè sono aree di memoria definite e inizializzate con un valore che non **può essere successivamente modificato**.

METTITI ALLA PROVA

→ • Variabili e costanti

Dato il valore del lato di un rombo, determina il suo perimetro e la sua area, l'area di un rettangolo avente come lati le sue diagonali e l'area di un quadrato avente i lati della stessa misura di quelli del rombo.

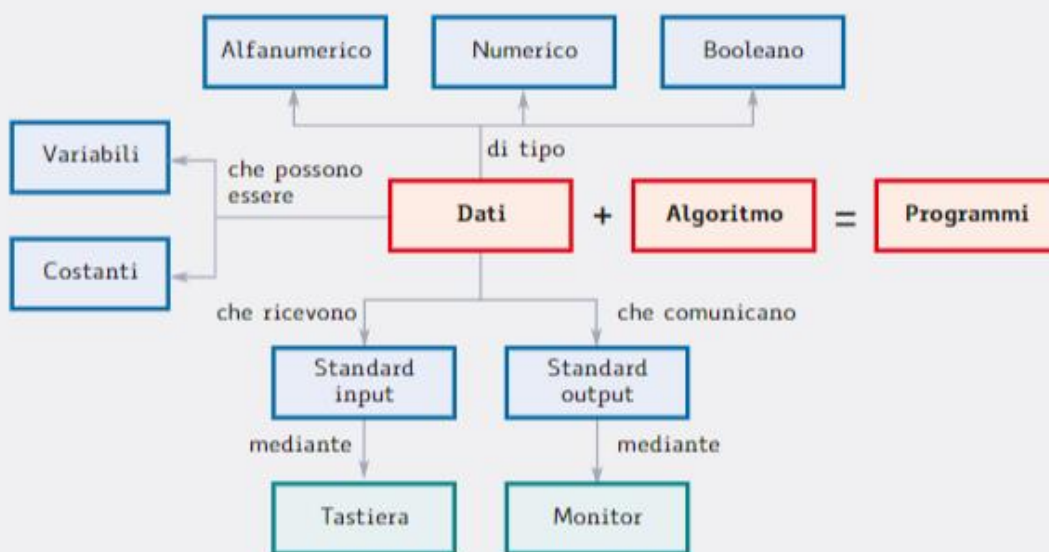
3

L'input e l'output dei dati

IN QUESTA LEZIONE IMPAREREMO...

- a effettuare l'output di dati sullo schermo
- a effettuare l'input dei dati da tastiera

MAPPA CONCETTUALE



La comunicazione con l'elaboratore

Il programma ha la possibilità di "comunicare" con l'utente tramite due modalità: di **input** e di **output**:

- l'**input** (dati in ingresso) consente al programma di ricevere dati e informazioni dall'esterno allo scopo di poterli elaborare;
- l'**output** (dati in uscita), invece, consiste nella comunicazione all'utente dei dati e delle informazioni così elaborati.

Tutti i programmi si basano sul seguente schema.



Tradotto in parole, significa che il risultato (**output**) è **ottenuto mediante** le **operazioni** elementari descritte nell'**algoritmo** ed è il **prodotto** della **elaborazione** (trasformazione) dei dati in ingresso (**input**).



In un PC, il più tipico dispositivo di **input** è la **tastiera** mentre il **monitor** è il dispositivo di **output** per eccellenza.

Nel **linguaggio di progetto**, una istruzione di **output** viene generalmente indicata con la parola riservata **scrivi**, cioè con un verbo che indica al calcolatore cosa deve fare.

```
scrivi (areaTriangolo) // scrivi sullo schermo il contenuto di una singola variabile
scrivi ("l'area del triangolo e' :" areaTriangolo) // composizione di un testo con una variabile
```

Sempre nel **linguaggio di progetto**, una istruzione di **input** viene invece generalmente indicata con la parola riservata **leggi**, che indica al calcolatore di ricevere da tastiera un valore e memorizzarlo in una **variabile**.

```
leggi (base) //leggi un numero e memorizzato nella variabile base
leggi (altezza) //leggi un numero e memorizzato nella variabile altezza
```

L'input e l'output in C++

Per eseguire le **operazioni di I/O** nel linguaggio C++ è necessario includere nel programma una **apposita libreria** in quanto il linguaggio non prevede al suo interno la gestione di tali operazioni.

Una **libreria** è una "collezione di operazioni" (**funzioni**) che estendono le **funzionalità base** previste da un linguaggio di programmazione". In C++ si parla di **libreria standard** nel caso di librerie che possono essere usate su qualsiasi sistema operativo: le principali sono **cstdlib**, **iostream**, **cmath**, **fstream**.

Cout

L'istruzione per comunicare con l'utente utilizza la parola riservata **cout** e ha la seguente forma.

```
cout << "Programma che calcola il perimetro e l'area di un rombo" << endl;
```

Il comando **endl** (fine riga) serve per andare alla successiva riga nello schermo.



È anche possibile visualizzare sulla stessa riga un testo seguito dal contenuto di una variabile.

```
cout << "la misura del perimetro e' :" << perimetro << endl;
```

Nel caso di variabili reali è anche possibile indicare il numero di cifre da visualizzare (virgola compresa) mediante il comando:

```
cout.precision(3); // visualizza 2 cifre decimali
```

Cin

L'istruzione per leggere una variabile inserita dall'utente utilizza la parola riservata `cin` e ha la seguente forma:

```
cin >> base;
```

Gli operatori `<<` e `>>` indicano in che direzione avviene il "flusso dei dati".



Dato che il programma viene eseguito in modalità testo, e quindi in una finestra DOS, alla sua terminazione tale finestra si chiude automaticamente: per mantenerla aperta in modo da poter vedere il risultato dell'elaborazione introduciamo una istruzione che "si pone in attesa" della pressione del tasto di Invio.

```
system( "PAUSE" ); // per non chiudere la finestra DOS
```

Scriviamo un primo programma che utilizza le istruzioni di I/O.

✓ PROBLEMA SVOLTO PASSO PASSO

✓ Il problema

Scriviamo il codice completo di un programma che legge il valore della base e dell'altezza di un rettangolo e successivamente ne calcola area e perimetro.

✓ La pseudocodifica e l'algoritmo risolutivo

La pseudocodifica completa è la seguente.

I affinamento
- leggi i dati
- calcola l'area
- calcola il perimetro
- scrivi i risultati

Il affinamento - pseudocodifica

```
leggi(base)  
leggi(altezza)  
area = base * altezza  
perimetro = (base+altezza)*2  
scrivi("l'area e' : " area)  
scrivi("il perimetro e' : " perimetro)
```



✓ La codifica in linguaggio di programmazione C++

```
#include <iostream>
using namespace std;      // per indicare lo spazio dei nomi delle variabili
int main()
{
    // dichiarazione variabili
    int base, altezza;
    int area, perimetro;
    // input dei dati
    cout << "Programma che calcola il perimetro e l'area di un rettangolo" << endl;
    cout << "inserire la base \t: ";
    cin >> base;
    cout << "inserire l'altezza \t: ";
    cin >> altezza;
    // elaborazione
    perimetro = (base + altezza) * 2;
    area = base * altezza;
    // output dei risultati
    cout << "la misura del perimetro\t: " << perimetro << endl;
    cout << "la misura dell'area \t: " << area << endl;

    system("PAUSE");      // per non chiudere la finestra DOS
    return 0;
}
```



Con il carattere "\" vengono aggiunte nell'output le opzioni di formattazione: riportiamo le principali nella seguente tabella.

```
\n    // manda la riga a capo
\t    // introduce una tabolazione
\f    // salto pagina
\'    // permette di avere un doppio', che altrimenti verrebbe considerato la chiusura stringa
\\    // introduce una barra contraria
```

✓ L'esecuzione del programma

```
Programma che calcola il perimetro e l'area di un rettangolo
inserire la base      : 5
inserire l'altezza   : 10
la misura del perimetro : 30
la misura dell'area   : 50
Premere un tasto per continuare . . . =
```

Il codice sorgente di questo programma lo trovi nel file [rettangolo.cpp](#).

L'input e l'output in Java

Il linguaggio Java implementa un sistema abbastanza complesso per effettuare le operazioni di I/O che si basa sulla classe `System`, descritta dettagliatamente in seguito, che rappresenta la "virtualizzazione del sistema" su cui l'applicazione è in esecuzione.



Ricordiamo che Java è stato progettato per essere multipiattaforma, e quindi il progetto del meccanismo di I/O è risultato essere tra le operazioni più complesse da definire per poter rendere il software assolutamente indipendente dall'hardware, sia di quello esistente negli anni '90, cioè al momento della definizione di Java, che di quello non ancora inventato e "disponibile solo nel futuro".

L'output

Per effettuare l'**output** utilizzeremo un membro (statico) di tale classe, `System.out`, che mette a disposizione alcune funzioni (metodi) specifiche, riportate di seguito.

```
System.out.print(<stringa da scrivere >);           // di seguito
System.out.println(<stringa da scrivere >);        // va a riga nuova
```

Alcuni esempi sono riportati di seguito.

```
System.out.print("scrivo un numero ");           // di seguito
System.out.println(" di seguito " + 123);        // di a riga nuova
System.out.println("la nuova riga e' a capo ");   // va a riga nuova
```

```
scrivo un numero di seguito 123
la nuova riga e' a capo
```

Nelle operazioni di **output** è necessario inserire un comando di formattazione in modo da stabilire il numero di decimali che si vogliono visualizzare e ciò si effettua mediante l'utilizzo di **specifiche di formato** (stringhe di formattazione) nel metodo `printf()`, sempre della classe `System.out`.

```
System.out.printf(<stringa_formattazione>, <numero>);
```

Le **specifiche di formato** iniziano con un segno di percentuale (%) e terminano con un convertitore che è un carattere che indica il tipo di argomento da formattare: tra il segno percentuale (%) e il convertitore si possono aggiungere flag opzionali per specificare il numero di cifre da visualizzare.

I principali convertitori sono riportati nella seguente tabella:

CONVERTITORE	FLAG	UTILIZZO	ESEMPIO	OUTPUT
d		Un numero decimale	<code>System.out.printf("%d", n);</code>	"46101"
	0x	x caratteri preceduti da 0	<code>System.out.printf("%08d", n);</code>	"00046101"
f		Un numero float	<code>System.out.printf("%f", pi);</code>	"3.141593"
	.x	x cifre dopo la virgola	<code>System.out.printf("%.3f", pi);</code>	"3.142"
	x.y	x cifre totali di cui y dopo la virgola	<code>System.out.printf("%8.4f", pi);</code>	"3.1416"

Sono anche disponibili convertitori per la formattazione delle date e degli orari: puoi visualizzare un esempio completo del loro utilizzo nel programma `TestFormat.java`.

In **Java** è anche possibile creare un formato e applicarlo ai numeri con i metodi `print/println` al momento dell'output mediante la seguente istruzione:

```
DecimalFormat format = new DecimalFormat ("0.##"); // 2 cifre decimali
```


che per poterla utilizzare è necessario aggiungere l'inclusione della apposita libreria con il comando:

```
import java.text.DecimalFormat;
```

Vediamo un esempio per ciascun tipo di variabile numerica.

```
DecimalFormat fmt0 = new DecimalFormat ("0"); // 0 cifre decimali
DecimalFormat fmt1 = new DecimalFormat ("0.#"); // 1 cifra decimale
DecimalFormat fmt2 = new DecimalFormat ("0.##"); // 2 cifre decimali
System.out.println("numero senza decimali : " + fmt0.format(123.456));
System.out.println("numero con 1 decimale : " + fmt1.format(123.456));
System.out.println("numero con 2 decimali : " + fmt2.format(123.456));
```

che in esecuzione produce il seguente output:

```
numero senza decimali :123
numero con 1 decimale :123,5
numero con 2 decimali :123,46
```



Un'ulteriore alternativa viene fornita da un altro membro (statico) della classe `System.out`, il metodo `System.out.format()`, che consente di creare delle stringhe formattate e di visualizzarle sullo schermo, sia direttamente che indirettamente, salvandole prima in una variabile.

L'input

Più articolato e complesso è il meccanismo che permette di effettuare le operazioni di **input**: tramite una sottoclasse di `System`, la classe `InputStream`, è possibile effettuare la lettura di un singolo carattere e, per riuscire a effettuare la lettura di una riga alla volta, è necessario bufferizzarlo.

Riportiamo a titolo di esempio un segmento di codice necessario per effettuare l'input di una variabile:

```
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader tastiera = new BufferedReader(input);
System.out.println("Inserire un dato: ");
String line = input.readLine();
```



Inoltre la classe `InputStreamReader` non possiede metodi comodi per la ricezione di dati numerici e stringhe: per ovviare a questo inconveniente `Java 5.0` ha introdotto la classe `Scanner`.

Senza approfondire il meccanismo, noi creeremo con essa un "oggetto di input", che chiameremo `in`, sul quale potremo richiamare semplici funzioni per leggere i dati.



Il codice di queste funzioni che utilizzeremo per effettuare le operazioni di I/O non è presente nella libreria base del linguaggio: per poterle richiamare è necessario aggiungere all'inizio del codice la seguente riga, in modo da includere la libreria voluta.

```
import java.util.Scanner;
```

Dopo il loro inserimento nel nostro programma possiamo effettuare la lettura di un dato e la conversione nel formato desiderato semplicemente con una sola istruzione:

```
int <variabile> = in.nextInt();
double <variabile> = in.nextDouble();
float <variabile> = in.nextFloat();
String <variabile> = in.next();           // parola delimitata da spaziatura (SP,\t,\n,\r)
String <variabile> = in.nextLine();     // riga delimitata da '\n' o '\r'
```

Scriviamo un programma che utilizza le istruzioni di I/O.

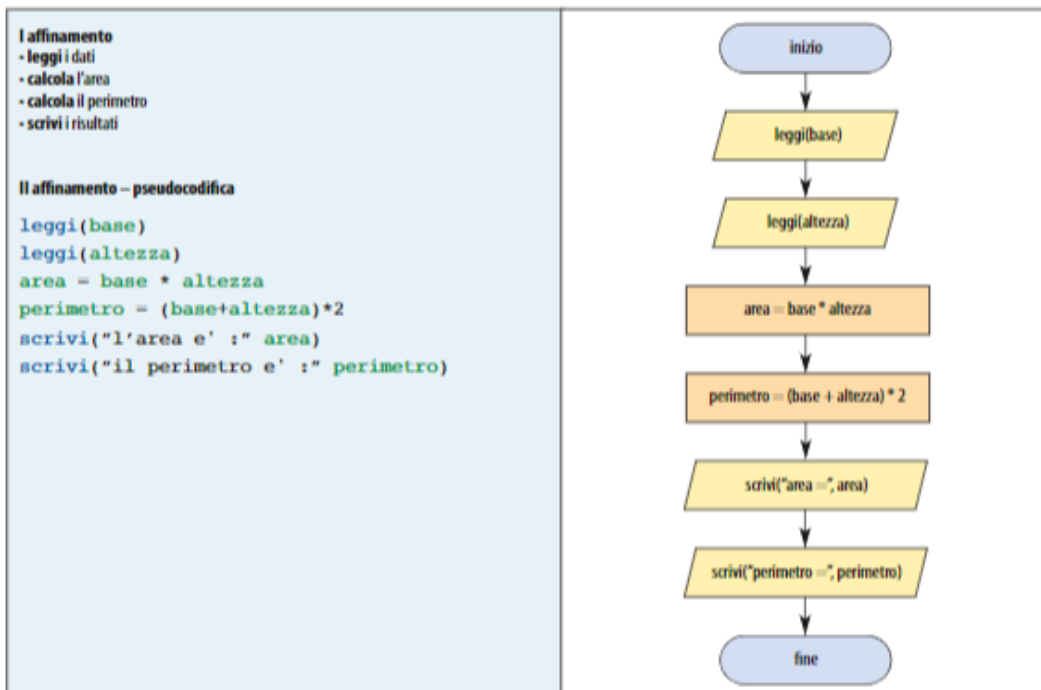
✓ PROBLEMA SVOLTO PASSO PASSO

✓ Il problema

Scriviamo il programma che calcola area e perimetro di un rettangolo dopo averne letto i valori della base e dell'altezza.

✓ La pseudocodifica e l'algoritmo risolutivo

La pseudocodifica completa è la seguente:



✓ La codifica in linguaggio di programmazione Java

```
import java.util.Scanner;           // libreria aggiunta
public class ProvaIOScanner        // nome della classe
{
    public static void main(String args[])
    {
        // definizione dell'oggetto per l'input
        Scanner in = new Scanner(System.in);
        // definizione delle variabili
        int base, altezza; // per i dati
        int area, perimetro; // per i risultati
        // fase di acquisizione dei dati
        System.out.print("Inserisci la base : ");
        base = in.nextInt();
        System.out.print("Inserisci l'altezza: ");
        altezza = in.nextInt();
        // esecuzione dei calcoli
        perimetro = (base + altezza) * 2;
        area = base * altezza;
        // visualizzazione dei risultati
        System.out.println("Il valore del perimetro e': " + perimetro);
        System.out.println("Il valore dell'area e': " + area );
    }
}
```

✓ L'esecuzione del programma

```
Inserisci la base : 6
Inserisci l'altezza: 5
Il valore del perimetro e': 22
Il valore dell'area e': 30
```

Il codice sorgente lo trovi nel file `ProvaIOScanner.java`.

➤ METTITI ALLA PROVA

➔ • Utilizzo di variabili • Istruzioni di I/O

AREA DIGITALE



Fare l'input
senza la classe Scanner

Calcola la temperatura media di un pomeriggio estivo effettuando quattro rilievi in predeterminate ore mediante un termometro centigrado.

Scrivi una versione del programma utilizzando due sole variabili per calcolare la media di quattro temperature.

Confronta la tua soluzione con quella riportata nei file `media1Sol.cpp` e `Media1Sol.java`.