FreeBASIC (autore: Vittorio Albertoni)

Premessa

Il linguaggio BASIC nasce nel 1964 presso l'Università di Dartmouth, ad opera di John Kemeney e Thomas Kurtz, per facilitare l'accesso al mainframe GE-225 anche a utenti non particolarmente dotati di conoscenze informatiche: tant'è vero che il nome del linguaggio è l'acronimo di **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode (codice di istruzione simbolica di uso generale per principiante).

Nel 1975, quando viene messo in vendita il primo computer da tavolo ad uso personale, il MITS Altair 8800, Bill Gates e Paul Allen creano il linguaggio Altair BASIC per programmarlo, derivandolo dal BASIC di Dartmouth. Il successo è tale che, nello stesso anno, i due danno vita alla Microsoft e l'Altair BASIC diviene Microsoft BASIC, proprietà intellettuale della Microsoft.

Anche i primi adattamenti sviluppati dalla Commodore (per VIC 20 e Commodore 64) e dalla Apple avvengono su licenza Microsoft.

Altra ondata di successo coincide con il lancio del PC IBM 5150 del 1981, che incorpora un dialetto del BASIC chiamato BASICA: subito dopo, nel 1983, partendo da BASICA la Microsoft sviluppa una nuova versione del BASIC, chiamata GW-BASIC, che decreta il predominio del BASIC Microsoft per la programmazione dei personal computer[1].

Al GW-BASIC fanno seguito rilasci di altre versioni, sempre da parte di Microsoft.
Nel 1985 esce QuickBASIC, che, per la prima volta nella storia del BASIC, incorpora un compilatore per eseguibili DOS. Tutte le versioni precedenti richiedono un interprete per essere eseguite.
Nel 1991 esce QBASIC, versione ridotta di QuickBASIC, senza compilatore ma, nel frattempo, viene rilasciata la prima versione di Visual BASIC, cavallo di battaglia della Microsoft fino alla versione 6 del 1998 ed è tuttora possibile programmare in Visual BASIC su Visual Studio per il nuovo ambiente .NET Framework.

FreeBASIC nasce come software libero nel 2004 ad opera di una equipe guidata da Andre Victor ed è distribuito con licenza libera GNU GPL.
Ha una sintassi molto simile a quella di QuickBASIC, con cui ha un alto grado di compatibilità. E' un linguaggio compilato ed è utilizzabile sui sistemi Microsoft Windows, DOS (in modalità protetta), Linux e FreeBSD.

In questo manualetto mi propongo di presentare le caratteristiche principali del linguaggio, utili per una sua conoscenza di base, per la produzione di programmi di non grande impegno e per avere la preparazione necessaria ad accedere senza fatica alla documentazione ufficiale in vista della realizzazione di cose più impegnative.

A questo proposito devo ricordare che il BASIC è un linguaggio vecchio e, anche nelle sue riedizioni più recenti, dimostra tutti gli anni che ha: per realizzare cose impegnative, oggi come oggi, non mi rivolgerei al BASIC, che è nato e rimane una cosa per «beginners».

Nel tempo è stato arricchito per fare cose che vadano oltre il due più due, FreeBASIC ha introdotto elementi per la programmazione a oggetti, esistono librerie di arricchimento, ma per le cose un tantino impegnative diventa anche molto complicato e non è detto che ci consenta di raggiungere l'obiettivo.

[1] Il nome GW-BASIC fu scelto personalmente da Bill Gates e nessuno conosce l'origine del GW. C'è chi dice sia un omaggio a Greg Whitten, il dipendente Microsoft che più si dedicò al progetto. C'è chi dice voglia alludere alle iniziali dei cognomi Gates e Whitten. Altra interpretazione vuole che si tratti delle iniziali delle parole Gee Whiz (perbacco) con cui Bill Gates accolse favorevolmente il risultato del lavoro di progettazione: come a dire perbacco! che basic!, alludendo alla ricchezza del linguaggio.

Indice

1 Installazione **3**

2 Come funziona: il primo programma **4**

3 Tipi base **4**

4 Variabili e costanti **6**

5 Operatori **7**
5.1 Operatori aritmetici 7
5.2 Operatori di confronto 7
5.3 Operatori logici 8

6 Subroutines e funzioni **8**

7 Funzioni precostituite **9**
7.1 per interfacciarsi con lo schermo 9
7.2 per interfacciarsi con la tastiera 9
7.3 per lavorare con i file 9
7.4 per la matematica 11

8 Interattività con l'utente **12**

9 Strutture di controllo **12**
9.1 Esecuzione condizionale 12
9.2 Ripetizione 14

10 Grafica **15**
10.1 Creare una finestra grafica 15
10.2 Disegnare 17
10.3 Tracciare 21
10.4 Rispondere al mouse 23
10.5 Rispondere alla tastiera 23
10.6 Creare movimento 25

11 Estensioni **26**

1 Installazione

FreeBASIC si trova su *https://www.freebasic.net/*.

Cliccando sul pulsante

apriamo la pagina per il download, dove troviamo i file binari per Linux e per Windows e, in Documentation, il manuale ufficiale completo in vari formati.

Linux

Il file più aggiornato che attualmente (febbraio 2020) possiamo scaricare è
`FreeBASIC-1.07.1-linux-x86_64.tar.gz` per sistemi a 64 bit, oppure
`FreeBASIC-1.07.1-linux-x86.tar.gz` per sistemi a 32 bit.
Se lavoriamo su Ubuntu possiamo trovare i file binari appositamente creati per Ubuntu 14.04, 16.04 e 18.04.

Decomprimiamo il file scaricato, ci posizioniamo nella directory in cui l'abbiamo decompresso con il terminale e digitiamo, come super-utenti (`sudo` o `su`, a seconda del sistema su cui lavoriamo), il comando `./install.sh -i`, con che installiamo il compilatore nella cartella di sistema `/usr/local`.

Purtroppo questa installazione non installa tutte le dipendenze di cui c'è bisogno per un perfetto funzionamento del compilatore.

Queste dipendenze sono:
per Debian/Ubuntu:

```
gcc
libncurses5-dev
libffi-dev
libgl1-mesa-dev
libx11-dev
libxext-dev libxrender-dev libxrandr-dev libxpm-dev
```

per Fedora:

```
gcc
ncurses-devel
libffi-devel
mesa-libGL-devel
libX11-devel libXext-devel libXrender-devel libXrandr-devel libXpm-devel
```

per OpenSUSE:

```
gcc
ncurses-devel
libffi-devel
xorg-x11-devel
```

Dobbiamo pertanto verificare che questi file siano installati con il gestore dei programmi (meglio ancora con Synaptic) e installare i mancanti.

Windows

Il file più aggiornato che attualmente (febbraio 2020) possiamo scaricare è:
`FreeBASIC-1.07.1-win64.zip` per sistemi a 64 bit,
`FreeBASIC-1.07.1-win32.zip` per sistemi a 32 bit.
In alternativa, per sistemi a 32 bit, abbiamo anche un installatore .exe, ma è meglio non usarlo.

Decomprimiamo il file .zip in `C:\`, dove comparirà così una directory `FreeBASIC-1.07.1-winXX` contenente il compilatore e inseriamo il relativo percorso nella variabile di sistema PATH, in modo da poter lanciare il compilatore da qualsiasi directory.

2 Come funziona: il primo programma

Le istruzioni che compongono il programma si scrivono con un qualsiasi editor di testo e si salva il file con estensione .bas.

Si apre il terminale (in Windows si chiama prompt dei comandi), ci si posiziona nella directory dove è stato salvato il file e si invoca il compilatore con il comando `fbc` seguito dal nome del file .bas: nella stessa directory troveremo l'eseguibile compilato, in Linux denominato come il file .bas senza estensione, in Windows denominato come il file .bas ma con estensione .exe.

Il più semplice programma, nella tradizione del Ciao mondo, può essere questo:

```
print "Ciao mondo"
```

che, con la compilazione, genera un eseguibile lanciando il quale vediamo comparire a terminale la scritta Ciao mondo.

Se lavoriamo in Windows e vogliamo lanciare l'eseguibile semplicemente cliccandoci sopra, dobbiamo aggiungere alla fine del listato del programma l'istruzione `sleep`, che mantiene aperto il terminale fino a quando premeremo un tasto qualsiasi: in caso contrario il terminale scompare immediatamente dopo il lancio del programma senza consentirci di leggere quanto vi è scritto.

Bene sapere che il compilatore FreeBASIC è case insensitive, cioè non distingue tra lettere maiuscole e lettere minuscole: l'istruzione `print` potrebbe anche essere scritta `Print`, `PRINT`, `pRinT` e il compilatore la riconoscerebbe in ogni caso.

Per inserire commenti nel listato del programma, cioè scritte che non debbano influire sulla compilazione ma servano solo per chiarire certi passaggi ad un lettore umano, possiamo usare i seguenti simboli:

`'` (apice), inserito sulla stessa riga dopo altre istruzioni definisce un commento fino alla fine della riga;

`Rem`, inserito all'inizio di una riga definisce un commento per tutta la riga;

tra i simboli `/'` e `'/` definiamo un commento su più righe o sulla stessa riga all'interno di una serie di istruzioni.

Per quanto riguarda la scrittura del listato del programma, che possiamo fare con un qualsiasi editor di testo (editor di testo e non word processor), è raccomandabile usare un editor pensato per la programmazione e che ci offra la possibilità di compilare il programma dallo stesso editor, semplicemente cliccando su un pulsante.

Un efficiente editor di questo tipo è Geany che troviamo all'indirizzo

https://www.geany.org/download/releases/

e che chi usa Linux può installare facilmente attraverso il gestore dei programmi.

Chi usa Windows può trovare un altro editor all'indirizzo

https://fbide.freebasic.net/download,

che si installa con lo stesso meccanismo visto prima per l'installazione del compilatore e che ha il grande vantaggio di contenere nell'help il manuale ufficiale di FreeBASIC, purtroppo in lingua inglese.

3 Tipi base

Il linguaggio FreeBASIC è fortemente tipizzato e tutti i valori su cui agisce un programma devono avere un tipo. Il tipo assegnato ad un valore identifica la natura di quel valore, le operazioni che si possono effettuare su di esso e le modalità con le quali esso viene inserito nella memoria del computer.

Come tutti i linguaggi compilati, il linguaggio FreeBASIC è a tipizzazione statica, cioè richiede che il tipo di ciascun valore venga stabilito nel codice sorgente[2].

Qui vediamo i principali tipi previsti dal linguaggio.

Tipi numerici

I tipi che possiamo assegnare ad un valore numerico sono i seguenti.

Intero

Per i numeri interi, quelli che non hanno decimali, FreeBASIC prevede le seguenti alternative:
. `Byte` e `UByte` con ampiezza di 8 bit,
. `Short` e `UShort` con ampiezza di 16 bit,
. `Long` e `Ulong` con ampiezza di 32 bit
. `Integer` e `UInteger` con ampiezza di 32 bit o 64 bit, a seconda del sistema,
. `LongInt` e `ULongInt` con ampiezza di 64 bit.

L'iniziale `U` identifica tipi per numeri senza segno (unsigned), cioè solo positivi.

Per avere un'idea delle dimensioni massime del numero corrispondente ai vari tipi basta elevare 2 alla potenza corrispondente all'ampiezza in bit. Ciò fornisce la dimensione del tipo Unsigned. Per la dimensione del tipo Signed (senza la U iniziale) la dimensione del tipo Unsigned va dimezzata, metà con segno negativo e metà con segno positivo.
Esempio:
Per sapere la dimensione massima del tipo UShort a 16 bit eleviamo 2 alla sedicesima e otteniamo 65536.
Dal momento che in informatica si parte sempre da 0 la dimensione massima del tipo UShort è 65535 e il numero rappresentato potrà variare tra 0 a 65535.
Per sapere la dimensione massima del tipo Short, con segno, dimezziamo 65536 e otteniamo 32768 e il numero rappresentato potrà variare tra -32768 e 32767.

Decimale

I decimali sono i numeri reali, in informatica numeri a virgola mobile (floating point numbers) e sono quelli che contengono una parte decimale.

L'alternativa prevista da FreeBASIC è la seguente:
. `Single`, con ampiezza di 32 bit (precisione singola),
. `Double`, con ampiezza di 64 bit (precisione doppia).

Precisione singola significa avere 8 cifre significative, tra parte intera e parte decimale, precisione doppia significa averne 17, virgola compresa.

Tipo stringa

La stringa è una successione di caratteri e corrisponde all'identificativo `string`.

Il suo contenuto si crea scrivendo i caratteri all'interno di una coppia di doppi apici ".

Se ne può fissare la lunghezza facendo seguire all'identificativo il numero di caratteri preceduto da asterisco.
`string *4` identifica una stringa di 4 caratteri.

I caratteri contenuti in una stringa possono essere cifre. In questo caso possiamo convertire la stringa in un vero e proprio tipo numerico con l'istruzione
`val(<stringa>)`
il numero che ne deriva è di tipo double.

Al contrario possiamo anche convertire un numero o una espressione numerica in una stringa con l'istruzione

[2]Nei linguaggi interpretati, soprattutto nei linguaggi di scripting (come Python, Javascript, Perl, Ruby, ecc.), la tipizzazione è dinamica, cioè può avvenire anche durante l'esecuzione del programma.

`str(<espressione_numerica>)`.

Tipo booleano

L'identificativo è `Boolean` ed ha ampiezza di 1 bit.

4 Variabili e costanti

Le variabili e le costanti sono delle locazioni di memoria, delle scatole, alle quali diamo un nome, destinate a contenere valori di un certo tipo.

Nel primo programma visto nel Capitolo 2 abbiamo usato l'istruzione `print` per visualizzare il saluto Ciao mondo.
In quel caso abbiamo indicato all'istruzione una stringa da visualizzare.
Nello stesso modo potremmo indicare un numero da stampare o anche un'espressione numerica, scritta utilizzando gli operatori che vedremo, per visualizzarne il risultato.
Il tutto utilizzando elementi volanti.

Se ci limitassimo a questo non andremmo lontano con i nostri programmi.

In un programma che debba fare qualche cosa in più di quattro conticini, che debba sviluppare algoritmi complessi nel corso dei quali siano da prendere decisioni su cosa fare in presenza di un valore piuttosto che di un altro è necessario tenere a disposizione i valori, poterli modificare e richiamare quando serve. A questo servono le variabili e le costanti.

Variabili

La variabile è una locazione di memoria destinata a contenere un valore modificabile nel corso del programma.

Essa va creata dichiarandone il nome e dichiarando il tipo di valore a lei destinato, eventualmente indicando anche un valore iniziale.

La sintassi per la dichiarazione di una variabile è:
`dim` <nome> `as` <tipo>

Per dichiarare più variabili dello stesso tipo possiamo scrivere
`dim as` <tipo> <nome>, <nome>, ...

Nel caso si voglia anche inizializzare la variabile, cioè assegnarle un valore iniziale:
`dim` <nome> `as` <tipo> `=` <valore>.
Se scriviamo `dim x as long` creiamo la variabile denominata x destinata a contenere un intero con ampiezza 32 bit con segno.
Se scriviamo `dim raggio as single = 7.56` creiamo la variabile raggio (per esempio di un cerchio) destinata a contenere un numero decimale con precisione singola e le assegniamo il valore iniziale 7, 56 (attenzione alla virgola, che in FreeBASIC è un punto).
Se scriviamo `dim saluto as string = "Ciao"` creiamo la variabile saluto e le assegniamo il valore iniziale Ciao.
Se scriviamo `dim nome as string` *10 creiamo la variabile nome destinata a contenere una stringa con un massimo di 10 caratteri.

Per i nomi possiamo utilizzare qualsiasi parola che non sia riservata al linguaggio.

Per il tipo possiamo indicare uno di quelli visti nel Capitolo 3.

Il valore può essere un numero, una espressione numerica, più o meno includente altre variabili, o una stringa (in tal caso sarà una successione di caratteri racchiusa tra apici).

Se quando dichiariamo una variabile vogliamo anche inizializzarla possiamo usare la seguente sintassi:
`var` <nome> `=` <valore>
In questo caso il tipo viene assegnato automaticamente in base al valore indicato.

Il programmino che abbiamo scritto nel Capitolo 2, se vi introduciamo l'uso di una variabile, diventa

```
dim saluto as string = "Ciao mondo"
print saluto
```
oppure, con la sintassi alternativa per le variabili inizializzate:
```
var saluto = "Ciao mondo"
print saluto
```
Come si vede, abbiamo innanzi tutto creato una variabile stringa che contiene la frase Ciao mondo e poi abbiamo detto a `print` di stampare quella variabile richiamandone il nome.
Il risultato è lo stesso che si ottiene con la scrittura del programma senza l'uso della variabile, ma ora abbiamo realizzato il vantaggio di avere nel programma una variabile utilizzabile in altre parti del programma semplicemente richiamandone il nome.

Costanti

La costante è praticamente una variabile il cui contenuto non può cambiare.

La sintassi per la dichiarazione di una costante è

`const` <nome> = <valore>

ove, per nome e valore vale quanto detto per le variabili nel precedente paragrafo.

Con questa sintassi il tipo viene automaticamente stabilito in base al valore indicato.

5 Operatori

Gli operatori collegano tra loro operandi di varia natura in espressioni che forniscono un risultato.

Quelli di uso più comune per i principianti sono i seguenti.

L'elenco completo degli operatori di FreeBASIC si trova nel manuale ufficiale in Language Documentation ▷ Operators List.

5.1 Operatori aritmetici

`+` per la somma tra numeri o stringhe
`-` per la sottrazione tra numeri
`*` per la moltiplicazione tra numeri
`/` per la divisione tra numeri
`\` per la divisione intera
`mod` per il modulo (resto della divisione intera)
`^` per l'elevamento a potenza

5.2 Operatori di confronto

Servono per confrontare due valori e il risultato che restituiscono è un valore booleano.

Sono i seguenti.

`=` uguale,
`<>` non uguale,
`<` minore,
`<=` minore o uguale,
`>` maggiore,
`>=` maggiore o uguale.

Gli operandi assoggettati al confronto devono avere lo stesso tipo.

5.3 Operatori logici

Forniscono come risultato un valore booleano e sono i seguenti.
`and`,
`or`,
`not`.

6 Subroutines e funzioni

Il linguaggio FreeBASIC è un linguaggio procedurale, cioè il programma scritto in FreeBASIC altro non è che una serie di istruzioni che il computer deve eseguire una via l'altra.

Può accadere che, nel corso del programma, vi siano delle cose che vanno eseguite parecchie volte e in questi casi conviene raggruppare in blocchi le istruzioni per eseguirle in modo che, tutte le volte che serve, si possa semplicemente richiamare il blocco anziché scrivere ogni volta le relative istruzioni.

Questi blocchi sono praticamente delle sotto-procedure e si definiscono
. subroutines se eseguono compiti senza restituire valori,
. funzioni se restituiscono un valore.

Subroutines

La sintassi per costruire una subroutine è
`sub` <identificativo>
    <istruzioni>
`end sub`
dove <identificativo> è il nome che diamo alla subroutine per poterla eseguire quando serve e <istruzioni> sono le istruzioni da eseguire.

Quando e dove serve otteniamo l'esecuzione delle istruzioni semplicemente scrivendo il nome dell'identificativo assegnato.

La subroutine deve essere stata costruita prima di richiamarla.

Funzioni

La sintassi per costruire una funzione è
`function` <identificativo> `(`<argomenti>`) as` <tipo_dato_restituito>
    <istruzioni>
`end function`
dove <identificativo> è il nome che diamo alla funzione per poterla eseguire quando serve, <tipo_dato_restituito> è il tipo del dato che la funzione deve ritornare e <istruzioni> sono le operazioni da eseguire per ottenere quel dato.
Tra le istruzioni non può mancare l'istruzione `return`, destinata ad indicare il dato che la funzione deve ritornare.
Se per eseguire la funzione occorre indicare uno o più argomenti essi vanno indicati al posto di <argomenti> con nome e tipo, separati da virgola se sono più di uno, con la sintassi <nome> `as` <tipo>.

Quando e dove serve otteniamo il dato ritornato semplicemente scrivendo il nome dell'identificativo assegnato alla funzione e indicando tra parentesi tonde gli eventuali argomenti richiesti, separati da virgola se più di uno.

La funzione deve essere stata costruita prima di richiamarla.
Esempio:
Una funzione per calcolare l'area del triangolo può essere scritta così:

```
function area_triangolo (b as double, h as double) as double
return b * h / 2
end function
```

Per calcolare e scrivere l'area di un triangolo di base 5 e altezza 4 diamo l'istruzione
`print area_triangolo(5, 4)`
ed otteniamo scritto il risultato 10.

7 Funzioni precostituite

FreeBASIC contiene una miriade di funzioni già predisposte per fare determinate cose.

Qui non le esamineremo tutte ma solo quelle più ricorrenti per predisporre programmi alla portata di dilettanti.

Nel manuale ufficiale, sotto la voce Runtime Library Reference, troviamo tutte le funzioni suddivise per argomento. Cliccando sui loro nomi possiamo vedere a cosa servono e come si usano.

7.1 per interfacciarsi con lo schermo

FreeBASIC può interfacciarsi con l'utente semplicemente attraverso il terminale (quello che in Windows si chiama prompt dei comandi) o in modalità grafica, cioè attraverso una finestra sulla quale possiamo scrivere e disegnare, utilizzare colori, ecc.

Della grafica ci occuperemo in un capitolo a parte; per ora accontentiamoci del terminale.

L'unica funzione precostituita che ci interessa per lavorare su terminale è `print`, che già abbiamo utilizzato nel Capitolo 2 per il nostro primo programmino.

Questa istruzione scrive ciò che indichiamo dopo di essa. Se indichiamo una stringa (caratteri tra doppi apici) viene scritta tale e quale. Se indichiamo un numero o un'espressione matematica viene scritto il numero o il risultato dell'espressione matematica, automaticamente e internamente convertiti in stringa.

Possiamo anche indicare una lista di cose da scrivere sulla stessa riga, separando queste cose con un punto e virgola (`;`) o con una virgola (`,`).
Se usiamo il punto e virgola le cose vengono scritte una via l'altra, senza spazi intermedi.
Se usiamo la virgola le cose vengono scritte tabulate in spazi di 14 caratteri.

Se vogliamo risparmiare in battiture bene sapere che l'istruzione `print` può essere sostituita da un semplice punto interrogativo (`?`).

7.2 per interfacciarsi con la tastiera

La principale funzione di questa categoria è quella per acquisire dati dalla tastiera per inserirli in una variabile: `input`.

La sua sintassi è:
`input` <stringa_messaggio>`,` <variabile>
dove <stringa_messaggio> serve per formulare la richiesta di immissione del dato da parte dell'utente e <variabile> è il nome della variabile ove inserire il dato immesso dall'utente. La variabile deve ovviamente essere stata precedentemente dichiarata.

Altra funzione che può tornare utile per reagire alla pressione di un tasto da parte dell'utente è `inkey`, che ritorna una stringa rappresentante il carattere del tasto premuto.

7.3 per lavorare con i file

A volte può essere utile o necessario che il nostro programma salvi su disco determinate informazioni per archiviarle e renderle disponibili per utilizzi successivi.

A questo scopo si crea un file in cui inserire o da cui estrarre informazioni.

Le funzioni di base per creare, alimentare e leggere il contenuto sono le seguenti:

`open` è la funzione che apre un file (se ancora non c'è lo crea) per lavorarci. Con essa si deve indicare il nome del file, il motivo per cui lo si apre e numerarlo. La sintassi della funzione è
`open` <nome_file> `for` <motivo> `as #`<numero>

dove
<nome_file> è una stringa contente il nome del file con il percorso per raggiungerlo,
<motivo> è rappresentato da una delle seguenti tre parole:
`output` se nel file si devono indirizzare dati dal programma,
`append` se al file si devono aggiungere dati dal programma,
`input` se dal file si devono estrarre dati per il programma,
<numero> serve per fare riferimento al file per lavorarci;

`close` è la funzione che chiude il file e che consolida il lavoro fatto su di esso. La sintassi della funzione è
`close #<numero>`
dove <numero> è il numero che avevamo assegnato al file con l'istruzione `open`.

`print` è la funzione con la quale inseriamo dati nel file. La sintassi è:
`print #<numero>, <dati>`
dove
<numero> è il numero che avevamo assegnato al file con l'istruzione `open`,
<dati> è ciò che vogliamo inserire nel file, come stringa tra apici o numero.

`line input` è la funzione con la quale estraiamo dati dal file. La sintassi è:
`line input #<numero>, <variabile_destinazione>`
dove
<numero> è il numero che avevamo assegnato al file con l'istruzione `open`,
<variabile_destinazione> è una variabile, in cui inserire ciò che estraiamo dal file;
la variabile deve essere stata precedentemente dichiarata.
La funzione `line input` legge usa sola riga del file; per leggere un intero file con più righe occorre inserirla in un ciclo (secondo uno dei metodi che vedremo in capitolo dedicato).

eof è la funzione che identifica la fine del file e restituisce `true` (`1`) alla fine del file. La sintassi è:
`eof(<numero>)`
dove <numero> è il numero che avevamo assegnato al file con l'istruzione `open`;
questa funzione può servire, per esempio, per chiudere il ciclo di cui abbiamo appena parlato.

Un esempio può essere utile. Il seguente programma

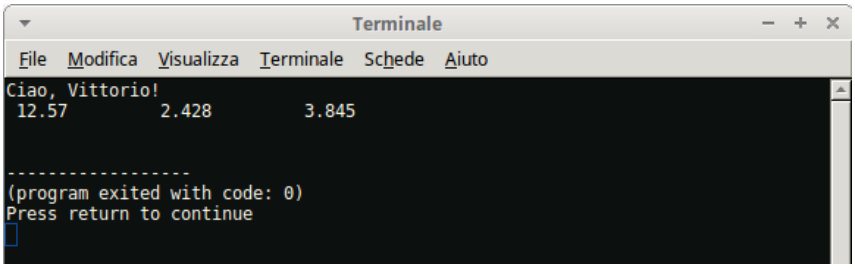
```
dim file as string = "/home/vittorio/Documenti/prova.txt"


open file for output as #1
print #1, "Ciao, Vittorio!"
close #1


open file for append as #1
print #1, 12.57, 2.428, 3.845
close #1


open file for input as #1
do until eof(1)
  dim contenuto as string
  line input #1, contenuto
  print contenuto
loop
close #1
```

produce il seguente risultato



7.4 per la matematica

Quando si richiamano queste funzioni va indicato tra parentesi tonde l'argomento su cui esse devono agire.
L'argomento può essere un numero, una variabile contenente un numero o un'espressione matematica, nel qual caso la funzione agisce sul risultato di questa espressione.

algebriche

`abs()` ritorna il valore assoluto
`exp()` ritorna una potenza del numero e
`log()` ritorna il logaritmo naturale
`sqr()` ritorna la radice quadrata
`fix()` ritorna la parte intera
`frac()` ritorna la parte frazionaria
`int()` ritorna il più alto intero inferiore o uguale
`sgn()` ritorna il segno

Dal momento che FreeBASIC non contiene una definizione preconfezionata della costante e all'occorrenza possiamo memorizzarne un valore ben approssimato con l'istruzione
`const e = exp(1)`.

trigonometriche

`sin()` ritorna il seno
`asin()` ritorna l'arcoseno
`cos()` ritorna il coseno
`acos()` ritorna l'arcocoseno
`tan()` ritorna la tangente
`atn()` ritorna l'arcotangente

Gli argomenti per le funzioni trigonometriche vanno espressi in radianti.

Dal momento che FreeBASIC non contiene una definizione preconfezionata della costante π all'occorrenza possiamo memorizzarne un valore ben approssimato con l'istruzione
`const pi = 2 * acos(0)`.

numeri casuali

`randomize()` insemina la generazione di numeri casuali da parte della funzione rnd. Se non si indica l'argomento l'inseminazione avviene sul valore del clock del computer.
`rnd` ritorna un numero casuale di tipo double compreso tra 0 e 1. Se si usa questa funzione senza prima aver invocato la precedente il numero ritornato sarà sempre quello.
Esempio:
Per scrivere un numero casuale compreso tra 0 e 100 possiamo usare questo codice:

```
randomize
print fix(rnd * 100)
```

8 Interattività con l'utente

Nel piccolo esempio «ciao mondo» che abbiamo visto nel Capitolo 2 e nella sua rivisitazione nel Capitolo 4 il programma conteneva già il dato da elaborare (la scritta "Ciao mondo" nel primo caso direttamente indicata e, nel secondo caso, indicata richiamando la variabile che la conteneva) e, una volta lanciato, forniva il risultato secondo quella scritta. Se ci accontentassimo di questo, per salutare qualche cosa che non sia il mondo ma sia un nostro amico, dovremmo prendere il programma che abbiamo scritto per salutare il mondo, ricopiarlo sostituendo alla parola "mondo" il nome del nostro amico, ricompilarlo e rilanciarlo. Così dovremmo fare per tutti gli amici che volessimo salutare.

Ma con quanto abbiamo visto nel precedente Capitolo 7 possiamo ora fare meglio e modificare il programma in modo che, una volta lanciato, chieda all'utente chi voglia salutare e il programma diventi finalizzato a rivolgere il saluto a chicchessia senza bisogno di essere riscritto tutte le volte.

Il programma diventa il seguente

```
dim nome as string
input "Come ti chiami?  ", nome
print "Ciao, "; nome; "!"
sleep
```

Lanciandolo saremo richiesti di indicare il nome di chi salutare e in un attimo vedremo il saluto rivolto a chi abbiamo indicato.

9 Strutture di controllo

Come ogni altro linguaggio di programmazione, FreeBASIC ha dei comandi per condizionare l'esecuzione di certe istruzioni al verificarsi di determinate condizioni oppure per la ripetizione dell'esecuzione di una o più istruzioni.

9.1 Esecuzione condizionale

if

L'istruzione `if`, chiamata istruzione di esecuzione condizionale (in inglese if è il nostro se), ci dà modo di assoggettare l'esecuzione di un blocco di istruzioni al verificarsi di una determinata condizione: se la condizione è vera viene eseguito il blocco di istruzioni, altrimenti si prosegue l'esecuzione del programma saltando il blocco stesso.

La sintassi è la seguente

```
if <condizione> then
   <istruzioni>
end if
```

dove <condizione> è una qualsiasi espressione che relaziona due valori attraverso operatori di confronto: se la condizione si verifica vengono eseguite la o le istruzioni indicate, altrimenti si passa oltre.

L'istruzione `if` si presta anche all'esecuzione condizionale a due rami. Per ottenere questo dobbiamo abbinarla all'istruzione `else` con questa sintassi

```
if <condizione> then
   <istruzioni>
else
   <istruzioni>
end if
```

In questo caso se la condizione si verifica vengono eseguite le istruzioni contenute nel primo blocco altrimenti vengono eseguite quelle contenute nel secondo blocco dopo else (altrimenti). In ogni caso proseguendo poi nell'esecuzione del resto del programma.

Possiamo infine gestire l'esecuzione condizionale a più rami abbinando a `if` l'istruzione `elseif` con questa sintassi

```
if <condizione> then
   <istruzioni>
elseif <condizione> then
   <istruzioni>
elseif <condizione> then
   <istruzioni>
elseif <condizione> then
   <istruzioni>
.....
```

e proseguire quanto vogliamo, chiudendo alla fine con `end if`.

select

Abbiamo appena visto come l'istruzione `if` possa essere applicata a una ramificazione multipla attraverso le istruzioni aggiuntive `elseif`.

L'istruzione `select` può convenientemente essere usata per semplificare la realizzazione della ramificazione multipla nel caso essa dipenda dal confronto tra diverse alternative che una variabile può assumere.

La sintassi è la seguente

```
select case <variabile>
   case <valore>
       <istruzione>
   case <valore>
       <istruzione>
   case <valore>
       <istruzione>
   .......
   case else
       <istruzione>
end select
```

dove <variabile> può essere di tipo intero o di tipo stringa e <valore> è uno dei valori che la variabile può assumere in corrispondenza al quale eseguire una certa istruzione.

Propongo un esempio nel quale il confronto è basato sul valore di una variabile stringa.

```
dim moneta as string
input "Scegli una moneta:  ", moneta
select case moneta
    case "euro"
        print "Europa"
    case "dollaro"
        print "U.S.A."
    case "sterlina"
        print "Regno Unito"
    case "yen"
        print "Giappone"
    case "yuan"
        print "Cina"
    case else
        print "non so"
end select
```

Il programma risponde indicando la zona geografica dove si usa una certa valuta indicata dall'utente e, anche se il programmatore si sforza di prevederle tutte, può sempre succedere che l'utente ne indichi una non prevista: al che il programma risponde «non so».

9.2 Ripetizione

for

Si usa quando si vuole ripetere l'esecuzione di una istruzione o di un blocco di istruzioni per un numero definito di volte.

La sintassi per l'uso di questa istruzione è

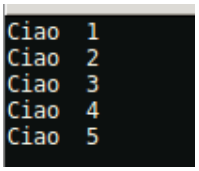
`dim` <contatore> `as` <tipo>
`for` <contatore> = <partenza> `to` <arrivo> [`step` <numero>]
    <istruzioni>
`next`

La variabile che ho battezzato `contatore`, per richiamarne la natura, in genere viene nominata semplicemente `i` e serve per contare le volte che viene eseguito il blocco di istruzioni contenuto tra le indicazioni <partenza> e <arrivo>: dopo le istruzioni, se ne prevede l'incremento di una unità e l'esecuzione termina dopo il raggiungimento del numero indicato come <arrivo>.
L'indicazione `step` è facoltativa e consente di scegliere come si debba scorrere il contatore. Se si omette l'indicazione, la conta avviene per unità (come dire `step 1`).

Questo piccolo programma

```
dim i as short
for i = 1 to 5
    print "Ciao"; " "; i
next i
```

produce il seguente risultato



while

Si usa per ripetere istruzioni fino a quando si verifica una certa condizione.

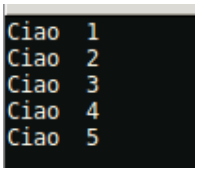
La sintassi è:

`while` <condizione>
    <istruzioni>
`wend`

Se la condizione è espressa attraverso l'uso di un contatore otteniamo gli stessi risultati che otteniamo con la funzione `for` vista prima

```
var i = 1
while i <= 5
   print "Ciao"; " "; i
i = i + 1
wend
```

produce il risultato



Ma sono possibili altri meccanismi.

Questo piccolo programma, per esempio, ritorna il carattere che corrisponde al tasto premuto sulla tastiera fino a quando non premiamo il tasto f minuscolo, nel qual caso il programma si arresta.

```
dim s as string
input "premi un tasto ", s
while s <> "f"
    print s
    input "premi un tasto ", s
wend
```

do

Si usa per ripetere istruzioni mentre o fino a quando si verifica una certa condizione.

La sintassi è:

`do while|until` <condizione>
<istruzioni>
`loop`

Le parole chiave `while` (mentre) e `until` (fino a quando) sono di uso alternativo.

Un'applicazione con `until` l'ho anticipata nel paragrafo 7.3 quando ho mostrato come leggere un file fino a quando se ne raggiunge la fine.

Un'applicazione con `while` può essere questo altro modo di ottenere ciò che abbiamo ottenuto con l'esempio precedente.

```
dim s as string
input "premi un tasto ", s
do while s <> "f"
    print s
    input "premi un tasto ", s
loop
```

10 Grafica

Il FreeBASIC ha una libreria grafica in 2D integrata, in parte compatibile con il QuickBASIC, che permette di gestire semplici primitive grafiche (linee e cerchi), il blitting e caratteristiche aggiuntive non presenti nella libreria grafica originale del QuickBASIC. La libreria non è dipendente dal sistema operativo per cui il codice è portabile da una piattaforma all'altra.

Per default FreeBASIC lavora su terminale non in modalità grafica e per lavorare in modalità grafica occorre creare una finestra grafica che sostituisca il terminale.

10.1 Creare una finestra grafica

La finestra grafica si crea con il comando **screen**.
La sintassi del comando è la seguente:
`screen` <modo>, <profondità_colore>
Ho indicato i due parametri più ricorrenti: altri ce ne sono per usi sofisticati e per quelli rimando al manuale.
<modo> rappresenta la risoluzione grafica,
<profondità_colore> rappresenta la risoluzione del colore.

risoluzione grafica

Si indica con un numero.

Prescindendo dalla possibilità di indicarla in modo antidiluviano con i numeri 1 e 2 in emulazione CGA, si indica con un numero compreso tra 7 e 21.

I numeri da 7 a 13 corrispondono, insieme ai due precedenti, alle risoluzioni grafiche del vecchio QuickBASIC e contemplano risoluzioni tra 320x200 pixel e 640x480 pixel in emulazione EGA e VGA ormai in disuso.

I numeri da 14 a 21 corrispondono alle risoluzioni originali del FreeBASIC e vanno da 320x240 pixel a 1280x1024 pixel.

Una risoluzione che ritengo ragionevole e più che sufficiente, soprattutto se lavoriamo su un portatile, è la 19, corrispondente a 800x600 pixel. Sapendo che la 20 corrisponde a 1024x768 pixel e la 21 corrisponde a 1280x1024 pixel.

E' possibile creare una finestra di dimensioni personalizzate con l'istruzione **screenres**, la cui sintassi è:
`screenres` <larghezza>, <altezza>, <profondità_colore>
dove, in luogo del <modo> previsto dall'istruzione `screen`, indichiamo larghezza e altezza, in pixel, della finestra che ci interessa. Se le dimensioni indicate non sono adatte al computer che stiamo utilizzando avremo messaggi di errore e dovremo cambiare la scelta.

risoluzione del colore

Questo argomento ha effetto solo per le risoluzioni grafiche da 14 a 21 e si indica con uno dei numeri 8, 16 o 32, che sono i bits per pixel destinati a descrivere il colore: con 8 bits arriviamo a differenziare 16 colori, con 32 bits arriviamo a sfumature che neanche Leonardo...

Se non indichiamo l'argomento, per default viene scelto il colore a 8 bits per pixel e potremo scegliere il colore indicandone un riferimento tabellare; se scegliamo 16 o 32 bits per pixel dovremo indicare il colore attraverso la codifica rgb.

Se non indichiamo né argomento né colore la finestra che creiamo con i comandi `screen` o `screenres` è a sfondo nero e accoglie scritte o disegni (primo piano) in bianco.

Per modificare la scelta di default abbiamo a disposizione il comando **color**, la cui sintassi è:
`color` <primo_piano>, <sfondo>
dove i due argomenti vengono indicati
. con un numero da 0 a 15 se non abbiamo scelto la risoluzione del colore o abbiamo scelto la risoluzione colore a 8 bits,
. con la funzione `rgb(`<rosso>, <giallo>, <blu>`)`, con <rosso>, <giallo> e <blu> indicati con numeri da 0 a 255 a seconda dell'intensità richiesta, se abbiamo scelto di lavorare con la risoluzione colore a 16 o 32 bits.

Non necessariamente dobbiamo indicare entrambi i parametri. Se vogliamo agire solo sul primo piano indichiamo un solo argomento. Se vogliamo agire solo sullo sfondo indichiamo un solo argomento preceduto da una virgola (,).

L'elenco dei 16 colori con risoluzione a 8 bits è il seguente:
0 nero
1 blu
2 verde
3 turchese
4 rosso
5 rosa
6 giallo
7 grigio
8 grigio scuro
9 blu brillante
10 verde brillante
11 turchese brillante
12 rosso brillante
13 rosa brillante
14 giallo brillante
15 bianco

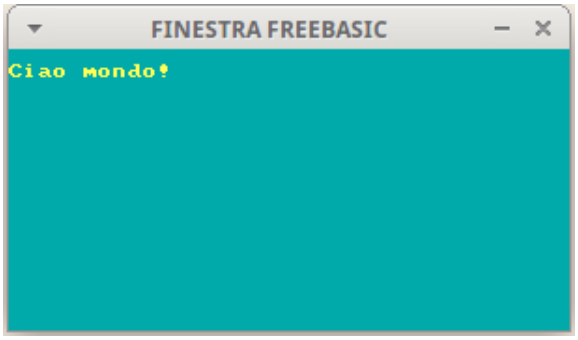
Per l'elenco di una lunga serie di colori corrispondenti a varie combinazioni rgb rimando alla voce Lista dei colori di Wikipedia.

Alla nostra finestra possiamo dare un titolo con l'istruzione `windowtitle(`<stringa_titolo>`)`.

Il seguente programmino

```
screenres 300, 150
color 14, 3
cls
windowtitle("FINESTRA FREEBASIC")
print
print "Ciao mondo!"
sleep
```

produce la seguente finestra grafica



I codici colore sono indicati con i riferimenti tabellari in quanto, non essendo stata specificata la risoluzione colore con l'istruzione `screenres`, è subentrata la risoluzione di default a 8 bits.

Richiamo l'attenzione sul fatto che, subito dopo aver inserito l'istruzione `color`, è necessaria l'istruzione `cls` (pulisci schermo), per ripulire lo schermo e ripristinarlo con i colori scelti. Inoltre è bene chiudere il programma con l'istruzione `sleep`, in quanto, come avviene in Windows, i programmi grafici partono con un semplice click sul loro nome anche in Linux e questa istruzione, come ho già avuto modo di spiegare, se lanciamo il programma in quel modo, lascia visibile la finestra fino a che non si preme un qualsiasi tasto; in caso contrario la finestra sparirebbe subito.

In questo modo, pur sempre interagendo con il computer attraverso la tastiera, possiamo eseguire tutti i nostri programmi in una finestra grafica anziché sul terminale.
In proposito chiarisco che la libreria grafica integrata in FreeBASIC non è strutturata in modo da produrre interfacce utente grafiche (GUI). A questo scopo possiamo ricorrere al software libero Glade/GTK che ben si integra con FreeBASIC, ovviamente avendo installato sul computer il software Glade e il toolkit GTK.
Altro chiarimento riguarda il fatto che la libreria grafica integrata in FreeBASIC è 2D, cioè bidimensionale. Per la tridimensionalità possiamo ricorrere a OpenGL, che ben si integra con FreeBASIC.

Ma vediamo cosa altro possiamo fare con la libreria integrata.

10.2 Disegnare

Ogni punto (pixel) della finestra grafica è identificato dalle sue coordinate: una in orizzontale, quella che in geometria analitica si usa chiamare x, ed una in verticale, quella che in geometria analitica si usa chiamare y.

Il punto di coordinate 0 e 0, che chiamiamo origine, è quello in alto a sinistra della finestra.

Attraverso le coordinate possiamo pertanto individuare le posizioni nella finestra e far funzionare alcuni strumenti di disegno rappresentati dalle seguenti istruzioni.

pset

Disegna un punto in una determinata posizione e con un determinato colore. La sintassi è:
`pset(x, y), <colore>`

dove x e y sono le coordinate orizzontale e verticale e <colore> è indicato secondo i modi visti nel precedente paragrafo. Se non si indica il colore il punto viene colorato di bianco.

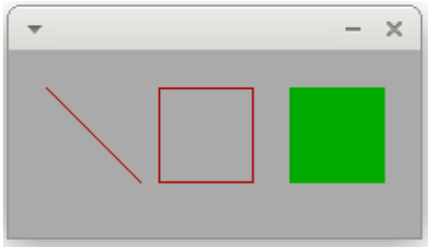
line

Disegna una linea (o un rettangolo) tra un punto della finestra e un altro. La sintassi è:
`line (x1, y1) - (x2, y2), <colore>`
dove (x1, y1) sono le coordinate del punto da dove parte la linea, (x2, y2) sono le coordinate del punto dove arriva la linea e <colore> è indicato secondo i modi visti nel precedente paragrafo. Se non si indica il colore la linea viene colorata di bianco.

Questa istruzione accetta un ulteriore argomento esprimibile con la lettera b, che sta per box o con le lettere `bf` che stanno per box filled.
Se indichiamo l'argomento `b` non verrà disegnata una linea ma un rettangolo con vertici opposti nei punti indicati e se indichiamo l'argomento `bf` il rettangolo verrà riempito del colore indicato.
Il seguente programmino

```
screenres 220, 100
color 4, 7
cls
line (20,20) - (70, 70)
line (80, 20) - (130, 70),, b
line (150, 20)- (200, 70), 2, bf
sleep
```

produce le seguenti figure



Notare la riga dove ho utilizzato l'argomento b, non indicando il colore in modo da usare il rosso già scelto come colore di primo piano: non avendo indicato il colore, ho dovuto inserire una virgola in modo da evidenziare come non utilizzata la posizione riservata al colore in quanto il compilatore, subito dopo l'indicazione dei punti per tracciare la linea, si aspetta l'indicazione o la non indicazione dell'argomento relativo al colore e se vi trova qualche cosa di diverso segnala errore e non procede nella compilazione.

circle

Disegna una circonferenza (o una ellisse o un arco) attorno ad un punto identificato come centro. La sintassi, con gli argomenti base per la circonferenza, è:
`circle (x, y), <raggio>, <colore>`
dove x e y sono le coordinate del centro, <raggio> è il raggio e <colore> è indicato secondo i modi già visti.

Immediatamente dopo questi argomenti possiamo indicarne altri due: <partenza> e <arrivo>, attraverso i quali disegnare, in senso anti-orario, un arco di circonferenza a partire dall'angolo indicato, in radianti, come <partenza> fino all'angolo indicato, sempre in radianti, come <arrivo>.

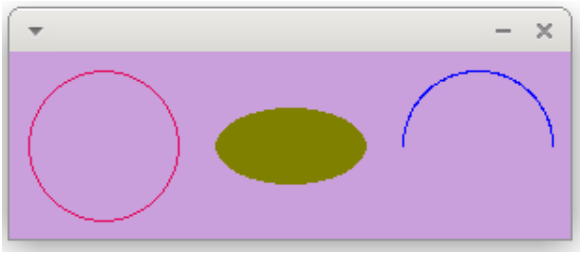
Immediatamente dopo questi ulteriori argomenti è eventualmente inseribile un altro argomento <aspetto> attraverso il quale possiamo disegnare un'ellisse schiacciando la circonferenza ideale in senso orizzontale, indicando un valore dell'argomento inferiore a 1, o verticale, indicando un valore dell'argomento superiore a 1.

Infine, dopo i posti riservati a tutti gli argomenti visti, c'è posto per un ultimo argomento, esprimibile con la lettera `f`, ad indicare se la figura chiusa (circonferenza o ellisse, non arco) debba essere riempita con il colore precedentemente indicato come colore di primo piano.

Questo programmino

```
screenres 300, 100, 32
color ,rgb(201, 160, 220)
cls
circle (50, 50), 40, rgb(227,11,92)
circle (150, 50), 40, rgb(128, 128, 0),,, 0.5, f
const pi = acos(-1)
circle (250, 50), 40, rgb (0, 0, 255), 0, pi
sleep
```

produce le seguenti figure



Notare come, avendo scelto la risoluzione colore a 32 bits, abbia dovuto indicare i colori con la codifica rgb (color glicine per lo sfondo, color lampone per la circonferenza, color verde oliva per l'ellisse e blu per l'arco di circonferenza).
Sempre ricordando che il compilatore prevede posti riservati ai vari argomenti: da qui, nell'istruzione per disegnare l'ellisse, i due spazi tra virgole ad indicare l'assenza degli argomenti <partenza> e <arrivo> che il compilatore si aspetta, in quelle posizioni, per l'eventuale disegno di un arco.

Il fastidio di dover ricordare tutti questi possibili argomenti con le relative posizioni è compensato dal fatto che, in FreeBASIC, con un paio di istruzioni possiamo disegnare praticamente di tutto.

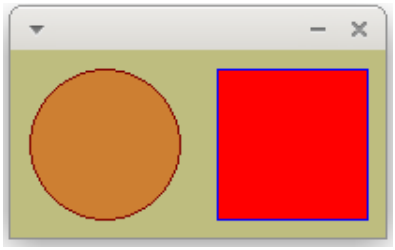
paint

Consente di colorare l'interno di una figura chiusa conservando un diverso colore per il bordo: cosa che non è possibile fare con l'opzione `f` che abbiamo visto prima. La sintassi è:
`paint (x, y), <colore_riempimento>, <colore_bordo>`
dove x e y sono le coordinate di un punto interno alla figura da colorare, <colore_riempimento> e <colore_bordo> vanno indicati nel solito modo, facendo attenzione che <colore_bordo> coincida con quello indicato per disegnare i contorni della figura.

Questo programmino

```
screenres 200, 100, 32
color ,rgb(190,189,127)
cls
circle (50, 50), 40, rgb(128,0,0)
paint (50, 50), rgb(205,127,50),rgb(128,0,0)
line (110, 10)-(190, 90), rgb(0,0,255),b
paint (121, 11), rgb(255,0,0), rgb(0,0,255)
sleep
```

produce le seguenti figure



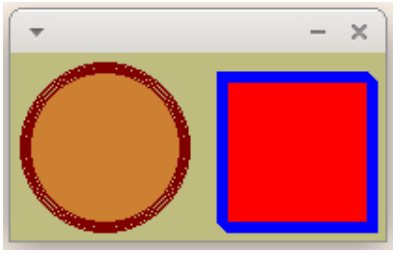
Abbiamo un beige verdastro per lo sfondo, un bordeaux per il bordo del cerchio, un bronzo per l'interno del cerchio, un blu per il bordo del quadrato e un rosso per l'interno del quadrato. Purtroppo i bordi non si vedono molto e, con FreeBASIC non abbiamo un modo diretto per dimensionare le linee.

Ma con un po' d'ingegno si arriva a tutto.

Se vogliamo che i contorni delle figure si vedano bene, per esempio con un bel tratto di 5 pixel, possiamo modificare così il nostro programmino:

```
screenres 200, 100, 32
color ,rgb(190,189,127)
cls
dim i as short
for i = 0 to 5
    circle (50, 50), 40+i, rgb(128,0,0)
next i
paint (50, 50), rgb(205,127,50),rgb(128,0,0)
dim ii as short
for ii = 0 to 5
    line (110+ii, 10+ii)-(190+ii, 90+ii), rgb(0,0,255),b
next ii
paint (121, 17), rgb(255,0,0), rgb(0,0,255)
sleep
```

producendo queste figure



con i bordi evidenziati a dovere, ottenuti disegnando le linee cinque volte, una vicina all'altra, a distanza di un pixel.

draw string

Colloca una scritta in una posizione voluta. La sintassi è:

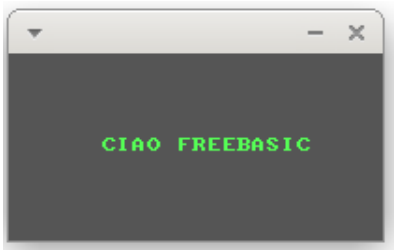
`draw string (x, y), <testo>, <colore>`

dove x e y sono le coordinate del punto che si troverà in alto a sinistra della scritta.

Questo programmino

```
screenres 200, 100
color ,8
cls
draw string (50, 45), "CIAO FREEBASIC", 10
sleep
```

produce questo



10.3 Tracciare

I disegni fatti nel precedente paragrafo sono da riga, squadra e compasso.

FreeBASIC ci offre la possibilità di tracciare altri percorsi, per esempio le curve che rappresentano funzioni matematiche.

Per fare cose di questo tipo abbiamo innanzi tutto bisogno di cambiare il canvas.

La finestra grafica che abbiamo creato nel paragrafo 10.1 è basata su un sistema di coordinate, tutte di segno positivo, con l'origine nel punto in alto a sinistra della finestra.
Ciò non consente di utilizzare anche coordinate di segno negativo come può essere necessario fare in certe situazioni come, per esempio, il tracciamento delle curve di funzioni matematiche.

Esiste una funzione che ci consente di trasformare la nostra finestra grafica in un piano cartesiano, con l'origine degli assi all'interno della finestra in modo che la finestra stessa sia divisa in quattro settori: quello in alto a destra con coordinate entrambe positive, quello in alto a sinistra con coordinata orizzontale negativa e coordinata verticale positiva, quello in basso a sinistra con entrambe le coordinate negative e quello in basso a destra con coordinata orizzontale positiva e coordinata verticale negativa. La sintassi di questa funzione è:
`window(x1, y1)-(x2, y2)`
dove la prima coppia di numeri indica il valore delle coordinate del punto in basso a sinistra del piano e la seconda coppia di numeri indica le coordinate del punto in alto a destra.

Se ha da essere un piano cartesiano dobbiamo disegnarne gli assi, con questa sintassi

```
dim x as single
for x = <a> to <b> step 0.001
    pset(x,0)
    pset(0,x)
next
```

dove <a> è il maggiore tra i valori assoluti di x1 e y1 e <b> è il maggiore tra x2 e y2.

Il tracciamento del grafico di una funzione avviene con la sintassi:

```
dim x as single
dim y as single
for x = x1 to x2 step 0.001
    y = <f(x)>
    pset (x, y), <colore>
next
```

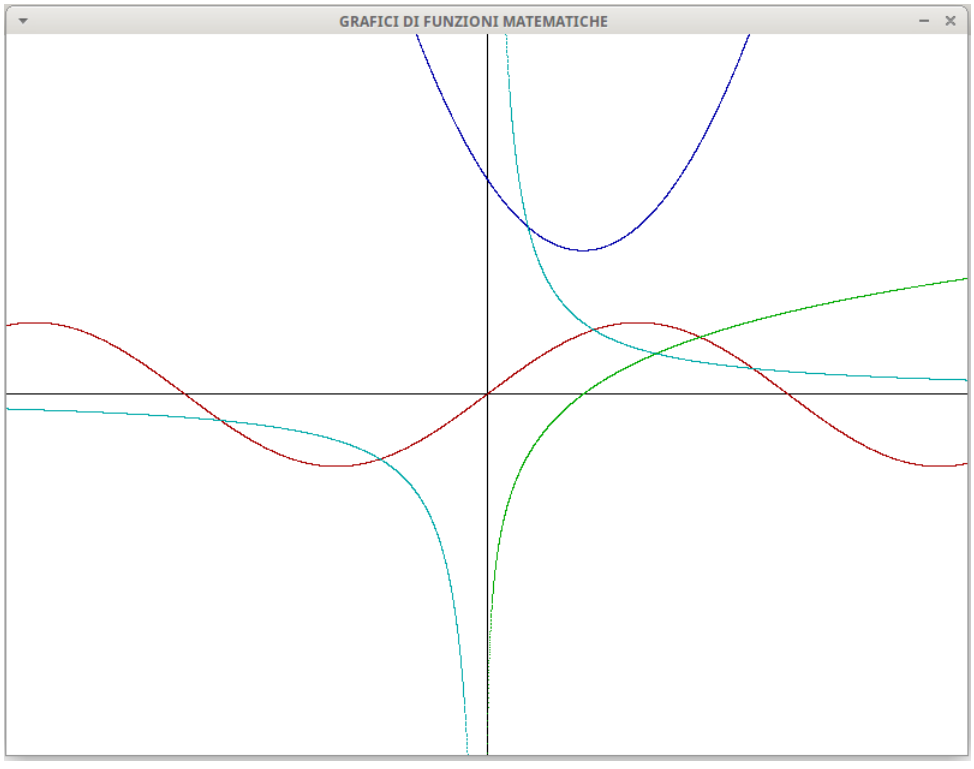
dove x1 e x2 designano l'intervallo dei valori della x per cui tracciare iol grafico, <f(x)> è l'espressione della funzione matematica e <colore> è la solita cosa di cui abbiamo parlato più volte.

Per riassumere il tutto propongo questo programmino di esempio

```
screen 19
windowtitle ("GRAFICI DI FUNZIONI MATEMATICHE")
color , 15
cls
window(-5,-5)-(5,5)
dim x as single
dim y as single
for x = -5 to 5 step 0.001
    pset(x,0), 0
```

```
    pset(0,x), 0
next
for x = -5 to 5 step 0.001
    y = x^2 - 2*x + 3
    pset (x, y), 1
next
for x = -5 to 5 step 0.001
    y = sin(x)
    pset (x, y), 4
next
for x = -5 to 5 step 0.001
y = 1/x
    pset (x, y), 3
    next
for x = -5 to 5 step 0.001
    y = log(x)
    pset (x, y), 2
next
sleep
```

che produce questa finestra



Ho scelto un piano cartesiano con origine al centro della finestra e con una scala di grandezze che va da -5 a 5 unità su entrambi gli assi, in modo che l'andamento delle curve nelle vicinanze dell'origine sia ben visibile.
Data la necessità di tracciare in corrispondenza ai valori in numeri reali che assumono le funzioni matematiche, i valori delle x e delle y sono stati dichiarati di tipo single (decimale a precisione singola) e data la dilatazione della scala introdotta ridimensionando una finestra di centinaia di pixel per asse ad una visibilità di poche unità per asse ho scelto uno step per i cicli di tracciamento di 0,001 unità: più questa grandezza è piccola, più nitida e continua è la traccia.

10.4 Rispondere al mouse

FreeBASIC ha due funzioni per rapportarsi al mouse: una per collocare il puntatore del mouse in un determinato punto dello schermo, l'altra per identificare posizione e stato del mouse.

La prima, setmouse, colloca il puntatore sullo schermo con la sintassi

`setmouse (x, y)`

dove x e y sono le coordinate del punto in cui collocare il puntatore.

La seconda, getmouse, identifica la posizione del mouse con la sintassi

`getmouse (x, y)`

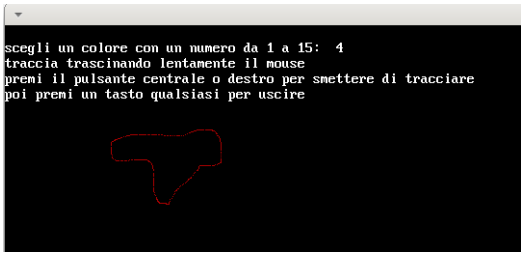
dove x e y sono le variabili in cui la funzione colloca i valori delle coordinate del punto dove si trova il puntatore.

Nella posizione successiva al valore della y la funzione identifica lo stato della ruota del mouse e in quella ancora successiva identifica lo stato del pulsante, restituendo, nel mouse a tre pulsanti il valore 0 se è premuto il tasto sinistro, il valore 1 se è premuto il tasto destro e il valore 2 se è premuto il tasto centrale e, nel mouse a due pulsanti, il valore 1 se è premuto il tasto sinistro e il valore 2 se è premuto il tasto destro.

Il seguente codice utilizza queste funzioni per fare un piccolo disegno a mano libera

```
screen 19
dim colore as short
print
input "scegli un colore con un numero da 1 a 15:  ", colore
print "traccia trascinando lentamente il mouse"
print "premi il pulsante centrale o destro per smettere di tracciare"
print "poi premi un tasto qualsiasi per uscire"
dim leggi_mouse as short
dim x as integer
dim y as integer
dim pulsante as integer
setmouse(200, 150)
do
    leggi_mouse = getmouse(x, y,, pulsante)
    pset(x, y), colore
loop until pulsante = 2
sleep
```

di cui segue un modestissimo esemplare



10.5 Rispondere alla tastiera

Nel paragrafo 7.2 abbiamo già fatto conoscenza con la funzione `inkey`, che acquisisce come stringa il carattere del tasto premuto e può essere utile per fare avvenire qualche cosa alla pressione di un certo tasto che corrisponda ad una certa lettera.

Ma per l'interfacciamento con la tastiera esiste un'altra funzione che può tornare molto utile anche per il riconoscimento di tasti che non rappresentano caratteri. Si tratta della funzione `multikey`, la cui sintassi è:

`multikey(<scancode>)`

dove <scancode> è il codice che la tastiera invia al processore per segnalare quale tasto è stato premuto.

L'elenco completo dei codici si trova nel manuale di FreeBASIC sotto la voce Keyboard Scan Codes.
In questo elenco i tasti sono individuati da parole che iniziano con le lettere sc, seguite dalla lineetta di sottolineatura e dal nome del tasto; di fianco è indicato il codice esadecimale corrispondente.
Con `sc_`<lettera> si identificano i tasti delle lettere.
Tanto per citare i più utili non corrispondenti a lettere dell'alfabeto:
. con `sc_escape` si identifica il tasto ESC e il suo codice esadecimale è `&h01`.
. con `sc_left`, `sc_right`, `sc_up`, `sc_down` si identificano i tasti freccia a sinistra, freccia a destra, freccia su e freccia giù, e i rispettivi codici sono `&h4B`, `&h4D`, `&h48` e `&h50`.
. con `sc_space` si identifica la barra spaziatrice e il suo codice è `&h39`.

Per la risposta alla tastiera dobbiamo anche approfondire la conoscenza di una funzione che ho introdotto nel Capitolo 2 e di cui ho parlato anche nel Paragrafo 10.1: la funzione `sleep`. L'uso di questa funzione senza indicare alcun argomento, come abbiamo fatto negli esempi svolti finora, ferma lo schermo fino a quando viene premuto un qualsiasi tasto e, alla pressione di un qualsiasi tasto, tutto scompare.

La funzione, però, può accettare due utilissimi argomenti.
Il primo argomento è un numero che indica, in millisecondi, la durata dell'attesa: millisecondi significa che 1000 è un secondo.
Il secondo argomento può assumere il valore 0 o il valore 1. Il valore 0, quello di default, ferma l'attesa alla pressione di un tasto qualsiasi; il valore 1 disabilita questa funzione, cioè instaura indifferenza alla pressione dei tasti. L'utilizzo di questo argomento torna necessario quando si voglia utilizzare la tastiera durante un'attesa.

Per esemplificare quanto contenuto in questo paragrafo propongo un semplice programma che ci dà modo di comandare un semaforo, facendogli assumere i colori rosso, giallo o verde alla pressione dei tasti `r`, `g` o `v` e, quando siamo stufi, di uscire dal gioco alla pressione di un certo tasto.

Ho ritenuto utile presentare il programma in due versioni, in una utilizzando la funzione `inkey`, nell'altra utilizzando la funzione `multikey`.

Con la funzione `inkey`:

```
screenres 100,200
sub istruzioni
    print
    print "r = rosso"
    print "g = giallo"
    print "v = verde"
    print "e = esci"
end sub
istruzioni
do
    if inkey = "r" then circle(50,90), 15, 4,,,,f
    if inkey = "g" then circle(50,130), 15, 14,,,,f
    if inkey = "v" then circle(50,170), 15, 2,,,,f
    sleep 2000, 1
    cls
    istruzioni
loop until inkey = "e"
```

Con la funzione `multikey`:

```
screenres 100,200
sub istruzioni
    print
    print "r = rosso"
    print "g = giallo"
    print "v = verde"
    print "ESC= esci"
end sub
istruzioni
do
    if multikey(&h13) then circle(50,90), 15, 4,,,,f
    if multikey(&h22) then circle(50,130), 15, 14,,,,f
    if multikey(&h2F) then circle(50,170), 15, 2,,,,f
    sleep 2000, 1
    cls
    istruzioni
loop until multikey(&h01)
```

In entrambi i casi, tenendo premuto uno dei tasti con le lettere r, g o v accendiamo le zone del semaforo dedicate alle luci dei colori rosso, giallo o verde; la luce rimane accesa un paio di secondi e poi scompare; per uscire dal programma dobbiamo premere il tasto e nel primo caso e il tasto `ESC` nel secondo caso.

10.6 Creare movimento

Il movimento si crea instaurando un ciclo nel quale si disegna in un certo punto dello schermo la figura destinata a muoversi e la si ridisegna dopo breve pausa in una posizione prossima alla precedente cancellando il precedente disegno.

Gli strumenti per fare questo li abbiamo già tutti, serve solo creatività per utilizzarli.

Con il seguente programmino disegniamo un cerchio sullo schermo e lo spostiamo utilizzando le frecce della tastiera

```
screen 19
dim as integer x, y
sub istruzioni
    print "sposta il cerchio con i tasti freccia"
    print "premi ESC per finire"
end sub
x = 300
y = 200
do
    'verifica se vengono premuti i tasti freccia
    if multikey(&h4B) then x = x - 1
    if multikey(&h4D) then x = x + 1
    if multikey(&h48) then y = y - 1
    if multikey(&h50) then y = y + 1
    'pulisce lo schermo e disegna il cerchio alle nuove coordinate
    cls
    circle(x, y), 30, 4,,,,f
    istruzioni
    sleep 10, 1
loop until multikey(&h01)
```

Con quest'altro programmino disegniamo un cerchio che si muove sullo schermo e rimbalza non appena tocca un bordo.

```
screenres 200, 200
dim as single x, y, a, b
x = 100
y = 100
a = 0.5
b = 0.7
do
    print "premi s per finire"
    circle(x,y),10,4,,,,f
    if x < 0 then a = - a
    if x > 200 then a = - a
    if y < 0 then b = - b
    if y > 200 then b = - b
    x = x + a
    y = y + b
    sleep 10
    cls
loop until inkey = "s"
```

Il ridisegno del cerchio per creare il suo movimento avviene aumentando, ad ogni ciclo, le coordinate del centro delle grandezze a e b; al raggiungimento di uno dei bordi la grandezza che aumenta la coordinata di quel bordo inverte di segno, in modo che il ridisegno avvenga in direzione contraria.

11 Estensioni

Per tutto ciò che abbiamo fatto finora, e non è poco, ci è bastata la dotazione base del linguaggio FreeBASIC.

Se, presa dimestichezza con il linguaggio, vogliamo fare cose più sofisticate abbiamo a disposizione tantissime librerie esterne.

Ne troviamo un elenco nel manuale FreeBASIC sotto la voce External Libraries Index.

Cliccando sui nomi veniamo sommariamente informati su cosa fa la libreria e ci viene indicato l'indirizzo web ove trovare informazioni complete.