

<i>I LINGUAGGI DI PROGRAMMAZIONE DEGLI ELABORATORI ELETTRONICI</i>	2
<i>GENERALITÀ SULLA STESURA DI UN PROGRAMMA</i>	2
<i>PROGRAMMI INTERPRETATI</i>	2
<i>PROGRAMMI COMPILATI</i>	3
<i>CENNI SUL SOFTWARE DI BASE DI UN PC</i>	3
<i>CENNI SUI PACCHETTI APPLICATIVI</i>	4
<i>LE CARTE DI FLUSSO (FLOW CHART)</i>	6
<i>Il problema della protezione del calcolo</i>	7
<i>LE STRUTTURE DECISIONALI DEL TURBO PASCAL</i>	8
<i>Strutture If Then e If Then Else</i>	8
<i>Osservazione:</i>	9
<i>Struttura decisionale Case ... of</i>	9
Esempio 1.....	9
<i>STRUTTURE ITERATIVE DEL TURBO PASCAL</i>	10
- <i>i loop infiniti</i>	10
- <i>REPEAT... UNTIL:</i>	10
Esempio 2.....	10
Esempio 3.....	11
- <i>WHILE...DO,</i>	11
Esempio 4.....	11
Esempio 5.....	11
- <i>FOR...DO,</i>	12
<i>Osservazioni</i>	12
<i>ESEMPI DI PROGRAMMAZIONE</i>	13
<i>Media aritmetica fra N valori</i>	13
<i>Risoluzione di una equazione di secondo grado</i>	14
<i>Conversione da Coordinate Cartesiane a Coordinate Polari</i>	16
<i>Determinazione del Baricentro di un sistema di forze</i>	19
<i>Equazione della retta passante per due punti</i>	19
<i>Algoritmo di Euclide per il calcolo del M.C.D. (massimo comune divisore)</i>	20
<i>Algoritmo di Euclide per il calcolo del m.c.m. (minimo comune multiplo)</i>	22

Tecniche di programmazione nei sistemi di elaborazione dati

prof. Cleto Azzani
 IPSIA Moretto di Brescia
 Brescia 1993 (rev Ottobre 1995)

I linguaggi di programmazione degli elaboratori elettronici

La programmazione degli elaboratori può essere attuata facendo uso del linguaggio macchina oppure di linguaggi di programmazione detti ad alto livello.

Il linguaggio macchina è un linguaggio specifico di una determinata CPU e quindi presuppone una buona conoscenza del set di istruzioni del modo di indirizzamento, delle modalità di esecuzione delle istruzioni da parte della CPU e dell'architettura del sistema utilizzato.

I linguaggi ad alto livello sono completamente indipendenti dal tipo di CPU su cui operano, sono molto più aderenti al modo di esprimersi di un essere umano e non richiedono una conoscenza specifica dell'architettura del sistema su cui viene redatto un determinato programma.

Tra i linguaggi ad alto livello ricordiamo :

- *FORTRAN* (FORmulae TRANslator) adatto per la risoluzione di problemi di tipo scientifico.
- *COBOL* (COmmon Business Oriented Language) adatto per la risoluzione di problematiche di tipo commerciale.
- *PL/1* adatto per applicazioni sia di carattere scientifico che commerciale (nato in casa IBM) primo esempio di linguaggio strutturato (MPL PL-Z).
- *BASIC* (Beginner's All-purpose Symbolic Instruction Code) linguaggio adatto per un primo impatto con la realtà del calcolatore: risolve sia problemi di tipo scientifico che problemi di carattere commerciale.
- *PASCAL* linguaggio adatto a risolvere sia problemi di tipo scientifico che problemi di altro genere. È un linguaggio strutturato per definizione.
- *C* linguaggio derivato dal PASCAL ed utilizzato per la scrittura di sistemi operativi o di altri strumenti software.

Generalità sulla stesura di un programma

Il programma scritto dal programmatore ed introdotto nella memoria dell'elaboratore viene detto programma SORGENTE (SOURCE-PROGRAM); quello materialmente eseguito dalla CPU e perciò codificato in linguaggio macchina viene denominato programma OGGETTO (OBJECT-PROGRAM). Sarà perciò necessario disporre di un traduttore che partendo da istruzioni redatte in linguaggio ad alto livello le trasformi in equivalenti istruzioni stese in linguaggio macchina. Ogni linguaggio dispone quindi di un traduttore in linguaggio macchina.

Un generico linguaggio ad alto livello può essere strutturato per funzionare in modo INTERPRETE o in modo COMPILATORE.

Nel modo INTERPRETE il programma sorgente si trova in memoria RAM e viene introdotto attraverso una tastiera che di solito opera in codice ASCII (American Standard Code for Information Interchange); quando viene attivata la esecuzione del programma l'interprete per ciascuna riga di programma effettua le seguenti operazioni:

- 1)- *Controllo sintattico sulla riga di programma.*
- 2)- *Traduzione della riga di programma in linguaggio macchina.*
- 3)- *Esecuzione di quella riga di programma.*

La fase 1) di controllo sintattico può dare origine a segnalazioni di errore che appaiono su terminale video con indicazione della riga di programma ove si è verificato l'errore e del tipo di errore riscontrato (codifica numerica) *SYNTAX-ERROR*. Per correggere l'errore è necessario riscrivere la riga di programma errata.

La fase 2) di traduzione in linguaggio macchina, può dare origine a segnalazioni di errore di tipo *COMPILATION-ERROR* per correggere le quali è necessario sempre riscrivere una o più righe errate di programma.

La fase 3) di esecuzione del segmento di programma oggetto anch'essa può dare origine a segnalazioni di errore di tipo *RUN-TIME-ERROR* per correggere le quali bisogna analizzare con attenzione qual'è la condizione che ha provocato l'errore.

Nel modo *COMPILATORE* il programma sorgente si trova di solito su disco sotto forma di "file" (archivio di informazioni su disco); la preparazione del sorgente su disco avviene attraverso l'uso di un apposito "strumento software" denominato *EDITORE*.

Il compilatore fa una analisi sintattica di tutto il programma sorgente fornendo eventuali segnalazioni di errore che servono al programmatore per correggere il programma. In caso il controllo sintattico abbia dato esito positivo, viene effettuata la traduzione in linguaggio macchina del programma sorgente, traduzione che viene registrata su disco sotto forma di "file".

Il programma oggetto così prodotto dal compilatore in genere non è ancora eseguibile in quanto privo delle necessarie informazioni base per eseguire operazioni matematiche più o meno complesse o operazioni di Input Output. Si rende necessaria un'ultima operazione denominata operazione di LINK delle librerie che viene attuata per mezzo di un altro "strumento software" chiamato *LINKING-LOADER*. Il programma oggetto prodotto dal compilatore, "linkato" con le routine di libreria è quindi in grado di funzionare autonomamente; anch'esso viene salvato su disco e costituisce il prodotto finale e conclusivo della compilazione.

Programmi interpretati

L'impiego di un linguaggio interprete presenta pregi e difetti che di seguito vengono elencati:

- Il programma viene modificato in modo molto semplice (riscrivendo la riga di programma errata) e in modo altrettanto semplice è possibile verificare la validità della correzione.
- Il programma sorgente è sempre presente in RAM quindi è di solito "listabile" su video o stampante (manca la segretezza richiesta in taluni casi).

- La velocità di esecuzione del programma è piuttosto bassa in quanto ogni volta che viene impartito il comando di esecuzione (RUN per il BASIC e PASCAL) vengono sempre e comunque ripetute le fasi di controllo sintattico e di traduzione in linguaggio macchina del programma sorgente in quanto a priori non è possibile stabilire se la versione di programma presente in memoria è quella corretta oppure no.

- L'area di memoria a disposizione è piuttosto modesta in quanto per consentire l'esecuzione di un programma interpretato deve essere presente in memoria oltre al programma sorgente, l'interprete (linguaggio- interprete) completo corredato di RUN-TIME-PACKAGE (biblioteca completa).

Programmi Compilati

L'impiego di un linguaggio compilatore presenta pregi e difetti che di seguito vengono elencati:

- Il programma sorgente non può essere modificato con rapidità in quanto è necessario ricorrere all' EDITORE per ricaricare il programma in memoria, correggere la o le righe errate e salvare su disco il programma corretto. La verifica richiede che venga ripetuta la procedura di compilazione al completo.

- Il programma che viene eseguito dalla CPU è redatto in linguaggio macchina e quindi non è "listabile" su video o stampante (ciò consente un ottimo grado di segretezza sul codice sorgente che è stato utilizzato dal programmatore autore del programma) ciò costringe il cliente a rivolgersi al programmatore per eventuali future modifiche.

- La velocità di esecuzione del programma è più elevata rispetto a quella di un programma interpretato (il controllo sintattico del programma e la traduzione il linguaggio macchina è stato effettuato una volta per tutte dal compilatore prima di produrre il programma oggetto).

- L'area di memoria a disposizione è la massima possibile in quanto per il suo funzionamento il programma oggetto non ha bisogno né del compilatore né tantomeno del programma sorgente che in casi di programmi di una certa lunghezza addirittura non potrebbe nemmeno stare nell'intera RAM del calcolatore.

Si conclude perciò che, ove un determinato linguaggio abbia la possibilità di configurarsi nei due modi sopra descritti, sarà opportuno operare in ambiente Interprete nella fase di stesura, di correzione e di verifica di funzionalità (fase di Debugging) del programma stesso (sfruttando quindi tutti i vantaggi dell'ambiente Interprete) poi sarà quindi opportuno trasferirsi in ambiente Compilatore per procedere alla traduzione in linguaggio macchina del programma predisposto (con tutti i conseguenti vantaggi).

Va comunque ribadito che, programmi di modeste dimensioni consentono di operare indifferente in modo Interprete e in modo Compilatore mentre programmi di cospicue dimensioni obbligano ad operare in modo Compilatore.

Gli ambienti TURBO-PASCAL, TURBO-C, TURBO-BASIC della Borland; QUICK-C, QUICK-BASIC della Microsoft sono ambienti integrati in cui si può operare sia in modo Interprete che in modo Compilatore. In ambiente TURBO-PASCAL i "files-sorgenti" redatti in linguaggio PASCAL hanno estensione PAS per distinguerli dai "files-oggetto" contenenti il codice macchina 8088-8086 che hanno estensione EXE .

Il FORTRAN è nato ed ha conservato la sua iniziale struttura di linguaggio Compilatore; il BASIC invece è nato come linguaggio Interprete e solo in tempi relativamente recenti sono stati sviluppati Compilatori BASIC

Cenni sul software di base di un PC

Se ci limitiamo ad esaminare l'ambito dei PC IBM e compatibili oggi così diffusi sentiamo parlare di BIOS, di POST, di sistema operativo MSDOS e di sistema operativo Windows. Il BIOS è un firmware residente sulle EPROM della scheda madre che rende possibile la gestione degli I/O verso le periferiche base (tastiera, stampante, monitor video, cicalino) esso in genere incorpora un programma denominato POST (Power On Self Test) cui è demandato il compito di fare un test sull'hardware nella fase di power on (alimentazione del PC) Fra i principali test effettuati dal firmware POST ricordiamo il test della RAM, il test della scheda video e di altre unità quali la tastiera, l'orologio real-time (RTC), ecc. Le funzioni di configurazione del PC contenute nel BIOS consentono inoltre di definire nei dettagli le caratteristiche delle unità residenti nel PC (tipo di scheda video, tipo e numero di porte seriali e parallele, tipo e numero delle unità floppy installate, tipo e caratteristiche logiche dell'Hard Disk connesso all'interfaccia dischi. Le informazioni introdotte nella fase di setup del PC vengono poi salvate all'interno di RAM batterizzate presenti sulla scheda madre (solitamente il RTC contiene una piccola area RAM in cui vengono registrati i parametri della macchina).

All'accensione del PC viene attivato innanzitutto il POST; se il test dà esito positivo un'apposito segmento del BIOS denominato BOOT effettua la ricerca del sistema operativo dalle unità disco (floppy ed Hard); all'interno di queste unità vi deve essere un supporto cosiddetto "bootable". Il programma di BOOT va alla ricerca sul disco di un BOOT record che contiene tutte le informazioni sui file del sistema operativo MSDOS residenti su quel disco; nel caso in cui il "boot-record" non venga trovato esce un messaggio di errore su video che invita a sostituire il disco con un disco sistema. Nel caso in cui il "boot-record" venga trovato il programma di BOOT carica nella RAM del PC il sistema operativo residente su disco e passa quindi il controllo a quest'ultimo (fase di interpretazione dei comandi MSDOS). Nel caso in cui il POST incontri problemi nella fase di test degli elementi del PC vengono emessi su video dei codici di errori che consentono attraverso la consultazione di una apposita tabella di rendersi conto della parte del PC che non funziona:

Il sistema operativo MSDOS è un complesso di programmi attraverso i quali vengono gestite le complesse problematiche del file-system del PC e inoltre tutte le problematiche concernenti gli I/O presenti nel PC. Il sistema operativo più diffuso è l'MS-DOS della Microsoft oggi giunto alla edizione 6.22. Fra le funzioni più importanti di MS-DOS ricordiamo la formattazione di Floppy-disk ed Hard-disk, la creazione e rimozione di directory, la copia, la movimentazione, la cancellazione di files su disco ecc. . Nella tabella di pagina seguente è riportata la finestra di Help di MS-DOS 6.2 dove è presentato l'elenco dei comandi, dei driver software e di altre utilities del sistema operativo.

Il sistema operativo Windows è dotato di funzionalità potenziate rispetto a MS-DOS; l'interfaccia utente è di tipo a finestre più amichevole e di più facile approccio (viene prevalentemente utilizzato il mouse piuttosto che la tastiera); l'edizione di Windows per Workgroup 3.11 consente di realizzare piccole reti d'ufficio nelle quali è consentito condividere fra i vari utenti le risorse di rete (files, stampanti, modem). Recentemente è stato introdotto sul mercato Windows 95 (4 settembre 1995) che è annunciato come il sistema operativo che rivoluzionerà il mondo dei PC nei prossimi anni in virtù della sua architettura a 32 bit; esso tuttavia richiede di essere installato su macchine di una certa dimensione (minimo dichiarato 386DX con almeno 4MB di RAM e almeno 40MB di Hard disk).

Cenni sui pacchetti applicativi

Con la progressiva diffusione delle tecniche di Office Automation (Automazione dell'ufficio) si sono venuti affermando dei pacchetti software applicativi funzionanti sia in ambiente DOS sia in ambiente WINDOWS che consentono di risolvere una considerevole mole di problemi; i pacchetti applicativi possono essere suddivisi in cinque categorie:

- a) word-processor (elaborazione testi)
- b) spread-sheet (fogli elettronici)
- c) data base manager (elaborazione elettronica di archivi)
- d) pacchetti grafici e di disegno
- e) pacchetti DTP (Desk Top Publishing)

I "word-processor" sono pacchetti software studiati per la stesura di testi (lettere, relazioni, ecc.) in formato elettronico; fra i più noti ricordiamo Wordstar, Framework della Ashton Tate, Word per DOS, Works per DOS, Word per Windows e Works per Windows della Microsoft, Wordperfect.

I fogli elettronici o spreadsheet sono pacchetti software introdotti sul mercato per creare prospetti, tabelle, per stendere bilanci, ecc.; fra i più noti ricordiamo 123 della Lotus, Excel 5 della Microsoft.

I data base manager sono pacchetti software studiati per risolvere le problematiche connesse con la elaborazione di archivi elettronici; fra i più noti ricordiamo DBASE di Ashton Tate, Access di Microsoft.

<Help di MS-DOS 6.2>

<ANSI.SYS>	<Erase>	<Nlfunc>
<Append>	<Exit>	<Numlock>
<Attrib>	<Expand>	<Path>
<Comandi Batch>	<Fasthelp>	<Pause>
<Break>	<Fastopen>	<Power>
<Buffers>	<Fc>	<POWER.EXE>
<Call>	<Fcbs>	<Print>
<Cd>	<Fdisk>	<Prompt>
<Chcp>	<Files>	<Qbasic>
<Chdir>	<Find>	<RAMDRIVE.SYS>
<Chkdsk>	<For>	<Rd>
<CHKSTATE.SYS>	<Format>	<Rem>
<Choice>	<Goto>	<Ren>
<Cls>	<Graphics>	<Rename>
<Command>	<Help>	<Replace>
<Comandi file Config.sys>	<HIMEM.SYS>	<Restore>
<Copy>	<If>	<Rmdir>
<Country>	<Include>	<ScanDisk>
<Ctty>	<Install>	<Set>
<Date>	<Interlnk>	<Setver>
<Dblspace>	<INTERLNK.EXE>	<SETVER.EXE>
<Note su dblspace>	<Comandi per uso internaz.>	<Share>
<DBLSPACE.SYS>	<Intersvr>	<Shell>
<Debug>	<Keyb>	<Shift>
<Defrag>	<Label>	<SIZER.EXE>
	<Lastdrive>	<Smartdrv>
<Deltree>	<Lh>	<SMARTDRV.EXE>
<Device>	<Loadfix>	<Sort>
<Device di Periferica>	<Loadhigh>	<Stacks>
<Devicehigh>	<Md>	<Submenu>
<Dir>	<Mem>	<Subst>
<Diskcomp>	<Memmaker>	<Switches>
<Diskcopy>	<MenuColor>	<Sys>
<DISPLAY.SYS>	<MenuDefault>	<Time>
<Dos>	<MenuItem>	<Tree>
<Doskey>	<Mkdir>	<Type>
<Dosshell>	<Mode Comandi>	<Undelete>
<DRIVER.SYS>	<More>	<Unformat>
<Drivparm>	<Move>	<Ver>
<Echo>	<Msav>	<Verify>
<Edit>	<Msbackup>	<Vol>
<EGA.SYS>	<Mscdex>	<VSafe>
<Emm386>	<Msd>	<Xcopy>
<EMM386.EXE>	<Multi-config>	

I pacchetti grafici e di disegno sono stati studiati appositamente per elaborazioni grafiche computerizzate o per stendere disegni di tipo meccanico, edile-civile, o di apparati elettrici ed elettronici; fra i più diffusi ricordiamo Corel Draw per la grafica; Autocad per l'ambito del disegno civile e meccanico, Orcad SDT per quanto concerne la stesura di disegni di tipo elettronico.

I pacchetti DTP o di editoria elettronica vengono usati per preparare fogli pubblicitari, notiziari, bollettini; fra i più diffusi ricordiamo Publisher di Microsoft, Ventura di Xerox e Page Maker della Aldus.

Le carte di flusso (flow chart)

Per **flow-chart** intenderemo uno schema grafico in cui è rappresentato in modo inequivocabile l'ordine sequenziale di svolgimento delle varie fasi (Input, Elaborazione, Decisioni, Output) che rendono possibile la esecuzione automatica di un algoritmo da parte di un sistema di elaborazione dati.

Non daremo qui una descrizione completa dei simboli grafici che possono essere usati nei diagrammi di flusso: sono troppo numerosi e, a mio parere, spesso non essenziali; distingueremo solo le funzioni principali e ne daremo la forma di rappresentazione più corrente.

INIZIO PROGRAMMA (s1)

Segnala l'inizio elaborazione in un flow-chart. Non necessariamente lo si trova in tutte le carte di flusso in quanto l'inizio è sempre posizionato in alto al foglio e generalmente sulla sinistra.

FINE PROGRAMMA (o di parte di programma) (s2)

Segnala la fine della fase di elaborazione in un flow-chart. Deve essere sempre presente.

INTERCONNESSIONE fra due FLOW CHART (o parti di esso) (s3)

Viene utilizzato per connettere due parti separate di un flow-chart che non possono essere disegnate unite sul medesimo foglio, ma che sono connesse e devono essere eseguite sequenzialmente una dopo l'altra. All'interno della circonferenza viene riportata o una lettera maiuscola dell'alfabeto o un simbolo numerico.

LINEA OPERATIVA (s4)

Rappresenta il verso della successione cronologica delle operazioni (solitamente dall'alto verso il basso) del foglio in cui è riprodotto il "flow-chart". In caso il flusso di elaborazione debba svilupparsi dal basso verso l'alto si deve obbligatoriamente indicare l'orientamento del percorso con una freccia.

FASE DI ELABORAZIONE (s5, s6)

In essa si specifica l'operazione di tipo logico o aritmetico e/o una operazione di trasferimento all'interno della memoria.

Nel primo dei due riquadri è stata riportata l'operazione di somma fra i contenuti di due zone memoria C e B ed il successivo trasferimento del risultato della somma nella zona memoria A. Nel secondo riquadro è stato indicato il trasferimento del contenuto della zona memoria C nella zona memoria B. Rientra in questo tipo di schematizzazione pure la chiamata ad un sottoprogramma (procedura in Pascal; subroutine in Basic).

NODO di confluenza (s7, s8, s9)

Indica la confluenza di più percorsi elaborativi in uno stesso punto del programma. Vi è però un'unica via di uscita. Per realizzare un nodo di confluenza si opera spesso in modo diverso ma il significato comunque è altrettanto chiaro: si tratta di far confluire le linee operative su una stessa fase elaborativa. Si noti che le due grafie s8 e s9 sono del tutto equivalenti in quanto la confluenza si sottintende avvenuta prima che una determinata operazione abbia luogo; è tuttavia preferibile utilizzare il simbolo s9 che non dà luogo ad equivoci.

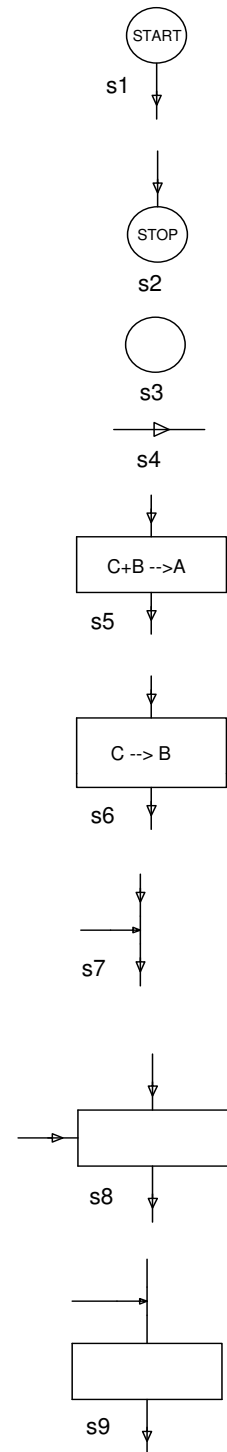


fig. 2

TRASFERIMENTO da e per l'esterno (s10, s11)

Rappresentano operazioni di Input ossia introduzione di dati dall'esterno tramite una delle unità dedicate (tastiera, Unità di memoria di massa, mouse, modem, etc.) ma anche operazioni di output ossia invio di dati o risultati di elaborazione verso apposite unità (video, stampante, unità di memoria di massa, modem etc.). L'unità di "default" per operazioni Input è la tastiera del terminale video, l'unità "default" per operazioni Output è il monitor del terminale video.

BLOCCO DECISIONALE (s12, s13, s14)

Vi si indicano i confronti con le due possibili alternative nella risposta; qualora la proposizione logica contenuta all'interno del box sia vera il flusso di esecuzione prosegue seguendo il cammino indicato dal SI (o Yes), in caso contrario il flusso prosegue seguendo la via che esce dal NO (No). La forma dipende solamente dalle preferenze del programmatore o analista.

Il problema della protezione del calcolo

Quando si devono effettuare calcoli in modo automatico con un sistema di elaborazione dati si devono prendere le dovute precauzioni per proteggere i calcoli ossia per evitare che in taluni casi la presenza di un dato errato blocchi completamente il sistema. Le più importanti operazioni matematiche che possono originare blocco per errore sono:

- a) la operazione di divisione
- b) la operazione radice di indice pari (e fra esse la radice quadrata)
- c) la operazione di calcolo del logaritmo

E' quindi importante prima di fare svolgere all'elaboratore una delle operazioni matematiche sopraindicate, accertarsi che il valore assunto dalla variabile indipendente appartenga al dominio di definizione della funzione stessa in altre parole:

- a) nel caso della divisione bisogna verificare che il denominatore sia diverso da zero
- b) nel caso di radice quadrata o di indice pari il radicando non deve assumere valori negativi
- c) nel caso di calcolo della funzione logaritmo l'argomento deve assumere valori sempre positivi non nulli.

Attraverso opportuni box decisionali posti prima che vengano effettuate le operazioni matematiche citate, si può prevedere un percorso alternativo al calcolo nei casi particolari sopraindicati.

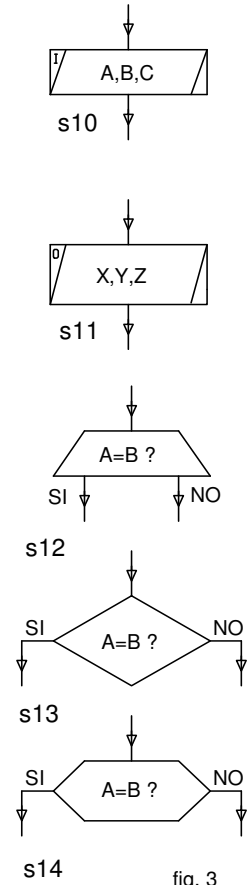


fig. 3

Le strutture decisionali del Turbo Pascal

In Pascal sono previste tre strutture decisionali: la struttura If Then, la Then Else e la struttura Case of.

Strutture If Then e If Then Else

In fig. 4, 5 e 6 sono riportati due segmenti di flow chart che schematizzano le tipologie di strutture decisionali If possibili.

L'istruzione IF elabora una espressione di *tipo Boolean* per determinare quale fra le due possibili alternative deve essere seguita. La clausola ELSE fa parte opzionalmente del costrutto. La struttura sintattica è la seguente :

```
IF <espressione-booleana> THEN
  <istruzione1>
  [ ELSE <istruzione2> ]
```

Se l'<espressione-booleana> assume il valore True (ossia se è vera), allora viene eseguita l'istruzione1> e di seguito verranno eseguite le istruzioni che seguono il costrutto IF.

Se l'<espressione-booleana> assume il valore False (ossia se è falsa), allora non viene eseguita l'istruzione1> ma viene eseguita l'istruzione2>.

Nel caso di fig. 4 il segmento di programma va tradotto come segue :

```
...
If A=0 Then C := B/A
Else C := B/2 ;
C := C + D ;
...
```

Si noti che l'istruzione1> e l'istruzione2> può essere costituita da un intero blocco di istruzioni comprese tra i delimitatori Begin End, ma potrebbe anche essere un'altra istruzione IF. Nel caso di fig. 5 il segmento di programma va tradotto come segue :

```
...
If A=0 Then
  Begin
    C := B/A ;
    D := B * C ;
  End
Else
  Begin
    C := B/2 ;
    D := C/4 ;
  End ;
C := C + D ;
...
```

La struttura di fig. 6 è del tipo If Then (manca completamente l'alternativa ELSE). In tal caso il segmento di programma va tradotto come segue :

```
...
If A=0 Then C := B/A ;
C := C + D ;
...
```

STRUTTURA IF-THEN-ELSE

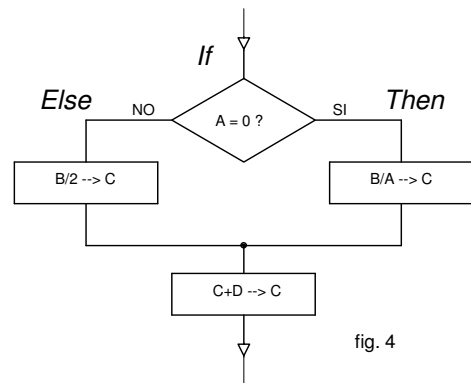


fig. 4

STRUTTURA IF-THEN-ELSE

CON PIU' DI UNA ISTRUZIONE NEI RANGHI THEN - ELSE

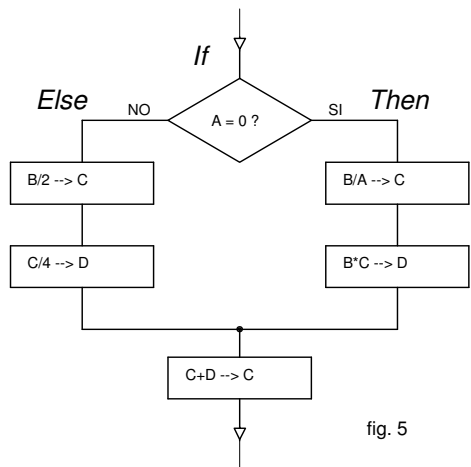


fig. 5

STRUTTURA IF-THEN

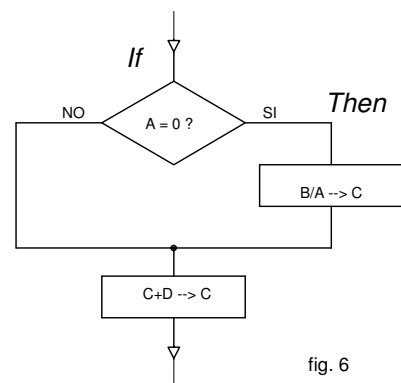


fig. 6

Osservazione:

Spesso nasce confusione quando si deve correttamente collocare il ";" in una istruzione IF. È opportuno in proposito ricordare che la frase ELSE deve essere considerata parte integrante dell'istruzione IF (non si tratta di una istruzione a sè stante) e quindi non deve essere preceduta dal ";". Infatti il compilatore interpreta il ";" come un separatore di istruzioni, e quindi non ha senso che ne venga collocato uno all'interno di una istruzione (ma questo è proprio quello che succede quando si fa precedere da un ";" la frase ELSE).

Struttura decisionale Case ... of

La struttura Case ... of o struttura decisionale multipla viene utilizzata quando in base al valore assunto da una grandezza ordinale debbano essere prese decisioni diversificate.

La istruzione CASE consente al programma di verificare se il valore di una variabile ordinale rientra fra quelli consentiti, e prevede la esecuzione di istruzioni appropriate per ogni valore consentito.

La forma sintattica della struttura decisionale Case... of è la seguente :

```
CASE <espressione-ordinale> OF
  <valore1> : <azione1>;
  <valore2> : <azione2>;
  .
  <valoreN> : <azioneN>;
  [ ELSE <azioneN+1>;]
END; { case }
```

Si noti che la clausola ELSE è opzionale; se fosse omessa, e se il valore assunto dalla <espressione- ordinale> non coincidesse con nessuno dei valori elencati nel costrutto "CASE", allora l'esecuzione proseguirebbe semplicemente con l'istruzione successiva alla CASE. L'uso del costrutto CASE in sostituzione di un gruppo di istruzioni IF rende i programmi più facilmente leggibili ed interpretabili.

Esempio 1

```
program CaseDemo;
```

```
var
  a,b,z : real;
  k : integer;
```

```
begin
  Write('Introduci due valori reali A e B'); Readln(A,B);
  Writeln('Dimostrativo Costrutto CASE OF');
  Writeln('1-Somma A+B');
  Writeln('2-Sottrai A-B');
  Writeln('3-Moltiplica A*B');
  Writeln('4-Dividi A/B');
  Writeln('5-Calcola Radice quadrata di A');
  Readln(k); {Il valore di k determina la scelta effettuata}
```

La struttura decisionale multipla

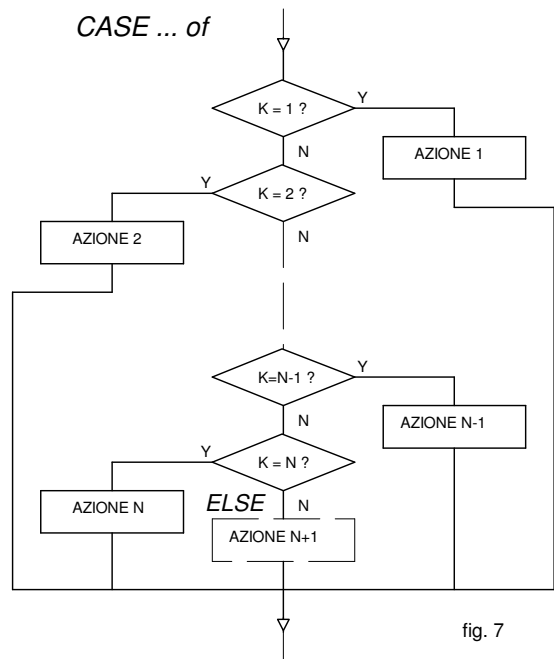


fig. 7

```

CASE k OF
  1 : Z := A+B;
  2 : Z := A-B;
  3 : Z := A*B;
  4 : Z := A/B;
  5 : Z := SQRT(A);
ELSE WriteLn('Valore di K fuori-range!');
end; { case }
IF (K>0) AND (K<6) THEN WriteLn('Z=',Z:10:3);
end. { CaseDemo }

```

Strutture Iterative del Turbo Pascal

Nella stesura di programmi per la elaborazione elettronica dei dati è molto frequente il ricorso a strutture iterative (ossia ripetitive) che sfruttano le vere potenzialità di un calcolatore capace, com'è noto di eseguire a grande velocità un considerevole numero di operazioni elementari. Esistono quattro tipi di strutture iterative:

- i loop infiniti

(si tratta di situazioni quasi sempre non desiderate che si possono ingenerare in un programma per errore) nel caso l'elaboratore ripeta ciclicamente senza mai potersi allontanare dal ciclo sequenze di operazioni; l'unico modo per uscire da un loop infinito è quello di inviare alla CPU una richiesta di interrupt non mascherabile oppure di attivare sul calcolatore la funzione di Reset.

- REPEAT...UNTIL:

In fig. 8 è riportato un segmento di flow chart che schematizza la struttura iterativa Repeat...Until

Si entra nel ciclo da R1. Premesso che i blocchi ELAB 1... ELAB K rappresentano genericamente nel flow-chart il rango del ciclo iterativo, il meccanismo operativo della struttura Repeat ... Until è il seguente: se la condizione riportata dopo la proposizione Until (vedi flow-chart) è FALSA viene eseguito il rango; quando la condizione è VERA il rango non viene più eseguito ed il flusso procede oltre (uscita da R2).

La forma sintattica della struttura iterativa Repeat ... Until è la seguente :

```

...
REPEAT
  <ELAB 1>;
  ...
  <ELAB K >;
UNTIL <Condizione>;
...

```

Esempio 2

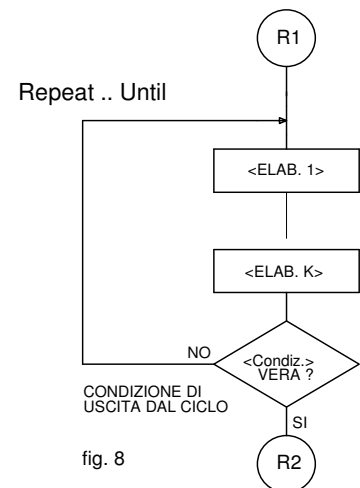
Si vuole accettare in input da tastiera un numero N intero solo se esso è positivo e minore di 25. Si procederà come segue:

```

...
REPEAT
  Write(' Introduci N ');
  Readln(N);
UNTIL (N > 0) AND (N < 25);
...

```

Se la condizione espressa dopo la frase Until è FALSA il rango del ciclo iterativo viene eseguito, se la condizione è VERA il rango del ciclo iterativo non viene più eseguito e l'elaboratore passerà ad eseguire le istruzioni successive al Repeat .. Until.



Esempio 3

Si vuole continuare a riempire di caratteri A all' infinito il terminale video; si può procedere come segue :

```

...
REPEAT
  Write('A');
UNTIL FALSE ;
...

```

La struttura REPEAT ... UNTIL è già di per se stessa un contenitore di istruzioni multiple (il rango del ciclo iterativo) con tale struttura non si deve perciò usare la coppia BEGIN END prevista in altri casi.

- WHILE...DO,

In fig. 9 è riportato un segmento di flow chart che schematizza la struttura iterativa While ... do. Si entra nel ciclo da W1. Premesso che i blocchi ELAB 1... ELAB K rappresentati nel flow-chart costituiscono il rango del ciclo iterativo, il meccanismo operativo della struttura While ... Do è il seguente: se la condizione riportata dopo la proposizione While (vedi flow-chart) è VERA viene eseguito il rango; quando la condizione è FALSA il rango non viene più eseguito ed il flusso procede oltre (uscita da W2).

La forma sintattica della struttura iterativa While ... do è la seguente :

```

...
WHILE <Condizione> DO
  Begin
    <ELAB. 1>;
    ...
    <ELAB. K>;
  End;
...

```

Esempio 4

Si vuole accettare in input da tastiera un numero N intero solo se esso è positivo e minore di 25. Si procederà come segue:

```

...
WHILE (N < 0) OR (N > 24) DO
  Begin
    Write(' Introduci N ');
    Readln(N);
  End;
...

```

Se la condizione espressa dopo la frase While è VERA il rango del ciclo iterativo viene eseguito, se la condizione è FALSA il rango del ciclo iterativo non viene più eseguito e l'elaboratore passerà ad eseguire le istruzioni successive al While ... Do.

Esempio 5

Si vuole continuare a riempire di caratteri A all' infinito il terminale video; si può procedere come segue :

```

...
While True Do Write('A');
...

```

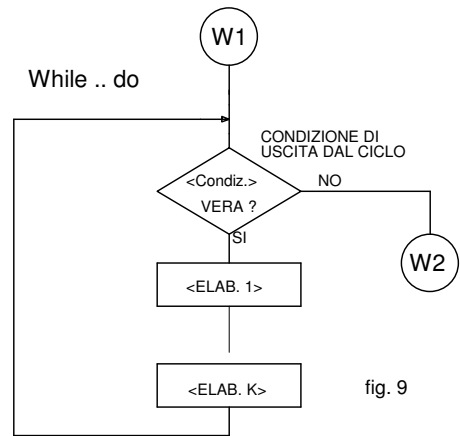


fig. 9

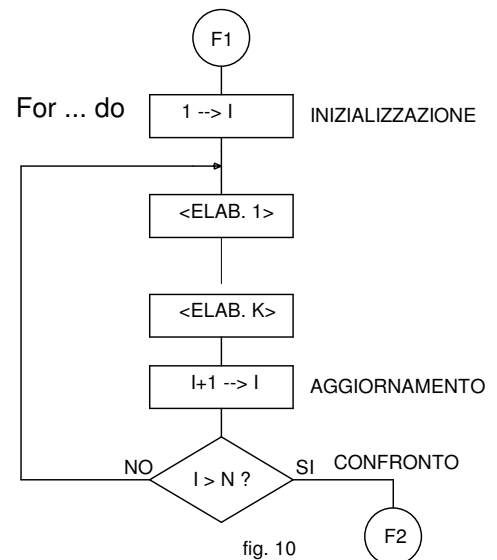


fig. 10

- FOR...DO,

In fig. 10 è riportato un segmento di flow chart che schematizza la struttura iterativa For ... do. Si entra da F1. La struttura FOR .. DO è leggermente diversa dalle precedenti. Innanzitutto essa è controllata da una variabile ordinale (solitamente detta contatore) cui è affidato il compito di controllare il numero di volte in cui viene eseguito il rango del ciclo iterativo; tale numero deve essere noto com'è ovvio sin dall'inizio del ciclo. Nella struttura rappresentata nel disegno a fianco, la variabile contatore I parte dal valore iniziale 1, una volta eseguito il rango (ELAB.1 ... ELAB.K) il contatore I viene aggiornato di una unità, il valore attuale di I viene confrontato con il valore N (numero di iterazioni del ciclo); se I risulta essere minore o eguale ad N si ripete il ciclo, se I risulta invece maggiore di N il ciclo si conclude e si esce da F2. La forma sintattica della struttura iterativa For ... do è la seguente :

```
FOR I:= 1 TO N DO
Begin
  <ELAB. 1>
  ...
  <ELAB. K>
End;
```

Osservazioni

Il rango presente nella struttura Repeat ... Until viene sempre eseguita almeno una volta anche se la condizione è già in partenza vera; infatti il confronto per decidere se ripetere oppure no è collocato proprio dopo l'esecuzione del rango.

Nel caso di While ... Do con condizione Falsa in partenza il rango, non viene eseguito in nessun caso; il confronto è infatti collocato in testa.

Nel caso di For ... Do la istruzione For traduce inizializzazione del contatore, aggiornamento dello stesso e confronto finale.

Esempi di programmazione

Passiamo ora ad esaminare una serie di esempi di carattere matematico che ci permettono di vedere all'opera le varie strutture iterative e decisionali brevemente presentate. Per ciascun esercizio verrà fornita innanzitutto una rapida trattazione preliminare sulle problematiche, si passerà alla stesura del flow-chart che schematizza l'algoritmo risolutivo ed infine si presenterà il programma vero e proprio redatto in Turbo Pascal 5.5 Borland.

Media aritmetica fra N valori

Supponiamo di disporre di N valori reali X_1, X_2, \dots, X_n e di volere elaborare la media aritmetica fra essi definita come segue:

$$M = \frac{1}{N} \sum_{i=1}^n x_i = \frac{x_1 + x_2 + \dots + x_N}{N}$$

per prima cosa faremo entrare il valore di N; il programma deve essere in grado di elaborare la media aritmetica qualunque sia il valore di N (intero positivo). Dovrò poi definire una zona di accumulo M (variabile semplice reale) all'interno della quale si viene a costruire progressivamente il numeratore (ossia la sommatoria) passo dopo passo mano a mano i dati vengono introdotti; ecco quindi la necessità di introdurre una struttura iterativa. La zona di accumulo M dovrà essere azzerata all'inizio per evitare che l'operazione di accumulo sia affetta da errori dovuti a quantità numeriche preesistenti all'atto della esecuzione del programma. Il contatore I descrivendo la sequenza dei numeri naturali da 1 a N controlla di fatto l'input dei dati x_1, x_2, \dots, x_n , il loro accumulo nella zona M e l'uscita. Il ciclo iterativo controllato da una variabile I viene realizzato in Pascal con il seguente costrutto:

```
FOR I:=1 TO N DO
Begin
.....
End;
```

All'interno della coppia Begin End vanno collocate le istruzioni che compongono il "Rango" del ciclo iterativo ossia l'input del generico valore x ed il suo accumulo nella zona memoria M; la inizializzazione, l'aggiornamento ed il confronto sul contatore I (presenti nella carta di flusso di fig. 11) sono parte della struttura del costrutto FOR.

Traduzione in linguaggio PASCAL

```
Program Media(input,output);
var
  X,M : real;
  I,N : integer;
Begin
  Write('N= ');      Readln(N);
  M := 0; {Azzerò la zona di accumulo}
  For I:=1 to N do
  Begin
    Write('X',I,' '); Readln(X); {Lettura del dato Xi}
    M := M+X; {Accumulo in memoria}
  End;
  M := M/N; {Calcolo Media}
  Writeln('Media = ',M:10:3);
End.
```

Media aritmetica

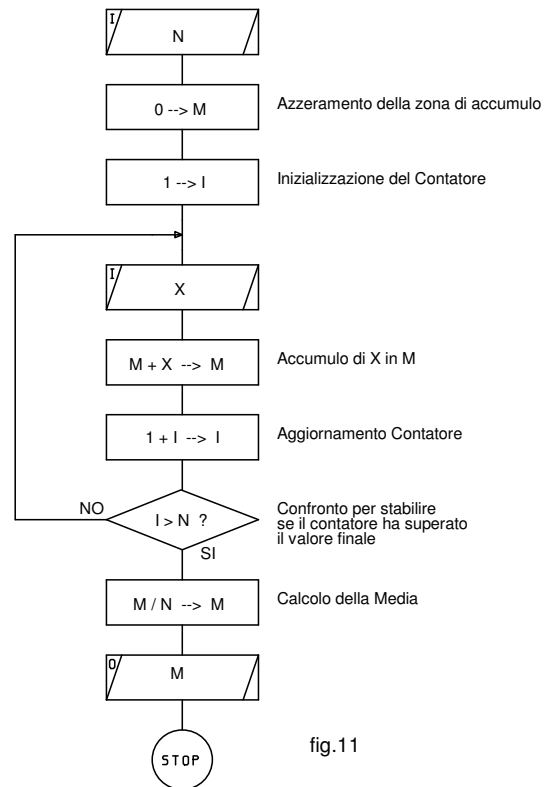


fig.11

Risoluzione di una equazione di secondo grado

Premesse di carattere matematico

Com'è noto il problema di risolvere una equazione di secondo grado del tipo :

$$a \cdot x^2 + b \cdot x + c = 0$$

ove a, b, c sono coefficienti reali porta alle seguenti soluzioni :

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \quad x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

il termine sotto radice viene denominato discriminante:

$$\Delta = b^2 - 4ac$$

È noto che le due soluzioni x_1 e x_2 possono essere calcolate solo se il coefficiente a risulta diverso da zero; infatti in tal caso il denominatore $2a$ delle espressioni di x_1 e x_2 risulta diverso da zero e quindi la divisione è effettuabile.

Nel caso in cui a risulti uguale a zero, l'equazione diviene di primo grado:

$$b \cdot x + c = 0$$

e il calcolo conduce alla soluzione:

$$x = -\frac{c}{b}$$

calcolabile solo se il coefficiente b risulta diverso da zero. Nel caso in cui a e b risultino entrambi uguali a zero si perviene all'espressione:

$$c = 0$$

che risulta essere una *identità* se c effettivamente assume valore nullo, mentre risulta essere una *equazione impossibile* se il coefficiente c assume valore diverso da zero. Un altro problema di calcolo riguarda la possibilità di estrarre in campo reale la radice quadrata:

$$\sqrt{\Delta} = \sqrt{b^2 - 4ac}$$

che come è noto impone la condizione :

$$\Delta \geq 0 \rightarrow b^2 \geq 4ac$$

Si parla di soluzioni reali e distinte se Δ risulta maggiore di zero, soluzioni reali e coincidenti se Δ risulta uguale a zero e di soluzioni complesse coniugate se Δ risulta minore di zero. Il calcolo delle soluzioni complesse si effettua agevolmente tenendo conto che:

$$\text{se } \Delta \leq 0 \rightarrow \Delta = -|\Delta| = 4ac - b^2$$

e quindi :

$$\sqrt{\Delta} = \sqrt{-|\Delta|} = \pm j\sqrt{|\Delta|}$$

da cui si ha:

$$x_{1,2} = -\frac{b}{2a} \pm j \frac{\sqrt{|\Delta|}}{2a} = R \pm jM$$

Le due soluzioni essendo complesse coniugate hanno la stessa parte reale R :

$$R = -\frac{b}{2a}$$

e parti immaginarie di segno opposto. Il coefficiente immaginario M vale :

$$M = \frac{\sqrt{|\Delta|}}{2a}$$

Nella carta di flusso la prima operazione è rappresentata dalla fase di input dei tre coefficienti A, B, C sui quali non viene effettuato alcun controllo; successivamente ci si chiede se A valga zero; in caso di condizione affermativa (equazione di primo grado) si procede a verificare se B e successivamente C valgano zero per discriminare fra soluzione di primo grado propria, soluzione impossibile e identità.

Nel caso in cui A risulti diverso da zero, (equazione di secondo grado) si procede al calcolo del discriminante; nel caso in cui Δ risulti maggiore o uguale a zero siamo in presenza di due soluzioni reali x_1 e x_2 , in caso contrario si procede al calcolo della parte reale R e della parte immaginaria M delle due soluzioni che risultano complesse coniugate.

Si osservi che la carta di flusso, discretamente complessa, presenta solamente un unico tratto di entrata (START) ed un unico tratto di uscita (STOP).

Ogni situazione è stata accuratamente codificata cosicché il sistema di elaborazione automatica (personal computer) è in grado di prendere in ogni situazione decisioni precise ed appropriate.

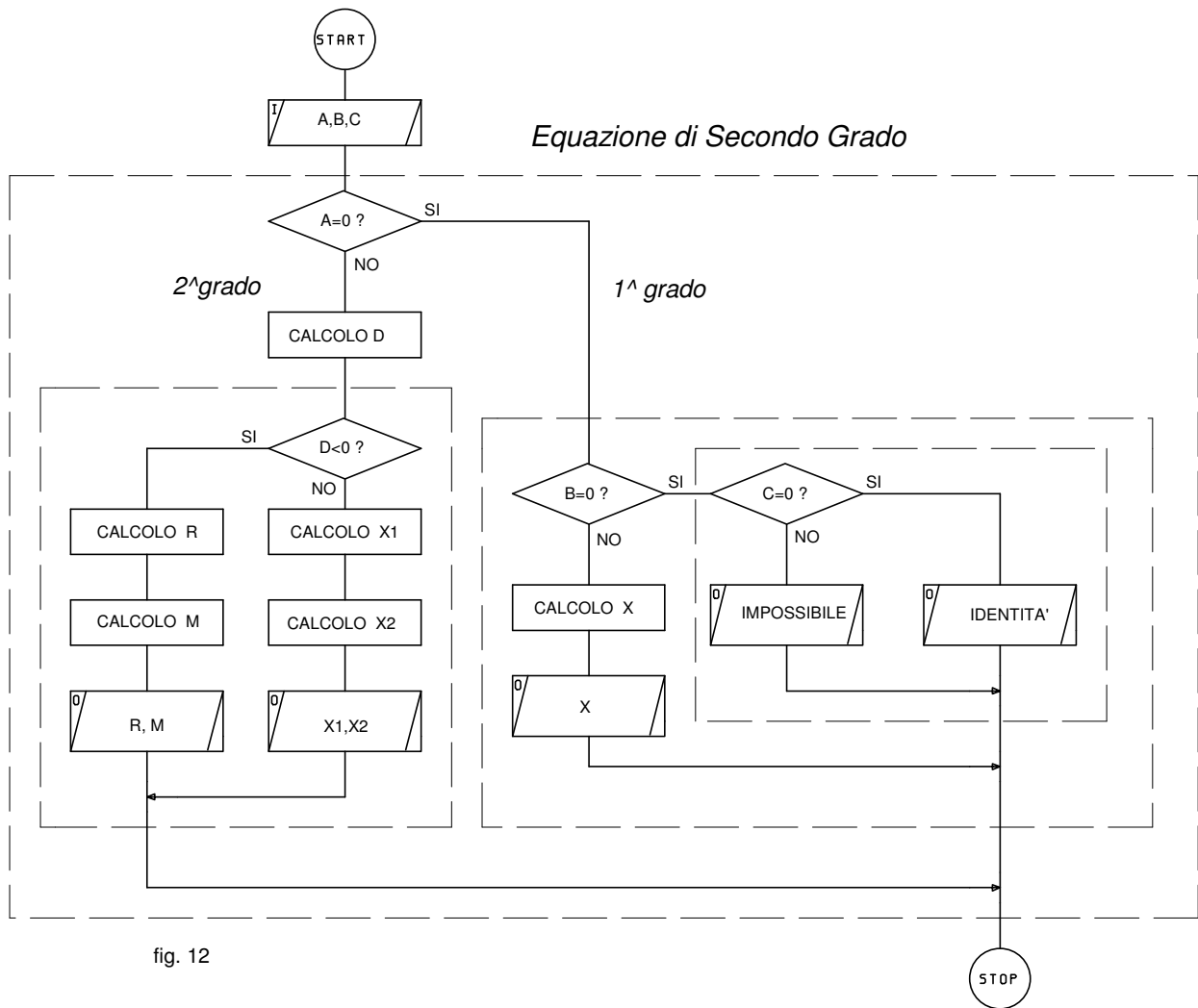


fig. 12

{ Questo programma trova le soluzioni reali di una equazione di 2.o grado.
 I dati richiesti in ingresso sono :
 i coefficienti reali A, B, C.
 I risultati forniti sono :
 1-) Le soluzioni X1 e X2 oppure
 2-) L'unica soluzione X nel caso A=0
 3-) I messaggi di avvertimento nei casi particolari (A=0 e B=0)
 4-) La parte reale R e immaginaria M delle soluzioni nel caso D risulti negativo }

```

Program Equaz2(Input,Output);
var A,B,C,D,X1,X2,X,R,M : real; {dichiarazioni relative alle variab.}
BEGIN
  Write(' Coefficiente A= ');
  Readln(A);
  Write(' Coefficiente B= ');
  Readln(B);
  Write(' Coefficiente C= ');
  Readln(C);
  If A=0 Then
    Begin { Risolvi equazione di 1^ grado }
      If B=0
        Then
          If C=0 Then Writeln('Identita ')
          Else Writeln(' Equazione Impossibile ')
        Else
          Begin
            X := -C/B;
            Writeln(' Una soluxione X= ',X:10:3);
          End; { fine ELSE If B=0 }
        End { fine THEN If A=0 }
      Else { If A=0 }
        Begin { Equazione di 2^ grado }
          D := B*B-4*A*C;
  
```

```

If D < 0 then
  Begin
    R := -B/(2*A)
    M := SQRT(-D)/(2*A);
    Writeln('Due soluzioni complesse coniugate');
    Writeln('R= ',R:10:3,' M= ',M:10:3);
  End { fine THEN If D<0 }
Else
  Begin
    X1 := (-B+SQRT(D))/(2*A);    X2 := (-B-SQRT(D))/(2*A);
    Writeln('Due soluzioni reali ');
    Writeln('X1= ',X1:10:3,' X2= ',X2:10:3);
  End; { fine ELSE If D<0 }
End; { if A=0 }
Readln;
END.

```

Conversione da Coordinate Cartesiane a Coordinate Polari

Premesse di carattere matematico

Sia dato un punto di coordinate $P=(A,B)$ si vogliono determinare le sue coordinate polari z e φ ove z rappresenta il modulo del vettore OP (O origine del sistema di assi cartesiani) e φ rappresenta lo sfasamento esistente fra la direzione individuata dal vettore OP e la direzione del semiasse positivo x .

$$z = \sqrt{a^2 + b^2}$$

$$\varphi = \arctg\left(\frac{b}{a}\right)$$

Il problema potrebbe sembrare di semplicissima impostazione ma così non è. Infatti, mentre il calcolo del modulo non presenta particolari problemi in quanto il radicando è sempre positivo o nullo, la determinazione della fase presenta due distinti problemi :

a)- Bisogna proteggere il calcolo dalla eventualità che il coefficiente a assuma valore zero; è facile convincersi che in tal caso φ assume il valore $\pi/2$ (90°) se b risulta positivo; assume il valore di $-\pi/2$ (-90°) se b risulta negativo.

b)- La funzione $\arctg(x)$ per valori di x compresi fra $-\infty$ e $+\infty$ restituisce sempre angoli compresi fra $-\pi/2$ e $\pi/2$. Ciò significa che a punti collocati nel II° e nel III° quadrante verrà attribuita erroneamente una fase che caratterizza il I° oppure il IV° quadrante. È sufficiente osservare che se la coordinata X di P (ossia il valore di a) risulta positiva, il punto P appartiene certamente al I° oppure al IV° quadrante e quindi la funzione $\arctg(x)$ restituisce il valore appropriato di φ , mentre se la coordinata X di P risulta negativa, il punto P è collocato nel II° o nel III° quadrante e quindi al valore restituito dalla funzione $\arctg(x)$ va aggiunto π per ottenere il valore appropriato di φ .

Si deve inoltre tenere conto che fra la misura di un angolo in gradi sessagesimali e in radianti sussiste la proporzione :

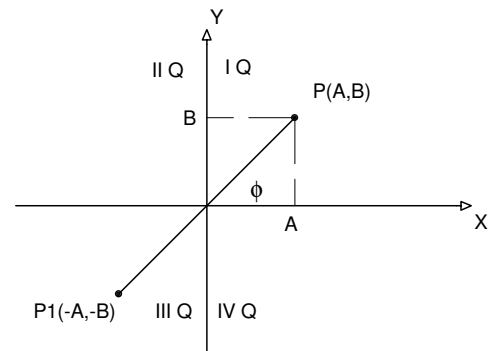
$$\alpha_o : \alpha_{rad} = 180 : \pi$$

da cui risolvendo rispetto ad α prima in gradi e poi in radianti si ha:

$$\alpha_o = \frac{\alpha_{rad} \cdot 180}{\pi} \quad \alpha_{rad} = \frac{\alpha_o \cdot \pi}{180}$$

Nella carta di flusso la prima operazione è rappresentata dalla fase di input delle coordinate cartesiane A, B del punto P sulle quali non viene effettuato alcun controllo; successivamente si procede al calcolo del modulo del segmento OP quindi si procede a verificare se A vale zero (protezione calcolo). Se A risulta diverso da zero viene calcolata la fase del vettore OP ; quindi si verifica se B è maggiore o minore di zero per apportare l'eventuale correzione alla fase φ come detto.

Se A risulta uguale a zero il segno di φ dipende dal segno di B ; se A e B risultano entrambi zero il modulo del vettore OP risulta nullo, la fase φ risulta indeterminata.



Conversione da Coordinate
Cartesiane a Polari

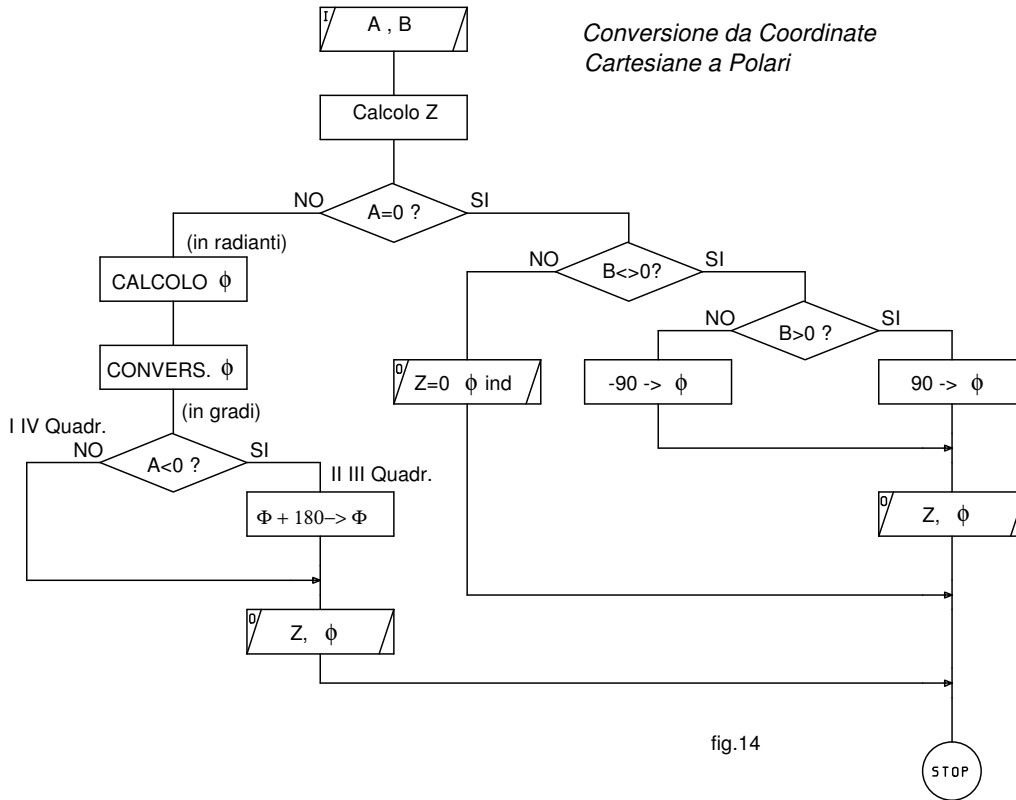


fig.14

*{ Questo Programma serve per convertire le coordinate cartesiane di un punto nelle corrispondenti coordinate polari.
 Dati Input : coordinate A e B del punto
 Dati Output : valore del modulo Z e della fase FI (in gradi sessagesimali.)
 Il programma tiene conto delle particolari situazioni che si verificano quando A vale 0 e del fatto che la funzione
 Arctg(x) fornisce valori di j compresi fra $-\pi/2$ e $+\pi/2$ }*

Program Conv(input,output);

var A,B,Z,FI,X : real;

BEGIN

writeln('Conversione da coordinate cartesiane a polari');

write('Coeff. a = '); readln(A);

write('Coeff. b = '); readln(B);

Z := SQRT(A*A+B*B); {Elaborazione del modulo }

if A=0 { se A=0 la fase é +90 oppure -90 }

then

begin

if B=0 then writeln(' Z=0 - Fase indeterminata')

else

begin

if B > 0 then FI := 90

else FI := -90;

writeln(' Z= ',Z:10:3, ' FI= ',FI:5:3);

end;

end

else

begin

X := ARCTAN(B/A); { X é espresso in radianti }

FI := (X*360)/(2*Pi); { converto in gradi sessagesimali }

{ Si noti che anche l'angolo FI+180 ammette la stessa tangente trigonometrica. Perció bisogna accertare se il punto é collocato nel 1.o o nel 3.o oppure nel 2.o o nel 4.o quadrante; analizzando il segno di A é possibile risolvere la questione.}

if A < 0 then FI := FI + 180;

writeln(' Z= ',Z:10:3, ' FI= ',FI:5:3);

end;

writeln(' premere <return> per continuare !'); readln;

END.

Determinazione del Baricentro di un sistema di forze

Premesse di carattere matematico

La determinazione del Baricentro G di un sistema di Forze è un classico problema di media ponderale; infatti dette F_1, F_2, \dots, F_n le N forze applicate in punti collocati a distanza X_1, X_2, \dots, X_n rispetto ad un riferimento O arbitrario, la risultante R del sistema di forze è data dall'espressione:

$$R = \sum_1^n F_i = F_1 + F_2 + \dots + F_n$$

il momento risultante del sistema di forze è dato dall'espressione:

$$M = \sum_1^n F_i \cdot x_i = F_1 x_1 + F_2 x_2 + \dots + F_n x_n$$

La distanza del Baricentro G dal riferimento comune O , è data dall'espressione :

$$X_G = \frac{M}{R} = \frac{\sum_1^n F_i \cdot x_i}{\sum_1^n F_i}$$

La formula è formalmente analoga a quella della media ponderale:

$$X_p = \frac{\sum_1^n p_i \cdot x_i}{\sum_1^n p_i}$$

Osservando il "flow-chart" si nota che per prima cosa faremo entrare il valore di N ; il programma deve essere in grado di elaborare la media ponderale qualunque sia il valore di N (intero positivo). Dovrò definire due zone di accumulo: M (variabile semplice reale) per accumulare il Momento complessivo delle forze, R (variabile semplice reale) per accumulare la Risultante del sistema di forze. Le zone di accumulo dovranno essere azzerate all'inizio. Il contatore I descrivendo la sequenza dei numeri naturali da 1 a N controlla di fatto l'input dei dati $F_1, x_1; F_2, x_2; \dots; F_n, x_n$, il loro accumulo nelle zone appropriate l'uscita quindi dal ciclo iterativo. Si pone poi un ulteriore problema: proteggere il calcolo contro la eventualità che la risultante R del sistema di forze assuma valore 0: in tal caso il baricentro risulta indeterminato. Viene ora riportata la traduzione in linguaggio Pascal coerente con quanto riportato nella carta di flusso.

```
Program Baric(Input, Output);
var R,M,F,X,B : real;
    N,I : integer;
```

Begin

```
  Writeln('Programma per determinare il Baricentro');
  Write('Numero di forze = ');
```

```
  Readln(N);
```

```
  R := 0; {annullo zona accumulo risultante}
```

```
  M := 0; {annullo zona accumulo momenti}
```

```
  For I := 1 to N do
```

```
    Begin
```

```
      Write('Forza ',I,' = ');
```

```
      Readln(F);
```

```
      Write('Braccio ',I,' = ');
```

```
      Readln(X);
```

```
      R := R + F; {accumulo forze}
```

```
      M := M + F*X; {accumulo momento}
```

```
    End; { FOR }
```

```
  If R = 0
```

```
    Then Writeln('Baricentro Indeterminato')
```

```
  Else
```

```
    Begin
```

```
      B := M/R;
```

```
      Writeln('Baricentro = ',B:7:2);
```

```
    End; { ELSE }
```

```
End.
```

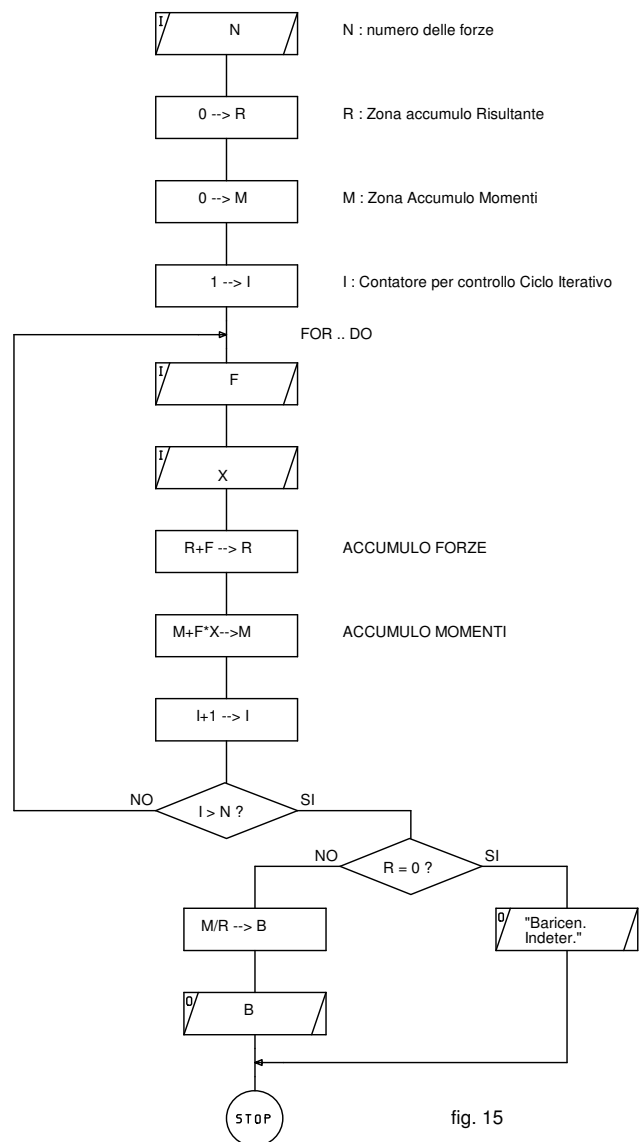


fig. 15

Equazione della retta passante per due punti

Premesse di carattere matematico

Per poter determinare l'equazione di una retta passante per due punti P1 e P2, è essenziale conoscere le coordinate dei due punti in questione: P1(x1,y1) e P2(x2,y2).

$$y = m \cdot x + p$$

com'è noto m rappresenta il coefficiente angolare esso è dato dall'espressione:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad x_2 \neq x_1$$

calcolabile solamente se risulta x2 diverso da x1. Il coefficiente p, ordinata del punto di intersezione fra la retta r e l'asse y, può essere determinato imponendo la condizione di passaggio di r per P1(x1,y1); risulta in tal caso :

$$p = y_1 - m \cdot x_1$$

oppure imponendo la condizione di passaggio di r per P2(x2,y2); risulta in tal caso :

$$p = y_2 - m \cdot x_2$$

Nel caso in cui risulti x1=x2 i due punti P1 e P2 risultano allineati sulla stessa verticale (l'angolo formato fra la congiungente P1 con P2 e il semiasse positivo x risulta pari a 90 gradi il coefficiente angolare non può essere calcolato). In tal caso l'equazione della retta è del tipo:

$$x = x_1 \quad \text{oppure} \quad x = x_2$$

Osservando il "flow-chart" si nota che per prima cosa vengono fatte entrare le coordinate dei due punti P1 e P2; successivamente si provvede a verificare se i due punti sono disposti su di una retta parallela all'asse x, in tal caso si provvede ad eseguire l'output dell'ascissa comune. Se i due punti hanno una ascissa differente, in primo luogo viene calcolato il coefficiente angolare m della retta e successivamente viene calcolato il valore di p; i risultati della elaborazione vengono poi messi anche in questo caso in output sul terminale video.

Traduzione in linguaggio PASCAL

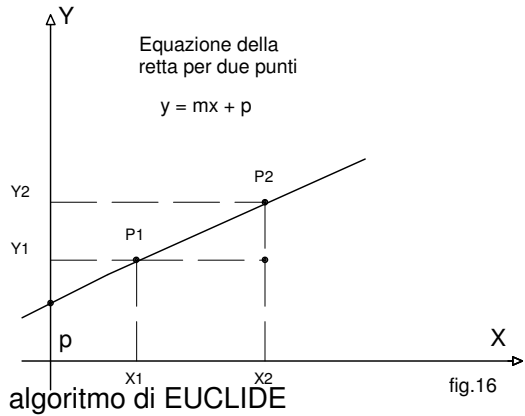
{ Questo programma determina l'equazione di una retta passante per due punti scritta nella forma : y = mx + p a partire dalle coordinate dei due punti P1(X1, Y1) e P2(X2, Y2). E' prevista una verifica per appurare se la retta è parallela all'asse y poiché in tal caso la retta ha equazione : x = X1 (o x = X2). }

*Program Retta(input,output);
var
X1,Y1,X2,Y2,M,P : real;*

*Begin
Write('Coordinate punto 1 : X1, Y1 ');
Readln(X1, Y1);
Write('Coordinate punto 2 : X2, Y2 ');
Readln(X2, Y2);
If (X1 = X2) Then Writeln(' Retta parallela asse y , X=',X1:5:2)
Else
Begin
M := (Y2-Y1)/(X2-X1);
P := Y1 - M*X1;
Writeln('Y=',M:5:2, 'x + (',P:5:2, ')');
End;
Writeln('Premere <return> per proseguire');
Readln;
End.*

Algoritmo di Euclide per il calcolo del M.C.D. (massimo comune divisore)

Si ricorda per per MCD fra due o più numeri interi positivi maggiori di 1 di cui sia stata effettuata la scomposizione in fattori primi, si intende quel



MCD algoritmo di EUCLIDE

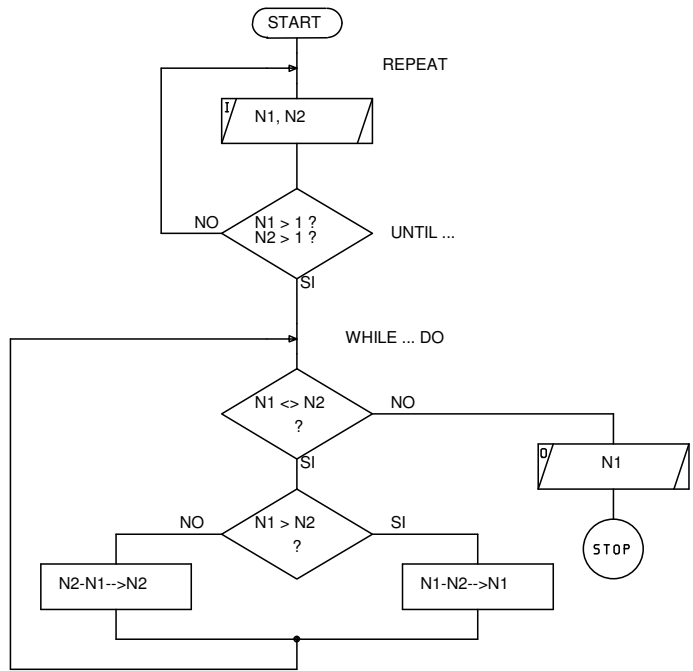


fig. 18

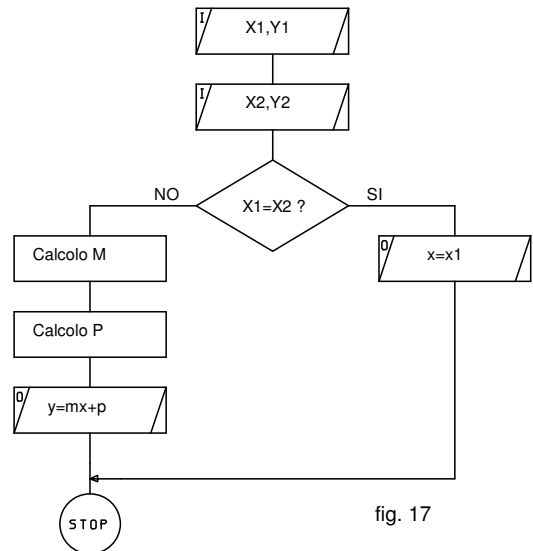


fig. 17

valore intero ottenuto moltiplicando i fattori comuni presi una sola volta con esponente minimo.

Il processo iterativo di ricerca del MCD é basato sulle ripetute sottrazioni fra il numero più grande e quello più piccolo finchè i due risultati si eguagliano.

La struttura iterativa REPEAT UNTIL che compare all'inizio del flow-chart é stata introdotta per eseguire l'input dei due dati N1, N2 e per controllare che tali dati siano maggiori entrambi della quantità 1.

*{ Algoritmo di Euclide per la determinazione del Massimo Comune
Divisore fra due numeri interi N1 ed N2 positivi e maggiori di 1. }*

Program MCD(input,output);

var

N1,N2 : integer;

Begin

Writeln(' Algoritmo di Euclide per la determinazione del MCD ');

N1 := 0;

While (N1<1) or (N2<1) do

Begin

Write('N1= '); Readln(N1);

Write('N2= '); Readln(N2);

End;

While N1 <> N2 do

Begin

If N1>N2 Then N1 := N1-N2

Else N2 := N2-N1;

End;

Writeln('MCD = ',N1);

End.

Algoritmo di Euclide per il calcolo del m.c.m. (minimo comune multiplo)

Si ricorda per per mcm fra due o piú numeri interi positivi maggiori di 1 si intende quel valore intero ottenuto considerando i fattori primi comuni e non comuni presi una sola volta con esponente massimo.

La struttura iterativa REPEAT UNTIL che compare all'inizio del flow-chart é stata introdotta per eseguire l'input dei due dati N1, N2 e per controllare che tali dati siano maggiori entrambi della quantità 1.

I dati N1 ed N2 che rispondono ai requisiti detti vengono poi copiati in due zone memoria ausiliarie D1 e D2.

Il processo iterativo di ricerca del mcm é basato proprio sulla definizione di mcm: infatti si procede a individuare fra N1 ed N2 il minore; si costruiscono i multipli del minore fra N1 e N2 e ci si arresta quando i due valori sono identici.

{ Algoritmo per la determinazione del minimo comune multiplo fra due numeri interi N1 ed N2 positivi e maggiori di 1. }

Program MCM(input,output);

var

N1,N2,D1,D2 : integer;

Begin

WriteLn(' Algoritmo per la determinazione del m.c.m. ');

N1 := 0;

While (N1<1) or (N2<1) do

Begin

Write('N1= ');

ReadLn(N1);

Write('N2= ');

ReadLn(N2);

End;

D1 := N1;

D2 := N2;

While N1 <> N2 do

Begin

If N1>N2 Then N2 := N2+D2

Else N1 := N1+D1;

End;

WriteLn('mcm = ',N1);

Write('Premere RETURN per continuare');

ReadLn ;

End.

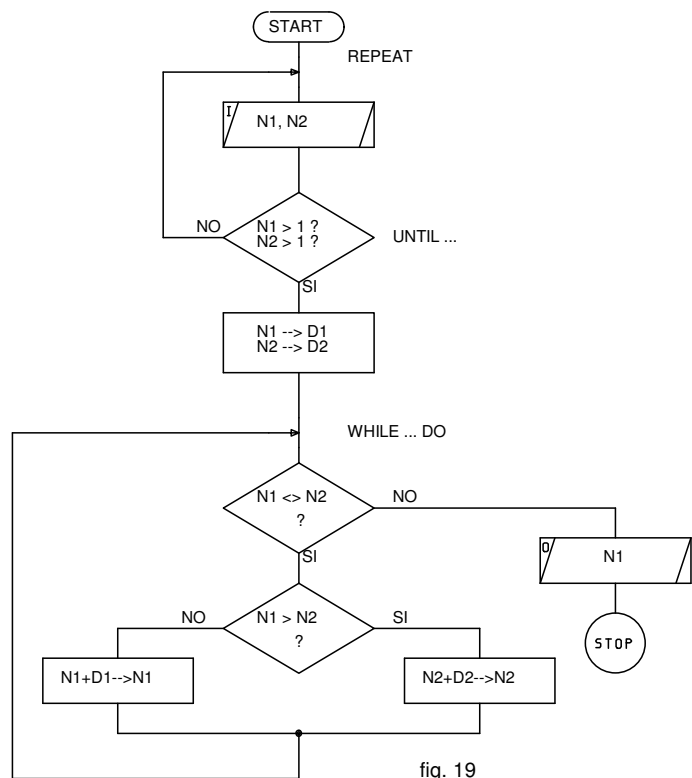


fig. 19