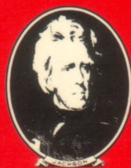
PROGRAMMARE IN PASCAL

Daniel-Jean David
e Jean-Luc Deschamps

P.S.I.



GRUPPO EDITORIALE JACKSON

EDIZIONE ITALIANA

PROGRAMMARE IN PASCAL

di

DANIEL-JEAN DAVID

e

JEAN-LUC DESCHAMPS

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Daniel-Jean David insegna informatica gestionale all'Universita' di Parigi-1 Pantheon-Sorbonne e applicazioni dei microprocessori all'ENSAM-Parigi. Si occupa di grafica, di tecniche di interfacciamento e di sistemi multiuso; inoltre e' uno specialista del microprocessore 6502 e ha tenuto numerosi seminari sull'argomento con particolare riferimento al KIM, al SYM e al PET/CBM.

Jean-Luc Deschamps insegna applicazioni dei microprocessori all'ENSAM-Parigi nei corsi DEA e cura le tesi di laurea degli studenti sullo stesso argomento. Egli inoltre si occupa di macchine utensili a controllo numerico, di controllo della produzione e di corsi di formazione professionale.

Traduzione: Rita Bonelli
Grafica e impaginazione: Francesca di Fiore
Marta Menegardo
Copertina: Marcello Longhini
Responsabile editoriale: Roberto Pancaldi

*Copyright 1982 Gruppo Editoriale Jackson

Tutti i diritti sono riservati. Nessuna parte di questo libro puo' essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Prima edizione: maggio 1982

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

P R E F A Z I O N E

Tradurre questo libro e' stato per me un piacevole lavoro, infatti ho potuto apprezzare molto due delle sue principali caratteristiche, e cioe', la chiarezza espositiva e lo sforzo costante di esporre i diversi argomenti con assoluta obiettivita'.

Leggendo il libro si avverte ad ogni pagina la consolidata esperienza informatica e didattica degli autori, che ci viene confermata dalle brevi note sulla loro attivita' in seconda di copertina.

Il libro, come viene del resto ribadito nell'introduzione, e' rivolto a tutti coloro che gia' sanno qualcosa di informatica. Indubbiamente prima di leggerlo con profitto e' necessario possedere almeno i concetti generali che sono alla base dell'informatica. Pero', a mio avviso, il libro puo' essere molto utile anche per le persone che, con poca esperienza di altri linguaggi, desiderano affrontare il Pascal. Queste non potranno apprezzare in pieno i frequenti riferimenti agli altri linguaggi simbolici, di cui il libro e' ricco, ma potranno apprendere il Pascal, grazie anche ai numerosi esercizi con soluzione finale.
Quest'ultimo aspetto del libro e' veramente notevole; ogni argomento e' seguito da numerosi esempi e da opportuni esercizi adatti a controllare lo stato dell'apprendimento prima di proseguire.

Ritengo che questo libro andrebbe usato nelle scuole medie superiori e nelle universita', come libro di testo per apprendere il Pascal servendosi di un Personal.

Un grazie quindi agli autori per la loro opera.

Rita Bonelli

S O M M A R I O

INTRODUZIONE.. 1

CAPITOLO 1 - LA PROGRAMMAZIONE STRUTTURATA
1.1. Scopo della programmazione strutturata......... 3
1.2. Regole della programmazione strutturata........ 5
1.3. Limiti della programmazione strutturata........ 10

CAPITOLO 2 - SGUARDO D'INSIEME SUL LINGUAGGIO PASCAL
2.1. Premessa....................................... 13
2.2. Comandi e dichiarazioni........................ 13
2.3. Istruzioni eseguibili.......................... 14
2.4. Dichiarazioni.................................. 16
2.5. Altri vantaggi del PASCAL...................... 20
2.6. I caratteri del PASCAL......................... 22
2.7. Punteggiatura e impaginazione.................. 22

CAPITOLO 3 - ISTRUZIONI SEQUENZIALI
3.1. Premessa....................................... 25
3.2. Istruzioni di lettura.......................... 25
3.3. Istruzioni di scrittura........................ 28
3.4. Istruzione aritmetica di assegnazione.......... 31
3.5. Gli operandi................................... 33
3.6. Le funzioni.................................... 36
3.7. Riepilogo...................................... 37
3.8. Struttura generale di un programma............. 38
3.9. Tipi standard.................................. 39

CAPITOLO 4 - ISTRUZIONI DI STRUTTURAZIONE
4.1. IF...THEN...ELSE............................... 43
4.2. I cicli WHILE e REPEAT......................... 46
4.3. Il ciclo FOR................................... 48
4.4. CASE... 51
4.5. GOTO... 52

CAPITOLO 5 - TIPI DI DATI
5.1. Premessa 55
5.2. Tipi scalari................................... 56
5.3. Tipi definiti mediante lista................... 57

5.4. Tipi definiti mediante intervallo.............. 59
5.5. Tipi strutturati.............................. 60
5.6. TYPE ARRAY.................................... 61
5.7. Prodotto scalare di due vettori 64
5.8. Ricerca di un elemento in una tabella 65
5.9. Ricerca dicotomica 66
5.10. Ordinamento per smistamento 67
5.11. Tabelle PACKED 68
5.12. Tabelle multidimensionali.................... 70
5.13. TYPE RECORD.................................. 72
5.14. Istruzione WITH 73
5.15. Registrazioni con varianti................... 74
5.16. TYPE SET..................................... 75
5.17. Operazioni sugli insiemi..................... 77

CAPITOLO 6 - PROCEDURE E FUNZIONI
6.1. Premessa 83
6.2. FUNCTION 86
6.3. PROCEDURE 87
6.4. Variabili locali e globali 90
6.5. Portata degli identificatori 92
6.6. Argomenti delle FUNCTION e delle PROCEDURE 94
6.7. La recursivita'............................... 96
6.8. La Torre di Hanoi 98
6.9. PROCEDURE FORWARD e PROCEDURE EXTERN 103

CAPITOLO 7 - FILES SEQUENZIALI
7.1. Premessa 107
7.2. Il tipo FILE 107
7.3. Scrittura dei file 108
7.4. Lettura dei file 109
7.5. Dichiarazioni programma 110
7.6. I file standard 110
7.7. Applicazioni di gestione 111
7.8. Il tipo TEXT 113

CAPITOLO 8 - GESTIONE DINAMICA DEI DATI
8.1. Introduzione alle strutture di dati 115
8.2. La pila 117
8.3. La fila d'attesa 120
8.4. Le liste 121
8.5. I grafi 122
8.6. Alberi 124
8.7. Trattamento dinamico dei dati 126
8.8. Liste dinamiche 129

CAPITOLO 9 - CONCLUSIONI: IL PUNTO SUL PASCAL
9.1. Criteri per la scelta di un linguaggio......... 135
9.2. Trattamenti permessi 135
9.3. Facilita' di scrittura 136
9.4. Diffusione del linguaggio 138
9.5. Portabilita'................................... 138
9.6. Conclusioni.................................... 140
9.7. Utilizzo sui Personal 141
9.8. Utilizzo nell'insegnamento 141
9.9. Utilizzo professionale 141
9.10. La questione del rendimento 142

APPENDICE A - PROGRAMMAZIONE STRUTTURATA IN BASIC...... 145

APPENDICE B - LE PAROLE CHIAVE DEL PASCAL.............. 153

APPENDICE C - DUE REALIZZAZIONI DEL PASCAL:
- IL PASCAL UCSD......................... 157
- IL PASCAL C.B.M......................... 161

APPENDICE D - SOLUZIONI DEGLI ESERCIZI................. 165

APPENDICE E - BIBLIOGRAFIA............................. 193

APPENDICE F - INDICE DEI PROGRAMMI..................... 195

I N T R O D U Z I O N E

Il linguaggio di programmazione Pascal suscita un interesse sempre crescente.
Si tratta di un entusiasmo passeggero, o, invece, del segno premonitore di una prospera carriera?

Per prima cosa, ha esso le qualita' necessarie per imporsi?

A credere ai suoi entusiasti sostenitori, il Pascal reca vantaggi tali, che alla domanda precedente non si puo' che rispondere affermativamente. Il Pascal sarebbe la panacea per risolvere tutti i problemi della programmazione.

Per quanto ci riguarda, noi pensiamo che non esistono panacee in informatica. Lo scopo di questo libro e' di fare il punto sui pregi ed i difetti del linguaggio Pascal.

Per i lettori che possiedono gia' un Personal e che sanno gia' programmare in Basic questo libro dovrebbe servire per fornire risposta alle seguenti domande:

. Quali sono i vantaggi del Pascal rispetto al Basic?

. Questi vantaggi giustificano l'acquisto di un interprete o di un compilatore Pascal?

Per i lettori che non possiedono ancora un Personal e stanno decidendone l'acquisto, la questione e' ancora piu' cruciale:

. Il Pascal e' assolutamente necessario per le applicazioni che io desidero realizzare?

. Devo per forza scegliere un calcolatore che disponga del Pascal?

Naturalmente, solo l'esame e la discussione approfondita delle vostre applicazioni possono determinare la vostra decisione. E questo e' quello che succede per qualunque decisione nel campo dell'informatica.

Lo scopo di questo libro e' quello di essere sufficientemente obiettivo in modo da mettervi in grado di prendere decisioni razionali e non passionali.

I vantaggi del Pascal (e sono numerosi) saranno descritti nel contesto delle applicazioni dove sono utilizzati. Per contro, verranno anche indicate le situazioni dove questi

non sono indispensabili.

Non si comincera' a discutere del Pascal senza descrivere la dottrina da cui il linguaggio ha avuto origine: la Programmazione Strutturata. Questo e' l'argomento del Capitolo 1.

Dal momento che non si puo' giudicare un linguaggio senza averne una buona conoscenza, la parte centrale dell'opera e' dedicata allo studio del Pascal.

Dopo una panoramica generale del linguaggio e delle sue principali caratteristiche (Capitolo 2), passeremo in rassegna le istruzioni sequenziali (calcolo e ingresso/uscita nel Capitolo 3), le istruzioni per strutturare i programmi (controlli, cicli, interruzione di sequenza nel Capitolo 4), i problemi dei blocchi e delle procedure (Capitolo 6).

Il Pascal e' particolarmente ricco riguardo ai tipi di dati che puo' trattare; questo e' l'argomento dei Capitoli 5, 7 e 8.

In conclusione l'originalita' del Pascal viene messa in luce confrontandolo con gli altri linguaggi concorrenti.

Oltre alle solite Appendici (lista delle parole-chiave, soluzione degli esercizi, bibliografia), l'appendice A vi mostra come , in un certo modo, e' possibile fare della programmazione strutturata in Basic.

Si suppone che il lettore conosca le nozioni fondamentali della programmazione per mezzo della pratica di un linguaggio di programmazione evoluto tipo Basic, Fortran o PL/1.

CAPITOLO 1

LA PROGRAMMAZIONE STRUTTURATA

1.1. SCOPO DELLA PROGRAMMAZIONE STRUTTURATA

Pur senza arrivare a dire che un programma e' una cosa "viva", si constata che ogni programma ha una storia ed una evoluzione.

Un programma deve essere "messo a punto" dopo la prima stesura; esso infatti non funziona mai, nella maggior parte dei casi, alla prima prova e sono quasi sempre necessarie delle modifiche nel corso delle prove successive.

Inoltre possono diventare necessarie successive modifiche per adattare il programma a nuove situazioni. Questo lavoro prende il nome di "manutenzione" del programma.

Per poter eseguire la manutenzione e la messa a punto di un programma e' necessario conoscerlo perfettamente.

E' necessario conoscere perfettamente le funzioni di ogni modulo, l'utilizzazione di ogni variabile, le interazioni di tutti gli elementi; in caso contrario una modifica operata in un punto fara' nascere errori in altri punti del programma.

L'esperienza mostra che conoscere bene un programma di una certa complessita' e' un compito molto difficile. Tale compito diventa ancora piu' difficile in fase di manutenzione se la persona incaricata non e' l'autore del programma.

L'esperienza ha del pari mostrato che e' molto difficile rileggere programmi scritti da altri, e che a volte e' difficile rileggere i propri programmi anche poco tempo dopo averli scritti.

Questo dipende dal fatto che in programmazione "si deve pensare a tutto!". Si devono cioe' aver presenti con precisione tutti gli elementi. Per esempio, per poter analizzare i riferimenti a una variabile, si deve conoscere il ruolo della variabile stessa, dove essa viene inizializzata, dove essa viene modificata, ecc......Questo costringe ad uno sforzo a volte penoso.

Lo scopo della programmazione strutturata e' quello di

rendere questo sforzo un po' meno penoso.

In effetti essa e' stata proposta in un momento nel quale la complessita' dei programmi era divenuta tale da rendere praticamente impossibile la manutenzione degli stessi.

Partendo dal fondamentale principio di produrre programmi piu' facili da comprendere (e quindi piu' facili da mettere a punto e da manutenere), gli scopi ed i mezzi della programmazione strutturata si possono porre a tre livelli:

. 1) Migliorare la leggibilita' del programma per mezzo della impaginazione e dell'autodocumentazione (presenza di commenti).

Si deve notare che alcuni credono che la programmazione strutturata si riduca a questo soltanto. In effetti, numerosi ed adeguati commenti sono molto importanti per rendere un programma comprensibile, ma questo non e' facilmente ottenibile dai programmatori. Nondimeno la programmazione strutturata non e' solo questo.

. 2) Scomporre il programma in moduli di dimensioni ragionevoli.

Qualunque cosa si faccia, a partire da una certa complessita' nessun programma puo' essere comprensibile. Questo e' semplicemente dovuto alla limitazione del potere di comprensione umana (quando i calcolatori scriveranno essi stessi i loro programmi, non ci saranno piu' tali problemi - o puo' darsi che ne nasceranno di piu'?).
Da questo e' nata l'idea di decomporre le funzioni complesse di un programma in una sequenza di funzioni semplici e ben definite. Ciascuna di queste funzioni verra' realizzata da un modulo la cui comprensione risultera' piu' facile.

. 3) Porre delle regole precise per la struttura del programma.

Per esaminare un programma e' necessario seguirne lo svolgimento passo a passo. Per esempio, le rotture di sequenza rendono penosa la lettura del programma con salti tra pagine diverse. La piu' pericolosa e' l'istruzione di salto incondizionato GOTO.
L'istruzione GOTO non e' indispensabile e questo puo' essere dimostrato. E' possibile scrivere dei programmi servendosi solamente di tre strutture fondamentali: la sequenza, la diramazione e l'iterazione.
La direttiva principale della programmazione strutturata e' quella di programmare usando solo queste tre strutture fondamentali.

Puo' inoltre essere evidenziato un quarto scopo della programmazione strutturata. Esso si e' gia' intravisto al punto 2) sopra esposto. Nelle sue forme piu' elaborate, la programmazione strutturata tende ad essere un metodo di analisi e di progettazione dei programmi; ed e' vero che la struttura di un programma nasce al momento del primo abbozzo.

Noi non insisteremo comunque su questo aspetto, dato che esso e' un punto di discordia tra gli specialisti; diversi metodi di analisi strutturata, derivati dalla programmazione strutturata, si affrontano in modo, e dobbiamo dirlo, abbastanza sterile. Per parte nostra ci atterremo all'aspetto "programmazione".

1.2. REGOLE DELLA PROGRAMMAZIONE STRUTTURATA

Un programma strutturato deve obbedire alle tre regole che seguono.

REGOLA 1

Il programma deve essere scomposto in piccoli moduli (sottoprogrammi) ben definiti ed individualizzati e la lunghezza di ogni modulo non deve, se possibile, superare una pagina.

In conseguenza il programma principale appare come una sequenza di chiamate a sottoprogrammi, aventi ciascuno un ruolo ben determinato:

```
chiamata modulo 1
chiamata modulo 2
.................
.................
```

Esso e' quindi facilmente leggibile e comprensibile.
Le interazioni tra i diversi moduli (scambi di dati) sono semplificate al massimo e, in ogni caso, perfettamente definite.
Ciascun modulo ha un solo punto di entrata ed un solo punto di uscita (cioe' il RETURN).

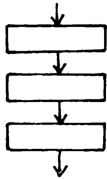
REGOLA 2

Ciascun programma e' costruito servendosi esclusivamente delle tre strutture fondamentali: sequenza, diramazione,

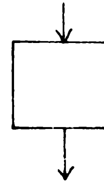
iterazione.

Segue il dettaglio delle tre strutture fondamentali.

LA SEQUENZA

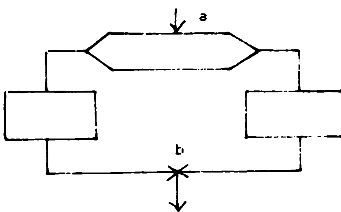


che puo' rappresentarsi anche cosi' ....>



Sequenza di operazioni che si susseguono senza alcuna condizione. Si ha un solo punto di entrata ed un solo punto di uscita. Per evitare eventuali ambiguita', la sequenza puo' essere compresa tra le parole: inizio......fine, adoperate come se fossero delle parentesi.

LA DIRAMAZIONE

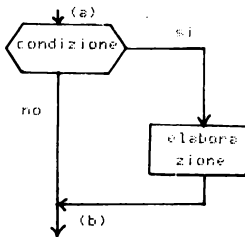


Essa ha la forma:

se la condizione e' vera fare questo, se non e' vera fare quest'altro.

Ciascuno dei due blocchi "questo" e "quest'altro" deve essere abbastanza corto. Se il blocco non puo' essere corto esso va sostituito con la chiamata ad un sottoprogramma. La cosa importante e' che i due rami della diramazione si ricongiungano nello stesso punto (B) vicino al punto di

partenza (A). I due rami sono allora sempre sotto gli occhi durante la lettura del programma.



In alcuni casi uno dei due blocchi "questo" o "quest'altro" puo' essere vuoto; questo corrisponde allo schema riportato a fianco. In parole: se la condizione e' vera fare "questo", altrimenti andare direttamente al punto di ricongiunzione (B) per contuinuare la sequenza. Anche in questo caso il punto (B) di ricongiunzione deve essere vicino all'analisi della condizione; quello che e' proibito e' di fare un salto ad un punto non visibile guardando il punto di partenza.

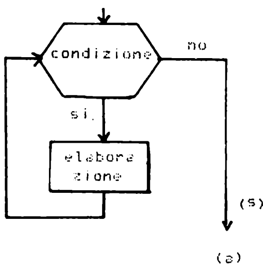
ESERCIZIO 1.1. - Tracciare, in due modi, il diagramma del calcolo: Y = valore assoluto di X.

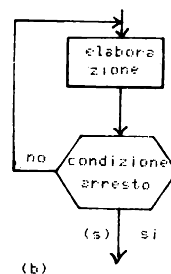
L' ITERAZIONE

Essa puo' avere due forme:

. a) fino a quando la condizione si mantiene vera fare "questo";
. b) ripetere "questo" fino alla condizione di arresto.

Le due forme risultano quasi, ma non del tutto, equivalenti se le condizioni analizzate sono tra loro contrarie. Esse corrispondono ai due schemi a) e b) che seguono.





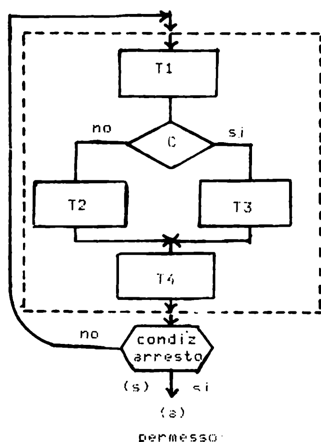
Per essere precisi, si vede che dopo la definizione in programmazione strutturata, nella forma a) l'analisi della condizione (test) deve essere all'inizio, mentre nella forma b) essa deve essere alla fine. Arrivando al punto di entrata della struttura con la condizione che determina l'arresto gia' realizzata, si vede che nella forma b) si ha una iterazione inopportuna.

ESERCIZIO 1.2. - Scrivere nel linguaggio usato precedentemente un programma che legga un file sequenziale e ne stampi il contenuto.

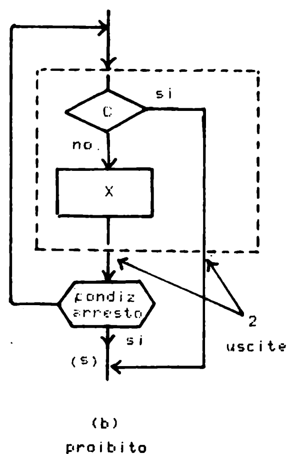
ESERCIZIO 1.3. - Tradurre in "linguaggio strutturato" il programma:

```
Basic: 10 FOR I = 1 TO 10          Fortran:     DO 30 I = 1,10
       20 PRINT I                               WRITE (6,10) I
       30 NEXT I                             30 CONTINUE
```

In programmazione strutturata e' consentito di uscire da un ciclo solo per mezzo dell'analisi della condizione (punto (S) nello schema sotto riportato). Tra le operazioni fondamentali del ciclo (quelle che devono essere ripetute) possono essere contenuti anche dei tests, ma essi devono essere nella forma " se, allora, se no", e non possono far uscire dal ciclo.



(a)
permesso



(b)
proibito

Il tratteggio presente nella figura a) circonda un modulo avente un solo punto di entrata ed un solo punto di uscita sempre sotto controllo.

Nella figura b) invece, quando si e' nel punto (S) non si sa se si proviene dal test di uscita o da un altra parte; questo puo' causare incomprensione del programma e possibili errori.

ESERCIZIO 1.4. - Il programma che segue, in Basic o in Fortran, cerca se esiste un "A" nella tabella di caratteri C o C$ e fornisce il rango K del primo "A" trovato.

```
10 DIM C$(50)                           INTEGER A C(50)
20 FOR K=1 TO 50                        DATA A/'A'
30 IF C$(K) = 'A' GOTO 50               DO 40 K=1,50
40 NEXT K                               IF (C(K) .EQ. A) GOTO 50
50 ..........                      40   CONTINUE
                                   50   ............
```

Tracciare il diagramma corrispondente e dire perche' non e' strutturato. Tracciare un diagramma strutturato che sia il piu' possibile simile al precedente. Dire perche' non e' pienamente soddisfacente. Tracciare infine un diagramma strutturato corretto.

Le strutture viste escludono l'istruzione GOTO. Infatti e' proprio questa l'istruzione che nuoce di piu' alla comprensione dei programmi. Se l'istruzione GOTO e' permessa, quando si esamina una parte di programma, non si sa mai precisamente da dove si proviene e quindi in quale stato si trovano le variabili. In conseguenza e' difficile fare una verifica del programma. E' proprio l'aver bandito l'istruzione GOTO che impedisce ai diagrammi di essere quei grovigli che erano arrivati ad essere.

REGOLA 3

Ogni programma deve essere opportunamente commentato ed impaginato in modo da facilitarne la leggibilita'.

In particolare ogni procedura deve iniziare con dei commenti che spieghino il suo ruolo e descrivano con precisione le variabili scambiate con il programma principale.

Analogamente devono essere spiegate tutte le articolazioni del programma. Per prima cosa le "astuzie", ma e' chiaro che nello spirito della programmazione strutturata le astuzie devono essere evitate.

Durante la stesura del programma, si cerchera' di metterne in evidenza la struttura usando dei margini convenienti; per esempio, tutte le istruzioni che fanno parte della stessa sequenza devono essere allineate.
Durante una scelta , l' "allora" e il "se no" corrispondenti devono essere allineati, analogamente il "fare" e il "fin tanto che", oppure, il "ripetere" e il "fino a quando".

ESEMPIO:

```
    REM ************************************************
    REM *       CALCOLO SOMMA 50 ELEMENTI TABELLA A       *
    REM ************************************************
        S <--- 0     ; REM  INIZIALIZZAZIONE
        I <--- 1     ; REM  PUNTA PRIMO ELEMENTO
        RIPETERE
            S <--- S + A(I)
            I <--- I + 1
        FINO A QUANDO    I > 0
    REM *   ORA SI HA LA SOMMA ****
```

ESERCIZIO 1.5. - Stendere un programma strutturato per mettere in ordine alfabetico una tabella di 50 nomi. Utilizzare il metodo dell' ordinamento " a bolle". Esso consiste in una successione di passi in ciascuno dei quali vengono esaminati successivamente gli elementi consecutivi a coppie. Se una coppia e' gia' in ordine si procede senza fare alcunche', se non lo e' si scambiano tra loro gli elementi. Si inizia un nuovo passo solo se nel precedente si e' operato almeno uno scambio in una coppia di elementi.

1.3. LIMITI DELLA PROGRAMMAZIONE STRUTTURATA

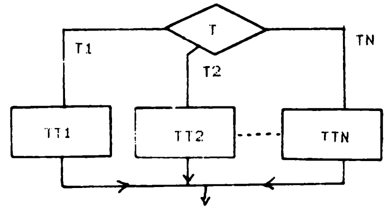
Si puo' fare della programmazione strutturata usando i linguaggi classici piu' comuni? Si.

Il lettore che conosce il Basic o il Fortran avra' subito visto che la diramazione puo' realizzarsi mediante un IF opportuno, e l'iterazione mediante un FOR o un DO. Per quanto riguarda il Basic, la questione viene esaurientemente trattata nell'Appendice A. Si ha una sola piccola difficolta' a livello di "si, allora, se no"; tale situazione deve essere simulata con i Basic che non dispongono della istruzione IF..THEN..ELSE.

In effetti, i piu' comuni linguaggi di programmazione ad alto livello hanno piu' possibilita' di quelle che sono necessarie per fare della programmazione strutturata. In

particolare, essi permettono di realizzare una struttura che non fa parte in senso stretto della programmazione strutturata, ma che alcuni autori vi aggiungono: la CASE.

Questa struttura ha la forma:




```
CASE T    TRA
     T1:  SEQUENZA 1 ;
     T2:  SEQUENZA 2 ;
     ........
     ........
     TN:  SEQUENZA N ;
```

Quando T = Ti viene eseguita la sequenza i. Questo non e' altro che una generalizzazione del GOTO calcolato del Fortran o del ON del Basic.

In effetti, e questa e' la vera difficolta', i linguaggi classici di programmazione consentono soprattutto di fare della programmazione non strutturata, e quindi illegibile. Essi sovente permettono di servirsi di artifici e trucchi che diventano incomprensibili 8 giorni dopo, ma che fanno risparmiare 2 istruzioni! Si puo' sempre sistemare un GOTO che fa risparmiare 1 microsecondo durante l'esecuzione (ma perdere 15 giorni durante la messa a punto!).

Confrontandola con i linguaggi classici, la programmazione strutturata arriva a privarsi volontariamente di alcune risorse dei linguaggi. Molti programmatori rifiutano questo, infatti essi si sentono frustrati se non utilizzano tutte le possibilita' dei linguaggi. Essi pensano di perdere in efficienza. Questo e' vero talvolta per quanto concerne i tempi di esecuzione, ma le differenze sono trascurabili, se si considera il risparmio di tempo durante la messa a punto e la manutenzione.

Altro motivo di frustrazione per i programmatori e' il problema dei commenti. E' vero che spesso non si ha il coraggio di scriverli (ne' d'altra parte di tenere documentazione della programmazione....).

E' proprio per le ragioni precedenti che sarebbe auspicabile poter disporre di un linguaggio di programmazione che:

.1) - Realizzi facilmente ed esattamente (senza "simulazione") le strutture fondamentali della programmazione strutturata.

.2) - Non possa realizzare altre strutture, in modo che i programmatori non si sentano frustrati dal fatto di poter utilizzare solo un sottoinsieme del linguaggio.

Questo e' precisamente lo scopo del linguaggio PASCAL, che e', per eccellenza, il linguaggio della programmazione strutturata.

Coloro che lo hanno concepito (il gruppo di Niklaus Wirth alla Scuola Politecnica Federale di Zurigo) lo hanno voluto cosi'. Il Pascal ha anche altri vantaggi, ma, in primo luogo, esso consente di fare della programmazione strutturata senza forzature, ne' restrizioni, ne' frustrazioni.

Esso quindi presenta un grande interesse per l'insegnamento, come primo linguaggio da studiare, perche' evita agli studenti di contrarre cattive abitudini nel programmare, cosa che capita con gli altri linguaggi, i quali favoriscono la ricerca dei trucchi.

La programmazione strutturata e' possibile anche con altri linguaggi, ma bisogna sapersi disciplinare, cosa che e' forse la piu' difficile.

PASCAL conduce "naturalmente" ad una programmazione chiara.

CAPITOLO 2

SGUARDO D'INSIEME SUL LINGUAGGIO PASCAL

2.1. PREMESSA

Prima di studiare in dettaglio le istruzioni del Pascal, faremo una panoramica del linguaggio che ci permettera' di metterne in luce la struttura.
E' inutile ripetere che tutti i linguaggi di programmazione ad alto livello hanno gli stessi tipi di istruzioni, ma vale la pena di ricordare che alcune categorie di istruzioni sono piu' sviluppate in qualche linguaggio rispetto ad altri. Noi ci sforzeremo di mettere in luce i "punti di forza" del Pascal.

2.2. COMANDI E DICHIARAZIONI

Si deve fare una prima distinzione tra istruzioni eseguibili o ORDINI e istruzioni non eseguibili o DICHIARAZIONI.

Questa distinzione e' ben nota ai lettori che hanno familiarita' con il Fortran (o il PL/1 o il Cobol), ma essa e' meno netta in Basic; per questa ragione dobbiamo chiarirne il significato.

A priori, tutte le istruzioni dovrebbero essere eseguibili, cioe' esse dovrebbero produrre una effettiva azione da parte del calcolatore, per esempio: leggere un numero dalla tastiera, eseguire un calcolo, ecc.....
In linguaggio macchina esistono solo istruzioni eseguibili. Nei linguaggi simbolici ad alto e basso livello si ha un comportamento diverso dovuto al processo di traduzione in linguaggio macchina del programma scritto in linguaggio evoluto. Si avranno, da una parte delle istruzioni tradotte in linguaggio macchina per effettuare delle operazioni effettive - esse sono le istruzioni eseguibili -, e d'altra parte delle istruzioni non tradotte in linguaggio macchina il cui scopo e' quello di aiutare il programma traduttore ad effettuare una buona traduzione.
Il miglior esempio di quest'ultimo tipo di istruzioni e' quello delle istruzioni per riservare spazio in memoria per una tabella (DIM in Basic, DIMENSION in Fortran); una tale istruzione non opera, ma influisce sul modo nel quale

vengono tradotte altre istruzioni. Una variabile viene trattata in modo diverso in dipendenza dal fatto di essere stata o meno preventivamente dimensionata.
Si puo' anche dire che le istruzioni eseguibili sono quelle che agiscono sui dati, mentre quelle dichiarative sono quelle che descrivono i dati che devono essere trattati.

A seconda dei linguaggi si ha un maggior o minore equilibrio tra i due tipi di istruzioni: in Cobol si hanno molte piu' istruzioni dichiarative che istruzioni eseguibili. Il Cobol e' un linguaggio per la gestione di grandi quantita' di dati sui quali si devono effettuare poche operazioni.
In Fortran invece si hanno piu' istruzioni eseguibili che istruzioni dichiarative e questo e' ancora piu' evidente in Basic. Anzi, in questo linguaggio, solo le istruzioni DATA e DIM possono essere considerate di tipo dichiarativo e in fondo non del tutto.

E' anche vero che, da questo punto di vista, la natura del traduttore (interprete o compilatore) gioca il suo ruolo, qualunque sia il linguaggio in questione. Con un interprete, ciascuna istruzione viene considerata isolatamente, tradotta ed eseguita. E' normale che in questo ambiente sia meno sentita la necessita' di molte dichiarazioni. Con un compilatore, invece, prima viene tradotto tutto il programma, poi esso sara' eseguito in blocco; le dichiarazioni aiutano a preparare la traduzione.

In Pascal si ha un buon gruppo di istruzioni non eseguibili. In effetti, come vedremo, il Pascal e' molto ricco per quanto riguarda i tipi di dati che esso puo' trattare; di conseguenza e' molto importante una accurata descrizione dei dati stessi.
Questo inoltre e' anche importante dato che il traduttore utilizzato e' di tipo compilativo.

Cominciamo ad analizzare le istruzioni eseguibili.

2.3. ISTRUZIONI ESEGUIBILI

Le istruzioni eseguibili si dividono in due categorie:
- le istruzioni senza condizioni o sequenziali;
- le istruzioni di strutturazione del programma.

Le istruzioni senza condizioni formano la sequenza di istruzioni che effettuano le operazioni richieste nel programma. Quando si arriva ad una istruzione senza condizioni essa viene eseguita, quando la sua esecuzione e'

terminata, si passa alla istruzione successiva.
Le istruzioni di strutturazione del programma, invece, non effettuano delle operazioni, ma determinano l'ordine secondo il quale devono essere eseguite le altre sequenze di istruzioni; questo genera la struttura del programma.

In Pascal , come negli altri linguaggi, le istruzioni senza condizioni sono di due tipi:
. le ISTRUZIONI DI CALCOLO (istruzione di assegnazione aritmetica della forma variabile := espressione aritmetica, dove ":=" si legge "prende il valore");
. le ISTRUZIONI DI INGRESSO-USCITA, o di manipolazione dei file: GET, PUT, RESET, REWRITE, READ, READLN, WRITE, WRITELN.

Come vedremo, queste ultime istruzioni sono delle procedure standard del Pascal, e non delle istruzioni propriamente dette, come le loro corrispondenti per esempio in Fortran.
A queste istruzioni sequenziali si possono aggiungere le istruzioni di DELIMITAZIONE DI SEQUENZA: BEGIN e END. Queste istruzioni servono a definire una istruzione composta come una sequenza di piu' istruzioni semplici. Questo evita ogni tipo di ambiguita' in una istruzione di strutturazione tipo WHILE.

Le istruzioni di strutturazione sono soprattutto quelle che consentono di suddividere un programma in moduli piu' piccoli. Questa e' una raccomandazione essenziale nella programmazione strutturata.

Secondo la tradizione, questo si ottiene servendosi di tre tipi di istruzioni:

.1) Una istruzione che mette in evidenza l'inizio del modulo.
In Pascal si ha l'istruzione PROCEDURE o FUNCTION. Ambedue servono per attribuire un nome al modulo. Si puo' aggiungere anche l'istruzione PROGRAM che in certe implementazioni del Pascal e' obbligatoria all'inizio del programma principale. (Queste tre istruzioni possono essere considerate delle dichiarazioni).

.2) Una istruzione per ritornare al programma chiamante.
In Pascal questo ruolo spetta all'istruzione END. (Si tratta del RETURN del Fortran o del Basic).

.3) Una istruzione di chiamata al sottoprogramma.
Per un modulo di tipo FUNCTION, la chiamata viene fatta in Pascal, come in Basic o in Fortran, riferendo il nome della funzione all'interno di una espressione aritmetica. Per un modulo di tipo PROCEDURE non esiste una CALL; la chiamata

viene fatta semplicemente citando il nome della procedura.

Subito dopo vengono le istruzioni che REALIZZANO LE STRUTTURE FONDAMENTALI DELLA PROGRAMMAZIONE STRUTTURATA:

IF...THEN...ELSE
WHILE...DO...
REPEAT...UNTIL...

ed a queste si aggiungono tre istruzioni supplementari, superflue perche' non strutturate, ma che possono essere utili. Esse sono:

FOR...DO... che realizza dei cicli controllati da un indice che si incrementa,
CASE...OF... che realizza la struttura CASE,
GOTO... salto incondizionato. Questa e' l'istruzione piu' paradossale in Pascal, dal momento che la programmazione strutturata e' nata proprio per sopprimere i GOTO! In effetti si puo' non usarla ed e' quello che noi faremo in tutti gli esercizi di questo libro.

2.4. DICHIARAZIONI

Lo scopo di queste istruzioni e' quello di descrivere i dati che vengono manipolati dalle istruzioni eseguibili. Tutti i dati sono riferiti nel programma per mezzo di un nome simbolico o identificatore. In Pascal, tutti gli identificatori utilizzati in una istruzione eseguibile devono prima essere stati dichiarati e descritti al compilatore Pascal, in una dichiarazione posta all'inizio del modulo, a meno che non si tratti di identificatori standard che si suppone siano gia' noti al compilatore, come TRUE, INTEGER o SIN, per esempio.

Gli elementi utilizzati piu' frequentemente sono le variabili. Ma in Pascal anche altri elementi possono ricevere un nome simbolico; la potenza del linguaggio sta proprio nella sua capacita' di simbolizzazione.

Esistono delle regole riguardo all'ordine di citazione degli elementi suscettibili di essere dichiarati; esse seguono.

. 1) DICHIARAZIONE DI ETICHETTE

Una etichetta e' un numero scritto all'inizio di una linea e seguito dai due punti (:) che lo separano dalla parte restante della linea stessa. Esempio:

```
10: A:=B+C
```

Le etichette devono essere dichiarate per mezzo dell'istruzione LABEL. Esempio:

```
LABEL 10, 20, 30;
```

A cosa servono le etichette? A localizzare l'istruzione per un GOTO come GOTO 10. In conseguenza, noi non parleremo piu' di etichette, dal momento che non utilizzeremo l'istruzione GOTO.

. 2) DICHIARAZIONE DI COSTANTI

Le costanti possono essere usate come in Fortran o in Basic. Esempio di uso della costante 10 in una istruzione:

```
A:=X+10
```

Si dice che la costante e' stata usata in forma letterale (literal).

In Pascal si puo' utilizzare una forma simbolica delle costanti, questo permette di dare loro un nome piu' significativo e di avere un programma piu' leggibile. Allora si deve usare una dichiarazione come:

```
CONST PI = 3.14; LINEEPAG = 60;
```

e scrivere: IF NUMLINEE < LINEEPAG

sara' piu' chiaro di: IF NUMLINEE < 60........

NOTA: E' possibile fare qualcosa di simile in Fortran o in Basic. Basta usare una variabile. Questa viene chiamata "parametro". Questo e' utile se un parametro puo' cambiare, come nell'esempio riportato il "numero di linee per pagina"; infatti basta cambiare una sola istruzione, quella dove il parametro riceve il suo valore.

. 3) DICHIARAZIONE DI TIPO

In Pascal si hanno un certo numero di tipi di dati standard; essi sono:

BOOLEAN, INTEGER, REAL e CHAR

(da notare che CHAR si riferisce alla stringa di 1 carattere; questo e' molto piu' restrittivo di quanto avviene in Basic per le stringhe!).

Pero', e questo e' un grosso vantaggio del Pascal, il programmatore puo' definire i suoi tipi di variabili, cosa che gli permette di manipolare classi di oggetti particolari e adatti al suo problema.

Un tipo puo' essere definito sia listando i simboli degli oggetti costituenti il tipo:

```
TYPE PAESAGGIO = (MARE, MONTAGNA, CAMPAGNA, CITTA')
```

sia definendo un intervallo di validita':

```
TYPE MINUGCENTO = 1..100
```

Infine, un tipo di oggetti puo' essere definito come insieme (SET OF) di elementi appartenenti a un tipo elementare definito.
Questo e' molto comodo, soprattutto perche' il compilatore inserisce automaticamente delle verifiche sulla conformita' dei tipi.

Torneremo su questo argomento in seguito e scopriremo alcune limitazioni.

. 4) DICHIARAZIONE DI VARIABILI

La dichiarazione di ogni variabile specifica il tipo della variabile e, eventualmente, la sua struttura, se si tratta di una tabella. Esempio:

```
VAR I, J: INTEGER;
    M: ARRAY[1..5,1..7] OF REAL;
```

definisce I e J come variabili intere e M come matrice reale di dimensioni 5 per 7.

La struttura ARRAY puo' essere imposta ad un tipo. Per esempio, certi compilatori hanno come tipo standard anche ALFA, che potrebbe anche essere dichiarato cosi':

```
TYPE ALFA=ARRAY[1..10] OF CHAR;
```

La struttura RECORD puo' anche essere imposta sia ad una variabile che ad un tipo.

Le variabili strutturate di tipo ARRAY sono composte da elementi tutti dello stesso tipo. Questo e' quello che succede anche in Fortran o in Basic.

In Pascal esistono pero' anche delle strutture complesse di variabili, nelle quali gli elementi sono di tipo diverso. Per esempio, un documento puo' contenere un nome (stringa di

caratteri), poi un numero, poi un numero di ore di lavoro. Questo puo' essere rappresentato da una variabile RECORD o da un tipo RECORD. Esempio:

```
VAR IMPIEGATO : RECORD
                           NOME:ALFA;
                           NUMERO:INTEGER;
                           NUMERORE:REAL END;
```

Un tipo strutturato si dichiara per esempio come:

```
TYPE COMPLEXE = RECORD PARTREAL:REAL;
                       PARIMMAG:REAL END;
```

e si potra' avere: VAR Z : COMPLEXE;

NOTA: In un modulo di programma Pascal le parole VAR e TYPE devono comparire una sola volta. Se si devono dichiarare molti simboli, si procedera' cosi':

```
TYPE MINUGCENT= 1..100;
     VACANZA= (MARE, MONTAGNA);
```

Questa panoramica delle istruzioni del Pascal ci ha mostrato quale ricchezza ci sia di tipi di dati e come le possibilita' al riguardo possano essere migliorate ulteriormente.

Questo porta naturalmente ad una certa complessita'. D'altra parte si hanno certe carenze, come il tipo CHAR che si riferisce a stringhe di 1 carattere, e si e' visto che il tipo ALFA, standard o definibile dall'utente, consente di aggirare questa difficolta'.

Si puo' comunque affermare che in generale e' difficile stabilire se un linguaggio e' superiore o inferiore ad un altro. Ogni volta che un linguaggio presenta una limitazione, fornisce anche i mezzi per superarla; inversamente tutte le volte che mostra di avere pregi straordinari si scopre che altri linguaggi lo superano in altri punti. Inoltre non si puo' considerare il miglior criterio di scelta il fatto che un linguaggio si scriva facilmente. Ci sembra miglior criterio guardare se le caratteristiche del linguaggio facilitano l'acquisizione di buone abitudini nel programmare. Alcune esperienze hanno dimostrato che, da questo punto di vista, il Pascal si trova in buona posizione.

2.5. ALTRI VANTAGGI DEL PASCAL

La possibilita' di definire tipi di dati convenienti per l'utente e la predisposizione alla programmazione strutturata sono due vantaggi indiscutibili del Pascal. Ora ne vedremo degli altri, che sono meno importanti, ma che hanno anche il loro peso.

STRUTTURA A BLOCCHI

Supponiamo di scrivere un programma in Basic. All'interno di un ciclo:

```
100 FOR I=1 TO N
150 NEXT
```

decidiamo di porre: 130 GOSUB 1000

Nel sottoprogramma che inizia in 1000 si ha un altro ciclo, governato ancora da I:
1050 FOR I = ... ed ecco l'errore!

Infatti modificando la I della istruzione 1050 viene modificata la stessa I che e' usata nella 100.

Si ha una mancanza di modularita': la variabile I usata nel sottoprogramma dovrebbe essere indipendente dalla variabile I del programma chiamante. Se cosi' fosse la variabile I sarebbe chiamata "variabile locale".

In Basic tutte le variabili sono "globali"; qualunque variabile e' valida per tutti i sottoprogrammi presenti.
In Fortran , invece, le variabili di un sottoprogramma sono locali. Pero' nasce il problema di far comunicare tra loro il programma principale e i sottoprogrammi e quindi servono delle variabili globali. Il Fortran risolve il problema in modo molto delicato o trasmettendo gli argomenti nella chiamata dei sottoprogrammi o usando la COMMON.

Il Pascal da' una buona soluzione a questo problema (come d'altronde Algol, PL/1 o LSE): un programma puo' essere suddiviso in blocchi delimitati sia da FUNCTION...END che da PROCEDURE...END. I blocchi possono essere contenuti uno dentro l'altro (inscatolati).

Se una variabile e' dichiarata all'inizio di un blocco, essa risulta locale per quel blocco. Una variabile conosciuta in un blocco e' conosciuta da tutti i sottoprogrammi interni al blocco stesso, a meno che essa non venga ridichiarata in un blocco interno. Esiste dunque la

possibilita' di avere variabili locali-globali a parecchi stadi diversi e questo permette di controllare come si vuole il programma.

GRADO DI SIMBOLIZZAZIONE

Qual'e' il principale progresso ottenuto nel passaggio dal linguaggio macchina all'assembler e poi dall'assembler ad un linguaggio ad alto livello? I dati sono trattati in forme via via piu' simboliche.

Quanto piu' il trattamento dei dati riferisce oggetti simbolici, tanto piu' le cose sono viste dall'alto e tanto piu' si ha una visione sintetica dell'elaborazione.

Gli oggetti che possono essere trattati in forma simbolica nei diversi linguaggi di programmazione sono:

- le variabili;
- le costanti;
- le etichette;
- i nomi dei sottoprogrammi.

La tabella che segue mostra quali sono trattabili in forma simbolica nei piu' comuni linguaggi di programmazione.

OGGETTI	L I N G U A G G I				
	Macchina	Basic	Fortran	Pascal	Cobol
Variabili	NO	SI	SI	SI	SI
Sotto-programmi	NO	NO	SI	SI	SI
Etichette	NO	NO	NO	NO(1)	SI
Costanti	NO	NO	NO	SI	NO(2)

(1) le etichette sono poco utilizzate in Pascal e non hanno ragione di esistere nella programmazione strutturata;
(2) esistono alcune costanti speciali in forma simbolica: SPACE, HIGH-VALUE,ecc.

La precedente tabella mostra che il Pascal e' ben piazzato da questo punto di vista e questo giova molto alla sua duttilita' in fase applicativa.

2.6. I CARATTERI DEL PASCAL

E' tradizione, quando si studia un linguaggio, fermare l'attenzione sul SET di caratteri disponibili.
Il SET dei caratteri standard Pascal comprende:

. LETTERE: A.....Z; a.....z;
. CIFRE: 0.....9;
(in questo libro lo zero e la lettera O sono chiaramente distinguibili e quindi non e' necessario barrare l'uno dei due);
. CARATTERI SPECIALI, OPERATORI O SEGNI DI PUNTEGGIATURA:
+ - * / V ∧ ¬ = ≠ < <= >= > ( ) [ ] { }
'(apostrofo) . , : ; e lo spazio.

Noi utilizzeremo inizialmente il set standard. Nascono delle difficolta' quando si deve lavorare su uno specifico calcolatore. A volte e' necessario usare un insieme ridotto di caratteri e per le stampanti un insieme ancora piu' ridotto.

Per questa ragione ad alcuni particolari caratteri e' stato attribuito un sinonimo e questo puo' essere usato se necessario. In particolare si ha un set ASCII nel quale si ha:

. le lettere minuscole non sono usate;
. sono usati i sinonimi che seguono:
V ↔ OR ∧ ↔ AND ¬ ↔ NOT ≠ ↔ <> ⩽ ↔ <=
⩾ ↔ >=
↑ ↔ @ { ↔ (* } ↔ *)
ed a volte anche i seguenti:
≠ ↔ # ∧ ↔ & V ↔ ! [↔ (.] ↔ .)
¬ ↔ ∿ {et}↔ %

Tutto questo comunque nuoce alla portabilita' dei programmi. Noi consacriamo un paragrafo alla portabilita'.

2.7. PUNTEGGIATURA E IMPAGINAZIONE

Come tutti i linguaggi, il Pascal ha delle regole di punteggiatura, piu' precise che in Fortran (dove mancano quasi del tutto) o in Basic (solo : per separare due istruzioni sulla stessa linea).

Nel Pascal il separatore tra due istruzioni e' il punto e virgola (;). La regola d'uso e' semplice:

. Tutte le istruzioni devono essere seguite dal ; a meno

che esse siano seguite da END, ELSE o UNTIL, o parole chiave simili.

. END e' seguita da ; a meno che:

. essa sia seguita da una parola chiave in una struttura concatenata;

. sia la END finale di una procedura; in questo caso essa e' seguita da un punto (.).

In caso di dubbio non ci si deve preoccupare di mettere un ; in piu' (questo crea solo delle istruzioni vuote). Il punto e virgola separa anche le diverse classi in una istruzione dichiarativa. Ne abbiamo gia' visto degli esempi.

L'uso di spazi ha alcune limitazioni (molto leggere e del tutto normali): non si possono usare spazi all'interno di parole chiave, le parole chiave devono essere seguite almeno da uno spazio e questo favorisce la leggibilita'.

Il Pascal incoraggia l'uso degli spazi per ottenere una impaginazione che faciliti la lettura del programma. Ne abbiamo visto degli esempi nelle istruzioni dichiarative. Naturalmente per le istruzioni di strutturazione il Pascal fa sue le raccomandazioni della programmazione strutturata (vedi Capitolo 1).
I programmi ottenuti sono allora perfettamente leggibili. Ma bisogna notare che questo non e' il solo pregio del Pascal!
In effetti, le regole di impaginazione sono solamente delle raccomandazioni; esse non sono per nulla obbligatorie. Niente vi impedisce di fare dei programmi "da cani" in Pascal!

Voi potete mettere piu' istruzioni sulla stessa linea ed anche scrivere una istruzione per meta' su una linea e per l'altra meta' sulla linea seguente. Esempio:

```
XXXXXXX ; YYYYY        invece di    XXXXXXX ;
YYYYY ;                             YYYYYYYYYY ;

IF X<0 THEN Y:=-       invece di    IF X<0
X ELSE Y:=X;                           THEN Y:=-X
                                       ELSE Y:=X;
```

Inversamente, niente vi impedisce di fare una buona impaginazione in Fortran o in Basic; per quest'ultimo lo mostriamo nell'Appendice A.

Allora perche' questa differenza? Perche' non fare una buona impaginazione in Fortran o in Basic, posto che la si

fa in Pascal?

E' una questione di abitudine, e, bisogna riconoscerlo, nella formazione delle abitudini entra in buona parte la responsabilita' degli insegnanti.

Se il Fortran ed il Basic fossero insegnati incoraggiando la chiarezza della esposizione, le buone abitudini verrebbero prese sin dall'inizio.

E' anche vero che la struttura del Pascal incoraggia, inconsciamente, la buona disposizione. D'altra parte, l'apparizione del Pascal dopo la programmazione strutturata ha certo favorito un cambio nelle abitudini. Noi consideriamo, in ogni caso, il fatto di insegnare il Pascal, come un impegno dell'insegnante a incoraggiare delle buone abitudini nella programmazione.

Ovunque in Pascal sono permessi degli spazi puo' essere inserito un commento includendolo tra parentesi graffe. Ma in Fortran ed in Basic si hanno altrettante possibilita' per inserire commenti, noi potremmo riprendere la stessa discussione di cui sopra....

Ci limitiamo a raccomandare vivamente di inserire numerosi ed appropriati commenti qualunque sia il linguaggio utilizzato.

ESERCIZIO 2.1. Questa panoramica sul Pascal dovrebbe essere sufficiente per scrivere in Pascal il programma dell'esercizio 1.5. (la soluzione e' nell'Appendice D).

Nota: I commenti nei programmi in Pascal devono essere contenuti in parentesi graffe. Dal momento che non possiamo usare tali parentesi nel testo, i commenti vengono delimitati da asterischi.

CAPITOLO 3

ISTRUZIONI SEQUENZIALI

3.1. PREMESSA

Nel corso di questo capitolo, esamineremo gli elementi fondamentali del Pasçal, e servendoci di essi, potremo gia' eseguire dei piccoli programmi.

In questo capitolo si esaminano solamente le istruzioni che vengono eseguite in modo sequenziale; sara' solamente nel prossimo capitolo che vedremo le istruzioni che permettono di passare da una sequenza ad un'altra: istruzioni di rottura di sequenza o istruzioni di strutturazione.

Le tre istruzioni sequenziali fondamentali corrispondono alle tre operazioni di base che sono sempre presenti durante il trattamento delle informazioni:

1. Acquisizione dei dati che devono essere elaborati.
2. Calcoli che permettono di elaborare nuove informazioni (risultati) a partire dai dati iniziali.
3. Emissione o stampa dei risultati.

3.2. ISTRUZIONI DI LETTURA

PRESENTAZIONE DEI DATI

I dati che devono essere letti e servono come punto di partenza per un calcolo, sono forniti in una forma che dipende dalle modalita' di utilizzo del calcolatore.

I due principali modi di utilizzo sono:

a. Grandi calcolatori con "elaborazione a lotti".
b. Micro-calcolatori singoli con "elaborazione conversazionale".

Un caso intermedio e' quello dei grossi calcolatori utilizzati in "tempo parziale". Questo caso e' riconducibile al caso b. dal momento che ogni utilizzatore dispone di un terminale conversazionale.

Nel caso a. l'utilizzatore, in generale, prepara un pacco

di schede, e, dopo un attesa che mette a dura prova la sua pazienza, riceve un listato con i suoi risultati.

Il pacco di schede comprende:

. le schede del programma Pascal;
. le schede dei dati di ingresso;
. le schede, chiamate di controllo, che servono per il sistema operativo.

Un pacco di schede potrebbe anche avere la seguente composizione:

. scheda di controllo che definisce il tipo di lavoro (dice per esempio a chi deve essere addebitato il tempo macchina);
. scheda di controllo per chiamare il compilatore Pascal;
. schede del programma Pascal;
. scheda di controllo di "fine pacco";
. scheda di controllo per chiedere l'esecuzione del programma;
. schede dei dati;
. scheda di controllo di "fine pacco".

I dati sono scritti sulle schede sotto forma di numeri. Due numeri sono separati uno dall'altro o almeno da uno spazio o dal fatto che si passa sulla scheda seguente.

Esempio: per fornire i numeri 1 1,5 2 si avra' una scheda come la seguente:

```
 1      1.5     2
```

notate il punto anglosassone al posto della virgola per i decimali.

Nel caso b. si utilizza una tastiera e un video invece della scheda scritta, e le risposte sono immediate. Ma le informazioni da scrivere alla tastiera sono dello stesso tipo delle precedenti e cioe' delle tre categorie:

. comandi per il sistema operativo;
. istruzioni Pascal;
. dati forniti in risposta ad istruzioni di lettura.

ISTRUZIONI DI LETTURA

Il Pascal fornisce due istruzioni per la lettura dei dati: READ e READLN, seguite dalla lista, contenuta tra parentesi, delle variabili che devono ricevere i dati.

Esempio: READ (A,B,C);

legge la scheda mostrata sopra e pone: A = 1, B = 1.5 e C = 2.

Se piu' istruzioni READ si susseguono, esse leggono dati dalla stessa scheda, fino a quando ce ne sono.

Esempio:
```
READ (A,B);
READ (C);
```

leggono dalla stessa scheda i tre dati, come nel precedente esempio.

READLN (X,Y); fa leggere le due variabili X e Y, poi passa alla scheda seguente, in attesa di letture successive.

READLN; puo' essere usata senza lista di variabili. Essa ha l'effetto di far passare alla scheda seguente (ignorando eventuali dati che si trovino ancora sulla scheda).

Esempio:
```
READLN (A,B);
READ C;
```

non avra' lo stesso effetto dei due primi esempi precedenti; infatti si avra' (sempre con la scheda di dati precedentemente usata) A = 1, B = 1.5, ma il dato 2 verra' ignorato per effetto di READLN con solo due variabili nella lista, e la variabile C ricevera' il primo dato presente sulla scheda successiva.

Se, durante una lettura, si arriva alla fine della scheda senza aver ottenuto valori per tutte le variabili della lista, la lettura continua sulla scheda seguente; il cambio della scheda ha lo stesso effetto separatore dello spazio tra i dati.

Esempio: due schede come queste
```
1   2   3
4   5
```
e l'istruzione READ (A,B,C,D);
fanno si che: A = 1, B = 2, C = 3 e D = 4.

ESERCIZIO 3.1. - Avendo le seguenti 3 schede di dati:
```
1   2   3
4   5
6
```
ed i comandi di lettuta:
```
READLN (A,B);
READ (C,D,E);
```
dire quali saranno i valori delle variabili.

I separatori non hanno effetto se le variabili sono di tipo CHAR. Si vedra' piu' avanti che una variabile CHAR

contiene un carattere alfanumerico.

Quando si chiede di leggere una lista di variabili CHAR, viene letto in sequenza qualunque carattere e' presente sulla scheda.

Esempio: se si ha una scheda con la scritta ALFREDO, l'istruzione:
READ (A,B,C,D); pone A = 'A', B = 'L', C = 'F', D = 'R'

Quando la scheda non contiene piu' dati, non si ha il passaggio automatico alla scheda seguente, ma le variabili restanti vengono riempite di spazi (blanc). Per passare alla scheda successiva e' necessaria una istruzione READLN.
Vedremo che e' possibile usare la funzione EOLN (fine di riga) per verificare se tutti i carattetri di una scheda sono stati letti.

ESERCIZIO 3.2. - Leggere i primi 4 caratteri di una scheda nelle variabili (di tipo CHAR) A, B, C, D e i primi 4 caratteri della scheda successiva nelle variabili E, F, G e H.

UTILIZZO IN MODO CONVERSAZIONALE

Resta valido tutto quello che si e' detto pur di sostituire la parola "scheda" con la parola "riga". Il passaggio alla riga seguente si ottiene usando il tasto "ritorno-carrello".

Il modo piu' sicuro e' sempre quello di leggere le variabili una per volta con delle istruzioni READLN (esempio: READLN (X);) e di terminare ogni dato con il tasto "ritorno-carrello".

3.3. ISTRUZIONI DI SCRITTURA

Il Pascal dispone di due istruzioni per la stampa dei risultati: WRITE e WRITELN.

Nell'utilizzo a schede i risultati vengono in generale stampati su carta; nel modo conversazionale essi compaiono sullo schermo.

Per evidenziare il valore delle variabili A, B e C, si scrivera' per esempio:

```
WRITE (A, B, C);   oppure
```

```
WRITELN (A, B, C);
```

I valori di A, B, C vengono evidenziati consecutivamente. Per esempio, se A, B e C valgono rispettivamente 1, 2 e 3, si vede apparire: 123.

La differenza tra le due istruzioni WRITE e WRITELN e' che, dopo la istruzione WRITE non si ha il passaggio alla linea seguente, mentre questo avviene dopo la istruzione WRITELN (quest'ultima istruzione esaurita la lista delle variabili manda a nuova linea).

Esempio: se A, B e C valgono 1, 2 e 3; con la sequenza:

```
WRITE (A, B, C);
WRITELN (A, B, C);
```

si ottiene: 123123;

mentre con la sequenza:

```
WRITELN (A, B, C);
WRITE (A, B, C);
```

si ottiene:

```
123
123
```

ESERCIZIO 3.3. - Considerate i due esempi sopra riportati e dite dove si avra' la stampa successiva.

WRITELN puo' essere usata senza lista di variabili. L'effetto e di mandare a nuova linea.

Gli elementi da stampare possono essere anche delle espressioni aritmetiche o delle stringhe di caratteri compresi tra apici.

Esempio:

```
WRITE ('LA SOMMA DI A E B RISULTA ',A+B);
```

produce la stampa di:

```
LA SOMMA DI A E B RISULTA 3
```

se i valori di A e B sono quelli usati sopra.

Le stringhe di caratteri sono indispensabili per spiegare il significato dei risultati che vengono stampati (se i risultati possibili sono molti vedere delle sequenze di cifre senza spiegazioni non e' accettabile). Inoltre esse

servono per distanziare i risultati inserendo degli spazi (negli esempi esposti non si poteva sapere se si trattava delle tre cifre 1, 2 e 3 o del numero 123!).

IMPAGINAZIONE

Oltre all'elemento da scrivere puo' essere specificato anche il "formato" nel quale esso deve essere scritto.

Per un risultato intero, si puo' specificare mediante un numero intero il numero minimo di caratteri da scrivere.

Per esempio con: WRITE (K:3);

si ottiene rispettivamente:

```
K=5        viene scritto    5
K=121        "      "     121
K=1538       "      "     1538
```

In conclusione, se l'elemento ha bisogno di almeno 3 caratteri, esso verra' scritto almeno con 3 caratteri inserendo eventualmente degli spazi a sinistra; se l'elemento ha bisogno di piu' caratteri, essi verranno scritti tutti (per esempio il Fortran in questa situazione si comporta diversamente).

Per un risultato reale, si devono specificare due numeri per i caratteri; il primo si riferisce al numero totale dei caratteri, come per gli interi o le stringhe, il secondo si riferisce al numero delle cifre decimali.

Cosi' il numero X=25.678, usando la istruzione

WRITE (X:10:4) viene scritto 25.6780 facendo precedere il 25 da 3 spazi.
Il segno, il punto decimale ed uno spazio iniziale, sempre presente, contano nel numero totale dei caratteri.

Quando per un numero reale non viene specificato il formato di scrittura, esso viene stampato cosi':
spazio segno Y.XXXEn e viene interpretato come:
Y.XXX x 10 elevato a n.

I parametri di formato possono anche essere scritti come espressioni aritmetiche a valore intero.

Quando si vuole stampare una intestazione facendola precedere da N spazi, con N calcolato, basta scrivere:

```
WRITELN ('':N,'intestazione');
```

e questo potra' servire per tracciare una curva.

ESERCIZIO 3.4. - Calcolare la somma dei valori delle variabili A e B e dare il risultato facendolo precedere da una frase esplicativa.
La stampa dovra' provocare il passaggio alla linea successiva.
Osservate la differenza con l'esempio riportato dopo l'esercizio 3.3..

CAMBIO DI PAGINA

Usando una stampante con carta ripiegata a fogli si puo' usare l'istruzione PAGE per portarsi all'inizio della pagina seguente.

Abbiamo letto dei dati e stampato dei risultati. Sappiamo anche impaginare i risultati, ma dobbiamo imparare a calcolarli!

3.4. ISTRUZIONE ARITMETICA DI ASSEGNAZIONE

L'istruzione fondamentale per il calcolo e' l'istruzione aritmetica di assegnazione. Essa ha la forma:

variabile := espressione aritmetica

Esempi:

```
C :=A+B ;
S :=PI*R*R ;
```

Essa ha il seguente effetto:

1. calcola l'espressione aritmetica che sta scritta a destra di :=
2. assegna il risultato nella forma adatta alla variabile scritta a sinistra di :=.

In Pascal una scrittura come: X :=X+1; non ha alcunche' di paradossale.

ESERCIZIO 3.5. - Qual'e' l'effetto della istruzione:
X:=X+1; ?

ESERCIZIO 3.6. - E' corretta l'istruzione A+B := C; ?

ESPRESSIONI ARITMETICHE

Una espressione aritmetica e' una combinazione, tanto complicata quanto si vuole, di operandi e di operatori aritmetici.

GLI OPERATORI

Gli operatori aritmetici del Pascal sono:

+	addizione
-	sottrazione
*	moltiplicazione
/	divisione a risultato reale
DIV	divisione a risultato intero
MOD	resto della divisione (X MOD Y fornisce il resto della divisione di X per Y).

Questi operatori si applicano ad operandi interi o reali (supponiamo, per il momento, una conoscenza intuitiva di questi tipi di operandi; essi verranno discussi in dettaglio piu' avanti) che possono essere mescolati.

DIV e MOD si applicano solo ad operandi interi ed i risultati prodotti sono interi.

Per gli operatori + - * il risultato e' dello stesso tipo dei due operandi se essi sono dello stesso tipo; il risultato e' reale se almeno uno dei due operandi e' reale. L'operatore / da' sempre un risultato reale, anche se i due operandi sono interi.

Esempio: 3/2 da' come risultato 1.5
3 DIV 2 da' come risultato 1.

Confrontando con il Basic e il Fortran si hanno gli operatori DIV e MOD in piu', ma non si ha l'operatore per l'elevamento a potenza! Pur essendo esplicitamente voluto da coloro che hanno predisposto il Pascal ed essendo facilmente ottenibile mediante una funzione definita dall'utente, questa e' una mancanza!

ESERCIZIO 3.7. - Scrivere una espressione equivalente a X MOD Y utilizzando DIV.

Esistono delle regole per la valutazione delle

espressioni. Infatti una espressione come A+B*C potrebbe sembrare ambigua; si calcola A +(B*C) oppure (A+B)*C ?

Queste regole sono analoghe a quelle del Fortran e del Basic:

1. Gli operatori moltiplicativi (* / DIV MOD) hanno la precedenza su quelli additivi (+ -). Per cui nel caso precedente e' A+(B*C).

2. Se due operatori appartengono allo stesso gruppo essi si applicano partendo da sinistra. Esempio: 4*5 DIV 10 da' come risultato 2 (la moltiplicazione viene eseguita prima).

3. Si puo' sempre imporre un ordine facendo uso delle parentesi. Per moltiplicare la somma di A e B per C si scrive: (A+B)*C.

Le regole di formazione delle espressioni aritmetiche sono recursive e possono portare gradualmente ad espressioni molto complicate.

In una istruzione di assegnazione aritmetica, in Pascal come in tutti gli altri linguaggi, gli operandi che figurano nell'espressione a destra di := sono semplicemente utilizzati per il calcolo. Le variabili non vengono modificate. L'unica variabile modificata e' quella che si trova a sinistra di :=.

3.5. GLI OPERANDI

Gli operandi delle espressioni aritmetiche sono le costanti, le variabili ed i riferimenti alle funzioni.

LE COSTANTI

Una costante e' utilizzata quando si deve fare riferimento ad un dato gia' esplicitamente conosciuto e non variabile.

Per esempio, se si deve moltiplicare X per 3 e si e' sicuri che 3 e' una costante che non cambiera' mai, si scrive: Y:=X*3;.

Il Pascal consente di esprimere le costanti in forma simbolica purche' esse siano state dichiarate tali con una istruzione CONST.

Esempio: CONST FATTORE = 3;

```
                    PI = 3.14159265;
e poi:      Y :=X*FATTORE
            S :=PI*R*R
```

Questa possibilita' ha 3 vantaggi:

1. Le costanti vengono espresse in forma mnemonica mediante un nome.
2. Se, a causa di una modifica, e' necessario cambiare la costante, si deve modificare il programma solo in un punto.
3. La costante viene definita una sola volta e puo' venire usata quante volte si vuole.

Domanda: In cosa una costante differisce da una variabile?

In effetti, in un linguaggio diverso dal Pascal, si sarebbe fatto ricorso ad una variabile per ottenere lo stesso risultato.
Le "costanti simboliche" del Pascal hanno un altro vantaggio. Supponete di avere in un programma una variabile con nome simbolico PU e di avere definito la costante simbolica PI=3.14159265. Se piu' avanti nel programma commettete l'errore di scrivere: PI := PU + AU (invece di scrivere: PU := PU + AU), negli altri linguaggi non vi viene segnalato alcun errore e tutti i calcoli nei quali si usa PI saranno sbagliati. Con il Pascal, invece, un errore di questo tipo viene segnalato, infatti il compilatore non consente di modificare il valore di una costante e PI e' stato dichiarato CONST.

LE VARIABILI

Le variabili sono gli operandi piu' frequentemente usati nelle espressioni aritmetiche.

Ricordiamo che una variabile e' un nome simbolico attribuito ad una parte di memoria (contenitore).

In Pascal ogni variabile deve essere dichiarata in una istruzione VAR. Lo scopo principale di questa istruzione e' di specificare il TIPO della variabile.

```
Esempio: VAR I : INTEGER;
             Z : REAL;
```

La specifica del tipo e' essenziale per consentire al compilatore di allocare per la variabile la quantita' di memoria necessaria.
La quantita' di memoria allocata per una variabile determina l'ordine di grandezza massimo del numero in essa contenuto e la sua precisione.

Questi parametri disgraziatamente variano in dipendenza dal compilatore e dal calcolatore usati.

Le PAROLE usate nei linguaggi di programmazione appartengono a 2 categorie:

Le PAROLE CHIAVE, che sono fisse ed hanno un preciso significato per il linguaggio (esempio: VAR, CONST, BEGIN, IF,....). Esse sono parole riservate nel senso che non devono essere utilizzate (per esempio come identificatori) in circostanze diverse da quelle per le quali il loro impiego e' previsto con il significato ad esse attribuito.

Gli IDENTIFICATORI, che sono arbitrari, sono inventati dal programmatore secondo i suoi desideri, pur di rispettare alcune regole nella formazione della parola. I migliori esempi di identificatori sono i nomi simbolici delle costanti e delle variabili.

Le regole per la formazione degli identificatori sono poco restrittive in Pascal. Un identificatore deve essere formato da lettere e cifre, ma il primo carattere deve essere una lettera.
Il numero dei caratteri e' libero, ma la maggior parte dei compilatori distingue tra loro gli identificatori in base ai primi 6, 8 o 10 caratteri. Due nomi come : PRINCIPE e PRINCIPESCO, se il compilatore considera 8 caratteri, riferiscono la stessa variabile.
Tenendo conto che in molti Basic la distinzione viene fatta su 2 caratteri si vede che in Pascal la situazione e' migliore. Il Pascal consente quindi di creare degli identificatori dotati di un buon significato mnemonico e questo ha la sua importanza.

ESERCIZIO 3.8. - Dite quali critiche si possono fare alla seguente dichiarazione di variabili:

```
        VAR   1RIMAVAR, SECONDAVAR : INTEGER ;
              ALBERTA, ALFREDO, ARTURO : REAL ;
              SOTTO, SOPRA, ALBERTA : INTEGER ;
              ENRICO-QUARTO : REAL ;
              H2SOA4, SOTTO : REAL ;
```

Naturalmente non si possono inserire all'interno degli identificatori ne' caratteri speciali ne' spazi (meno libero che in Fortran, ma preferibile).

Esiste una categoria di parole intermedia tra le parole-chiave e gli identificatori; esse sono gli IDENTIFICATORI STANDARD.

Essi sono degli identificatori che sono stati predefiniti implicitamente dal sistema. Tra di essi troviamo:

. nomi di tipi (esempio: INTEGER, REAL);
. nomi di costanti (esempio: TRU, FALSE, MAXINT (massimo intero trattabile));
. nomi di funzioni o di procedure standard (esempio: READ, WRITE (procedure), SIN, EXP (funzioni)).

Gli identificatori standard non sono parole riservate; essi possono essere usati per uno scopo diverso da quello previsto ridefinendoli. Questo modo di procedere e' pero' formalmente sconsigliato e nella pratica ci si comporta come se essi fossero parole riservate.
Le chiamate di procedure standard appaiono come delle nuove istruzioni Pascal e noi abbiamo utilizzato in questo modo le procedure di lettura e scrittura.

3.6. LE FUNZIONI

In una espressione aritmetica si puo' fare riferimento ad un certo numero di funzioni matematiche predefinite. Esempio:

```
Y:= X + 3*(Z+COS(2*X+1))
```

La chiamata della funzione si ottiene citandone il nome seguito dagli argomenti anche sotto forma di espressione. Per prima cosa viene calcolata l'espressione che funge da argomento della funzione, poi viene calcolata la funzione. Il valore ottenuto viene utilizzato nel proseguimento dei calcoli.

Sono disponibili le seguenti funzioni aritmetiche:

. con risultato reale usando argomenti sia reali che interi:

SIN(X)	seno	angolo in radianti
COS(X)	coseno	" " "
ARCTAN(X)	arcotangente	" " "
EXP(X)	esponenziale	e elevato a X
LN(X)	logaritmo neperiano	
SQRT(X)	radice quadrata	

. con risultato dello stesso tipo dell'argomento (reale o intero):

```
ABS(X)          valore assoluto
SQR(X)          quadrato        X elevato a 2
                (attenzione per chi e' abituato al
                 Basic, SQR del Basic diventa SQRT
                 in Pascal).
```

. conversione da reale a intero:

```
TRUNC(X)        parte intera con troncamento
ROUND(X)        parte intera con arrotondamento.
```

Queste due ultime funzioni sono indispensabili dato che una espressione reale non puo' essere assegnata ad una variabile intera. Per contro una espressione intera puo' essere assegnata ad una variabile reale senza problemi.

ESERCIZIO 3.9. - L'espressione I DIV J e' definita solo per I e J interi. Come fare per numeri reali?

ESERCIZIO 3.10. - Calcolare Z = X elevato a Y.

ESERCIZIO 3.11. - Tradurre in Pascal la formula per calcolare le radici di una equazione di secondo grado.

3.7. RIEPILOGO

Fino ad ora abbiamo passato in rassegna gli elementi base del linguaggio Pascal. Dal momento che sappiamo:

- leggere dei dati;
- fare dei calcoli;
- scrivere dei risultati;

possiamo cominciare a scrivere dei programmi in Pascal.

Bene, scriviamo un programma molto semplice! Vogliamo preparare un programma che legga il raggio di un cerchio, ne calcoli la superficie corrispondente e scriva il risultato del calcolo. E' molto facile:

```
READ (RAGGIO);
S := PI*SQR(RAGGIO);
WRITELN ('SUPERFICIE = ',S);
```

In effetti quello che abbiamo scritto puo' e deve essere perfezionato. Se si lavora in ambiente conversazionale si deve far precedere l'istruzione READ da una frase che

richieda il dato da leggere:

```
WRITE ('SCRIVI IL RAGGIO DI UN CERCHIO');
```

Se si lavora in ambiente non conversazionale si deve ricordare anche il raggio quando si scrive il risultato:

```
PAGE;
WRITELN ('RAGGIO = ', RAGGIO, ' SUPERFICIE = ', S);
```

Per semplicita' continuiamo a riferirci alla prima versione, quella adatta ad ambiente conversazionale. Anzi, d'ora in poi non torneremo piu' su questa distinzione, ma essa deve essere presente agli utilizzatori per ottenere dei buoni programmi.

Nel nostro programma mancano altre cose che sono essenziali.

3.8. STRUTTURA GENERALE DI UN PROGRAMMA

Le tre istruzioni sopra riportate danno la parte essenziale del nostro programma, ma esso e' incompleto.

Il gruppo di istruzioni eseguibili del programma deve essere compreso tra BEGIN e END. (La END finale del programma deve essere seguita da un punto).

Inoltre il programma deve avere una intestazione che rechi il suo nome:

Esempio: `PROGRAM CERCHIO ;`

L'intestazione deve precisare tra parentesi tutti i simboli esterni utilizzati, procedure e file. Spesso vengono precisati i file standard di lettura (INPUT) e di scrittura (OUTPUT):

Esempio: `PROGRAM CERCHIO (INPUT, OUTPUT);`

Con i compilatori UCSD questa dichiarazione non e' obbligatoria ed e' sufficiente la prima forma di intestazione riportata. In altre versioni non e' necessaria alcuna intestazione.

Tra PROGRAM e BEGIN si trovano le dichiarazioni che definiscono le variabili.

Devono essere indicate in ordine:

. Le dichiarazioni LABEL (vedere Capitolo 2, non se ne parlera' piu').

. Le dichiarazioni CONST delle costanti simboliche.

. Le dichiarazioni TYPE che definiscono i tipi non standard.

. Le dichiarazioni VAR che definiscono i tipi e la struttura delle variabili.

```
Esempio:  CONST PI = 3.14159265;
          VAR   RAGGIO, S  :REAL;
equivalente a: VAR RAGGIO: REAL;
                   S     : REAL;
```

Cosi' il nostro programma per il calcolo della superficie di un cerchio sara' finalmente completo:

```
PROGRAM CERCHIO;
CONST PI = 3.14159265;
VAR RAGGIO, S : REAL;
BEGIN
  READ (RAGGIO);
  S:=PI*SQR(RAGGIO);
  WRITELN ('SUPERFICIE = ', S);
END.
```

L'esempio illustra le regole per la punteggiatura ricordate nel Capitolo 2.

ESERCIZIO 3.12. - Scrivere il programma che risolve il problema inverso del precedente, cioe' calcolare il raggio di un cerchio di cui e' data la superficie.

ESERCIZIO 3.13. - Calcolare il volume della sfera.

ESERCIZIO 3.14. - Calcolare il volume di un cilindro di raggio R e di altezza H.

Per il momento utilizziamo solamente i tipi di dati standard. Vedremo, e questo e' molto nuovo, che l'utilizzatore puo' definire i propri tipi di dati non standard. Tra i tipi standard fino ad ora abbiamo trattato gli interi (INTEGER) e i reali (REAL).

3.9. TIPI STANDARD

INTEGER: sono i numeri interi. In dipendenza dal calcolatore usato gli interi possono essere di 16, 32 o 48 bit.

La costante standard MAXINT fornisce il piu' grande intero disponibile. MAXINT va da 32767 (per 16 bit) a 2.8*10

elevato a 14 (2.8E14, numero enorme).

ESERCIZIO 3.15. - Fate stampare MAXINT per sapere:
.1 se il vostro compilatore dispone di MAXINT;
.2 quali sono i numeri che potete utilizzare.

REAL: sono i numeri decimali. In dipendenza dal calcolatore utilizzato la precisione va da 9 a 14 cifre significative e l'ordine di grandezza va da 10 elavato a 30 a circa 10 elevato a 300.

Altri tipi standard ammessi dal Pascal sono i seguenti.

CHAR: e' di tipo CHAR una variabile che puo' contenere uno qualunque dei caratteri disponibili.

```
Esempio:  VAR   X :CHAR;
          X := 'A';
```

I caratteri possono essere confrontati: IF X = 'W' THEN..

I caratteri sono ordinati: 'A' < 'B' < 'C'.....
'0' < '1' < '2'.....

La funzione ORD(X) fornisce il numero corrispondente al carattere nel set di caratteri.
La funzione CHR(I) fornisce il carattere corrispondente al numero I, con 0 <I<63.
La funzione SUCC(X) fornisce il carattere che segue X.
La funzione PRED(X) fornisce il carattere che precede X.
Questo tipo e' un po' restrittivo se confrontato con la stringa del Basic. Alcuni compilatori consentono anche i tipi che seguono:

ALFA, catena di 10 caratteri (tale numero e' fisso, ma puo' variare tra le diverse implementazioni).
STRING, consentito da alcuni compilatori UCSD. E' simile alle variabili stringa del Basic con liberta' nel numero dei caratteri.

BOOLEAN: sono le variabili di tipo logico. Esse possono avere solo due valori che corrispondono alle costanti TRUE (vero) e FALSE (falso).

```
Esempio:  VAR COND   :BOOLEAN;
          COND := espressione logica
```

Le espressioni logiche sono formate da:

. costanti, variabili o funzioni logiche;

. condizioni (cioe' espressioni aritmetiche collegate da operatori relazionali, come: A < 2*X+3);

. espressioni collegate da operatori booleani:

Λ	(o AND)	che forma l'E logico; X AND Y e' vero se X e Y sono ambedue veri;
V	(o OR)	che forma l'O logico; X OR Y e' vero se almeno uno tra X e Y e' vero;
¬	(o NOT)	che e' la negazione.

In una espressione complessa, nella quale sono mescolati tutti i tipi di operatori, le regole per la priorita' sono le seguenti:

```
.piu' alti: NOT
            * / DIV MOD AND    (operatovi moltiplicativi)
            + - OR             (operatori addittivi)
.piu' bassi:= < > <= >= <>     (operatori relazionali)
```

Queste regole sono un po' diverse e, bisogna riconoscerlo, meno pratiche di quelle del Fortran e del Basic.

ESERCIZIO 3.16. - Controllate se C (tipo CHAR) e' una cifra.

ESERCIZIO 3.17. - Controllate se I e' divisibile per J.

FUNZIONI LOGICHE

Alcune funzioni hanno un valore logico:

ODD(X) e' vera se l'intero X e' pari.

EOF(file) e' vera se si e' arrivati alla fine del file. EOF viene usata solo trattando file di lettura standard INPUT.

EOLN(file) o solo EOLN: come sopra, ma per controllare la fine di una riga.

Abbiamo visto le basi della programmazione in Pascal. Ci resta ancora della strada da percorrere.

In particolare abbiamo visto solo le istruzioni di lettura e scrittura piu' elementari; ne vedremo di piu' nel capitolo dedicato ai file.

Per contro abbiamo gia' visto quasi tutto quello che riguarda le istruzioni per il calcolo.

Vediamo subito le istruzioni che permettono di strutturare il programma.

CAPITOLO 4

ISTRUZIONI DI STRUTTURAZIONE

4.1. IF...THEN...ELSE

Ecco un capitolo nel quale il Pascal si mostra nella sua originalita'. Le istruzioni di rottura di sequenza consentite in questo linguaggio sono quelle della programmazione strutturata.

IF...THEN...ELSE, che significa:
se e' cosi'...allora...se no...., si scrive:

```
    IF espressione logica THEN
                            istruzione 1
                          ELSE
                            istruzione 2;
    istruzione 3;
```

Se l'espressione logica e' vera viene eseguita prima l'istruzione 1 e poi l'istruzione 3. Se l'espressione logica e' falsa viene eseguita prima l'istruzione 2 e poi l'istruzione 3.

Esempio banale:

```
    IF       A>B     THEN
                       WRITELN ('A MAGGIORE DI B')
                     ELSE
                       WRITELN ('A MINORE O UGUALE A B');
    WRITELN ('TERMINATO');
```

```
si ottiene in stampa:   A MAGGIORE DI B
                        TERMINATO
oppure:                 A MINORE O UGUALE A B
                        TERMINATO
```

ESERCIZIO 4.1. - Vi siete dimenticati che esiste la funzione ABS e volete calcolare il valore assoluto di Y. Come fate?

IF NIDIFICATI

Istruzione 1 e istruzione 2 possono essere delle istruzioni composte, cioe' della forma:

```
      BEGIN i 1; i 2; i 3 END;
```

In un blocco di questo tipo si possono avere degli IF..THEN..ELSE o anche dei cicli REPEAT o WHILE.

Esempio: Risoluzione dell'equazione di primo grado: AX + B = 0. Se A<>0 X = -B/A. Se A = 0 scrivere "impossibile" o "indeterminata" a seconda del valore di B.

```
   IF A = 0 THEN
                IF B = 0 THEN WRITELN ('INDETERMINATA')
                         ELSE WRITELN ('IMPOSSIBILE')
            ELSE
                BEGIN
                    X := -B/A;
                    WRITELN ('RADICE = ', X)
            END;
```

Osservate l'impaginazione consigliata e la punteggiatura.

Si sarebbe potuto usare un BEGIN tra THEN e IF B = 0 e l'END corrispondente tra IMPOSSIBILE e ELSE. Provate a fare cosi' per esercizio.

ESERCIZIO 4.2. - Riscrivere il testo precedente incominciando con IF A<>0....
Scrivere il programma completo con dichiarazioni, letture,...

IF...THEN

Si ha un caso particolare quando la clausola ELSE e' vuota. Esso potrebbe essere scritto cosi':

```
            IF  condizione  THEN  istruzione 1
                            ELSE;
            istruzione 3;
```

E questo significa: eseguire istruzione 1 se la condizione risulta vera, poi eseguire istruzione 3; se la condizione risulta falsa eseguire subito istruzione 3.

Per ottenere lo stesso risultato e' meglio scrivere:

```
            IF  condizione  THEN istruzione 1;
            istruzione 3;
```

Si raccomanda di osservare l'uso della punteggiatura.

L'istruzione 1 puo' a sua volta essere una istruzione composta:

```
IF condizione THEN BEGIN i1; i2; i3 END;
istruzione 3;
```

Fate attenzione; in questo caso il posto del ; e delle parole BEGIN e END e' molto importante. Sbagliando si puo' cambiare completamente il significato.

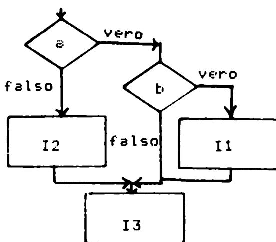
A volte puo' essere piu' prudente usare ELSE o BEGIN...END. D'altra parte nei casi di nidificazione di IF con ELSE con IF senza ELSE, un ELSE si riferisce sempre all'ultimo IF incompleto trovato.

Il modo di scrivere le istruzioni puo' aiutarne la comprensione, ma non e' esso che da' senso al programma. Esempio:

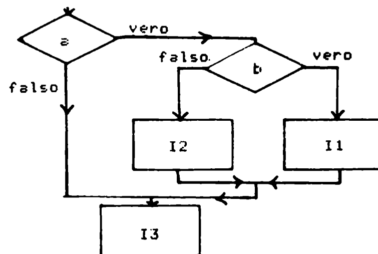
```
IF a THEN IF b THEN i1
     ELSE i2;
i3;
```

nelle intenzioni del programmatore corrisponde ad diagramma (a) sotto riportato e lo si deduce dal modo come sono state scritte le istruzioni. Il calcolatore da' invece l'interpretazione che corrisponde al diagramma (b) sotto riportato, cioe' il calcolatore si comporta come se si fosse scritto:

```
IF a THEN IF b THEN i1
                ELSE i2;
i3;
```



(a)



(b)

ESERCIZIO 4.3. - Scrivere il testo precedente in modo che per il calcolatore corrisponda al diagramma (a) sopra riportato.

4.2. I CICLI WHILE E REPEAT

Il Pascal consente tre diverse strutture per i cicli. Di queste due sono proprie della programmazione strutturata; la terza serve soprattutto per ricerche in tabelle.

Le prime due sono:

```
REPEAT                    cioe':      RIPETERE istruzioni
  istruzioni                          FINO A QUANDO la con=
UNTIL condizione                      dizione di arresto
                                      sia verificata.
```

e

```
WHILE condizione          cioe'       FINTANTO CHE la condi=
DO istruzioni                         zione di continuazione
                                      e' soddisfatta
                                      ESEGUIRE istruzioni.
```

Si potrebbe pensare che:

```
 REPEAT                   e           WHILE cond. DO
     i1;                                          BEGIN i1;
     i2                                                 i2
 UNTIL NOT cond.                                  END
```

siano del tutto equivalenti. Questo non e' completamente vero. La differenza si manifesta quando si arriva al ciclo con la condizione d'arresto gia' soddisfatta (o con la condizione di continuazione non piu' verificata).

In questo caso con REPEAT le istruzioni del ciclo sono eseguite una volta, mentre con WHILE esse non sono eseguite del tutto.

Questo diverso comportamento dipende dal fatto che con REPEAT il controllo per la continuazione si trova dopo le istruzioni del ciclo (controllo in coda), mentre con WHILE questo controllo si trova prima delle istruzioni del ciclo (controllo in testa) e quindi ci si accorge subito che la condizione per l'arresto e' gia' verificata (confrontate con i diagrammi del Cap. 1).

Notate, dal punto di vista della punteggiatura, che se con WHILE si ha piu' di una istruzione da ripetere, si deve usare BEGIN...END.

E'. necessario, sia con WHILE che con REPEAT, che l'esecuzione delle istruzioni fondamentali del ciclo modifichi il risultato della condizione, altrimenti il ciclo

durera' all'infinito.

Esempio: Spesso si devono elaborare dei numeri man mano che si leggono. L'elaborazione deve continuare fino a quando non si arriva alla fine del file.

. Leggere dei numeri, uno per scheda, e calcolarne la somma.

```
S:=0;
WHILE NOT EOF DO
                BEGIN READLN (N);
                       S:= S + N
                END;
WRITELN ('SOMMA = ',S);
```

Con alcuni calcolatori si dovra' scrivere: WHILE NOT EOF (INPUT) DO...

. Leggere dei numeri, uno per scheda, e stampare il piu' piccolo.

```
MIN:= MAXINT;
WHILE NOT EOF DO
                BEGIN READLN (N);
                       IF N < MIN THEN MIN:=N
                END;
WRITELN ('IL MINORE = ',MIN)
```

L'algoritmo e' evidente; si comincia con un minimo provvisorio uguale a MAXINT. Ogni volta che si trova un elemento minore, questo diventa il minimo provvisorio. Quando il ciclo e' terminato questo diventa il minimo definitivo. Notate il valore iniziale di MIN con MAXINT; in tale modo al primo confronto esso verra' eliminato. Se si elaborano numeri reali, analogamente si deve porre inizialmente in MIN un numero che sia sicuramente piu' grande di tutti gli altri possibili, come, per esempio, MIN:=1.0E20.

ESERCIZIO 4.4. - Stampare la media di una serie di numeri letti da schede.

ESERCIZIO 4.5. - Leggere dei numeri reali positivi minori di 10 elevato a 20, che si trovano uno per ogni scheda. Stampare il massimo ed il minimo dei numeri letti.

Per tutti gli esercizi di questo tipo scrivete un programma completo, con le dichiarazioni, e provatelo su un calcolatore dotato di compilatore Pascal.

Negli esempi si sono viste nidificazioni di controlli (test) e di cicli. E' permessa qualunque nidificazione. I cicli possono essere nidificati e possono essere mescolate le strutture REPEAT e WHILE. Questa e' una delle ragioni della potenza del Pascal. Con la scappatoia del BEGIN...END, una sequenza di istruzioni, per quanto sia complicata, diventa una istruzione composta e puo' essere incorporata a sua volta in una istruzione complicata.

Esempio: cercare il massimo comun divisore (MCD) tra due numeri A e B, usando l'algoritmo che segue. Si suppone che sia A>B. Sottrarre B da A fino a quando si ha un nuovo A<B. A questo punto sottrarre A da B fino a quando si ha un nuovo B <nuovo A. Ricominciare il calcolo e terminarlo quando si ha nuovo B = nuovo A. Questo valore e' il MCD cercato.

```
PROGRAM MCD;
VAR A,B,X,Y:INTEGER;        *X e Y servono per conser=
BEGIN                        vare A e B*
   READ (A,B); X:=A; Y:=B;
   REPEAT
       WHILE A>B DO A:= A-B;
       WHILE A<B DO B:= B-A;
   UNTIL A = B;
   WRITELN ('MCD DI ', X, ' E ', Y, ' = ',A)
END.
```

Abbiamo visto le strutture fondamentali della programmazione strutturata. Il Pascal offre, inoltre, altre strutture utili in alcuni casi.

4.3. IL CICLO FOR

Riprendiamo in esame l'esercizio per il calcolo della media (Eserc. 4.4.), ma supponiamo che non sia conosciuto a priori il numero NB degli elementi di cui calcolare la media. Si potrebbe scrivere:

```
BEGIN
READLN (NB);
S:=0; CONT:=0;
WHILE NOT (CONT>NB) DO
                  BEGIN READLN (N);
                        S:= S+N;
                        CONT:= CONT+1
                  END;
```

```
          .....
          .....
```

Il Pascal consente di abbreviare il programma incorporando in una sola frase le istruzioni di inizializzazione, incremento e controllo di CONT. Il programma diventa:

```
READLN (NB);
S:=0;
FOR CONT:= 1 TO NB DO
    BEGIN
      READLN (N);
      S:= S+N
    END;
MEDIA:= S/N;
```

Da quanto abbiamo visto risulta che la struttura FOR si incarica di gestire completamente la variabile CONT (chiamata indice corrente). Questa struttura si legge cosi': per CONT (indice corrente) = 1 (partenza) fino a NB (arrivo) (per valori interi consecutivi) fare......
La struttura assomiglia molto al FOR del Basic ed al DO del Fortran, con queste differenze:

. Se il valore di partenza dell'indice e' gia' maggiore del valore di arrivo quando si arriva al ciclo, questo non viene eseguito, mentre nei due linguaggi sopra citati esso e' eseguito almeno una volta sempre. Si puo' concludere che il FOR del Pascal e' equivalente alla struttura WHILE.

. La struttura e' meno potente, infatti e' limitata a valori interi e consecutivi dell'indice. E' sempre possibile arrangiarsi usando una variabile ausiliaria. La struttura FOR e' stata implementata nel linguaggio soprattutto per scandire tabelle; nel seguito si vedranno degli esempi di questo tipo.

Si devono dire ancora due cose:

. Il ciclo FOR assomiglia molto al WHILE che abbiamo gia' visto meno che per un dettaglio; all'uscita dal ciclo l'indice corrente e' indefinito nel Pascal standard, mentre con WHILE esso ha il valore di arrivo + 1.

. Esiste un'altra forma per la struttura FOR:

```
FOR I:= partenza DOWNTO arrivo DO.....
```

dove I assume successivamente i valori: arrivo-1, arrivo-2,....; cioe' corrisponde allo STEP -1 del Basic.

ESERCIZIO 4.6. - Preparare una tavola della funzione SIN X per X che varia da 0 a 92 gradi, scrivendo su due colonne, cosi':

```
*********************** ***********************
*   X   *   SIN X     * *   X   *   SIN X     *
*********************** ***********************
*   0   *             * *  46   *             *
*   .   *             * *   .   *             *
*   .   *             * *   .   *             *
*  45   *             * *  92   *             *
*********************** ***********************
```

Domanda: i valori di partenza e di arrivo possono essere dati sotto forma di espressione?
Certamente, essi devono essere interi e vengono calcolati una volta sola all'inizio del ciclo.

Ora cominciamo a occuparci di un programma sui numeri primi. Si tratta di stampare i numeri primi minori di un numero limite NLIM, letto da una scheda.
Seguiremo inizialmente un metodo molto semplice: cercare se il numero N (candidato ad essere primo) e' divisibile per ciascuno dei numeri minori di esso. Se si trova un divisore, questo significa che il numero non e' primo. Eliminiamo a priori il numero 1, sicuramente primo.

```
PROGRAM PRIMO1
VAR NLIM, N, I: INTEGER;     *I divisore di prova*
    PR        : BOOLEAN;     *indica se N primo*
BEGIN
   WRITELN ('NUMERI PRIMI');
   READ (NLIM);
   FOR N:= 2 TO NLIM DO
       BEGIN
          PR:= TRUE; I:=1;
          WHILE  (I<N-1) AND PR DO
                 BEGIN
                    I:= I+1;
                    IF N MOD I=0 THEN PR:=FALSE
                 END
          IF PR  THEN WRITELN (N)
       END;
    WRITELN ('NON NE SONO STATI TROVATI ALTRI')
 END.
```

E' semplice portare alcuni miglioramenti a questo programma. Questo modo di procedere serve per illustrare un metodo unniversale per lo sviluppo dei programmi, chiamato metodo di "raffinamento progressivo".
La programmazione strutturata rende facile questa

metodologia e questo e' uno dei suoi importanti vantaggi.

PRIMO RAFFINAMENTO

Il programma appena scritto trova 2 come numero primo, ma nel seguito non risulta piu' necessario controllare i numeri pari. Bastera' modificare il programma come segue:

. sacrificare il 2 e quindi cominciare con N:=3
. utilizzare un WHILE al posto del FOR:
  WHILE N<NLIM DO
. terminare questo ciclo con N:=N+2

SECONDO RAFFINAMENTO

E' inutile provare con un divisore I>radice(N) infatti se esiste un divisore A>radice(N), questo significa che N = A*B con B<radice(N), e quindi B sarebbe gia' stato trovato. Per questa ragione modificate il WHILE interno in:

```
WHILE (I<=ROUND(SQRT(N))) AND PR DO.
```

4.4. CASE

Questa istruzione permette di realizzare dei controlli a piu' di due uscite.

Essa si presenta nella forma:

```
CASE espressione-controllo    OF
   valore 1: istruzione 1;
   valore 2: istruzione 2;
   ....
   ....
   valore n: istruzione n
END;
```

Quando espressione-controllo assume il valore i, viene eseguita l'istruzione i (eventualmente con effetto nullo). I valori i devono essere delle costanti. Possono essere raggruppati piu' valori, se il corrispondente trattamento e' il medesimo.

Se espressione-controllo assume un valore che non e' compreso tra gli n valori considerati, il programma termina con segnalazione di errore. Espressione di controllo e costanti devono essere dello stesso tipo.

Questa istruzione e' piu' potente delle corrispondenti in Fortran (GOTO calcolato) e in Basic (ON x GOTO), dato che la

selezione avviene per valori che non devono essere per forza interi o per forza consecutivi.

Esempio:

Tra i dati caratteristici di un cliente di solito compare una lettera che denota lo sconto pattuito secondo questa regola:

- 'A' sconto 10%
- 'B' sconto 5% se l'importo supera 1 milione, altrimenti niente
- 'C' sconto 10%
- 'D' sconto 5%

```
VAR     LC: CHAR;
        MONTANTE, APAGARE: REAL;
.........
   CASE   LC  OF
        'A','C' : APAGARE:= 0.9*MONTANTE
        'B'     : BEGIN
                    IF MONTANTE > 1000000.0
                         THEN APAGARE :=0.95*MONTANTE
                         ELSE APAGARE :=MONTANTE
                  END;
         'D'    : APAGARE :=0.95*MONTANTE
     END;    *fine della scelta*
```

ESERCIZIO 4.7. - (Puramente scolastico) Sostituire:

```
IF c THEN i1
     ELSE i2;
```

con la struttura CASE.

Nota: Alcune implementazioni del Pascal dispongono della clausola "OTHERWISE":

```
CASE  X   OF
  X1 : i1
  X2 : i2
  ....
  Xn : in
OTHERWISE ix;
```

Se X non e' uguale ad alcuno degli xi allora viene eseguito ix.

4.5. GOTO

Come abbiamo gia' detto, ecco l'istruzione piu' contraria allo spirito del Pascal. Di fatto per scoraggiare al suo

uso, l'istruzione deve essere contrassegnata da una etichetta che deve essere dichiarata in una istruzione LABEL.

Esempio:

```
LABEL 1234;
.......
GOTO 1234;
```

Il solo uso valido del GOTO in Pascal e' per creare una uscita quando nel ciclo interno di una procedura si verifica una condizione del tutto eccezionale (spesso dovuta ad un errore). In sostanza e' necessario uscire dal ciclo prima del termine naturale delle iterazioni.

Facciamo notare che alcuni compilatori Pascal (Texas Instruments) hanno introdotto una istruzione speciale per uscire dai cicli: la ESCAPE, per evitare il GOTO.

ESERCIZIO RIEPILOGATIVO 4.8. - Leggere una frase (le parole sono separate da spazi, tutta la frase sta su una scheda e termina con un punto). Stampare il numero di vocali ed il numero di consonanti presenti.

ESERCIZIO RIEPILOGATIVO 4.9. - Risolvere l'equazione di secondo grado:

A*X**2 + B*X + C = 0.

CAPITOLO 5

TIPI DI DATI

5.1. PREMESSA

Lasciando da parte le procedure, noi abbiamo ora in mano tutti gli strumenti per costruire dei programmi, almeno per quanto concerne le istruzioni eseguibili.

Sappiamo quanto serve per il trattamento elementare dei dati. Sappiamo leggere dei dati da elaborare, eseguire delle operazioni su di essi, e, cosa molto importante per l'utente, stampare dei risultati.

Queste elaborazioni elementari formano delle sequenze di operazioni; sappiamo anche passare da una sequenza ad un altra, o, in certe condizioni, scegliere tra due sequenzè possibili.

Inoltre noi sappiamo ripétere una sequenza di istruzioni un determinato numero di volte, fino a quando risulta verificata una condizione. Questa capacita' del calcolatore di ripetere la stessa sequenza di operazioni molte volte, dopo che il programmatore l'ha scritta una sola volta, e' uno dei fattori che lo rendono potente.

Per il momento noi sappiamo elaborare 4 tipi di dati, i 4 tipi standard del Pascal, che, come negli altri linguaggi, sono: gli interi (INTEGER), i reali (REAL), i booleani (BOOLEAN) e i caratteri (CHAR).

Ebbene, e questo e' uno dei suoi pregi, il Pascal sa trattare molti altri tipi di dati. Alcuni tipi possono essere creati dal programmatore per soddisfare le sue esigenze.

Infatti, partendo dai tipi elementari di cui sopra (chiamati tipi scalari), e' possibile creare dei tipi di dati strutturati che permettono di trattare in blocco un gran numero di dati elementari. E, dato che questa costruzione e' recursiva, sara' possibile, a poco a poco, creare tipi di dati molto elaborati.

Questo e' uno dei grandi fattori della potenza del linguaggio Pascal. In effetti, molto spesso, un problema e' per tre quarti risolto quando si e' trovata la forma adatta

per rappresentare i dati da trattare. Al contrario, senza il tipo di dati adatto, puo' essere necessario combattere con algoritmi molto complicati.

Ci occuperemo ora di due categorie di tipi di dati. Per prima cosa dei tipi elementari (o scalari), dove si tratta un dato elementare alla volta. E' il caso dei tipi standard del Pascal: un dato I di tipo intero permette di trattare con il nome I un solo numero intero.

Dopo tratteremo la categoria dei tipi strutturati, dove, con il nome di una variabile, e' trattata una grande quantita' di dati che hanno delle relazioni tra loro e che appartengono ciascuno ai tipi scalari precedentemente definiti.

Per esempio, il Pascal sa trattare delle tabelle (ARRAY) come il Fortran o il Basic. C'e' una piccola difficolta' per i principianti: la struttura puo' essere indifferentemente definita su una variabile o su un tipo. Si potra' dire che la variabile M e' una tabella di, per esempio, 10 x 20 interi. Oppure il tipo matrice potra' essere definito come una tabella di 10 x 20 interi e poi dire che la variabile M appartiene al tipo matrice.

Infine, questo concetto di tipo e' cosi' potente, che il Pascal vi fa rientrare il concetto di file ("file" e' un tipo strutturato particolare, vedi Cap. 7) e la gestione dinamica dei dati (con il tipo puntatore, vedi Cap. 8).

Naturalmente le definizioni dei tipi si fanno con delle dichiarazioni particolari, le dichiarazioni TYPE. La dichiarazione TYPE attribuisce un identificatore, il nome del tipo, ad una descrizione del tipo stesso, cioe' di dati che possono essere qualificati come appartenenti a quel tipo.

Vediamo ora i tipi scalari.

5.2. TIPI SCALARI

L'utilizzatore puo' definire un tipo scalare non standard semplicemente definendo il valore che puo' assumere un oggetto appartenente a quel tipo.

Per esempio, se si vuole definire il tipo booleano, esso e' un dato che puo' avere il valore TRUE (vero) o FALSE (falso).

In Pascal ci sono due modi per definire un tipo scalare: la semplice lista dei possibili valori, o la definizione di un intervallo per mezzo dei valori di un tipo gia' precedentemente definito.

5.3. TIPI DEFINITI MEDIANTE LISTA

Viene fornita la lista dei simboli e delle costanti che fanno parte del tipo:

```
TYPE ANIMALE = (CANE, GATTO, VACCA, CAVALLO);
     GIORNI  = (LUNEDI', MARTEDI', MERCOLEDI', GIOVEDI',
                VENERDI', SABATO, DOMENICA);
```

Nell'esempio precedente, CANE diventa un nome di costante e questa costante e' uno dei valori permessi nel tipo ANIMALE. Sara' allora possibile scrivere:

```
VAR BESTIA : ANIMALE
    BESTIA := CANE
IF BESTIA = VACCA THEN....
```

La cosa piu' interessante della definizione di tipo e' che il Pascal effettua ogni volta delle verifiche di conformita' sul tipo. Cosi', se, per errore, si scrive:

```
BESTIA := MERCOLEDI';
```

viene segnalato un messaggio di errore.

Gli elementi che appartengono ad un tipo cosi' definito vengono ordinati in modo conforme alla definizione iniziale. Cosi' nell'esempio precedente:

```
CANE < GATTO
GIOVEDI' > LUNEDI'
```

ESERCIZIO 5.1. - Quale sara' la stampa ottenuta da:

```
M := MERCOLEDI' ; D := DOMENICA;
WRITELN (D<M);
```

Quale dichiarazione e' necessaria in questo programma?

Le funzioni ORD, PRED e SUCC sono valide per gli elementi del tipo definito mediante lista.

ORD e' il numero d'ordine dell'elemento nella lista degli elementi del tipo, cominciando da 0. Cosi':

```
ORD (GATTO) = 1, ORD (LUNEDI') = 0
```

SUCC e' l'elemento seguente:

```
SUCC (GATTO) = VACCA
```

PRED e' l'elemento precedente:

```
PRED (GATTO) = CANE
```

Attenzione: SUCC (dell'ultimo) e PRED (del primo) non sono definiti.

Si possono realizzare dei cicli come i seguenti:

```
VAR I : GIORNO;
FOR I = MARTEDI' TO SABATO DO....
```

Analogamente per WHILE:

```
WHILE I < SABATO DO
                 BEGIN
                 ...
                 ...
                 I := SUCC (I)
                 END;
```

quest'ultimo funziona dato che la valutazione di SUCC e' sempre possibile. Se si fosse scritto: WHILE I <= DOMENICA, si sarebbe avuto un errore.

Si vede dunque quale flessibilita' fornisca questa possibilita' del Pascal di definire tipi di oggetti secondo le necessita' dell'utilizzatore e di poter trattare questi tipi con le operazioni standard del linguaggio.

Esiste pero' una limitazione che "demolisce" quasi tutto l'edificio. Supponete che dopo aver scritto:

```
VAR K : GIORNO;
K := MARTEDI';
```

voi tentiate di scrivere: WRITE (K). Non ottenete MARTEDI'. Cioe' o non otterrete alcunche' o otterrete un messaggio diagnostico.

Analogamente, in lettura, un nome di costante come MARTEDI' non potra' essere presente su una scheda; cioe' non si potra' avere:

```
VAR J : GIORNO
```

```
READ (J);
```

e leggere da una scheda MARTEDI'.

Vedremo piu' avanti come si puo' superare questa difficolta' usando una catena di caratteri. Il fatto di dover ricorrere a questo limita molto l'interesse dei tipi definiti mediante lista.

Un'ultima osservazione: una stessa costante non puo' essere assegnata a due diversi tipi; cosi' non si puo' scrivere:

```
TYPE GIORNO = (LUN, MART, MERC, GIOV, VEN, SAB, DOM);
     WEEKEND = (SAB, DOM);
```

questo non e' consentito.

I tipi definiti mediante intervallo ci forniscono la soluzione dei problemi ora posti.

5.4. TIPI DEFINITI MEDIANTE INTERVALLO

Un tipo puo' essere definito valido per i valori contigui di un intervallo per mezzo dei valori di un tipo precedentemente definito.

Esempio:

```
TYPE ANNI = 0...120:
     LAVORATIVI = LUN...VEN;
     WEEKEND = SAB...DOM;
```

Non e' possibile, disgraziatamente, definire un intervallo di numeri reali.

L'interesse di una dichiarazione di questo tipo, qualora si sappia a priori che il tipo di dati considerati non deve uscire da un intervallo noto, e' di procurare un messaggio di errore in caso contrario.

Esempio: Durante l'acquisizione di dati riguardanti degli individui, in particolare la loro eta', se si definisce:

```
VAR ETA' : ANNI;
```

e l'operatore scrive 150 invece di 15, si ha un messaggio di errore.

Con un altro linguaggio di programmazione sarebbe stato necessario introdurre un controllo esplicito sulle eta' introdotte. Infatti non possono essere accettati dati errati negli archivi.

Discutiamo il valore di questa possibilita' in Pascal. Essa evita di mettere un controllo esplicito ogni volta che si riceve una variabile del tipo considerato, ma bisogna pensare a definire il tipo intervallo.
Cosa succede in caso di errore?
Con il tipo definito dal Pascal il programma si ferma con un messaggio di errore. Con un controllo esplicito di veridicita' il programma puo' prendere una decisione opportuna senza fermarsi. Quest'ultimo comportamento e' di gran lunga preferibile in fase di acquisizione dei dati: in caso di errore di battitura non si deve arrestare il programma, bisogna inviare un messaggio all'operatore chiedendogli di correggere il dato.

ESERCIZIO 5.2. - Definire un tipo LETTERA e un tipo CIFRA.

Abbiamo appena visto due possibilita' del Pascal a priori seducenti, ma per le quali sono state messe in evidenza alcune limitazioni.

Prima di passare ai tipi strutturati, facciamo un' ultima precisazione.

Se un tipo deve essere utilizzato una sola volta (cioe' in una sola dichiarazione VAR), e' inutile attribuire un nome al tipo; basta definirlo nella dichiarazione VAR, cosi':

```
VAR ETA' : 0...120;
```

5.5. TIPI STRUTTURATI

Siamo ora arrivati ai tipi di dati dove un oggetto e' formato da piu' elementi. E' proprio qui che la potenza del linguaggio e l'immaginazione del programmatore possono essere pienamente utilizzate.

Il primo tipo strutturato che esamineremo e' la tabella (ARRAY). Esso e' il solo che ha un equivalente in Fortran e in Basic, ma in Pascal ha alcune possibilita' in piu'; in particolare l'assegnazione dei valori a ciascun elemento della tabella mediante una sola istruzione.

Il tipo che esamineremo dopo e' il RECORD. Diversamente

dalla tabella, esso consente di raggruppare elementi di tipo diverso. Esso ha un equivalente in PL/1 (le "strutture") e in Cobol (i livelli gerarchici dei dati), ma non in Fortran e in Basic.

Infine esamineremo il tipo SET (insieme) che e' proprio solo del Pascal. Nei capitoli seguenti vedremo poi il tipo FILE (file di dati) e un suo caso particolare TEXT, e il tipo POINTER (puntatore) che permette di gestire i dati in modo dinamico.

Tutti i tipi strutturati si costruiscono come raggruppamenti di elementi che appartengono a un tipo scalare precedentemente definito. Si dice che questo costituisce il tipo base della struttura.

La costruzione puo' essere disposta a piani o gerarchicizzata: si costruira' un tipo molto complesso a poco a poco, raggruppando degli elementi, poi raggruppando dei gruppi,ecc...

5.6. TYPE ARRAY

Supponiamo che si vogliano trattare dei vettori nello spazio a tre dimensioni. E' naturale che si desideri, sia raggruppare le tre componenti di un vettore e definire il vettore V, che accedere a una componente Vi.

Il Pascal, come del resto il Fortran, il Basic e altri linguaggi, lo consente. Si ha tuttavia una prima differenza.

In Fortran o in Basic si deve dichiarare nello stesso modo ciascun vettore:

```
DIM V(3), W(3)....
```

In Pascal si puo' fare cosi':

```
VAR V : ARRAY [1..3] OF REAL;
    W : ARRAY [1..3] OF REAL;
```

ma se si hanno piu' vettori dello stesso tipo, si puo' creare un tipo "vettore a 3 dimensioni":

```
TYPE VECT3 = ARRAY [1..3] OF REAL;
```

e poi:

```
VAR V : VECT3;
    W : VECT3;
    ......
```

(Notate l'uso dei : e di =).

Una componente del vettore si individua con:

V [2] oppure V [I]

e la parte entro le parentesi quadre si chiama indice.

Per i sistemi che non dispongono delle parentesi quadre si utilizza "(." e ".)".

E' possibile utilizzare una componente:

X := 3*V[2] o assegnargli un valore:

V[2] := Y - EXP (U)

La tabella puo' anche essere richiamata nella sua totalita' in due operazioni:

- l'assegnazione V := W
  ricopia il vettore W nel vettore V;

- la comparazione (solamente per = o diverso (<>)):
  IF V = W THEN...

In tutti i casi bisogna che i tipi concordino, cioe' abbiano le stesse dimensioni e lo stesso tipo di base.

Per quanto riguarda le operazioni di lettura e scrittura il comportamento dipende in buona misura dal sistema che si ha a disposizione. Si devono distinguere le tabelle di numeri e le stringhe di caratteri:

```
TYPE CHAINE = ARRAY [1..22] OF CHAR;
VAR MESSAGGIO : CHAIN;
```

I possibili comportamenti sono i seguenti:

1 - Non e' possibile ne' la lettura ne' la scrittura delle tabelle in blocco.

2 - La lettura e/o la scrittura in blocco e' possibile solo per le tabelle di numeri. Si deve notare che il problema e' piu' difficile per le stringhe di caratteri e, in conseguenza, si ha l'impossibilita' di una lettura in blocco di una stringa nel Pascal standard (in effetti e' impossibile sapere se gli spazi iniziali di una scheda devono essere considerati o meno). Per contro e' sempre possibile stampare una stringa di caratteri in blocco:

```
   WRITE ('STRINGA DI CARATTERI');
o  MESSAGGIO := 'STRINGA DI CARATTERI';
   WRITE (MESSAGGIO);
```

3 - Nel Pascal UCSD, dove il tipo STRING e' standard, e' naturalmente possibile leggere e scrivere un oggetto di tipo STRING.

In tutti i casi, la lettura o la stampa puo' essere sempre ottenuta elemento per elemento con l'aiuto di un ciclo FOR:

```
FOR I = 1 TO INDICEMAX DO
        BEGIN
           READLN (TAB [I]);
           WRITELN (TAB [I])
        END;
```

ESERCIZIO 5.3. - Suggerite le dichiarazioni da premettere all'esempio sopra riportato. Quanti elementi si leggono da ogni scheda?

Questo modo di procedere e' il solo che permette di controllare al meglio cio' che succede alla fine della linea, sia in ingresso che in uscita.

Questo e' il modo utilizzato dal Basic standard. Il Basic, se possiede le operazioni MAT, puo' operare in blocco in tutti i modi sulle tabelle. Esso va quindi piu' lontano del Pascal in questo.

ESERCIZIO 5.4. - Supponiamo che gli elementi della tabella precedentemente citata, TAB, siano 5 a 5 sulle schede dei dati. Stampateli 10 per ogni linea nel formato:

```
     -sx.yyy---sx.yyy---.....  dove - sta per spazio e
                                    s per segno.
```

Negli esercizi che seguono supporremo che, salvo esplicita indicazione, siano possibili l'immissione e la stampa delle tabelle, per mezzo di un metodo disponibile sul Pascal in uso.

Ma noi non abbiamo ancora visto cio' che rende originali le tabelle in Pascal. Se riguardiamo la dichiarazione di tabella dell'inizio del paragrafo, vediamo che tutto avviene come se noi avessimo scritto:

```
TYPE VETTORE = ARRAY [tipo intervallo] OF REAL;
```

Questo e' precisamente quello che si puo' fare. Piu' generalmente, e' possibile indicizzare per mezzo di un qualunque tipo intervallo o lista:

```
TYPE DIMENSIONI = 1..10;
     GIORNO = (LUN, MART, MERC, GIOV, VEN, SAB, DOM);
     VETTORE = ARRAY [DIMENSIONI] OF REAL;
VAR V : VETTORE;
    SILAVORA : ARRAY [GIORNO] OF BOOLEAN;
    J : GIORNO;
```

e si potra' avere:

```
FOR J := LUN TO VEN DO
    BEGIN
        SILAVORA [J] := TRUE
    END;
SILAVORA [SAB] := FALSE;
SILAVORA [DOM] := FALSE;
 ...
 ...
IF SILAVORA [J] THEN ...
```

Questo mostra bene come sia comodo questo modo di usare le tabelle. Trattiamo nel seguito alcuni esempi per acquistare un po' di pratica.

5.7. PRODOTTO SCALARE DI DUE VETTORI

La formula matematica e' la seguente:

$$U.V = \sum_{i=1}^{n} u_i \, v_i \qquad \text{dunque:}$$

```
CONST N = 20;
TYPE  DIMENSIONI = 1..N;
      VETTORE    = ARRAY [DIMENSIONI] OF REAL;
VAR   U, V : VETTORE;
      UV   : REAL;
      I    : DIMENSIONI;
UV := 0;
FOR   I := 1 TO N DO
      BEGIN
         UV := UV + U[I] * V[I]
      END;
```

5.8. RICERCA DI UN ELEMENTO IN UNA TABELLA

Ricerca della presenza nella tabella TAB del valore X e del valore corrispondente IT dell'indice. Se l'elemento non e' presente, la risposta puo' essere, per esempio, IT = N+1 (N definisce la dimensione della tabella).

Prendiamo come esempio una tabella di caratteri che costituisce una parola. Ricerchiamo se la lettera 'K' e' presente.

```
CONST N = 10;    *parola lunga al massimo 10 lettere*
      X = 'K';   *lettera cercata*
TYPE DIM = 1..N;
     PAROLA = ARRAY [DIM] OF CHAR;
VAR  TAB : PAROLA;
     I   : DIM;
     IT  : INTEGER;
....             *qui si metteranno le istruzioni per
....              la lettura di TAB*
IT := N+1;
FOR  I = 1 TO N DO
     BEGIN
        IF TAB [I] = X THEN IT := I
     END;
IF IT <= N THEN WRITELN ('SI HA UN K PER INDICE = ',IT)
           ELSE WRITELN ('NON SI HA UN K');
```

Questo programma esemplifica la "ricerca sequenziale". Essa non e' la piu' efficiente, ma viene usata spesso, soprattutto per i file.

Prima di esporre un metodo piu' efficiente (applicabile solo alle tabelle ordinate), corregeremo due difetti del programma precedente.

ESERCIZIO 5.5. - Nel programma precedente, se si hanno piu' lettere 'K' nella parola, IT mantiene il valore relativo all'ultima ripetizione della lettera. In certi casi sara' richiesto il valore relativo alla prima apparizione della lettera cercata. Per questo, restando nell'ambito della programmazione strutturata, si deve introdurre una variabile booleana TROVATO e introdurre un ciclo UNTIL TROVATO (non val la pena di continuare a scandire la tabella quando quello che si cercava e' gia' stato trovato). Questo va bene, ma se l'elemento e' assente? Il metodo consiste nel porre una "sentinella", cioe' un elemento uguale al valore cercato nel rango N+1 (il tipo dovra' essere modificato in

conseguenza). A questo punto lo stesso test determina l'arresto ed il fatto che l'elemento cercato e' stato trovato. Provate.

5.9. RICERCA DICOTOMICA

Supponiamo di avere una tabella di 100 elementi, ordinati in modo crescente. Vogliamo sapere se e' presente un elemento uguale a 38.5; attenzione, il controllo non puo' essere fatto per perfetta uguaglianza trattandosi di numeri reali, a causa della limitata precisione di cifre dei calcolatori. I controlli vanno realizzati nella forma:

```
ABS (X-Y) < Z    oppure    (X > Y-T) AND (X< Y+T)
```

Noi procederemo, passo a passo, dividendo l'intervallo di ricerca per 2.

Per esempio, nel primo passaggio noi esamineremo l'elemento TAB [50]. Se TAB [50] = X il procedimento e' terminato. Se invece TAB [50] > X, allora si deve continuare a cercare dall'indice 1 all'indice 50, e se TAB [50] < X si deve continuare a cercare dall'indice 50 all'indice 100.

```
CONST       N = 100;
            X = 38.5;
VAR         INF, SUP, IT : INTEGER;
            TROVATO : BOOLEAN;
            TAB     : ARRAY [1..N] OF REAL;
INF := 1;
SUP := N;
TROVATO := FALSE;
REPEAT
    IT := (INF + SUP) DIV 2;      *nuovo limite*
    IF (TAB [IT] > X-0.0001) AND (TAB [IT] < X+0.0001)
                   THEN TROVATO := TRUE
                   ELSE
                     IF TAB [IT] < X THEN INF := IT+1
                                     ELSE SUP := IT-1
UNTIL TROVATO OR (INF > SUP);
```

ESERCIZIO 5.6. - Trovare l'elemento massimo in una tabella di numeri interi positivi (stesso metodo dell'eserc. 4.5.).

5.10. ORDINAMENTO PER SMISTAMENTO

Noi non possediamo ancora tutte le informazioni che ci permetterebbero di comprendere in tutti i suoi dettagli il programma dell'esercizio 2.1.. Tuttavia noi comprendiamo il metodo nelle sue grandi linee.

Si tratta del metodo dell'ordinamento a bolle; per mettere in ordine una tabella si scambiano tra loro due elementi consecutivi se essi non sono in ordine. Si prendono in considerazione tutte le coppie possibili durante una ispezione della tabella e se si e' avuto almeno uno scambio si inizia una nuova ispezione.

Questo metodo e' poco efficace per tabelle di grandi dimensioni. Allora noi cercheremo di introdurre un nuovo metodo, il metodo di Shell, dal nome del suo inventore (Shell Sort in inglese).

Shell e' partito dalla seguente osservazione: quello che non va bene nel metodo di ordinamento a bolle e' che l'intervallo tra gli elementi scambiati e' sempre piccolo; esso e' sempre uguale a 1. Supponiamo, mettendoci nel caso piu' favorevole, che l'elemento maggiore si trovi in testa alla tabella alla partenza. Esso deve trovarsi in fondo alla tabella alla fine dell'ordinamento. Esso arriva in fondo dopo N-1 confronti e N-1 scambi, mentre se si usassero degli intervalli di N/2 basterebbero 2 confronti e 2 scambi.

Il metodo di Shell consiste nel considerare all'inizio dei grandi intervalli tra gli elementi scambiati e nel diminuire poi gli intervalli. Il primo intervallo considerato sara' N/2 (N grandezza della tabella), poi esso sara' diviso per 2 ogni volta (fino a quando diventa uguale a 1).

Per ogni valore dell'intervallo e' realizzato una specie di ordinamento a bolle dove il confronto si opera tra TAB [I] e TAB [I + SCARTO] includendolo in un ciclo che opera sull'intervallo variabile.

```
PROGRAM SHELL;          *ordin. di Shell in modo crescente*
   CONST NB = 50;       *numero degli elementi*
   VAR   SCA,           *indicatore di scambio*
         SCARTO,        *scarto tra gli elementi confr.*
         I   : INTEGER;
         TAB : ARRAY [1..NB] OF INTEGER;
   PROCEDURE SCAMBIO;
      VAR    X : INTEGER;
      BEGIN
         X := TAB [I];
         TAB [I] := TAB [I+SCARTO];
         TAB [I+SCARTO] := X
      END;                *fine dello scambio*
```

```
BEGIN                           *inizio programma principale*
   WRITELN ('TABELLA NON ORDINATA');
   FOR I := 1 TO NB DO
         BEGIN
            READ (TAB [I]);
            WRITE (TAB [I] : 5)
         END;
   WRITELN; WRITELN;
                                 *inizio ordinamento*
   SCARTO := NB;
   REPEAT                        *ciclo sui diversi scarti*
      SCARTO := SCARTO DIV 2;
      REPEAT                     *ciclo sui passi*
         SCA := 0;
         FOR I := 1 TO NB - SCARTO DO
             BEGIN
                IF TAB [I] > TAB [I+SCARTO]
                       THEN  BEGIN
                                SCA := 1;
                                SCAMBIO
                             END
             END
      UNTIL SCA = 0
   UNTIL SCARTO = 1;
   WRITELN ('TABELLA ORDINATA');
   FOR I := 1 TO NB DO WRITE (TAB [I] : 5);
END.
```

ESERCIZIO 5.7. - Partendo dalla tabella contenente gli 8 elementi: 9, 8, 4, 6, 3, 1, 2, 5; fate manualmente un ordinamento in ordine crescente usando il metodo a bolle e poi rifatelo usando il metodo di Shell. Confrontate il numero di confronti e di scambi che avete operato. Non dimenticate l'ultimo passaggio nel quale non operate scambi.

Abbiamo insistito su questi algoritmi, per prima cosa perche', anche se il Pascal ha altri tipi di dati strutturati, la tabella e' assolutamente fondamentale, e poi perche' questo ci ha permesso di mettere in evidenza come il linguaggio si presti bene per sviluppare questi algoritmi.

Ma abbiamo ancora qualcosa da vedere circa le tabelle.

5.11. TABELLE PACKED

In tutte le dichiarazioni di tabelle si puo' aggiungere l'attributo PACKED (compatte). Esempio:

```
TYPE ALFA = PACKED ARRAY [1..10] OF CHAR;
```

Una tabella packed puo' essere usata esattamente come una tabella normale. La differenza sta solo nello spazio di memoria occupato.

Si sa che un carattere occupa 8 bit. Supponiamo di utilizzare una macchina a 32 bit come un 360. Se dichiariamo un ARRAY [ ] OF CHAR normale, ogni carattere sara' messo in una parola; se lo dichiariamo PACKED i caratteri saranno raggruppati 4 per parola.

Quindi la parola chiave PACKED permette di risparmiare memoria. Questo pero' va a scapito della velocita' di calcolo; infatti, per accedere ad un elemento di una tabella compatta, si deve trovare dapprima in quale parola si trova l'elemento e poi individuarlo all'interno della parola.

La tabella deve essere quindi impaccata e disimpaccata. Esistono due procedure, la PACK e la UNPACK, che consentono di farlo. Esempio:

```
VAR A : ARRAY [1..N] OF TYPEDEA;
     PA: PACKED ARRAY [1..N] OF TYPEDEA;
UNPACK (PA,A,I)  equivalente al ciclo:
   FOR K=1 TO N DO A[K-1+I] := PA [K];
PACK (A,I,PA)    equivalente al ciclo:
   FOR K=1 TO N DO PA [K] := A [K-1+I];
```

ESERCIZIO 5.8. - Non basta fare A:= PA; ?

NOTA: Le costanti stringa di caratteri (es. 'BUONGIORNO') sono del tipo PACKED ARRAY [1..num-car.] OF CHAR.

Ne consegue che se si ha:

```
VAR U : ARRAY [1..10] OF CHAR;
     V : PACKED ARRAY [1..10] OF CHAR;

U := 'BUONGIORNO' oppure IF U = 'BUONGIORNO'....
non e' corretto, mentre:
V := 'BUONGIORNO' oppure IF V = 'BUONGIORNO'....
e' corretto.
```

Ritornando al problema della lettura delle tabelle, notiamo che la lettura delle stringhe di caratteri risulta piu' semplice se la variabile e' PACKED.

5.12. TABELLE MULTIDIMENSIONALI

L'elemento base di una tabella puo' essere una tabella:

```
TYPE VETTORE = ARRAY [1..N] OF REAL;
     MATRICE = ARRAY [1..N] OF VETTORE;
VAR  V : VETTORE;
     M : MATRICE;
```

Un elemento della matrice si indica cosi': M [I] [J] := 3;

M [J] e' allora un vettore che fa parte della matrice, considerato in blocco.
Si puo' scrivere: M [J] := V;

Poter fare questo rende il Pascal superiore al Basic e al Fortran, ma esso puo' procedere cosi' solo per l'ultima dimensione introdotta, cioe' per ARRAY OF piu' a sinistra in una dichiarazione del tipo:

```
TYPE X = ARRAY [S] OF ARRAY [S] OF ARRAY [S]
```

Un linguaggio come l'APL consente di fare cosi' per tutte le dimensioni.

M [I] [J] puo' altrettanto chiaramente essere scritto:

M [I,J].

La matrice puo' essere dichiarata nella forma:

```
TYPE MATRICE = ARRAY [1..N, 1..N] OF REAL;
```

ESERCIZIO 5.9. - Avendo due matrici [1...N, 1..N], A e B, calcolare il loro prodotto C. Si ottiene una matrice [1..N, 1..N] dove il generico elemento e':

$$C_{IJ} = \sum_{K=1}^{N} A_{IK} * B_{KJ}$$

Questo esercizio e' classico. Si ha un doppio ciclo per gli indici I e J per ottenere tutte gli elementi C. All'interno di esso vengono fissati I e J e viene calcolato un valore C con un nuovo ciclo dove l'indice e' K. Riportiamo il programma senza le istruzioni di ingresso e uscita.

```
PROGRAMMA PRODMATRICI;
   CONST N = 10;
```

```
    TYPE MATRICE = ARRAY [1..N, 1..N] OF REAL;
    VAR A, B, C : MATRICE;
        I, J, K : 1..N;
  BEGIN
    FOR I := 1 TO N DO
        FOR J := 1 TO N DO
            BEGIN
               C [I,J] := 0.0;
               FOR K := 1 to N DO
                   C [I,J] := C [I,J] + A [I,K] * B [K,J]
            END
  END.
```

Con questo programma viene messa in evidenza una lacuna del Pascal in confronto ad altri linguaggi. Non e' possibile avere dimensioni variabili. In questo caso sarebbe stato conveniente poter scrivere una PROCEDURE generale valida per il prodotto di qualsiasi coppia di matrici.

ESERCIZIO 5.10. - Preparare un programma che trasformi una matrice A nella sua trasposta :

$$A^T \quad \text{dove} \quad A^T_{IJ} = A_{JI}$$

Proponiamo una soluzione al problema delle costanti simboliche che formano un tipo. Ci serviamo di una tabel'a di stringhe di caratteri PACKED.

```
  TYPE     STRINGA = PACKED ARRAY [1..3] OF CHAR;
           GIORNO = (LUN, MAR, MER, GIO, VEN, SAB, DOM);
           NUMGIORNO = ARRAY [GIORNO] OF STRINGA;
  VAR      OGGI : GIORNO;
```

Inizializzazione:

```
  NUMGIORNO [LUN] := 'LUN';
  NUMGIORNO [MAR] := 'MAR';
      ..
      ..
  NUMGIORNO [DOM] := 'DOM';
```

Si potra' scrivere:

```
  WRITELN (NUMGIORNO [OGGI]);
```

5.13. TYPE RECORD

Per mezzo di una tabella si possono raggruppare N elementi dello stesso tipo, per esempio, l'ammontare del fatturato negli ultimi 10 anni o i nomi di tutti i clienti. Spesso e' pero' necessario raggruppare elementi di tipo diverso.

Questo e' normalmente il caso per le registrazioni su file.
Per esempio, sara' necessario avere per ogni cliente:

- il numero identificativo
- il nome
- l'indirizzo
- il codice postale
- il nome del rappresentante
- se gli e' accordato uno sconto
- il volume d'affari nell'anno precedente
- il volume d'affari nell'anno in corso.

Il Pascal dispone del tipo RECORD per realizzare questo (RECORD vuole proprio dire "registrazione"). Per il caso ora visto la dichiarazione in Pascal sara' la seguente:

```
TYPE CLIENTE = RECORD
               NUMERO : INTEGER;
               NOME   : PACKED ARRAY [1..10] OF CHAR;
               INDIR  : PACKED ARRAY [1..30] OF CHAR;
               CPOST  : PACKED ARRAY [1..20] OF CHAR;
               NUMRAPR: INTEGER;
               SCONTO : BOOLEAN;
               VOLPREC: REAL;
               VOLATT : REAL
               END;
```

Sara' allora possibile scrivere:

```
VAR CLI : CLIENTE;
```

e un elemento di CLI potra' essere ottenuto cosi':

```
CLI.NOME := 'ROSSI';
IF CLI.SCONTO THEN...
```

Questo puo' essere combinato con un indice:

```
TYPE IMPRESA = ARRAY [1..50] OF CLIENTI;
VAR I1 : IMPRESA;
```

A questo punto il volume d'affari del quinto cliente dell'impresa I1 e':

```
I1 [5].VOLATT
```

Il primo carattere del nome di questo cliente e':

```
I1 [5].NOME [1]
```

ESERCIZIO 5.11. - Una linea di un buono d'ordine comprende:

```
Articolo           20 caratteri
Prezzo unitario    reale
Quantita'          intero
Ammontare          reale
```

Scrivete la dichiarazione di tipo necessaria.

5.14. ISTRUZIONE WITH

L'inizializzazione di un intero RECORD e' un po' pesante:

```
CLI.NUMERO := 1000;
CLI.NOME := 'ROSSI';
CLI.INDIRIZZO :='....';
CLI.CPOST := '....';
CLI.NUMRAPR := 10;
CLI.SCONTO := TRUE;
....
```

L'istruzione WITH consente di ripetere CLI (o un altro qualificatore); basta scrivere:

```
WITH CLI DO
     BEGIN
        NUMERO := 1000;
        NOME := 'ROSSI';
        ..
        ..
        VOLATT := ...
      END;
```

Questo e' utile soprattutto se si hanno parecchi livelli di RECORD e quindi piu' qualificatori:

```
IMPRESA.REPARTO.SERVIZIO.CLI.NOME :=....
```

Bastera' allora scrivere:

```
WITH IMPRESA, REPARTO, SERVIZIO, CLI DO
     BEGIN
        NOME :=....
     END;
```

5.15. REGISTRAZIONI CON VARIANTI

E' possibile che tutte le registrazioni appartenenti alla stessa serie non abbiano esattamente la stessa struttura. Prendiamo in esame, come esempio, una lista di impiegati.

Sara' senz'altro presente il loro nome e la loro data di nascita; poi un codice che specifichi la situazione familiare: "C" per celibe, "M" per coniugato, "D" per divorziato, "V" per vedovo. Nel caso "celibe" questo sara' sufficiente. Nel caso "coniugato", comparira' anche il nome del coniuge e la data del matrimonio. Anche nei casi "divorziato" o "vedovo" comparira' una data.

In Pascal si fara' una dichiarazione di questo tipo:

```
TYPE NNN  = PACKED ARRAY [1..10] OF CHAR;
     DATA = RECORD
               GG : 1..31;
               MM : 1..12;
               AA : 0..99
            END;
     IMPIEGATO = RECORD
                     COGNOME : NNN;
                     NOME    : NNN;
                     DATNA   : DATA;
                     CASE STAFAM : CHAR OF
                        'C' : ( );
                        'M' : (COGNCONIUGE : NNN;
                               NOMCONIUGE  : NNN;
                               DATAMATR    : DATA);
                        'D' , 'V' : (DATADV : DATA)
                     END;
VAR OCCUP : IMPIEGATO;
```

Notate come la parola CASE determina il campo di variabilita' di STAFAM ed il suo tipo. L'istruzione di utilizzo potra' essere del tipo:

```
READ (IMPIEGATO);
IF IMPIEGATO.STAFAM = 'M' THEN
        WRITELN ('DATA MATRIMONIO',IMPIEGATO.DATAMATR)
                           ELSE
        WRITELN ('NON CONIUGATO');
```

Notate che se l'impiegato non e' coniugato non si ha possibilita' di accesso a una DATAMATR. Si deve esaminare sempre il campo di variabilita' per sapere a quali campi e' possibile accedere.

ESERCIZIO 5.12. - Come si dovrebbero modificare le istruzioni appena viste se si fosse in presenza di un WITH IMPIEGATO DO?

ESERCIZIO 5.13. - Selezionate solo sul mese del matrimonio ed apportate le conseguenti modifiche.

Disponiamo quindi di una struttura di dati molto potente che puo' essere trattata da un programma e ne vedremo l'utilizzo a proposito dei file, dei quali questa puo' essere considerata il complemento ideale.

Vediamo ora un'altra struttura di dati che ci permette di servirci della teoria degli insiemi.

5.16. TYPE SET

La nozione di insieme, fondamentale in matematica, si comprende bene anche solo intuitivamente. Un insieme e' molto semplicemente la collezione dei suoi elementi.

Se sono dati gli elementi: A, B, C; a partitre da essi si possono formare i seguenti insiemi:

```
[ ]    (insieme vuoto)
[A] [B] [C]
[A,B] [A,C] [B,C]
[A,B,C]
```

Ebbene, se in Pascal A, B, C formano un tipo:

```
TYPE ELEMENT = (A,B,C);
```

gli 8 insiemi sopra elencati sono degli oggetti del tipo:

```
TYPE INS = SET OF ELEMENT;
```

Si vede dunque che in Pascal e' possibile creare degli insiemi se si ha un gruppo di oggetti appartenenti allo stesso tipo. Questi insiemi formano un tipo SET OF, che

raggruppa tutti gli insiemi che e' possibile formare con gli elementi del tipo base.

Osservate bene le modalita' di scrittura che seguono:

- Lista per un tipo, uso parentesi rotonde:
  TYPE COLORE = (BLEU, BIANCO, ROSSO)

- Lista elementi di un insieme, uso parentesi quadre:
  STOFFE := [BLEU, BIANCO, ROSSO]

- Intervallo per un tipo, senza parentesi:
  TYPE INTEGER = 1..5

- Intervallo per un insieme, uso parentesi quadre:
  [1..5]   e' la stessa cosa di [1, 2, 3, 4]

ESERCIZIO 5.14. - Scrivete le dichiarazioni compatibili con l'assegnazione:

```
STOFFE := [BLEU, BIANCO, ROSSO]
```

La definizione degli insiemi, che non deve essere confusa con quella dei tipi, e' formalmente piu' semplice. In questo caso e' permessa la mescolanza degli intervalli e delle liste:

```
V := [1..5, 7, 9..11]
```

Vi ricorderete delle difficolta' incontrate con:

```
TYPE GIORNOLAVORO = (LUN,MAR,MER,GIO,VEN,SAB);
     WEKEND = ( SAB,DOM)
```

dato che SAB appartiene a due tipi. Esse possono essere risolte con:

```
TYPE GIORNO = (LUN,MAR,MER,GIO,VEN,SAB,DOM);
     PERIODO = SET OF GIORNO;
VAR LAVORATIVO,WEKEND : PERIODO;
LAVORATIVO := [LUN..SAM]
WEEKEND := [SAB,DOM]
```

Questo e' dunque piu' semplice, ma permangono delle limitazioni. Esse sono dovute al problema della "cardinalita'". Di cosa si tratta?

La cardinalita' di un tipo e' il numero totale di dati che gli appartengono. Per esempio, la cardinalita' del tipo 1..5 e' 5; infatti ci sono 5 elementi: 1, 2, 3, 4, 5.
Se la cardinalita' di un tipo e' N, la cardinalita' del

SET OF di quel tipo e' 2 elevato a N.

In effetti un elemento del tipo SET e' un insieme di elementi del tipo base. In conseguenza, per rappresentarlo, bisogna dire per ciascuno degli N elementi del tipo base, se e' presente o non e' presente nell'insieme.
Quindi, servono N bit per codificare un insieme, e questo spiega perche' non e' possibile formare un insieme a partire da un tipo base che abbia una cardinalita' troppo alta.

Nella maggior parte delle implementazioni del Pascal si hanno dei limiti che vanno da 60 a 256.

Cosi'non puo' esistere il tipo SET OF INTEGER, mentre e' possibile costruire un tipo SET OF tipi intervallo. Analogamente il tipo base non puo' essere strutturato. Per contro si possono creare dei FILE OF SET OF (vedi Cap. seguente), come delle tabelle di insiemi, e questo permette di superare la limitazione dovuta alla cardinalita'.

5.17. OPERAZIONI SUGLI INSIEMI

In Pascal sono definite le operazioni abituali nella teoria degli insiemi. Esse sono definite solo entro insiemi dello stesso tipo, cioe' SET OF lo stesso tipo base.

Abbiamo gia' visto l'operazione di ASSEGNAZIONE:

```
TYPE BASE = (B1,B2,...,BN);
     INS  = SET OF BASE;
VAR E1,E2,E3 : INS;
    A1,A2,A3 : BASE;
E1 := [B1,B2];
E1 := E3;
E2 := [A1];          (sono assegnazioni corrette)
```

RIUNIONE: Si scrive, per esempio: E3 := E1+E2;
essa crea un insieme formato da tutti gli elementi che appartengono sia a E1 che a E2.

E1 E2

ESERCIZIO 5.15. - Si ha:

```
E1 := [B1,B2];
E2 := [B3,B4];
E3 := E1+E2
```

Quanto vale E3?

ESERCIZIO 5.16. - Stessa domanda se: E1 := [B2,B3];

INTERSEZIONE: Si scrive in Pascal: E3 := E1 * E2.
Essa produce l'insieme formato dagli elementi comuni ai due insiemi di partenza. Cosi', con i dati dell'esercizio 5.16., E1 * E2 vale [B3]

COMPLEMENTO: E3 := E1-E2 e' l'insieme formato dagli elementi di E1 che non appartengono a E2. Cosi', con i dati dell'esercizio 5.16., E1-E2 vale [B1]

OPERAZIONI DI RELAZIONE

Queste operazioni danno un risultato booleano. Anche queste sono definite solo entro insiemi dello stesso tipo.

. Uguaglianza: E1 = E2 risulta TRUE se i due insiemi sono identici, cioe' contengono esattamente gli stessi elementi.

. Disuguaglianza: E1 <> E2 risulta TRUE se E1 ed E2 hanno almeno un elemento non comune.

. Inclusione: E1 <= E2 oppure E2 >= E1 risulta TRUE se E2 contiene almeno tutti gli elementi di E1.

APPARTENENZA

Questa relazione viene definita tra un elemento del tipo base ed un insieme. Il risultato e' booleano. L'operatore di appartenenza IN e' una parola riservata.

A1 IN E2 risulta TRUE se l'elemento A1 appartiene ad E2.

Questo e' molto comodo nelle applicazioni. Per esempio, se si dichiara:

```
   VAR X : CHAR;
IF X IN ['A','E','I','O','U'] controlla se X e' una vocale
```

in modo molto piu' semplice che non con l'eserc. 4.8..

ESERCIZIO 5.17. - Tradurre in Pascal l'inclusione stretta:

$E1 \subset E2$.

Risulta possibile la lettura di un insieme per mezzo di letture successive dei suoi elementi (se essi sono di un tipo standard) e della loro riunione successiva partendo dall'insieme vuoto.

```
TYPE CARATTERI = SET OF CHAR;
VAR CAR : CARATTERI;
     C   : CHAR;
BEGIN
   CAR := [ ];          *inizializzazione insieme vuoto*
   REPEAT
      READ (C);
      CAR := CAR + [C] *ricongiunzione caratteri letti*
   UNTIL EOF
END.
```

La stampa dell'insieme CAR si fara' nel modo seguente, se si suppone sia formato solo da lettere:

```
TYPE CARATTERI = SET OF 'A'..'Z';
VAR CAR : CARATTERI;
     C   : 'A'..'Z';
BEGIN
   FOR C := 'A' TO 'Z' DO
        BEGIN
           IF C IN CAR THEN WRITE (C)
END.
```

La maggior parte dei compilatori Pascal offre la funzione CARD (X) che da' come risultato il numero degli elementi dell'insieme.

Trattiamo ora un esempio piu' completo e mostriamo come diventa semplice usando gli insiemi. Purtroppo le limitazioni esistenti non consentono di trattare allo stesso modo altri casi interessanti. Saremo allora obbligati a sostituire gli insiemi con le tabelle, e questo e' il trattamento classico.

CRIVELLO DI ERATOSTENE

Il metodo del crivello di Eratostene permette di trovare tutti i numeri primi inferiori a N molto piu' rapidamente che non il metodo delle divisioni successive, gia' precedentemente da noi utilizzato. Il metodo e' molto semplice. Si scrivono tutti i numeri da 2 a N (si sa che 1 e' primo).

In seguito, ad ogni passo, viene estratto il primo numero trovato; esso viene dichiarato primo e vengono eliminati tutti i suoi multipli. Si continua fino ad esaurimento dei numeri, come segue:

```
   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
2  3   5   7   9  11  13  15
3  5   7  11  13
5  7  11  13
```

ora sono tutti primi.

```
PROGRAM CRIVELLO;
     CONST N=100;
     TYPE INS = SET OF 2..N;
     VAR CRI : INS;
         LL,RESTO : INTEGER;
         K,L : 2..N;
BEGIN
     CRI := [ ]
     FOR K := 2 TO N DO
         CRI := CRI + [K];  *creazione crivello*
     RESTO := N-1;
     WHILE RESTO <> 0 DO
        BEGIN
          K := 2;  *ricerca minor elemento crivello*
          WHILE NOT (K IN CRI) DO
                K := K + 1;
          WRITELN (K, 'NUMERO PRIMO');
          LL := K; *eliminazione K e suoi multipli*
          WHILE LL <=N DO
             BEGIN
               L := LL+
               IF L IN CRI THEN
                        BEGIN
                           CRI := CRI - [L];
                           RESTO := RESTO - 1
                        END;
               LL := LL + K
              END
         END
END.
```

Si vede la comodita' degli insiemi anche se l'obbligo di trattare L ed LL per ragioni di conformita' di tipo non e' molto soddisfacente.

D'altra parte noi rischiamo di trovare un limite anche al disotto di N=100 mentre e' al di sopra che il metodo del crivello diventa preferibile a quello delle divisioni

successive.

Per superare le limitazioni introduciamo il metodo del crivello sotto forma di tabella booleana: TRUE = presente, FALSE = eliminato. L'architettura del programma rimane quasi la stessa e questo sta a dimostrare che si puo' lavorare bene con le strutture classiche.

```
PROGRAM CRIVELLOSEC;
    CONST N = 10000         *si puo' andare anche oltre*
    TYPE CR = PACKED ARRAY [2..N] OF BOOLEAN;
    VAR CRI : CR;
        RESTO : INTEGER;
        K : 2..N;
        L : INTEGER;
BEGIN
    FOR K := 2 TO N DO CRI [K] := TRUE;
    *creazione crivello, inizialmente presenti
     tutti gli elementi*
    RESTO := N-1;
    WHILE RESTO <> 0 DO
       BEGIN
         K := 2;
         *ricerca elemento minore*
         WHILE NOT CRI [K] DO K := K + 1;
         WRITELN (K, 'NUMERO PRIMO');
         L := K;
         *eliminazione di K e suoi multipli*
         WHILE L <= N DO
            BEGIN
               IF CRI [L] THEN
                   BEGIN
                     CRI [L] := FALSE;
                     RESTO := RESTO - 1
                   END;
               L := L + K;
            END
       END
END.
```

Ora abbiamo manipolato i principali tipi di dati offerti dal Pascal, insistendo soprattutto sui tipi strutturati, dato che sono i piu' importanti nelle applicazioni.

Ripetiamo ancora una volta che le tabelle permettono di fare tutto, ma che, quando si puo' usarli, i tipi speciali del Pascal permettono applicazioni interessanti.

Ci restano da esaminare altri due tipi, ancora piu' speciali:

. Il tipo FILE, dato che il Pascal forma un tipo standard con i file.

. Il tipo POINTER che permette di gestire la memoria in modo dinamico e di organizzare delle strutture di liste.

Prima pero' vediamo come possiamo rendere modulari i nostri programmi con l'aiuto delle PROCEDURE e delle FUNCTION. Questo e', non dimentichiamolo, uno dei principali imperativi della programmazione strutturata.

CAPITOLO 6

PROCEDURE E FUNZIONI

6.1. PREMESSA

Il Pascal ci offre la possibilita' di rendere i programmi modulari, cioe' di dividerli in entita' piu' semplici, ben individualizzate, il cui scopo sia facilmente definibile e comprensibile. Questo facilita enormemente la stesura e la comprensione dei programmi.

Per illustrare questa possibilita' del linguaggio facciamo riferimento, per esempio, agli esercizi 1.5. e 2.1..

In seguito a una condizione IF...THEN si deve provvedere a scambiare tra loro due elementi appartenenti alla tabella in esame. Noi potremmo scrivere le istruzioni per lo scambio a quel punto del programma, ma questo:

. ci farebbe perdere il filo del ragionamento;
. renderebbe il programma meno chiaro.

E' senz'altro preferibile prendere nota nel modo piu' sintetico possibile che vogliamo procedere allo scambio degli elementi, ma rimandare a dopo la stesura delle istruzioni necessarie per lo scambio vero e proprio, e proseguire nel ragionamento precisando cosa si deve fare quando non necessita lo scambio.

Questo presenta dei vantaggi:

. Non ci siamo persi nei dettagli della scrittura del procedimento di scambio , ma al contrario, scrivendo solo la chiamata della procedura SCAMBIO, abbiamo potuto dedicarci all'ossatura generale del programma.

. La programmazione risulta piu' chiara e piu' leggibile, grazie alla scelta giudiziosa del nome della procedura chiamata, il cui ruolo puo' anche essere precisato da opportuni commenti.

Piu' in generale, Pascal ci permette di suddividere qualunque elaborazione in procedure dotate ciascuna di un nome. Una procedura viene richiamata semplicemente citando il suo nome (non si ha la parola chiave CALL come in Fortran).

Il programma principale risulta cosi' molto leggibile; esso e' formato da una sequenza di chiamate alle diverse procedure, con eventualmente qualche controllo (test), ma il concatenamento delle diverse tappe dell'elaborazione rimane mdlto visibile.

Una procedura puo' essa stessa definire e chiamare altre procedure e cosi' di seguito, cosa che costituisce, in certo modo, una sorta di cicli nidificati.

Questo facilita il modo di concepire i programmi, modo chiamato di analisi discendente (top down). Le grandi tappe dell'elaborazione (primo livello) sono definite per prime, poi ciascuna di queste tappe viene affinata (secondo livello), poi ciacuna delle procedure del secondo livello viene affinata, ecc... Questo metodo degli affinamenti successivi e' molto comodo.

Le procedure sono invece compatibili con il metodo contrario, detto dell'analisi ascendente (down top). La definizione delle procedure appare allora come l'aggiunta di nuove istruzioni al linguaggio.

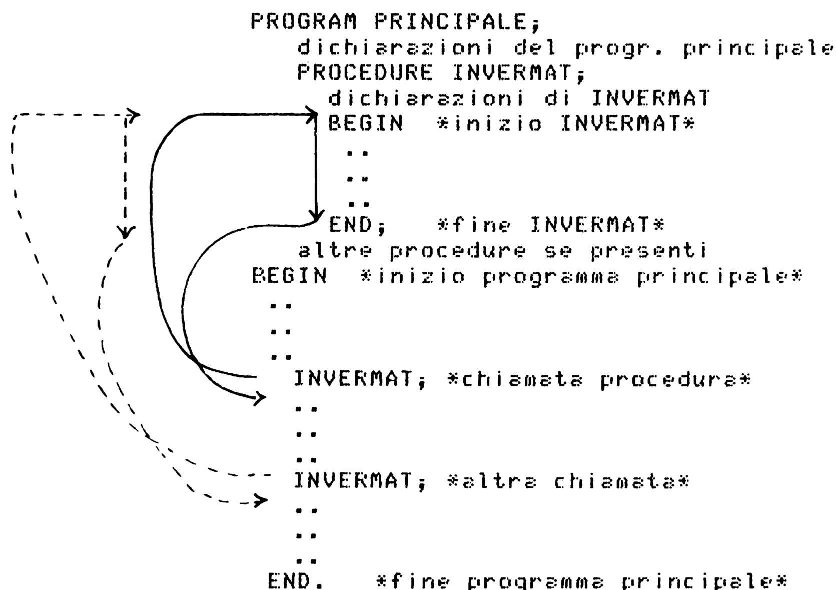
Per esempio, quando e' necessaria una procedura di scambio, la si scrive. Si costruisce cosi' uno stock di procedure che formano un livello; il programma al livello superiore consistera' in una serie di chiamate a queste procedure, come se esse fossero delle istruzioni del linguaggio.

Al livello piu' basso sono utilizzate le istruzioni vere e proprie del linguaggio.
Notiamo che noi utilizziamo come istruzioni del Pascal alcune strutture che sono, di fatto, delle procedure o delle funzioni standard, come SIN o WRITE.

Qual'e' il metodo migliore? L'analisi discendente o ascendente? Noi pensiamo, e l'abbiamo sempre fatto, che il meglio sia adottare un metodo misto; noi partiamo dall'alto, ma nello stesso tempo, noi scriviamo (o almeno definiamo) tutte le procedure elementari delle quali possiamo prevedere la necessita'. Quando le due progressioni si ricongiungono, il programma e' scritto.

Un altro vantaggio enorme delle procedure e delle funzioni e' che, molto spesso in una elaborazione, si hanno delle parti di trattamento utilizzate piu' volte. Per esempio, invertire una matrice in diversi punti del programma. Non si va cosi' ogni volta a riscrivere l'inversione della matrice. Si scrive una procedura e la si chiama ogni volta che serve.

Da cui la struttura seguente:



In un certo senso, le definizioni delle procedure fanno parte delle dichiarazioni del programma principale (sono poste alla fine delle dichiarazioni).

La procedura termina con la parola END (seguita da un punto e virgola) che corrisponde al RETURN del Fortran e del Basic. Ricordiamo che la END del programma principale termina con un punto.

La figura, di cui sopra, mostra che la chiamata della procedura corrisponde ad un salto all'inizio della stessa. La procedura viene percorsa fino alla parola END. A questo punto si sviluppa la parte piu' interessante del meccanismo; si ha un salto proprio subito dopo la chiamata che ha reso attiva la procedura. Detto in altro modo, il meccanismo "ricorda" da dove e' venuta la chiamata (tratteggio pieno o puntinato nella figura). Questo meccanismo e' classico, esiste in tutti i linguaggi di programmazione. Ci importa poco qui sapere come esso e' realizzato (vedere paragr. 8.2.), ma bisogna comprendere bene il suo funzionamento e sapere che esiste.

L'economia che ne consegue e' immensa; lo spazio di memoria per la procedura viene occupato una sola volta, lo sforzo per scriverla viene fatto una sola volta, senza contare che puo' essere utilizzata una procedura gia'

scritta.

Resta un punto da precisare nell'esempio visto. Come indicare alla procedura INVERMAT quale matrice essa deve invertire? Ritorneremo su questo problema.

Come il Fortran tra gli altri, il Pascal offre due tipi di moduli: le PROCEDURE e le FUNCTION.

6.2. FUNCTION

Le FUNCTION sono semplicemente una generalizzazione delle funzioni standard del Pascal; l'utilizzatore puo' egli stesso definire le FUNCTION che gli servono.

La chiamata di una funzione si fa nominandola all'interno di una espressione aritmetica, come per le funzioni standard.

Supponiamo di aver definito una funzione tangente TG(X), si potra' avere:

```
Z :=2.0*Y-W*TG(3*X+5)
```

In questo caso l'esecuzione di TG produrra' un salto alle istruzioni di calcolo della funzione. Prima del ritorno la funzione avra' ricevuto un valore, questo valore viene sostituito a TG nel calcolo e esso prosegue.

Questo fa supporre che la funzione sia stata scritta dopo le dichiarazioni iniziali. Per il nostro esempio della tangente calcolata a partire dalle funzioni standard SIN e COS, si dovra' scrivere:

```
FUNCTION TG(X:REAL):REAL;
    BEGIN
    TG := SIN(X)/COS(X)
    END;
```

La prima linea viene chiamata la testata della funzione. Il secondo REAL della linea definisce il tipo della funzione, cioe' il tipo del valore che verra' usato in tutte le espressioni che chiamano la funzione.

Questo tipo e' obbligatoriamente scalare, quindi una funzione puo' fornire un solo risultato scalare. Tra parentesi figura quella che viene chiamata la lista degli argomenti della funzione. Qui ne abbiamo uno solo: X. Viene dichiarato anche il suo tipo: REAL. L'argomento gioca il

ruolo di parametro formale per il calcolo della funzione.

La X all'interno della funzione non e' la stessa che compare nel programma chiamante. Prima di entrare nella funzione, il Pascal calcola 3*X+5 e trova il suo valore V. Questo valore viene copiato nella X della funzione ed ha luogo il calcolo della stessa.

Si dice che X e' locale alla funzione (ci ritorneremo sopra) e che l'argomento e' un argomento fornito alla funzione.

Prima della END che segna la fine della funzione si deve avere una assegnazione di valore all'identificatore TG; questo e' il valore della funzione.

ESERCIZIO 6.1. - Scrivere una funzione che calcoli Y elevato a X, dato che il Pascal non possiede l'operatore per l'elevamento a potenza. Si suppongano tutti i tipi REAL.

6.3. PROCEDURE

Le FUNCTION sono un po' limitate, per questo esistono le PROCEDURE che hanno un campo di validita' maggiore.

Se una FUNCTION restituisce un "valore", una PROCEDURE produce degli "effetti"; calcolo di parecchi risultati, modifica di parecchie variabili, o anche effetti fisici sull'ambiente in un programma di controllo di processo.

Per esempio, scriviamo una procedura il cui effetto e' di tracciare una linea formata dalla ripetizione del carattere C, N volte.

```
PROCEDURE TRACCIA (C:CHAR;N:INTEGER);
    CONST MAX = 100;
  BEGIN
     IF N > MAX THEN N := MAX;
     FOR I := 1 TO N DO
         WRITE (C);
     WRITELN
  END;
```

ESERCIZIO 6.2. - Questa procedura puo' sevire a sottolineare dei risultati stampati. Essa puo' servire anche per tracciare un istogramma. Si ha una tabella RISULTATI [1..10], dove ciascun elemento e' il risultato di una classe

(compreso tra 1 e 100). Tracciate l'istogramma.

Ecco qui un punto fondamentale. Supponiamo che la procedura TRACCIA sia chiamata con un argomento maggiore di 100.

La procedura TRACCIA modifica il suo argomento in questo caso; essa lo riporta a 100 per non rischiare di fare un tratteggio piu' lungo della linea di stampa; proviamo:

```
N :=120               Otterremo:
WRITELN (N);          120
TRACCIA ('*',N)       *** 100 volte in tutto ***
WRITELN (N);          120
```

La N e' stata modificata nella procedura, ma non nel programma chiamante!

Questo e', in effetti, un vantaggio del Pascal sul Fortran, per esempio. Gli argomenti forniti sono protetti, mentre in Fortran, non sono protette neppure le costanti.

```
        FORTRAN                            PASCAL

SUBROUTINE TOTO (K)             PROGRAM ZOZO
K = 2                              VAR I := INTEGER;
RETURN                             PROCEDURE TOTO (K:INTEGER);
END                                BEGIN
...                                K := 2
I = 1                              END;
WRITE (6,10) I                  BEGIN
CALL TOTO (I)                      I := 1;
I = 1                              WRITELN (I);
WRITE (6,10) I                     TOTO (I);
...                                WRITELN (I);
                                   ...

stampa     1                       stampa     1
           2                                  1
```

Questa trasformazione, nociva, della costante dovuta alla confusione che fa il Fortran tra gli argomenti forniti e quelli ritornati viene chiamata "effetto collaterale".

Bene, ma io vorrei anche avere degli argomenti di ritorno in Pascal. Per esempio, vorrei scrivere una procedura che legga una matrice A e vorrei poter usare la matrice nel programma chiamante.

```
LEGGOMAT(A);
Z := A[I,J]...
```

Pascal lo permette, naturalmente. Ma esso distingue gli argomenti di ritorno accompagnandoli con la dichiarazione VAR nella lista degli argomenti. Cosi' si scrivera':

```
PROCEDURE LEGGOMAT (VAR A:ARRAY [1..10,1..10] OF REAL);
   VAR I,J : INTEGER;
BEGIN
   FOR I := 1 TO 10 DO
        BEGIN
           FOR J := 1 TO 10 DO
               READ (A [I,J]);
           READLN
        END
 END;
```

ESERCIZIO 6.3. - Cosa succederebbe se uno avesse scritto:

```
   PROCEDURE TRACCIA (C:CHAR,VAR N:INTEGER);
e  PROCEDURE TOTO (VAR K:INTEGER); ?
```

Noi pensiamo che questo meccanismo e' ora abbastanza chiaro. E' sottinteso che un argomento VAR deve apparire come variabile nell'istruzione di chiamata, infatti, al ritorno esso deve contenere un valore eventualmente variato, cosa che non si puo' ottenere con una costante o una espressione aritmetica. Esempio:

PROCEDURE TOTO (VAR K : type); non puo' essere chiamata con:

```
TOTO (3*X+1);
```

Per contro, il nome della variabile non ha alcuna ragione di essere lo stesso nel programma chiamante e nella procedura. Esempio:

```
VAR X:type;
PROCEDURE TOTO (VAR Y:type);
TOTO (X);
```

Al momento della chiamata si avra' copia dell'X chiamante nello Y chiamato; al momento del ritorno si avra' copia dell'Y chiamato nell'X chiamante.

La seconda copiatura non ha luogo se l'argomento non ha l'attributo VAR. Quello che e' assolutamente obbligatorio e' che X e Y siano dello stesso tipo (altrimenti le copie sono impossibili).

Un nome di file puo' essere l'argomento di una procedura, ma esso deve essere di genere VAR.

6.4. VARIABILI LOCALI E GLOBALI

Noi potremmo scrivere diversamente il nostro programma per la lettura della matrice A:

```
 PROGRAM MATRICE;
     VAR A:ARRAY [1..10,1..10] OF REAL;
         I,J INTEGER;
     PROCEDURE LEGGOMAT;
         VAR I,J : INTEGER
     (come sopra)    READ (A[I,J]);
     END;
 BEGIN
     LEGGOMAT;
         A [I,J] ....(utilizzo)
```

uesto significa che una variabile (qui A), conosciuta nel programma chiamante e' conosciuta anche in tutte le procedure e le funzioni chiamate.

Si dice che questa variabile e' "globale".

E' esattamente cosi' che le cose si svolgono in Basic, e in tale linguaggio non si ha altro modo di trasmettere delle informazioni tra il programma chiamante ed i sottoprogrammi, dal momento che non sono disponibili gli argomenti.

Questo costituisce un considerevole svantaggio del Basic. In effetti:

1. La trasmissione delle variabili globali e' talvolta meno comoda della trasmissione degli argomenti.
Cosi', la nostra seconda versione della procedura per la lettura della matrice non puo' leggere che la matrice A. Se vogliamo leggere una matrice B, con la prima versione, la chiamata si scrive: LEGGOMAT (B); mentre con la seconda versione sarebbe necessario un trasferimento: B:=A, dopo la chiamata di LEGGOMAT. Inoltre e' possibile avere una matrice A da conservare dopo che la procedura l'avra' modificata.

2. Le variabili globali sono qualche volta nocive. Torniamo alla nostra procedura TRACCIA. Essa utilizza una variabile I come indice di ciclo. Nell'esercizio 6.2. noi abbiamo fortunatamente introdotto una variabile supplementare K da usare come indice nel ciclo dove

TRACCIA viene chiamata:

```
FOR K:=1 TO 10 DO
      TRACCIA ('*',RISULTATO [K]);
```

Se noi non l'avessimo fatto, cioe' se avessimo utilizzato la stessa vatiabile I:

```
FOR I:=1 TO 10 DO
      TRACCIA ('*',RISULTATO [I]);
```

sarebbe stata una catastrofe. In effetti la variabile I e' globale. Quindi essa viene modificata ad ogni chiamata di TRACCIA, e il ciclo del programma principale viene completamente rovinato nell'esempio appena visto.

Il rimedio viene dalle variabili locali. E' possibile introdurre in una procedura una variabile di lavoro, come l'indice di un ciclo, che sara' riconosciuta solo all'interno della procedura. Questa variabile viene chiamata locale.

Le variabili locali favoriscono di molto la modularita' dei programmi; ogni procedura ha le sue variabili locali, e, durante la stesura di una procedura non e' necessario pensare alle altre procedure, cosa che e' proprio lo scopo della modularita'.

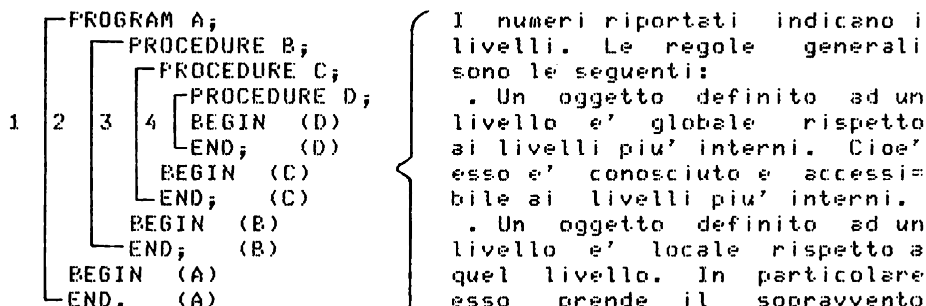
Perche' una variabile sia considerata come locale, basta dichiararla in testa alla procedura. Una variabile non dichiarata all'inizio della procedura e' considerata globale, cioe' si accede alla variabile cosi' come essa e' conosciuta nel programma. Una variabile dichiarata in una procedura risulta locale per quella procedura; essa viene distinta da una variabile avente lo stesso nome e conosciuta nel programma principale. Noi ne abbiamo gia' visto degli esempi. Cosi' nel programma della matrice le variabili I e J sono dichiarate. La procedura LEGGOMAT dichiara anche le variabili I e J ed esse vengono distinte dalle loro omonime. In tale modo la chiamata di LEGGOMAT non rischia di creare scompiglio tra gli indici del programma chiamante.

ESERCIZIO 6.4. - Riscrivete il programma dell'esercizio 6.2. in modo che le variabili che devono esserlo siano locali.

6.5. PORTATA DEGLI IDENTIFICATORI

La nozione di portata gioca un suo ruolo su tutti gli oggetti, tipi, costanti, procedure, e non soltanto sulle variabili, anche se nel caso delle variabili l'importanza e' maggiore.

Si possono "inscatolare" a volonta' le definizioni delle procedure. Ogni volta si definisce un livello piu' profondo. Esempio:



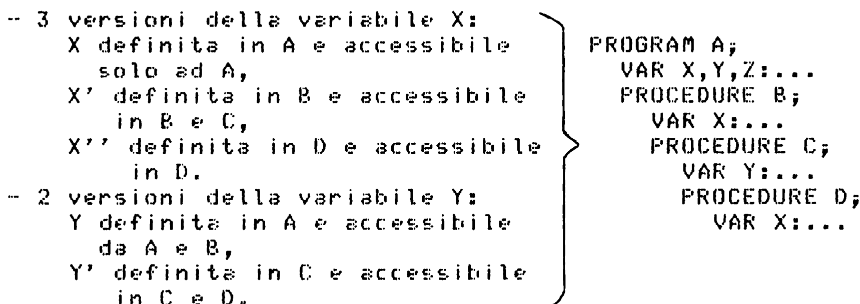
```
  ┌PROGRAM A;
  │  ┌──PROCEDURE B;
  │  │  ┌PROCEDURE C;
  │  │  │  ┌PROCEDURE D;
1 │2 │3 │4 │BEGIN  (D)
  │  │  │  └END;   (D)
  │  │  │ BEGIN  (C)
  │  │  └END;   (C)
  │  │   BEGIN  (B)
  │  └───END;   (B)
  │ BEGIN  (A)
  └END.   (A)
```

I numeri riportati indicano i livelli. Le regole generali sono le seguenti:

. Un oggetto definito ad un livello e' globale rispetto ai livelli piu' interni. Cioe' esso e' conosciuto e accessi= bile ai livelli piu' interni.

. Un oggetto definito ad un livello e' locale rispetto a quel livello. In particolare esso prende il sopravvento sugli oggetti aventi lo stesso nome e definiti a livelli superiori, questo vale per il livello di definizione e per quelli piu' interni fino a quando venga definito nuovamente un oggetto avente lo stesso nome e che prenda a sua volta il sopravvento.

Cosi' con le definizione date di fianco si ha:



- 3 versioni della variabile X:
  - X definita in A e accessibile solo ad A,
  - X' definita in B e accessibile in B e C,
  - X'' definita in D e accessibile in D.
- 2 versioni della variabile Y:
  - Y definita in A e accessibile da A e B,
  - Y' definita in C e accessibile in C e D.

```
PROGRAM A;
  VAR X,Y,Z:...
  PROCEDURE B;
    VAR X:...
    PROCEDURE C;
      VAR Y:...
      PROCEDURE D;
        VAR X:...
```

Per contro Z risulta globale per tutto il programma.

In C, per esempio, non si ha alcun modo di accedere alla X che e' stata definita in A; si puo' solo accedere alla versione di X definita in B.

Un oggetto non puo' essere raggiunto ad un dato livello se

non e' stato definito a quel livello, o ad un livello superiore, in una procedura in atto quando la procedura dove ci si trova e' attiva. Esempio:

```
PROGRAM A;
    VAR X;
    PROCEDURE B;
       VAR Y;
    END;
    PROCEDURE C;
```

E' possibile, in C, di accedere a X (livello superiore) ma non a Y perche' B, dove Y e' definito e' allo stesso livello ma esclu= sivo di C; non e' B che chiama C B non e' attivo quando C e' chia= mato.

Da questo punto di vista, gli argomenti di una procedura sono locali per quella procedura.

In una procedura si puo' accedere a 3 tipi di oggetti:

- gli oggetti locali,
- gli argomenti,
- gli oggetti globali (venuti dai livelli meno profondi).

L'accesso agli oggetti di un livello piu' interno e' impossibile. L'accesso agli oggetti di un livello piu' esterno e' ugualmente impossibile, se non c'e' una catena di chiamate inscatolate di procedure che colleghino questo livello al livello attuale.

Quest'ultimo punto deriva dal fatto che nel Pascal una procedura non esiste in memoria che quando essa e' attiva (in corso di esecuzione). Questo e' molto diverso da quello che succede in Fortran o in Basic. Per esempio:

```
PROGRAM TOTO;
   PROCEDURE A;
     VAR X:ARRAY [1..5000] OF REAL;
   ..
   ..
   END;
   PROCEDURE B;
     VAR X:ARRAY [1..5000] OF REAL;
   ..
   ..
   END;
   PROCEDURE C;
     VAR Y:ARRAY [1..5000] OF REAL;
   ..
   ..
   END;
 BEGIN
     A;
     ..
```

```
      ..
      B;
      ..
      ..
      C;
   END.
```

In nessun momento si potra' avere piu' di una procedura A, B e C attiva. Dunque non si occupera' mai piu' della parte di memoria necessaria per conservare 5000 elementi della tabella, mentre in Fortran sarebbe stato necessario memorizzarne 15000.

Il Pascal permette dunque di risolvere automaticamente i problemi di caricamento in memoria con salvataggio (overlay), e questo fa risparmiare molta memoria.

ESERCIZIO 6.5. - Scrivere un programma di moltiplicazione di matrici (confronta con il Cap. 5) suddiviso nelle procedure: lettura di matrici, calcolo del prodotto, stampa della matrice. In seguito, per ciascuno degli oggetti intervenuti, dire quale e' la sua portata, tenendo conto delle regole sulla portata degli identificatori.

6.6. ARGOMENTI DELLE FUNCTION E DELLE PROCEDURE

Una funzione o una procedura puo' essa stessa costituire un argomento per una procedura.

Per esempio, se noi vogliamo scrivere una procedura per il calcolo di un integrale, e' chiaro che la funzione da integrare e' un parametro della nostra procedura.

Il Pascal standard lo consente (attenzione: non e' il caso del Pascal UCSD).

L'intestazione della procedura si scrive:

```
PROCEDURE INTEG (FUNCTION F REAL; VAR S:REAL...)
```

e la funzione e' utilizzata in INTEG nella forma:

```
Y:=F(X)...
```

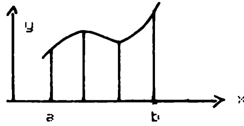
mentre la chiamata sara' nella forma INTEG (SIN,S,...) se si vuole integrare la funzione seno.

Noi potremo anche far riferimento ad una funzione definita da noi. Questo e' obbligatorio in alcune implementazioni del Pascal che vietano di usare come argomento una funzione

standard. La soluzione e' di definire una funzione non standard identica alla funzione standard voluta:

TOTO := SIN(X) e poi chiamare INTEG (TOTO,S,...).

Scriviamo questa procedura. Oltre alla funzione da integrare e al risultato S, gli argomenti saranno i limiti A e B dell'integrazione e il numero di intervalli elementari di integrazione (utilizziamo il metodo dei trapezi).



L'intervallo per ogni passo e':

h =(b-a)/n mentre per S vale:

$$S = h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+ih)\right)$$

da cui la procedura:

```
PROCEDURE INTEG (FUNCTION F:REAL; VAR S:REAL;
                 A,B:REAL; N:INTEGER);
   VAR X,H :REAL;   *H passo di integrazione*
  BEGIN
   H:=(B-A)/N;
   S:=(F(A)+F(B))/2;
   X:=A+H;
   REPEAT
     S:=S+F(X);
     X:=X+H;
   UNTIL X > B-H/2;
   S:=S*H
END;
```

La sola cosa da notare e' che, di norma, si sarebbe dovuto scrivere UNTIL X=B, ma per delle ragioni di precisione di calcolo e' molto difficile controllare l'uguaglianza tra numeri reali, da cui la nostra scrittura.

ESERCIZIO 6.6. - Scrivere un programma che provi questa procedura per calcolare:

$$\int_0^{\pi} SIN\ X\ DX \qquad \text{(valore esatto 2)}$$

controllando la precisione ottenuta per diversi valori di N: 5, 10, 15, 20, 25.

NOTA: Notate la forma della lista degli argomenti nella testata della procedura o della funzione. Si hanno degli elementi separati dal punto e virgola. Ciascun elemento puo'

essere un nome di PROCEDURE o di FUNCTION, un tipo o un nome VAR, un tipo o un nome, un tipo. Non si hanno virgole a meno che non si abbiano piu' nomi dello stesso tipo (;nome1,nome2:tipo;).

6.7. LA RECURSIVITA'

Una restrizione ben nota ai programmatori Fortran e' che un sottoprogramma non potra' in alcun caso richiamare se stesso!

Ebbene in Pascal questa restrizione non esiste. Una PROCEDURE o una FUNCTION puo' richiamare se' stessa. Questo si chiama "recursivita'". Facciamo notare che questo e' vero anche in alcune implementazioni del Basic, ma, dato che i sottoprogrammi Basic non possono gestire argomenti, gli usi possibili sono molto ridotti.

A cosa puo' servire questo? Ebbene, un certo numero di problemi si pongono in modo ricorsivo, cioe' la loro soluzione presuppone la soluzione di una versione piu' semplice dello stesso problema.

Per esempio, si sa che la definizione del fattoriale di N e':

. fattoriale 0=1
  fattoriale N=N*fattoriale (N-1)

In Pascal si puo' scrivere la funzione seguente per calcolare il fattoriale seguendo direttamente la sua definizione:

```
FUNCTION FATTORIALE (N:INTEGER) :INTEGER;
  BEGIN
      IF N=0 THEN FATTORIALE := 1
             ELSE FATTORIALE := N*FATTORIALE (N-1)
  END;
```

Supponiamo di chiamarla cosi': Z := FATTORIALE (3).

A ciascuna chiamata recursiva della funzione , una versione della procedura diventa attiva con una versione della variabile locale N e le diverse versioni dell'argomento N aumentano, per cosi' dire. Si avra' successivamente:

```
N=3  diverso da 0 e quindi chiamata per N=2              3

N=2     "       "  "  "   "        "     "  N=1          2
                                                         3

N=1     "       "  "  "   "        "     "  N=0          1
                                                         2
                                                         3

                              e nella pila si avra':     0
                                                         1
                                                         2
                                                         3
```

a questo punto, dato che N=0, comincia il ritorno:

```
N=0 e quindi FATTORIALE := 1                             1
                                                         2
                                                         3

N=1 "    "          "      := 1*1 = 1                    2
                                                         3

N=2 "    "          "      := 2*1 = 2                    3

N=3 "    "          "      := 3*2 = 6
```

da cui il risultato finale. Questo schema mostra chiaramente il meccanismo della pila: una pila e' una struttura di dati dove un nuovo dato puo' sempre essere aggiunto alla sommita', ed il solo dato prelevabile e' quello che si trova alla sommita'.

E' una pila la struttura che serve per memorizzare gli indirizzi di ritorno dei sottoprogrammi ed e' questo che permette le nidificazioni.

Il difetto del Fortran e' che esso gestisce una pila solo per gli indirizzi di ritorno dei sottoprogrammi, mentre il Pascal gestisce una pila per ogni argomento e variabile locale e questo consente la recursivita'.

Detto questo, facciamo notare che anche se non si dispone della recursivita' si puo' calcolare il fattoriale. In molti casi la definizione recursiva di un problema si puo' trasformare in definizione iterativa. Per il fattoriale basta scrivere:

```
FUNCTION FAC (N:INTEGER) :INTEGER;
    VAR I,F:INTEGER;
  BEGIN
    I:=0;F:=1;
    WHILE I < N DO
```

```
          BEGIN
             I:=I+1;
             F:=I*F
          END;
      FAC:=F
   END;
```

L'utilizzo della funzione iterativa e' piu' redditizio riguardo al tempo di esecuzione ed alla occupazione di memoria di quello della funzione recursiva. Dunque e' meglio, quando si puo', utilizzare una forma iterativa invece di una forma recursiva. Ma, in certi casi, il problema e' recursivo in modo cosi' naturale che e' impossibile non utilizzare la forma recursiva.

Esempio: Supponiamo di voler scrivere una procedura per interpretare le espressioni aritmetiche:

```
   PROCEDURE INTERP (ESPRESSIONI:STRING);
```

L'espressione da interpretare e' una sequenza di caratteri. Ma si sa che si devono calcolare per prima cosa tutte le parti di espressione racchiuse tra parentesi: INTERP ('XXXX(sotto-espressione)XXXX') fara' dunque appello a INTERP(sotto-espressione). In questo problema e' naturale la recursivita'. Molti programmi che fanno parte del sistema operativo sono di questo tipo. La recursivita' del Pascal e' uno degli elementi che ne fanno un linguaggio utilizzabile per scrivere routine di sistema.

6.8. LA TORRE DI HANOI

Trattiamo ora un esempio dove la forma recursiva si inserisce naturalmente e dove la forma iterativa non sarebbe facilmente applicabile. E' il problema della Torre di Hanoi. Si dispone di 3 paletti sui quali possono essere infilati dei dischi di diverso diametro. Alla partenza, N dischi di diametro decrescente sono infilati sul paletto 1.

Si devono passare i dischi sul paletto 2 aiutandosi con il paletto 3 e seguendo queste regole:

.1 - Si puo' togliere un disco per volta;
.2 - I dischi devono sempre essere infilati su un paletto, non possono essere appoggiati sul tavolo di fianco ai paletti;
.3 - I dischi infilati su un paletto devono sempre trovarsi in ordine di diametro descrescente. Non si puo' posare su un disco un altro di diametro superiore.

Ebbene, questo problema ha una soluzione recursiva semplice; infatti se chiamiamo SPOST (N,I,J) la procedura che sposta N dischi dal paletto I al paletto J:

- sappiamo effettuare lo spostamento per 1 disco: SPOST (1,I,J) consiste molto semplicemente nello stampare I => J

- supponiamo di aver risolto il problema per N-1, cioe' che noi sappiamo spostare rispettando le regole N-1 dischi superiori. Questo e' possibile perche' e' il disco di diametro maggiore ad essere "neutralizzato". I movimenti conformi alle regole per gli N-1 dischi piu' piccoli resteranno conformi alle regole per N. Dunque, se N <>1, SPOST (N,I,J) si scrive:
SPOST (N-1,I altro(I,J));
SPOST (1,I,J);
SPOST (N-1,altro(I,J),J);
ed e' tutto. Altro(I,J) si riferisce al terzo paletto se I e J sono i primi 2. Se i paletti sono numerati 1,2 e 3, allora: altro(I,J)= 6-I-J.
Il programma si scrive allora:

```
PROGRAM HANOI;
   VAR N : INTEGER;
   PROCEDURE SPOST (N,I,K:INTEGER);
      BEGIN
         IF N=1 THEN WRITELN (I,'=>',J)
                ELSE BEGIN
                        SPOST (N-1,I,6-I-J);
                        SPOST (1,I,J);
                        SPOST (N-1,6-i-j,j)
                     END
      END;
   BEGIN
      READ(N);
      SPOST(N,1,2)
   END.
```

Il risultato del programma sara' la lista dei movimenti successivi da fare:
1 => 3   1 => 2   3 => 2 ....

dove I => J si interpreta nella forma "prendere il disco piu' in alto (e anche il piu' piccolo) dal paletto I e porlo sopra al disco che si trova gia' nel paletto J".

Si vede la semplicita' che apporta la recursivita' in un problema a priori complicato; si puo' quasi fare a meno di ragionare, e' il meccanismo della pila che svolge il suo lavoro.

ESERCIZIO 6.7. - Bene, noi andiamo a lavorare un po' lo stesso. Dobbiamo stampare sulla stampante o sullo schermo gli stati successivi dei paletti. Il numero dei dischi e' limitato a 10. Alla partenza la situazione sara' questa, se ci sono 5 dischi:

```
linea 1          I                  I                  I
      2          I                  I                  I
      3          I                  I                  I
      4          I                  I                  I
      5          I                  I                  I
      6         =I=                 I                  I
      7        ==I==                I                  I
      8       ===I===               I                  I
      9      ====I====              I                  I
     10     =====I=====             I                  I

col.  1          11        22       32        43       53        63
```

Il disegno sara' contenuto entro una matrice di 10 righe e 63 colonne. I paletti saranno rappresentati da delle I e i dischi da simboli =. Per comodita' si terra' una matrice immagine ridotta a 10 righe e 3 colonne che apparira' inizialmente cosi' (a immagine del disegno di cui sopra):

```
   0   0   0       Queste due matrici saranno variabili
   0   0   0       globali:
   0   0   0     VAR DISEGNO:ARRAY[1..10,1..63] OF CHAR;
   0   0   0         RIDOTTA:ARRAY[1..10,1..3] OF INTEGER;
   0   0   0
   1   0   0         COLL    :ARRAY [1..3] OF INTEGER;
   2   0   0        Quest' ultimo vettore COLL contiene il
   3   0   0        numero delle colonne dove sono disegna=
   4   0   0        te le I dei paletti (11,32,53).
   5   0   0
```

Introduciamo ora le procedure necessarie:

INIT: che crea il disegno iniziale in due fasi:
- . inizializza le matrici RIDOTTA e DISEGNO;
- . traccia il disegno.

MOVIMENTO: che traccia lo stato finale di ogni movimento in cinque fasi:

- . ricerca il disco piu' alto sul paletto di partenza e annota le sue dimensioni;
- . toglie il disco dal paletto dove si trova;
- . ricerca il posto libero piu' basso sul paletto di arrivo;
- . mette a posto il disco sul paletto;
- . traccia il disegno.

Segue il programma ed un esempio per N=3. I risultati ottenuti qui sono quelli che derivano dall'aver applicato la modifica suggerita nell'esercizio 6.8..

ESERCIZIO 6.8. - Apportate il seguente perfezionamento: se N<10 non e' necessario stampare su 10 righe. I paletti non devono necessariamente essere piu' alti della torre iniziale.

```
PROGRAM HANOI;
 VAR N:INTEGER;

     DISEGNO:ARRAY[1..10,1..63]OF CHAR;
     RIDOTTA:ARRAY[1..10,1..3]OF INTEGER;
     COLL:ARRAY[1..3]OF INTEGER;
 PROCEDURE INIT(N:INTEGER);
   VAR I,J,K:INTEGER;
  BEGIN  * init *
   COLL[1]:=11;[COLL[2]:=32;COLL[3]:=53;
   FOR I:=1 TO 10 DO
    BEGIN
     FOR J:=1 TO 63 DO DISEGNO[I,J]:=' ';
     FOR J:=1 TO 3 DO RIDOTTA[I,J]:=0;
     FOR J:=1 TO 3 DO
      BEGIN
       K:=COLL[J];
       DISEGNO[I,J]='I';
      END
    END;
   FOR I:=10 DOWNTO 11-N DO
    BEGIN
     K:=N+I-10;
     RIDOTTA[I,1]:=K;
     FOR J:=COLL[1]-1 DOWNTO COLL[1]-K DO
                      DISEGNO[I,J]:='=';
     FOR J:=COLL[1]+1 TO COLL[1]+K DO
                      DISEGNO[I,J]:='=';
    END;
   WRITELN;WRITELN;
```

```
   FOR I:=1 TO 10 DO
    BEGIN
     FOR J:=1 TO 63 DO WRITE(DISEGNO[I,J]);
     WRITELN
    END;
   WRITELN;WRITELN
 END;  * init *
PROCEDURE TOGLIERE(N,I,J:INTEGER);
 PROCEDURE MOVIMENTO(DEP,ARR:INTEGER);
  VAR I,J:INTEGER;
      IA,ID,P:INTEGER;  * p nessun disco da togliere *
  BEGIN  * movimento *
       * trovare il posto e modificare RIDOTTA *
      ID:=1;
      WHILE RIDOTTA[ID,DEP]=0 DO ID:=ID+1;
      P:=RIDOTTA[ID,DEP];
      RIDOTTA[ID,DEP]:=0;
      IA:=10;
      WHILE RIDOTTA[IA,ARR]<>0 DO IA:=IA-1;
      RIDOTTA[IA,ARR]:=P;
       * togliere il disco *
      FOR J:=COLL[DEP]-1 DOWTO COLL[DEP]-P DO
          DISEGNO[ID,J]:=' ';
      FOR J:=COLL[DEP]+1 TO COLL[DEP]+P DO
          DISEGNO[ID,J]:=' ';
       * mettere il disco *
      FOR J:=COLL[ARR]-1 DOWNTO COLL[ARR]-P DO
          DISEGNO[IA,J]:='=';
      FOR J:=COLL[ARR]+1 TO COLL[ARR]+P DO
          DISEGNO[IA,J]:='=';
        * disegno *
      FOR I:=1 TO 10 DO
        BEGIN
         FOR J:=1 TO 63 DO
           WRITE (DISEGNO[I,J]);
           WRITELN
        END;
      WRITELN;WRITELN;
    END;  * movimento *
  BEGIN   * spostamento *
    IF N=1 THEN MOVIMENTO(I,J)
           ELSE
             BEGIN
               TOGLIERE(N-1,I,6-I-J);
               TOGLIERE(1,I,J);
               TOGLIERE(N-1,6-I-J,J)
              END
  END;   * togliere *
 BEGIN   * hanoi *
  READ(N);
  INIT(N);
  TOGLIERE(N,1,2)
```

```
END.

        =I=                I                  I
       ==I==               I                  I
      ===I===              I                  I


         I                 I                  I
       ==I==               I                  I
      ===I===             =I=                 I


         I                 I                  I
         I                 I                  I
      ===I===             =I=               ==I==


         I                 I                  I
         I                 I                 =I=
      ===I===              I                ==I==


         I                 I                  I
         I                 I                 =I=
         I              ===I===             ==I==


         I                 I                  I
         I                 I                  I
        =I=             ===I===             ==I==


         I                 I                  I
         I               ==I==                I
        =I=             ===I===               I


         I                =I=                 I
         I               ==I==                I
         I              ===I===               I
```

Ci restano da vedere ancora alcuni dettagli sulle procedure, ma questi, sfortunatamente, dipendono molto dall'implementazione del Pascal disponibile.

6.9. PROCEDURE FORWARD E PROCEDURE EXTERN

Diciamo PROCEDURE, ma tutto quello che segue e' valido anche per le funzioni.

Non e' molto gradevole dover mettere le procedure richiamate da una procedura all'inizio di quest'ultima, come appare dall'esempio che segue:

```
PROGRAM A;
  PROCEDURE B;
  END;
  PROCEDURE C;
    PROCEDURE D;
    END;
  END;
  PROCEDURE E;
  END;
BEGIN
END.
```

La procedura B puo' essere chiamata da A, C, D e E. La procedura D puo' essere chiamata solo da C. La procedura C puo' essere chiamata da A e da E. La procedura E puo' essere chiamata solo da A.

Sarebbe piu' naturale poter scrivere il riferimento quando serve, ma stendere le procedure piu' tardi.

Il Pascal lo permette: e' solo necessario porre la testata della procedura che sara' richiamata tra le dichiarazioni della procedura chiamante, facendola seguire dalla parola FORWARD. Esempio:

```
FUNCTION RANDOM (X:real) : REAL;FORWARD;
```

La scrittura della procedura chiamata, preceduta da una testata abbreviata, verra posta dopo la END della procedura chiamante.

```
END;
FUNCTION RANDOM;
   BEGIN
   ..
   ..
   END;
```

Questo modo di procedere assomiglia di piu' a quello che si fa abitualmente in Fortran. Ma il Fortran permette di richiamare procedure che non fanno materialmente parte del testo: i sottoprogrammi di biblioteca che vengono compilati

indipendentemente dal programma chiamante.

Il Pascal permette anche questo (a seconda delle implementazioni) sostituendo la parola FORWARD con la parola EXTERN. E' anche possibile preparare una biblioteca di programmi scritti in altri linguaggi, per esempio Fortran. In questi casi la parola FORTRAN sostituisce la parola EXTERN.

Lasciamo al lettore la cura di documentarsi con precisione sulle possibilita' offerte dall'implementazione del Pascal che usa, soprattutto riguardo alla possibilita' di utilizzare sottoprogrammi scritti in altri linguaggi.

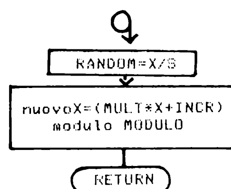
ESERCIZIO 6.9. - Non esiste in Pascal una funzione standard per ottenere numeri pseudo-casuali (come avviene invece in Basic). Noi compenseremo questa mancanza scrivendo la funzione RANDOM (X) che fornisce un numero reale pseudo-casuale compreso tra 0 e 1. La presenteremo come funzione FORWARD. Le chiamate successive saranno della forma:

```
X:=valore iniziale
.....RANDOM (X)......   (non ridefinire X per le chiamate
     .                   successive dato che e' RANDOM
     .                   che lo ridefinisce).
.....RANDOM (X).....
```

Si dimostra matematicamente che la sequenza definita' cosi':

```
RO = X (valore iniziale qualunque)
RN+1 = (moltiplicatore * RN + incremento) modulo (modulo)
```

e' una sequenza pseudo-random a condizione che i numeri (costanti) moltiplicatore, incremento e modulo, siano ben scelti (questo e' il metodo chiamato delle congruenze lineari).



Ad ogni chiamata di RANDOM si devono fare le operazioni riportate nello schema a fianco. S e' una costante che fissa la grandezza dei numeri da produrre. Il cal= colo risulta facilitato se si usa MODULO = 2 elevato a N, dove N e' il numero dei bit disponibile sul calco= latore per gli interi. In tale caso il modulo si ottiene automaticamente. Noi usiamo

il modulo 2 elevato a 16, che e' uguale a 65536. Si constata allora che MULT = 25173 e INCR =13849 danno buoni risultati. Si ottiene una sequenza di valori compresi tra 0 e 65535 e per avere dei valori compresi tra 0 e 1 si deve dividere per S = 65535.

ESERCIZIO 6.10. - Scrivere una procedura CURVA (F,A,B) che tracci il grafico della funzione F, considerando l'asse delle X nel senso di svolgimento della carta. A e B sono i limiti delle ascisse. Tracciare 60 punti rappresentativi, su 60 linee. Si supponga che F sia fornita modificandola con un fattore di scala conveniente affinche' risulti Fmin>1 e Fmax<100 (per esempio per la funzione seno, si avra' F(X)=50 + 49*SIN(X)). Ciascun punto si ottiene cosi': se Y = TRUNC(F(X)) per la X che corrisponde alla linea considerata, si tracciano su questa linea Y punti e poi un asterisco. Si prenda esempio dal grafico di SIN(X) tra 0 e 2PI(pigreco).

CAPITOLO 7

FILE SEQUENZIALI

7.1. PREMESSA

Vediamo ora un tipo strutturato molto particolare offerto dal Pascal; il tipo FILE.

Le operazioni consentite sugli oggetti del tipo FILE non sono le stesse rispetto a quelle consentite per gli altri tipi. Per esempio non si ha una assegnazione del tipo Y:=X (che dovrebbe rappresentare la copia del file X nel file Y) e neppure un confronto del tipo IF X=Y.

In effetti, le operazioni consentite sono esattamente quelle necessarie per gestire le entrate e le uscite sequenziali. In conseguenza il tipo FILE sara' associato, nella maggior parte dei casi, a queste operazioni di entrata e uscita. Inoltre il Pascal puo' consentire la costruzione di "file interni" che non corrispondono al tipo di operazioni prima citate.

Una grossa limitazione del sistema e' che esso puo' essere applicato solo ai file sequenziali. Nel Pascal standard non e' previsto di poter usare l'accesso diretto e quindi si e' privati della comodita' dell'uso dei dischi. Il Pascal UCSD e' stato esteso in quella direzione, senza introdurre tipi supplementari di dati, ma aggiungendo delle procedure standard che effettuano delle operazioni fisiche di entrata ed uscita con il metodo dell'accesso diretto.

Questo capitolo sara' abbastanza corto, dato che il soggetto e' gia' stato in parte svolto nel Cap. 3.

7.2. IL TIPO FILE

Supponiamo di avere le dichiarazioni che seguono:

```
VAR ARTICOLO : TYPEARTICOLO
    FILE1    : FILE OF TYPEARTICOLO
```

esse servono per creare un file sequenziale di nome FILE1 il quale e' una sequenza ordinata di oggetti del tipo TYPEARTICOLO. Dato che la sequenza e' finita, il primo

oggetto e l'ultimo sono determinati.

I file possono essere formati da oggetti di qualunque tipo, anche se raramente si formano FILE OF FILE!

In realta' i piu' comuni sono i FILE OF CHAR o dei FILE OF TYPEARTICOLO, dove TYPEARTICOLO e' un RECORD.
Questo e' normale, gli elementi di un file sono le registrazioni.

La cosa piu' straordinaria con il tipo FILE e' che, ad ogni istante, e' accessibile uno ed un solo elemento del file (in una tabella, per esempio, tutti gli elementi sono disponibili contemporaneamente).
La miglior immagine che si puo' avere di cio' si ha facendo riferimento alla situazione dell'ingresso o dell'uscita sequenziale dei dati. Quando dei dati registrati su un nastro magnetico vengono letti, ad ogni istante si ha un solo dato al di sopra del quale si trova posizionata la testina di lettura e scrittura. Esso e' il dato accessibile. Si parla di una finestra di accesso (che avanza per mezzo delle istruzioni PUT e GET).

L'elemento accessibile del file si chiama, nel nostro esempio, FILE1@. In alcune macchine il carattere che segue il nome invece di essere una "at" (@) e' un altro carattere speciale (come "freccia su" o altro).

FILE1@ ha il ruolo di identificatore di una zona di memoria capace di contenere una registrazione (RECORD).

Le operazioni sui file consistono in dei trasferimenti fisici tra la zona di memoria (generalmente chiamata buffer) e la periferica che supporta il file.

Per riempire il buffer prima di scriverlo (trasferirlo fuori), si puo' usare questa istruzione:

```
FILE1@ := ARTICOLO;
```

e per utilizzarlo dopo una lettura:

```
ARTICOLO := FILE1@;
```

7.3. SCRITTURA DEI FILE

La creazione di un file si ottiene con:

REWRITE (FILE1); che fa posizionare all'inizio del file,

seguito da una successione delle seguenti coppie di istruzioni:

```
FILE1@ := ARTICOLO;
PUT (FILE1);
```

che scrivono gli articoli successivi.

Queste due ultime istruzioni possono essere concentrate in una usando la PROCEDURE standard:

```
WRITE (FILE1,ARTICOLO);
```

la quale evita di gestire il buffer FILE1@. Il secondo argomento, che indica cosa deve essere registrato, deve necessariamente essere del tipo base del file.

7.4. LETTURA DEI FILE

Si deve per prima cosa chiamare la PROCEDURE standard:

```
RESET (FILE1);
```

Essa serve per riposizionarsi all'inizio del file e per trasferire il primo elemento nel buffer.
In seguito le letture si ottengono con la sequenza:

```
ARTICOLO := FILE1@;
GET (FILE1);
```

GET trasferisce l'elemento davanti al quale si trova la testina di lettura nel buffer. Il contenuto del buffer viene utilizzato nella successiva coppia di istruzioni ARTICLE:=FILE1@; GET.....

In altre parole, si e' sempre avanti di un GET, e precisamente del GET che e' implicito nella istruzione RESET. Procedendo cosi', si e' in grado di riconoscere la fine del file. In quel momento FILE1@ risulta indefinito e la funzione booleana EOF(FILE1) diventa TRUE.

Analogamente a quanto detto per la scrittura, la coppia ARTICLE :=...;GET...;puo' essere sostituita dalla chiamata della PROCEDURE standard:

```
READ(FILE1, ARTICOLO);    se ARTICOLO e' di tipo standard;
                          (vedi paragrafo 7.9.)
```

che posiziona automaticamente la variabile booleana

EOF(FILE1).

Notate che READ e WRITE possono riferirsi a piu' dati consecutivi. Esempio:

```
READ(FILE, ARTICOLO1, ARTICOLO2, ARTICOLO3);
```

ESERCIZIO 7.1. - FILE1 e' un file di numeri reali. Stampate il suo contenuto, ponendo un numero per riga.

In questo esercizio viene gia' posto il problema della dichiarazione di un file nella testata di un programma. Torneremo sull'argomento piu' avanti. Per il momento notiamo che e' possibile fare delle aggiunte ad un file sequenziale. Basta eseguire delle frasi WRITE dopo che risulta EOF vero. Per contro, non e' possibile fare degli aggiornamenti su un file sequenziale. Si deve procedere ad una ricopiatura.

ESERCIZIO 7.2. - Aggiungere il numero (pigreco) alla fine del file dell'esercizio 7.1..

7.5. DICHIARAZIONI PROGRAMMA

I file "interni", cioe' non associati a periferiche di ingresso/uscita, devono essere dichiarati come VAR.

I file "esterni", cioe' quelli associati a periferiche di ingresso/uscita, devono, naturalmente, essere dichiarati come VAR, ma essi devono anche essere citati nella frase PROGRAM all'inizio del programma:

```
PROGRAM GESTIONE (FILE1, FILE2);
```

Come si vede i nomi dei file hanno lo stesso ruolo degli argomenti delle procedure.

7.6. I FILE STANDARD

Il Pascal conosce due file standard: l'entrata standard del sistema chiamata INPUT e l'uscita standard chiamata OUTPUT.

Le componenti di questi due file possono presentare delle differenze tra un sistema ed un altro a seconda che il

sistema sia interattivo o meno.

Quando il nome del file non viene citato nelle frasi READ e WRITE sono sottintesi rispettivamente INPUT e OUTPUT:

```
READ (ARTICOLO);
WRITE (ELEMENTO);
```

In EOF e' sottinteso INPUT; IF EOF... equivale a IF EOF (INPUT)...
ma la forma completa puo' anche essere scritta:

```
READ (INPUT, ARTICOLO);
```

Le operazioni RESET(INPUT) e REWRITE(OUTPUT) non si devono mai scrivere esplicitamente.

Esse di norma sono eseguite automaticamente all'inizio di ogni programma e non devono mai essere eseguite in mezzo ad un programma; la stampante o il lettore di schede non devono essere riavvolti come un nastro! Consultate le appendici per certe particolarita' del CBM in questo campo.

La RESET(INPUT) e la REWRITE(OUTPUT) sono a volte eseguite automaticamente a condizione che INPUT e/o OUTPUT siano citati nella testata del programma:

```
PROGRAM TOTO (INPUT,OUTPUT,FILE1);
```

Questa citazione di INPUT e OUTPUT nella testata del programma e' a seconda dei sistemi, obbligatoria (qualche volta anche se i file standard non sono utilizzati), facoltativa o proibita (come in UCSD).
I file standard INPUT e OUTPUT sono in effetti dei file TEXT. Vedremo quindi altri particolari che li riguardano nel paragrafo 7.8.; prima, pero', trattiamo come esercizio due applicazioni gestionali.

7.7. APPLICAZIONI DI GESTIONE

I problemi gestionali sono il dominio prediletto dei file i cui elementi sono del tipo RECORD. Questo permette di dare alle registrazioni la struttura voluta.

ESERCIZIO 7.3. - Le singole registrazioni di un file del personale di una azienda hanno la seguente struttura:

numero impiegato :intero
cognome :stringa di 10 caratteri

```
nome                         :stringa di 10 caratteri
indirizzo                    :stringa di 30 caratteri
n.tessera assicurativa       :intero di 13 cifre (si suppone
                              sia accettato dal sistema)
mansione                     :stringa di 10 caratteti
livello                      :intero
```

Il file e' ordinato per numero impiegato crescente. Inoltre si hanno delle scheda di aggiornamento distinte da un codice che puo' essere:

A per aggiunto;
L per licenziato;
M per modificato.

Se si tratta di licenziamento, la scheda non reca altri dati, mentre negli altri due casi essa reca i dati nuovi o modificati. Scrivete il programma che crea il nuovo file del personale aggiornato. Per semplificare il problema della fine dei file supponete che il pacco di schede termini con una scheda tappo recante il codice M e la copia dei dati dell'ultimo impiegato del file principale. Cosi' si arrivera' contemporaneamente alla fine dei due file. Supponete inoltre che il pacco di schede di aggiornamento non contenga errori (se volete aggiungete il trattamento di eventuali errori).

ESERCIZIO 7.4. - Un file contiene due tipi scheda:

primo tipo: codice "C", scheda cliente recante il nome e l'indirizzo del cliente;
secondo tipo: codice "B", scheda buono d'ordine recante la quantita' (intero) , la descrizione del=l'articolo (stringa 30 caratteri) e il prezzo unitario (reale).

Tutte le schede ordine di un cliente seguono la scheda del cliente. Tutti i clienti hanno almeno una scheda ordine. Il problema consiste nella stampa delle fatture nella forma:

```
                                    Nome cliente
                                    Indirizzo


                 Quantita'  Descrizione  Prezzo   Totale
Linea ordine     ........   ...........  ......   ......
                 ........   ...........  ......   ......
                                         Totale:  ......
```

Questi due esercizi mostrano che il Pascal e' comodo per trattare applicazioni che si presentano spesso nei problemi gestionali. Naturalmente la loro realizzazione e' possibile

anche in Basic e Fortran, ma magari meno comodamente.

7.8. IL TIPO TEXT

Il Pascal puo' essere facilmente utilizzato per applicazioni di trattamento di testi; infatti esiste un tipo standard per questo.

Infatti tutto avviene come se si disponesse di:

```
TYPE TEXT = PACKED FILE OF CHAR;
```

ed anche:

```
VAR INPUT, OUTPUT : TEXT;
```

I file standard del sistema sono di tipo TEXT, ma se ne possono creare altri.

Una caratteristica di questi file e' che essi sono organizzati in linee il cui carattere finale e' il carattere di fine linea. Si hanno di solito 80 caratteri per linea per INPUT e 132 per OUTPUT.

Quando in fase di lettura si incontra il carattere di fine linea il buffer e' considerato come contenente uno spazio e la condizione EOLN (o EOLN(FILE)) e' a TRUE.

Il Pascal ha introdotto per i file TEXT le procedure READLN e WRITELN.

```
WRITELN e' equivalente a WRITE ( );
                         WRITE (car. fine linea);

READLN(FILE) e' equivalente a:
             WHILE NOT EOLN(FILE) DO GET(FILE);
             GET(FILE);
```

Anche qui se READLN e WRITELN sono usati senza nomi di file, INPUT e OUTPUT sono sottintesi. READLN e WRITELN possono avere uno o piu' parametri di dati. Essi possono anche non averne (e questo vuol dire semplicemente di andare a capo), mentre READ e WRITE devono avere almeno un parametro.

Una differenza importante rispetto ai file ordinari e' che le operazioni di lettura e scrittura sui file TEXT incorporano le conversioni di tipo; non e' necessario fare:

READ(variabile di tipo CHAR), si avra' la conversione delle stringhe di caratteri nei numeri del tipo voluto.

Si ha pero' una difficolta' con i tipi strutturati. In generale e' necessario un ciclo per leggere gli elementi uno per volta, salvo che per il tipo:

```
STRING = PACKED ARRAY [1..N] OF CHAR
```

ESERCIZIO 7.5. - Copiare un testo conservando le linee.

ESERCIZIO 7.6. - Leggere un testo formato da lettere e nel quale le parole sono separate da spazi. Determinare la frequenza di ogni lettera e la frequenza delle coppie di lettere (digrammi).

Un ultimo particolare: in alcuni sistemi, quando un file TEXT viene mandato su stampante, il primo carattere di ogni linea non viene stampato. Esso viene utilizzato per comandare il movimento della carta secondo la seguente convenzione:

+	per restare sulla stessa linea (sovraimpressione)
spazio	per andare a capo
0	per saltare una linea
1	per cambiare pagina.

Sono le stesse convenzioni usate in Fortran.

7.9. PROCEDURE READ E WRITE RIDEFINITE

Le procedure standard READ e WRITE sono in generale nelle diverse implementazioni del Pascal capaci solo di leggere o scrivere variabili di tipo standard. Per trattare variabili di tipo strutturato si deve o usare sequenze del tipo:

```
ARTICOLO :=F@;            F@ := ARTICOLO;
GET(F);                   PUT(F);
```

o definire una procedura READ o WRITE o READLN, dato che si ha il diritto di "ridefinire" gli identificatori standard.

Abbiamo supposto, per semplicita', di procedere cosi' nella risoluzione delli esercizi 7.4. e 7.5., ma incoraggiamo i lettori a tentare la soluzione con GET e PUT.

CAPITOLO 8

GESTIONE DINAMICA DEI DATI

8.1. INTRODUZIONE ALLE STRUTTURE DI DATI

Arriviamo ora ad uno dei concetti piu' originali del Pascal: la gestione dinamica dei dati, cioe' la possibilita', quando dei dati non sono utilizzati, di liberare lo spazio di memoria da essi occupato, per dedicarlo ad altri dati. Naturalmente il Pascal ha introdotto un nuovo tipo di dati, il tipo POINTER, per poter realizzare quanto detto sopra.

Per poter apprezzare i vantaggi dei diversi tipi di dati che il Pascal permette di trattare, diamo ora qualche nozione generale sulle strutture dei dati. In seguito, per ogni organizzazione di dati trattata, vedremo quali possibilita' offre il Pascal.

Abbiamo gia' trattato un buon numero di strutture di dati. Qualunque raggruppamento di dati viene carattetizzato dalla sua struttura e dalla sua organizzazione cioe' da:

. i legami o le relazioni che esistono tra i dati dell'insieme (per esempio, i dati sono ordinati o classificati?);
. le operazioni elementari che si possono fare sui dati , e, in certi casi, il modo particolare nel quale si possono fare alcune operazioni elementari (per esempio, nell'organizzazione di una fila d'attesa si puo' aggiungere un elemento solo dopo tutti gli altri).

Vediamo subito quali sono le operazioni elementari che riguardano in generale qualunque insieme di dati.

ACCESSO

La prima operazione da realizzare su un dato appartenente a un insieme e' di potervi accedere per consultazione o modifica.

Nel caso di una tabella di dati, l'accesso ad un qualunque elemento e' possibile se si conosce la sua posizione. Se questa non e' conosciuta si deve fare una ricerca sistematica. Nel caso la tabella sia ordinata, si potra' fare una ricerca dicotomica, guadagnando in rapidita'. In

una struttura a liste concatenate bisogna seguire passo passo la catena per accedere ad un elemento.

Alcune strutture impongono delle restrizioni di accesso; nella struttura FILE e' accessibile un solo elemento ad un dato istante. In una struttura a fila d'attesa un elemento puo' essere aggiunto solo in fondo e solo il primo elemento e' accessibile.

AGGIUNTA

L'introduzione di un nuovo elemento in un insieme (SET) e' facile.

In una tabella si pongono due problemi; se la tabella non e' piena e non e' in ordine, l'elemento si aggiunge in fondo. Se la tabella e' ordinata si deve inserire l'elemento al posto giusto e spostare gli elementi che vengono dopo. Questa inserzione e' molto piu' semplice con una struttura a liste, come vedremo. Con un file l'aggiunta e' possibile solo alla fine, dopo essersi posizionati in fondo al file. Con una pila o una fila d'attesa l'elemento aggiunto puo' solo andare in fondo.

SOPPRESSIONE

La soppressione di un elemento di un insieme (SET) e' facile.

In una tabella essa e' complicata; si deve o colmare il buco creato, o lasciare l'elemento al suo posto marcandolo come soppresso. In una pila il solo elemento che puo' essere soppresso e' quello che era stato messo per ultimo. In una fila d'attesa e' quello che era stato introdotto per primo. In un file sequenziale il solo modo per sopprimere un elemento e' di ricopiare i dati su un nuovo file.

Tra le strutture ora citate, noi conosciamo bene l'insieme, il file sequenziale e la tabella. Queste strutture si realizzano molto bene in Pascal, dato che esse sono associate ad un tipo, rispettivamente SET, FILE e ARRAY. Nell'ultimo caso si puo' avere un ARRAY di RECORD. Quest'ultima struttura permette di realizzare le altre organizzazioni.

Diamo ora un breve cenno a qualche altra organizzazione di dati ed al modo per realizzarla in Pascal.

8.2. LA PILA

La pila e' una organizzazione caratterizzata dal fatto che il solo elemento accessibile e' l'ultimo aggiunto. Esso prende il nome di sommita' della pila. Un elemento puo' solo essere aggiunto alla sommita'. Esso diventa la nuova sommita'. Solo l'elemento che si trova alla sommita' puo' essere prelevato; quello sotto diventa la nuova sommita'.

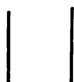
Questo e' esattamente il comportamento di una pila di piatti. Si dice anche che si segue la regola LIFO, last in, first out (ultimo entrato, primo uscito).

Questa struttura e' molto importante; essa serve a generare gli indirizzi di ritorno dei sottoprogrammi, dato che permette le chiamate nidificate. Ad ogni chiamata l'indirizzo di ritorno e' messo in una pila.

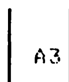
Il ritorno si ottiene togliendo dalla pila l'indirizzo che si trova alla sommita' ed operando un salto a quell'indirizzo.

Per esempio, qui sotto viene mostrato lo stato di una pila per successive chiamate di sottoprogrammi:

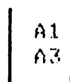
```
PROGRAM...;
 PROCEDURE A;
  PROCEDURE B;
  END; *B*
 BEGIN *A*
  B; *A1*
  B; *A2*
 END; *A*
BEGIN
  A; *A3*
END.
```



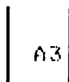
vuota



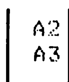
chiamata A



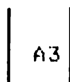
prima chiamata B



ritorno B



seconda chiamata B



ritorno B

ritorno A

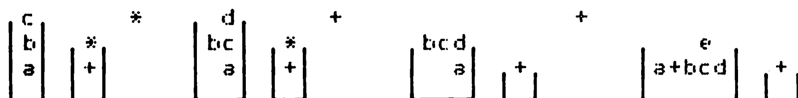
ESERCIZIO 8.1. - Data la struttura del programma che segue, disegnate gli stati successivi della pila degli indirizzi di ritorno.

```
PROGRAM...;
 PROCEDURE A;
  PROCEDURE B;
  END;  *B*
 BEGIN  *A*
  B;  *A1*
 END;   *A*
BEGIN
  A;  *A2*
  A;  *A3*
END.
```

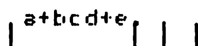
La struttura a pila si ritrova anche nella interpretazione delle espressioni aritmetiche. La regola e' la seguente (supponiamo che tutte le operazioni siano a due operandi):

. gli operandi sono messi in una prima pila come sono incontrati e gli operatori sono messi in una seconda pila.
. quando si comincia a prelevare dalle pile si prende l'ultimo operatore dalla pila degli operatori e si usa per i due operandi piu' in alto nella pila degli operandi. Il risultato viene posto nella pila degli operandi.
. prima di mettere nella pila un operatore esso viene confrontato con l'operatore che sta alla sommita' della stessa pila. Se il nuovo operatore ha una priorita' inferiore o pari, si toglie dalla pila fino ad avere un operatore di priorita' inferiore alla sommita' della pila.
. quando si arriva alla fine dell'espressione, si toglie dalla pila.
. quando si incontra una parentesi chiusa si toglie dalla pila fino a trovare una parentesi aperta.

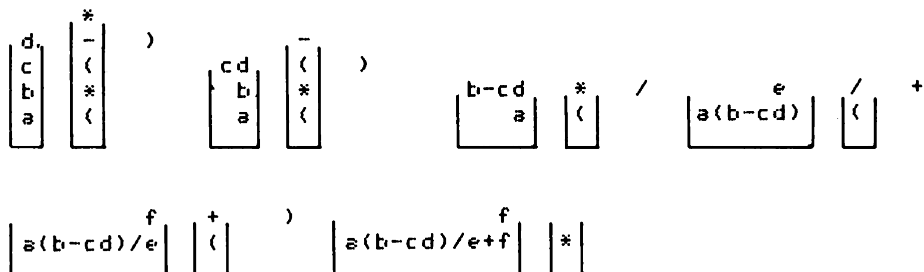
Esempi: a + b * c * d + e



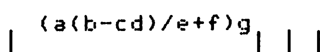
e finalmente:



(a * (b - c * d) / e + f) * g



e finalmente:



ESERCIZIO 8.2. - Disegnare lo stato delle pile per calcolare:
(a + b) * (a - b).

Come organizzare una pila in Pascal? E' molto facile. Basta una tabella per contenere gli elementi della pila e una variabile supplementare per contenere l'indice della sommita'.

```
VAR PILA: ARRAY [1..MAX] OF ...;
    SOMMIT : INTEGER;
```

L'essenziale della procedura IMPILARE (ELEMENTI) e' allora:

```
IF SOMMIT = MAX THEN WRITELN ('PILA PIENA')
                ELSE BEGIN
                       SOMMIT := SOMMIT + 1;
                       PILA [SOMMIT] := ELEMENTI
                     END;
```

Mentre la procedura DEPILARE (ELEMENTI) si riduce a:

```
IF SOMMIT = 0 THEN WRITELN ('PILA VUOTA')
              ELSE BEGIN
                     ELEMENTI := PILA [SOMMIT];
                     SOMMIT := SOMMIT -1
                   END;
```

Il principale inconveniente consiste nel fatto che si e' obbligati a fissare (quindi a prevedere) il numero massimo

di elementi della pila (MAX). Vedremo piu' avanti un modo per superare questa limitazione.

ESERCIZIO 8.3. - Il Pascal permette la recursivita' delle procedure dal momento che oltre alla pila per gli indirizzi di ritorno esso mantiene una pila per ogni argomento e ogni variabile locale. Supponete che questo non sia realizzabile. Riscrivete il programma HANOI del Cap. 6 (senza disegno) sistemando N, I e J in tabelle di variabili globali. Dopo scrivete una versione Basic del programma.

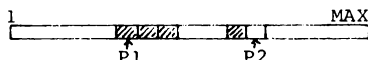
8.3. LA FILA D'ATTESA

La fila d'attesa assomiglia alla pila, solo che la regola seguita e' FIFO: first in, first out (primo entrato, primo uscito).

Questa e' la regola con la quale viene smaltita una coda. Si hanno due processi concorrenti: uno, il produttore, aggiunge elementi alla fila e li aggiunge alla fine, l'altro, il consumatore, prende gli elementi che sono presenti da piu' tempo.

Anche in questo caso, una tabella permette di realizzare questa struttura in Pascal. Pero' questa volta occorrono due puntatori: P1 che punta all'elemento da prelevare, P2 che punta alla posizione libera dove puo' essere aggiunto un elemento.

Si deve sempre prevedere un numero massimo di elementi per la tabella:

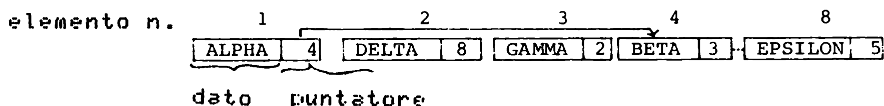


ESERCIZIO 8.4. - Scrivete la parte essenziale della procedura METTERE(ELEMENTO) e TOGLIERE(ELEMENTO).

La tabella viene gestita come un buffer circolare. Se P2 raggiunge P1, la pila e' piena. Se P1 raggiunge P2 la pila e' vuota. Questo puo' essere notato usando le variabili booleane PIENA e VUOTA inizializzate rispettivamente a FALSE e TRUE. Se PIENA non viene chiamata la procedura METTI.

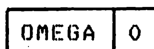
8.4. LE LISTE

In una struttura a liste ciascun elemento ha un successivo; pero' gli elementi non sono in ordine. In conseguenza ogni elemento e' formato da due parti: il dato propriamente detto piu' un puntatore che contiene l'indirizzo dell'elemento successivo. Esempio:

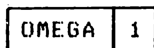


Uno speciale puntatore determina qual'e' il primo elemento, oppure, spesso per convenzione, il primo elemento e' il numero 1 nell'ordine.

L'ultimo elemento della lista possiede uno speciale puntatore che con il suo particolare valore indica che non esiste un successivo, per esempio con il valore 0.



Quando l'ultima registrazione punta alla prima, si avra' invece:



e la lista si dice "circolare".

Quando si ha una serie di puntatori per cui ciascun elemento punta solo all'elemento che lo precede, la lista si dice "reciproca".
Quando invece si hanno contemporaneamente per ogni elemento due puntatori, in avanti e all'indietro, la lista si dice "doppia".
Una lista si dice "multipla" se esistono piu' serie di puntatori che permettono di realizzare diversi tipi di ordinamenti.

L'accesso ad un elemento si ottiene percorrendo la lista in base ai puntatori. La soppressione di un elemento e' facile; l'elemento da togliere non viene soppresso fisicamente, basta aggiornare il puntatore dell'elemento precedente in modo che punti all'elemento successivo a quello da eliminare.

L'aggiunta di un elemento e' pure facile. L'elemento da aggiungere viene posto fisicamente alla fine della lista, basta aggiornare i puntatori in modo che il suo predecessore punti alla giusta posizione del nuovo elemento e questo punti al suo elemento successivo.

Come realizzare questo tipo di struttura in Pascal? E' molto facile, basta una tabella di registrazioni:

```
 TYPE ELEMENTO = RECORD
                     DATO  : tipo desiderato;
                     PUNTATORE : INTEGER;
                   END;
     LISTA ARRAY [1..MAX] OF ELEMENTO;
  VAR MALISTA : LISTA;
      PRIMO   : INTEGER;
      LIBERO  : INTEGER;
```

PRIMO e' lo speciale puntatore al primo elemento. LIBERO e' il puntatore al primo elemento libero nella tabella. Ogni puntatore contiene l'indice nella tabella dell'elemento successivo a quello considerato.

Naturalmente la struttura RECORD potrebbe contenere piu' puntatori nel caso di lista multipla.

Si ha ancora la limitazione di dover stimare a priori la dimensione massima della lista.

ESERCIZIO 8.5. - Sopprimete il quinto elemento di MALISTA (attenzione e' il quinto elemento nell'ordinamento della tabella, ma non necessariamente fisicamente il quinto elemento della tabella).

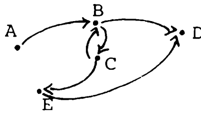
ESERCIZIO 8.6. - Inserite l'elemento INFOR tra l'elemento di rango I e quello di rango I+1 della lista MALISTA.

La lista e' il prototipo di altre strutture rappresentabili con l'aiuto di puntatori; i grafi e il caso particolare degli alberi.

8.5. I GRAFI

Un grafo e' un disegno formato da punti chiamati "sommita'" o "nodi" collegati tra loro per mezzo di frecce o

archi. Quando i legami tra i nodi non sono frecce, ma linee non orientate (si dicono anche "spigoli"), il grafo si dice "bidirezionale".



A ——— B corrisponde a

I grafi sono importanti per le applicazioni; una rete di strade si puo' rappresentare con un grafo, come pure una rete di canali. Gli archi del grafo possono essere associati a dei numeri, per esempio distanze o prezzo del pedaggio per una rete di strade, portata massima per delle canalizzazioni. Il grafo e' detto "pesato".

Come rappresentare un grafo in un calcolatore? Un primo metodo puo' essere quello di creare una matrice N x N se il grafo ha N nodi. La matrice sara' booleana se il grafo non e' pesato, e tale che A [I,J] = TRUE se esiste un arco che congiunge il nodo I con il nodo J, A [I,J] = FALSE nel caso contrario.

Una proprieta' interessante e' la seguente: se si forma il quadrato logico della matrice :

$$A^2_{ij} = \Sigma A_{ik} . A_{kj}$$

dove Σ e' la somma logica (OR) e . il prodotto logico (AND), la matrice prodotto e' tale che l'elemento Aij risulta TRUE, se esiste un cammino formato da 1 o 2 archi che vanno da i a j. Estendendo e considerando la matrice potenza logica di ordine P della matrice A, il suo generico elemento risulta TRUE se esiste un cammino di al massimo P archi che collega i due nodi interessati. Oltre la potenza ennesima la matrice non evolve piu' e se un generico elemento risulta FALSE e' perche non si ha alcun cammino che congiunge i due nodi relativi.

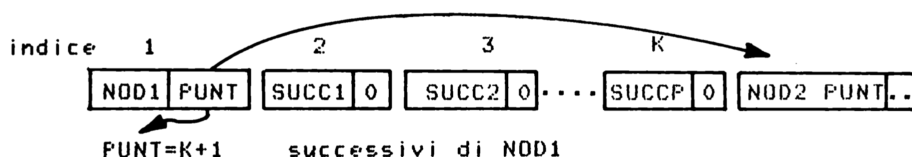
Questa rappresentazione permette di trattare facilmente alcune applicazioni di ricerca di un cammino in un grafo, ma risulta un po' pesante, dato che si devono elaborare matrici di ordine N x N.

ESERCIZIO 8.7. - Suggerite una rappresentazione per un grafo pesato.

Sono state proposte altre rappresentazioni che ricorrono ai puntatori: ciascun nodo punta verso il successivo. Per un grafo non pesato si potranno fare queste dichiarazioni:

```
TYPE SOMMIT = RECORD
                NOME : STRING;
                PUNT : INTEGER;
VAR GRAFO : ARRAY [1..MAX] OF SOMMIT;
```

e la rappresentazione avra' il seguente aspetto:



Ogni nodo e' seguito dai suoi successori. Quando un nodo e' considerato come successore il suo puntatore e' a 0. E' solo come testa di lista che il nodo ha un puntatore al prossimo nodo testa di lista.

ESERCIZIO 8.8. - Disegnate la tabella corrispondente al grafo disegnato all'inizio del paragrafo.

Anche qui e' necessario prevedere il numero massimo di elementi che puo' contenere la tabella. Quando la tabella non e' piena, LIBERO punta al primo elemento vuoto. L'ultimo nodo viene riconosciuto perche' punta verso un elemento vuoto. La lista dei predecessori puo' essere gestita contemporaneamente.

8.6. ALBERI

Un albero e' un grafo particolare dove non si hanno mai cerchi e dove un nodo puo' avere un solo predecessore. Il primo elemento si chiama "radice". Gli elementi terminali si chiamano "foglie". Un esempio molto comune della struttura ad albero e' l'albero genealogico.

Una possibile rappresentazione e' la seguente: l'insieme dei "figli" di uno stesso "padre" forma una lista nella quale ogni figlio punta verso il fratello che lo segue. L'ultimo fratello punta a 0. Ogni padre ha un puntatore verso il primo figlio.

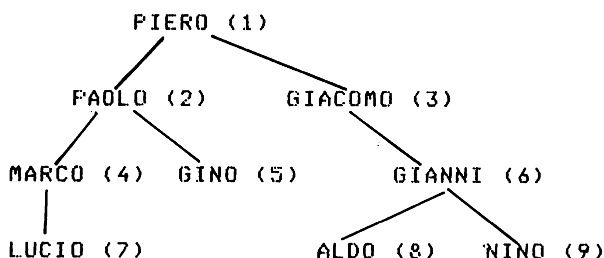
I figli dell'ultima generazione (foglie) puntano a 0. Ogni figlio ha anche un puntatore verso suo padre. Lo "antenato"

(radice) punta a 0.

In Pascal la struttura deve essere contenuta in un ARRAY OF RECORD:

```
TYPE GENE = RECORD
              NOME   : STRING;
              PADRE  : INTEGER;
              FRAT   : INTEGER;
              FIGLIO: INTEGER;
            END;
       ALBERO=ARRAY [1..MAX] OF GENE;
VAR FAMIGLIA : ALBERO;
```

Esempio: (tra parentesi figura l'indice nella tabella).



Si ha:

1	PIERO	0	0	2
2	PAOLO	1	3	4
3	GIACOMO	1	0	6
4	MARCO	2	5	7
5	GINO	2	0	0
6	GIANNI	3	0	8
7	LUCIO	4	0	0
8	ALDO	6	9	0
9	NINO	6	0	0

Dove il primo numero dopo il nome punta all'indietro, il secondo punta a destra ed il terzo punta verso il basso.

ESERCIZIO 8.9. - Trovate il padre di GIANNI.

Anche qui si deve prevedere il massimo numero di elementi. La struttura e' abbastanza facile da trattare. Se per esempio si e' dimenticato GIULIO, terzo figlio di PAOLO, questo verra' aggiunto come decimo elemento e si dovra' correggere solo il secondo puntatore di GINO per mandarlo a 10.

10 GIULIO	2	0	0

GINO	2	10	0

Un albero nel quale tutti i padri hanno due figli si chiama albero binario. Esso interviene nella interpretazione delle espressioni aritmetiche.

Una operazione abituale sugli alberi e' la ricerca per ordine. Essa consiste nel prendere ad ogni nodo l'elemento successivo piu' a sinistra. Quando si arriva ad una foglia si risale e si prende il nodo restante piu' a sinistra. L'albero viene percorso dall'alto verso il basso e da sinistra verso destra.

Esempio: per l'albero su esposto l'ordine di percorrenza e':

PIERO, PAOLO, MARCO, LUCIO, GINO, GIACOMO, GIANNI, ALDO, NINO.

Per una consultazione di questo tipo e' comodo servirsi di una pila che contenga il numero dell'ultimo nodo esaminato. Si "impila" quando si scende lungo l'albero e si "depila" quando si risale.

Ecco terminata la nostra introduzione alle principali strutture di dati. Abbiamo visto come possono essere organizzate in Pascal usando dei metodi classici e questo ci e' servito per far pratica di alcune possibilita' offerte dal linguaggio.

Il difetto principale delle realizzazioni trattate fino ad ora (per altro esse risultano molto comode) e' il loro carattere statico. Si deve sempre prevedere una dimensione massima per le tabelle. Vedremo ora che il Pascal consente di non preoccuparsi delle dimensioni massime delle tabelle, gestendo direttamente i puntatori ed occupando o liberando le zone di memoria in modo dinamico a seconda dei bisogni del programma.

8.7. TRATTAMENTO DINAMICO DEI DATI

Supponiamo di voler costruire una lista i cui elementi sono del tipo:

```
TYPE ELEM = RECORD
              INF : type...;
                .
                .
            END;
```

Il Pascal permette di definire un tipo, il tipo puntatore, i cui elementi sono dei puntatori verso gli oggetti del tipo ELEM:

```
TYPE PUNT = @ELEM;
VAR P,PRIMO:PUNT;
```

La prima linea si legge "tipo PUNT uguale puntatore verso oggetti del tipo ELEM".

Quale valore assegnare ad una variabile come P? Le due sole assegnazioni possibili sono:

P := NIL;
P := puntatore verso un oggetto dello stesso tipo di quello verso il quale P punta.
Esempio: P:=PRIMO;

La prima assegnazione fa si' che P punti verso nulla; NIL e' la costante "puntatore vuoto". La seconda fa si' che P punti verso lo stesso elemento che PRIMO.
L'elemento verso il quale P punta si indica con P seguito dal carattere "at" (@, o da altro). La parte "dato" dell'elemento si indica, come d'abitudine, con P@.INF.

Le variabili puntate si trovano in una zona speciale della memoria, la zona dinamica (chiamata in inglese "the heap", il cumulo).
I puntatori invece sono in una zona statica (lo stack, la pila) o in una zona dinamica. In effetti una registrazione dinamica puo' contenere puntatori, e questo e' precisamente quello che succede con una lista.

Non si deve confondere l'effetto di:

P := Q (P punta verso lo stesso oggetto che Q)

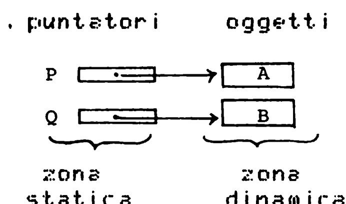
con quello di:

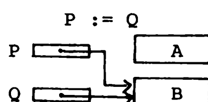
P@ := Q@ (l'oggetto verso il quale punta P ha lo stesso valore che l'oggetto verso il quale punta Q).

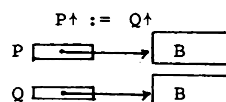
Questo viene evidenziato dalla figura che segue:

```
TYPE PUNT = @OGGETTO;
VAR  P, Q = PUNT;
```

dopo: dopo:







Ma non abbiamo ancora visto come creare un oggetto puntato. Questo e' lo scopo della istruzione NEW (procedura standard):

```
NEW(P);
```

che riserva nella zona dinamica lo spazio necessario per un oggetto puntato da P. E' necessario che prima sia stato definito P come puntatore verso l'oggetto, ed inoltre che sia stato definito il tipo dell'oggetto, in modo che il sistema sia in grado di definire la quantita' di memoria necessaria per l'oggetto stesso.

NEW(P) fa tre cose:

- ricerca nella zona dinamica il primo spazio capace di contenere un oggetto del tipo considerato;

- riserva tale spazio;

- assegna a P il valore dell'indirizzo di memoria corrispondente a questo spazio.

Questo era esattamente quello che noi facevamo prima, quando gli oggetti puntati venivano sistemati in un ARRAY OF RECORD. Solo che noi dovevamo farlo esplicitamente, mentre, con le variabili dinamiche, il lavoro lo fa il sistema. Inoltre noi non dobbiamo piu' preoccuparci della dimensione massima della tabella, dal momento che se ne occupa il sistema.

Tuttavia, lo spazio di memoria dinamica disponibile non e' illimitato, e, se noi facciamo troppe NEW una dopo l'altra, si avra' un messaggio di errore del tipo "supero di capacita' di memoria".

Si ha comunque un enorme vantaggio nell'utilizzo delle variabili dinamiche che permette il Pascal. Quando ci accorgiamo che una variabile dinamica non serve piu' (per esempio, un elemento di lista soppresso), noi possiamo

liberare la zona di memoria occupata scrivendo:

```
DISPOSE(P);
```

Con gli ARRAY OF RECORD potevamo risistemare i pseudo-puntatori in modo da rendere inaccessibile l'elemento soppresso, pero' esso restava in memoria. Se noi avessimo voluto riutilizzare lo spazio di memoria liberato, avremmo dovuto realizzare una gestione abbastanza complicata degli spazi liberati; infatti essi sono sparpagliati in una tabella.

In questo caso e' il sistema che pensa a gestire questi spazi.

NOTE:

- Alcune implementazioni del Pascal dispongono di una procedura DISPOSE incompleta, che aggiona i puntatori (essa fa P:=NIL), ma non libera lo spazio corrispondente in memoria.

- Esiste una altra coppia di procedure per la gestione della memoria dinamica: MARK e RELEASE. L'uso di queste procedure e' abbastanza complicato e noi non entriamo nei dettagli.

- La sequenza: NEW(P);
  NEW(P); e' errata. Infatti non si potra' accedere che al secondo dato creato. Si dovra' procedere invece cosi':

```
NEW(P);
Q :=P;
NEW(P);
```

e si avra' che Q@ sara' il primo dato e P@ il secondo.

- Ricordiamo ancora che a seconda dei calcolatori viene usato o il carattere @ o il carattere "freccia su" o altro.

8.8. LISTE DINAMICHE

Ora siamo in grado di creare le nostre liste dinamiche. Il problema e' che la registrazione del tipo ELEMENTO deve contenere, come componente, un puntatore verso un oggetto del tipo ELEMENTO. Si e' in presenza di un circolo vizioso: per definire il puntatore e' necessario che sia definito l'elemento; per definire l'elemento deve essere stato

definito il puntatore.

Fortunatamente il Pascal permette di scrivere:

```
TYPE PUNT = @ELEMENTO;
     ELEMENTO = RECORD
                  INFORM : type oggetto;
                  SEGUENTE : PUNT
                END;
VAR P, PRIMO:PUNT;
    X : type oggetto;
```

La dichiarazione di PUNT e', con la procedura FORWARD, l'unica occasione nella quale in Pascal si fa riferimento ad un oggetto non ancora definito.

In partenza la lista e' vuota. Noi scriviamo:

```
PRIMO := NIL;
```

In seguito, creiamo un posto per un oggetto che leggiamo su una scheda:

```
NEW(P);
READ(P@.INFORM);   oppure   READ(X);
                            P@.INFORM:=X;
                            (vedere nota alla fine del
                             Cap.7.)
```

Noi assegnamo al puntatore che si trova nell'elemento il valore di PRIMO (cioe NIL):

```
P@.SEGUENTE := PRIMO;
```

poi facciamo puntare PRIMO verso l'elemento:

```
PRIMO :=P;
```

Ripetendo questo procedimento, creiamo tutte le nostre liste. Bisogna notare che gli elementi vengono letti in ordine inverso a quello della lista; il procedimemnto di lettura di un nuovo elemento lo fa inserire in cima alla lista:

```
PROGRAM LETTLISTA;
    (dichiarazioni precedenti)
   BEGIN
     PRIMO :=NIL;
     READ(X);
     WHILE NOT EOF DO
       BEGIN
         NEW(P);
         P@.INFORM :=X;
```

```
        P@.SEGUENTE :=PRIMO;
        PRIMO :=P;
        READ(X);
       END;
   END.
```

In altre parole, con questa tecnica si forma una pila, dal momento che l'ultimo elemento introdotto e' il primo accessibile. Per costruire la lista, questo non e' molto fastidioso; basta presentare le informazioni in ordine inverso sulle schede dei dati.

ESERCIZIO 8.10. - Leggere la lista supponendo che gli elementi siano nell'ordine. Si potrebbe anche leggere come fatto sopra e poi sistemare tutti i puntatori.

ESERCIZIO 8.11. - Cosa dovrebbe essere cambiato per ottenere una lista circolare? Quale problema puo' essere semplificato dall'uso di una lista circolare?

L'operazione fondamentale per un tale tipo di lista e' quella di percorrerla nell'ordine. Questo permette di stamparne tutti gli elementi o di rispondere a domande del tipo: esiste un elemento il cui valore e'...?

Riportiamo, qui sotto, l'esempio della stampa della lista completa. Le dichiarazioni sono uguali alle precedenti.

```
P :=PRIMO
WHILE P<>NIL DO
   BEGIN
     WRITELN (P@.INFORM);
     P := P@.SEGUENTE;
   END;
```

Altre operazioni frequenti sono la soppressione o l'inserimento di un elemento. Se supponiamo che il dato, nei nostri elementi, sia di tipo STRING (dei nomi), l'operazione di soppressione della parola 'JOJO' si scrivera' come segue (supponendo che la parola esista nella lista):

```
P :=PRIMO;
IF P@.INFORM ='JOJO'
   THEN PRIMO := P@.SEGUENTE
   ELSE BEGIN
          WHILE P@.INFORM<>'JOJO' DO
            BEGIN
              PREC :=P;
              P := P@.SEGUENTE;
```

```
            END;
          PRECE.SEGUENTE :=P@.SEGUENTE;
        END;
    DISPOSE(P);
```

Ora vogliamo inserire 'LULU' dopo 'JOJO': (dichiarazione supplementare: VAR Q : PUNT;).

```
    P := PRIMO;
    WHILE P@.INFORM<>'JOJO' DO P :=P@.SEGUENTE;
    *P punta verso JOJO*
    NEW(Q); *viene creato il posto per LULU*
    Q@.INFORM := 'LULU';
    Q@.SEGUENTE := P@.SEGUENTE;
    P@.SEGUENTE :=Q;
```

ESEERCIZIO 8.12. - Considerate un caso di inserzione piu' delicato: inserire 'LULU' prima di 'JOJO'.

ESERCIZIO 8.13. - Si vuole costruire una lista doppia, cioe' con due puntatori in ogni registrazione; uno verso l'elemento seguente ed uno verso l'elemento precedente. Si hanno due puntatori esterni: PRIMO e ULTIMO. Supponete che la lista diretta sia gia' costruita. Si tratta di costruire la lista inversa, cioe' di definire i valori del secondo puntatore.

Ora abbiamo visto i principali modi di trattare i puntatori. Quelli concernenti le liste si adattano anche ai grafi e agli alberi, con l'appropriato trattamento dei puntatori.

La conclusione che si puo' trarre da questo capitolo e' che l'esistenza della gestione dinamica delle variabili in Pascal compensa dall'assenza delle dimensioni variabili per le tabelle. In realta' l'importanza delle dimensioni variabili e' quella di poter realizzare delle procedure generalizzate e di compilarle una volta sola. Pero' con i progressi ottenuti nella tecnica delle compilazioni, risulta sempre meno fastidioso dover ricompilare un programma.

In assenza delle dimensioni variabili, se si desidera ottenere un programma abbastanza generalizzato, si deve assegnare una dimensione massima superiore alle necessita' attuali, in modo da lasciare un margine di sicurezza. Si rischia pero' di superare la capacita' della memoria. E' per questo che un uso combinato delle diverse tecniche puo' risultare molto redditizio.

Supponiamo di dover leggere delle registrazioni senza sapere quante sono, e che ogni registrazione contenga molte informazioni, e che si debbano ordinare le registrazioni in ordine alfabetico in base a dei nomi.

Le registrazioni verranno sistemate nella zona dinamica, ma i puntatori saranno in una tabella nella zona statica.

L'algoritmo di ordinamento portera' sulla tabella dei puntatori sui quali saranno effettuate permutazioni fino ad ottenere che l'ordine dei puntatori sia quello degli elementi ordinati. PUNT[I]@.ELEM sara' l'elemento iesimo nell'ordinamento.

Le dichiarazioni essenziali saranno le seguenti:

```
TYPE PTR = @ELEM;
     ELEM = RECORD
                 COGNOME : STRING;
                 NOME : STRING;
                 DATANASC : STRING;
                 SPOSATO  : BOOLEAN;
                  ..
                  ..
                 IMPIEGATO : STRING;
                  ..
                  ..
                 INFORM : type...;
                  ..
               END;
VAR PT : ARRAY [1..100] OF PTR;*si suppongono meno di
                                100 elementi*
     X  : PTR;                  *variabile ausiliaria*
     N  : INTEGER;              *numero elementi*
```

Le istruzioni essenziali saranno le seguenti:

Lettura dei dati:

```
N := 0;
NEW(X);
READ (X@);
WHILE NOT EOF DO
   BEGIN
     N := N+1;
     PUNT [N] := X;
     NEW(X);
     READ(X@);
   END
```

Ordinamento:

```
REPEAT
```

```
   FOR I := 1 TO N-1 DO
     BEGIN
       IF PUNT [I]@.NOM>PUNT [I+1]@.NOM
              THEN SCAMBIO
     END
  UNTIL ECH = 0;
```

La procedura SCAMBIO sara':

```
PROCEDURE SCAMBIO;
  BEGIN
     ECH := 1;
     X := PUNT[I];
     PUNT[I] := PUNT[I+1];
     PUNT[I+1] := X;
  END;
```

ESERCIZIO 8.14. - Scrivere la parte di programma che stampa i dati ordinati. Si puo' ritrovare l'ordine di partenza?

Consigliamo vivamente il lettore, servendosi degli esempi riportati, di scrivere il programma completo. Siamo stati obbligati a prevedere una tabella di 100 elementi, ma questo non e' grave dato che ogni puntatore occupa, di norma, poco spazio, e comunque meno di quello occupato da un elemento. Quello che sarebbe stato grave, sarebbe stato il dover riservare spazio per 100 registrazioni. Al contrario si occupa solo lo spazio per N elementi.

ESERCIZIO 8.15. - Si e' consumato esattamente lo spazio corrispondente ad N elementi?

DOMANDA: Una FUNCTION puo' essere del tipo puntatore (risultato = puntatore)? Un argomento puo' essere un puntatore?

- Si: per esempio, FUNCTION PQ (P:PTR) : PTR;

CAPITOLO 9

CONCLUSIONI: IL PUNTO SUL PASCAL

9.1. CRITERI PER LA SCELTA DI UN LINGUAGGIO

E' ora giunto il momento di fare il punto sui pregi e i difetti del Pascal.

Passeremo in rassegna le qualita' che si ritengono generalmente auspicabili per un linguaggio di programmazione, ed esamineremo a quale grado il Pascal possiede queste qualita', confrontandolo con i suoi concorrenti.

Una difficolta' e' che certe caratteristiche di un linguaggio possono essere considerate dei pregi o dei difetti a seconda del punto di vista dal quale si guardano.

In conseguenza i vantaggi di un linguaggio devono essere valutati in funzione delle applicazioni. Tenteremo di fare questo per categorie di applicazioni alla fine del capitolo.

9.2. TRATTAMENTI PERMESSI

La prima cosa da esaminare in un linguaggio sono le operazioni disponibili. Al livello piu' basso significa considerare gli operatori e le funzioni presenti, ad un livello piu' elevato definire i tipi di applicazioni che possono essere affrontati.

Da questo punto di vista, il Pascal e' ragionevolmente potente. Esso possiede meno operatori aritmetici dell'APL e possiede poche funzioni matematiche, ma l'utente puo' definire le proprie funzioni come desidera, e troppe funzioni -create per motivi di concorrenza - nuocciono alla portabilita' del linguaggio.

Una deficienza puo' essere (e questo e' risolto in UCSD) la mancanza di funzioni sulle stringhe di caratteri.

Fanno parte di questo gruppo di caratteristiche i modi con i quali si puo' accedere alle risorse della macchina. Si ha un compromesso tra l'uso di un linguaggio evoluto, che normalmente non permette che un accesso limitato e poco

controllato a risorse come i registri interni, e l'uso del linguaggio macchina, che per definizione autorizza questi accessi.

Alcuni linguaggi evoluti permettono di accedere alle possibilita' del linguaggio macchina. E' il caso del PL/M (o linguaggi similari) sui microprocessori, e anche un po' il caso del Basic (come le istruzioni POKE e PEEK, USR e simili).

Il Pascal non offre alcuna possibilita' in questo campo per delle ragioni di portabilita'.

Sempre nello stesso gruppo, le strutture elaborative definiscono la forma dei programmi che possono essere scritti con il linguaggio.

Qui si ha un punto forte del Pascal, che incorpora le strutture della programmazione strutturata. Questo e' un vantaggio indiscutibile, soprattutto nella fase di apprendimento del linguaggio. I programmi sono molto chiari e si puo' costruire qualunque struttura partendo da un piccolo numero di strutture standard.

Infine, molto determinante per capire quali tipi di elaborazioni il linguaggio consente, e' l'insieme dei tipi e delle strutture di dati che si possono definire e trattare.
Da questo ounto di vista il Pascal e' il linguaggio piu' ricco: oltre ai tipi abituali di dati (interi, reali, booleani, carattere), l'utente puo' definire i propri tipi. Per quanto riguarda i tipi strutturati, il Pascal possiede le tabelle (come in Basic e in Fortran), le strutture gerarchiche (come in Cobol e in PL/1), ma anche gli insiemi, i puntatori, ecc....

La sola debolezza da questa parte sono alcune restrizioni sui numeri reali (una sola gamma di precisione disponibile, impossibilita' di definire degli insiemi o degli intervalli di numeri reali) e l'assenza nel Pascal standard delle istruzioni per trattare le stringhe di caratteri. Si deve notare che quest'ultima cosa e' corretta nel Pascal UCSD.

9.3. FACILITA' DI SCRITTURA

La seconda qualita' che poniamo in evidenza per un linguaggio di programmazione e' la facilita' di scrittura. Dal punto di vista della "liberta' di scrittura" il Pascal e' ben messo. Le restrizioni sugli spazi bianchi sono ragionevoli e favoriscono una adeguata impaginazione.

Le restrizioni sulla punteggiatura (i ;) sono un po' piu' delicate, ma sono logiche. E' ragionevole che le principali parole chiave siano riservate.

Facciamo notare che gli utenti piu' accorti non desiderano che un linguaggio sia troppo facile da scrivere. Inoltre se la scrittura e' troppo facile, questo incoraggia le cattive abitudini sanza apportare reali vantaggi. E' idiota inserire degli spazi nelle parole chiave, e' insensato scegliere un identificatore identico ad una parola chiave, anche se il linguaggio utilizzato consente di farlo!

Per contro, il Basic che proibisce che un identificatore contenga una parola chiave e' troppo restrittivo. Da questo punto di vista il Pascal tiene il giusto mezzo.

Una caratteristica che favorisce la stesura dei programmi e' il "grado di simbolizzazione" del linguaggio (vedere Cap. 3.). Noi abbiamo gia' visto che il Pascal e' ben messo da questo punto di vista; esso permette di attribuire un nome simbolico ai sottoprogrammi (e questo non e' permesso dal Basic), alle costanti e anche ai tipi.

La "concisione" di un linguaggio e' nello stesso tempo un pregio e un difetto. Una qualita' perche' i linguaggi troppo verbosi finiscono per essere antipatici; e' fastidioso dover scrivere molte linee per una operazione da nulla (confrontare con il Cobol).
Ma questo puo' anche essere un difetto, perche' va a diminuire la leggibilita' del programma. APL, per esempio, permette di scrivere una elaborazione abbastanza complessa in una sola linea, ma un programma dove si sfrutti molto questa possibilita' diventa difficile da comprendere.
Anche a questo proposito il Pascal dimostra un ragionevole equilibrio; senza gli eccessi dell'APL, esso permette delle scritture semplici, sfruttando le costruzioni della programmazione strutturata.

La necessita' di dichiarare tutte le variabili nuoce alla concisione, ma essa e' per altro molto utile alla leggibilita' del programma ed obbliga il programmatore a fare una buona preparazione per il suo programma pensando prima a tutte le variabili da utilizzare.

La "facilita' di apprendimento" del linguaggio si collega alle caratteristiche ora esaminate. Sebbene un po' meno facile del Basic (il programma piu' semplice del calcolo della superficie di un cerchio e' piu' lungo in Pascal che in Basic a causa delle dichiarazioni iniziali), il Pascal e' molto facile da affrontare. In effetti, ricorrendo alla programmazione strutturata e' possibile programmare

qualunque procedura usando un piccolo numero di strutture standard. D'altra parte il Pascal non ha quella che puo' essere ritenuta una barriera eretta contro i principianti, ci riferiamo al FORMAT.

Usando la struttura recursiva delle definizioni, si arriva abbastanza presto in Pascal a dei programmi complessi, difficili per i principianti. Ma bisogna anche dire che il Pascal e' il linguaggio che permette, ad un livello piu' elevato, di abbordare il piu' facilmente possibile nozioni d'informatica teorica, come la recursivita', i puntatori e altro.

Vediamo ora due qualita' molto importanti per la scelta di un linguaggio di programmazione. L'utilizatore avra' un considerevole aiuto dal fatto di poter utilizzare un gran numero di programmi gia' scritti da altri. Perche' questo sia possibile sono necessarie due cose:

- Usare un linguaggio molto diffuso per poter trovare gia' scritti programmi validi su diversi argomenti.
- Poter adattare al proprio sistema un programma scelto, senza troppa fatica. Questo e' il problema della portabilita'.

9.4. DIFFUSIONE DEL LINGUAGGIO

Attualmente il Pascal e' battuto da altri linguaggi su questo punto. Notoriamente dal Basic nel campo dei Personal, e dal Fortran sui mini e sui grossi calcolatori.

Ma questa situazione evolve rapidamente, e si puo' dire che presto tutti i Personal avranno il Pascal (pero' bisogna comprare il compilatore e questo ha un costo). Attualmente il Pascal e' un linguaggio abbastanza diffuso, e, tenendo conto dell'interesse che suscita, la sua diffusione aumentera' rapidamente.

9.5. PORTABILITA'

Si dice che un programma e' "portabile", quando scritto per una macchina A, esso puo' essere utilizzato senza modifiche su una macchina B (naturalmente dando gli stessi risultati!).

La portabilita' e' un fatto che da' credito al linguaggio utilizzato. E' in parte proprio per una ragione di

portabilita' che sono stati introdotti i linguaggi evoluti, per superare la non portabilita' dei linguaggi macchina.

I nemici della portabilita' sono: le restrizioni del linguaggio, le estensioni, le esigenze particolari dovute ai sistemi di elaborazione.

I creatori del Pascal hanno combattuto a fondo questi nemici per assicurare al linguaggio la massima portabilita'. Il miglior mezzo per evitare le varianti che rischiavano di essere introdotte dagli autori dei diversi compilatori era di fornire il compilatore insieme al linguaggio. E' quello che hanno fatto gli autori: essi forniscono un compilatore scritto in Pascal semplificato. Per realizzare il Pascal su una determinata macchina, basta scrivere nel linguaggio proprio della macchina un interpretatore del Pascal semplificato.

Da cosa dipende allora il fatto che, sebbene tutti gli autori sembrano riuniti per assicurare una perfetta portabilita', noi non abbiamo mai smesso nei precedenti capitoli di segnalare le differenze tra le diverse macchine?
Ebbene questo dipende dal fatto che la versione della quale noi abbiamo parlato, detta Pascal Zurich, aveva alcune restrizioni veramente pesanti (notoriamente la mancanza della elaborazione delle stringhe di caratteri e l'assenza dei file ad accesso diretto).

E' per questo che un'altra versione, il cui scopo era di compensare questi difetti, e' stata messa a punto all'Universita' di California a San Diego.

Il Pascal UCSD si presenta, anch'esso, sotto forma di compilatore scritto in un linguaggio intermedio, il codice P, ed e' sufficiente interpretarlo sul calcolatore di cui si dispone.

A questa fondamentale caratteristica si aggiungono le varianti minori che sono necessarie per passare da una macchina all'altra, dovute al diverso numero di bit per parola in ogni macchina, per esempio: la cardinalita' massima del tipo base di un insieme.

In questo stesso ambito, la precisione dei numeri reali puo' dare dei problemi piu' gravi nel calcolo scientifico.

Il difetto del Pascal al riguardo e' quello di autorizzare un solo tipo di numeri reali, che, per contro, offre una precisione diversa a seconda delle macchine. Quindi, e questo e' il difetto piu' insidioso della portabilita', dei programmi identici gireranno su diverse macchine, ma essi rischieranno di dare risultati diversi se si accumulano

degli errori di arrotondamento.

Questo e' compensato in Fortran, dove, sulle macchine che danno meno precisione, si puo' ricorrere ai numeri reali in doppia precisione, e questo comporta una modifica (non troppo pesante) del programma, ma si ha portabilita' maggiore a livello dei risultati.

La vera soluzione di questo problema si ha nel PL/1 dove, per ogni variabile reale si puo' chiedere il numero di cifre significative desiderato. La' si ha una completa portabilita' ed e' deplorabile che il Pascal non abbia ripreso questa tecnica.

Dunque, e' questo e' un peccato, nonostante le buone intenzioni manifestate dai suoi creatori, "il Pascal non e' dotato di una perfetta portabilita'": esso rimane nel plotone dei linguaggi concorrenti.

9.6. CONCLUSIONI

Noi ora abbiamo visto i pregi e i difetti del Pascal. Bisogna riconoscere che il bilancio e' molto positivo e, in effetti, tutti i linguaggi apparsi dopo il Pascal si ispirano direttamente ad esso.

E' il caso, notoriamente, del linguaggio ADA, scelto dal Dipartimento della Difesa degli Stati Uniti, linguaggio che si dichiara riunisca tutti i possibili vantaggi. E' stato formato un comitato per proporre un Pascal ANS (steso come il Cobol ANS) che, noi speriamo, correggera' i difetti che abbiamo segnalato.

Noi non diremo, come fanno alcuni, che il Pascal soppiantera' tutti gli altri linguaggi, perche' i linguaggi non scompaiono cosi' facilmente (ricordate da quanto tempo e' stata annunciata la scomparsa del Fortran), ma, in ogno modo, il Pascal ha diritto a un posto nel plotone di testa dei linguaggi di programmazione.

Per riassumere le nostre conclusioni in funzioni delle diverse categorie dei possibili utilizzi, prenderemo in prestito la presentazione fatta dalla rivista "L'Ordinateur Individuel", limitandoci agli elementi essenziali.

9.7. UTILIZZO SUI PERSONAL

A FAVORE
. Pascal UCSD e' (o sara' al piu' presto) disponibile sulla quasi totalita' dei Personal.
. Chiarezza dei programmi grazie alla programmazione strutturata.

CONTRO
. Necessita un Personal con configurazione di gamma alta (memoria e dischetti).
. Costa abbastanza comprare il linguaggio.
. Pascal non porta vantaggi nelle normali applicazioni, per le quali risulta piu' complicato del Basic.
. Esistono meno programmi scritti in Pascal che non in Basic.

9.8. UTILIZZO NELL'INSEGNAMENTO

A FAVORE
. Favorisce l'insegnamento della programmazione strutturata e permette di inculcare buone abitudini piu' facilmente degli altri linguaggi.
. Permette di abbordare facilmente alcune nozioni di informatica teorica (recursivita', tipi di dati, strutture dinamiche).

CONTRO
. Piu' difficile da affrontare del Basic per i principianti. A nostro avviso esso si impone se si vuole raggiungere un alto livello teorico, ma il Basic si impone se si vuole imparare a programmare il piu' rapidamente possibile una semplice applicazione.

9.9. UTILIZZO PROFESSIONALE

La chiarezza dei programmi, ottenuta grazie alla programmazione strutturata e' qui molto importante, puo' darsi anche piu' di qualche anno fa, dal momento che questi tipi di programmi sono diventati piu' lunghi e complicati. Tutto cio' che diminuisce il costo dello sviluppo e della messa a punto dei programmi e' fondamentale nell'utilizzo professionale.

Per essere piu' precisi, suddividiamo questo gruppo in tre

sottogruppi di applicazioni.

CALCOLO SCIENTIFICO

Un elemento sfavorevole importante e' il modo nel quale il Pascal tratta i numeri reali. L'impossibilita' di definire la precisione desiderata (possibile in modo ridotto in Fortran e in alcuni Basic con le variabili a doppia precisione, e possibile completamente in APL e soprattutto in PL/1) e', per il calcolo scientifico, uno svantaggio certo che pone il Pascal in cattiva luce in questo ambito.
Ammesso questo le altre caratteristiche del Pascal sono favorevoli al calcolo scientifico.

PROBLEMI GESTIONALI

Abbiamo gia' visto su alcuni esempi che il Pascal e' un buon linguaggio per i problemi gestionali, soprattutto grazie alla possibilita' di definire dei dati composti nella struttura RECORD.
Per contro, l'assenza dell'accesso diretto e del trattamento delle stringhe di caratteri e' molto limitativo nel Pascal Zurich. Solo una versione UCSD che consente l'accesso diretto e le stringhe di caratteri e' utilizzabile comodamente per i problemi gestionali.
In conseguenza controllate bene le caratteristiche del Pascal che vi propongono prima di acquistarlo.

RICERCA IN INFORMATICA

Questo campo, dove si fa appello alla recursivita', ai dati dinamici, agli insiemi,ecc...e' il dominio ideale per il Pascal.
Il Pascal e' eccellente per qualunque ricerca su algoritmi non numerici. Si puo' anche utilizzarlo con un certo successo nella stesura di sistemi operativi.

9.10. LA QUESTIONE DEL RENDIMENTO

Questa questione non e' ancora stata affrontata in questo libro. Alcuni mettono in evidenza il fatto che il Pascal, essendo per meta' compilato e per meta' interpretato (vedere App. C), dovrebbe dare prestazioni superiori al Basic che e' interamente interpretato (se il Pascal avesse un cosi' alto rendimento non necessiterebbe di tanta memoria come i 48K dell'Apple, uno dei Personal sul quale e' stato realizzato).

Per prima cosa, il fatto di essere interpretato (cioe' tradotto istruzione per istruzione, con esecuzione immediata

di ogni istruzione), o compilato (cioe' tradotto in blocco e poi eseguito in blocco) non dipende dal linguaggio. Questo dipende dalla implementazione considerata. Infatti esiste qualche implementazione interpretativa (leggete conversazionale) del Fortran o del PL/1, anche se questi linguaggi sono di solito compilativi. Inoltre esistono delle versioni compilate del Basic, chiamato allora CBASIC, anche se il Basic e' in generale interpretativo. Come si sa, la compilazione rende l'esecuzione piu' veloce che nel caso dell'interpretazione.

Il fatto che nelle implementazioni attualmente sul mercato (niente impedira' di realizzarne altre interamente compilate e la Texas propone un Pascal compilato sui suoi microcalcolatori basati sul microprocessore 9900, o interamente interpretate) il Pascal sia misto (si ha dapprima una compilazione del Pascal in un linguaggio intermedio e poi una interpretazione del linguaggio intermedio) rischia piuttosto di conferirgli gli svantaggi dei due metodi!

La sola realizzazione del Pascal dove tutti gli elementi concorrono a dare la massima velocita' e' quella della Pascal Microengine di Western Digital. In questa realizzazione, il processore ha come linguaggio macchina il linguaggio intermedio P nel quale il Pascal e' compilato. La velocita' e' allora quella di un sistema compilato.

Ma, ripetiamolo, questi confronti si riferiscono alle realizzazioni del linguaggio e non alla sua natura.

Noi non possiamo qui decidere completamente se si deve utilizzare il Pascal. Voi dovete fare le vostre scelte in funzione delle vostre applicazioni, il nostro ruolo era quello di passare in rassegna le caratteristiche piu' importanti da esaminare in vista di queste scelte. Speriamo di aver assolto al nostro compito.

Alcuni troveranno che siamo stati troppo critici verso il Pascal, ma nulla nuocerebbe di piu' all'espansione di questo linguaggio di un'armata di utenti scontenti perche' convinti ad una cattiva scelta da proseliti troppo attivi.

APPENDICE A

PROGRAMMAZIONE STRUTTURATA IN BASIC

Il Pascal e' stato inventato soprattutto per realizzare facilmente la programmazione strutturata e per presentarla bene, ma anche gli altri linguaggi consentono di realizzarla. Vediamo questo piu' in dettaglio per il Basic.

LE STRUTTURE DI BASE

Il Basic non possiede a priori le strutture di base della programmazione strutturata:

```
se...allora...se no
ripetere...fino a che..
fintanto che... fare...
```

ma queste strutture possono essere simulate con l'aiuto delle istruzioni di salto del Basic.

La forma ridotta "se..allora.." e' realizzabile direttamente in BAsic da "IF..THEN..".

Quando si devono eseguire parecchie istruzioni se la condizione e' verificata: in Pascal:

```
IF..THEN
 BEGIN
  istr.1;
  ..
  istr.n;
 END
```

si puo', in alcuni Basic, scrivere le istruzioni una dopo l'altra dopo il THEN separandole con i due punti (:), cosi':

```
IF...THEN istr.1: istr.2: ....: istr.n
```

Questo e' il caso del Basic Microsoft (PET, TRS) ma si hanno due limitazioni:

- la sequenza di istruzioni deve stare sulla stessa linea;
- questo nuoce all'impaginazione del programma.

In ogni caso si puo' ricorrere ad una simulazione, come

per il caso generale sotto riportato. Alcuni Basic (pochi, per la verita') possiedono l'istruzione: IF...THEN...ELSE. In tutti i casi si puo' procedere cosi':

```
.(a) - IF NOT condizione THEN GOTO n
     - REM SI condizione
     - ....   (istruzioni da eseguire se la condizione
     - ....    e' verificata)
     - GOTO p
     n REM NO condizione
     - ....   (istruzioni da eseguire se la condizione
     - ....    non e' verificata)
     p ....   (seguito)

.(b) - IF condizione THEN GOSUB n: GOTO p
     - ...    (istruzioni per ELSE)
     p        (seguito)
     - ...
     n ....   (sottoprogramma da eseguire se la
     - ....    condizione e' verificata)
     - RETURN
```

Notate che il ricorso ad un sottoprogramma del caso (b) puo' essere raccomandabile anche in Pascal.

La struttura "ripetere.....fino a...." si codifica come segue:

```
.(c) p IF condizione-arresto GOTO n
     - ....    (istruzioni da ripetere)
     - ....
     - GOTO p
     n ....     (seguito)
```

Notate che in molti Basic THEN GOTO e' sovrabbondante e basta usate o l'uno o l'altro; se nel vostro caso questo non e', aggiungete THEN dove manca.

La simulazione di cui sopra non e' del tutto esatta. Infatti, se durante il primo passaggio la condizione-arresto e' gia' verificata, le istruzioni da ripetere non vengono mai eseguite. In Pascal con REPEAT...UNTIL si ha almeno una esecuzione. Possiamo usare la forma (d) che segue:

```
.(d) p REM RIPETERE
     - ....       (istruzioni da ripetere)
     - ....
     - IF NOT condizione-arresto GOTO p
     - ....       (seguito)
```

Per la sequenza WHILE...DO..., invece, le istruzioni da ripetere non devono essere mai eseguite se la condizione non

e' realizzata all'inizio. Da cui la sequenza:

```
 .(e) p IF NOT condizione GOTO n
      - ....        (istruzioni da ripetere)
      - ....
      - GOTO p
      n ....        (seguito)
```

Utilizzando solo le strutture fin qui proposte, si ottiene una programmazione Basic molto chiara. Naturalmente non si devono usare le GOTO in modo diverso da come sopra mostrato.

Paradossalmente, le istruzioni di strutturazione del Pascal che risultano un aggiunta alla programmazione strutturata sono forse piu' facili da simulare: esse infatti corrispondono alle "sopravvivenze" dei linguaggi non strutturati.

- GOTO... e' la contropartita del GOTO del Pascal.
- il ciclo FOR..END e' direttamente effettuato dal FOR..NEXT del Basic, ma si hanno le seguenti differenze:

.1- Il FOR del Basic e' effettuato almeno una volta, qualunque sia la partenza, mentre in Pascal se il limite di partenza e' superiore al limite di arrivo, la sequenza non viene effettuata.

.2- Il FOR del BASIC non ha bisogno della parola chiave DOWNTO se il passo e' negativo.

.3- Il FOR del Basic non ha bisogno che l'indice corrente sia intero. Da questo punto di vista esso e' piu' potente del Pascal (una delle rare occasioni!). In Pascal non si hanno possibilita' di STEP.

ESERCIZIO A.1. - Realizzare qualcosa per X=1, arrivando a X=3 con passo 0.5 (si scrive in Basic FOR X=1 TO 3 STEP 0.5).Ottenerlo in Pascal diversamente che con X:=X+0.5. Pensate ai tipi.

La struttura CASE del Pascal ha un corrispondente in Basic, ma ben piu' restrittivo, ON ... GOTO oppure ON... GOSUB.

Per esempio:

```
 5 ON K GOTO 10,20,50,60
10 (istr. se K=1):GOTO 70
20 (istr. se K=2):GOTO 70
50 (istr. se K=3):GOTO 70
60 (istr. se k=4)
70 .....
```

```
corrisponde in Pascal a:  CASE K OF
                             1 : (istr. se K=1);
                             2 : (istr. se K=2);
                             3 : (istr. se K=3);
                             4 : (istr. se K=4);
                            END; (fine della deviazione)
```

ma, in Pascal, i valori di K non hanno bisogno di essere degli interi consecutivi e K puo' essere di un tipo speciale definito dal programmatore.

I SOTTOPROGRAMMI

Uno dei principali imperativi della programmazione strutturata e' quello di suddividere i programmi in piccoli moduli. Pascal lo permette grazie alle FUNCTION e alle PROCEDURE.

Il Basic possiede una forma molto ridotta di FUNCTION, le funzioni definibili con DEF FN. La funzione si deve ridurre a una linea (quindi essa non puo' contenere dei test), il suo nome deve essere FN seguito da una lettera e quindi non puo' avere significato mnemonico.

Il Basic permette i sottoprogrammi chiamati con GOSUB. Si ha qualche deficienza rispetto alle PROCEDURE Pascal, ma l'essenziale c'e'.

La restrizione essenziale, a nostro avviso, dal punto di vista dell'ottenimento di programmi parlanti, e' che il Basic non permette di richiamare un sottoprogramma con un nome, ma si deve scrivere GOSUB ad un numero di linea. Il solo modo di compensare questo fatto e' quello di inserire appropriati commenti. Esempio:

```
100 GOSUB 1000:REM INVERSIONE MATRICE
....
1000 REM ROUTINE INVERSIONE MATRICE
```

Le altre restrizioni sono:

- la non esistenza di variabili locali (vedere paragr. 6.4.), ma questo puo' essere superato con un po' di attenzione quando si programma;
- l'impossibilita' di usare degli argomenti. Questo e' un po' piu' spiacevole.Per esempio, se noi supponiamo che la routine 1000 sia scritta per invertire una matrice A, per poter usare la stessa routine per invertire una matrice B, questa deve essere trasferita in A prima di fare GOSUB 1000. In Pascal basta scrivere INVERSIONE(B) o, in Fortran CALL INV(B).

Se riassumiamo quanto precede, vediamo che in definitiva, dal punto di vista delle istruzioni, e' possibile fare della programmazione ben strutturata in Basic ed anche in Fortran. Perche' allora si dice che il Basic non e' un linguaggio strutturato?
E' abbastanza triste doverlo dire, ma sembra che questo dipenda da una sorta di tradizione nella presentazione dei programmi. La tradizione vuole che si presentino i programmi Basic (o Fortran) con dei commenti sparpagliati, senza alcun tentativo di impaginazione, mentre la tradizione (iniziata, bisogna riconoscerlo, dopo la programmazione strutturata) vuole che i programmi Pascal siano ben presentati. Estendiamo questa tradizione al Basic.

COMMENTI

Il Basic possiede una istruzione che permette di mettere dove si vuole dei commenti in un programma, e' l'istruzione REM. Non evitiamo di usarla.

Noi possiamo avere:

- un commento che occupa un'intera linea, per esempio come titolo di un sottoprogramma:

```
1000 REM SOTTOPROGRAMMA Di ORDINAMENTO
```

- una linea tutta di asterischi o di altri caratteri grafici per inquadrare:

```
5000 REM ********************************
```

- un breve commento alla fine di una linea:

```
100 istruzioni : REM spiegazioni
```

(notate che non si puo' scrivere:

```
100 REM spiegazioni : istruzioni
```

infatti la maggior parte dei Basic non analizza per eseguire quello che viene dopo REM).

La sola limitazione e' che con i Basic interpretativi i commenti occupano spazio in memoria. Lo stesso inconveniente si presenta riguardo ad una buona impaginazione. Nondimeno, bisogna notare che le configurazioni di Personal ora disponibili, soprattutto per i problemi gestionali, offrono quantita' di memoria abbastanza rilevanti (16 o 32 K) grazie alla diminuzione dei costi della stessa. Questa tendenza non fara' che accentuarsi e permettera' di poter usare piu' memoria a vantaggio della leggibilita' dei programmi. Non esitate quindi a mettere dei commenti ed a curare l'impaginazione. In primo luogo, conviene utilizzare una scrittura con tutti gli spazi separatori necessari.

IMPAGINAZIONE

L'abitudine di allineare a destra tutte le istruzioni Basic non e' assolutamente imposta dal linguaggio.
Niente impedisce di scalare le istruzioni e questo permette di mettere in evidenza dei livelli di allineamento che sono rappresentativi della logica del programma.

Problema: Con l'interprete Basic che io uso, tutti gli spazi che lascio all'inizio della linea (tra il numero di linea e la prima parola) spariscono. Quindi io non posso fare allineamenti.

Questo e' in realta' il comportamento della maggior parte degli interpreti Basic. Bisogna quindi ricorrere a qualche trucco. Con gli interpretatori della Microsoft basta iniziare la linea con ':', cosi':

```
10:      A=A+1
```

Questa regola vale anche per il CBM 8032.

Per una conveniente impaginazione si potranno utilizzare gli allineamenti che seguono.

```
   Istruzione
IF condizione THEN
   istruzione seguente
```

Un margine leggermente spostato verso destra va rispettato per le istruzioni in normale sequenza. Le REM e le IF si spostano leggermente verso sinistra. Esempio:

```
100 REM CALCOLO
110 :     A=5
120 IF A<B THEN X=2
130 :     Y=0


1000 IF NOT condizione THEN GOTO 1100
1005 REM
1010 REM SI condizione
1020 :   A=3
1030 :   B=8
1040 :   GOTO 1200
1050 REM
1100 REM NO condizione
1110 :   C=12
1120 REM
1200 ......
```

Si vede come, con un gioco di scalamenti, delle REM non seguite da parole, viene messa in luce la struttura logica

del programma.

Ovviamente le strutture possono essere concatenate e questo implica degli scalamenti supplementari per i livelli piu' interni; ne vediamo un esempio per i cicli ripetitivi.

```
1000 :     S=0:I=0
1010 REM RIPETERE
1020 :     :   I=I+1
1030 :     :   S=S+A(I)
1040 :     IF NOT I>=10 GOTO 1010
1050 .......
```

la ripetizione spaziata dei ':' nelle linee 1020 e 1030 rende piu' visibile il ciclo.
Per quanto riguarda il FOR..NEXT valgono gli stessi principi; il FOR e il NEXT sono allineati tra loro mentre tutte le istruzioni fondamentali del ciclo vengono spostate verso destra.

Noi pensiamo che gli esempi precedenti siano sufficienti per comprendere le regole da applicare per l'impaginazione. Non ci sono regole assolute, ma ognuno si comportera' come crede al fine di rendere leggibile il programma.

ESERCIZIO A.2. - Traducete in Basic strutturato il programma dell'esercizio 1.5., utilizzando un FOR per il ciclo piu' interno.

Quanto visto mostra che e' possibile imitare il Pascal con dei linguaggi non strutturati. Questo non impedisce al Pascal di essere il linguaggio con il quale la programmazione strutturata si realizza piu' naturalmente, e non gli toglie le altre specifiche qualita' che abbiamo analizzato in questo libro.

APPENDICE B

LE PAROLE CHIAVE DEL PASCAL

CARATTERI DI BASE

		Pag.
A...Z a...z	Lettere	22
0...9	Cifre	22
+ - * /	Addizione, sottrazione, moltiplicazione, divisione reale	22
AND OR NOT o (¬ V Λ)	Operatori logici	22
= ≠ (o <>) < > <= >=	Operatori relazionali	22
( ) [ ] (. .)	Parentesi	22
* * o { }	Indicatori di commenti	22
:=	Simbolo di assegnazione	31
'	Apostrofo	22
. , : ; ..	Punteggiatura	22
@ o ↑	Indicatore di puntatore	22

PAROLE CHIAVE DEL LINGUAGGIO

Le parole che seguono sono riservate , cioe' non si possono usare come indicatori.

	Pag.		Pag.
AND (e)	41	NIL (niente)	130
ARRAY (tabella)	61	NOT (no)	41
BEGIN (inizio)	15	OF (di)	51
CASE (casi)	51	OR (o)	41
CONST (costante)	17	PACKED (impaccato)	68
DIV (diviso)	32	PROCEDURE (procedura) .	87
DO (fai)	46	PROGRAM (programma) ...	38
DOWNTO (diminuito)	49	RECORD (struttura)	72
ELSE (se no)	43	REPEAT (ripeti)	46
END (fine)	15	SET (insieme)	75
FILE (file)	107	THEN (allora)	43
FOR (per)	48	TO (fino a)	48
FUNCTION (funzione)	86	TYPE (tipo)	17
GOTO (vai a)	52	UNTIL (fino a che)	46
IF (se)	43	VALUE (valore)*	153
IN (nel)	78	VAR (variabile)	18
LABEL (etichetta)	16	WHILE (mentre)	46
MOD (modulo)	32	WITH (con)	73

*) VALUE non e' considerata parola chiave in tutte le realizzazioni del Pascal. Al contrario, quasi tutte le nuove implementazioni ammettono FORWARD (piu' avanti - pag.xxx)

come parola chiave, permettendo di rimandare a dopo la definizione di una procedura.

INDICATORI STANDARD

Questa parte contiene i nomi di funzioni, procedure, tipi e costanti standard del sistema. Quelle contrassegnate da un + sono particolari di alcuni compilatori e rischiano di non essere definiti nel Pascal di cui voi disponete. Questi indicatori possono essere ridefiniti da voi nel vostro programma per l'uso che piu' vi conviene.

COSTANTI

			Pag.
FALSE		(falso)	40
TRUE		(vero)	40
MAXINT	+	(il piu' grande intero)	40

TIPI

INTEGER		(intero)	39
REAL		(reale)	40
BOOLEAN		(booleano)	40
CHAR		(carattere)	40
TEXT		(file di testo)	113
ALFA	+	(alfanumerico)	40

FILE STANDARD DEL SISTEMA

INPUT		(ingresso dati)	110
OUTPUT		(uscita dati)	110

FUNZIONI

ABS		(valore assoluto - reale va in reale, intero va in intero)	37
SQR		(quadrato - reale va in reale, intero va in intero)	37
ODD		(pari - intero va in booleano)	41
TRUNC		(parte intera con troncamento - reale va in intero)	37
ROUND	+	(arrotondato - reale va in intero)	37
SUCC		(successivo)	40
PRED		(precedente)	40
ORD		(ordinale)	40
CHR		(carattere di cui si da' l'ordinale)	40
SIN		(seno - angolo in radianti reale o intero va in intero)	36
COS		(coseno - angolo in radianti reale o intero va in intero)	36

ARCTAN (arcotangente - risultato in radianti) .. 36
EXP (esponenziale) 36
LN (logaritmo neperiano) 36
SQRT (radice quadrata) 36
EOF (fine file - bool. TRUE se END OF FILE) . 41
EOLN (fine linea - bool. TRUE se fine linea) . 41

PROCEDURE

PUT (scrive su file) 109
GET (legge da file) 109
RESET (rimette a inizio file) 109
REWRITE (riscrive) 108
READ (lettura) 26
READLN (lettura linea) 27
WRITE (scrittura) 28
WRITELN (scrittura linea) 29
NEW (allocazione puntatori) 128
DISPOSE + (liberazione puntatori) 129
MARK + (marcatura puntatori) 129
RELEASE + (liberazione puntatori) 129
PACK (impaccamento tabelle) 68
UNPACK (disimpaccamento tabelle) 68
PAGE + (va a nuova pagina) 31
DATE + (fornisce la data GG/MM/AA) 155
TIME + (ora) 155
HALT + (arresta il programma) 155

ESTENSIONI UCSD

TIPI

STRING Stringa di caratteri

PROCEDURE

INSERT Inserzione in una stringa
DELETE Cancellazione da una stringa
CONCAT Concatenazione di stringhe
COPY Accesso ad una parte di stringa

PROCEDURE DI ENTRATA E USCITA

BLOCKREAD Lettura di un blocco
BLOCKWRITE Scrittura di un blocco
SEEK Accesso diretto su disco
GOTOXY Invio del cursore in una determinata posizione sul video.

FUNZIONI

SIZEOF	Numero di byte assegnati a una variabile
IORESULT	Stato risultante da una operazione di ingresso o uscita.

ESTENSIONI CBM

CARATTERI

$	Introduce una costante esadecimale
_	Sottolineatura (SHIFT e $) e' ammesso negli identificatori
#	Inclusione di un pezzo di testo sorgente

FUNZIONI

PEEK	Lettura dalla memoria
GETKEY	Accettazione di un carattere da tastiera

FUNZIONI BINARIE

ANDB	AND logico
ORB	OR
XORB	OR esclusivo
NOTB	Contrario
SHL	Spostamento a sinistra
SHR	Spostamento a destra
IOERROR	Errore di entrata/uscita
RANDOM	Numero a caso
HOURS	Ore
MINUTES	Minuti
SECONDS	Secondi

PROCEDURE

POKE	Scrittura in memoria
ORIGIN	Fissa l'origine delle variabili dinamiche
VDU	Scrittura al video
WRHEX	Scrittura esadecimale
WRHEX2	" "
RDHEX	Lettura esadecimale
IOTRAP	Autorizza o inibisce i messaggi di errore
BREAKS	Attiva o inibisce il tasto STOP
SETTIME	Aggiorna l'orologio
CHAIN	Carica e chiama un programma in overlay.

APPENDICE C

DUE REALIZZAZIONI DEL PASCAL

IL PASCAL UCSD

La prima versione del Pascal e' stata realizzata a Zurigo da N. WIRTH; noi la chiameremo Pascal Zurich. Essa si presenta sotto forma di compilatore verso un linguaggio intermedio, che e' un Pascal semplificato chiamato linguaggio P.

Il compilatore e' anche scritto in linguaggio P. Per ottenere un Pascal su una qualunque macchina basta scrivere un interpretatore del linguaggio P sulla specifica macchina.

Il Pascal Zurich ha dato luogo a realizzazioni orientate in prevalenza verso i grossi calcolatori: CDC. Univac, Iris 80.

Alcuni ricercatori dell'Universita' di California di San Diego hanno creato un'altra versione del Pascal, piu' orientata verso i minicalcolatori e i microprocessori, la versione UCSD.

Come la versione precedente, essa si presenta sotto forma di compilatore verso il linguaggio intermedio P, che dopo basta interpretare.

Il linguaggio P UCSD assomiglia piu' del Zurich ad un linguaggio macchina e quindi risulta piu' facile da interpretare su un microprocessore.

Come abbiamo segnalato nel Cap. 9, esiste un microprocessore, il Pascal Microengine della Western Digital, nel quale il linguaggio macchina e' il linguaggio P-UCSD. Esso e' costruito con l'aiuto di 5 circuiti integrati: un operatore aritmetico, un controllore delle microistruzioni e 3 memorie ROM.

Passiamo ora a descrivere le principali differenze tra il Pascal UCSD e il Pascal Zurich.

RESTRIZIONI DEL PASCAL UCSD

Si hanno 3 restrizioni:

- (fastidioso per l'applicazione del paragr. 6.6.) manca la possibilita' di usare come argomenti FUNCTION o PROCEDURE;

- (corrisponde di fatto all'estensione delle dichiarazioni delle unita') non si dichiarano i file utilizzati nella testata PROGRAM, cosa che noi abbiamo sempre fatto;

- non esistono le procedure PACK, UNPACK e DISPOSE.

ESTENSIONI DEL PASCAL UCSD

Le estensioni del Pascal UCSD riguardano soprattutto i file (accesso diretto e file interattivi) e le stringhe di caratteri.

Noi non citiamo qui che i punti piu' importanti che definiscono le possibilita' supplementari apportate dalle estensioni. Per i dettagli piu' specifici il lettore leggera' il manuale del sistema che usa.

PROCEDURE E FUNZIONI CHE AGISCONO SUI FILE

Oltre alle procedure classiche che abbiamo visto (READ, GET, ecc.), si ha:

UNITBUSY (numero-unita') (Booleano): fornisce il valore TRUE se l'unita' specificata e' occupata. Sulla maggior parte dei sistemi e notoriamente sul Microengine i numeri standard delle unita' sono:

1: console (tastiera scrivente con eco del carattere)
2: sistema (tastiera scrivente senza eco del carattere)
4: disco sistema
6: stampante.

UNITCLEAR (numero-unita'): annulla tutte le operazioni sull'unita' specificata.

UNITREAD (numero-unita', tabella, lunghezza, numero-blocco, asinc). E' la routine di lettura di base. Si leggono i byte dati da 'lunghezza' in 'tabella' dall'unita' indicata, cominciando dal blocco indicato (indirizzo assoluto sul disco). Se asinc=1 il trasferimento e' asincrono; se asinc=0 esso e' sincrono.

UNITWRITE (numero-unita', tabella,lunghezza, numero-blocco, asinc). E' la routine di scrittura di base e usa gli stessi parametri della precedente di lettura.

UNITWAIT (numero-unita'): attende che il trasferimento in corso sull'unita' indicata sia terminato.

FILE SENZA TIPO

La dichiarazione VAR FILE : FILE; e' legale. Essa definisce un file senza specificare un tipo; a questo file si accede con le funzioni BLOCKREAD e BLOCKWRITE.

BLOCKREAD (FILE, TABELLA, numero blocchi da trasferire, numero-blocco).

BLOCKWRITE (FILE, TABELLA, numero blocchi da trasferire, numero-blocco).

Il valore fornito dalla funzione e' il numero di blocchi effettivamente trasferito. TABELLA e' una tabella che contiene le informazioni che devono essere trasferite. Numero-blocco e' l'indirizzo assoluto del blocco da cui iniziare il trasferimento.

IORESULT e' una funzione che fornisce il valore 0 se il trasferimento e' andato a buon fine, il numero dell'errore nel caso contrario.

FILE INTERATTIVI

La dichiarazione: VAR FILE : INTERACTIVE; definisce FILE come un file interattivo.
Il comportamento e' lo stesso che per il tipo TEXT salvo che RESET riferito ad un file interattivo fa riposizionare all'inizio del file senza domandare dati (vedere paragr. 7.6.).
Analogamente READ(F,X) equivale a GET (F); X:=F ; per un file interattivo (si ha una inversione).
Il file standard INPUT e' INTERACTIVE, altrimenti, dato che si ha per esso un RESET automatico all'inizio del programma, si avrebbe subito una richiesta di dati fastidiosa per l'operatore.

ACCESSO DIRETTO

Oltre a UNITREAD, UNITWRITE, BLOCKREAD e BLOCKWRITE che procurano un accesso diretto assoluto, si dispone di SEEK che procura un accesso diretto 'relativo' in un file. E' piu' prudente usare l'accesso relativo, dato che esso verifica che l'operazione avvenga nell'ambito del file desiderato. Gli accessi assoluti non hanno alcun tipo di verifica.

SEEK (FILE,N) ha per effetto che il prossimo GET o PUT avviene sulla registrazione N. La prima registrazione nel file viene riferita con N=0. Tra due SEEK ci vuole almeno

una operazione GET o PUT.

STRINGHE DI CARATTERI

VAR X : STRING; definisce X come stringa di caratteri di lunghezza fissa ad ogni assegnazione:

X:='BUONGIORNO'

La lunghezza massima e', se non dichiarata, di 80 caratteri. Ma si puo' superare questo massimo con una dichiarazione:

VAR Y : STRING [n]; dove n<=255.

Le stringhe di caratteri possono ricevere una assegnazione, si possono confrontare tra loro e possono essere lette o scritte. Se si cerca di leggere una stringa quando EOLN e' TRUE, si ottiene la stringa nulla (vuota).

FUNZIONI OPERANTI SULLE STRINGHE

CONCAT (X,Y,..,Z): produce una stringa che e' la concatenazione delle stringhe argomento.

COPY (STRINGA, INIZIO, N): produce una stringa formata da N caratteri prelevati da STRINGA a partire dalla posizione INIZIO.

LENGHT (STRINGA): fornisce la lunghezza (INTEGER) di STRINGA.

POS (MOTIVO, STRINGA): fornisce la posizione della prima occorrenza di MOTIVO in STRINGA, 0 se non trova MOTIVO.

DELETE (STRINGA, INIZIO, N): sopprime N caratteri di STRINGA a partire dalla posizione INIZIO.

INSERT (MOTIVO, STRINGA, INIZIO): inserisce MOTIVO in STRINGA a partire dalla posizione INIZIO.

Altre procedure agiscono su PACKED ARRAY OF CHAR. Esse sono: SCAL, FILLCHAR, MOVELEFT e MOVERIGHT. Non entriamo nei dettagli perche' la loro azione dipende dal sistema considerato.

SIZEOF (nome di variabile o di tipo): fornisce il numero di byte dedicati a quella variabile o al tipo.

VARIE

Alcuni Pascal UCSD hanno degli 'interi lunghi' dichiarati come segue:
VAR L : INTEGER [n]; dove n e' il numero di cifre decimali desiderato. Noi desidereremmo di poter avere l'analogo per i numeri reali.
Gli interi lunghi possono, in linea di principio, apparire dovunque possono apparire gli interi, ma non come argomenti di procedure, salvo che in STR e TRUNC.
La procedura STR (L,S) converte l'intero, o l'intero lungo L, nella stringa di caratteri S.

EXIT: questa istruzione permette, in caso di errore, di uscire da una procedura.

CASE: nel Pascal standard se il valore selezionato non corrisponde ad alcuno dei valori per i quali e' stato definito un appropriato trattamento, si ha errore. In UCSD, si passa alla istruzione seguente, senza segnalazione di errore, e questo puo' essere un difetto.

IL PASCAL CBM

Il Pascal del CBM e' un Pascal Zurich e di questo possiede tutte le caratteristiche, segnatamente la procedura DISPOSE. Esso accetta come argomenti di procedure nomi di procedure o di file.
Esso possiede qualche estensione propria, e qualche estensione che va nella direzione dell'UCSD. La sua procedura READ accetta di leggere un PACKED ARRAY [ ] OF CHAR; per questo e' facile dotarlo di procedure per la gestione di stringhe di caratteri.

Esso funziona in due modi: residente, con il compilatore in memoria e allora un programma puo' essere editato ed eseguito consecutivamente, e su disco. In quest'ultimo caso si deve editare un programma, salvare il testo sorgente (con PUT), poi compilarlo (comando COMP) e eseguirlo (comando EX). Il comando GET richiama il testo sorgente dalla memoria per le correzioni.

L'editor accetta tutte le istruzioni Basic in modo diretto piu' UPPER e LOWER che fanno passare da maiuscole a minuscole, AUTO (numerazione automatica), HEX e DECIMAL che operano le conversioni esadecimale-decimale, LOCATE che compila un programma e lo rende eseguibile in ambiente Basic, LINK che permette la fusione di piu' file-oggetto compilati separatamente.

ESTENSIONE DEL LINGUAGGIO

Il CBM ammette identificatori parlanti (mnemonici) come PREZZO-MEDIO, dove '-' e' il carattere ottenuto con SHIFT e $. Il CBM accetta costanti espresse in esadecimale, introdotte con $, come $FF00.

FILE E BUS IEEE

Le procedure RESET (F) e REWRITE (F) sono studiate per ammettere le seguenti forme:

RESET o REWRITE (F,'nome') aprono in lettura o scrittura un file permanente su disco, assegnandogli il nome indicato.

RESET o REWRITE (F,NP,AS,'nome') aprono un file sulle periferiche IEEE NP, con l'indirizzo secondario AS e il nome eventualmente indicato.

Per esempio dopo un REWRITE (OUTPUT,4,0), tutte le stampe effettuate con WRITELN vanno sulla stampante.

REWRITE (F,8,15,'comando') manda tutti i comandi voluti sul disco, fornendo un mezzo per fare l'accesso diretto.

LINGUAGGIO MACCHINA

Anche se questo e' contrario allo spirito del linguaggio, il Pascal del CBM permette di richiamare una procedura in linguaggio macchina che inizia all'indirizzo esadecimale xxxx. Basta dichiararla come: PROCEDURE NOME; EXTERN $xxxx;

Si dispone di funzioni o procedure PEEK (PEEK(X) = contenuto della locazione di memoria X), GETKEY:CHAR (GETKEY = carattere premuto sulla tastiera e rispone con CHR(0) se non viene premuto alcun tasto; WHILE GETKEY = CHR(0) DO; crea un attesa fino a quando viene premuto un tasto), ORIGIN (da evitare, fissa l'origine delle variabili dinamiche) e VDU (L,C,X) (scrive il carattere X alla linea L e alla colonna C del video).

FUNZIONI BINARIE

Un certo numero di funzioni effettuano delle operazioni binarie su interi di 16 bit: ANDB (X,Y) = X Y, ORB (X,Y) = X Y, XORB(X,Y) = X Y, NOTB (X) = X negato,SHL (X,n) = X sposta di n bit a sinistra, SHR = spostamento a destra.

PROCEDURE DI ENTRATA E USCITA

WRHEX e WRHEX2 scrivono in esadecimale. RDHEX legge in esadecimale. IOTRAP (FALSE) inibisce i messaggi di entrata/uscita, IOTRAP (TRUE) li attiva. IOERROR e' il numero dell'errore di entrata/uscita riscontrato (0=nessun errore). BREAKS (TRUE) attiva e BREAKS (FALSE) disattiva il tasto STOP.

ORA E NUMERI A CASO

La procedura SETTIME (H,M,S) predispone l'ora, i minuti ed i secondi. Le funzioni HOURS, MINUTES e SECONDS danno l'ora. RANDOM e' un numero a caso compreso tra 0 e 255. Esempio: RANDOM + (RANDOM MOD 128)*256 e' un numero a caso compreso tra 0 e 32767.

OVERLAY

Scrivendo: #nome-file, viene incluso nel testo sorgente Pascal il contenuto del file nome-file.

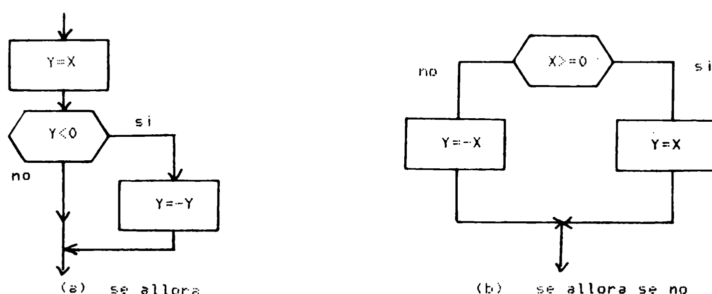
CHAIN (nome-file) fa caricare ed eseguire in overlay, cioe' andando a sostituire al programma chiamante in memoria il programma oggetto contenuto nel file indicato.

APPENDICE D

SOLUZIONI DEGLI ESERCIZI

Nota: Non potendo usare le parentesi graffe, i commenti sono inseriti nei programmi delimitandoli con asterischi.

ESERCIZIO 1.1.



(a) se allora (b) se allora se no

ESERCIZIO 1.2.

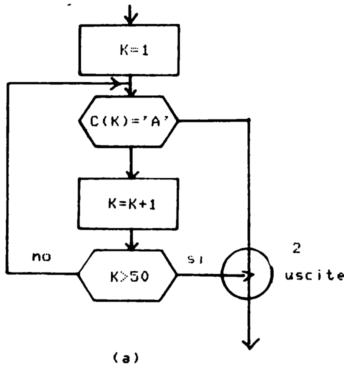
Fino a quando non si e' alla fine del file leggere la registrazione e stamparla, oppure: ripetere lettura registrazione, stampa fino alla fine del file.

ESERCIZIO 1.3.

```
I = 1;
ripetere stampare I; I = I+1 fino a quando I>10
```

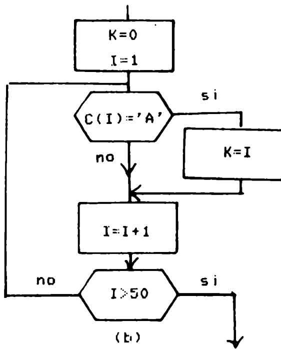
Qui si devono far vedere esplicitamente le operazioni relative all'indice corrente I (cosa che e' automatica con il FOR del Basic e il DO del Fortran), cosa che e' spesso fastidiosa per i principianti.

ESERCIZIO 1.4.



(a)

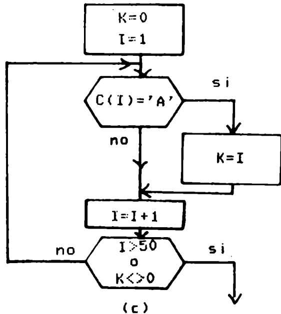
Il diagramma a blocchi riportato (a) presenta un ciclo dal quale si puo' uscire senza passare dalla condizione di arresto. Esso dunque non e' strutturato. Il diagramma a blocchi (b) invece e' strutturato. Esso usa una variabile in piu' e questo e' normale. Il rango K dove e' stato trovato un A non e' lo stesso del rango I dell'elemento corrente in esame. Se si hanno piu' A nella tabella, K conterra' il rango dell'ultimo. Questo non e' quello che e' richiesto dal testo. Viene richiesto di fermarsi quando si trova il primo A (quindi K<>0). Da cio' il diagramma a blocchi (c).



(b)

Il programma si scrivera':

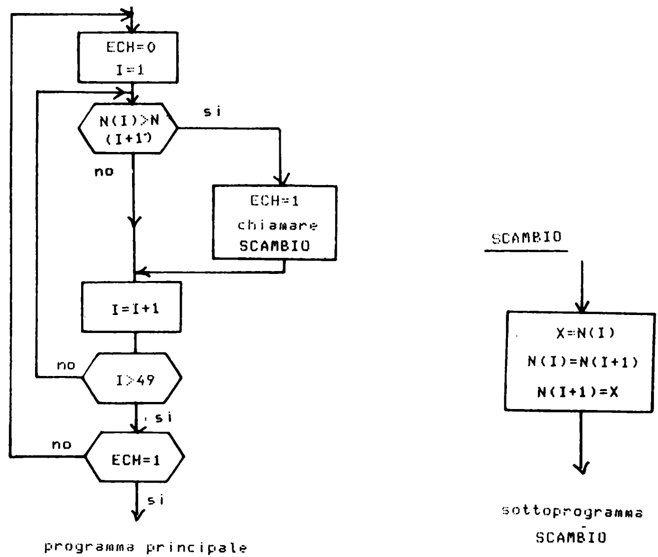
```
K=0
I=1
Ripetere
    se C(I)='A' allora K=I
    I=I+1
    fino a quando I>50 o K<>0
```



(c)

A ciascuno stadio (passaggio da (a) a (b) o da (b) a (c)), la programmazione strutturata ci ha obbligato ad analizzare meglio il nostro problema:
- stabilire che ci vogliono due variabili distinte K e I;
- trovare la giusta condizione di arresto.

ESERCIZIO 1.5.



programma principale

sottoprogramma SCAMBIO

```
REM ******************************************************
REM *              ORDINAMENTO ALFABETICO                *
REM *         N TABELLA DI 50 NOMI DA ORDINARE           *
REM *              ECH INDICATORE DI SCAMBIO             *
REM *     I, I+1 INDICI DELLE COPPIE DA CONFRONTARE      *
REM *            SOTTOPROGRAMMA CHIAMATO:SCAMBIO         *
REM ******************************************************
REM *
REM ** CICLO PER I PASSAGGI
    RIPETERE   ECH = 0
               I   = 1
          REM ** UN PASSAGGIO
              RIPETERE SE N(I)>N(I+1)
                       REM SE NON IN ORDINE
                           ALLORA PARTENZA ECH = 1
                                      CHIAMARE SCAMBIO
                                  FINE
                       I = I+1     ;REM VA ALLA COPPIA SEG.
               FINO A  I>49        ;REM FINE PASSAGGIO
    FINO A ECH=0       ;REM FINE DEL CICLO PASSAGGI
REM *
REM ******************************************************
REM *         SOTTOPROGRAMMA SCAMBIO ELEMENTI            *
REM *        POSIZIONE I E I+1 DELLA TABELLA N           *
REM *              X VARIABILE DI COMODO                 *
REM ******************************************************
REM *
    SOTTOPROGRAMMA SCAMBIO
        X   N(I)
        N(I)   N(I+1)
        N(I+1)    X
    RITORNO
```

ESERCIZIO 2.1.

```
*****************************************************
*              ORDINAMENTO ALFABETICO               *
*           N TABELLA DI 50 NOMI DA ORDINARE        *
*              ECH INDICATORE DI SCAMBIO            *
*      I, I+1 INDICI DELLE COPPIE DA CONFRONTARE    *
*             PROCEDURA CHIAMATA: SCAMBIO           *
*****************************************************
 PROGRAMMA ORDINAMENTO
 CONST NB = 50;     *numero nomi*
 TYPE STRINGA = ARRAY [1..20] OF CHAR;
 VAR ECH,I:INTEGER;
     N : ARRAY [1..NB] OF CHAIN;
 BEGIN                *inizio ordinamento*
   REPEAT             *ciclo dei passaggi*
      ECH := 0;
      I := 1;
      REPEAT          *un passaggio*
         IF N[I] > N [I+1]
        *se non in ordine*
            THEN   BEGIN
                     ECH := 1;
                     SCAMBIO
                   END;
         I := I+1    *coppia seguente*
       UNTIL I>49    *fine di un passaggio*
    UNTIL ECH=0      *fine ciclo dei passaggi*
  END.               *fine programma*
*******************************************************
*                  PROCEDURA SCAMBIO                  *
*         DEGLI ELEMENTI DI POSIZIONE I E I+1         *
*               X VARIABILE DI COMODO                 *
*******************************************************
 PROCEDURE SCAMBIO
 VAR X : STRINGA;
 BEGIN
    X := N [I];
    N [I] := N [I+1];
    N [I+1] := X
 END;   *fine scambio*
```

La stesura precedente e', di fatto, non corretta in un solo punto (abbiamo voluto seguire lo schema dell'esercizio 1.5.). Bisognerebbe che la procedura (tutto quello compreso nel tratteggio) venisse posta prima del BEGIN che inizia l'ordinamento. Infatti le procedure devono essere definite prima della prima istruzione eseguibile. Inoltre il programma, come riportato, e' sprovvisto di entrata e uscita dati.

ESERCIZIO 3.1.

```
A:=1   B:=2   C:=4   D:=5   E:=6
```

ESERCIZIO 3.2.

```
READ (A,B,C,D);
READLN;
READ (E,F,G,H);
```

oppure

```
READLN (A,B,C,D);
READ (E,F,G,H);
```

ESERCIZIO 3.3.

```
123123          123
                123
```

ESERCIZIO 3.4.

```
WRITELN ('LA SOMMA DI',A,'E',B,'RISULTA',A+B);
```

ESERCIZIO 3.5.

Incrementare (aumentare di 1) il valore di X.

ESERCIZIO 3.6.

No! Non si puo' avere piu' di una variabile a sinistra del segno :=, e non una espressione aritmetica. A+B non puo' essere un identificatore di variabile dato che e' presente il segno +.

ESERCIZIO 3.7.

```
X - X DIV Y*Y
```

ESERCIZIO 3.8.

Nessuno degli identificatori utilizzati e' molto mnemonico. ALBERTA e' dichiarata due volte. SOTTO e' dichiarata due volte. 1RIMAVAR non e' corretta perche' inizia con una cifra. ENRICO-QUARTO contiene un carattere speciale.

ESERCIZIO 3.9.

```
TRUNC(I/J)
```

ESERCIZIO 3.10.

```
Z:=EXP(Y*LN(X));
```

E' del tutto uguale a quello che succede in BASIC con: Z=X elevato Y, o in Fortran con: Z=X**Y.

ESERCIZIO 3.11.

```
X1:=(-B+SQRT(SQR(B)-4*A*C))/(2*A)
```

Il primo segno, e' quello chiamato 'meno unario'. Agisce solo sul termine che lo segue. Deve essere all'inizio di una espressione; A*-B non e' corretto e si deve scrivere A*(-B). Si raccomanda di usare le parentesi per evitare ambiguita'.

ESERCIZIO 3.12.

```
PROGRAM RAGGIOCERCHIO;
CONST PI=3.14159265;
VAR RAGGIO,S:REAL;
BEGIN
   READ (S);
   RAGGIO:=SQRT(S/PI);
   WRITELN(' RAGGIO = ',RAGGIO)
END.
```

ESERCIZIO 3.13.

Unica istruzione un po' nuova:

```
V:=4/3*PI*R*SQR(R)
```

ESERCIZIO 3.14.

```
PROGRAM CILINDRO;
CONST PI = 3.14159265;
VAR R,H,V :REAL;
BEGIN
   WRITELN (' RAGGIO  ALTEZZA');
   READ (R,H);
   V:=PI*SQR(R)*H;
   WRITELN ('VOLUME CILINDRO DI RAGGIO ',R,
        'E ALTEZZA ',H,' = ',V)
 END.
```

ESERCIZIO 3.15.

```
PROGRAM PROVA;
BEGIN
  WRITELN ('MASSIMO INTERO',MAXINT)
END.
```

Programma senza dichiarazioni.

ESERCIZIO 3.16.

```
IF (C>='0') AND (C<='9') THEN...
```

ESERCIZIO 3.17.

```
IF I MOD J = 0 THEN.....
```

ESERCIZIO 4.1.

```
IF X>0 THEN Y:=X;
       ELSE Y:=-X;
```

ESERCIZIO 4.2.

```
PROGRAM EQUAG1;         *equazione primo grado AX+B=0*
VAR  A,B:REAL;          *coefficienti*
       X:REAL;          *incognita*
BEGIN
   READ (A,B);
   IF A<>0 THEN BEGIN
                   X:=-B/A;
                   WRITELN ('RADICE: ',X)
                END;
           ELSE IF B=0 THEN
                          WRITELN (' INDETERMINATA ')
                        ELSE
                          WRITELN (' IMPOSSIBILE ')
 END.
```

ESERCIZIO 4.3.

```
IF a THEN IF b THEN I1
                ELSE
     ELSE I2;
I3;
```

oppure

```
IF a THEN BEGIN
               IF b THEN I1
            END
         ELSE I2;
I3;
```

ESERCIZIO 4.4.

```
PROGRAM MEDIA;
VAR S,N : REAL;
    CONT :INTEGER;
BEGIN
   S:=0; CONT=0;
   WHILE NOT EOF DO
                 BEGIN READLN(N);
                       S:=S+N;
                       CONT := CONT+1
                 END;
   WRITELN (' MEDIA ', S/CONT)
END.
```

ESERCIZIO 4.5.

```
PROGRAM MAXEMIN;
VAR MIN,MAX,N:REAL;
BEGIN
   MIN := 1.0E20;
   MAX := 0;
   WHILE NOT EOF DO
                 BEGIN READLN(N);
                       IF N<MIN THEN MIN:=N
                       IF N>MAX THEN MAX:=N
                 END;
   WRITELN ('MIN. = ',MIN,' MASS. = ',MAX)
END.
```

ESERCIZIO 4.6.

```
PROGRAM SENO;
CONST PI = 3.14159265;
VAR  X,X2:INTEGER;
     Y,Y2:REAL;
```

```
BEGIN
   PAGE;
   WRITELN ('-I--X--I---SIN-X---II--X--I---SIN-X---I');
   WRITELN ('----------------------------------------');
   FOR X:=0 TO 45 DO
       BEGIN
          X2:=X+46;
          Y :=SIN(PI*X/180);
          Y2:=SIN(PI*X2/180);
          WRITELN(' I',X:3,'  I',Y:9:5,'  II',X2:3,'  II'
                   ,Y2:9:3,'  I')
        END;
    WRITELN (' --------------------------------------');
END.
```

ESERCIZIO 4.7.

```
CASE c OF
     TRUE: i1;
     FALSE:i2;
END;
```

ESERCIZIO 4.8.

```
PROGRAM CONTAVOCALI;
   VAR C:CHAR;     *caratteri esaminati*
       NBVOCALI,NBCONSON:INTEGER;
BEGIN
   NBVOCALI :=0;
   NBCONSON :=0;
   PAGE;
   REPEAT
      READ(C);WRITE(C);
      IF (C='A')OR(C='E')OR(C='I')OR(C='O')OR(C='U')
         THEN NBVOCALI := NBVOCALI+1
         ELSE BEGIN
                IF C<>' ' THEN NBCONSON := NBCONSON+1
              END
   UNTIL C='.';
   WRITELN;WRITELN;
   WRITE(' CI SONO ',NBVOCALI,' VOCALI E ');
   WRITELN (NBCONSON-1,' CONSONANTI');
END.
```

Domanda: quali modifiche sarebbero necessarie se si usasse WHILE al posto di REPEAT?

- La modifica piu' vistosa sarebbe di dover mettere una lettura supplementare prima di WHILE. Infatti il controllo si fa su C e C deve essere letto, anche se esso resta nel

ciclo come lettura corrente, relativa all'iterazione successiva.

```
   READ(C);
   WHILE C<>'.' DO
        BEGIN
        READ(C)
        END;
```

Questa struttura e' piu' conforme alle regole di gestione dei file che comportano una lettura iniziale prima del ciclo ed una lettura corrente dentro il ciclo. Notate anche che nella prima versione si e' scritto NBCONSON-1 per ovviare al fatto che era stato contato il punto come consonante. Con WHILE questo non succede.

ESERCIZIO 4.9.

```
PROGRAM EQ2D;
   VAR  A,B,C :REAL;     *coefficienti*
        DELTA :REAL;     *discriminante*
        RE,IM :REAL;
BEGIN
  READ(A,B,C);
  IF (A=0) AND (B=0)
     THEN WRITELN (' EQUAZIONE DEGENERE')
     ELSE
       IF A=0
          THEN WRITELN (' RADICE SEMPLICE ',-C/B)
          ELSE
          IF C=0
             THEN WRITELN(' LE RADICI SONO ',-B/A,'E 0.0')
             ELSE
              BEGIN
               RE:=-B/(2*A);
               DELTA:=B*B-4*A*C;
               IM:=SQRT(ABS(DELTA))/(2*A);
               IF DELTA >0
                  THEN WRITELN (' LE RADICI SONO ',
                       RE+IM,' E ',RE-IM)
                   ELSE IF DELTA=0
                         THEN WRITELN(' RADICE DOPPIA',RE)
                         ELSE WRITELN(' RADICI COMPLESSE',
                           RE,'+I',IM,' E ',RE,'-I',IM)
               END;
END.
```

ESERCCIZIO 5.1.

FALSE

E' necessaria la dichiarazione: VAR D,M : GIORNO;
e naturalmente la definizione del tipo GIORNO come visto prima.

ESERCIZIO 5.2.

```
TYPE LETTERA = 'A'..'Z';
     CIFRA   = '0'..'9';
```

Attenzione: in alcune realizzazioni del Pascal si ha che 'A'<'B'<'C'...,ma i loro codici non sono consecutivi. In tali casi il tipo LETTERA puo' contenere degli intrusi. Il problema non si pone per le cifre che sono sicuramente rappresentate da codici consecutivi.

ESERCIZIO 5.3.

```
CONST INDICEMAX = 50;   *per esempio*
TYPE TABELLA = ARRAY [1..INDICEMAX] OF REAL;
VAR TAB : TABELLA;
```

Un elemento per scheda e un elemento per linea.

ESERCIZIO 5.4.

```
INIT:=-4;
FOR SCHEDA := 1 TO 10 DO
     BEGIN
       INIT := INIT+5;
       FOR I := INIT TO INIT+4 DO
            BEGIN
              READ (TAB [I] )
            END;
       READLN
     END;
INIT:=-9;
PAGE;
FOR LINEA := 1 TO 5 DO
     BEGIN
        INIT := INIT+10;
        FOR I := INIT TO INIT+9 DO
             BEGIN
                WRITE (TAB [I] : 10 : 4)
             END;
        WRITELN
      END;
```

ESERCIZIO 5.5.

```
      CONST NP1 = 11;     *1+delle dimensioni*
            X   = 'K';
      TYPE DIM = 1..NP1;
           PAROLA = ARRAY [DIM] OF CHAR;
      VAR TAB : PAROLA;
          IT  : INTEGER;
   BEGIN
      FOR IT:=1 TO NP1-1 DO
            BEGIN READ (TAB [IT] ) END;
      TAB [NP1]:=X;
      IT:=0;
      REPEAT
          IT:=IT+1
      UNTIL TAB [IT] = X;
    END.
```

e il ciclo contiene un solo controllo per UNTCL. Dopo, a seconda del valore di IT si sa se X era o meno presente. Non si e' mostrata la stampa dei risultati (sara' la stessa dell'esempio precedente).

ESERCIZIO 5.6.

```
  CONST N=20;
  VAR  MAX,I :INTEGER;
       TAB : ARRAY [1..N] OF INTEGER;
......
......
  MAX :=0;
  FOR I = 1 TO N DO
      BEGIN
         IF TAB [I] > MAX THEN MAX := TAB [I]
      END;
  WRITELN (MAX);
```

ESERCIZIO 5.7.

Metodo bolle : 42 comparazioni, 22 scambi;
metodo Shell : 34 comparazioni, 9 scambi.

Se il numero degli elementi e' notevole con il metodo Shell si guadagna in velocita'.

ESERCIZIO 5.8.

A:=PA non e' corretto dato che le due tabelle non sono dello stesso tipo; PA e' packed mentre A non lo e'.

ESERCIZIO 5.9.

L'esercizio e' gia' svolto nel testo.

ESERCIZIO 5.10.

```
PROGRAM TRANSPOSTA;
    CONST N=10;
    TYPE MATRICE = ARRAY [1..N,1..N] OF REAL;
    VAR A  : MATRICE;
        X  : REAL;        *per lo scambio*
        I,J:1..N;
BEGIN
    FOR I:=2 TO N DO
        FOR J:=1 TO I-1 DO
            BEGIN
               X:=A [I,J];
               A [I,J] := A [J,I];
               A [J,I] := X
            END
END.
```

ESERCIZIO 5.11.

```
TYPE ORDINE = RECORD
       ARTICOLO : PACKED ARRAY [1..20] OF CHAR;
       PU       : REAL;
       QUANT    : INTEGER;
       IMPORTO  : REAL
      END;
```

ESERCIZIO 5.12.

```
WITH IMPIEGATO  DO
 IF STAFAM='M'  THEN
                  WRITELN('DATAMATR',DATA)
                ELSE
                  WRITELN('NON SPOSATO')
 END;
```

ESERCIZIO 5.13.

```
   WRITELN('DATA MATRIMONIO',IMPIEGATO,DATAMATR,MM)
OPPURE:
   WITH IMPIEGATO,DATAMATR DO
   ....WRITELN ('...',MS)
```

ESERCIZIO 5.14.

```
   TYPE COLORE = (BLEU,BIANCO,ROSSO,GIALLO,VERDE,NERO);
        STOFFE = SET OF COLORE;
        STOFFA : STOFFE;
```

ESERCIZIO 5.15

```
  E3 vale [B1,B2,B3,B4] oppure [B1..B4]
```

ESERCIZIO 5.16.

```
  E3 vale [B2,B3,B4]
  B3 e' contato una sola volta
```

ESERCIZIO 5.17.

```
  (E1 <= E2) AND (E1 <> E2)
```

ESERCIZIO 6.1.

```
   FUNCTION POT (Y,X:REAL) : REAL;
       BEGIN
          POT := EXP(X*LN(Y))
       END;
```

ESERCIZIO 6.2.

```
   PROGRAM ISTOGRAMMA;
       VAR EFFETT : ARRAY [1..10] OF 1..100;
           I,K : INTEGER;
       PROCEDURE TRACCIA.....
           .....
           END;  *la procedura che abbiamo scritto*
     BEGIN
       FOR K := 1 TO 10 DO
           TRACCIA ('*',EFFETT [K]);
     END.
```

La traccia ottenuta ha la forma:

```
***
********
******
****
**
....
```

ESERCIZIO 6.3.

La modifica di N (o quella di K) avrebbe agito sul programma chiamante e avremmo avuto la stampa di:

```
     120
     cento asterischi
     100
e    1
     2
```

ESERCIZIO 6.4.

```
PROGRAM ISTOGRAMMA;
  VAR EFFETT : ARRAY [1..10] OF 1..100;
      I : INTEGER;
  PROCEDURE TRACCIA (C:CHAR;N:INTEGER);
      CONST MAX = 100;
      VAR I : INTEGER;
    BEGIN
      IF N>MAX THEN N:=MAX;
      FOR I:= 1 TO N DO
          WRITE(C);
      WRITELN
    END;
  BEGIN
     FOR I := 1 TO 10 DO
         TRACCIA ('*',EFFETT [I]);
 END.
```

ESERCIZIO 6.5.

```
PROGRAM MOLTMAT;
   CONST N=10;       *dimensioni matrici*
   TYPE MATRICE = ARRAY [1..N,1..N] OF REAL;
   VAR  A,B,C  : MATRICE;
   PROCEDURE LETMAT (VAR A:MATRICE);
       VAR I,J : INTEGER;
       BEGIN
       FOR I := 1 TO N DO
           BEGIN
```

```
            FOR J := 1 TO N DO
                READ (A [I,J]);
            READLN
          END
      END;
  PROCEDURE PRODMA (A,B:MATRICE;VAR C:MATRICE);
      VAR I,J,K :INTEGER;
      BEGIN
      FOR I := 1 TO N DO
          BEGIN
             FOR J := 1 TO N DO
                 BEGIN
                   C [I,J] := 0.0;
                   FOR K := 1 TO N DO
                       C [I,J] := C[I,J]+A[I,K]*B[K,J];
                 END
           END
       END;
  PROCEDURE STAMPA (A:MATRICE);
      VAR I,J : INTEGER;
      BEGIN
      FOR I := 1 TO N DO
          BEGIN
             FOR J := 1 TO N DO
                 WRITE (A[I,J]:8:2);
             WRITELN
          END
      END;
  BEGIN
      LETMAT(A);
      LETMAT(B);
      PRODMAT(A,B,C);
      PAGE;
      WRITELN ('RISULTATO '); WRITELN;
      STAMPA(C)
  END.
```

La costante N e il tipo MATRICE sono globali: conosciuti in tutti i programmi. Le matrici A,B,C definite in MOLTMAT sono conosciute in MOLTMAT. Sono altre versioni quelle conosciute nelle procedure. Le variabili I,J,K sono locali nelle loro procedure. Notate che qui si ovvia alla mancanza di dimensioni variabili del Pascal, per cambiarle basta cambiare il valore di N (ricompilando naturalmente).

ESERCIZIO 6.6.

```
PROGRAM CALCINTEG;
    CONST PI = 3.14159265;
    VAR N : INTEGER;
        S : REAL;
```

```
    PROCEDURE INTEG.....
    END;
  BEGIN
    N:=5;
    REPEAT
      INTEG (SIN,S,0.0,PI,N);
      WRITELN (' N= ',N,' S= ',S);
      N:=N+5
    UNTIL N>25
  END.
```

ESERCIZIO 6.7.

Eseguito nel testo.

ESERCIZIO 6.8.

Per INIT risulta facile: N e' l'altezza della torre, da cui FOR I:=1 TO 10 del ciclo di stampa diventa FOR I:=11-N TO 10 DO.

Per MOVIMENTO e' piu' difficile dato che N e' locale a TOGLIERE e non rappresenta l'altezza totale. E' meglio introdurre una variabile globale ALTEZZA definita nel programma principale HANOI (VAR ALTEZZA : INTEGER;), calcolata in INIT come ALTEZZA := N per cui il ciclo diventa:

FOR I := 11-ALTEZZA TO 10 DO.

ESERCIZIO 6.9.

```
PROGRAM ALEAT;
  FUNCTION RAND(R:INTEGER):REAL;FORWARD;
  PROCEDURE CHIAMA;
     VAR X,I :INTEGER;
     BEGIN
       X:=100;
       FOR I:=1 TO 100 DO
           WRITELN (RAND(X));
     END;
   FUNCTION RAND;
     CONST MODULO = 65536;
           MOLT = 25173;
           INC  = 13849;
     BEGIN
       RAND:=R/65535;
       R:=(MOLT*R+INC) MOD MODULO
     END;
   BEGIN
     CHIAMA
```

```
    END.
```

Questo programma e' stato provato su una macchina che tratta interi di 32 bit.

ESERCIZIO 6.10.

```
  PROGRAM CURVASENO;
     CONST PI = 3.14159265;
     FUNCTION F(X:REAL) : REAL;
        BEGIN
           F := 50+49*SIN(X)
        END;
     PROCEDURE CURVA (FUNCTION F:REAL;A,B:REAL);
        VAR X,H:REAL;
            I,Y :INTEGER;
     BEGIN
        H:=(B-A)/59;
        PAGE;
        X:=A;
        FOR I:=1 TO 60 DO
            BEGIN
               Y:=TRUNC (F(X));
               WRITELN (' ':Y, '*':1);
               X:=X+H
            END
        END;
     BEGIN      *programma principale*
        CURVA(F,0.0,2*PI)
     END.
```

ESERCIZIO 7.1.

```
  PROGRAM LISTAREAL (FILE1,OUTPUT);
     VAR FILE1:FILE OF REAL;
         NUMERO : REAL;
     BEGIN
       RESET (FILE1);
       REPEAT
         READ (FILE1,NUMERO);
         WRITELN (NUMERO)
       UNTIL EOF (FILE1)
     END.
```

Si vede come i file utilizzati sono dichiarati nell'istruzione PROGRAM; nella stessa e' dichiarato il file standard di OUTPUT.
In alcune versioni del Pascal non e' necessario dichiarare i file standard, in altre non si devono dichiarare.

ESERCIZIO 7.2.

```
  PROGRAM PIUPI(FILE1);
     CONST PI = 3.14159265;
     VAR FILE1:FILE OF REAL;
   BEGIN
      RESET (FILE1);
      REPEAT
         GET (FILE1)
      UNTIL EOF (FILE1);
      WRITE (FILE1,PI)
   END.
```

ESERCIZIO 7.3.

```
 PROGRAM AGGIORNA(INPUT,VECCHIO,NUOVO);
    TYPE STRING10 = PACKED ARRAY [1..19] OF CHAR;
         STRING30 = PACKED ARRAY [1..30] OF CHAR;
         PERSONA  = RECORD
                    NUM              :INTEGER;
                    COGNOME,NOME     :STRING10;
                    INDIR            :STRING30;
                    ASS              :INTEGER;
                    MANSIONE         :STRING10;
                    LIVELLO          :INTEGER
                    END;
          SCHEDA  = RECORD
                      CASE COD : CHAR OF
                        'L' : (NO : INTEGER);
                        'A','M' : (AGGIU:PERSONA);
                       END;
          VAR VECCHIO,NUOVO :FILE OF PERSONA;
              IMPIEGATO : PERSONA;
              MOV       : SCHEDA;
              MOVNUM    : INTEGER;
              M         : BOOLEAN;
 BEGIN
    RESET (VECCHIO); REWRITE (NUOVO);
    READ (VECCHIO,IMPIEGATO);
    READ (MOV);
    IF MOV.COD = 'L' THEN MOVNUM :=MOV.NO
                     ELSE MOVNUM :=MOV.AGGIU.NUM;
    REPEAT
       M := FALSE;
       IF MOVNUM>IMPIEGATO.NUM
            THEN BEGIN
                   WRITE (NUOVO,IMPIEGATO);
                   READ (VECCHIO,IMPIEGATO)
                 END;
```

```
            ELSE CASE MOV.COD OF
                 'L' : READ (VECCHIO,IMPIEGATO);
                 'A' : BEGIN
                         WRITE(NUOVO,MOV.AGGIU);
                         M := TRUE
                       END;
                 'M' : BEGIN
                         WRITE(NUOVO,MOV.AGGIU);
                         READ(VECCHIO,IMPIEGATO);
                         M:=TRUE
                       END;
              IF M THEN BEGIN
                   READ(MOV);
                   IF MOV.COD = 'L' THEN MOVNUM:=MOV.NO
                                    ELSE
                                    MOVNUM:=MOV.AGGIU.NUM
                         END
   UNTIL EOF *questa condizione basta tenuto conto della
             ipotesi semplificativa fatta*
END.
```

Si dovrebbe aggiungere il seguente controllo di errore: verificare che se MOVNUM<IMPIEGATO.NUM e quindi:
se 'M' o 'L' allora MOVNUM = IMPIEGATO.num.

ESERCIZIO 7.4.

```
   PROGRAM FATTURA;
      TYPE STRING10 = PACKED ARRAY [1..10] OF CHAR;
           STRING20 = PACKED ARRAY [1..30] OF CHAR;
           SCHEDA   = RECORD
                        CASE COD : CHAR OF
                          'C' : (NOME:STRING30;
                                 INDIR:STRING30);
                          'B' : (QT:INTEGER;
                                 ART:STRING30;
                                 PXU:REAL)
                      END;
      VAR COM : SCHEDA;
          Q   : INTEGER;
          ARTI: STRING30;
          PU,IMP,TOT :REAL;
   BEGIN
      PAGE;
      READ(COM);
      WRITELN(COM.NOME);
      WRITELN(COM.INDIR);
      TOT:=0.0;
      REPEAT
         READ(COM);
         IF COM.COD = 'B'
```

```
          THEN BEGIN
               WITH COM DO
                    BEGIN
                      Q:=QT;ARTI:=ART;PU:=PXU
                    END;
               IMP:=PU*Q;
               TOT:=TOT+IMP;
               WRITELN(Q:5,ARTI,35,'A',PU:10:2,IMP:15:2)
               END;
           ELSE BEGIN
                  WRITELN(' ':50,'TOTALE',TOT:15:2);
                  PAGE;
                  WRITELN(COM.NOME);
                  WRITELN(COM.INDIR);
                  TOT:=0.0
                END
   UNTIL EOF;
   WRITELN (' ':50,'TOTALE',TOT:15:2)
END.
```

Per questo esercizio e il precedente riguardare nota del paragrafo 7.9..

ESERCIZIO 7.5.

```
   PROGRAM COPY;
      VAR C : CHAR;
   BEGIN
      WHILE NOT EOF DO
          BEGIN
            WHILE NOT EOLN DO
               BEGIN
                  READ(C);
                  WRITE(C)
                END;      *fine linea*
             READLN;
             WRITELN      *si passa alla linea seguente*
           END
   END.
```

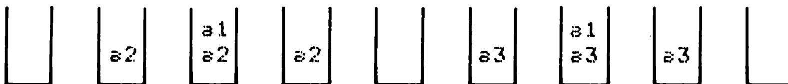
ESERCIZIO 7.6.

```
   PROGRAM FREQLETT;
      VAR X:CHAR;
          FREQ:ARRAY ['A'..'Z'] OF INTEGER;
               LETT:SET OF 'A'..'Z';
      BEGIN
         LETT:= ['A'..'Z'];
         FOR X:= 'A' TO 'Z' DO FREQ [X]:=0;
         WHILE NOT EOF DO
```

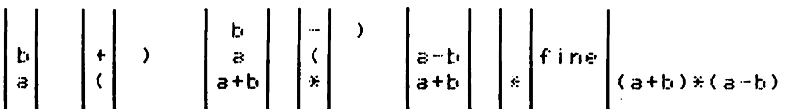
```
                BEGIN
                   WHILE NOT EOLN DO
                      BEGIN
                        READ(X);
                        IF X IN LETT THEN FREQ[X]:=FREQ[X]+1;
                      END;
                      READLN
                 END
    END.

    PROGRAM FREQCOPPIE;
       VAR X,Y : CHAR;
           FCOPPIE : ARRAY['A'..'Z','A'..'Z'] OF INTEGER;
           LETT : SET OF 'A'..'Z';
    BEGIN
       LETT := ['A'..'Z'];
       FOR X:='A' TO 'Z' DO
           BEGIN
              FOR Y:='A' TO 'Z' DO FCOPPIE[X,Y]:=0;
           END;
       READ(X);
       WHILE NOT EOF DO
           BEGIN
             WHILE NOT EOLN DO
                BEGIN
                   READ(Y);
                   IF(X IN LETT) AND (Y IN LETT)
                      THEN BEGIN
                           FCOPPIE[X,Y]:=FCOPPIE[X,Y]+1;
                           X:=Y
                           END
                 END;
                 READLN
            END
    END.
```

ESERCIZIO 8.1.



ESERCIZIO 8.2.



ESERCIZIO 8.3.

```
PROGRAM HANOI2;
   VAR NP,IP,JP : ARRAY [1..50] OF INTEGER;
   P : INTEGER;    *puntatore della pila*
   N,I,J : INTEGER;

 PROCEDURE TOGLI;
   BEGIN
     N:=NP[P];I:=IP[P];J:=JP[P];
     IF N=1 THEN WRITELN (I,'=>',J)
            ELSE BEGIN
                   P:=P+1;
                   NP[P]:=N-1; IP[P]:=I;
                   JP[P]:=6-I-J;
                   TOGLI;

                   P:=P+1;
                   NP[P]:=1;IP[P]:=I;JP[P]:=J;
                   TOGLI;

                   P:=P+1;
                   NP[P]:=N-1;IP[P]:=6-I-J;JP[P]:=J;
                   TOGLI
                 END;
     P:=P-1;N:=NP[P];I:=IP[P];J:=JP[P]
   END;
 BEGIN
   READ(N);
   P:=1;NP[P]:=N;IP[P]:=1;JP[P]:=2;
   TOGLI
END.
```

E in Basic:

```
 10 DIM NP(50),IP(50),JP(50)
 20 INPUT "NUM. DISCHI ";N
 30 P=1:NP(P)=N:IP(P)=1:JP(P)=2
 40 GOSUB 100
 50 END
100 N=NP(P):I=IP(P):J=JP(P)
110 IF N=1 THEN PRINT I;"=";J:GOTO 200
120 P=P+1:NP(P)=N-1:IP(P)=I:JP(P)=6-I-J
130 GOSUB 100
140 P=P+1:NP(P)=1:IP(P)=I:JP(P)=J
150 GOSUB 100
160 P=P+1:NP(P)=N-1:IP(P)=6-I-J:JP(P)=J
170 GOSUB 100
200 P=P-1:N=NP(P):I=IP(P):J=JP(P)
210 RETURN
```

ESERCIZIO 8.4.

```
Mettere: VIDE:=FALSE;
         FILA[P2]:=ELEMENTO;
         IF (P2>P1) AND (P2=MAX)        *non si aggiungono
            THEN P2:=1                   elementi fino a
            ELSE P2:=P2+1;               quando PIENO resta
         IF P2=P1 THEN PIENO:=TRUE;      vero*
Togliere: PIENO:=FALSE;
          ELEMENTO:=FILA[P1];           *non si tolgono ele=
          IF P1=MAX THEN P1:=1           menti fino a quando
                    ELSE P1:=P1+1;       VUOTO resta vero*
          IF P1=P2  THEN VUOTO:=TRUE
```

ESERCIZIO 8.5.

```
   PROGRAM TOGLI5;
   *dichiarazioni dell'esempio precedente*
   *continuazione dichiaraz. VAR*
         I,PREC,P : INTEGER;
      BEGIN
        P:=PRIMO;
        FOR I := 1 TO 5 DO
            BEGIN
               PREC:=P;
               P:=MALISTA[P].PUNTATORE
            END;
        MALISTA[PREC].PUNTATORE:=P
    END;
```

ESERCIZIO 8.6.

```
   PROCEDURE INSER(INFO,tipo voluto,I:INTEGER);
      VAR K,PREC,P :INTEGER;
          PIENO    :BOOLEAN;
     BEGIN
      P:=PRIMO;
      FOR K:=1 TO I DO
          BEGIN
            PREC:=P;
            P:=MALISTA[P].PUNTATORE
          END;
      MALISTA[PREC].PUNTATORE:=LIBERO;
      MALISTA[LIBERO].PUNTATORE:=P;
      MALISTA[LIBERO].INFORMA:=INFO;
      LIBERO:=LIBERO+1;
      IF LIBERO=MAX THEN PIENO:=TRUE;
     END;
```

Le dichiarazioni che definiscono la lista sono nel programma chiamante, e questo rende le variabili globali. La procedura non deve essere chiamata se PIENO e' vero:

```
 IF NOT PIENO THEN INSER (INFO,5);  *per esempio*
```

ESERCIZIO 8.8.

1	2	3	4	5	6	7	8	9
A | 3	B | 0	B | 6	C | 0	D | 0	C | 9	B | 0	E | 0	D | 10

10	11
E | 12	D | 0

LIBERO = 12

ESERCIZIO 8.9.

Ci limitiamo alle istruzioni essenziali.

```
  I:=1;
  WHILE FAMIGLIA[I].NOME<>'GIANNI' DO I:=I+1;
  I:=FAMIGLIA[I].PADRE;
  WRITELN ('PADRE DI GIANNI: ',FAMIGLIA[I].NOME);
```

ESERCIZIO 8.10,

```
   PROGRAM LETTLISTA;
   .....*dichiarazioni viste all'inizio del paragrafo*
      *cont. VAR* PREC : PTR;
      BEGIN
        NEW (P);
        READ(P@.INFORM);
        PRIMO:=P;
        PREC:=P;
        READ(X);
        WHILE NOT EOF DO
            BEGIN
               NEW (P);
               PREC@.SEGUENTE:=P;
               PREC:=P;
               P@.INFORM
               READ(X)
            END;
         PREC@.SEGUENTE:=NIL
   END.
```

ESERCIZIO 8.11.

Basta che l'ultimo puntatore punti verso il primo elemento. Nell'esercizio 8.10. basta scrivere prima dell'ultima linea:

```
PREC@.SEGUENTE := PRIMO
```

La lista circolare puo' essere molto utile per trattare problemi di file di attesa, ne abbiamo gia' parlato nel paragrafo 8.3.. Basta avere due puntatori esterni PRIMO e ULTIMO. PRIMO punta verso l'elemento di testa della fila di attesa, quello da togliere per trattare un elemento. Ultimo punta verso il posto dove verra' aggiunto un elemento che entrera' nella fila.

ESSERCIZIO 8.12.

```
NEW (Q);
Q@.INFORM := 'LULU';
P:=PRIMO;
IF P@.INFORM = 'JOJO'
   THEN PRIMO:=Q
   ELSE BEGIN
          WHILE P@.INFORM <> 'JOJO' DO
            BEGIN
              PREC:=P;
              P:=P@.SEGUENTE
            END;
           PREC@.SEGUENTE:=Q
          END;
 Q@.SEGUENTE:=P;
```

ESERCIZIO 8.13.

```
PROGRAM LISTINV;
   TYPE PTR= @ELEM;
        ELEM = RECORD
                  INFO :TYPE;
                  SEG,PREC :PTR
               END;
    VAR  P,PREC,
         PRIMO,ULTIMO :PTR;
  BEGIN
    PRIMO@.PREC:=NIL;
    P:=PRIMO@.SEG;
    PREC:=PRIMO;
    WHILE P<>NIL DO
      BEGIN
```

```
        P@.PREC:=PREC;
        PREC:=P;
        P:=P@.SEG
      END;
    ULTIMO:=PREC
  END.
```

ESERCIZIO 8.14.

```
  FOR I:=1 TO N DO
      BEGIN
        X:=PT[I];
        WRITELN(X@)
      END.
```

Le frasi di scrittura possono essere modificate a seconda delle esigenze di impaginazione dei risultati. Una volta eseguito il programma, non si puo' ritrovare l'ordine iniziale. Per ottenere questo si sarebbe dovuta mantenere una copia della tabella iniziale. Un altro modo per ottenere questo sarebbe quello di servirsi di puntatori memorizzati insieme agli altri dati. Questi puntatori, non venendo modificati, servirebbero per ricordare l'ordine iniziale.

ESERCIZO 8.15.

No: N+1. Infatti si ha un NEW (X) che corrisponde alla lettura del fine file.

ESERCIZIO A.1.

Si potrebbe pensare di definire un nuovo tipo:

```
  TYPE MEZZOINTERO = (1.0,1.5,2.0,2.5,3.0)
  VAR X : MEZZOINTERO
  FOR X := 1.0 TO 3.0 DO
```

ma questo non puo' funzionare con la maggior parte delle implementazioni del Pascal, infatti creerebbe delle costanti come 1.0 o 1.5 che apparterrebbero a due tipi (mezzointero e reale), e questo non e' consentito.

ESERCIZIO A.2.

```
1000 REM ****************************
1010 REM *  ORDINAMENTO ALFABETICO    *
1020 REM *  N$ TABELLA DI 50 NOMI     *
1030 REM *  SCA INDICATORE DI SCAMBIO *
1040 REM *  I,I+1 INDICI DELLA COPPIA *
1050 REM *  ESAMINATA                 *
1060 REM *  SOTTOPROGRAMMA SCAMBIO    *
1065 REM *  IN 2000                   *
1070 REM ****************************
1080 REM
1090 :         DIM N$(50)
1100 REM RIPETIZIONE CICLO DEI PASSI
1110 :   :      SCA=0
1120 :   :      FOR I=1 TO 49  : REM UN PASSO
1130 :   :      :   IF N$(I)<=N$(I+1) GOTO 1170
1140 :   :      :   REM SE NON VERO
1150 :   :      :   SCA=1
1160 :   :      :   GOSUB 2000   : REM SCAMBIO
1170 :   :      NEXT I  :REM FINE DI UN PASSO
1180 :   IF SCA=1 GOTO 1100
1190 :   REM FINE DEL CICLO DEI PASSI
1200 END : REM FINE PROGRAMMA PRINCIPALE
2000 REM*
2010 REM ******************************
2020 REM * SOTTOPROGRAMMA SCAMBIO      *
2030 REM * X$ VARIABILE DI COMODO      *
2040 REM * N$(I) E N$(I+1) ELEMENTI TAB.*
2050 REM ******************************
2060 REM *
2070 :      X$=N$(I)
2080 :      N$(I)=N$(I+1)
2090 :      N$(I+1)=X$
2100 RETURN
```

Potete vedere che si ha una bella somiglianza con il Pascal. Potete provare a scrivere lo stesso programma in Basic non strutturato e vedrete che occupa meno memoria. La chiarezza di questa versione vale la spesa del maggior consumo di memoria. Se volete provare il programma, dovrete aggiungere le istruzioni di ingresso e uscita.

APPENDICE E

BIBLIOGRAFIA

Oltre ai manuali che descrivono le implementazioni del Pascal sul vostro calcolatore, alle riviste specializzate che riportano diversi articoli sul Pascal, potete consultare i seguenti libri:

In francese:

N.WIRTH - Introduction a' la programmation systematique - Masson 1977

In inglese:

B.W.LIFFICK Ed. - The byte book of Pascal - Byte publ. 1979

P.GROGONO - Programming in Pascal - Addison Wesley 1978

I.R.WILSON, A.M.ADDYMAN - A practical introduction to Pascal - Springer 1978

K.JENSEN, N.WIRTH - Pascal - User manual and report - Springer 1975

N.WIRTH - Algorithms + Data structures = Programs - Prentice Hall 1975

Articoli:
Segnaliamo: "Pascal News", la rivista del Pascal User Group c/o Computer Studies Group - Mathematics Department - The University - SOUTHAMPTON S09 5NH (Great Britain),.

In inglese:
M.N.CONDICT: The Pascal dynamic array controversy and a method for enforcing global assertions - ACM-SIGPLAN 12.11,23 (1977).

R.CONRADI: Further critical comments on Pascal, particularly as a system programming language - ACM-SIGPLAN 11.11,8 (1976).

E.W.DIJKSTRA: GOTO statement considered harmful - com. ACM 11.3.147 (1968) (Atto di nascita della programmazione strutturata!).

A.N.HABERMANN: Critical comments on the programming language Pascal - ACTA INFORMATION 3.1.45 (1973).

E.N.KITTLITZ: Another proposal for variable size array in Pascal - ACM-SIGPLAN 12.1,82 (1977).

O.LECARME: Structured programming, programming teaching and the language Pascal - ACM-SIGPLAN 9.7,15 (1974).

O.LECARME, P.DESJARDINS: Reply to a paper by A.N.HABERMANN on the programming language Pascal - ACM-SIGPLAN 9.10,21 (1974).

O.LECARME, P.DESJARDINS: More comments on the programming language Pascal - ACTA INFORMATICA 4,231 (1975).

H.F.LEDGARD, M.MARCOTTY: A genealogy of control structures - com. ACM 18.11,629 (1975).

B.J.Mc LENNAN: Note on dynamic array in Pascal - ACM-SIGPLAN 10.9,39 (1975).

J.WELSH, W.J.SNEERINGER, C.A.R.HOARE: Ambiguities and insecurities in Pascal - SOFTWARE PRACTICE AND EXPERIENCE - 7, 685 (1977).

N.WIRTH: Design of a Pascal compiler - SOFTWARE PRACTICE AND EXPERIENCE - 1, 309 (1971)

MN.WIRTH: Comment on a note on dynamic arrays in Pascal - ACM-SIGPLAN 11.1,37 (1976).

In francese:
B.LANG: Le language Pascal - Serie di artiroli in MICROSYSTEMES - 7,98 (1979); 10,91; 11,61 (1980).

C.DISABEAU: Presentation de Pascal - L'ORDINATEUR INDIVIDUEL 7,53 (1979).

A.ALABAU, J.FIGUERAS, S.PINCON: Pascal et les ordinateurs individuels - L'ORDINATEUR INDIVIDUEL 13,60 (1979).

APPENDICE F

INDICE DEI PROGRAMMI

Superficie di un cerchio.................................... 39

Calcolo del minimo e del massimo............... 47, 172, 176

Media.. 47, 49, 172

Numeri primi.. 50

Tavola della funzione seno...................... 50, 172

Conteggio delle vocali di una parola............. 53, 173

Equazione di secondo grado....................... 53, 174

Ricerca di una lettera in una parola............. 65, 176

Ricerca di un elemento (metodo dicotomico)........ 66, 176

Ordinamento (Shell).............................. 67, 176

Prodotto di matrici......................... 70, 94, 179

Matrice trasposta................................ 71, 177

Lettura di un insieme................................. 79

Numeri primi con il crivello di Eratostene............ 79

Istogrammi....................................... 87, 178

Lettura di una matrice................................ 90

Integrale di una funzione......................... 95, 180

Fattoriale.. 96, 97

Torre di Hanoi.................................... 98, 181

Numeri a caso.................................... 105, 181

Grafico di una curva............................. 106, 182

Stampa di un file................................ 110, 182

Aggiornamento.............................. 111, 183

Fatturazione............................... 112, 184

Copia di un testo.......................... 114, 185

Frequenza di lettere e di digrammi................. 114, 185

Uso della pila.............................. 119

Gestione della pila........................ 120, 187

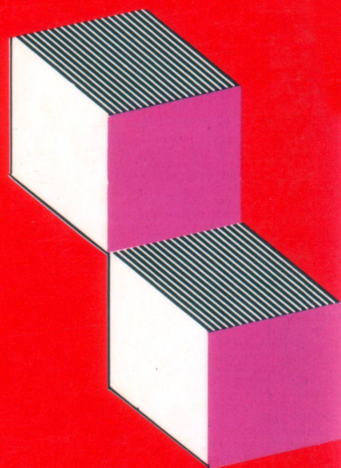
Creazione di una lista.................. 130, 131, 189

Percorso in una lista........................ 131

Cancellazione e inserzione................ 131, 132

Lista doppia............................... 132, 190

Ordinamento con dati dinamici.............. 133, 191



Lo scopo del libro è di fare il punto sul Pascal. Per i lettori che già possiedono un personal computer e che sanno programmare in BASIC, è importante poter individuare i vantaggi del Pascal sul BASIC, e se questi vantaggi giustificano o creano l'acquisto di un interprete o di un compilatore Pascal.
Per coloro invece che stanno decidendo l'acquisto di un sistema, la questione è ancora più importante. Devono infatti sapere se il Pascal è assolutamente necessario per le applicazioni che desiderano realizzare e se devono per forza scegliere un calcolatore che disponga del Pascal.
I vantaggi del Pascal che emergono (e sono numerosi) vengono descritti nel contesto delle applicazioni in cui sono realizzati. Per contro vengono anche indicate le situazioni dove questi non sono indispensabili.
Non si poteva discutere di Pascal senza descrivere la dottrina da cui il linguaggio ha avuto origine: la *programmazione strutturata*, ed è da questa che parte la trattazione dell'argomento, per passare nella parte centrale dell'opera, allo studio vero e proprio del Pascal, e concludersi con l'analisi dei tipi di dati che il Pascal può trattare.

L.14.000

Cod. 514A ISBN 88-7056-116-X

GRUPPO
EDITORIALE
JACKSON

PROGRAMMARE IN PASCAL

Daniel-Jean David
e Jean-Luc Deschamps

59