

## APPUNTI DI INFORMATICA – PROBLEMI E ALGORITMI

### 1.1 CONCETTI INTRODUTTIVI

#### Definizione di problema

Un problema<sup>1</sup> è un quesito nel cui enunciato si forniscono i *dati* necessari per giungere, mediante *calcoli* o elaborazioni, alla *soluzione* richiesta nell'enunciato stesso.

#### Che cosa significa risolvere un problema

Dal punto di vista informatico, risolvere un problema significa **ricercare** ed **esprimere** un elenco di istruzioni (algoritmo) che, interpretate da un **esecutore**, conducano da determinate informazioni iniziali (dati) a informazioni finali (risultati).

#### Entità coinvolte

Nel processo di risoluzione di un problema, sono generalmente coinvolte le seguenti entità:

- ◆ **Committente**: colui che pone il quesito.
- ◆ **Risolutore**: colui che mette a punto l'algoritmo stabilendo quali calcoli e/o elaborazioni sono necessarie per ottenere i risultati richiesti.
- ◆ **Esecutore**: chi, uomo o macchina, esegue l'algoritmo per giungere ai risultati cercati.
- ◆ **Utente**: chi fornisce i dati iniziali all'esecutore e attende i risultati.

#### Differenza fra risultato e soluzione

Il risultato è costituito dall'insieme dei dati finali. La soluzione è l'algoritmo che permette di ottenere i risultati.

#### Definizione di istruzione

L'istruzione è un comando elementare che dice all'esecutore qual è l'operazione da eseguire in quel momento. La tipologia e la sintassi delle istruzioni dipendono dalle caratteristiche dell'esecutore.

#### Definizione di algoritmo

Un algoritmo<sup>2</sup> è un insieme di istruzioni che soddisfa le seguenti proprietà:

1. E' **finito**: si conclude dopo un numero finito di passi, cioè l'esecuzione termina in un tempo finito.
2. E' **definito** e **preciso**: ogni istruzione è definita in maniera non ambigua, univocamente interpretabile dall'esecutore; ad ogni passo deve essere definita una e una sola operazione successiva.
3. Se ci sono dati in ingresso, la tipologia di questi è precisata (es. numeri interi, numeri reali, stringhe, ..).
4. Fornisce almeno un **risultato**: un algoritmo che non fornisce risultati è inutile.
5. E' **eseguibile**: tutte le istruzioni possono essere eseguite in un tempo finito, cioè l'esecutore è in grado, con le risorse a sua disposizione, di eseguire ogni istruzione indicata.
6. Risolve una **classe di problemi**: garantisce risultati corretti per tutti i problemi appartenenti alla stessa classe. Tuttavia ciò non vieta di scrivere un algoritmo specifico per una particolare istanza di problema.

In sintesi, l'algoritmo deve essere **finito**, **definito**, **eseguibile** e **deterministico**.

#### Differenza fra classe di problemi e istanza di problema

Una classe di problemi è un problema nel cui enunciato almeno un dato è fornito in forma generica (*parametro*); un'istanza di problema è un problema nel cui enunciato tutti i dati sono forniti tramite valori specifici (*costanti*).

<sup>1</sup> Vocabolario della lingua italiana - Lo Zingarelli 1994 - N. Zingarelli

<sup>2</sup> D. Knuth - The Art of Computer Programming

Per esempio l'enunciato "Calcolare l'area di un rettangolo di base **B** e altezza **H**" rappresenta una classe di problemi, mentre "Calcolare l'area di un rettangolo di base **2** e altezza **3**" e "Calcolare l'area di un rettangolo di base **17** e altezza **53**" sono due delle infinite istanze della stessa classe di problemi.

### Definizione di programma

Il programma è un algoritmo espresso con un linguaggio di programmazione (*Pascal, C, C++, Java, ..*) corredato di:

1. **dichiarazione** delle risorse minime per la sua esecuzione tramite un PC;
2. **documentazione** adeguata affinché il programma possa mantenersi efficiente nel tempo ed essere modificato all'occorrenza;
3. **manuale** per l'utente finale che dia precise indicazioni sul suo uso.

### Esercizi:

- 1) *Tre amici, Aldo, Giovanni e Giacomo girano per l'Europa. Aldo e Giovanni sono ottimi camminatori e riescono a procedere a 8 km/ora per lungo periodo. Giacomo invece ha un piede ingessato e può solo guidare una piccola auto che viaggia a 60 km/h. L'auto può portare due sole persone. Definire la procedura che devono seguire i tre amici per viaggiare alla massima velocità, compatibilmente con i limiti del problema. Calcolare inoltre tale velocità.*
- 2) *Un pastore deve attraversare un fiume portando con sé un lupo, una capra e un cavolo. Egli può far uso di una barchetta che può ospitare solo un altro viaggiatore. Tenendo presente che il lupo tende a mangiare la capra e questa il cavolo, cosa può fare il pastore per raggiungere il suo scopo? Quali sono i dati del problema? Qual è il risultato? Qual è la soluzione? Evidenziare la differenza fra il risultato e la soluzione.*
- 3) *Descrivere, sotto forma di algoritmo (finito, definito, eseguibile e deterministico), una ricetta per la preparazione di una pietanza a piacere.*

## 1.2 UN APPROCCIO SISTEMATICO ALLA RISOLUZIONE DEI PROBLEMI

Ogni volta che si vuole risolvere un problema con l'aiuto di un **calcolatore** è necessario individuare un algoritmo. Questa fase risulta facilitata se il problema stesso è affrontato attraverso un ragionamento metodico.

Nella piena consapevolezza di non poter dare indicazioni precise e complete per risolvere qualsiasi problema (avremmo trovato l'algoritmo per trovare gli algoritmi!), qui di seguito descriviamo i passi necessari alla trasformazione di un problema concreto in algoritmo, in accordo ad un approccio sistematico.

### 1.2.1 Primo passo – individuare e classificare i dati

Il primo passo consiste nel comprendere con precisione il problema da risolvere. Ciò avviene tramite l'individuazione e la classificazione precisa di tutti e solo i **dati** d'interesse, eliminando ogni sorta d'ambiguità e d'informazione superflua o fuorviante eventualmente presenti nel testo del problema.

#### I dati sono classificabili da più punti di vista

a) In base all'allusione:

- dati **IMPLICITI** - sono dati, che pur non essendo stati espressi nell'enunciato del problema, sono **sottintesi** ed essenziali per la determinazione dei risultati finali;
- dati **ESPLICITI** - sono i dati chiaramente evidenziati nell'enunciato del problema.

b) In base alla possibilità di modifica:

- dati **COSTANTI** - detti semplicemente **costanti** (al femminile), sono dati non modificabili; per esempio nell'enunciato "*Calcolare l'area di un rettangolo*" sono dati impliciti la *base* e l'*altezza* in quanto senza di essi è impossibile calcolare l'area di un rettangolo, mentre nell'enunciato "*Determinare l'equivalente in Lire di N Euro*" il dato implicito è *1936,27*. Le costanti non possono variare nel corso di una elaborazione e tanto meno da una elaborazione all'altra;
- dati **VARIABILI** - detti semplicemente **variabili** (al femminile), sono dati suscettibili di variazioni; essi possono variare da un'elaborazione all'altra o nel corso di una stessa elaborazione. Le variabili, durante l'elaborazione, giocano il ruolo di **contenitori di valori**.

c) In base al numero di elementi che possono contenere contemporaneamente:

- dati **SEMPLICI** o SCALARI - possono contenere un solo elemento per volta costituito da una grandezza atomica non ulteriormente scomponibile, come ad esempio un numero intero o un numero reale;
- dati **STRUTTURATI** o COMPOSTI - possono contenere più elementi contemporaneamente e vengono utilizzati per trattare in modo globale più informazioni semplici, come ad esempio i voti finali di uno studente, l'elenco degli studenti di una classe, ecc..

d) In base al *tipo* di valori che il dato può assumere e alle *operazioni* che possono essere eseguite su di essi:

- dati **NUMERICI** - contengono numeri, come le misure dei lati di un rettangolo o il prezzo di un prodotto. Su questi dati si può operare con le usuali operazioni matematiche (addizione, sottrazione,...). Questi dati possono essere ulteriormente distinti in **interi** e **reali** in modo da rappresentare "meglio" le informazioni a cui si riferiscono;
- dati **ALFANUMERICI** - contengono caratteri alfabetici, cifre o segni speciali e sono utilizzati per rappresentare stringhe di caratteri come nominativi e indirizzi. Ulteriori caratteristiche e tipi dipendono dal linguaggio di programmazione e verranno dettagliati al momento opportuno;
- dati **BOOLEANI** - contengono valori logici come SI/NO, VERO/FALSO, TRUE/FALSE. In molti casi i valori di questo tipo di dato sono rappresentati con i numeri interi **0** per FALSO e **1** per VERO;

e) In base all'utilizzo (in questo contesto vengono considerati solo i dati variabili):

- variabili di **INPUT** - servono per fornire all'esecutore i dati su cui operare;

- variabili di **LAVORO** o di **APPOGGIO** - servono all'esecutore per l'esecuzione stessa; spesso l'individuazione di queste variabili è legata alla strategia risolutiva stessa che il risolutore intende adottare;
- variabili di **OUTPUT** - sono usate dall'esecutore per comunicare i risultati dell'elaborazione.

### Esempi:

- 1) Nell'enunciato "*Calcolare l'area di un rettangolo*" la *base* e l'*altezza* sono dati impliciti in quanto senza di essi è impossibile calcolare l'area di un rettangolo, mentre l'*area* cercata è un dato esplicito; questi dati sono tutti variabili, semplici e numerici; inoltre la *base* e l'*altezza* sono variabili di input mentre l'*area* è una variabile di output.
- 2) Nell'enunciato "*Determinare il corrispettivo in Lire di N Euro*" il dato implicito è 1936,27 che è anche un dato costante, semplice e numerico (reale); *N* e il *corrispettivo* sono dati espliciti, numerici e variabili; *N* è una variabile di input mentre il *corrispettivo* è una variabile di output.
- 3) Nell'enunciato "*Determinare il corrispettivo in Lire di N Euro*" il dato implicito è 1936,27 che è anche un dato costante, semplice e numerico (reale); *N* e il *corrispettivo* sono dati espliciti, numerici e variabili; *N* è una variabile di input mentre il *corrispettivo* è una variabile di output.

Un modo conveniente di procedere in questa fase consiste nel compilare la **tabella delle costanti** e la **tabella delle variabili**, contenenti nome (solo per le variabili), classi d'appartenenza e significato di ogni dato:

#### TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione

#### TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione

I nomi delle variabili sono scelti accuratamente dal risolutore in accordo con il significato dei dati stessi.

### Esempi:

1.a) "*Dati i pesi di tre oggetti, calcolare il peso medio*".

#### TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione
3	intero	Numero di oggetti

#### TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
P1	reale	X			Peso del primo oggetto
P2	reale	X			Peso del secondo oggetto
P3	reale	X			Peso del terzo oggetto
S	reale		X		Somma dei pesi
PM	reale			X	Peso medio

2.a) "*Sul prezzo di un prodotto viene applicato lo sconto del 3%. Calcolare il prezzo scontato*".

#### TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione
3	intero	Ragione dello sconto
100	intero	Costante di proporzionalità dello sconto

## TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
COSTO	reale	X			Costo del prodotto
SCONTO	reale		X		Sconto sul prezzo di vendita
PREZZO	reale			X	Prezzo scontato

**3.a)** "Calcolare la retribuzione netta di un lavoratore, noti il costo di un'ora di lavoro, il numero di ore lavorate e una trattenuta per oneri sociali pari al 27% della retribuzione lorda".

## TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione
27	intero	Ragione della trattenuta
100	intero	Costante di proporzionalità della trattenuta

## TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
CL	reale	X			Costo di un'ora di lavoro (in Euro)
N	intero	X			Numero di ore
RL	reale		X		Retribuzione lorda
ONERI	reale		X		Trattenuta per oneri sociali
RN	reale			X	Retribuzione netta

**4.a)** "La scuola rimborsa il 15% del costo dell'abbonamento se lo studente abita in provincia, usa l'autobus ed è lontano almeno 20 Km dalla scuola. Alle stesse condizioni, se usa il treno il rimborso è del 10%. Calcolare l'ammontare del rimborso".

## TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione
15	intero	Ragione del rimborso per gli studenti che abitano in provincia, usano l'autobus e sono distanti almeno 20 Km dalla scuola
10	intero	Ragione del rimborso per gli studenti che abitano in provincia, usano il treno e sono distanti almeno 20 Km dalla scuola
100	intero	Costante di proporzionalità dei rimborsi
20	intero	Distanza minima dalla scuola per ottenere un rimborso

## TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
MEZZO	stringa	X			Mezzo di trasporto: "treno", "autobus"
LOCALITA	stringa	X			Provenienza: "comune", "provincia"
COSTO	reale	X			Costo dell'abbonamento
RIMBORSO	reale			X	Ammontare del rimborso

**Nota** - il risolutore, in questo caso, potrebbe ritenere più conveniente "codificare" il mezzo di trasporto e il luogo di provenienza" tramite numeri interi, come illustrato di seguito:

## TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
MEZZO	intero	X			Mezzo di trasporto: <b>1</b> significa AUTOBUS <b>2</b> significa TRENO
LOCALITA	intero	X			Luogo di provenienza: <b>1</b> significa COMUNE, <b>2</b> significa PROVINCIA
COSTO	reale	X			Costo dell'abbonamento
RIMBORSO	reale		X		Ammontare del rimborso

5.a) "Dati i pesi di  $N$  oggetti, calcolare il peso medio".

TABELLA DELLE COSTANTI: **nessuna costante!**

TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
N	intero	X			Numero di oggetti
P	reale	X			Peso di un oggetto
I	intero		X		Contatore dei pesi
S	reale		X		Somma dei pesi
PM	reale			X	Peso medio

**ATTENZIONE:** sebbene i "pesi degli oggetti" trovino una collocazione naturale tra i tipi di dati STRUTTURATI, conviene rappresentarli con una variabile semplice che sarà opportunamente manipolata dall'algoritmo (come verrà chiarito meglio in seguito).

Per ora in fase d'analisi dei dati, conveniamo di rappresentare un dato STRUTTURATO tramite una variabile semplice, escluse le *stringhe alfanumeriche*.

### Esercizi:

Costruire la tabella delle costanti e la tabella delle variabili per ciascuno dei seguenti problemi:

- 1) Calcolare il perimetro e l'area di un quadrato.
- 2) Dati il valore di un deposito bancario e il tasso di interesse annuo, calcolare gli interessi maturati dopo 25 giorni.
- 3) Sul prezzo di un prodotto viene praticato uno sconto del 3% se costa meno di 500 Euro e del 5% per prezzi superiori a 500 Euro. Calcolare il prezzo da pagare.
- 4) Ogni giorno vengono registrate, in una stazione meteorologica, le temperature minima e massima. Alla fine del mese si vuole conoscere la media delle temperature minime e la media delle massime.
- 5) Per il lavoro di un operaio vengono registrati l'orario di entrata e l'orario di uscita sia al mattino che al pomeriggio. Calcolare il totale delle ore e dei minuti lavorati e la paga spettante, nota la paga oraria, per un giorno di lavoro.
- 6) Uno studente esamina le informazioni sulle verifiche scritte che ha fatto durante l'anno scolastico, e vuole contare quante sono le prove sufficienti.

## 1.2.2 Secondo passo – individuare e descrivere le operazioni

La messa a punto di un algoritmo è un atto essenzialmente creativo. La **conoscenza** dell'argomento e l'**esperienza** del risolutore sono sicuramente indispensabili per un corretto approccio alla risoluzione di un problema. Questo passo è sicuramente il più complesso e difficile, perché non esiste alcun metodo universale per trovare, da soli o collettivamente, un algoritmo relativo ad un problema non ancora risolto. Una tecnica efficace per costruire algoritmi è rappresentata dal metodo degli **affinamenti successivi**. Esso consiste nell'affrontare un problema semplificandolo e nell'aggiungere, in momenti successivi, dettagli che permettono di giungere alla soluzione cercata. L'analisi che si compie quando si applica questo metodo è del tipo **dall'alto verso il basso** (*top-down*) e corrisponde ad una prima scomposizione del problema in sottoproblemi. Ogni sottoproblema può, a sua volta, essere scomposto fino al livello più elementare, che corrisponde ad un'istruzione dell'algoritmo.

Il secondo passo consiste nell'*individuare* e *descrivere* le **istruzioni** da far compiere all'esecutore sui dati della *tabella delle costanti e delle variabili* per ottenere i risultati finali, in accordo con i vincoli e le relazioni imposti dal problema.

### Classificazione delle istruzioni

In linea di principio un algoritmo può essere costituito da una sola istruzione, ma ciò accade raramente. Molto spesso si hanno più istruzioni, anche di tipo differente, opportunamente legate tra loro. Tali istruzioni devono rispondere ad una struttura organizzativa precisa; solo da una opportuna relazione tra le azioni si potrà ottenere il processo di trasformazione dei dati necessario per ottenere i risultati cercati.

Sulla base di tali considerazioni, le istruzioni si distinguono in:

1. istruzioni di **dichiarazione**
2. istruzioni **operative**
3. istruzioni di **controllo**

#### 1. Istruzioni di dichiarazione

Le istruzioni di dichiarazione descrivono le costanti e le variabili utilizzati nell'algoritmo, definendone tipo e struttura. Sono previste dalla maggior parte dei linguaggi di programmazione.

#### 2. Istruzioni operative

Le istruzioni operative corrispondono ad azioni direttamente eseguibili dall'esecutore. Queste vengono ulteriormente suddivise in:

- a) istruzioni di ASSEGNAZIONE
- b) istruzioni di INPUT/OUTPUT

L'istruzione di **ASSEGNAZIONE** permette di "assegnare" un valore ad una variabile. Il valore può essere una costante, il valore di un'altra variabile oppure il risultato di un'espressione. Il simbolo generalmente usato per l'assegnazione è il simbolo = (uguale). Per esempio

$$A=(B+C+2)/C$$

significa: "calcola il valore dell'espressione aritmetica di destra e assegna tale valore alla variabile di nome A". In generale per espressione si intende un oggetto costituito da operandi, siano essi costanti o variabili, e da operatori e/o "designatori" di funzioni, opportunamente legati in base alle regole del calcolo aritmetico o logico.

L'istruzione di assegnazione costituisce la base di ogni elaborazione e, per quanto di immediata comprensione, richiede attenzione nell'uso. Ad esempio non è possibile assegnare un numero a una stringa. L'assegnazione gode delle seguenti proprietà:

- è **conservativa a destra**: in un'assegnazione del tipo  $A=B$  il contenuto della variabile di destra B viene copiato nella variabile di sinistra A; la variabile di destra B non perde il suo valore;
- è **distruittiva a sinistra**: in un'assegnazione del tipo  $A=B$  il contenuto della variabile di sinistra A viene sostituito con una copia del valore della variabile di destra B; la variabile di sinistra A viene modificata.

Le istruzioni di **INPUT/OUTPUT** permettono lo scambio di dati tra l'utente e l'esecutore. Le istruzioni di **INPUT** permettono all'utente di comunicare i dati iniziali su cui l'esecutore deve effettuare l'elaborazione (variabili di input). Le istruzioni di **OUTPUT** permettono all'esecutore di comunicare i risultati dell'elaborazione all'utente (variabili di output).

La sintassi delle istruzioni di input/output dipende dalle caratteristiche del linguaggio di programmazione; per ora conveniamo di far riferimento ad esse in *linguaggio di progetto*<sup>3</sup> con "LEGGI" e "SCRIVI" rispettivamente. Così, per esempio

### LEGGI N

significa: "acquisisci un valore dall'esterno (il dato fornito dall'utente) e assegnalo alla variabile N", mentre

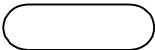


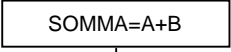

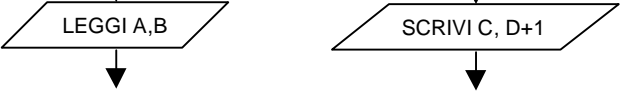
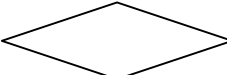
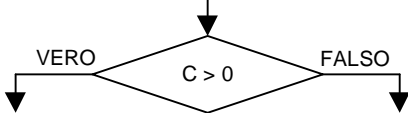
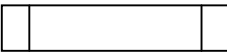
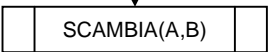

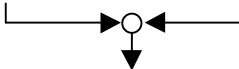
### SCRIVI N

significa: "prendi il valore della variabile N e comunicalo all'esterno (all'utente)".

### La descrizione delle istruzioni tramite i diagrammi a blocchi

I *diagrammi a blocchi* o *flow chart* costituiscono uno dei primi e più diffusi formalismi grafici per la descrizione degli algoritmi. Le istruzioni sono rappresentate tramite appositi simboli grafici, chiamati blocchi, uniti da archi orientati (cioè linee munite di frecce) che indicano la successione delle operazioni da svolgere nell'algoritmo. I diagrammi a blocchi sono privi di istruzioni di dichiarazione.

Le forme più comuni per i blocchi sono i seguenti:

Blocco	Simbolo	Esempi
Inizio/Fine		
Istruzione di assegnazione		
Istruzione di INPUT/OUTPUT		
Istruzione di controllo (espressione relazionale)		
Sottoalgoritmo		
Connettore (punto di raccordo)		

<sup>3</sup> Sottinsieme del linguaggio naturale privo di ambiguità che facilita la stesura dell'algoritmo (pseudocodice).

### Esempio

Problema: "Calcolare l'area di una circonferenza di raggio R".

Soluzione:

#### TABELLA DELLE COSTANTI

Valore	Tipo	Descrizione
3.14	reale	Valore di PI-GRECO

#### TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
R	reale	X			Raggio della circonferenza
AREA	reale			X	Area della circonferenza

Pseudocodice	Diagramma a blocchi	Linguaggio C++
<p>VARIABILI: R, AREA;</p> <p>INIZIO LEGGI R; AREA=R*R*3.14; SCRIVI AREA; FINE</p>	<pre> graph TD     INIZIO([INIZIO]) --&gt; LEGGI_R[/LEGGI R/]     LEGGI_R --&gt; AREA_CALC[AREA = R * R * 3.14]     AREA_CALC --&gt; SCRIVI_AREA[/SCRIVI AREA/]     SCRIVI_AREA --&gt; FINE([FINE])                     </pre>	<pre> #include &lt;iostream.h&gt;  main() { float R, AREA;    cin&gt;&gt;R;   AREA=R*R*3.14;   cout&lt;&lt;AREA; }                     </pre>

Nota: Le caratteristiche e la sintassi del linguaggio C++ verranno discusse e approfondite in classe.

### 3. Istruzioni di controllo

Le istruzioni di controllo vengono utilizzate per descrivere **azioni strutturate** quali la *selezione* di un cammino tra due possibili all'interno di un algoritmo e/o l'*iterazione* di un gruppo di istruzioni.

Le strutture di controllo fondamentali per la descrizione degli algoritmi sono le seguenti:

1. Sequenza
2. Selezione
3. Iterazione

Vedremo in seguito che tali strutture corrispondono ad istruzioni disponibili nella maggior parte dei linguaggi di programmazione di alto livello. Inoltre esiste il seguente teorema di Jacopini-Böhm 1966:

**"Ogni algoritmo può essere espresso con le sole tre strutture di controllo fondamentali"**

L'enunciato di questo teorema afferma che una volta individuato un algoritmo per un dato problema, per quanto possa essere complesso, lo si potrà sempre esprimere mediante tutte e sole le suddette strutture linguistiche. Per questo motivo alcuni linguaggi di programmazione non prevedono altre strutture oltre quelle indicate.

#### 1. La Sequenza

La sequenza è la struttura per la quale le istruzioni vengono eseguite solo nell'ordine in cui sono specificate. L'algoritmo per il *calcolo dell'area della circonferenza* visto sopra è descritto con una Sequenza formata da tre istruzioni che devono essere eseguite nell'ordine in cui sono specificate per ottenere il risultato corretto.

**Esercizi:**

Analizzare e codificare le soluzioni dei seguenti problemi:

- 1) *Dati i problemi relativi agli esempi 1.a, 2.a e 3.a della sezione precedente (parzialmente risolti), codificare le rispettive soluzioni in linguaggio di progetto e tramite Diagramma a Blocchi.*
- 2) *Calcolare il perimetro e l'area di un quadrato.*
- 3) *Dati il valore di un deposito bancario e il tasso di interesse annuo, calcolare gli interessi maturati dopo 25 giorni.*
- 4) *Per il lavoro di un operaio vengono registrati l'orario di entrata e l'orario di uscita sia al mattino che al pomeriggio. Calcolare il totale delle ore e dei minuti lavorati e la paga spettante, nota la paga oraria, per un giorno di lavoro.*
- 5) *Dati il valore di un deposito bancario e il tasso di interesse annuo, calcolare gli interessi maturati dopo N anni, M mesi e G giorni (semplificazione: mesi di 30 giorni).*

**2. La Selezione**

La selezione è una struttura linguistica che permette di:

- eseguire certe istruzioni al verificarsi di una certa condizione (selezione unaria o a una via)
- eseguire certe istruzioni al verificarsi di una certa condizione ed altre istruzioni se non si verifica (selezione binaria o a due vie).

La **condizione** è un'espressione *relazionale*, cioè un insieme di operandi e operatori opportunamente organizzati, che può assumere solo due valori: **vero** oppure **falso**.

Nella sua forma più semplice l'espressione relazionale descrive un confronto fra due quantità che, nel momento in cui viene effettuato dall'esecutore, può risultare soddisfatto (VERO) oppure no (FALSO). Le quantità coinvolte costituiscono gli operandi dell'espressione e il tipo di confronto viene stabilito da un apposito operatore relazionale. La tabella seguente riassume il significato dei simboli utilizzati per gli operatori più comunemente usati in linguaggio C:

Operatori relazionali del linguaggio C			
Simbolo	Descrizione	Esempio	Risultato
>	Maggiore	X>4	Vero se il valore di X è superiore a 4, falso altrimenti
>=	Maggiore o Uguale	X>=4	Vero se il valore di X è superiore o uguale a 4, falso altrimenti
==	Uguale	X==4	Vero se il valore di X è uguale a 4, falso altrimenti
!=	Diverso	X!=4	Vero se il valore di X è diverso da 4, falso altrimenti
<	Minore	X<4	Vero se il valore di X è inferiore a 4, falso altrimenti
<=	Minore o Uguale	X<=4	Vero se il valore di X è inferiore o uguale a 4, falso altrimenti

A partire da espressioni semplici è possibile costruire espressioni relazionali composte mediante i connettivi logici AND (congiunzione), OR (disgiunzione) e NOT (negazione). La tabella seguente descrive quelli del linguaggio C:

Connettivi logici del linguaggio C			
Simbolo	Descrizione	Esempio	Risultato
&&	Congiunzione logica	(X>4) && (X<6)	Vero solo se X vale 5
	Disgiunzione logica	(X<5)    (X>5)	Vero solo se X è diverso da 5
!	Negazione logica	!(X>4)	Vero solo se X è minore o uguale a 4

SELEZIONE UNARIA		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
SE (condizione) ALLORA INIZIO ... ... FINE	<pre>                     graph TD                         Start(( )) --&gt; Cond{condizione}                         Cond -- VERO --&gt; Circle(( ))                         Cond -- FALSO --&gt; Circle                         Circle --&gt; End(( ))                     </pre>	<pre>                     if( condizione )                     {                     ...                     ...                     }                 </pre>

**Nota:** Quando il corpo di una istruzione di controllo è costituito da una sola istruzione, in linguaggio di progetto non è necessario delimitarlo con INIZIO .. FINE.

### Esempio

Problema: "Invertire il segno di un numero intero N se è negativo".

Soluzione:

#### TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
N	intero	X		X	Il numero da elaborare

Pseudocodice	Diagramma a blocchi	Linguaggio C++
VARIABILI: N;  INIZIO LEGGI N; SE (N<0) ALLORA N = -N; SCRIVI N; FINE	<pre>                     graph TD                         Start([INIZIO]) --&gt; Read[/LEGGI N/]                         Read --&gt; Cond{N &lt; 0}                         Cond -- VERO --&gt; Process[N = -N]                         Process --&gt; Circle(( ))                         Cond -- FALSO --&gt; Circle                         Circle --&gt; Write[/SCRIVI N/]                         Write --&gt; End([FINE])                     </pre>	<pre>                     #include &lt;iostream.h&gt;                      main()                     { int N;                      cin&gt;&gt;N;                     if(N&lt;0)                     N = -N;                     cout&lt;&lt;N;                     }                 </pre>

SELEZIONE BINARIA		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
SE (condizione) ALLORA INIZIO ... ... FINE ALTRIMENTI INIZIO ... ... FINE		<pre>                     if( condizione )                     {                         ...                     }                     else                     {                         ...                     }                 </pre>

**Esempio**

Problema: "Determinare il maggiore fra due punteggi".

Soluzione:

TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
A	intero	X			Il primo punteggio
B	intero	X			Il secondo punteggio
MAX	intero			X	Il punteggio più alto

Pseudocodice	Diagramma a blocchi	Linguaggio C++
VARIABILI: A, B, MAX;  INIZIO LEGGI A; LEGGI B; <b>SE (A&gt;B)</b> <b>ALLORA</b> <b>MAX = A;</b> <b>ALTRIMENTI</b> <b>MAX = B;</b> SCRIVI MAX; FINE		<pre>                     #include &lt;iostream.h&gt;                      main()                     { int A, B, MAX;                          cin&gt;&gt;A;                         cin&gt;&gt;B;                         if(A &gt; B)                             MAX = A;                         else                             MAX = B;                         cout&lt;&lt;MAX;                     }                 </pre>

### Esercizi:

Analizzare e codificare le soluzioni dei seguenti problemi:

- 1) La scuola rimborsa il 15% del costo dell'abbonamento se lo studente abita in provincia, usa l'autobus ed è lontano almeno 20 Km dalla scuola. Alle stesse condizioni, se usa il treno il rimborso è del 10%. Calcolare l'ammontare del rimborso.
- 2) Sul prezzo di un prodotto viene praticato uno sconto del 3% se costa meno di 500 Euro e del 5% per prezzi superiori a 500 Euro. Calcolare il prezzo da pagare.
- 3) Date le età di due fratelli, determinare la differenza di età tra il fratello maggiore e il fratello minore.

### 3. L'iterazione

L'iterazione, detta anche *ciclo*, è una struttura che permette di ripetere una o più operazioni in sequenza per un numero finito di volte. Il gruppo di istruzioni da eseguire *iterativamente* è detto *corpo* del ciclo. Il numero di iterazioni dipende dal valore di un'espressione relazionale, detta *condizione di uscita* o di *terminazione*. L'esecutore ripete il ciclo fino a quando la condizione rimane vera. Affinché l'iterazione possa terminare, è necessario che il corpo del ciclo contenga almeno un'istruzione che, al momento opportuno, inverte il valore della condizione da vero a falso. Senza tale istruzione il ciclo verrebbe eseguito infinite volte (*loop infinito*).

Nella maggior parte dei casi, quando si usano i cicli, è necessario predisporre una o più variabili di lavoro da utilizzare come "contatore" o come "accumulatore":

- un **contatore** è una variabile intera che si usa per contare il numero di volte che si è eseguita una certa iterazione; di solito si inizializza a 0 oppure a 1 e poi si incrementata ad ogni iterazione fino a quando non raggiunge un *valore finale*, in accordo con la condizione del ciclo;
- un **accumulatore** è una variabile, non necessariamente intera, che si usa per conservare il risultato parziale di operazioni, quali somme o prodotti, successive; si imposta a 0 se si usa per accumulare somme, a 1 se si accumulano prodotti.

La posizione del corpo del ciclo rispetto alla condizione di terminazione caratterizza vari tipi di strutture iterative delle quali si parlerà in dettaglio più avanti nell'ambito dello studio di un linguaggio di programmazione reale. Per ora diamo una breve descrizione dei due tipi maggiormente usati: l'iterazione con controllo in **testa** e l'iterazione con controllo in **coda**.

**Iterazione con controllo in testa**

In questo tipo di iterazione la condizione di uscita è posizionata in testa al corpo del ciclo. Questo significa che l'esecutore prima controlla la condizione e poi, eventualmente, esegue le istruzioni del corpo (e ripete finché la condizione è vera). Se la condizione è falsa sin dall'inizio, le istruzioni del ciclo non vengono eseguite neanche una volta. In alcuni linguaggi di programmazione questo tipo di iterazione viene detta **ciclo while**.

ITERAZIONE CON CONTROLLO IN TESTA - CICLO while		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
FINCHE(condizione) INIZIO ... ... FINE	<pre>                     graph TD                         Start(( )) --&gt; Cond{condizione}                         Cond -- VERO --&gt; Body[...]                         Body --&gt; Start                         Cond -- FALSO --&gt; Exit(( ))                     </pre>	<pre>                     while( condizione )                     {                         ...                     }                 </pre>

**Iterazione con controllo in coda**

In questo tipo di iterazione la condizione di uscita è posizionata in coda al corpo del ciclo. Questo significa che l'esecutore prima esegue le istruzioni del corpo e poi controlla la condizione e, eventualmente, ripete finché la condizione è vera. Se la condizione è falsa sin dall'inizio, le istruzioni del ciclo vengono eseguite comunque una volta. In alcuni linguaggi di programmazione questo tipo di iterazione viene detta **ciclo do-while**.

ITERAZIONE CON CONTROLLO IN CODA - CICLO do..while		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
RIPETI INIZIO ... ... FINE FINCHE(condizione);	<pre>                     graph TD                         Start(( )) --&gt; Body[...]                         Body --&gt; Cond{condizione}                         Cond -- VERO --&gt; Start                         Cond -- FALSO --&gt; Exit(( ))                     </pre>	<pre>                     do                     {                         ...                     }                     while( condizione );                 </pre>

**Esempio**

Problema: "Dati i pesi di N oggetti, calcolare il peso medio".

Soluzione:

TABELLA DELLE VARIABILI

Nome	Tipo	INPUT	LAVORO	OUTPUT	Descrizione
N	intero	X			Numero di oggetti
P	reale	X			Peso di un oggetto
I	intero		X		Contatore dei pesi
S	reale		X		Somma dei pesi (accumulatore)
PM	reale			X	Peso medio

Codifica tramite il ciclo while		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
<p>VARIABILI: N, P, I, S, PM;</p> <p>INIZIO LEGGI N; S=0; I=0; <b>FINCHE( I&lt;N )</b> INIZIO LEGGI P; S=S+P; I=I+1; FINE PM=S/N; SCRIVI PM; FINE</p>	<pre> graph TD     Start([INIZIO]) --&gt; ReadN[/LEGGI N/]     ReadN --&gt; S0[S = 0]     S0 --&gt; I0[I = 0]     I0 --&gt; Cond{I &lt; N}     Cond -- VERO --&gt; ReadP[/LEGGI P/]     ReadP --&gt; Sum[S = S + P]     Sum --&gt; IncI[I = I + 1]     IncI --&gt; Cond     Cond -- FALSO --&gt; CalcPM[PM = S/N]     CalcPM --&gt; WritePM[/SCRIVI PM/]     WritePM --&gt; End([FINE])     </pre>	<pre> #include &lt;iostream.h&gt;  main() { int N, I;   float P, S, PM;    cin&gt;&gt;N;   S=0;   I=0;   while( I&lt;N )   {     cin&gt;&gt;P;     S=S+P;     I=I+1;   }   PM=S/N;   cout&lt;&lt;PM; }     </pre>

Codifica tramite il ciclo do..while		
Pseudocodice	Diagramma a blocchi	Linguaggio C++
<p>VARIABILI: N, P, I, S, PM;</p> <p>INIZIO LEGGI N; S=0; I=0; RIPETI INIZIO LEGGI P; S=S+P; I=I+1; FINE FINCHE( I&lt;N ); PM=S/N; SCRIVI PM; FINE</p>	<pre> graph TD     INIZIO([INIZIO]) --&gt; LEGGI_N[/LEGGI N/]     LEGGI_N --&gt; S_0[S = 0]     S_0 --&gt; I_0[I = 0]     I_0 --&gt; Conn(( ))     Conn --&gt; LEGGI_P[/LEGGI P/]     LEGGI_P --&gt; S_S_P[S = S + P]     S_S_P --&gt; I_I_1[I = I + 1]     I_I_1 --&gt; I_N{I &lt; N}     I_N -- VERO --&gt; Conn     I_N -- FALSO --&gt; PM_S_N[PM = S/N]     PM_S_N --&gt; SCRIVI_PM[/SCRIVI PM/]     SCRIVI_PM --&gt; FINE([FINE])     </pre>	<pre> #include &lt;iostream.h&gt;  main() { int N, I;   float P, S, PM;    cin&gt;&gt;N;   S=0;   I=0;   do   {     cin&gt;&gt;P;     S=S+P;     I=I+1;   } while( I&lt;N );   PM=S/N;   cout&lt;&lt;PM; }     </pre>

**Esercizi:**

Analizzare e codificare le soluzioni dei seguenti problemi:

- 1) Uno studente esamina le informazioni sulle verifiche scritte che ha fatto durante l'anno scolastico, e vuole contare quante sono le prove sufficienti.
- 2) Calcolare la somma dei primi 10 numeri interi successivi a un dato numero N.
- 3) Calcolare la somma dei primi N numeri pari successivi a un dato numero A.
- 4) Calcolare la somma dei quadrati dei primi N numeri.
- 5) Data una sequenza di numeri, contare quelli positivi e quelli negativi.
- 6) Su un insieme di articoli viene applicato il 15% di sconto. Dato il prezzo di ciascun articolo, calcolare la somma degli sconti.

### 1.2.3 Terzo passo – verificare la correttezza della soluzione

Una volta ottenuto l'algoritmo è necessario verificarne la correttezza. Un metodo di verifica molto efficace consiste nel simulare l'esecuzione dell'algoritmo tramite la **tabella di traccia** (*trace table*). A partire da istanze specifiche del problema, con l'ausilio di questa tabella è possibile eseguire *passo-passo* le istruzioni dell'algoritmo, e verificare ad ogni passo la correttezza dei valori assunti dalle variabili in gioco. In questo modo è possibile scoprire e correggere eventuali anomalie presenti nella logica risolutiva, tornando al *primo passo* dell'analisi. È importante sottolineare che ai fini della simulazione è necessario utilizzare istanze del problema che consentono di percorrere tutte le possibili ramificazioni dell'algoritmo, scegliendo opportuni valori iniziali per saggiare la bontà anche nei "casi limite". **Vedi esempi.**

