Unità didattica 6

DALL'ANALISI ALL'ESECUZIONE: PRIMI PROGRAMMI

SVILUPPO DI PROGRAMMI PER RAFFINAMENTI SUCCESSIVI (TOP-DOWN)

Dopo aver analizzato l'intero processo necessario per passare dalla situazione reale in cui si è generato il problema alla stesura del programma Pascal che lo risolve, nella presente unità didattica vengono proposti alcuni problemi che vi consentiranno di provare ad applicare compiutamente le nozioni acquisite, portandovi poco per volta a realizzare dei programmi con un approccio di tipo professionale.

Proposta di lavoro

Per effettuare un lancio pubblicitario, un commerciante di vino avvia una campagna di vendita esclusiva di Barbera in bottiglioni, in cui propone uno sconto per acquisti superiori ai 50 litri.

Progettare e codificare un programma che consenta di stabilire il totale che ogni cliente deve pagare.

Risposta

Analisi della situazione reale

Questa fase si compie presso l'utente finale (committente del programma) ed è necessaria per definire le specifiche di progetto.
In primo luogo è indispensabile esaminare l'ambiente di lavoro in cui il sistema di elaborazione dovrà essere calato, allo scopo di chiarire le interrelazioni tra l'operatore e il calcolatore.

Immaginiamo che, nel corso di una prima visita presso il negozio del vinaio e attraverso un colloquio con quest'ultimo, si siano appurate le seguenti modalità di vendita al singolo cliente:

① si riceve l'ordinazione, in numero di bottiglioni;

② viene calcolato il costo del vino, applicando eventualmente lo sconto;

③ se viene richiesta la consegna a domicilio viene applicato un sovrapprezzo fisso;

④ si riceve il denaro, si fornisce lo scontrino e si autorizza la consegna.

A questo punto occorre concordare con l'utente quali operazioni lasciare a suo carico e quali demandare al programma (minore è la complessità del programma, più bassi sono i costi).

Inoltre, nel successivo colloquio immaginiamo si sia concordato che :

- l'utente abbia una sufficiente dimestichezza con il sistema di elaborazione e sia in grado di gestire in proprio tutte le operazioni necessarie ad attivare il programma e interagire con esso;
- tutte le operazioni di cassa, cioè la ricezione del denaro nonché il calcolo e distribuzione del resto, sono completamente a carico dell'utente;
- le operazioni di calcolo del totale da pagare e l'eventuale arrotondamento sono a carico del programma.

Il terzo passo consiste nel descrivere in prima approssimazione la linea logica del programma, al fine di evidenziare eventuali elementi mancanti.

Nel nostro caso, riconosciamo che il programma deve:

① richiedere il numero di bottiglioni ordinati;

② calcolare il totale, comprensivo dell'eventuale sconto;

③ applicare l'eventuale sovrapprezzo per il trasporto;

④ effettuare l'eventuale arrotondamento del totale;

⑤ comunicare il risultato.

Da ciò emerge che per la realizzazione del programma il vinaio ci deve fornire dati precisi sulla percentuale di sconto da applicare, sul prezzo al litro del vino e sul sovrapprezzo relativo al trasporto.

Specifiche di progetto

Mentre le considerazioni precedenti non avevano un riscontro diretto sulla documentazione del programma, la definizione delle specifiche di progetto, oltre a rappresentare il punto di partenza per la costruzione dell'algoritmo risolutivo, risultano essere parte integrante del manuale per il manutentore che, ricordiamo, è parte integrante del programma stesso.

- il programma non è autopartente, ma deve essere lanciato ogni volta dall'utente;
- il programma fornisce il totale per un solo cliente e quindi deve essere rilanciato ogni volta;
- il prezzo del Barbera è di L. 2.000 al litro, IVA compresa e viene venduto esclusivamente in bottiglioni da litri 1,5 cad;
- per quantitativi superiori ai 50 litri viene praticato uno sconto del 10%;
- i vuoti sono a perdere;
- per la consegna a domicilio è previsto un sovrapprezzo di L. 5.000;
- il risultato deve essere arrotondato alle 100 lire;
- deve essere stampato esclusivamente uno scontrino di cassa con il nome della ditta "Bacco e soci" e il totale.

Costruzione del programma

Modello 1

Questo modello, prendendo spunto da quanto emerso nell'analisi, esprime, attraverso i commenti di programma, la descrizione della sequenza principale delle azioni che l'esecutore deve compiere per ottenere lo scopo del programma stesso.

```
program VINAIO;
 uses .....
 const .....
 var .....
begin
  {acquisizione del numerc di bottiglioni ordinato dal cliente}
  {calcolo del totale di vino ordinato espresso in litri}
  {calcolo del totale lordo}
```

```
  {calcolo dello sconto e correzione del totale lordo}
  {applicazione eventuale del sovrapprezzo per il trasporto}
  {arrotondamento alle 100 lire del totale generale}
  {comunicazione del risultato}
  {stampa dello scontrino}
end.
```

Dopo aver stilato questo modello, occorre esaminare attentamente i singoli commenti, per poter decidere quali tra essi possono essere trasformati direttamente in una sequenza di istruzioni Pascal e quali richiedono invece un approfondimento dell'analisi.

In questo caso, se esaminiamo i singoli commenti, ci accorgiamo immediatamente che l'unico punto che richiede un'ulteriore analisi prima di poter essere codificato in Pascal è quello relativo all'arrotondamento del totale alle 100 lire.

In pratica occorre prendere il valore del totale (ad esempio 7.565), dividerlo per 100 in modo da esprimerlo in centinaia (75,65), arrotondare questo valore all'intero più vicino (76) e infine moltiplicarlo per 100 in modo da ritornare all'unità di misura originaria (7.600).

Dato che la libreria di sistema mette a disposizione la funzione **round** che effettua automaticamente l'arrotondamento, tutta l'operazione può essere espressa da una formula del tipo:

$$\text{TOTALEARROTONDATO} = \text{round}(\frac{\text{TOT}}{100}) * 100$$

Modello 2

```
program VINAIO;
 uses Crt, Printer;
 const
  CAPACITABOTTIGLIONE=1.5;
  PREZZOUNITARIO = 2000;
  PERCENTUALEDISCONTO=20;
  SOVRAPPREZZOTRASPORTO = 5000;
 var
  NUMEROBOTTIGLIONI:integer;
  LITRI, TOTALELORDO, TOTALESCONTATO, SCONTO, TOTALE: real;
  TOTALEARROTONDATO: longint;
  RISPOSTA:char;
 begin
  Clrscr;     {procedura di pulizia video, contenuta nel modulo Crt}
  {acquisizione del numero di bottiglioni ordinato dal cliente}
  writeln('Inserire il numero di bottiglioni ordinati');
  readln(NUMEROBOTTIGLIONI);
  {calcolo del totale di vino ordinato espresso in litri}
  LITRI:=NUMEROBOTTIGLIONI*CAPACITABOTTIGLIONE;
```

```
{calcolo del totale lordo}
TOTALELORDO:=LITRI*PREZZOUNITARIO;
{calcolo dello sconto e correzione del totale lordo}
if LITRI<=50
        then  SCONTO := 0
        else  SCONTO := TOTALELORDO*PERCENTUALEDISCONTO/100;
TOTALESCONTATO:=TOTALELORDO-SCONTO;
{applicazione eventuale del sovrapprezzo per il trasporto}
writeln; {salta una riga sul video}
writeln('E" stata richiesta la consegna a domicilio? (s/n)');
readln(RISPOSTA);
if  RISPOSTA='s'
     then  TOTALE:=TOTALESCONTATO+SOVRAPPREZZOTRASPORTO
     else  TOTALE:=TOTALESCONTATO;
{arrotondamento alle 100 lire del totale generale}
TOTALEARROTONDATO:=(round(TOTALE/100))*100;
{comunicazione del risultato}
writeln('Il cliente deve pagare lire ',TOTALEARROTONDATO);
{stampa dello scontrino}
writeln(lst, 'BACCO & soci');
writeln(lst);      {salta una riga in stampa}
writeln(lst,NUMEROBOTTIGLIONI,' bottiglioni = litri ',LITRI:6:1);
writeln(lst);
writeln(lst,'Totale da pagare L. ',TOTALEARROTONDATO);
writeln(lst)
end.
```

Piani di prova

Una volta caricato il programma in macchina e, attraverso la compilazione, verificata la sua correttezza sintattica, occorre sottoporre il prodotto ad una serie di test per saggiarne la correttezza di funzionamento. In particolare bisogna predisporre un certo numero di dati di prova in modo che il processore esegua almeno una volta tutte le istruzioni contenute nel programma.

Il nostro programma presenta al suo interno due diramazioni: la prima riguardante lo sconto e la seconda la richiesta di consegna a domicilio.
Di conseguenza, un piano di prova che consenta di attraversare tutti i rami del programma può essere il seguente:

prova	n. bottiglioni	trasporto	risultato atteso
1	10	no	30000
2	10	si	35000
3	40	no	96000
4	40	si	101000

Manuale per l'utente

L'ultima fase consiste nella stesura del manuale per l'utente, che deve contenere tutte le informazioni che possono semplificare l'uso del programma e ridurre le possibilità di una gestione errata dello stesso.

Nome del programma:	VINAIO 25/9/90
Autore:	Baruggio P.
Funzioni svolte:	Calcolo del totale che il cliente deve pagare.
Note operative:	Alla richiesta che compare sul video inserire il numero di litri di vino ordinati. Il valore inserito deve essere intero, positivo e minore di 32767. Attenzione: la procedura non prevede controllo sui dati inseriti; eventuali errori si ripercuoteranno sul risultato.
Risultati prodotti:	Compare su video il totale, già scontato, comprensivo di IVA e viene stampato un semplice scontrino di cassa.
Precauzioni:	Prima di lanciare il programma assicurarsi che la stampante sia accesa e sia fornita di carta.

Alcune considerazioni

❶ Come appare evidente dalle specifiche di progetto, il programma non permette all'utente di variare direttamente i parametri prefissati (costo unitario, tasso di sconto, quantitativo minimo per ottenere lo sconto ecc.), che sono stati definiti nel codice come costanti.

Una soluzione alternativa poteva prevedere l'inserimento da parte dell'utente, a ogni esecuzione, dei valori relativi a questi parametri, che sarebbero stati trattati allora come variabili.

Nel nostro caso una soluzione di questo tipo sarebbe stata improponibile in quanto, poiché il programma deve essere rilanciato all'arrivo di ogni cliente, l'utente avrebbe dovuto perdere molto tempo per inserire continuamente gli stessi dati iniziali.

Si è preferito dichiarare i parametri come costanti simboliche in testa al programma, anziché usare i loro valori direttamente all'interno delle istruzioni; per consentire una più facile manutenzione del programma stesso.

Infatti, in questo modo, per modificare il valore di uno dei parametri costanti (ad esempio la percentuale di sconto) basta intervenire sulla sezione di dichiarazione, senza dover scorrere tutto il programma alla ricerca delle istruzioni che fanno riferimento a quel valore.

Così si riduce anche la possibilità di introdurre nuovi errori nel programma (è facile dimenticare una sostituzione o confondere tra loro parametri simili o addirittura uguali) quando si vanno ad apportare delle modifiche.

❷ La prima specifica di progetto ipotizzava il fatto che l'utente possedesse le competenze necessarie per interagire con il calcolatore e il programma, attraversando le fasi di caricamento, compilazione ed esecuzione del programma.

Nella realtà però accade spesso che tali competenze non siano patrimonio degli utenti finali dei programmi.
In questo caso è necessario mettere a punto un prodotto che possa essere attivato nel modo più semplice possibile.

Un primo passo consiste nel creare un programma che possa essere lanciato in esecuzione senza dover richiamare l'ambiente Turbo Pascal. Ciò significa produrre un programma scritto in linguaggio macchina e che quindi possa essere interpretato dal Sistema Operativo come un comando.

Questo risultato può essere ottenuto semplicemente attivando l'opzione di compilazione: "Destination Disk" con la quale viene creato un file eseguibile (.exe) con lo stesso nome del file sorgente.

La tecnica top-down

Dalle considerazioni emerse nelle unità didattiche precedenti e nell'esempio testé sviluppato, risulta chiaro che la programmazione non si riduce a un semplice processo di invenzione e di formulazione di algoritmi, ma richiede attenzione ai dettagli e l'uso di una *tecnica opportuna di analisi del problema* da risolvere.

Infatti, la scelta della condotta risolutiva dipende da molti fattori, tra cui le caratteristiche del problema in esame, il sistema di elaborazione a disposizione, il tempo e il denaro disponibili, l'impiego a cui è destinato il programma e così via.

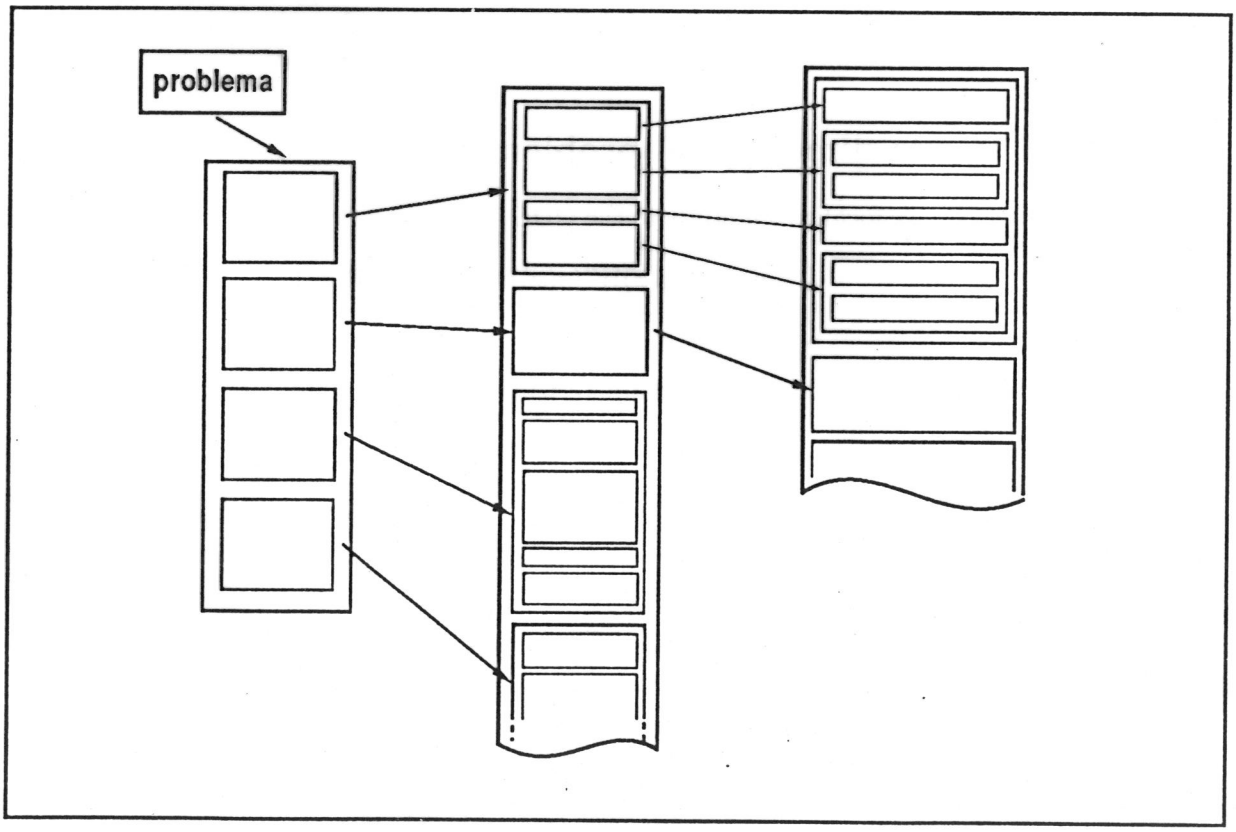
Nella progettazione, inoltre, si alternano, in modo non sempre distinto, fase di analisi e di scelte realizzative e quindi non è sempre facile mettere a punto l'algoritmo risolutivo e completarlo, partendo semplicemente dagli obiettivi prefissati. Di conseguenza, una buona tecnica di progettazione deve consentire al programmatore di sviluppare il proprio lavoro per passi e di focalizzare l'attenzione, di volta in volta, solo sui singoli nodi da risolvere, riducendone gradatamente la difficoltà.

In particolare, nella progettazione si parte sempre da un'analisi generale del problema e, dopo aver esplicitato le specifiche di progetto, si passa ad elencare la sequenza delle operazioni fondamentali che il programma deve compiere (*modello 1*).

Ciò permette di avere contemporaneamente una visione complessiva della condotta risolutiva e una definizione precisa, attraverso i commenti, dei nodi di difficoltà da affrontare.

Come abbiamo visto, la fase successiva consiste nell'esaminare i singoli commenti del *modello 1*, sviluppandoli o in codice Pascal (quando si è in grado di farlo) o in una serie di commenti più dettagliati e più facili da tradurre nel linguaggio di programmazione.
Naturalmente, in questa fase di maggior approfondimento, è possibile che emergano nuove situazioni problematiche che, una volta risolte, porteranno alla definizione di nuove specifiche di progetto di cui bisognerà tener conto nei passi successivi.
Questa operazione di raffinamento verrà ripetuta, passo dopo passo, fino a quando ogni singolo commento presente nel programma sarà stato completamente trasformato in istruzioni interpretabili dall'esecutore, cioè in codice Pascal.

Dato che il processo di scomposizione del problema ed il contemporaneo sviluppo e raffinamento del programma rappresentano un continuo approfondimento degli aspetti specifici, il modo di procedere è "dall'alto verso il basso" (o "dal generale al particolare") e viene generalmente indicato come tecnica di ***sviluppo top-down o per raffinamenti successivi.***



Proposta di lavoro

In un piccolo paese della Maremma all'inizio della primavera, nell'ambito della locale festa paesana, è stato organizzato un rodeo con iscrizione libera.

Il singolo concorrente può scegliere, prima di iniziare la propria prova, se cimentarsi nel cavalcare un toro (coeff. di difficoltà 1,75), un torello (coeff. 1,35) o un puledro (coeff. 1).

La prova viene seguita da 5 giudici, ciascuno dei quali esprimerà un punteggio compreso tra 1 e 10 con due cifre decimali: al concorrente verrà assegnato il punteggio che si ricava moltiplicando la media dei voti ottenuti per il coefficiente di difficoltà.

In caso di parità tra due concorrenti, dato che il premio in palio è unico, verrà dichiarato vincitore il concorrente che ha effettuato la gara per primo (gli animali erano più freschi e quindi la difficoltà maggiore).

Il direttore di gara, alla fine della prova di ogni concorrente, comunica a gran voce il punteggio complessivo ottenuto dal concorrente stesso e il nome ed il punteggio del concorrente che conduce la classifica provvisoria e, alla fine del rodeo, proclama il vincitore.

Sapendo che la gara si effettua solo se vi sono almeno 5 iscrizioni ed immaginando che il direttore di gara abbia a disposizione un Personal Computer, progettare un programma che lo aiuti a svolgere il suo compito.

Risposta

Analisi della situazione reale

Dato che il testo fornisce sufficienti indicazioni sulle modalità di svolgimento del rodeo, analizziamo direttamente il comportamento del direttore di gara al fine di determinare le operazioni che dovranno essere compiute dal programma.

Il direttore:

① prima della gara lancia l'esecuzione del programma;

② per ogni concorrente inserisce nel computer il nome, il tipo di cavalcatura prescelto e, alla fine della prova, i 5 voti dei giudici.
Inoltre, sulla base dei risultati forniti dal programma comunica, via microfono, il punteggio complessivo ottenuto dal concorrente, nonché il nome e il punteggio del concorrente che si trova momentaneamente in testa;

③ alla fine della gara proclama il vincitore.

Sulla base di questa prima analisi possiamo riconoscere che il programma deve:

① per ogni concorrente, fino alla fine della gara, richiedere i dati, calcolare il punteggio ottenuto, comunicarlo e, confrontandolo con quello del concorrente primo in classifica, determinare chi conduce a questo punto il rodeo;

② alla fine della gara comunicare il nome del vincitore e il punteggio ottenuto.

Specifiche di progetto

Dall'analisi precedente e da un colloquio con il responsabile del rodeo, abbiamo dedotto quanto segue:

- non si conosce a priori il numero di concorrenti partecipanti, dato che le iscrizioni sono aperte anche durante la gara, ma il rodeo viene aperto solo quando ci sono almeno 5 iscritti;
- le regole per l'assegnazione e il calcolo del punteggio sono quelle contenute nel testo;
- in caso di parità si devono applicare, per definire il vincitore, le norme contenute nel testo;
- la chiusura della gara è sotto il completo controllo del giudice unico;
- il programma, prima di definire il punteggio complessivo di un concorrente, deve consentire l'eventuale correzione dei voti inseriti;
- il direttore di gara non è interessato a conoscere l'intera classifica ma solo il nome di chi conduce la gara.

Costruzione del programma

Modello 1

```
program RODEO
 uses .....
 const .....
 var .....
 begin
  {acquisizione dati del primo concorrente}
  {calcolo del punteggio complessivo}
  {visualizzazione del risultato del primo concorrente, che conduce per ora la gara}
  {per ogni altro concorrente, fino alla fine della gara,
        acquisizione dati
        calcolo del punteggio
        comunicazione punteggio calcolato
        determinazione del primo classificato
        comunicazione dei dati del primo classificato}
  {proclamazione del vincitore}
 end.
```

Analizzando il primo modello ci rendiamo conto che il programma è suddiviso in tre parti:

- una di inizializzazione, che viene eseguita solamente all'inizio della gara;
- una ciclica che viene ripetuta per ogni concorrente;
- una conclusiva che viene eseguita una sola volta alla fine della gara.

In questa seconda fase di raffinamento, conviene puntare l'attenzione sulla rappresentazione del ciclo iterativo e in particolare sulle modalità di uscita dallo stesso. Dato che le iscrizioni possono avvenire anche durante lo svolgimento della gara, è evidente che la terminazione della stessa può essere decretata solo dal direttore e non può quindi essere gestita automaticamente dal programma.
E' dunque necessario che all'interno del ciclo sia presente un'operazione di input/output che consenta al direttore di comunicare al programma se la gara è finita o no.

Possiamo a questo punto scrivere il secondo modello, esplicitando il ciclo di **repeat ... until** e la condizione di uscita.

Modello 2

```
program RODEO
 uses .....
 const .....
 var
  RISPOSTA:char;
```

```
 begin
  {acquisizione dati del primo concorrente}
  {calcolo del punteggio complessivo}
  {visualizzazione del risultato del primo concorrente, che conduce per ora la gara}
  repeat  {per ogni altro concorrente}
   {acquisizione dati}
   {calcolo del punteggio}
   {comunicazione punteggio calcolato}
   {determinazione del primo classificato}
   {comunicazione dei dati del primo classificato}
   {gestione della condizione di uscita}
   writeln('E" finita la gara ? (s/n) ');
   readln(RISPOSTA);
  until RISPOSTA:='s' ;    {è finita la gara}
  {proclamazione del vincitore}
 end.
```

Con questo piccolo passo abbiamo eliminato un nodo di difficoltà: ora ci attende un problema più semplice, che siamo in grado di risolvere completamente.

Modello 3

```
program RODEO;
 uses Crt;
 var
  RISPOSTA:char;
  COEFFICIENTEDIFF, VOTO1, VOTO2, VOTO3, VOTO4,VOTO5:real;
  PUNTEG, PUNTEGVINCITORE: real;
  NOMECONCORRENTE, NOMEVINCITORE:string;
 begin
  Clrscr;
  {acquisizione dati del primo concorrente}
  writeln('Inserire il nome del primo concorrente');
  readln(NOMECONCORRENTE);
  writeln('Inserire il coefficiente di difficolta" della prova');
  readln(COEFFICIENTEDIFF);
  writeln('Inserire i cinque voti ottenuti');
  readln(VOTO1, VOTO2, VOTO3, VOTO4, VOTO5);
  {calcolo del punteggio complessivo}
  PUNTEG:=COEFFICIENTEDIFF*(VOTO1+VOTO2+VOTO3+VOTO4+VOTO5)/5;
  {visualizzazione del risultato del primo concorrente, che conduce per ora la gara}
  writeln('il concorrente ', NOMECONCORRENTE);
  writeln(' ha riportato ', PUNTEG:6:2, ' punti');
  NOMEVINCITORE:=NOMECONCORRENTE;
  PUNTEGVINCITORE:=PUNTEG;
  repeat  {per ogni altro concorrente}
    {acquisizione dati}
    writeln('Inserire il nome del nuovo concorrente');
    readln(NOMECONCORRENTE);
    writeln('Inserire il coefficiente di difficolta" della prova');
    readln(COEFFICIENTEDIFF);
```

```
    writeln('Inserire i cinque voti ottenuti');
    readln(VOTO1, VOTO2, VOTO3, VOTO4, VOTO5);
    {calcolo del punteggio}
    PUNTEG:=COEFFICIENTEDIFF*(VOTO1+VOTO2+VOTO3+VOTO4+VOTO5)/5;
  {comunicazione punteggio calcolato}
  writeln('il concorrente ', NOMECONCORRENTE);
  writeln(' ha riportato ', PUNTEG:6:2, ' punti');
  {determinazione del primo classificato}
  if PUNTEG>PUNTEGVINCITORE
     then
       begin
        NOMEVINCITORE:=NOMECONCORRENTE;
        PUNTEGVINCITORE:=PUNTEG
       end;
  {comunicazione dei dati del primo classificato}
  writeln('conduce la gara il concorrente ', NOMEVINCITORE);
  writeln( ' con ', PUNTEGVINCITORE:6:2, ' punti');
  {gestione della condizione di uscita}
  writeln('E'' finita la gara ? (s/n)');
  readln(RISPOSTA);
until RISPOSTA='s' ;     {è finita la gara}
{proclamazione del vincitore}
writeln('Ha vinto la gara il concorrente');
writeln(NOMEVINCITORE, ' con ', PUNTEGVINCITORE:6:2, ' punti');
repeat until keypressed;
end.
```

Piani di prova

Il nostro programma presenta al suo interno una diramazione e un ciclo: di conseguenza, un piano di prova che consenta di attraversare tutti i rami del programma può essere il seguente:

prova 1: si caricano i dati di 5 soli concorrenti di cui il primo è il vincitore;

prova 2: si caricano i dati di 5 soli concorrenti di cui l'ultimo è il vincitore;

prova 3: si caricano i dati di 5 concorrenti con 2 vincitori a pari merito.

Manuale per l'utente

Nome del programma:	RODEO 5/10/90
Autore:	Nova N.
Funzioni svolte:	Determinazione del vincitore del rodeo.
Note operative:	Alle richieste del computer inserire i dati necessari. Per il controllo di fine gara inserire s o n minuscole.

Attenzione: ogni risposta diversa da "s" verrà interpretata come "n".

Risultati prodotti: Visualizza il nome e il punteggio del concorrente che ha ultimato la gara e del primo in classifica.

Alcune considerazioni

❶ Se esaminiamo il programma, ci accorgiamo immediatamente che esiste una parte consistente di codice (quella relativa all'acquisizione dei dati, al calcolo del punteggio e alla sua visualizzazione) ripetuta sia dentro sia fuori dal ciclo iterativo. Viene allora spontaneo chiedersi se non è possibile portare anche l'acquisizione dei dati del primo concorrente all'interno del ciclo, eliminando la parte esterna.

La risposta è affermativa!

Ma per modificare il programma in tal senso bisogna tener conto che la prima volta in cui si incontra la struttura di selezione:

```
if PUNTEG>PUNTEGVINCITORE
   then
    begin
     NOMEVINCITORE:=NOMECONCORRENTE;
     PUNTEGVINCITORE:=PUNTEG
    end;
```

viene effettuato un confronto tra due variabili (PUNTEG e PUNTEGVINCITORE) di cui la seconda, se viene eliminata la parte di acquisizione esterna al ciclo, non si sa cosa contiene. Occorre quindi, per lo meno, inizializzare tali variabili con dei valori opportuni che non pregiudichino il buon funzionamento del programma.

Quest'ultimo allora può essere modificato nel modo seguente:

```
program RODEO;
 .....
begin
 Clrscr;
  {inizializzazione delle variabili relative al vincitore}
 NOMEVINCITORE:=' ';
 PUNTEGVINCITORE:=-1;

  repeat  {per ogni concorrente}
  .....
 until RISPOSTA='s';
 .....
end.
```

Con questa modifica, inoltre, il programma può funzionare anche nel caso in cui alla gara partecipi un solo concorrente.

SUGGERIMENTI PRATICI SULL'USO DEL TOP-DOWN

Mentre per realizzare il primo programma di questa unità didattica abbiamo usato due modelli, per sviluppare il secondo abbiamo ritenuto necessario ricorrere addirittura a tre modelli.

Gli stessi programmi affrontati da altri sono stati realizzati attraverso un numero diverso di modelli; voi stessi, probabilmente, avete attraversato un numero di passi diverso dal nostro.

Ci si potrebbe chiedere allora con quale criterio si deve definire il numero di modelli da utilizzare nella stesura di un programma.

La risposta è sibillina: *tanti quanti ne servono!*

Non esiste infatti un numero preciso o un numero calcolabile di modelli da usare, dato che la decisione sulla quantità di raffinamenti da porre in atto dipende esclusivamente dalle capacità e dall'esperienza del singolo programmatore.

La cosa importante **è procedere per piccoli passi,** in modo da rendere semplice il passaggio ad un modello successivo.

Ad esempio, nell'esercizio del rodeo, nel passaggio dal *modello 1* al *modello 2* ci siamo concentrati su un **unico** nodo di difficoltà (lo sviluppo del ciclo iterativo), trascurando tutto il resto e siamo riusciti così a costruire **facilmente** un nuovo modello in cui questo nodo di difficoltà era completamente risolto.

In questo modo abbiamo reso più semplice ciò che resta da risolvere, dato che, rispetto al problema originale, siamo riusciti ad eliminare un nodo di difficoltà.

Risulta quindi evidente che ripetendo più volte il processo di raffinamento, si risolvono a ogni passo dei problemi limitati (e quindi di facile soluzione) e si riduce contemporaneamente la difficoltà del problema complessivo (che risulta anch'esso più facile da risolvere), fino a trasformarlo in uno banale.

E' importante quindi evitare la tentazione a saltare dei modelli, poiché, il tempo necessario a effettuare un unico passo complesso può essere maggiore di quello necessario alla stesura di diversi modelli tra loro molto vicini.

ESERCIZI

1. Gioco dei fiammiferi: "su un tavolo vengono disposti 11 fiammiferi allineati. Due giocatori devono raccogliere alternativamente 1, 2 o 3 fiammiferi. Perde il giocatore che è obbligato a raccogliere l'ultimo fiammifero rimasto

le Programmazione

attività che permette di affrontare in modo sistematico l'ideazione e la realizzazione del software

I circuiti elettronici dei calcolatori sono in grado di eseguire operazioni estremamente semplici, calcoli aritmetici o logici tra interi byte o word.
Le operazioni elementari sono poi combinate in modo complesso per produrre operazioni complesse

L'insieme ordinato di istruzioni elementari impartito ad un calcolatore per svolgere un compito prende il nome di programma

Seguendo le istruzioni impartite dal programma il computer produce dati in output manipolando i dati di input

Le istruzioni e l'ordine delle istruzioni in un programma devono essere dettagliate ed inequivocabili

ISTRUZIONI l'insieme di istruzioni eseguibili da un elaboratore è specifico della CPU in uso

linguaggio Macchina { È uno insieme di semplici istruzioni codificate in codice binario }

Il linguaggio macchina è di difficile utilizzo e specifico della CPU

Per la semplificazione della scrittura di programmi è stata introdotta ASSEMBLER che rappresenta le istruzioni del linguaggio macchina mediante codici mnemonici in forma testuale.

Un programma scritto in ASSEMBLER viene poi tradotto in linguaggio macchina da apposito codificatore

detti ASSEMBLATORI. Tra tutti i li-p-eppi di programmazione
L'ASSEMBLER è quello più vici-o al li-p-eppio macchi-e

DIFETTI: è difficile scrivere programmi molto complessi
ogni diverso CPU richiede un diverso programma (mancanza di portabilità), impossibilità di utilizzo del software su elaboratori distinti)

Portabilità e semplicità nella scrittura sono ottenuti utilizzando i
Linguaggi di alto livello
che consentono di descrivere la sequenza di operazioni in modo più vici-o al li-p-eppio comune-
Scritto un programma in uno di questi li-p-eppi esso viene tradotto in li-p-eppio macchi-e da altri programmi detti
interpreti o compilatori (ognuno di ogni comp)

Il più-o li-p-eppio di alto livello è stato il FORTRAN (FORmula TRANslator) ideato per l'uso scientifico 54-57 da J. Backus
Gradualmente seguirono altri li-p-eppi
Algol, Cobol APL ('60) LISP ('62) SNOBOL PL/I ('65)
('71) PASCAL e Prolog, C ('74), ADA ('79) ... C++ ('85)
Java ('94) ed altri ancora-

Il PASCAL fu sviluppato da NIKLAUS WIRTH dell'univ. di Zurigo
il più-o ed i-corporò in modo coerente i concetti
della PROGRAMMAZIONE STRUTTURATA

Prima di procedere ad illustrare i concetti base della programmazione ricordiamo la organizzazione della memoria di un calcolatore

Memoria centrale (RAM) aggregato di byte o celle o locazioni
a ciascuna delle è associato un numero progressivo detto indirizzo

il processore localizza le celle attraverso l'indirizzo e può di scrivere o leggere un dato ③

(si utilizzano talvolta più celle per le costruzione di word 2,4,8 byte)

Un byte è dunque caratterizzato da

1 indirizzo fisico
1 dato (un numero memorizzato al suo interno)

Registri all'interno del processore vi sono delle celle di memoria dette registri
I dati vengono prelevati dalla RAM dentro i registri e poi la ALU li manipola e riscrive i dati nei registri e poi di nuovo in memoria centrale

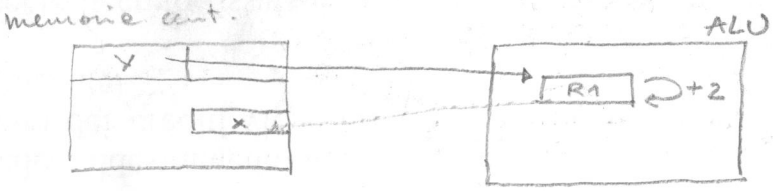
Memorie di Massa normalmente basate su tecnologie di tipo magnetico per la memorizzazione permanente
byte e word in memorie di massa vengono organizzati appunto in record detti file

Programma scritto in file trasferito dal comp. ---
Un esempio di partizione delle memorie
Linguaggio assembler versione simbolica del codice macchina
le operazioni sono indicate con nomi mnemonici
indirizzi di memoria con simboli

Differenza con linguaggio macchina una cella di memoria viene associata ad un nome invece che al suo indirizzo fisico

ex istruzione $x = y + 2$

```
LOAD   Y, R1   ; carica il valore di y nel registro R1
ADD    2, R1   ; somma 2 al valore in R1
STORE  R1, X   ; memorizza il valore in R1 nella cella X
```



Assemblatore traduce ogni istruzione in linguaggio macchina

la prima fase è la costruzione della simbol table in cui ai simboli ut. si associano gli indirizzi (riserva spazio)

traduce ogni singola ist. in linguaggio macchina

Nei linguaggi di alto livello questi passaggi vengono sintetizzati in espressioni leggibili dal programmatore ad esempio

$X = Y + 2$ in fortran.

VARIABILI COSTANTI e OPERATORI

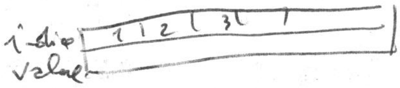
Un'astrazione della nozione di cella di Memoria è costituita dal concetto di

Variabile caratterizzata da un Nome (indirizzo) e dal valore che possono assumere via via durante l'esecuzione delle istruzioni di assegnamento cont. nel programma.

Come abbiamo visto lo spazio di memoria necessario per una variabile dipende dai ~~tt~~ valori che essa assume → TIPI DI DATO

i più diffusi

- BOOLEANO (boolean): possono assumere solo valore vero o falso
  1 bit * non indirizzabile si usa un byte (operatori AND, OR, NOT, ...
- INTERO (Integer): qualunque valore intero pos. o neg.
  L'archit. limita il ~~max~~ max mod oltre il quale overflow
  (operatori +, -, *, / o DIV) (divis. quoziente e resto Pascal resto MOD)
- Reale (Real): numeri in virgola mobile (con precisione singola doppia, o quadrupla di precisione)
- Carattere (Char) il tipo Char in Pascal un qualsiasi valore ASCII
- array (1,..n, 1...m)
  var schede : array(1...n)



Costanti valori fissi esplicitamente scritti nei programmi

numeri A = 2.156

Pascal stringhe A = 'PIERO'

esempio

```
VAR
   aperto : boolean ;
   posizione : integer;
   distanza : real ;
   lettera : char;
```

} dichiarazione di tipo

```
aperto := true ;
posizione := 123 ;
distanza := 1E-16 ;
lettera := 'char';
```

} assegnazione + istruzioni di assegnazione le più semplici

In generale un istruzione di assegnazione coinvolge variabili contenti operatori esiti --- es $\boxed{x = x + 1}$

il vecchio valore contenuto in x viene sostituito dal vecchio incrementato di 1

STRUTTURE DI CONTROLLO : costrutti con la funzione di definire l'ordine secondo cui singole istruzioni o gruppi di esse vengono eseguite

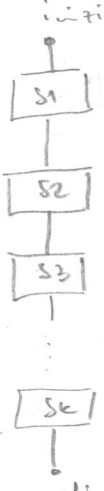
Tre generi di strutture di controllo { sequenza, selezione, ripetizione

Si dicono linguaggi strutturati quelli che utilizzano in modo esclusivamente istruzioni di controllo di questo tipo

esempi

- Sequenza se S1, S2, ---, Sk sono istruzioni (o statement) la sequenza

```
S1 ;
S2 ;
---
Sk;
```



il controllo "fluisce" sequenzialmente attraverso la lista

rompere l'uovo sbatterlo oppure le fasi mescolare oppure cotte

unico pt. d'inizio e unico pt. finale

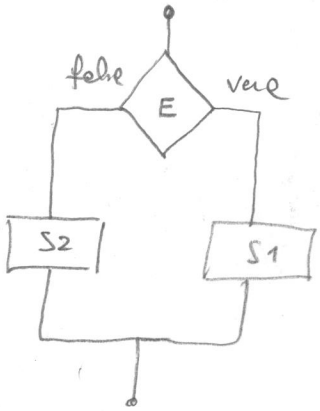
In Pascal il ; indica la separazione tra due istruzioni e le concatena -

- Selezione si controlla la direzione del flusso attraverso la verità di un'espressione booleana E

Se E vera allora S1 altrimenti S2

esempio quotidiano "se vuoto il bucato allora sciacquatelo altrimenti strizzatelo e poi versatelo altrimenti versate direttamente l'olio"

Anche se



unica entrata
uscita

una ramificazione delle istruzioni

esempio

a)
```
IF a>0
THEN a:=a+1
ELSE b:=a;
```

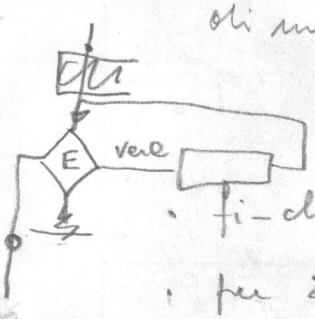
```
IF a>0
THEN a=a-1;
```

```
IF veicolo
case veicolo of
  1 : tone:=0;
  2 : tone:=10;
  3 : tone:=20;
END;
```

```
IF veicolo=1
Then
  tone:=0
Else
  IF veicolo=2
  THEN
    tone:=10
  ELSE
    tone:=20
```

- CICLO CONDIZIONATO

se E è una espressione si può condizionare la ripetizione di una sequenza di istruzioni



finchè E esprime (S1, ..., Sk)

- finché non si emulpe e cuocere a fuoco lento
- per 20 minuti cuocere

ex somma dei primi cento numeri

```
sum=0
FOR i:1 to 100 DO
  sum = sum+i;
```

o

```
i=1
sum=0
While i<=100 DO
BEGIN
  sum = sum+i
  i=i+1
END
```

STRUTTURA

① INTESTAZIONE vien spec. il nome del prog.
② Dichiarazioni descritti in modo formale le variabili su cui il prog. deve operare
③ Istruzioni descritte le seq. di azioni da compiere tramite l'utilizzo delle strutture di controllo

~~prog~~ PROGRAM nome del programma; ← nelle ultime versioni ipersw. comp.
& intestazione

[DICHIARAZIONI]

begin

[istruzioni]

end. ← chiude la sequenza di azioni del programma

Sezione DICHIARAZIONI

tre parti { dichiarazione dei moduli usati dal Programma
USES
dichiarazione delle costanti
CONST
Dichiarazione delle variabili
VAR

moduli insieme di sottoprogrammi di utilità richiamabili dall'interno dei programmi

ex
1) modulo Crt proc. e funz. per l'uso del video
2) modulo Printer gestione stampante
3) " Graph grafica dello schermo

~~ex~~ sintassi

USES nome mod., nomemodulo, ---, nome modulo;

Tale dichiarazione può essere o meno → procedure standard.

Sezione costanti

ex

~~Dichiarazione delle costanti~~ → identificatore

PIGRECO = 3.14;
SALUTO = 'ciao';
AngoloGiro = 360;

identificatore { il primo carattere non è lettera { 1pi no ; non no MOTTO sì
il nome non può cont. spazi bianchi

Variabili

VAR
nome variabile : tipo ;
nome variabile, ..., nome variabile : tipo ;

tipo { real reali + , - , * , / divisione
integer interi oppure longint (int -32768 e 32767 con 2 DIV MOD resto
char + ASCII -2147483648 e 2147483647)
string sequenze di car. 1-255
operatori + concatenazione
boolean

Istruzioni

tra i delim. begin , end oppure ;

→ { espressioni
operazioni di i-p. uscite
costrutti di controllo

ASSEGNAZIONE

nome VAR := espressione
↖ combinazione di op. di e op. ri per funzioni.
precedente i-d de parent. tonde

operatori di relazione

= (uguale), <> diverso, >, <, >=, <=

per op di tipo carattere → rif ASCII

Priorità

not					↑ priorità
*	/	div	mod	and	
+	-	or			
=	<>	<	>	<=	>=

...cedure I/O

INPUT: readln → acquisisce dati da tastiera

```
readln (variabile);
readln (var1, var2, ...);
```

→ sep spazio bianco tenere bott.
il test per andare a capo

quindi si ferma e aspetta immissione

OUTPUT; writeln (cost. espressioni, variabili);

ex

```
Somma := 50
Media := 1.5
writeln (Media)          writeln (somma media)

      1.5                501.5
                         ↑ senza spazio

writeln ('Media = ', Media, ' Somma = ', Somma)
```

per ott. la stampa su carta si usa ~~la~~ il modulo printer

```
program STAMPA;
 uses Crt, Printer;
 begin
   writeln ('Questo sul video')
      "     (lst, 'e questo stampante')
 end.
```

Visualizzazione dati reali

notazione esp floating point $\pm$ mantissa $\times 10^{\pm \text{esponente}}$

Prob. non univoco

... $0.01234\ 10^5$, 0.1234×10^4, 1.234×10^4, $12.34\ 10^3$ ...

↑ rapp E   ↖ forma normalizzata

ex

```
A := 0.002;
writeln(A);     →  2.0000000 E-3      (10)
B := 2000.12;
write (B);      →  2.00012000 E+3     (10)
                                          writeln(A; 18:0)
                                          ...12
writeln (R : n : m);         A := 12.1234;
          n = tot caratteri  writeln (A);    1.212340 E+1
          m = cifre dopo virg.
```

istruzioni strutturate { sequenze, selezione, iterazione

ciascuna struttura {
1) contiene una o più istruzioni
2) è vista come unico blocco logico
3) può cont. una o più ist. strutturate
~~4)~~

sequenza ~~se~~ seq. esec. secondo l'ordine indicato

```
begin
  ist 1
  ist 2
  ...
  ist n
end.
```

ex

```
begin
  A := 10
  readln (B)
  C := A+B;
end;
```

Selezione

```
if condizione
  then
    sequenza 1
  else
    sequenza 2;
```

iterazione
For Do repeat until

Funzioni matematiche di sistema — funz. di libreria Pascal

radice quadrata	$\sqrt{x}$	sqrt (x) → real (↑ real)
al quad	x^2	sqr (x) ← stesso tipo
exp	e^x	exp (x) ← real
log	$\log_e x$	ln (x)
valore ass.	$\lvert x \rvert$	abs (x) ← stesso tipo
Parte intera	int (x)	int (x) → real
fraz.		frac (x) → real
Troncamento		trunc (x) ↔ longint
Arrot. int più vicino		round (x) longint
~~Rip~~ seno cos arctg		sin (x), cos (x), arctan (x) → real

zioni e procedure per le stringhe

1) lunghezza length (STR) → integer

```
STR := 'lunghezza media'
~~length (STR)~~
lunghezza = length (STR)
writeln (lunghezza);        15
```

2) copy (STR, POSIZIONE, N)

restituisce la sottostringa in STR formata da N caratteri a partire da Posizione

ex
```
STR := 'PPLUTTO';
STR1 := copy (STR, 2, 3);      - STR1 = PLU
```

3) delete (STR, POSIZIONE, N)

```
STRINGA := 'concorso';
delete (STRINGA, 3, 3)      = stringa vale corso
```

4) insert (STR1, STR2, POSIZIONE)

```
STRINGA1 = 'dido'
STRINGA2 := 'enti')
insert (ST1, ST2, 4)
```

ST1 vale candidati

commenti

vecchia tre {...} o (*...*)