IL LIBRO DI KBASIC

MANUALE PER LO SVILUPPO DI APPLICAZIONI KBASIC

Prima Edizione

Questa edizione si applica alla release 1.0 di KBasic e a tutte quelle pubblicate in seguito, a meno che diversamente indicato. Assicurati di usare l'edizione adatta alla versione del prodotto. Il termine „KBasic“ come usato in questa pubblicazione, si riferisce all'insieme dei prodotti KBasic (Gennaio 2006).

A proposito di questo libro

Benvenuto a 'Il Libro di KBasic', la tua guida per lo sviluppo di applicazioni KBasic. In questo libro troverai tutte le informazioni necessarie per creare con successo i tuoi programmi KBasic. Se completi il libro, sarai in grado di scrivere semplici o complessi programmi sia GUI (Graphical User Interface, *ndt*) che in semplice BASIC standard. Questo manuale ti assisterà per quasi tutti i problemi di programmazione della vita di tutti i giorni.

Se inizi con KBasic forse con questo libro ti rilasserai in un posto piacevole e leggerai il primo capitolo imparando il linguaggio di programmazione. Più tardi allora, nel momento libero, vorrai probabilmente provare gli esempi.
Comunque, lo scopo principale di questo libro è di rimanere accanto alla tastiera, per servirti come veloce e pratico riferimento durante la programmazione. I programmatori KBasic con meno esperienza ricevono un'ampia introduzione al linguaggio. L'orientamento agli oggetti e altre caratteristiche di KBasic, sono spiegati da numerosi esempi di uso pratico per gli elementi più importanti del linguaggio.

Troverai dei riferimenti completi al linguaggio di programmazione nell'ambiente di sviluppo integrato KBasic (KBasic Software Atelier), e perciò questo libro giunge senza dettagliati riferimenti. Nell'IDE di KBasic troverai una lista completa e dettagliata di tutti gli oggetti, i loro eventi e procedure, metodi e caratteristiche, così come i riferimenti completi di tutti gli operatori e comandi. A parte una definizione precisa della sintassi, in ogni caso troverai una spiegazione particolareggiata dei parametri disponibili.

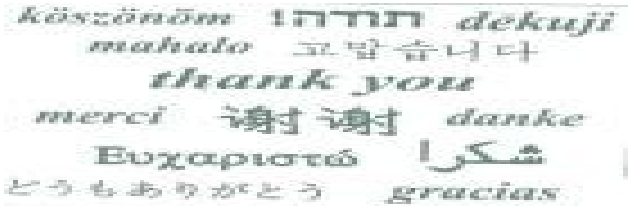
Perché dovresti leggere questo libro

Questo libro è fatto per persone che hanno qualche conoscenza di base del BASIC, in più principianti assoluti faranno facilmente dei progressi. Se sei uno sviluppatore Visual Basic con esperienza, ti sentirai veramente a casa.

Questo libro è rivolto a persone che

- Vogliono imparare un semplice e potente linguaggio di programmazione per Linux e Windows o Mac OS X
- Sono sviluppatori con esperienza in C/C++, Java o Visual Basic, che vogliono passare a KBasic o allargare le loro capacità
- Hanno sentito che c'è un nuovo attraente linguaggio di programmazione per Linux, Mac OS X e Windows, che desiderano avere

A parte ciò. Questo libro
- E' quello che devi avere se stai sviluppando seriamente con KBasic
- Ti dà una solida preparazione dei fondamenti di KBasic
- Ti guida passo per passo attraverso le caratteristiche dei linguaggi



Note di ringraziamento

Sono molto grato a tutte le persone che mi hanno aiutato a completare questo libro. Ringraziamenti speciali vanno a Nadja, la mia ragazza, e a tutti quelli che hanno comprato KBasic Professional o mi hanno sostenuto in altro modo, e un grande ringraziamento va ai miei genitori.

Circa l'autore

Bernd Noetscher è uno sviluppatore software e lo sviluppatore principale del linguaggio di programmazione KBasic. Nel suo tempo libero fa ballo, legge molti libri e suona il pianoforte. S'interessa anche di cinema e teatro.

Il suo sito personale è www.berndnoetscher.de.

Sostenitori

Grazie! Nadja Hepp per la correzione delle bozze e per i beta tester e altri sostenitori. Grazie a tutti quelli che in Internet hanno inviato correzioni e suggerimenti, siete stati enormemente utili nel migliorare la qualità di questo libro, non avrei potuto farlo senza di voi: grazie a tutti!

Traduzione italiana

A cura di Fabrizio Pani: fabph@hotmail.com

Dacci il tuo feedback

Ti chiedo di aiutarmi a migliorare questo libro. Come lettore tu sei il più importante commentatore e critico del mio libro. Rispettiamo la tua opinione e vorremo sapere cosa ti piace o cosa potremo fare di meglio. Se trovi un errore negli esempi o nel testo, allora scrivi un e-mail a info@KBasic.com. Grazie!

Per favore nota che non possiamo aiutarti in problemi tecnici correlati al soggetto di questo libro, e che a causa di un elevato volume di e-mail che riceviamo, potremo non essere in grado di rispondere ad ogni messaggio.

Chi non vuole più migliorare, non è più bravo!

Migliorare senza fermarsi significa per noi, soddisfare i desideri dei nostri clienti, perché il nostro scopo principale è la soddisfazione del cliente.

Per raggiungere uno scopo così alto, abbiamo bisogno del tuo aiuto. Se abbiamo commesso qualche errore per favore diccelo. Solo se li conosciamo, possiamo correggere questi errori.

Per favore scrivi a info@KBasic.com – vorremo anche ricevere commenti positivi. Grazie mille.

Nuovo set di manuali su KBasic

KBasic è stato sviluppato da KBasic Software e ha suscitato molto interesse in Internet, nel business e tra gli sviluppatori. Ci siamo obbligati a documentare questa nuova eccitante tecnologia con un insieme di nuovi manuali. Questa serie tratterà i riferimenti al linguaggio, volumi introduttivi e riferimenti alle API, e anche argomenti avanzati di programmazione KBasic, come database e reti.

Convenzioni usate in questo libro / modi di scrittura

Il testo normale appare scritto in Arial. Ecco un esempio:
Questo è testo normale

Sintassi e codice sorgente appaiono scritti in Courier New e sfondo grigio. Ecco l'esempio:

```
Dim i As Integer
```

Riferimenti importanti e parole chiavi sono in corsivo:

```
Argomenti
```

Fonti per ulteriori informazioni

Il sito del produttore di KBasic: www.kbasic.com

Informazioni che coprono la programmazione in BASIC: archivi Internet su Visual Basic 6 per esempio; cerca in Internet usando www.google.com

Indice

Introduzione

Avrai probabilmente già sentito molto di KBasic. Non leggeresti questo libro, se non hai considerato KBasic una nuova e importante tecnologia. Adesso avrai presumibilmente già scaricato la Personal Edition di KBasic e giocato un po' con i programmi d'esempio. Ora questo ti incita a scrivere i tuoi programmi KBasic...

Hai mai voluto sapere come funziona un programma per computer? Hai mai voluto sederti presso la tua tastiera e creare musica digitale nello schermo del tuo computer? Se è così, potrebbe esserci un programmatore da qualche parte dentro di te, che attende di uscire fuori.

Potresti aver trovato la programmazione non solo minacciosa ma anche orribile. Ti diventano i capelli grigi ogni volta che tenti di scrivere un semplice file script (un insieme di comandi interpretati ed eseguiti riga per riga, *ndt*) ? Se ti senti così, 'Il Libro di KBasic' è qui per provarti che la programmazione è divertente, remunerativa e, cosa più importante, non difficile. Prima di iniziare a sviluppare programmi, potrebbe aiutare una comprensione di base di tutto ciò che è a riguardo. Probabilmente hai qualche idea su cosa è un programma e come funziona, o non avresti preso questo libro per iniziare. Qualsiasi siano le tue idee sulla programmazione, questa presentazione ti darà la conoscenza. Dopo aver letto questa introduzione, puoi trovare che la tua conoscenza sulla programmazione è abbastanza buona; o potresti realizzare che ne sai di programmazione come di costruire un aeroplano. In entrambi i casi, ne vale la pena...

Il grande mistero

Il mondo della programmazione ha un mistero tenuto nascosto per bene. Non sentirai i programmatori parlarne. In qualsiasi momento stessi usando un computer, presumibilmente troverai questo segreto difficile da credere. Cionondimeno è vero come il cielo non è rosso. Stai per scoprire un fatto interessante: i computer sono stupidi. Francamente, un computer non può fare assolutamente niente da solo. Senza programmatori, i computer sono inutili come automobili senza benzina. I computer possono fare solo quello che gli viene detto di fare. Se ci pensi per un minuto, troverai che questo significa che i computer possono solo eseguire compiti che gli esseri umani sanno già come risolvere.
Allora perché i computer sono straordinari? La cosa buona dei computer non è che sono intelligenti, ma che possono eseguire calcoli interminabili in secondi.
Gli sviluppatori, naturalmente, sono le persone che dicono ai computer cosa fare. Con questo non si sta dicendo che quando usi un computer lo stai programmando. Per esempio, quando ti siedi davanti al tuo elaboratore di testi e fai una lettera alla tua persona amata, non stai dando comandi al computer. Stai solo usando i comandi contenuti nel programma. E' il programma, il quale è stato scritto da un programmatore, che effettivamente dice al computer cosa fare.
Probabilmente usi il computer per molte attività oltre all'elaborazione di testi, come organizzare dati in un foglio di calcolo, o forse per giocare ai videogiochi. In tutti questi casi, esegui un programma, che a sua volta fornisce istruzioni al computer. Come utente di un computer, sei all'apice della catena. Il punto è che se vuoi impartire direttamente dei comandi al tuo computer, devi imparare a scrivere programmi.

Perché dovresti imparare a programmare?

Ci sono tante ragioni per imparare a programmare quanti sono gli individui al mondo. Solo tu sai cos'è della programmazione che ti fa desiderare impararla.
Ma alcuni motivi sono che vuoi:

- Imparare la programmazione per lavoro o scuola
- Essere in grado di scrivere i programmi di cui hai bisogno veramente
- Imparare di più su come funziona un computer
- Far colpo sui tuoi amici o la ragazza o persino il ragazzo
- Stai cercando un hobby divertente o remunerativo

Sono tutte buone ragioni. Potresti averne una migliore, ma qualunque essa sia, dopo aver iniziato con la programmazione, troverai che è sia affascinante che interessante e che crea dipendenza.

Cosa è un programma per computer?

Un programma è semplicemente una lista di istruzioni che dice al computer cosa fare. Il computer segue queste istruzioni o comandi, una ad una, fino a che raggiunge la fine del programma.
Ogni linea in un programma è solitamente un singolo comando che il computer deve eseguire. Ciascun comando esegue solo un piccolo compito, come scrivere un nome sullo schermo o addizionare due numeri. Quando metti centinaia,
migliaia, o anche centinaia di migliaia di questi comandi assieme, il tuo

calcolatore può fare grandi cose: risolvere molto velocemente dei problemi matematici, stampare un documento, disegnare un'immagine, o giocare a 'SimCity'. Come vedrai nel prossimo paragrafo, i programmi per computer possono essere scritti in uno dei tanti linguaggi di programmazione.

Linguaggi di programmazione

I computer non capiscono il tedesco o l'inglese, o qualsiasi altro linguaggio umano. Non sono in grado neanche di comprendere il BASIC, il linguaggio di programmazione su cui è basato KBasic. I computer capiscono solo una cosa, il linguaggio macchina, che è composto interamente dei numeri 1 o 0. Ma linguaggi di programmazione come il BASIC consentono alle persone di scrivere programmi in un linguaggio simile all'inglese, l'interprete BASIC lo modifica in linguaggio macchina affinché il computer possa comprenderlo.
I programmi KBasic sono un dialetto del linguaggio di programmazione BASIC, che fu sviluppato non per aiutare i computer, ma per aiutare le persone a dare un significato ai dati numerici (numeri binari, *ndt*) privi di senso con cui "pensano" le macchine. Il KBasic rimpiazza molti dei numeri usati dal linguaggio macchina con parole e simboli che noi umani possiamo più facilmente capire e ricordare. Inoltre, KBasic mette in grado un programmatore di assemblare visivamente la finestra di un programma dagli elementi di una toolbox (casella degli strumenti, *ndt*) . Quindi è più facile per te comunicare con il tuo computer come sviluppatore software, invece di pensare come una macchina puoi pensare come un essere umano.

Storia del nome

Qualche volta, vedi il nome del linguaggio BASIC scritto in lettere minuscole come in: Basic. Anche KBasic usa questo spelling. Comunque, BASIC effettivamente inizia come un acronimo, è per questo che puoi anche vedere il nome scritto con tutte le lettere maiuscole. L'acronimo BASIC sta per Beginner's All-purpose Symbolic Instruction Code (codice di istruzioni simboliche di uso generale per principianti, *ndt*).

Un programma BASIC usa simboli e parole (e un po' di numeri) che le persone possono comprendere. Com'è possibile che un computer possa capire ed eseguire il BASIC? La soluzione è che, quando tu carichi KBasic, stai anche caricando un compilatore, che è un programma speciale che prende le parole e i simboli di un programma KBasic e li traduce in linguaggio macchina che il computer può comprendere. Il tuo computer non avrebbe alcuna idea di cosa fare con il tuo programma senza „l'interpretazione" del compilatore.
Dopo tutto, esistono molti tipi di linguaggi di programmazione, includendo Java, C++ e BASIC... Ma tutti i linguaggi di programmazione hanno una cosa in comune: Possono essere letti dagli umani e, perciò, devono essere convertiti in linguaggio macchina prima che il computer possa comprenderli.

Tutti i tipi di BASIC

Ci sono molte versioni del linguaggio BASIC, di cui KBasic è solo una. Nuove versioni di DOS vennero con QBasic. Puoi anche scendere al tuo locale negozio di software e comprare Visual Basic. Tutti questi pacchetti software ti consentono di creare programmi in BASIC, ma tutti implementano il linguaggio BASIC un po' diversamente. Comunque, delle versioni BASIC menzionate qui, KBasic ti mette in grado di scrivere applicazioni per Linux o Mac OS X e non solo applicazioni per Windows.

Compilatori e interpreti

Alcuni linguaggi di programmazione, come alcuni tipi di BASIC, convertono un programma in linguaggio macchina una riga alla volta quando il programma è in esecuzione. Altri linguaggi, come Java e KBasic, usano un compilatore per convertire l'intero programma tutto in una volta prima di ogni esecuzione. In ogni caso, tutti i linguaggi devono essere convertiti in codice macchina affinché il computer capisca Il programma.
Un compilatore trasforma il tuo programma in un file eseguibile che può girare direttamente, senza un compilatore o interprete. Un programma eseguibile è effettivamente un programma in linguaggio macchina che è pronto per essere letto e capito dal tuo computer. Con poche eccezioni, la maggior parte dei linguaggi di programmazione hanno un compilatore. Ma oggi è anche difficile distinguere tra compilatori e interpreti, poiché esistono forme miste di entrambi come Java o KBasic.

Modo di programmare

Ora che sai qualcosa sui programmi, come ti occupi di scriverne uno? Creare un programma è facile, sebbene possa essere un processo lungo. Scrivere un programma KBasic richiede uno sviluppo per passi simile a quello che usi scrivendo un documento o un report. La lista seguente delinea questi possibili gradini:

1. Comincia con un concetto di programma e disegna una bozza su carta di come potrebbe apparire su schermo.
2. Crea il programma usando la casella degli strumenti di KBasic e l'editor del codice sorgente.
3. Salva il programma su disco.
4. Esegui il programma e osserva come funziona.
5. Correggi gli errori di programmazione.
6. Ritorna al punto 2.

La maggior parte dei passi nel processo di programmazione sono ripetuti più volte come si scoprono e correggono errori, come puoi vedere. Persino programmatori con esperienza non possono scrivere programmi esenti da errori o il programma è molto corto. I programmatori spesso impiegano più tempo per rifinire i loro programmi che inizialmente scrivendoli. E' importante fare dei ritocchi, poiché noi umani non siamo logici come ci piace pensare. Inoltre le nostre menti sono incapaci di ricordarsi ogni dettaglio richiesto per far girare perfettamente un programma. La maggior parte di noi sono fortunati se possono ricordare i nomi dei nostri presidenti. Solo quando un programma va in crash o fa qualcos'altro di inaspettato possiamo sperare di trovare quelli errori che si nascondono nei programmi. Esperti di computer dicono che non c'è una cosa come un programma senza bug (errori di programmazione, lett. Insetti, *ndt*) . Dopo che inizi a scrivere programmi interi, vedrai quanto è vera questa affermazione.
E' anche possibile scrivere programmi in altri modi rispetto a quelli descritti sopra... Troverai alcune idee in Internet, che è una grande fonte d'informazione per ogni genere di sviluppatore.

Correzione dei bug

I bug sono errori di programmazione che impediscono al tuo programma di funzionare correttamente. Prima che un programmatore possa distribuire il suo software al pubblico, deve essere certo di aver „schiacciato“ bug il più possibile, che significa correggere il più possibile gli errori.

E' difficile la programmazione?

Dopo aver letto tutto quello che implica scrivere ed eseguire un programma, potresti essere un po' a disagio. Dopo tutto hai questo libro, poiché ti ha promesso d'insegnarti a programmare in KBasic. Nessuno ti ha avvisato di misteriosi argomenti come linguaggio macchina, interpreti, compilatori, e bug di programmi.
La programmazione è facile o no?
Probabilmente, si e No.
E' facile imparare a scrivere piccoli programmi con KBasic. Il linguaggio KBasic
È logico, simile all'inglese, e facile da capire. Con solo un po' di pratica, puoi scrivere molti programmi utili e divertenti. Tutto quello di cui hai bisogno è il tempo:

- Di leggere questo libro e
- L'ambizione di scrivere pochi programmi per conto tuo.

In effetti, quello che imparerai in questo libro è abbastanza per chi non aspira a diventare un programmatore professionale.

Comunque, se vuoi fare della programmazione una carriera, hai molto da imparare che non è incluso in questo libro introduttivo. Per esempio, considera un elaboratore di testi come OpenOffice, che richiese a dozzine di programmatori molti anni di scrittura. Per scrivere un software così complesso, devi avere una buona conoscenza di come funziona il computer. In più, devi aver trascorso molti anni ad imparare le abilità della programmazione professionale.
Ancora, c'è molto che puoi fare con KBasic, sia che tu sia interessato a scrivere utilità, piccole applicazioni, o persino videogiochi. e, passo per passo, scoprirai che la programmazione in KBasic non è così difficile come potresti aver pensato.

Quanto devo conoscere di KBasic per usarlo?

E' possibile sviluppare programmi KBasic senza dover scrivere una riga di codice sorgente. Ci sono molti posti per ottenere programmi KBasic, come libri, Internet, e persino amici e colleghi. Puoi modificare questi programmi in KBasic e avere un applicazione pronta all'uso. Ad ogni modo, qualche volta questi programmi hanno bisogno da semplici a ampie modifiche ed è qui dove accadono sfide reali alla vostra abilità di programmazione.

Qualcosa che hai bisogno di sapere

Prima di iniziare seriamente a scrivere i tuoi programmi con KBasic, è importante conoscere cosa sono i programmi e come funzionano. Questa conoscenza di base ti aiuterà a capire perché KBasic fa alcune delle cose che fa, così come facilitarti nel localizzare errori nei tuoi software. Dovresti ricordarti i seguenti punti:

- Un computer può solo fare quello che un essere umano lo istruisce a fare
- Ci sono molti linguaggi di programmazione. Il linguaggio che impari in questo libro è una forma di BASIC
- Un programma è una lista di comandi che il computer esegue dall'inizio alla fine
- Un programma BASIC deve essere convertito in linguaggio macchina prima che il computer possa comprenderlo. Il compilatore KBasic fa questa conversione
- Scrivere un programma è molto di più che scrivere un documento di testo. Devi scrivere alcune bozze prima che il programma sia completo e pronto
- Programmare con KBasic è facile o difficile come vuoi che esso sia

Cosa ti rende più facile capire KBasic:
Dovresti avere una comprensione di base della programmazione object-oriented (orientata agli oggetti, *ndt*) come quella del C++. Molti dei principi e concetti di KBasic sono basati su quelli che si trovano in C++. Ci sono anche alcune notevoli differenze tra KBasic e C++ che dovresti conoscere, inoltre.
Se stai usando database, una comprensione di base dello structured query language (SQL) (linguaggio d'interrogazione strutturato, *ndt*) e dei modelli client-server è necessaria per costruire programmi KBasic più complessi „database-aware“.

Introduzione al linguaggio di programmazione KBasic

Benvenuto al linguaggio di programmazione KBasic. Questo capitolo si riferisce a

- Cosa è esattamente KBasic
- Perché dovresti imparare KBasic, le caratteristiche e i vantaggi di fronte a diversi linguaggi di programmazione
- Primi passi dentro KBasic, che conoscenza essenziale è richiesta e qualche conoscenza di base
- Come scrivere il tuo primo programma KBasic

Cosa è KBasic?

KBasic è un potente linguaggio di programmazione, che è semplicemente intuitivo e molto veloce da imparare. KBasic rappresenta inoltre (sviluppato per Linux, Mac OS X e Windows) un ulteriore ponte tra Linux, Mac OS X e Windows. KBasic è un nuovo linguaggio di programmazione, un dialetto BASIC supplementare, correlato a Visual Basic 6 e Java. Più esattamente si dice che KBasic è un linguaggio di programmazione object-oriented e guidato dagli eventi, sviluppato da KBasic Software (www.kbasic.com), è particolarmente progettato per le esigenze di sviluppatori GUI (Graphical User Interface, interfaccia grafica utente, *ndt*) e ha perciò una sua forza nello sviluppo di interfacce grafiche. Per gli sviluppatori C/C++ il BASIC ha generalmente una cattiva reputazione come di un linguaggio di programmazione per bambini, che non è vero appare chiaramente con questo libro. Con KBasic puoi scrivere quasi tutte le applicazioni che forse sarebbero state scritte in C/C++, se non ci fosse KBasic.

Concisamente in una frase

KBasic è a facile da utilizzare, orientato agli oggetti, interpretato, stabile, indipendente dalla piattaforma, veloce e moderno linguaggio di programmazione.

Avvertenza

Per essere compatibili con le versioni future di KBasic, dovresti evitare certi vecchi elementi del linguaggio KBasic, che sono ancora supportati per ragioni storiche. Usa invece gli elementi moderni del linguaggio. Puoi trovare maggiori dettagli nelle rispettive note di aiuto. Per favore considera anche i riferimenti per la migrazione da Visual Basic 6 alla fine di questo libro. In più, solo le nuove e moderne caratteristiche del linguaggio sono descritte in questo libro, dato che molto elementi del linguaggio sono supportati solo per ragioni storiche da KBasic; es. 'GoSub', 'DefInt' e 'On Error GoTo' sono elementi obsoleti del linguaggio e perciò non sono descritti in questo libro.

Esempi

Gli esempi in questo libro sono in Inglese (commenti e sintassi sono tradotti in italiano, *ndt*)

Passato, presente e futuro di KBasic

KBasic è stato originariamente sviluppato come software open source. Il progetto fu avviato da Bernd Noetscher nell'estate del 2000. Dopo molte delusioni, a causa della mancanza di supporto da parte di volontari, decise di sviluppare il linguaggio di programmazione come prodotto commerciale per Linux, Mac OS X e Windows per garantire che siano completamente soddisfatti gli alti requisiti del prodotto. Al momento, la versione 2.0 di KBasic è in fase di sviluppo, è più user-friendly (usabile, *ndt*), più efficiente, e ha più librerie e nuove caratteristiche orientate agli oggetti.

Cosa c'è ancora bisogno di sapere su KBasic?

KBasic è object-oriented

Questo significa che tu come programmatore ti concentri sui dati e sui metodi nella tua applicazione – come manipolare i dati, invece di pensare solo alle procedure. Se ti è familiare la programmazione procedurale in BASIC, probabilmente avrai la convinzione che devi cambiare il modo di scrivere i tuoi programmi, se usi KBasic. Non sei però tenuto ad utilizzarlo in quel modo, puoi programmare solo con procedure. Se ti familiarizzi con esso, di quanto è potente questo nuovo paradigma object-oriented, ti ci abituerai, poiché vedrai come è facile sviluppare complesse applicazioni riutilizzabili chiaramente e in maniera modulare. KBasic è orientato agli oggetti in relazione al binding (codice per usare una libreria scritta in un linguaggio diverso da quello che si usa, *ndt*) delle librerie C++ (quelle Qt, *ndt*), o alle classi KBasic built-in (lett. Incorporati, precisamente tipi, parole chiavi predefiniti, ndt), con vera ereditarietà. Al contrario di altri linguaggi di programmazione KBasic è progettato dal principio come un linguaggio di programmazione object-oriented. Quindi la maggior parte delle cose in KBasic sono oggetti; i tipi di dati semplici, come quelli numerici, stringhe e valori booleani, sono le sole eccezioni. Sebbene KBasic sia progettato in modo che appaia simile a VB6, noterai che KBasic evita molti problemi di VB6 in modo elegante.

KBasic è facile da imparare

KBasic è un linguaggio facile da usare. Uno scopo della progettazione è la semplicità d'uso. Sviluppo veloce, tracing (tracciamento, *ndt*) più semplice degli errori e facilità d'apprendimento. KBasic è costruito sul modello del BASIC standard, la sintassi è compatibile con Visual Basic 6, a parte i componenti di Visual Basic, ed è perciò molto familiare. Pertanto la migrazione è quasi semplice e veloce. KBasic contiene idee object-oriented di Java e C++, in base alla quale elementi che confondono sono stati omessi. KBasic usa (come Java) reference (riferimenti, *ndt*) invece di puntatori agli oggetti, in più il garbage collector (lett. spazzino, raccoglitore di rifiuti, *ndt*) gestisce completamente in automatico l'allocazione di memoria nel modo più semplice possibile per il programmatore. Non hai puntatori invalidi e memory leak (perdite di memoria, *ndt*), invece puoi velocizzare il tuo tempo nello sviluppo di algoritmi e nel loro uso. KBasic è effettivamente un completo ed elegante linguaggio di programmazione.

Interpretato

KBasic produce pseudo-codice invece di codice macchina. Per eseguire un programma KBasic puoi usare l'ambiente di sviluppo KBasic (IDE) o la voce di menu 'Run/Make Binary' (solo nella versione professionale). KBasic è pertanto un linguaggio di programmazione interpretato. Lo pseudo-code (o *p-code*, *ndt*) KBasic è un formato indipendente dall'architettura, il codice è stato progettato per eseguire in maniera efficiente programmi in diverse piattaforme.

Stabile

KBasic è progettato per avere la possibilità di scrivere software molto affidabile e durevole. Naturalmente, KBasic non rende non necessario il controllo di qualità. E' ancora possibile scrivere software inaffidabile. Comunque KBasic elimina certi tipi di errori di programmazione, cosicché è più facile scrivere software attendibile. KBasic è un linguaggio tipizzato, in grado di fare degli esami molto dettagliati dei tipi potenziali d'inconsistenze nel tuo programma. Perciò KBasic è più fortemente tipizzato del BASIC standard. Una delle caratteristiche più importanti riguarda l'affidabilità è la gestione della memoria. KBasic non supporta i puntatori, il che esclude il pericolo di guastare la memoria e anche sovrascrivere dati. Similmente, il garbage collector di KBasic previene memory leak e altri errori maligni con l'allocazione e la deallocazione dinamica della memoria. In più, l'interprete KBasic esamina diverse cose a run-time (lett. momento dell'esecuzione, ciò che si verifica durante l'esecuzione di un programma, *ndt*) , es. se array e stringhe sono ancora entro i loro limiti. Il trattamento delle eccezioni è un'altra nuova possibilità: KBasic offre questa caratteristica per scrivere programmi più stabili. Un eccezione è un segnale per una condizione anormale del programma, che si verifica come un errore. Quando usi le istruzioni try/catch/finally, puoi mettere tutte le tue routine di elaborazione degli errori in un unico punto, il che rende più semplice gestire e correggere gli errori.

KBasic è veloce

KBasic è un linguaggio di programmazione interpretato. Perciò, non sarà mai veloce come un linguaggio compilato in codice macchina, come il C. Effettivamente, KBasic è 20 volte più lento del C. Ma invece di andartene via, dovresti tenere a mente che questa velocità è abbastanza rapida per quasi tutti i tipi di applicazioni GUI, che spesso attendono che l'utente inserisca qualcosa. Studiando la potenza di KBasic è importante pensare alla collocazione di KBasic tra gli altri linguaggi di programmazione. Ad un'estremità sono i linguaggi ad alto livello interamente interpretati come PHP e shell UNIX. Questi linguaggi sono molto utili per creare un abbozzo, ma sono eseguiti lentamente. All'altra estremità ci sono il linguaggi di programmazione compilati interamente in linguaggio macchina. Questi linguaggi forniscono programmi di estrema velocità, ma non sono molto portabili o fidati. KBasic è in mezzo a questi linguaggi. Le prestazioni di KBasic sono molto meglio dei linguaggi ad alto livello, ma con la facilità e portabilità di questi. Sebbene KBasic sia veloce come un programma in C compilato, è un ambiente indipendente dalla piattaforma, che consente di scrivere programmi stabili per Windows, Mac OS X e Linux.

Perché KBasic ha successo

Tradizionalmente, i linguaggi di programmazione hanno sofferto dell'atteggiamento che tu dovresti abbandonare tutto quello che sai e iniziare da zero con un nuovo insieme di concetti e una nuova sintassi, sostenendo che è meglio a lungo termine perdere tutto il vecchio bagaglio che viene insieme ad esso. Questo può essere vero, a lungo termine, ma a breve termine, molto di quel bagaglio era di valore. Gli elementi di maggior valore possono non essere il codice di base esistente. Se sei un valido programmatore VB6 e devi abbandonare tutto quello che sai di Visual Basic 6 per adottare un nuovo linguaggio, immediatamente diventi molto meno produttivo per molti mesi, fino a che la tua mente si adatta al nuovo paradigma. Laddove se puoi trarre vantaggio dalla tua preesistente conoscenza di VB6 ed espanderla, puoi continuare ad essere produttivo con quello che già sai mentre entri nel mondo della programmazione object-oriented. Poiché ognuno ha il suo modello mentale di programmazione, questo mossa è abbastanza disordinata come lo è senza il costo aggiunto di iniziare con un nuovo linguaggio. Il problema d'imparare un nuovo linguaggio è la produttività. Nessuna compagnia può permettersi di perdere improvvisamente la produttività di un ingegnere del software perché sta imparando un nuovo linguaggio. KBasic è molto simile a VB6, non una nuova completa sintassi e modello di programmazione. Ti consente di continuare a creare codice utile, applicando le caratteristiche gradualmente come le impari. Questa può essere una delle ragioni più importanti del successo di KBasic. In più, la maggior parte del tuo codice attuale Visual Basic 6 è ancora fattibile in KBasic.

Un semplice esempio

Basta! Ora hai una buona comprensione di tutto quello che è KBasic, smettiamo di discutere frasi astratte e studiamo un esempio concreto di codice KBasic. Ecco il preferito, amato da tutti, „Ciao Mondo!“:

```
' Inizio del programma

CLS
Print "Ciao Mondo!"
Print
Print
Print "                / `._      .        .       _.' \ "
Print "              '.@ = `.      \      /      .' = @.'"
Print "               \ @`.@ `.     \    /     .' @.'@ / "
Print "                \;`@`.@ `.    \ /    .' @.'@`;/  "
Print "                 \`.@ `.@ `'.(*).'` @.' @.'/    "
Print "                  \ '=._`. @ :=: @ .'_.=' /     "
Print "                   \ @  '.'..'='..'.'  @ /      "
Print "                    \_@_.==.: = :.==._@_/       "
Print "                    /  @ @_.: = :._@ @  \       "
Print "                   /@ _.-'  : = :  '-._ @\      "
Print "                  /`'@ @ .-': = :'-.@ @`'`\     "
Print "                  \.@_.=` .-: = :-. `=._@./     "
Print "                    \._.-'   '.'   '-._./       "
Print
Print "... hai appena fatto il tuo primo programma KBasic!"
```

Inserisci queste righe in una nuova finestra di codice KBasic. Dopo averlo eseguito in KBasic, vedrai una bella farfalla nello schermo.

Ok, ora andiamo al prossimo capitolo.

Sviluppo rapido di applicazioni

Puoi usare KBasic per le sue caratteristiche di programmazione visuale per sviluppare velocemente applicazioni. Nel Form Designer punti e clicchi per:

- Progettare l'interfaccia utente del tuo programma
- specificare il comportamento degli elementi dell'interfaccia utente
- Delineare la relazione tra l'interfaccia utente e il resto del tuo programma

In aggiunta alle sue caratteristiche di programmazione visuale, KBasic ti dà la capacità di eseguire rapidamente molti compiti, includendo:

- Creare nuovi elementi del programma. In KBasic, un elemento del programma è uno dei seguenti:
- Progetto: l'elemento di massimo livello di un programma in KBasic. Un progetto contiene classi, moduli e form.
- Form: la caratteristica visuale di KBasic
- Classe/modulo: un concetto del linguaggio KBasic. Le classi contengono metodi e altre istruzioni.
- Sub/function/method: (sottoprogramma/funzione/metodo, *ndt*) un concetto del linguaggio KBasic.
- Importare ed esportare codice da e verso il file system

KBasic ti dà gli strumenti di programmazione di cui hai bisogno per sviluppare codice industrial-strength (lett. di robustezza industriale, *ndt*). Specificatamente, tu puoi:

- usare il debugger visuale completamente integrato per esaminare il codice mentre è in esecuzione
- Sfogliare il codice a livello di progetto, classe o metodo

Come posso ottenere KBasic?

KBasic versione Professional

Puoi comprare KBasic Professional. Puoi ottenerlo direttamente dal suo produttore www.kbasic.com, dove puoi acquistarlo per 24.95 Euro (comprese spese di spedizione, circa 25 $). L'acquirente ottiene due versioni del programma: una per Windows, una per Mac OS X e Linux; e una licenza, che è per Windows, Mac OS X e Linux, e il diritto di scrivere quanti programmi commerciali si desiderano senza pagare royalty.
Se hai delle domande, scrivi un email a: sales@KBasic.com

KBasic versione Personal

C'è anche un edizione personale gratuita, con cui puoi provare tutti gli esempi di questo libro.

Download della *Personal Edition* su: www.kbasic.com/download.html

Installazione di KBasic / descrizione installazione / richieste del sistema

KBasic è facile da installare, ha un installer e necessita tutto sommato di circa 200 MB di spazio su disco rigido.

Ufficialmente supportati sono:
- Windows XP/2000
- Linux con KDE >= 3.3.3 (o Qt >= 3.3.3)
- Mac OS X (>= 10.2)

KBasic non girerà sui seguenti sistemi in futuro:
- Windows 95/98/ME

Hai bisogno di almeno 64 MB RAM e la tua CPU dovrebbe girare almeno a 400 MHz.
Risoluzione dello schermo: 1024X768, 32bit, True Color

Avvio di KBasic

Puoi lanciare KBasic facendo una delle seguenti:
- Su Linux, seleziona KBasic sul desktop
- Su Windows, seleziona KBasic nel menu programmi
- Su Mac OS X, seleziona KBasic sul desktop

Ora che KBasic è in servizio: dopo che avvii KBasic, appare la finestra principale di KBasic. Questa finestra è usata per accedere ad altre finestre, creare elementi del programma, e vedere il contenuto degli elementi del programma.

FAQ

Qual è la differenza tra KBasic e Visual Basic 6?

KBasic è molto simile a VB6, il che lo rende facile da imparare per gli sviluppatori Visual Basic 6. Ma ci sono anche alcune importanti differenze tra KBasic e VB6, es. Il pre-processore o il meccanismo di maneggiamento delle eccezioni. Una delle principali differenze tra KBasic e VB6 è che KBasic è un vero linguaggio di programmazione object-oriented con la possibilità di definire classi, di ereditare da esse e creare oggetti da classi. KBasic contiene tutti gli elementi della sintassi VB6 e supporta controlli e altri componenti in modo molto simile a VB6. Visual Basic 6 è completamente supportato poiché la sua sintassi è pienamente supportata. Inoltre, KBasic offre in più caratteristiche orientate agli oggetti ed estensioni.

Un programma KBasic contiene una o più classi, moduli, form o solo funzioni o sub (subroutine, *ndt*). Poiché in VB6 è possibile avere un namespace globale, funzioni globali o sub e variabili globali. KBasic non supporta un preprocessore, ma questo è previsto per il futuro. (es. '#ifdef' e '#if'). Teoricamente non è necessario, perché KBasic è un linguaggio di programmazione indipendente dalla piattaforma e perciò non esiste alcuna dipendenza legata alla piattaforma, che avrebbe bisogno di tale funzionalità. Inoltre, i tipi di dati semplici sono gli stessi, salvo che la loro dimensione potrebbe cambiare. Le dimensioni modificate sono: 'Boolean' è 1 Byte, 'Integer' è 32 bit e 'Long' è 64 bit in KBasic.

Posso importare automaticamente programmi Visual Basic 6 esistenti?

KBasic includerà un convertitore VB6 nella prossima major release. Nel frattempo puoi manualmente copiare i tuoi file e aprirli in KBasic.

In quali piattaforme è disponibile KBasic?

Al momento di scrivere questo libro esistono versioni di KBasic per Linux, Mac OS X e Windows.

Dove posso trovare maggiori informazioni su KBasic? Esempi, documentazione, e news?

Prima leggi questo libro. Oltre a questo, puoi ottenere news e nuove informazioni su www.kbasic.com. Là fuori in Internet ci sono molti esempi di programmazione. Molti siti di Visual Basic 6 o archivi BASIC si concentrano su utili esempi di programmi, che possono essere facilmente usati all'interno di KBasic con alcune piccole modifiche.

Sintassi KBasic

La sintassi di una sub, funzione o istruzione nella voce dell'help mostra tutti gli elementi, che sono necessari a usare correttamente la sub, funzione o istruzione. Come puoi capire queste informazioni è mostrato alle righe seguenti.

Esempio: Sintassi della funzione MsgBox

```
MsgBox(prompt[, pulsanti] [, titolo] [, file help, contesto])
```

Gli argomenti, che sono dentro [ ], sono opzionali. (Non scrivere queste [ ] nel tuo codice KBasic). L'unico argomento che devi dare alla funzione MsgBox è quello per mostrare il testo: 'prompt'.

Argomenti di funzioni o subroutine possono essere usati con l'aiuto del loro nome o posizione. Per usare gli argomenti non devi ignorare la posizione, come scritta nella sintassi. Li devi scrivere nell'ordine esatto in cui appaiono nella sintassi. Tutti gli argomenti devono essere separati da una virgola.

Esempio:

```
MsgBox("La risposta è corretta!", 0, "finestra con risposta")
```

Se desideri usare un argomento con il suo nome, utilizza il nome dell'argomento, i due punti e il segno uguale (:=) e il valore dell'argomento. Puoi scrivere questi argomenti con nome nell'ordine che preferisci. Esempio:

```
MsgBox(title:="finestra con risposta", prompt:="La risposta è
corretta!")
```

Alcuni argomenti sono scritti dentro le {}, nella sintassi di funzioni o sub.

```
Option Compare {Binary | Text}
```

Nella sintassi dell'istruzione 'Option Compare': { } assieme a | significa che uno degli elementi deve essere scritto. (Non scrivere queste { } nel tuo codice KBasic). La seguente istruzione definisce che il testo sarà confrontato e ordinato in modo case insensitive (insensibile alla forma, o grafia, maiuscola e minuscola, *ndt*).

```
Option Compare Text
```

Sintassi dell'istruzione 'Dim'

```
Dim NomeVar[([Indici])] [As Tipo] [, NomeVar[([Indici])] [As Tipo]]
...
```

'Dim' è una parola chiave nella sintassi dell'istruzione 'Dim'. L'unico elemento necessario è NomeVar (il nome della variabile). La seguente istruzione crea tre variabili: myVar, nextVar e thirdVar. Queste variabili sono dichiarate come 'Variant' automaticamente (o 'Double' in 'VeryOldBasic Mode').

```
Dim myVar, nextVar, thirdVar
```

L'esempio che segue dichiara una variabile di tipo 'String' (stringa, *ndt*). Se hai dichiarato esplicitamente il tipo di dato della variabile, questo aiuterà KBasic ad ottimizzare il consumo di RAM e a trovare errori nel tuo codice.

```
Dim myAns As String
```

Se vuoi dichiarare molte variabili in una riga, dovresti dichiarare esplicitamente il tipo di dato di ogni variabile. Variabili senza tipo di dato dichiarato prendono il tipo di default, che è 'Variant'.

```
Dim x As Integer, y As Integer, z As Integer
```

Nella seguente istruzione x e y hanno il tipo di dato 'Variant'. Solo z ha il tipo di dato 'Integer'.

```
Dim x, y, z As Integer
```

Devi mettere ( ) o [] (stile moderno), se vuoi dichiarare un array. Gli indici dell'array sono opzionali. La seguente istruzione dichiara un array dinamico di nome myArray.

```
Dim myArray(100)
Dim myArrayModernStyle[200]
```

KBasic non è solo un linguaggio di programmazione, ma anche tre.

Attraverso uno dei seguenti comandi puoi passare ad un modo:

- Se vuoi usare le nuove caratteristiche di KBasic (default)
```
Option KBasic
```
- Se vuoi usare il vecchio codice VB6
```
Option OldBasic
```
- Per codice BASIC molto vecchio simile a QBasic dovresti usare:
```
Option VeryOldBasic
```

E' possibile usare tutti e tre i modi nei tuoi programmi, es. un modulo usa un modo e un altro modulo ne usa un altro. Semplicemente metti una di queste righe in cima al tuo modulo. Di default è 'Option KBasic'.

Sviluppo software con KBasic

Una tipica applicazione KBasic consiste di form, moduli e classi e altri oggetti, che insieme sono la tua applicazione; form e loro controlli, e anche modificare dei valori nei controlli o cliccare sui pulsanti e generare eventi utente. Puoi reagire a tali eventi assegnando a questi codice KBasic.

Programmazione guidata da eventi vs. programmazione tradizionale

Quando un programma tradizionale, procedurale, è in esecuzione, l'applicazione in se controlla le parti del programma che dovrebbero essere eseguite, ignorando gli eventi. L'applicazione parte con il codice iniziale e attraversa il suo codice sorgente come lo ha definito lo sviluppatore. Se necessario vengono chiamate alcune procedure.

Poiché in applicazioni controllate dagli eventi un'azione è eseguita dall'utente o da un evento del sistema, che porta all'esecuzione della procedura-evento, l'ordine in cui il codice è eseguito dipende dall'ordine degli eventi, che dipendono dalle azioni dell'utente. Questo è il principio delle interfacce grafiche utente e della programmazione guidata dagli eventi. L'utente compie qualche azione e il programma reagisce a questo.

Come gira un'applicazione guidata dagli eventi?

Un evento è un azione, riconosciuta dai tuoi form o controlli. Gli Oggetti in KBasic conoscono alcuni eventi predefiniti e reagiscono a essi automaticamente. Per far reagire un controllo ad un evento, puoi scrivere una procedure-evento.

Tipicamente un'applicazione gira come segue:
1. L'utente avvia l'applicazione
2. Il form o controllo del form riceve un evento. L'evento può essere sollevato da un azione dell'utente (es. tasto premuto) o dal tuo codice (es. evento 'Open', se il tuo codice apre un form).
3. Se c'è una procedura-evento per l'evento, il codice evento desiderato viene eseguito
4. L'applicazione attende il prossimo evento

Alcuni eventi sollevano automaticamente altri eventi. Maggiori informazioni sono fornite nell'help di KBasic Software Atelier.

Tre passi di programmazione

Creare un semplice programma KBasic è più facile di pungolare un gatto in corsa. Hai bisogno solo di completare tre passi, dopo i quali avrai un programma che può girare fuori dall'ambiente di programmazione KBasic, come ogni altra applicazione tu possa aver installato nel tuo computer. I tre passi sono i seguenti:

1. Crea l'interfaccia utente del programma
2. Scrivi il codice sorgente, che fa fare al programma quello che si suppone debba fare
3. Compila il programma in un file eseguibile che può girare come applicazione standalone (autonoma, *ndt*) (in grado di avviarsi senza essere caricato in KBasic – solo nella versione professionale di KBasic). Questo può essere fatto dal compilatore del KBasic Software Atelier.

Naturalmente, ci sono molti dettagli coinvolti in ognuno di questi passi, specialmente se stai scrivendo un programma lungo. Come lavori, completerai questi passi nell'ordine in cui sono elencati sopra. Comunque, ritornerai anche frequentemente dal passo 2 a quello 1 per rifinire la tua interfaccia utente. Potresti persino tornare indietro dal passo 3 al passo 2 se scopri dei problemi dopo aver compilato il tuo programma.

Programmazione object oriented con KBasic

La programmazione object oriented è uno dei più importanti concetti degli ultimi anni ed è diventata una tecnica molto usata. Contiene questioni come

- Cosa sono classi e oggetti e come sono correlati tra loro
- Il comportamento e gli attributi di classi e oggetti
- Ereditarietà delle classi e come il design del programma ne è influenzato

Oggetti e classi

KBasic è un linguaggio di programmazione object oriented. L'orientamento agli oggetti è un argomento che è stato usato così tante volte, che semplicemente non ha alcun significato concreto. Object oriented vuol dire in termini KBasic i seguenti concetti:

- Classi e oggetti in KBasic
- Creare oggetti
- garbage collection per liberare oggetti (dalla memoria, *ndt*) non usati
- La differenza tra variabili di classi e variabili di istanza e la differenza tra metodi di classe e metodi di istanza
- Estensioni di classi per creare sottoclassi (classi figlie)
- Override (classe o metodo che scavalca il suo genitore, lett. scavalcamento, *ndt*) di metodi di classi e polimorfismo quando si arriva a chiamare i metodi
- Classi abstract (astratte, *ndt*)

Se sei uno sviluppatore Java o uno sviluppatore con esperienza in un altro linguaggio di programmazione object oriented, conoscerai molti dei concetti di KBasic. Se non ne conosci alcuno, non devi essere spaventato, poiché non è per niente difficile fare progressi con oggetti e classi.

Programmazione orientata agli oggetti significa che ogni oggetto è parte di un altro oggetto. Esattamente come nel mondo reale, le automobili consistono di molti oggetti come finestrino, sterzo e così via.

C'è solo una classe (piano di costruzione) per il finestrino e lo sterzo. Perciò esiste una classe finestra e una classe sterzo, ma molti oggetti finestra o sterzo, che sono istanze di una classe. Un'istanza di una classe è anche chiamata oggetto di una classe. Le classi esistono quando scrivi il tuo codice, gli oggetti sono creati quando il tuo codice viene eseguito da KBasic usando le tue classi!

Ogni classe ha i seguenti elementi:
●Classe genitore
●Attributi (variabili, costanti...)
●Metodi

Classe genitore significa che tutti gli attributi e metodi della classe genitrice sono anche nella nuova classe in aggiunta agli attributi e metodi che hai dichiarato in questa nuova classe.
Gli attributi possono essere proprietà, costanti o variabili, contenuti in una classe.

I metodi sono procedure e funzioni di una classe. I metodi sono plausibilmente la parte più importante di ogni linguaggio object oriented. Laddove classi e oggetti forniscono la struttura, e classi e variabili d'istanza offrono il modo di conservare gli attributi di quella classe o oggetto, i metodi effettivamente forniscono il comportamento di un oggetto e definiscono come un oggetto interagisce con gli altri oggetti del sistema.

Variabili e metodi possono essere associati ad una classe (una volta in memoria) o associati ad una istanza (per quante istanze della classe esistono) nella dichiarazione di una classe (al di fuori di un metodo):

- Variabile d'istanza
  ```
  Dim instanceVar
  ```
- Metodo d'istanza
  ```
  Sub instanceSub()
  ```
- Variabile di classe
  ```
  Static staticVar
  ```
- Metodo di classe
  ```
  Static Sub staticSub()
  ```

Metodo di classe o variabile di classe significa che possono essere usati senza alcun oggetto creato. Metodo d'istanza o variabile d'istanza vuol dire che possono essere usati solo assieme ad un oggetto. Le variabili d'istanza sono come le variabili locali di una procedura . Le variabili locali esistono solo dopo la chiamata della procedura, le variabili d'istanza esistono solo per tutto la durata della vita di un oggetto. Le variabili di classe sono globali e disponibili in qualunque momento. Le variabili di classe sono buone per comunicare tra differenti oggetti nella stessa classe o per tenere traccia di stati globali fra un insieme di oggetti.

Le classi possono contenere anche costanti, è esattamente come dichiarare le variabili. Le costanti sono dichiarate nell'area dichiarazione di una classe.

Le classi possono dare tutti i loro elementi ai figli, il che è chiamato ereditarietà.

Ereditarietà delle classi

L'ereditarietà è la parte più importante della programmazione orientata agli oggetti.

KBasic supporta l'ereditarietà singola, che significa che ogni classe eredita da una classe genitore. Non è possibile ereditare da molte classi. Questo è come in Java. In un lontano futuro, sarà possibile usare interfacce per fare molte delle cose che puoi fare con l'ereditarietà multipla. Come in Java, tutti gli oggetti in KBasic sono creati con 'New'.

Una classe è una collezione di dati e metodi, che lavorano su questi dati. Dati e metodi sono usati per descrivere il contenuto e il comportamento di un oggetto.

Gli oggetti sono istanze di una classe: Non puoi fare molto con una singola classe, ma molto di più con un'istanza di una classe, un oggetto singolo, che contiene variabili e metodi.

Significa esattamente questo: object-oriented vuol dire che gli oggetti sono al centro, non le procedure e qualche variabile.

Sintassi KBasic

Se vuoi leggere e scrivere velocemente dei programmi, devi conoscere la sintassi di KBasic. La sintassi definisce come scrivere programmi. Come scrivere ogni elemento del linguaggio, come funzionano assieme e come usarli. Quindi nelle seguenti pagine conoscerai tutti gli elementi del linguaggio.

Istruzioni ed espressioni

Le istruzioni sono delle azioni complete in KBasic. Possono contenere parole chiavi, operatori, variabili, costanti ed espressioni. Ogni istruzione può essere classificata in una di queste tre:

- Istruzioni come dichiarazioni o definizioni di classi, moduli, variabili, costanti, procedure, funzioni, metodi, proprietà
- Istruzioni di assegnazione, che assegnano un valore o un'espressione ad una variabile o costante, oppure proprietà
- Istruzioni eseguibili: che compiono delle azioni. Queste istruzioni chiamano un metodo, funzione, procedura o loop (ciclo, *ndt*) o una condizione. Le istruzioni eseguibili spesso contengono operatori condizionali o matematici (persino espressioni complesse).

Istruzioni multilinea

Normalmente, ogni istruzione sta in una riga, ma qualche volta è più leggibile mettere un'istruzione in molte righe. Se è così, puoi usare la linea continua underscore (_). Nell'esempio che segue si usa (_) :

```
Sub field()
      Dim myVar As String
      myVar = "Bernd"
      MsgBox Prompt:="Ciao " & myVar, _
            Title:="Ciao"
End Sub
```

Puoi scrivere molte istruzioni in una sola riga usando i due punti (:). Per esempio:

```
Print „Hi": Print „Ho"
```

Variabili e tipi di dati

Mentre si elaborano espressioni è spesso necessario memorizzare valori per poco tempo. Per esempio, vorresti calcolare valori diversi, confrontarli, e a seconda di essi, eseguire diverse operazioni. Ma per confrontare diversi valori hai bisogno di memorizzarli. Memorizzare non significa che li salvi su disco, ma in memoria, poiché ne hai bisogno durante l'esecuzione del tuo programma.

KBasic usa variabili per immagazzinare valori durante esecuzione. Dopo che il programma smette di girare tutte le variabili (persino gli oggetti) sono cancellati. Una variabile può modificare il suo valore in qualsiasi momento. Puoi cambiare il suo valore assegnandoli un'espressione, un'altra variabile o una costante. Una variabile è uno spazio in memoria usato da KBasic per conservare dei valori, quindi le variabili non sono memorizzate in un file. Come un file, una variabile ha un nome (usato per accedere alla variabile e per identificarla) e anche un tipo di dato, che specifica il tipo d'informazione che una variabile contiene.

KBasic conosce le variabili d'istanza, le variabili di classe, le variabili globali e le variabili locali. Le variabili d'istanza sono parte di un oggetto, le variabili di classe sono parte di una classe, le variabili globali possono essere usate dappertutto e le variabili locali sono parte di una procedura, funzione o subroutine (chiamata anche metodo nelle classi).

Dichiarazione di variabili / dichiarazione esplicita

Prima di poter usare le variabili, le devi dichiarare, il che vuol dire che devi definirne il nome e il tipo di dato.

L'istruzione 'Dim' dichiara una variabile. 'Dim' è una delle tante istruzioni di dichiarazione di variabili, che differisce leggermente dalle altre. Altre istruzioni di dichiarazione sono 'Redim', 'Static', 'Public', 'Private', 'Protected' e 'Const'.

```
Dim myName As String
```

Le variabili possono essere dichiarate dappertutto, in procedure, funzioni e subroutine, e non solo in cima alle procedure, funzioni e sub. Ma la maggior parte del tempo sono in testa.

```
Sub doEvent()
  Dim myName As Integer
End Sub
```

Ogni linea del codice sorgente può contenere molte istruzioni di dichiarazione di variabili. Che tipo di dato hanno le variabili dipende dal modo in cui le dichiari. Se dai ad ogni variabili esplicitamente uno specificatore di tipo, non sarà usato il tipo di default. Il tipo di dato di default dipende dal modo che hai scelto. In 'KBasic Mode' e 'OldBasic Mode' è 'Variant', in 'VeryOldBasic' è 'Double'.

```
Dim firstname, surname, lastname As String
```

firstname è di tipo 'Variant'
surname è di tipo 'Variant'
lastname è di tipo 'String'

O esplicitamente con differenti tipi:

```
Dim myName As String, age As Integer
```

Se desideri usare un tipo di dato, devi scriverlo per ogni variabile, altrimenti sarà usato il tipo di dato di default. Nella seguente istruzione tutte le variabili sono dichiarate come 'Integer'.

```
Dim intX As Integer, intY As Integer, intZ As Integer
```

Nell'istruzione che segue intX e intY sono dichiarate come 'Variant' e intZ come 'Integer'.

```
Dim intX, intY, intZ As Integer
```

Le variabili possono avere un valore iniziale. Per fare ciò, hai due strade:

O

```
Dim name = „sunny sun“ As String
```

Oppure

```
Dim name As String = „sunny sun“
```

Qui sono dichiarate molte variabili in una riga, solo lastname ha un valore „sunny sun“.

```
Dim name, sirname, lastname = „sunny sun“ As String
```

Se non viene assegnato nessun valore dalla dichiarazione, è usato quello di default, che è 0 se è una variabile numerica. Se è una stringa si usa una stringa nulla di lunghezza 0 (““). Le variabili Object sono impostate a 'Null' (cioè non referenziano a nessuna classe, *ndt*). Le Variant sono 'Empty'.

I nomi delle variabili non devono essere più lunghi di 128 caratteri e non devono usare tutti i caratteri possibili. Punto, virgola e altri caratteri non stampabili non devono essere usati, solo l'underscore (_) (sottolineato, *ndt*) è consentito.
Importante! Normalmente KBasic non fa differenza se il nome di una variabile è scritto in minuscolo o maiuscolo, infatti puoi scriverlo diversamente ad ogni riga, ma per essere compatibili con le future versioni, dovresti scrivere sempre uguale il nome delle variabili. Esistono alcune eccezioni, se usi i binding, hai sempre scritto il nome dei metodi come menzionato nella documentazione.
Quindi Named, named, naMED sono collegate alla stessa variabile.
Usa le regole seguenti, se vuoi nominare procedure, funzioni, subroutine, costanti, variabili o argomenti in KBasic:

- Usa una lettera (A-Z, a- z) o underscore (_) come primo carattere
- Non usare mai lo spazio ( ), il punto (.), il punto esclamativo (!), o i seguenti caratteri @, &, $, #, „ nei nomi.
- Il nome può contenere numeri, ma non deve iniziare con un numero
- generalmente, non usare nomi che sono già impiegati da elementi predefiniti del linguaggio KBasic, come parole chiavi (es. 'If'), built-in (es. 'Print'), classi ecc., poiché condurrebbe ad una collisione di nomi, in cui ci si aspetta l'elemento originale del linguaggio, cosicché il tuo programma non compilerà.
- Non puoi usare variabili con lo stesso nome nello stesso contesto. es. Non puoi dichiarare la variabile *age* due volte in una procedura, ma puoi dichiararla in una procedura e in un modulo allo stesso tempo, perché sono dichiarate in contesti differenti (in luoghi differenti).
- Non devi usare parole riservate, come parole chiavi o predefinite ('If' o 'Print')

Dichiarazioni di variabili in ambiti differenti

Puoi mettere un'istruzione di dichiarazione dentro una procedura, funzione, subroutine, metodo per dichiarare una variabile in un contesto locale. In più, puoi collocare una dichiarazione di variabile globale, variabile di classe, variabile d'istanza nella sezione dichiarativa di una classe o modulo. La posizione della dichiarazione determina il contesto della variabile. La visibilità di una variabile non può essere modificata durante l'esecuzione del programma, ma puoi avere diverse variabili con lo stesso nome in posti diversi. Per esempio, se hai la dichiarazione di una variabile di nome *price* in una procedura, e la stessa dichiarazione in un modulo, tutti gli usi di questa variabile nella procedura sono connessi alla variabile locale price, tutti gli usi all'esterno della procedura sono connessi alla variabile globale price.

Una variabile di nome sName e di tipo 'String' è dichiarata nel seguente esempio:

```
Dim sName As String
```

Se questa istruzione è parte di una procedura, allora è possibile usare questa variabile solo nella stessa procedura. Se l'istruzione è parte della sezione dichiarativa di un modulo, puoi usarla in tutte le procedure del modulo, ma non all'esterno del modulo (infatti è dichiarata come 'Private' implicitamente). Per avere la possibilità di usarla dappertutto puoi usare la parola chiave 'Public'. Esempio:

```
Public sName As String
```

Uso dell'istruzione 'Public'

Puoi usare l'istruzione 'Public' per dichiarare una variabile pubblica in un contesto di modulo o classe, il che significa che sono accessibili dovunque.

```
Public sName As String
```

Uso dell'istruzione 'Private'

Puoi usare l'istruzione 'Private' per dichiarare variabili private in un contesto di modulo o classe, che significa che sono accessibili solo dallo stesso ambito (visibilità di modulo, tutte le procedure del modulo, visibilità di classe, tutti i metodi della classe).

```
Private myName As String
```

Se l'istruzione 'Dim' è usata in un contesto di modulo o classe, è trattata come un'istruzione 'Private'. Probabilmente, vuoi usare l'istruzione 'Private' per avere codice più chiaro e leggibile.

Uso dell'istruzione Protected

Puoi usare l'istruzione 'Protected' per dichiarare variabili protette in un contesto di classe, il che vuol dire che sono accessibili solo dallo stesso ambito (visibilità di classe, in tutti i metodi della classe, nelle sottoclassi, nei metodi delle sottoclassi). E' utile se vuoi evidenziare la gerarchia di ereditarietà delle tue classi.

```
Protected myName As String
```

Uso dell'istruzione 'Static'

Static ha tre diversi significati, a seconda del contesto in cui è usata.

- *Static dentro una classe, ma all'esterno di un metodo*
  Se usi un'istruzione 'Static' invece di una 'Dim', la variabile è dichiarata come variabile di classe, il che significa che può essere usata senza un'istanza (di oggetti) di questa classe. Una variabile di classe esiste solo una volta durante l'esecuzione.
- *Static all'esterno di una classe, ma dentro una procedura (subroutine o funzione) o metodo*
  Se usi un'istruzione 'Static' invece di una 'Dim', la variabile è dichiarata come locale statica, il che vuol dire che una volta che la variabile è stata dichiarata, non è distrutta abbandonando la procedura. La prossima volta che si entra nella procedura, il valore della variabile esiste ancora. Perciò, una variabile locale statica è dichiarata solo una volta quando si usano chiamate ricorsive ad una procedura.
- *Static all'esterno di una classe, ma prima di una procedura (sub o funzione) o metodo*
  Se usi un'istruzione 'Static' prima della parola chiave 'Sub' o 'Function' tutte le variabili locali sono dichiarate come 'Static', il che significa che una volta che una variabile è stata dichiarata, essa non è distrutta dopo aver lasciato la procedura. La prossima volta che si entra nella procedura il valore della variabile esiste ancora. Perciò una variabile locale statica è dichiarata solo una volta quando si usano chiamate ricorsive ad una procedura.

Puoi usare la parola chiave 'Static' con altre parole chiavi ('Public', 'Protected', 'Private').

Dichiarazione di variabile automaticamente / dichiarazione implicita

Per ragioni storiche, è possibile usare variabili senza una dichiarazione. E' supportato in un modo speciale, in cui tutte le variabili sono dichiarate automaticamente, quando usi il nome di una variabile, che non è stato usato prima. Importante! Non dovresti usare questo metodo, a meno che non usi del vecchio codice BASIC, che vuoi ancora utilizzare senza modificarlo. Un buon stile di programmazione è dichiarare tutte le variabili con il loro tipo, che rende più facile agli altri capire il tuo codice. Eviti inoltre errori di battitura, quando scrivi diversamente il nome di una variabile (intendendo la stessa variabile) per errore.

Per attivare la dichiarazione implicita, scrivi le seguenti righe in cima al tuo programma:

```
Option OldBasic
Option Explicit Off
```

Uso dell'istruzione 'Option Explicit'
(supportata solo in 'Mode OldBasic' e 'Mode VeryOldBasic')

Come menzionato nel paragrafo precedente, puoi dichiarare implicitamente una variabile in KBasic, semplicemente usa un'istruzione di assegnamento o un'espressione con un nome di variabile, che desideri utilizzare. Tutte le variabili dichiarate implicitamente hanno il tipo di dato 'Variant' ('Option OldBasic') o 'Double' ('Option VeryOldBasic'). Le variabili di tipo 'Variant' necessitano di più spazio in memoria della maggior parte degli altri tipi di dati. Il tuo programma è più veloce quando dichiari esplicitamente le variabili con il tipo di dato più piccolo possibile. Un altro vantaggio è che usando una dichiarazione eviti collisioni di nomi o errori di battitura.

Se vuoi dichiarare esplicitamente in 'Option OldBasic' o 'Option VeryOldBasic', dovresti scrivere in cima a tutti moduli e file di classe 'Option Explicit On' cosicché viene sollevato un errore di compilazione, quando una variabile non è dichiarata o è digitata male.

Se stai usando il 'KBasic Mode', devi dichiarare tutte le variabili. E' come 'Option Explicit On'. Ad ogni modo 'KBasic Mode' è impostato di default in tutti i programmi.

Importante! Devi dichiarare ogni volta array dinamici o fissi. Inoltre puoi mischiare i modi nel tuo programma; in un file è impostato 'Option OldBasic', nell'altro file 'Option VeryOldBasic'. Sei libero d'impostare 'Option Explicit Off' per i vecchi modi.

Variabili locali

Il valore di una variabile locale è disponibile solo dentro la procedura. Non puoi accedere a questo valore all'esterno della procedura. Quindi è possibile avere una variabile con lo stesso nome in differenti procedure, senza collisione di nomi.

Esempio:

```
Sub test1()
  Dim i As Integer

  i = 1
End Sub

Sub test2()
  Dim i As Integer

  i = 19
End Sub
```

Istruzione di assegnamento

L'istruzione di assegnamento attribuisce alla variabile un valore o un'espressione con variabili, costanti, numeri ecc. L'assegnamento include sempre il segno (=). L'esempio seguente mostra un' assegnamento.

```
Dim yourName As String
yourName = InputBox("Come ti chiami?")
```

```
MsgBox "Ti chiami " & yourName
```

Non hai bisogno della parola chiave 'Let'. E' obsoleta e opzionale. Un esempio mostra come usare 'Let':

```
Let yourName = InputBox("Come ti chiami?")
```

L'istruzione 'Set' è usata in VB6 per assegnare un oggetto ad una variabile oggetto. 'Set' non è necessaria in KBasic e non deve essere usata. Semplicemente tralasciala.

Dopo la dichiarazione di una variabile, puoi usarla, es. facendo un assegnamento (=).

```
Dim Cool As Boolean
Dim myName As String

Cool = True
Name = „Julie“
```

Cool contiene 'True' ('vero', *ndt*), Name contiene 'Julie' dopo l'assegnamento.

Tempo di vita di una variabile

Il tempo di vita di una variabile è il tempo di esistenza di una variabile o in cui ha un valore. Il valore di una variabile può essere cambiato durante la sua esistenza. Una variabile fuori dal suo campo di azione non ha più un valore. Nel caso di una procedura, una variabile esiste dopo l'istruzione 'Dim'. Le variabili locali esistono fino a che non si lascia la procedura, dopo che si rientra nella procedura sono tutte ricreate dopo l'istruzione 'Dim'.

Se non cambi il valore di una variabile durante il suo tempo di vita, contiene il valore inizializzato all'avvio del programma. Così una variabile dichiarata in una sub contiene il valore assegnato finché non si lascia la subroutine. Se la stessa sub viene richiamata, il valore viene impostato nuovamente a quello d'inizializzazione.

Se usi variabili dichiarate static nelle procedure, esistono solo una volta in memoria, a differenza di due o più volte quando una procedura è chiamata usando la tecnica ricorsiva. Se usi variabili di modulo o variabili di classe dichiarate globali, è lo stesso, esistono sempre e solo una volta in memoria in fase di esecuzione. Se usi variabili di istanza devi considerare che esistono solo assieme alla loro istanza (oggetto) a cui appartengono.

Punto di dichiarazione

Dovresti dichiarare ogni variabile e procedura dentro una classe o almeno all'interno di un modulo.

Sintassi:

```
Dim Nome[([Indice])] [As Tipo] [, Nome[([Indice])] [As Tipo]] ...

Dim Nome [= Espressione] [As Tipo]

Dim Nome [As Tipo] [= Espressione]

[Public | Protected | Private | Dim | Static] Nome [= Espressione]
[As Tipo]
```

Tipi di dati

I tipi di dati descrivono il tipo di dato che è memorizzato nella variabile. Se dichiari una variabile devi anche dargli un tipo. KBasic supporta tutti i tipi di dati di VB6 e molti di più. Puoi usare uno dei seguenti tipi:

- Un tipo di dato semplice (es. 'Integer')
- Il nome di una classe predefinita (classe KBasic, classe di binding)
- Una classe definita dall'utente
- Un tipo di dato definito dall'utente
- Un'enumerazione definita dall'utente

Tipi di dati semplici

Possono memorizzare numeri e testo. Sono chiamati semplici perché sono predefinite in KBasic e non sono complessi come le classi. Ogni tipo di dato semplice ha dei confini (un range, *ndt*), che rappresenta il più piccolo e il più grande numero che può essere memorizzato. Se il valore è troppo grande, perde di precisione, che significa che perde informazione!

La seguente tabella contiene tutti i tipi di dati supportati da KBasic con il loro necessario spazio in memoria.

Tipo di dato */ Dimensione / Valore*

- *Boolean (booleano)* / 1 / 'True' (vero) o 'False' (falso)
  sono memorizzati come numeri a 8 bit (1 Byte), ma che possono essere solo 'True' o 'False',
- *Byte* / 1 / 0..255
  sono memorizzati come numeri a 8 bit (1 Byte) senza segno, che possono essere >= 0 e <= 255
- *Short (corti)*/ 2 / -2^15 fino a 2^15-1
  sono immagazzinati come numeri a 16 bit (2 Byte). I valori possono essere -32.768 o 32.767 oppure tra -32.768 e 32.767

- *Integer (intero)*/ 4 / -2^31 fino a 2^31 –1
  sono memorizzati come numeri a 32 bit (4 Byte). I valori possono essere -2^31 >= e <= +2^32. Puoi usare variabili Integer per simulare enumerazioni. Come 0 = black, 1 = white e così via. Ad ogni modo, non dovresti usarlo per enumerazioni come in VB6, meglio usare il nuovo approccio con la keyword 'Enum'. Il tipo di suffisso è (%).
- *Long (lungo)*/ 8 /-2^63 bis 2^63 –1
  Sono immagazzinati come numeri a 64 bit (8 Byte). I valori possono essere -2^63 >= e <= +2^64. Il tipo di suffisso è (&).

- *Int16* / 2 /
  Intero con segno con valori fino a 2^16
- *Int32* / 4 /
  Intero con segno con valori fino 2^32
- *Int64* / 8 /
  Intero con segno con valori fino 2^64
- *UInt16* / 2 /
  Intero con segno con valori fino 2^16
- *Uint32* / 4 /
  Intero con segno con valori fino 2^32
- *Uint64* / 8 /
  Intero con segno con valori fino 2^64

- *String (Stringa)*/ dimensione che dipende dalla lunghezza della stringa / illimitata

Ci sono due tipi di stringhe. Stringhe con lunghezza fissa e con lunghezza dinamica. Le stringhe dinamiche possono essere lunghe sino a 2^31 caratteri. Normalmente, stringhe o argomenti stringa sono dinamici. Le stringhe dinamiche diventano più piccole o più grandi quando si assegnano nuovi valori.

- *Stringa* / lunghezza fissa / lunghezza di una stringa
  Le stringhe fisse possono essere lunghe fino a 2^31 caratteri. Le stringhe fisse sono usate solo assieme alla keyword 'Type'. Il tipo di sufisso è ($).
  myStringVar = „1234 Names symbols .!/& keywords IF ? - “
- *Character (carattere)* / 2 /4
  riservato per il futuro.
- *Single* / 4 /
  Sono memorizzati come numeri in virgola mobile (4 Byte) a singola precisione. I valori possono essere tra -3,402823^38 o -1,401298^-45 o tra 1,401298^-45 fino a 3,402823^38
  Il suffisso è (!).
- *Double* / 8 /
  Sono memorizzati come numeri in virgola mobile (8 Byte) con doppia precisione. I valori possono essere tra -1,79769313486232^308 fino a -4,94065645841247^-324 o tra 4,94065645841247^-324 fino a 1,79769313486232^308
  il suffisso è (#).
- *Object (oggetto)* / 4 / + dimensione dell'oggetto
  Sono immagazzinati come reference a 32bit (4 Byte).
- *Date/Time* / 8
  Sono memorizzati come numeri (8 Byte). Una data letterale deve essere del tipo #yyyy-mm-dd# .
  Un tempo letterale deve essere del tipo #hh:mm:ss# .
- *Currency (valuta)* / 8
  Sono immagazzinati come numeri (8 Byte). I valori possono essere tra -922.337.203.685.477,5808 e 922.337.203.685.477,5807. Il suffisso è (@).

- *Variant* / 40
  Una variabile 'Variant' è sempre grande 40 byte.
- *Classi* /
  vedi Object
- Tipi di dati definiti dall'utente / vedi il prossimo capitolo
  la dimensione dipende dai suoi elementi

Tipi definiti dall'utente

Sono stati usati pesantemente negli anni '80 e '90, ma oggi fanno un passo dietro gli oggetti ordinari. I tipi definiti dall'utente sono un gruppo di variabili di tipo differente memorizzati in una variabile. Questi possono essere semplici tipi di dati o persino tipi di dati definiti dall'utente.

Sintassi:

```
Type Nome
 Nome [(Indice)] As Tipo
  ...
End Type
```

Tipi Classi/oggetti:

Le variabili possono memorizzare oggetti (in verità, riferimenti a oggetti). Queste sono variabili di tipo 'Object' e hanno dimensione 4 byte. A differenza di VB6, l'assegnamento di oggetti ad una variabile non necessita della parola chiave 'Set'. L'istruzione '=' è riconosciuta correttamente da KBasic.

Sebbene una variabile oggetto possa contenere qualsiasi oggetto (in effetti, il riferimento a quell'oggetto), il binding (collegamento con il reference, *ndt*) è eseguito in run-time (in questo caso si parla di *late binding* [binding tardivo], *ndt*). Se preferisci il binding precoce (*early binding*), puoi usare una classe che eredita l'oggetto della classe, come TextBox. Puoi forzare il binding precoce usando anche un'istruzione di cast.

Tipo Variant

Il tipo 'Variant' è usato automaticamente se non specifichi un tipo di dato per un argomento, costante, procedura o variabile. Ma c'è un eccezione, se usi una variabile nel modo 'VeryOldBasic' non ha il tipo di dato 'Variant', ma 'Double'. Variabili di tipo 'Variant' possono contenere stringhe, date, valori di tempo, booleani o persino numerici. La dimensione di un 'Variant' è 40 byte. L'accesso a tipi di dati variant è lento.

Il seguente esempio crea due variabili di tipo 'Variant':

```
hisVar = 3

Dim myVar

Dim yourVar As Variant
```

La prima istruzione dichiara implicitamente una variabile (automaticamente).

In più, un 'Variant' può memorizzare i seguenti valori:

- 'Null' (un riferimento a nessun oggetto, *ndt*)
- 'Empty' (nessun valore, *ndt*)

Suffisso delle variabili

I nomi di variabili possono essere usati per determinarne tipo. Storicamente puoi appendere un simbolo speciale che mostra il tipo di dato della variabile. KBasic supporta questo vecchio stile di programmazione per ragioni storiche.

Esempio:

```
Dim i% ' variabile Intera
Dim l& ' variabile Long
Dim s$ ' variabile Stringa
```

Sono supportati i seguenti simboli.

- *Integer*
  il tipo di segno per 'Integer' è (%).
- *Long*
  il tipo di segno per 'Long' è (&).
- *String*
  il tipo di segno per 'String' è ($).
- *Single*
  il tipo di segno per 'Single' è (!).
- *Double*
  il tipo di segno per 'Double' è (#).
- *Currency*
  il tipo di segno per 'Currency' è (@).

Commenti

Il simbolo di commento (') è usato in molte linee di codice in questo libro. I commenti possono spiegare una procedura o istruzione. KBasic ignora tutti i commenti mentre compila ed esegue il tuo programma.

I commenti sono utili per descrivere procedure o istruzioni. KBasic ignora tutti i commenti mentre esegue il tuo programma. Se vuoi scrivere un commento, scrivi il simbolo (') e direttamente dopo il testo del commento. I commenti sono stampati su schermo in verde.

KBasic conosce quattro modi per scrivere dei commenti. Il simbolo ‘ e REM es.

```
REM questo è un commento

‘ anche questo è un commento
```

E come in Java

```
/* inizio del commento e fine del commento */

/** inizio del commento e fine del commento */
```

I commenti sono estremamente utili quando si tratta di far capire il tuo programma a te o ad altri programmatori. Quindi i commenti, normalmente, descrivono come funziona il tuo programma.

Convenzioni per i nomi

Quando programmi in KBasic, dichiari e nomini molti elementi come procedure (funzioni e sub), variabili e costanti, e così via. Per questi elementi, ci sono delle regole su come nominarli. Le regole sono:

Tutti i nomi

- devono iniziare con una lettera
- devono contenere lettere, numeri o il segno (_); punti, persino le virgole sono proibite
- non devono essere più lunghi di un'ampiezza definita
- Non devono contenere parole riservate

Una parola riservata è una parola che è parte di KBasic, con un significato predefinito. Keyword (es. 'If' o 'Then'), funzioni built-in (es. 'Mid') e operatori (es. 'Mod'). Una lista completa di parole riservate è inclusa alla fine di questo libro.

Literal

Oltre a parole chiavi, simboli e nomi, un programma KBasic contiene literal. un literal è un numero o stringa che rappresenta un valore. Es. 12, „Hi“, true,

Ci sono diversi literal numerici.

- *Byte, Short, Integer, Long, Int16, Int32, Int64, Uint16, Uint32, Uint64*
  1, 2, -44, 4453
- *Hex*
  &HAA43
- *Binary*
  &B11110001
- *Octal*
  &O1234
- *Single (Decimale)*
  sempre in formato Inglese
  21.32, 0.344, -435.235421.21
- Double (Decimale)
  sempre in formato Inglese
  212.23
- Currency
  sempre in formato Inglese
  45.3@
- *Date / Time*
  sempre in formato Inglese
  #January 1, 1993#
- *String*
  E' semplice testo ma deve iniziare con un (“) e finire con (“), cosicché KBasic può riconoscerlo come stringa. Un doppio (“) dentro una stringa è interpretato come singolo (“). Non ci sono sequenze di escape come in C++. Possono essere usate adoperando funzioni predefinite.

  „hello“

- Unicode sarà supportato in futuro.
- *Boolean*
  'True' o 'False'
  Quando fai il cast di un valore numerico in booleano. Valori con 0 sono 'False'. Valori con -1 sono 'True'.

Espressioni

Espressioni significano valori. Possono contenere parole chiavi, operatori, variabili, costanti o perfino altre espressioni. Esempi di espressioni sono:

- 1 + 9 (numero operatore numero)
- myVar (variabile)
- myVar TypeOf Object (variabile keyword nome di classe)

Le espressioni possono restituire un valore o No. Se un espressione è alla destra di un'istruzione di assegnamento (=), l'espressione da o deve rendere un valore.

```
myVar = 1 + 9
```

1 + 9 è 10 e questo valore dell'espressione (10) è memorizzato in myVar.

E' esattamente lo stesso, quando l'espressione è usata come parametro in una chiamata di funzione

```
MyProcedure(1 + 9)
```

Le espressioni possono essere calcolate in fase di esecuzione e rappresentano un valore. Il risultato può essere usato per assegnarlo ad una variabile o ad un'altra espressione come abbiamo vista sopra.

Costanti

Le costanti sono come le variabili eccetto che non possono modificare i loro valori. Quando dichiari una costante tu gli assegni un valore, che non può essere alterato durante tutta la vita del programma. Esempio:

```
Const border As Integer = 377
```

Questa istruzione 'Const' dichiara una costante border di tipo 'Integer' e con un valore di 377.

Ci sono le stesse regole per dichiarare le costanti come per le variabili. Un'istruzione 'Const' ha la stessa visibilità di una variabile. Per dichiarare una costante dentro una procedura, devi porre l'istruzione 'Const' dentro questa procedura (normalmente una delle prime istruzioni). Per dichiarare una costante accessibile da tutte le procedure, devi metterla in un modulo (o classe) dove si trovano le procedure.

Quest'istruzione 'Const' dichiara una costante conAge come public con un tipo di dato 'Integer' e un valore di 34.

```
Public Const conAge As Integer = 34
```

Le costanti possono contenere tutti i tipi di dati semplici, che sono i seguenti: Boolean, Byte, Short, Integer, Long, Int16, Int32, Int64, Uint16, Uint32, Uint64, Currency, Single, Double, Date, String o Variant.

Puoi anche dichiarare molte costanti in una riga.

```
Const conAge As Integer = 39, conSalary As Currency = 3500
```

Questa istruzione 'Const' dichiara le costanti conAlter e conSalary.

Sintassi:

```
Const Nome = Espressione

Const Nome [As Tipo] = Espressione [, Nome [As Tipo] = Espressione]
...

[Public | Protected | Private] Const Nome [As Tipo] = Espressione
```

Operatori e Operandi

KBasic supporta tutti gli operatori VB6 e molti di più. Operatori sono '+' o '-' per esempio, per cui gli operatori sono utili per eseguire operazioni come sommare o sottrarre.
Operandi sono numeri, stringhe o variabili (o espressioni). KBasic supporta l'overload degli operatori per i Binding. In futuro sarà possibile utilizzare l'overloading degli operatori dentro classi KBasic come in C++.

Ci sono differenti tipi di operatori.

Operatori di calcolo

Sono usati per il calcolo di due operandi. Ognuno di questi operatori ha bisogno di due operandi. Il meno (-) può essere usato per negare i valori, come -9 significa 9 negativo, +9 significa 9 positivo, 9 vuol dire 9 positivo.

Gli operatori di calcolo sono:

- *+ Addizione*
  Utile per addizionare due numeri o stringhe. Se sommi un numero e una stringa KBasic non può essere ogni volta sicuro, se il risultato debba essere un numero o una stringa. Quindi è meglio usare l'operatore '&' invece di '+' per le stringhe.
  es. `3 + 4`
- *- Sottrazione*
  Utile per sottrarre un numero da un altro numero o per ottenere il valore negativo di un'espressione numerica.
  es. `5 - 7`
- ** Moltiplicazione*
  Utile per moltiplicare due numeri
  es. `33 * 7`
- */ Divisione*
  Utile per dividere due numeri. Il risultato è in virgola mobile.
  es. 2 / 5
- *\ Divisione intera*
  Il risultato è di tipo 'Integer'.
  es. `29 / 5`
- *Mod Modulus, anche noto come resto di una divisione intera*
  Da il resto del risultato di una divisione intera
  es. `45 Mod 10`

Quindi una divisione Intera ha come esito un valore 'Integer'.

Esempio:

x is 10, y is 4

```
Print x + y      ' è 14
Print x - y      ' è 6
Print x * y      ' è 40
Print x / y      ' è 2.5
Print x \ y      ' è 2
Print x Mod y    ' è 5
```

Normalmente il risultato ha il tipo di dato dell'operando (= espressione) con più precisione.

Operatori di assegnamento

Nel momento in cui scrivo KBasic supporta più del normale operatore di assegnamento (=). Come in C++, supporta gli operatori di assegnamento estesi.
+= Istruzione somma (es. `var += 1` è come `var = var + 1`)
-= Istruzione di sottrai (es. `var -= 1` è come `var = var - 1`)
/= Istruzione di dividi (es. `var /= 1` è come `var = var / 1`)
**= Istruzione moltiplica* (es. `var *= 1` è come `var = var * 1`)

Se usi i binding, puoi utilizzare in più ulteriori operatori di assegnamento C/C++ come *|= l'assegnamento Or. Vedi la documentazione sui binding per maggiori dettagli*.

Incremento e Decremento

Il C++ conosce due operatori in più. '++' e '- -' solo ora sono supportati per i binding. Ma puoi scrivere invece var = var + 1 e anche var = var – 1. O in futuro usare invece INC(var) (incremento, +1) o DEC(var) (decremento, -1).

Confronto

KBasic ha diversi operatori per confrontare espressioni: Questi operatori fanno un confronto tra due operandi (espressioni) e il risultato è 'True' o 'False':

- *= uguale*
  es. `x = 3`
- *<> diverso*
  es. `x <> y`
- *< minore*
  es. `x < 3`
- *> maggiore*
  es. `x > 4`
- *<= minore o uguale*
  es. `x <= 4`
- *>= maggiore o uguale*
  es. `x >= 3`

Se usi i binding puoi utilizzare in più ulteriori operatori di confronto C++ come *==-. Vedi la documentazione sui binding per maggiori dettagli*.

Operatori logici (operatori booleani)

Sono utili per fare operazioni su bit, combinare bit assieme. Il risultato è 'True' o 'False'.
A differenza di VB6, KBasic supporta veri operatori logici di decisione ('AndAlso', 'OrElse'), che qualche volta sono meglio.

- *AndAlso*
  Entrambi gli operandi (espressioni) devono essere 'True'
- *OrElse*
  Uno degli operandi (espressioni) deve essere 'True'

Ma puoi anche usare gli operatori sui bit invece di 'AndAlso' o 'OrElse', che funzionano in modo molto simile.

Operatori sui bit

- *And and*
  es. `4 And 6`
  Utile per eseguire congiunzioni logiche di due espressioni. Il risultato è 'True' se entrambe le espressioni sono 'True'.
- *Or or*
  es. `33 Or 8`
  Utile per eseguire disgiunzioni logiche di due espressioni. Il risultato è 'True' se una delle espressioni è 'True'.
- *Xor or esclusivo*
  es. `77 Xor 3`
  il risultato è 'False' se entrambe le espressioni sono 'True'. Oppure il risultato è 'True' se una delle espressioni è 'True'.
- *Not not*
  e.g. `Not 5`
  Utile se desideri negare espressioni. Per di più, 'Not' inverte i bit di un byte.
- *BitAnd, BitOr, BitXor, BitNot* sono lo stesso di 'And', 'Or', 'Xor' e 'Not' che sono supportati per ragioni di compatibilità.

Il risultato è -1 ('True'), quando tutti i bit sono impostati (binario 1111 1111) oppure 0 quando tutti i bit non sono impostati (binario 0000 0000).

Se usi i binding puoi usare in più ulteriori operatori di confronto C++ come << oppure >>. *vedi la documentazione sui binding per maggiori* dettagli.
Ma in futuro potrai scrivere SHL (numero di cifre) o SHR (numero di cifre).

Altri operatori

- *^ Potenza*

Utile se vuoi usare l'elevamento a potenza. Se ci sono molte potenze in un'espressione, queste sono valutate da sinistra a destra.

- *Like (in futuro)*

Utile per confrontare due stringhe (usando un pattern).
Se stringa e pattern corrispondono il risultato è 'True'.

- *New crea un nuovo oggetto di una classe*
- *( ) o [ ]*

accesso a indici di array o alla lista degli argomenti di procedure

- *. operatore punto*

necessario per chiamare metodi o elementi nei type o enumerazioni

- *& concatenamento stringa*

Utile per aggiungere una stringa ad un'altra

- *Eqv*

significa x Eqv y = Not (x Xor y)

- *Imp*

significa x Imp y = Not (x And Not y)

- *Is (in futuro)*

Utile per scoprire se due oggetti sono della stessa classe.

Se usi i binding, puoi utilizzare in più ulteriori operatori C++ come i seguenti. Vedi documentazione sui binding per maggiori dettagli.

- *++ incremento*
- *-- decremento*
- *[ ] accesso agli indici*
- *|= assegnamento or*
- *&= assegnamento and*
- *| or*
- *! Not*
- *== è uguale*
- *!= diverso*
- *^= assegnamento (elevamento a) potenza*
- *<< scorrimento a sinistra*
- *>> scorrimento a destra*

Concatenamento stringa

E' anche possibile sommare due stringhe (aggiungere una stringa ad un'altra). Il risultato è il tipo di dato 'String'. Per quello esistono due operatori: '+' e '&'. Ma '+' è differente se un operando è di tipo diverso da 'String'. Perciò, se vuoi essere sicuro, usa sempre '&' per il concatenamento di stringa.

- *'+'*
- *'&'*

Esempio: Name + „is a „ + color + „bird“

Se il primo operando è una stringa, quando usi '+', tutti gli altri '+' di quell'espressione sono trattati come (applicati su, ndt) stringhe.

Ordine degli operatori

KBasic supporta anche l'ordine degli operatori VB6, che significa in quale ordine sono eseguiti gli operatori di un'espressione. Normalmente un'espressione viene eseguita da sinistra a destra.

es.
x = 10 + 10 / 2 risulta in 15 e non 10, poiché / è calcolato prima di +.

Qui c'è la descrizione sull'ordine/priorità degli operatori dall'alto verso il basso, che vuol dire che '*' è eseguito prima di 'And' per esempio. Sono pure inclusi gli operatori di binding (tra parentesi):

- . ( ) [ ]
- Not, BitNot, !, ++, --, (unario +), (unario -)
- New
- ^
- * / \ Mod
- + - &
- Imp Eqv
- < > <= >= = Is Like == !=
- And BitAnd (& C/C++ And)
- Or Xor BitOr BitXor (| C/C++ Or) (^ C/C++ Xor)
- AndAlso
- OrElse

Puoi sempre usare le parentesi () per cambiare l'ordine.

es.
X = (10 + 10) / 2 risulta 10 e non 15, perché + è calcolato prima di / grazie alle parentesi.

Evitare collisioni di nomi

Una collisione di nomi ha luogo quando desideri usare un nome che è stato già definito in KBasic (o persino dallo stesso KBasic = parole chiavi).

Puoi evitare collisioni di nomi conoscendo il concetto di *scope* (ambito, visibilità o *campo d'azione*, *ndt)*. Ci sono diversi tipi di ambiti in KBasic: ambito di una procedura, ambito globale, ambito di classe, ambito di modulo, senza dimenticare i modificatori di accesso ('Public', 'Private', 'Protected'). Puoi avere collisioni di nomi nelle seguenti situazioni:

- Se un identificatore è accessibile da diversi ambiti
- Se un identificatore ha diversi significati nello stesso ambito

La maggior parte delle collisioni di nomi può essere evitata utilizzando nomi pienamente qualificati di identificatori. Esempio:

```
aModule.aSub(Module1.Var1)
```

Editare il codice sorgente

Non è molto diverso editare un testo in un elaboratore di testi o in KBasic Software Atelier. Hai un cursore che mostra la posizione corrente, dove puoi scrivere. Puoi trovare e sostituire il tuo codice sorgente come in un elaboratore di testi e così via...

Lavorare con gli oggetti

Poiché KBasic è un linguaggio di programmazione object-oriented, probabilmente vorresti sapere come usare queste caratteristiche. I seguenti problemi necessitano di essere discussi:

- Come posso creare una classe?
- Come posso creare un oggetto (istanza) di una classe?
- Come posso usare variabili di classe, metodi di classe, variabili d'istanza e metodi d'istanza?
- Come posso convertire un oggetto in un altro oggetto?

Crea nuovi oggetti

O crei un oggetto basata su una classe predefinita di KBasic, o lo crei da una tua classe definita.

Uso di New

Una variabile, che dovrebbe contenere l'oggetto ed è dichiarata, non crea l'oggetto in una volta. Questa variabile ha solo il tipo di dato di un oggetto e rappresenta solo un reference ad un oggetto, che deve essere ancora creato. La creazione degli oggetti è fatta usando la speciale keyword 'New', che genera un oggetto (dandogli il nome di classe che si vuole) e ritorna un reference al nuovo oggetto, che viene memorizzato in una variabile di tipo class. E' il modo di creare gli oggetti come in Java o C++.

Esempio:

```
s = New Control()
```

Non hai bisogno di scrivere le parentesi, ma spesso hai degli argomenti quando crei un oggetto, che devono essere scritti dentro le parentesi. 'New' è seguito dal nome della classe di cui tipo è il nuovo oggetto. La classe ha una speciale procedura chiamata costruttore, la quale è chiamata da 'New'. Questo costruttore esegue del lavoro iniziale per un oggetto e restituisce un reference al nuovo oggetto creato. A differenza delle procedure normali, non puoi chiamare direttamente un costruttore; invece, i costruttori sono chiamati da KBasic automaticamente quando crei un nuovo oggetto con 'New'.
Ancora un'altro esempio:

```
s = New Timer(begin, end)
```

Cosa succede quando usi 'New'?

Usando 'New' si crea una nuova istanza (o oggetto) di una classe. Viene eseguito il costruttore corrispondente a quella classe. Ci sono due cose importanti da considerare, quando desideri usare o dichiarare costruttori:

- Il nome del costruttore deve corrispondere al nome della classe
- Il valore restituito è sempre un'istanza di quella classe (implicitamente). Non è necessaria una dichiarazione del tipo di dato da restituire, né si usa la parola chiave 'Sub' o 'Function'. Un costruttore non deve esplicitamente restituire valori usando la keyword 'Return'.

Sintassi:

```
VariabileOggetto = New NomeClasse[(Argomenti)]
```

```
VariabileOggetto = New NomeClasse()

VariabileOggetto = New  NomeClasse
```

Alcuni costruttori

Qualche volta, vorresti usare un oggetto in diversi modi o eseguire l'azione iniziale in diverse maniere. Puoi fare così, usando diversi costruttori per la stessa classe, che significa che ogni classe può avere diversi costruttori con argomenti differenti

Esempio:

```
Class test1

  Constructor test1()
  End Constructor

  Constructor test1(i As Integer)
  End Constructor

  Constructor test1(i As Integer, m As String)
  End Constructor

End Class
```

'Null' (o 'Nothing')

Il valore di default per le variabili oggetto è 'Null'. 'Null' (o 'Nothing') è una parola riservata, che vuol dire che la variabile oggetto non ha ancora un reference. 'Null' può essere assegnato solo alle variabili oggetto.

Rilasciare oggetti automaticamente

Se non hai più bisogno di una variabile oggetto (o oggetto), puoi semplicemente dimenticarti di essa, il che significa che non hai bisogno di cancellare o liberare esplicitamente l'oggetto, poiché KBasic ha un ben noto meccanismo, che lo fa automaticamente (chiamato garbage collection). Per cui ti puoi concentrare nella programmazione dell'algoritmo.

Liberare oggetti manualmente

Puoi anche esplicitamente liberare o rilasciare gli oggetti (la loro memoria, ndt), usando la keyword 'Null'. Semplicemente assegna 'Null' alla variabile desiderata.

```
myVar = Null
```

Gli oggetti che non sono più referenziati, sono automaticamente cancellati da KBasic. Questo accade quando nessuna variabile oggetto fa riferimento a tale oggetto, poiché la variabile oggetto è stata cancellata automaticamente abbandonando l'attuale procedura o il valore è stato esplicitamente impostato a 'Null' da te.

Per esempio:

```
VarOggetto = Null
VarOggetto = Nothing
```

Questo vuol dire che non ti curi di rilasciare gli oggetti, puoi semplicemente dimenticartene. Questo è molto diverso dal C++ e da altri linguaggi di programmazione.

Creare una classe

Definire una classe è facile. Normalmente ogni classe è figlia della classe speciale 'Object' direttamente o indirettamente attraverso un'altra classe. Una classe può ereditare da un'altra classe (essere figlia di quella classe). La dichiarazione di una classe consiste del nome della classe e del nome della classe genitore, inoltre hai costruttori, qualche volta distruttori, qualche variabile (variabili di classe e variabili d'istanza) e procedure (metodi, metodi di classe, metodi d'istanza) e naturalmente delle proprietà e costanti.

```
Class oak Inherits tree
```

Le parti sono:
- Variabili/ Costanti / Proprietà / Tipi / Enumerazioni
- Costruttori
- Distruttori
- Funzioni
- Sub

```
End Class
```

Una classe può essere dichiarata solo dentro un modulo (non raccomandato) oppure dentro un file di classe. Dichiarare una classe dentro un modulo di un form non è possibile, sebbene in effetti contenga una classe, che eredita implicitamente dalla classe 'Form'.

Ed è possibile dichiarare molte classi in un file di classe.

Le classi non sono eseguite, ma i metodi lo sono

Una nuova classe creata contiene solo la parte di dichiarazione (nome di classe e nome del genitore), ma non metodi. Questi devono essere creati da te. KBasic non esegue alcuna classe, ma metodi dentro la classe.

Accesso ad oggetti

L'accesso ad oggetti è fatto usando l'operatore punto(.).

Esempio:

```
NomeClasse.variabile
```

Questa sintassi è abbastanza simile ad accedere a elementi di tipi di dati definiti dall'utente ('Type' ... 'End Type').

Accesso a variabili di classe e variabili d'istanza

L'accesso alle variabili è fatto usando l'operatore punto (.).
Esempio:

```
mioOggetto.variabile
```

Quando accedi a variabili di classe, usi il nome della classe, quando hai accesso a variabili d'istanza usi il nome della variabile oggetto:

```
NomeClasse.VariabileDiClasse
NomeOggetto.VariabileDistanza
```

Importante! Le variabili di classe esistono solo una volta, quindi non è importante quanti oggetti di quella classe esistono. La classe in se stessa esiste solo una volta, così pure le variabili di classe. Puoi usare le variabili di classe senza aver creato prima un oggetto di quella classe. La classe è disponibile in qualsiasi momento mentre è in esecuzione il tuo programma. E' come una variabile globale.

Comunque le variabili d'istanza sono parte di un oggetto di una classe, quindi esistono tante variabili d'istanza quanti sono gli oggetti di quella classe e solo per il tempo di vita di un oggetto. Hai capito? Se dichiari una variabile all'interno di una classe, puoi segnare una variabile con la parola chiave 'Static' se deve essere una variabile di classe. Se non lo fai, è di default una variabile d'istanza.

Esempio:

```
Static miaVariabileDiClasse As Integer
Private miaVariabileDistanza As String
```

Altro esempio:

```
miaClass.VariabileDiClasse = 23

Dim o As miaClasse
```

Accesso ad un metodo:

```
o.VariabileDistanza = 99
```

Metodi di classe e metodi d'istanza

Come le variabili anche i metodi possono essere correlati a una classe o oggetto. I metodi di classe sono sempre presenti, i metodi d'istanza sono presenti solo quando l'oggetto è presente. I metodi di classe sono come le funzioni globali. Non puoi usare 'Me' o 'Parent' all'interno di un metodo di classe, poiché non c'è un oggetto.

Quando definisci un metodo, puoi usare la parola chiave 'Static' per definire un metodo di classe. Se la tralasci, il metodo è di default un metodo d'istanza.

Esempio:

```
Static Sub mioMetodoDiClasse()
...
End Sub

Sub mioMetodoDistanza
...
End Sub
```

Chiamata di metodi

Ancora una volta usi l'operatore punto (.) per accedere ai metodi.

```
mioOggetto.mioMetodo()
```

Riferimenti a oggetti

Usando '=' puoi assegnare un reference di una variabile oggetto ad un'altra variabile oggetto. Entrambe le variabili in quel caso si riferiscono allo stesso oggetto.

Esempio:

```
Dim i As tree
Dim s As tree

i = New tree
s = i
```

Importante! 'Set', noto da VB6, non è permesso, e non è necessario per assegnare oggetti in KBasic, poiché è obsoleto.

Copia di oggetti

Poiché gli oggetti non sono passati per valore nelle procedure, un assegnamento di un oggetto non copia un oggetto. Invece devi fornire i metodi appropriati nelle cassi, che possono copiare un oggetto. In futuro, KBasic fornirà oggetti con un metodo *clone*, che sarà in grado di fare una copia di oggetti.

Confronto di oggetti

Un'altro problema del fatto che gli oggetti sono dati per reference è, che l'operatore '=' di confronto testa se due variabili oggetto fanno riferimento allo stesso oggetto, e non se entrambe le variabili oggetto hanno lo stesso contenuto. Invece devi fornire metodi propri nelle classi, che possono paragonare un oggetto. In futuro KBasic fornirà oggetti con un metodo di confronto, che sarà in grado di confrontare oggetti. Questo sarà il metodo *equals*.

Riassunto: oggetti, proprietà, metodi ed eventi

Un **oggetto** (**object**) è un elemento di un'applicazione, per esempio una tabella, un form o report. Per usare tale elemento puoi utilizzare le collection fornite da KBasic per tali oggetti.

Una **collection** contiene diversi oggetti dello stesso tipo come una collection di form contiene tutti i form della tua applicazione. Gli elementi di una collection sono accessibili per mezzo di un indice o nome.

Accesso tramite indice:

```
Sub myClose()
     Forms(1).Close
End Sub
```

Accesso tramite nome:

```
Sub closeForm()
     Forms("myForm2").Close
End Sub
```

Se la collection contiene metodi utilizzabili per tutti gli elementi di quella collection, puoi accedervi attraverso l'uso del nome della collection e del metodo voluto.

```
Sub closeAll()
     Forms.Close
End Sub
```

Un **metodo** è parte di un oggetto e contiene codice eseguibile, che può essere chiamato in qualsiasi momento

```
Sub addNewEntry(newEntry As String)
     Combo1.Add(newEntry)
End Sub
```

Una **proprietà** è parte di un oggetto, che memorizza valori come dimensione, colore o posizione di un oggetto. Puoi modificare il valore di una proprietà. Se vuoi cambiarlo devi scrivere il nome dell'oggetto, un punto (.), il nome della proprietà, il segno di uguale (=) e il nuovo valore della proprietà.

```
Sub changeName(newTitle)
     myForm.Caption =  newTitle
End Sub
```

Alcune proprietà sono solo leggibili. Vedi la voce appropriata dell'help per maggiori dettagli.

Ottenere il valore di una proprietà:

```
Sub getForm()
     formName = Screen.ActiveForm.Caption
```

```
    MsgBox formName
End Sub
```

Un **evento** è un'azione che è riconosciuta da un oggetto, es. se qualcuno ha cliccato con il mouse o ha premuto un tasto, allora una speciale procedura, una procedura evento, viene eseguita. Le procedure evento possono essere chiamate da eventi del sistema o manualmente dal tuo programma come normali procedure.

```
Sub CheckBox1_Click(m As Mouse)
    Print "Hi"
End Sub
```

Creazione di variabili oggetto

Puoi pensare ad un oggetto come ad una variabile, come dovrebbe essere in sé stesso un oggetto. Puoi accedere all'oggetto attraverso una variabile oggetto, chiamarne un metodo o una proprietà.

Come creare una variabile oggetto?

1. Dichiara una variabile oggetto
2. Assegna un oggetto alla variabile oggetto, usando la parola chiave 'New' per creare un nuovo oggetto o assegnare un oggetto esistente

Dichiarazione di una variabile oggetto

Per dichiarare una variabile oggetto, puoi usare l'istruzione 'Dim' o un'altra istruzione di dichiarazione (come 'Public', 'Private', 'Protected' o 'Static'). Una variabile, che fa riferimento ad un oggetto, deve essere di tipo 'Variant' o 'Object', o di un tipo specifico di oggetto (come 'Form', 'Font'...)

Qualche esempio di dichiarazione di oggetti:

```
' Object1 come tipo Variant
Dim Object1

' Object1 come generico oggetto
Dim Object1 As Object

' Object1 come oggetto di tipo Font
Dim Object1 As Font
```

Quando usi una variabile oggetto, senza dichiarazione, ha di default il tipo di dato 'Variant' (l'uso senza dichiarazione è possibile solo nel modo OldBasic!).

La dichiarazione di una variabile oggetto, di tipo object, è necessaria quando non sai che tipo di oggetto vorresti avere in una sub generica, per esempio.

Se conosci l'oggetto specifico, è utile dichiarare la variabile oggetto con lo stesso tipo di dato che ha l'oggetto.

Esempio con un normale oggetto e uno di tipo specifico:

```
Dim Objekt1 As Object       ' Oggetto

Dim Objekt1 As Sample       ' Oggetto di tipo Sample
```

La dichiarazione di tipi di oggetti specifici ha alcuni vantaggi come il controllo sui tipi, codice più leggibile o più veloce in esecuzione.

Assegnamento di oggetti ad una variabile oggetto

L'istruzione '=' ti permette di assegnare oggetti ad una variabile oggetto. Una variabile oggetto può avere il valore 'Null' (o 'Nothing') oppure un riferimento ad un oggetto.

Alcuni esempi:

```
Object1 = Object2     ' reference ad un oggetto
Object1 = Null        ' imposta a nessun reference per l'oggetto
```

Puoi combinare la dichiarazione di una variabile oggetto con quella di un oggetto, quando usi la keyword 'New' e l'istruzione '='.

Esempio:

```
Dim Object1 As Object = New Object    ' creazione e assegnamento
```

Quando assegni il valore 'Null' ad una variabile oggetto, che prima conteneva un reference ad un oggetto, eviti errori come modificare accidentalmente l'oggetto usando la variabile oggetto.

Esempio:

```
If Not Object1 Is Null Then
  ' la variabile oggetto fa riferimento ad un oggetto
  ...
End If
```

Uso dell'istanza corrente o oggetto / Me / Parent

La parola chiave 'Me' fa riferimento all'istanza corrente (o oggetto), in cui è eseguito attualmente il codice. 'Parent' significa l'attuale oggetto genitore. Normalmente, è la classe corrente (classe definita dall'utente o classe di un form). Tutti i metodi di quell'oggetto possono accedere all'oggetto attuale, che 'Me' referenzia. 'Me' è molto utile se vuoi dare l'oggetto ad un'altra procedura al di fuori della classe attuale, dove dovrebbe essere usato. In generale, 'Me' e 'Parent' sono come una variabile d'istanza di quell'oggetto, dichiarata implicitamente.

Esempio:

```
Sub changeObjectColor(Object1 As Object)
     Object1.BackColor = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub

changeObjectColor(Me) ' istruzione dentro l'oggetto
```

Secondo uso di 'Parent'

C'è un'altro significato di 'Parent'. Se una classe genitore contiene diversi costruttori, puoi determinare esplicitamente il costruttore genitore da chiamare usando la parola chiave 'Parent' con il costruttore desiderato della classe genitrice. Ma quest'istruzione deve essere la prima nel costruttore di una classe figlia, poiché il costruttore di una classe genitore deve essere chiamato prima che possa succedere qualsiasi azione nel figlio. Normalmente, viene automaticamente usato il corretto costruttore per chiamare il genitore, se non lo scrivi esplicitamente.

Esempio:

```
Class movies

  Protected sMovieName As String

  Sub printName
    print sMovieName
  End Sub

  Constructor movies(s As String)
    sMovieName = s
  End Constructor

End Class


Class movies2 Inherits movies

  Constructor movies2(ByRef s As String)
    Parent.movies(s + "2")
  End Constructor

End Class


Dim k As Integer = 9

Dim m As New movies2("final fantasy")

m.printName()
```

Casting di oggetti e tipi di dati semplici

Esistono alcune funzioni per il casting (conversione di tipo, *ndt*) di tipi di dati semplici come 'CInt'.

Mentre si usano gli oggetti, c'è anche la possibilità di eseguire il cast di oggetti in una classe genitore o figlia. Mentre si esegue il casting di un oggetto, esso può essere o ridotto (upcasting) oppure esteso (downcasting). La gerarchia dell'ereditarietà mostra quale oggetto è un oggetto figlio di un altro oggetto. In cima c'è il genitore di tutti gli oggetti, la classe 'Object', in fondo seguono le classi figlie.

Upcasting e downcasting implicitamente

Downcasting (conversione di un riferimento di un oggetto di una classe base, ad un riferimento di una classe derivata, *ndt*), l'oggetto 'o' contiene più metodi, variabili e così via come oggetto 'm' e viene perciò esteso:

```
Dim m As Control
Dim o As CommandButton ' control è la classe genitore
o = m
```

Upcasting (conversione di un riferimento di un oggetto di una classe derivata ad un riferimento alla classe base, *ndt*), l'oggetto 'h' contiene meno metodi, variabili e così via come 'm' e viene perciò ridotto:

```
Dim h As Control
Dim k As CommandButton ' control è la classe genitore
h = k
```

E' anche possibile un esplicito upcasting e downcasting:
DA FARE #1

Confrontare oggetti

Se usi '=' con variabili oggetto in un'espressione, l'espressione diventa 'True', se entrambe le variabili puntano allo stesso oggetto. Per confrontare entrambe le variabili oggetto o contesti, devi fornire metodi adeguati all'interno delle classi, che usi per paragonare questi oggetti.

Chiedere di oggetti e classi

Determinare i tipi di oggetti in esecuzione:

```
If TypeOf myObject Is myClass Then
```

Restituisce 'True', se il nome della classe è quello cercato, o 'False' se è diverso

Sintassi:

```
TypeOf objectVariable Is ClassName
```

Subclassing ed ereditarietà

C'è un'altra possibilità di estendere le tue classi o persino quelle KBasic. Questo si chiama ereditarietà, che vuol dire che tu hai una classe genitore e la usi come template (modello, *ndt*) per una nuova classe, in cui aggiungi nuovi metodi, variabili e così via. Tutti gli elementi come metodi e variabili della classe genitore sono automaticamente dentro la nuova classe (nascosti) e sono accessibili e utilizzabili (a seconda dalla dichiarazione nella classe genitore). Se vuoi cambiarne il comportamento puoi anche sottoporre a override un metodo della classe genitrice (ha lo stesso nome e argomenti), cosicché l'algoritmo della classe genitore usa il nuovo metodo nella nuova classe invece del metodo originale nella classe genitrice.

L'ereditarietà è molto potente ed è una caratteristica importante dei linguaggi di programmazione orientati agli oggetti.

Usando la parola chiave 'Inherits' dici a KBasic quale classe genitrice usi per la nuova classe:

```
Class movies

  Protected sMovieName As String

  Sub printName
    print sMovieName
  End Sub

  Constructor movies(s As String)
    sMovieName = s
  End Constructor

End Class


Class movies2 Inherits movies
...
End Class
```

Poiché ogni classe ha una classe genitore, tutte le classi formano assieme una gerarchia di classi (o gerarchia di ereditarietà). E' come un albero, con una radice (classe genitore 'Object') e molti ramoscelli (molte classi figlie).

Catena di costruttori

Quando dichiari una classe, KBasic garantisce che possa essere usata per creare un oggetto, poiché inserisce automaticamente un costruttore nascosto, di default, che viene chiamato quando usi questa classe senza argomenti. Se crei un costruttore, KBasic bada alla sua prima istruzione, e inserisce meccanicamente una chiamata al costruttore per la classe genitore, se non c'è nessuna chiamata esplicita di un costruttore genitore. Il costruttore di default può essere sempre scavalcato (sottoposto a override).

Questo significa che la chiamata di costruttori è come una catena. Tutti i costruttori sono connessi assieme. Ogni volta che costruisci un oggetto, esso segue tutti i costruttori di tutte le classi genitrici fino alla classe genitore 'Object'. Così il primo costruttore chiamato è il costruttore di 'Object', poi seguono i costruttori figli delle classi discendenti.

Il costruttore di default

Se una classe non chiama un costruttore genitore esplicitamente, KBasic lo nasconde implicitamente. Se una classe non ha un costruttore, KBasic inserisce implicitamente un costruttore standard, che chiama il costruttore padre automaticamente.

Variabili nascoste

Una variabile esistente in una classe genitore, può essere sottoposta a overload (lett. s*ovraccaricata*, ndt) da una variabile nella classe figlia attraverso l'ereditarietà, sebbene sia possibile accedere ad entrambe le variabili con lo stesso nome. Si chiama nascosta, poiché non è più automaticamente visibile. Se desideri accedere alla variabile genitore, devi scrivere 'Parent', punto (.) e il nome della variabile:

```
Parent.myVariable  ' accedo alla variabile genitore
myVariable         ' accedo alla variabile dell'oggetto corrente
```

Metodi nascosti

Un metodo esistente, come una variabile, può essere sottoposto a overload in una classe genitrice da un metodo della classe figlia, sebbene sia possibile accedere ad entrambi i metodi con lo stesso nome. Si chiama nascosto, poiché non è più automaticamente visibile. I metodi nascosti sono così chiamati metodi sottoposti a override. Un override ha luogo se la classe figlia ha un metodo con la stessa firma (nome, tipo restituito, stessi argomenti) di uno della classe genitore. Il metodo padre è ora sottoposto a override. Se desideri accedere al metodo padre, devi scrivere 'Parent', punto (.) e il nome del metodo:

```
Parent.myMethod()      ' accedo al metodo genitore
myMethod()             ' accedo al metodo dell'oggetto corrente
```

Override di metodi

L'override ha luogo se la classe figlia ha un metodo con la stessa firma (nome, tipo di ritorno, stessi argomenti) della classe genitore. Il metodo genitore è ora sottoposto a override. Così se un metodo viene chiamato dentro la classe figlia, è usato il nuovo metodo definito, perfino dai metodi del genitore (inclusi dalla classe figlia).

L'Override dei metodi è una tecnica molto importante nella programmazione orientata agli oggetti. Ma stai attento a non confondere l'override con l'overload. Sottoporre a overload significa che si hanno molti metodi con lo stesso nome in una classe, ma con diversa firma (diverso tipo di ritorno o argomenti).

KBasic tratta in maniera quasi uguale variabili e metodi in una classe, non è diverso quando si tratta di scavalcare metodi e si nascondono variabili. Puoi facilmente accedere a variabili nascoste, facendo un cast al tipo genitore o usando la keyword 'Parent'. La differenza è interamente nel significato, poiché le variabili sono usate diversamente in una classe figlia e genitore (effettivamente usano la loro variabile), mentre i metodi sono usati da entrambe (sia la classe che quella genitrice usano lo stesso metodo!).

Ricerca dinamica di metodi

Mentre compila KBasic non può sapere quale metodo di quale classe dovrebbe essere chiamato, poiché potrebbe essere codificato in maniera tale che deve essere cercato dinamicamente a runtime. Questo è chiamato *late binding*, il che significa che non è così veloce come l'*early binding*, che ha luogo quando tu chiami funzioni o procedure di un modulo.

Chiamare un metodo sottoposto a override

Puoi accedere facilmente ad un metodo sottoposto a override, tramite casting della sua variabile oggetto al tipo del genitore o usando la parola chiave 'Parent'.

Esempio:

```
Class A
  Dim i As Integer

  Function f()
    Return i
  End Functon

End Class

Class B Inherits A
  Dim i As Integer     ' questa variabile nasconde i in A

  Functon f()         ' questo metodo scavalca f() in A
    i = Parent.i + 1        ' accesso A.i
    Return Parent.f() + i   ' accesso A.F()
  End Functon
End Class
```

Se fai l'override di un metodo hai due metodi. Il vecchio nella classe genitore e il nuovo in quella figlia. Si può accedere al vecchio esplicitamente usando 'Parent' e al nuovo usando 'Me' o la chiamata standard di un metodo.

Un ulteriore esempio:

```
' esempio di una classe
Class being

  Constructor being()
    Print "being.Constructor!!!!"
  End Constructor

  Sub cry()
    Print "being.cry"
  End Sub

End Class


Class body Inherits being

  Constructor body()
    Print "body.Constructor!!!!"
  End Constructor

  Sub cry()
    Print "body.cry"
  End Sub

End Class


Class face Inherits being

  Constructor face()
    Print "face.Constructor!!!!"
  End Constructor

  Sub cry()
    Print "face.cry"
  End Sub

End Class

Dim l[10] As being

l[3] = New being
l[4] = New face
l[5] = New body

' polimorfismo
l[3].cry()
l[4].cry()
l[5].cry()
```

Nascondere dati

Nascondere dati (variabili e costanti...) è una tecnica (chiamata anche *information hiding, ndt)* molto importante nella programmazione object-oriented. Significa che non si può accedere a tutti i dati in una classe, ma attraverso metodi della stessa classe, in cui sono presenti i dati, cosicché sono visibili solo dentro la classe. Questo aiuta a ridurre gli errori quando si usano dati interni di una classe da altre classi o moduli, o in generale facendo un cattivo stile di programmazione. Hai metodi pubblici e variabili che sono accessibili da tutti gli altri, e hai metodi privati e variabili che dovrebbero essere usati solo dalla classe stessa e da quei metodi pubblici.

Ulteriori motivi per nascondere dati sono:

- variabili interne che sono visibili all'esterno, mischiare le API per altri sviluppatori ecc. Nasconderli porta a classi piccole ed eleganti con poche variabili e metodi visibili agli altri.
- Se una variabile è visibile, devi documentarla. Riducendo le variabili nascondendole aiuta a diminuire il tempo di lavoro sulla documentazione.

Modificatori d'accesso

Per nascondere variabili e metodi, devi dichiararli come 'Private':

```
Class plane

  Private wings As Integer       ' visibile solo dentro questa classe

  Private Funcion countWings()   ' visibile solo dentro questa classe
    ...
  End Function

End Class
```

Una variabile **private** (privata, *ndt*) è visibile nella classe in cui è stata dichiarata, il che vuol dire che solo i metodi di questa classe possono usare questa variabile. Metodi privati e variabili private non sono visibili alle classi figlie della classe in cui sono dichiarati (Di fatto sono presenti nella memoria di quell'oggetto, ma non sono visibili), metodi e variabili non *private* sono sempre visibili alle sottoclassi.

Oltre a 'Private', esistono 'Public' e 'Protected' come modificatori di accesso. **Protected** è utile se la variabile o il metodo devono essere visibili solo alle classi figlie e alla classe in cui sono dichiarati. **Public** vuol dire che ogni parte del tuo programma (qualsiasi altra classe o modulo) può accedere e usare la parte dichiarata pubblica della tua classe (variabile o metodo). Il modificatore di default è 'Public', che significa che se non scrivi 'Public' davanti alla variabile o al metodo, esso/a è automaticamente dichiarata come public.

Alcune dritte per usare i modificatori d'accesso
- Usa 'Private' per metodi e variabili, che dovrebbero essere usati solo nella classe, ma non all'esterno.
- Usa 'Public' per metodi e variabili, che dovrebbero essere accessibili dovunque.

Classi e metodi abstract

In KBasic è possibile definire metodi, che non hanno del codice all'interno, ma che possono essere usati come template per le sottoclassi. Questa tecnica è chiamata dichiarazione abstract (astratta, *ndt*). I metodi abstract sono utili quando non hai codice in un metodo di una classe genitore, ma che deve essere presente nella classe genitrice per motivi di design. Le classi figlie sono ora obbligate ad implementare questo metodo con del codice in esso. E' anche possibile contrassegnare un'intera classe come abstract, il che significa che non puoi creare un oggetto di quella classe, perché questa classe contiene dichiarazioni, ma nessun codice. Il codice dovrebbe essere incluso nelle sottoclassi, che poi può essere usato dopo la parola chiave 'New'.

Normalmente non usi metodi o classi abstract, ma potrebbe essere vantaggioso. Alcune regole quando ci si occupa di metodi astratti o classi astratte:
- Ogni classe, contenente un metodo astratto, è automaticamente dichiarata come abstract. Una classe abstract deve contenere almeno un metodo astratto.
- Una classe astratta non può essere usata per creare un oggetto.
- Una classe figlia di una classe abstract può essere usata per creare un oggetto, se tutti i metodi abstract sono stati sottoposti a override con implementazione di codice. Adesso non è abstract.
- Se una classe figlia di una classe abstract non scavalca tutti i metodi diventa una classe abstract.

Descrizione delle classi predefinite di KBasic

KBasic ha classi e componenti come li ha Visual Basic 6.
DA FARE #2

Array

Se hai lavorato con altri linguaggi di programmazione, probabilmente conoscerai il concetto di array. Gli array sono variabili che assomigliano ad una catena. Tutti gli elementi della catena hanno lo stesso tipo di dato a cui si può accedere tramite un indice. Queste variabili hanno un solo nome ma diverso indice, cosicché si possono distinguere; ma sono poste in un singolo blocco in memoria. Puoi usare l'array come un unico blocco, o solo un elemento dell'array. Gli array sono dichiarati allo stesso modo delle variabili (usa 'Dim', 'Static', 'Private', 'Public' o 'Protected').

Usare array porta a migliore codice (più chiaro, più flessibile), perché puoi usare dei cicli per accedere a migliaia di variabili (gli array). Gli array hanno dei confini, un limite superiore e uno inferiore. Gli elementi di un array sono tra questi confini. Se un array ha tipo di dato 'Variant', ogni suo elemento può contenere un diverso tipo di dato.

Differenti tipi di array

Ci sono due tipi di array, che sono differenti solo in un aspetto: uno ha una dimensione fissa in compilazione e l'altro può modificare la sua dimensione dinamicamente a run-time.

- *Array dinamico*
  Cambia le dimensioni durante l'esecuzione
- *Array statico*
  Dimensioni fisse

E' possibile che non conosci la lunghezza di un array mentre programmi o vorresti modificare la sua dimensione a runtime. Perciò è utile usare gli array dinamici. Un altro vantaggio è che puoi liberare la memoria, quando usi array dinamici, allorché non ne hai più bisogno. Gli array statici rimangono sempre in memoria.

Dichiarazione

Dichiarazione di un array usando 'Dim' e dichiarazione della lunghezza in ( ) o [ ]:

Sintassi:

```
Dim nomeVariabile(Indice) As Tipo

Dim nomeVariabile[Indice] As Tipo

Dim nomeVariabile[Indice, Indice, ...] As Tipo

Dim nomeVariabile[Indice  To Indice] As Tipo

Dim nomeVariabile[Indice  To Indice, Indice To Indice, ...] As Tipo
```

Esempio:

```
Dim i(100) As Integer
```

Crea un array con 100 variabili 'Integer'.

Accesso agli elementi di un array

Puoi accedere ad un elemento di un array quando scrivi un'espressione intera tra parentesi () o [] dopo il nome dell'array:

```
i(3) = 10
```

Accesso con posizione. Nell'esempio la posizione è 1:

```
i(1) = 0
```

L'indice viene sempre controllato così se è troppo piccolo o troppo grande, incorri in un errore. Questa è un'ulteriore caratteristica di KBasic, di evitare crash del programma.

L'espressione che sta nell'indice, può essere calcolata a run-time. In altre parole è consentita ogni espressione numerica:

```
N = 100

Dim i(n) As Integer
```

Ottenere il limite superiore e inferiore

UBound si usa per il limite superiore:

```
UBound (arrayVar [,(Dimensione])
```

LBound si usa per il limite inferiore:

```
LBound (arrayVar [,(Dimensione])
```

Default: Un array con 10 elementi dichiarati è conteggiato da 0 a 10 (incluso 10), il che vuol dire che quel 0 e 10 sono indici utilizzabili e ci sono 11 elementi dichiarati.

Di fatto un array contiene sempre un elemento in più di quello dichiarato, quando non specifichi un limite inferiore e superiore.

Dichiarare esplicitamente il limite inferiore

```
Dim i(50 To 100) As Integer
```

Questo crea un array con un limite inferiore di 50.

Esempio:

```
Dim curCosts (364) As Currency

Sub createArray ()
     Dim curCosts (364) As Currency
     Dim intI As Integer

     For intI = 0 To 364
          curCosts (intI) = 20
     Next
End Sub
```

Modificare il limite inferiore

Puoi usare l'istruzione 'Option Base' in cima ad una classe o modulo di file, per modificare l'indice di default di un array da 0 a 1. Nel seguente esempio il primo indice disponibile sarà 1:

```
Option Base 1

Dim curCosts(365) As Currency
```

Puoi cambiare il limite superiore di un array usando la parola chiave 'To'. Esempio:

```
Dim curCosts(1 To 365) As Currency
Dim sWeekday(7 To 13) As String
```

Uso di array multidimensionali

KBasic supporta anche gli array multidimensionali. La dimensione totale è limitata a 32. Come creare un array multidimensionale è mostrato nell'esempio che segue:

```
Dim i(100, 50, 400)
```

L'istruzione seguente dichiara un array a due dimensioni:

```
Dim sngMulti(1 To 5, 1 To 10) As Single
```

Un array bidimensionale è come una matrice. Il primo argomento può essere pensato come le righe e il secondo come le colonne. Se usi array con due o più dimensioni, puoi facilmente usare il ciclo 'For-Next' per iterare e accedere ai suoi elementi.

```
Sub changeArray ()
     Dim intI As Integer, intJ As Integer
     Dim sngMulti(1 To 5, 1 To 10) As Single

     ' imposta i valori dell'array
     For intI = 1 To 5
          For intJ = 1 To 10
               sngMulti(intI, intJ) = intI * intJ
               Print sngMulti(intI, intJ)
          Next intJ
     Next intI
End Sub
```

Salvare valori 'Variant' in array

Ci sono due modi di creare array con elementi di tipo 'Variant'. Primo crei un array con tipo di dato 'Variant'. Esempio:

```
Dim varData(3) As Variant
varData(0) = "Christiane Roth"
varData(1) = "60529 Frankfurt"
varData(2) = 26
```

Ma puoi anche usare la funzione 'Array', che ritorna un array di tipo 'Variant'. Esempio:

```
Dim varDaten As Variant
varData = Array(" Christiane Roth", "60529 Frankfurt", 26)
```

Accedere all'array è lo stesso per entrambi i metodi. Semplicemente usa l'indice. Esempio:

```
Print "Il dato " & varDaten(0) & " è stato memorizzato."
```

Modificare le dimensioni di un array dinamico

Dichiara l'array usando l'istruzione 'Dim' e senza l'indice, ma con le parentesi () o [], es.

```
Dim a() As Integer
```

Riserva l'esatto conteggio del numero degli elementi (dell'array) con 'Redim'.

L'istruzione 'Redim' può essere usata solo dentro procedure. Ha la stessa sintassi di 'Dim'. Ogni istruzione 'Redim' può modificare il numero degli elementi, e anche il limite inferiore e superiore di quell'array. Con 'Redim'.non si possono modificare le dimensioni.

Ogni volta che usi 'Redim' tutti i valori dell'array sono cancellati, se non usi la keyword 'Preserve'. KBasic imposta i valori a 'Empty' (quando il tipo è 'Variant'), o al valore 0 (tipi numerici), o "" (se 'String'). I tipi Object saranno 'Null'.

Esempio:

```
Function calc() As Integer
  Dim Matrix1 () As Integer

  Redim Matrix1 (22, 33)
  ...
```

Puoi anche usare variabili invece di literal (infatti è permessa ogni espressione):

```
Redim Matrix1 (a, b)
```

Sintassi:

```
Redim Nomevariabile(Indice)

Redim Nomevariabile[Indice]

Redim Nomevariabile[Indice, Indice, ...]

Redim Nomevariabile[Indice  To Indice]

Redim Nomevariabile[Indice  To Index, Indice To Indice, ...]
```

Array-reset / array-delete

Il comando 'Erase' cancella gli array e libera la memoria da essi occupata.

```
Erase array1[, array2]
```

Quando usi array statici tutti i valori dell'array vengono cancellati. KBasic imposta i valori a 'Empty' (quando il tipo è 'Variant'), o a 0 (tipi numerici), oppure a "" (se 'String'). I tipi Object saranno 'Null'. La memoria non è liberata. Se è un array dinamico, verrà liberata la memoria.

Controllo di flusso

Le istruzioni che prendono decisioni ed eseguono cicli, sono note come strutture di controllo.

Decisioni

Significa l'uso d'istruzioni condizionali per decidere cosa eseguire nel tuo programma. Le istruzioni condizionali testano se una data espressione è 'True' o 'False', dopo di che sono eseguite delle istruzioni che dipendono dal test. Normalmente una condizione usa un espressione, in cui viene usato un operatore di confronto per comparare due valori o variabili. KBasic supporta tutte le istruzioni condizionali di VB6 e molto di più.

Decisione singola - If

E' usata per eseguire un insieme di istruzioni, se è impostata una condizione (istruzione 'If'). Se la condizione è 'True' allora vengono eseguite le istruzioni dopo 'Then' (allora, *ndt*) e le istruzioni dopo 'Else' (altrimenti, *ndt*) vengono saltate. Se la condizione è 'False' allora vengono invece eseguite le istruzioni dopo 'Else. Se la voce (token) dopo 'Then' è un numero di riga allora è eseguita un'istruzione goto. Se la condizione è vera e non c'è un'istruzione 'Then' nella stessa riga, vengono eseguite le istruzioni fino a che non si trova una riga con 'End If'. Se sei in 'KBasic Mode' l'espressione deve essere booleana, altrimenti è permessa ogni altra istruzione numerica.

Sintassi:

```
If Espressione Then Istruzione

If Espressione Then Istruzione : Else Istruzione

If Espressione Then LineNo

If Espressione Then LabelName:

If Espressione Then
  [istruzioni]
End If

If Espressione Then
  [Istruzioni]
Else
  [Istruzioni]
End If

If Espressione Then
  [Istruzioni]
ElseIf  Espressione
  [Istruzioni]
Else
  [Istruzioni]
End If

If Espressione Then
```

```
  [Istruzioni]
ElseIf Espressione
  [Istruzioni]
ElseIf Espressione
  [Istruzioni]
Else
  [Istruzioni]
End If

If Espressione Then
  [Istruzioni]
ElseIf  Espressione
  [Istruzioni]
End If
```

'Else' introduce una condizione di default in un'istruzione If multilinea.

'ElseIf' introduce una condizione secondaria in un'istruzione If multilinea.

'End If' termina un'istruzione If multilinea.

'If' valuta un' 'espressione' ed esegue l'istruzione 'Then' se è 'True' o (opzionalmente) l'istruzione 'Else' se è 'False'. KBasic consente istruzioni 'If' multilinea con i casi 'Else' e 'ElseIf', che finiscono con 'End If'. Questo funziona poiché zero è interpretato come 'False' e qualsiasi (valore) non-zero è interpretato come 'True'

Esempio:

```
Dim i As Integer
Dim n As Integer

If i 1 Then
  n = 11111
ElseIf i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

IIf – if corto

Restituisce uno o due valori che dipendono da un'espressione. Esempio:

```
testing = IIf(Test1 > 1000, "big", "small")
```

Sintassi:

```
IIf(Expression, ThenReturnExpression, ElseReturnExpression)
```

Select Case

L'istruzione 'Select Case' è molto più complicata di un'istruzione 'If':

In molte situazioni, potresti voler confrontare lo stesso valore (o espressione) con molti valori differenti, ed eseguire un pezzo di codice a seconda di quanto equivale. Questo è esattamente ciò a cui serve l'istruzione 'Select Case'.

Esso introduce l'istruzione multilinea di selezione condizionale. L'espressione data come argomento a 'Select Case' sarà valutata dalle istruzioni 'Case' che seguono. L'istruzione 'Select Case' conclude con 'End Select'. Come implementate attualmente, le istruzioni 'Case' possono essere seguite da valori stringa, in questo caso possono essere eseguite espressioni complesse. Quindi l'espressione di test può essere 'True', 'False', o una numerica o un'espressione stringa.

Esegue l'istruzione dopo il 'Case' a cui corrisponde l'espressione, poi salta all'istruzione 'End Select'. Se non c'è alcuna corrispondenza, ed è presente 'Case Else', allora vengono eseguite le istruzioni di default successive a 'Case Else'.

Sintassi:

```
Select Case Espressione
Case Espressione
  [Istruzioni]
Case Espressione
  [Istruzioni]
End Select

Select Case Espressione
Case Espressione
  [Istruzioni]
Case Espressione To Espressione
  [Istruzioni]
Case Is Espressione
  [Istruzioni]
Case Else
  [Istruzioni]
End Select
```

Esempio:

```
Dim i As Double
Dim n As Integer

i = 4

Select Case i
Case 0
  n = 0
Case 1, 2
  n = 1122
Case 4 TO 10
```

```
  n = 441000
Case Is = 9
  n = 9999
Case Else
  n = 999999
End Select
```

Switch – select case corto

Restituisce un valore che dipende da un'espressione. Esempio:

```
Dim s As String
Dim i As Integer
i = 1
s = Switch(i = 1, "Bible", i = 2, "Casanova")
Print s
```

Sintassi:

```
Switch(Espressione, ReturnExpression[,Expression, ReturnExpression,
... ])
```

Choose – select case corto

Restituisce un valore da una lista di valori tramite un indice.

```
Dim s As String
s = Choose(1, "un", "deux", "troi")
Print s
```

Sintassi:

```
Choose(Expression, ReturnExpression[, ReturnExpression, ... ])
```

Salto incondizionato

La maggior parte di questo capitolo è stato dedicato ai branch (ramo, diramazione del codice, *ndt)* condizionali. Se ricordi, comunque, i programmatori possono anche usare i branch incondizionali. Questo tipo di branching può essere compiuto usando l'istruzione 'GoTo', che forza l'esecuzione del programma a saltare ad uno specifico numero di linea o etichetta. Poiché i numeri di linea nei programmi KBasic sono ora obsoleti (ad ogni modo sono supportati), non devi preoccuparti di come usarli. Puoi, comunque, voler usare le label (etichette, ndt).

'GoTo' esegue un salto incondizionato. 'GoTo' viene sempre eseguito senza condizioni

sintassi:

```
GoTo {LineNo | Label:}

GoTo myExit:
GoTo nextStep:
```

Istruzione With

Un'istruzione molto utile è 'With'. Con l'istruzione 'With' sei in grado di raggruppare assegnamenti o istruzioni, che referenziano lo stesso oggetto. Così il tuo codice diventa più leggibile e modificabile in maniera più elegante. Riduce anche codice ridondante.

```
Sub formSection ()
      With myClass.
            .Value = 30
            .Font.Bold = True
      End With
End Sub
```

Puoi scrivere un'istruzione 'With' dentro un'altra istruzione 'With':

```
Sub setValue ()
      With j(3)
        .e.bkname = "Frankfurter Zoo"
        With .e
          .isbn ( 99 ) = 333
        End With
      End With
End Sub
```

Istruzioni di loop

Le istruzioni che controllano decisioni e cicli in KBasic, sono chiamate strutture di controllo come già sai. Normalmente, ogni comando viene eseguito solo una volta, ma in molti casi è utile e necessario eseguire un comando diverse volte fino a che è stato raggiunto uno stato (condizione) definito. I loop ripetono comandi che dipendono da una condizione. Alcuni loop reiterano dei comandi mentre una condizione è 'True', altri ripetono dei comandi mentre una condizione è 'False'. Per di più, ci sono altri loop che reiterano un numero fisso di volte, altri ripetono per tutti gli elementi di una collection.

For Next

Questo ciclo è utile quando sai esattamente quante volte le istruzione dovrebbero essere ripetute. Quindi esso definisce un ciclo che gira per un numero di volte specificato.

Sintassi:

```
For contatore = start To stop [Step espressionePasso]
  [Istruzioni]
Next [contatore]
```

'contatore' è una variabile numerica, che è utilizzata per memorizzare l'attuale valore della variabile contatore mentre il ciclo è in esecuzione. 'start' (inizio, ndt) è un'espressione numerica che determina il valore iniziale dei conteggi del ciclo. 'stop', è un'espressione numerica che immagazzina il valore finale in cui si fermerà il loop. Se desideri contare per passi differenti rispetto al valore 1 di default, usa 'espressionePasso' per determinare il valore specifico con cui incrementare il contatore.

Se usi 'Exit For', questo ti farà uscire subito dal ciclo 'For Next' per continuare con la riga che segue l'istruzione 'Next'. Di solito questa è in connessione con una clausola 'If' (If qualcosa Then Exit For) per uscire dal loop prima della fine specificata.

L'istruzione 'Next' è l'estremità inferiore del ciclo e incrementa 'contatore' del valore dato da 'espressionePasso' o di 1 se non è definito nessun incremento 'espressionePasso' . Il loop sarà ripetuto finché il valore di 'contatore' è più piccolo/più grande di o uguale al valore di 'stop'. L'indicazione di 'contatore' può essere omessa per l'istruzione 'Next', comunque questo è deprecato poiché può portare a confusione quando si annidano diversi cicli ' 'For Next'.

Note:
- la velocità dei cicli 'For Next' dipende dal tipo di variabili che usi.
I cicli 'For Next' saranno i più veloci quando 'contatore' è una variabile intera e 'start', 'stop' e 'espressionePasso' sono espressioni intere.

- Puoi contare all'indietro usando 'espressionePasso' con un valore d'incremento negativo.

- Abbi particolare cura quando annidi loop 'For Next'; ricorda di terminare l'ultimo ciclo iniziato (vedi esempio) o altrimenti avrà luogo un ciclo infinito.

```
' Esempio 1:
Dim ctr As Integer

For ctr = 1 To 5
  Print "Z";
Next ctr

' Output:
' ZZZZZ

' Esempio 2:
' Qui usiamo STEP
' La stringa risultante è la stessa
' dell'esempio 1

Dim ctr As Integer

As Integer ctr = 1 To 50 Step 10
  Print "Z";
Next ctr

' Output:
' ZZZZZ
```

```
' Esempio 3:
' Questi sono loop annidati.
' Il ciclo "y" è quello interno
' e perciò terminerà prima:

Dim x, y As Integer

For x = 1 To 5
  Print "Riga" + Str$(x)
  For y = 1 To 5
    Print "Z";
  Next y
  Print "-"
Next x

' Output:
' Riga 1
' ZZZZZ-
' Riga 2
' ZZZZZ-
' Riga 3
' ZZZZZ-
' Riga 4
' ZZZZZ-
' Riga 5
' ZZZZZ-
```

Ottimizza cicli For...Next

Dovresti usare variabili contatore di tipo 'Integer' piuttosto che 'Variant', poiché 'Integer' è più veloce.

Esempio:

```
Dim rabbit As Integer      ' contatore di tipo Integer

For rabbit = 0 To 3276
Next rabbit


Dim hedgehog As Variant    ' contatore di tipo Variant.

For hedgehog = 0 To 3276
Next hedgehog
```

Nel primo esempio, vedi che gira veloce. E' normalmente un fatto che: più piccolo il tipo di dato, più veloce può essere usato. Variabili di tipo 'Variant' sono sempre usate più lentamente dei tipi di dati utilizzati direttamente.

For Each

DA FARE #3

Altri tipi di cicli

Puoi usare i seguenti cicli quando non sei sicuro di quante volte dovrebbe essere eseguito un comando. Pertanto esistono diverse keyword: 'Do', 'While', 'Loop', 'Until' o ('Wend') .

Ci sono due modi diversi di usare la parola chiave 'While' per verificare una condizione dentro un'istruzione 'Do...Loop'. Puoi verificare la condizione prima che siano eseguiti i comandi dentro il loop, o puoi testare la condizione dopo che i comandi del loop sono stati eseguiti almeno una volta.
Se la condizione è 'True' (nella procedura 'SubBefore' che segue) saranno eseguiti i comandi all'interno del loop. Se imposti 'myNumber' a 9 invece di 20, non sarà eseguito nessun comando all'interno del ciclo. Dentro la procedura 'SubAfter' tutti i comandi sono eseguiti almeno una sola volta, poiché la condizione non è vera.

```
Sub SubBefore()
  Counter = 0
  myNumber = 20
  Do While myNumber > 10
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop
  MsgBox "Il ciclo è stato eseguito " & Counter & " time(s)."
End Sub

Sub SubAfter()
  Counter = 0
  myNumber = 9
  Do
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop While myNumber > 10
  MsgBox "Il ciclo è stato eseguito " & Counter & " time(s)."
End Sub
```

Ti è permesso di usare la parola chiave 'Until' in due circostanze per verificare una condizione di un'istruzione 'Do...Loop'. Puoi testare la condizione prima del loop (vedi procedura 'SubBefore' nell'esempio), o puoi testare la condizione dopo che si è entrati nel ciclo (vedi procedura 'SubAfter' nell'esempio che segue). Il loop sarà ripetuto finché la condizione è 'False'.

```
Sub SubBefore()
  Counter = 0
  myNumber = 20
  Do Until myNumber = 10
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop
  MsgBox "Il ciclo è stato eseguito  " & Counter & " time(s)."
End Sub

Sub SubAfter()
  Counter = 0
  myNumber = 1
  Do
```

```
    myNumber = myNumber + 1
    Counter = Counter + 1
  Loop Until myNumber = 10
  "Il ciclo è stato eseguito  " & Counter & " time(s)."
End Sub
```

Ciclo Do While...

Un gruppo di istruzioni che consentono di definire un loop che sarà iterato finché una certa condizione rimane 'True'.

- 'Do'
Avvia il ciclo e deve essere la prima istruzione

- 'Loop'
Termina il ciclo e deve essere l'ultima istruzione

- 'While'
Lascia ripetere il loop mentre la condizione è 'True'

- condizione
Un'espressione numerica o stringa con risultato 'True' o 'False'

- 'Exit Do'
Esce dal ciclo in questa precisa riga e fa continuare il programma sotto l'istruzione Loop'

- 'Iterate Do'
Salta direttamente alla condizione 'Loop'

Sintassi:

```
Do While Espressione
  [Istruzioni]
Loop
```

Esempi:

Nel primo esempio il loop sarà ripetuto fintanto che 'xyz' rimane 5:

```
Do While xyz = 5
(linee di codice)
Loop
```

Per favore nota che le linee di codice non saranno mai eseguite se 'xyz' non è 5 quando si entra nel ciclo.

Per superare il problema di "codice mai eseguito", puoi spostare la condizione alla fine del ciclo:

```
Do
  (linee di codice)
Loop While xyz = 5
```

Ora le linee di codice saranno eseguite almeno una volta prima che il ciclo controlli il valore di 'xyz' e decida se uscire o iterare l'esecuzione.

Se c'è una seconda condizione che può insorgere da qualche parte dentro il loop, rendendo necessario uscire dal ciclo prima che le linee di codice terminino all'interno, può servire l'istruzione 'Exit':

```
Do
  (linee di codice)
  If abc = 0 Then Exit Loop
  (linee di codice)
Loop While xyz <> 5
```

Questo significa che, se 'abc' diventa zero nell'istruzione 'If' dentro il ciclo, si uscirà dal ciclo. Saranno saltate le restanti linee di codice.

Qualche volta è necessario saltare le restanti linee di codice ma cionondimeno rimanere dentro il ciclo. Ora puoi usare 'Iterate' invece:

```
Do
  (linee di codice)
  If abc = 0 Then Iterate Loop
  (linee di codice)
Loop While xyz <> 5
```

Questo vuol dire che se 'abc' diviene zero in qualsiasi momento dentro il loop, Si oltrepasserà il resto delle linee di codice all'interno del ciclo per saltare direttamente a "Loop While xyz = 5". Ora il loop può controllare il valore di xyz e decidere cosa fare.

Puoi usare la condizione con espressioni come 'And' e 'Or':

```
Do While x < 10 And y < 10
  (linee di codice)
Loop
```

Reitererà mentre x e y sono più piccoli di 10.

Nota: Per favore stai attento quando annidi diversi cicli l'uno dentro l'altro. Talvolta una variabile esaminata nel ciclo esterno è modificata nel ciclo interno e perciò il loop esterno non finirà mai:

```
Do While counter < 10
  (linee di codice)
  counter = 0
  Do
```

```
    counter = counter + 1
  Loop While counter <> 5
Loop
```

Sarebbe una buona idea usare variabili separate per ogni ciclo. Lo stesso potrebbe succedere all'interno di un ciclo 'For...Next':

```
Do While counter < 10
  (linee di codice)
  For counter = 1 To 5
    (linee di codice)
  Next counter
Loop
```

Inoltre, stai attento a non scambiare differenti cicli:

```
Do While counter < 10
  (linee di codice)
  For i = 1 TO 5
    (linee di codice)
  Loop
Next i
```

Questo risulterà in un errore poiché 'Loop' deve apparire dopo 'Next'.

```
x = 5
Do While x = 5
  Print x
Loop
```

Do...Loop Until

Si ripete fino a che resta impostata una condizione. Esempio:

```
x = 0
Do
  Print x
  x = x + 1
Loop Until x > 3
```

Sintassi:

```
Do
  [Istruzioni]
Loop Until Espressione
```

Do...Loop While

Reitera mentre rimane impostata una condizione. Esempio:

```
x = 0
Do
```

```
  Print x
  x = x + 1
Loop While x < 3
```

Sintassi:

```
Do
  [Istruzioni]
Loop While Espressione
```

Do Until...Loop

Un gruppo d'istruzioni che ti mettono in grado di definire un loop che si ripeterà finché una certa condizione è 'True'.

```
x = 0
Do Until x > 3
  Print x
  x = x + 1
Loop
```

Sintassi:

```
Do Until Espressione
  [Istruzioni]
Loop
```

While...Wend

'While' inizia un ciclo 'While...Wend'. Il ciclo termina con 'Wend', e l'esecuzione reitera attraverso il ciclo fintanto che l'espressione è 'True'. Esempio:

```
While True
  Print 1234
Wend
```

Sintassi:

```
While Espressione
  [Istruzioni]
Wend
```

While...End While

E' lo stesso di 'While...Wend', ma con parole chiavi leggermente diverse, come hai notato. Reitera attraverso il ciclo finché l'espressione è 'True'.

```
While True
  Print 1234
End While
```

Sintassi:

```
While Espressione
  [istruzioni]
End While
```

Abbandonare esplicitamente un ciclo

Normalmente un ciclo viene abbandonato quando la sua condizione lo consente, ma qualche volta potrebbe essere utile uscire da un ciclo molto prima. Puoi uscire manualmente da un ciclo usando l'istruzione 'Exit Do'. Di conseguenza sarà lasciato il ciclo corrente. Se vuoi abbandonare i cicli 'For', devi usare 'Exit For'.

- 'Exit For' – abbandona un ciclo for
- 'Exit Do' – abbandona un ciclo do

Per esempio, potresti usare 'Exit Do' in un'istruzione 'If' o 'Select Case', se vuoi abbandonare un ciclo infinito.

Sintassi:

```
Exit For
Exit Do
```

Verificare esplicitamente una condizione

Normalmente una condizione di loop viene verificata dopo ogni iterazione, ma talvolta potrebbe essere utile verificare la condizione molto prima. Puoi verificare manualmente una condizione di loop usando l'istruzione 'Iterate Do'. Di conseguenza sarà verificata di nuovo la condizione del ciclo. Se vuoi testare condizioni di cicli 'For' devi usare 'Iterate For'.

- 'Iterate For' – testa manualmente una condizione di un ciclo 'For'
- 'Iterate Do' – testa manualmente una condizione di un ciclo 'Do'

Sintassi:

```
Iterate For
Iterate Do
```

Strutture di controllo annidate

Puoi incorporare strutture di controllo dentro altre strutture di controllo, es. Puoi mettere un'istruzione For'-loop dentro un'istruzione 'If'. Queste si chiamano strutture di controllo annidate.

Puoi inserire tante strutture di controllo quante ne vuoi ma dovresti indentare (mettere delle tabulazioni, *ndt*) ogni riga, affinché il tuo codice sia più facile da leggere (vedi gli esempi in questo libro, che sono sempre indentati e formattati).

Procedure

Probabilmente i tuoi programmi sono stati alquanto corti, ognuno concepito per dimostrare una singola tecnica di programmazione. Quando inizi a scrivere programmi reali, comunque, scoprirai che possono crescere fino a molte pagine di codice. Quando i programmi diventano lunghi, iniziano anche a divenire difficili da organizzare e leggere. Per avere una soluzione a questo problema, programmatori professionisti hanno sviluppato qualcosa chiamata programmazione modulare, usando procedure. Una procedura è come un piccolo programma dentro il programma principale.

Il codice sorgente KBasic è, normalmente, all'interno di una procedura. Una procedura è un insieme di comandi dentro le parole 'Sub' e 'End Sub' o 'Function' e 'End Function'. Ci sono diversi tipi di procedure.

- *Sub (dentro un modulo o classe)*
  Una sottoprocedura contiene comandi e istruzioni, ma non restituisce un valore o non può essere usata in espressioni. Le procedure evento sono sempre sottoprocedure. Una procedura evento è in relazione ad un form o ad un controllo. Quando KBasic si accorge che si è verificato un evento di un controllo, allora chiama la procedura evento collegata.
- *Funzioni (all'interno di un modulo o classe)*
  Una funzione contiene comandi e istruzioni. Una funzione restituisce sempre un valore, es. Il risultato di una complessa operazione matematica. Poiché le funzione ritornano valori, possono essere usate in espressioni. Le subroutine (sottoprocedure, *ndt*) possono avere argomenti.
- *Sub (dentro una classe)*
  meglio nota come *metodo*, senza un valore di ritorno. E' come una sottoprocedura.
- *Funzioni (all'interno di una classe)*
  meglio nota come *metodo* con un valore di ritorno. E' come una funzione.

Sottoprocedura

Una sottoprocedura può avere argomenti, es. variabili, espressioni o costanti, che sono passati alla sottoprocedura mentre la si chiama. Molte funzioni predefinite di KBasic hanno argomenti.

Sintassi:

```
Sub Nome([Argomenti])
  [Istruzioni]
End Sub

Sub Nome([Argomenti]) [Throws Nome, ...]
  [Istruzioni]
End Sub
```

Funzioni

Una sottoprocedura può avere argomenti, es. variabili, espressioni o costanti, che sono passati alla subroutine quando la si chiama. Le funzioni restituiscono valori. Molte funzioni predefinite di KBasic hanno argomenti (es. Abs).

Sintassi:

```
Function Nome([Argomenti]) [As Tipo]
  [Istruzioni]
End Function

Function Nome([Argomenti]) [As Tipo] [Throws Nome, ...]
  [Istruzioni]
End Function
```

Argomenti

Una procedura può avere tanti argomenti quanti ne vuoi, per tutti gli scopi pratici. Ma devi essere sicuro che gli argomenti che specifichi nella chiamata della procedura corrispondano esattamente nel tipo e nell'ordine, agli argomenti nella sottoprocedura.

Per usare più di un argomento in una procedura, separa gli argomenti con una virgola. Puoi passare uno o più argomenti ad una procedura, sebbene devi tenere a mente che gli argomenti sono passati nell'ordine in cui appaiono nella chiamata. La riga della procedura deve elencare gli argomenti nello stesso ordine in cui sono elencati nella chiamata.

Se una procedura non ha argomenti, non devi scrivere nessuna espressione o variabile dentro le parentesi, quando la chiami.

Tutte le istruzioni e i comandi sono eseguiti solo dopo la chiamata della procedura.

Gli argomenti hanno nomi. Se hai molti argomenti, li puoi separare tramite virgola (,). Ogni argomento è come una dichiarazione di variabile, gli assomiglia e conduce a variabili dichiarate automaticamente, quando sono eseguite le istruzioni e i comandi di una procedura.

Sintassi degli argomenti:

```
Nome As Tipo

[ByVal | ByRef] Nome As Tipo

[ByVal | ByRef] Nome [As Tipo]

[ByVal | ByRef] Nome [()][As Tipo]

[ByVal | ByRef] [Optional] Nome [()][As Tipo] [= Espressione]
```

Un tipo può essere uno dei built-in (tipi scalari), 'Variant' o il tipo object/class.
Se non dici a KBasic il tipo, presuppone quello di default, che di norma è 'Variant'.

'ByRef' e 'ByVal' sono modificatori. Se chiami una procedura, e un argomento è una variabile e l'argomento corrispondente viene dichiarato 'ByRef', allora KBasic fa tutte le operazioni su quell'argomento sulla variabile data, e non copia il valore della variabile. 'ByVal' risulta nella copia del valore di una variabile. Quindi se i comandi dentro la procedura modificano il valore dell'argomento, la variabile non ne è influenzata. Se usi 'ByRef' viene modificata!

Argomenti con nome ed opzionali

Ci sono due modi per passare degli argomenti ad una procedura. Un modo è quello di nominare gli argomenti, l'altro è quello della relativa posizione (default).

Argomenti – il modo normale

```
PassArg(Frank", 26, #2-28-79#)
```

Argomenti con nome:

```
PassArg(intAge = 26, birthDate := #2/28/79#, strName := "Frank")
```

Se usi i nomi di argomento puoi elencare gli argomenti nell'ordine che preferisci. E' probabilmente più leggibile.

Esempio:

```
Sub passArg(strName As String, intAge As Integer, birthDate As Date)
     Print strName, intAge,  birthDate
End Sub
```

Un argomento con nome contiene il nome dell'argomento, i due punti (:) e il segno di uguale (=), e l'espressione che l'argomento dovrebbe avere. Gli argomenti con nome sono anche utili quando usi procedure con argomenti opzionali.

Gli argomenti opzionali non sono sempre presenti. Dipende se sono stati dati alla procedura. Quindi non devi passarli alla subroutine. Gli argomenti opzionali possono avere dei valori di default come si vede nell'esempio seguente:

```
Sub optionaleArg(str As String, Optional strCountry As String =
"Germany")
...
End Sub
```

Se non passi un argomento opzionale alla procedura, viene usato il valore di default.

Potrebbe essere utile controllare (la presenza di) un argomento con la funzione 'IsMissing':

```
Sub optionaleArg(str As String, _
Optional strCountry As String = "Germany")
     If IsMissing(strCountry) Then
          Print str
     Else
          Print str, strCountry
     End If
End Sub
```

Puoi chiamare la procedura di sopra con le seguenti righe:

```
optionaleArg(str := „test1“, strCountry := "Germany")
optionaleArg(str := „test1“)
```

Sono anche possibili argomenti con valore di default come si vede nell'esempio che segue:

```
Sub optionaleArg(str As String, strCountry As String = "Germany")
...
End Sub
```

Scrivere funzioni

Una funzione contiene comandi e istruzioni, che sono dopo 'Function ...' e prima di 'End Function'. Una funzione è come una sottoprocedura, ma può anche restituire un valore. Quella è la loro unica differenza. Una funzione può anche avere argomenti. Un valore può essere restituito usando il nome della funzione come una variabile, che viene usata in un'istruzione di assegnamento (vecchio stile). Ma dovresti usare la keyword 'Return' per ritornare valori, che è il nuovo stile.

Esempio:

```
Sub Main()
     temp = InputBox(Prompt:= _
          "Fahrenheit?")
     MsgBox "Fahrenheit = " & Celsius(temp)
End Sub

Function Celsius(fGrad)
     Celsius = (fGrad - 32) * 5 / 9
End Function

Main()
```

Chiamata di sottoprocedure e funzioni

Per chiamare una sottoprocedura dentro un'altra procedura, devi scrivere il nome della subroutine da chiamare, e tutti gli argomenti necessari. La parola chiave 'Call' non è necessaria. Se la utilizzi, tutti gli argomenti devono essere posti dentro le parentesi. Dovresti usare procedure per organizzare e spezzare il codice, per renderlo più leggibile.

Esempio:

```
Sub Main()
     MultiBeep 56
     DoMessage
End Sub

Sub MultiBeep(no)
     Dim counter As Integer
     For counter = 1 To no
          Beep
     Next counter
End Sub

Sub DoMessage()
     Print "Tee time!"
End Sub

Main()
```

Aggiungere commenti ad una procedura

Quando crei una nuova procedura o modifichi del codice, potrebbe essere utile commentare il nuovo codice. Questi commenti non hanno effetto sul tuo programma, quando è eseguito. Aiutano solo a far capire il codice agli altri sviluppatori. I commenti iniziano con ('). Questo carattere dice a KBasic di ignorare tutto il testo fino a che è stata raggiunta la fine della riga. Ci sono altri tipi di commenti, puoi trovare maggiori informazioni a riguardo nel capitolo precedente.

Chiamata di procedure con lo stesso nome

Puoi chiamare una sottoprocedura, da qualsiasi parte tu voglia. Se è un diverso modulo, potrebbe essere utile usare il nome del modulo per indirizzare la sottoprocedura. Esempio:

```
Sub Main()
     Module1.myProcedure()
End Sub
```

Suggerimenti per chiamare procedure

- Quando modifichi il nome di classi o moduli, sii sicuro di cambiare ogni punto in cui è usato il nuovo nome di classe o modulo, altrimenti KBasic incorrerà in un errore. Puoi anche usare la voce di menu *Edit/Replace*.

- Per evitare collisioni di nomi, dovresti dare nomi univoci alle tue procedure .

Lasciare una procedura

Puoi abbandonare una procedura ad ogni riga al suo interno. Per far ciò, puoi usare le parole chiavi 'Exit Sub' dentro una sottoprocedura o 'Exit Function' all'interno di una funzione.

Sintassi:

```
Exit Sub

Exit Function  ' dentro una funzione
```

Chiamare procedure con più di un argomento

Il seguente esempio mostra una chiamata di sottoprocedura con più di un argomento.

```
Sub Main()
     costs (99800, 43100)
     Call costs (380950, 49500) ' vecchio stile
End Sub

Sub costs (price As Single, income As Single)
     If 2.5 * income <= 1.8 * price Then
          Print "La casa è troppo cara."
     Else
          Print "La casa è economica."
     End If
End Sub
```

Uso di parentesi nel chiamare una funzione

Per ottenere il valore di ritorno di una funzione, hai bisogno di una variabile. Esempio:

```
answer3 = MsgBox("Sei soddisfatto del tuo reddito?")
```

Se non hai l'esigenza di un valore di ritorno, puoi ignorarlo e usare la funzione come una sottoprocedura ordinaria. Esempio:

```
MsgBox "Operazione compiuta!"
```

Scrivere procedure ricorsive

Le procedure hanno memoria limitata, per cui se una procedura chiama se stessa, consuma ancora più memoria. Quando le procedure chiamano sé stesse si definiscono procedure ricorsive
Esempio:

```
Function doError(Maximum)
      doError = doError(Maximum)
End Function
```

Questo errore può essere nascosto quando due procedure si chiamano l'un l'altra una volta e ancora una volta o se non è usata nessuna condizione di break. Ma in alcune situazioni le procedure ricorsive potrebbero essere utili. Esempio:

```
Function Fakulty (N)
     If N <= 1 Then   ' fine ricorsione
       Fakulty = 1    ' (N = 0) non più chiamate
     Else ' Fakulty è chiamata di nuovo
       ' quando N > 0.
       Fakulty = Fakulty(N - 1) * N
     End If
End Function
```

Dovresti verificare una procedura ricorsiva per essere sicuro che funzioni come ci si aspetta. Se viene sollevato un errore, controlla la condizione di uscita. Cerca di minimizzare il consumo di memoria:

- Rilascia variabili non usate
- Evita i tipi di dati 'Variant'
- Controlla la logica della procedura. Meglio usare cicli all'interno di cicli che chiamate ricorsive.

Overloading di procedure

In altri linguaggi, il nome del metodo (o funzione, o procedura) è abbastanza per distinguerlo da altri metodi nel programma. In KBasic le procedure non si distinguono solo tramite il loro nome, ma anche dai loro argomenti e dipendono da una classe o modulo. Se chiami una procedura e ci sono due o più procedure con lo stesso nome, KBasic prova a verificare quale procedura intendi, controllando gli argomenti.

Definire procedure con lo stesso nome, ma con diversi argomenti è chiamato overloading. Potrebbe servire in qualche situazione. Non confonderlo con l'overriding.

Procedure evento

Quando KBasic riconosce che un form o un controllo ha sollevato un evento, esso chiama automaticamente la procedura evento interessata dentro un form.

Procedure generiche

KBasic chiama una procedura evento dopo che si è verificato un evento. Le normali procedure vengono chiamate solo esplicitamente.

Se hai bisogno di diverse procedure evento per eseguire un compito, potresti creare una sottoprocedura, che è chiamata all'interno della procedura evento. E' un modo migliore di copiare tutto il codice in ogni procedura evento, e quindi di duplicare.

Puoi creare normali procedure in form o moduli. Se la normale procedura è legata solo a un form, collocala lì. Se è in relazione con l'intero programma, mettila in un modulo.

Funzioni

Le funzioni sono molto simili alle sub, entrambe sono procedure, ma le funzioni restituiscono un valore e possono essere usate in espressioni.

Sintassi:

```
Function Nome([Argomenti])[As Tipo]
  [Istruzioni]
End Function

Function Nome([Arguments])[As Tipo] [Throws Nome, ...]
  [Istruzioni]
End Function
```

Gli argomenti di una funzione si usano come quelli di una sottoprocedura. La definizione di una funzione viene fatta usando la keyword 'Function'. Dovresti anche ritornare un valore per ogni funzione. Per maggiori informazioni sull'uso di procedure vedi il capitolo precedente.

Tipi di dati delle funzioni

Come le variabili hanno tipi di dati, anche le funzioni gli hanno. Quando definisci una funzione, definisci anche un tipo di ritorno, che è il tipo di dato della funzione. Se non definisci un tipo di ritorno, è assunto il tipo di default, che normalmente è 'Variant'.

Ritorno da una funzione

Se vuoi impostare il valore di ritorno di una funzione e anche uscire dalla funzione, dovresti usare 'Return'. Con questa istruzione abbandoni immediatamente la funzione.

Sintassi:

```
Return Espressione
```

Proprietà

La maggior parte degli oggetti e controlli hanno delle proprietà, che puoi pensare come niente di più complicato degli attributi. Per esempio, come hai già imparato, un controllo 'CommandButton' ha una proprietà chiamata 'Caption' (titolo, *ndt*), che determina il testo che appare nel pulsante. Molti altri controlli, come il familiare 'Label' (etichetta, *ndt*), hanno anche loro una proprietà 'Caption'. Alcune proprietà sono comuni in KBasic, laddove altre sono associate solo ad un singolo controllo o oggetto. Queste proprietà sono built-in in KBasic, ma è anche possibile definire delle tue proprietà dentro una classe utente (non built-in, *ndt*).

Definire una proprietà è come definire una procedura. Infatti, una proprietà contiene due procedure proprietà. Una legge la proprietà (Get), l'altra la scrive (Set). In aggiunta, devi dichiarare una variabile, che contiene il valore della proprietà. Perché dovrei usare una proprietà, quando posso invece usare una variabile, potresti chiederti? Bene, è più facile accedere ad una variabile, ma potrebbe essere più utile controllare qualcosa mentre accedi ad una variabile; e con le *property* sei in grado di controllare valori o condizioni, quando cerchi di accedere alla proprietà. E' come nascondere dati. Puoi controllare che il valore di una property sia sempre corretto.

Per i dettagli dovresti esaminare gli esempi, che giungono con KBasic.

Sintassi:

```
Property Set Nome(Argomento)
  [Istruzioni]
End Property

Property Get Nome As Tipo
  [Istruzioni]
End Property


Property Nome As Tipo

  Get
    [Istruzioni]
  End Get

  Set(Argomento)
    [Istruzioni]
  End Set

End Property
```

Conversione

Normalmente, non devi modificare il tipo di dato di un espressione. La maggior parte delle volte ciò viene fatto automaticamente per te senza avviso. Ma qualche volta potrebbe essere importante forzare una conversione, poiché altrimenti il tuo calcolo potrebbe essere sbagliato.

Esistono alcune funzioni built-in di conversione per ogni tipo di dato:

```
y% = CInt (espressione)  ' restituisce un Integer
y% = CLng (espressione)  ' restituisce un Long
y! = CSng (espressione)  ' restituisce un Single
y' = CDbl (espressione)  ' restituisce un Double
y@ = CCur (espressione)  ' restituisce un Currency
y = CVar  (espressione)  ' restituisce un Variant
y = CBool (espressione)  ' restituisce un Boolean
y = CByte (espressione)  ' restituisce un Byte
y = CShort(espressione)  ' restituisce un Short
y = CDate (espressione)  ' restituisce un Date
```

Modificatori / Ambiti

Questo è un aspetto molto importante del linguaggio di programmazione KBasic o di un moderno linguaggio in generale. Quindi sii sicuro di comprenderlo del tutto!

Gli scope (visibilità, ambito, o campo d'azione, *ndt*) dicono a KBasic quando una variabile, una costante, o una procedura, può essere usata, effettivamente da quale parte del codice sorgente. Ci sono differenti scope (luoghi astratti nel codice):

- Scope di procedura
- Scope globale
- Scope di modulo privato
- Scope di modulo pubblico
- Scope di classe privata
- Scope di classe protected
- Scope di classe pubblica

Gli ambiti sono utili quando desideri organizzare il tuo programma o vuoi evitare collisioni di nomi.

Quando ti riferisci ad una variabile all'interno delle definizioni dei tuoi metodi, KBasic controlla la definizione di quella variabile prima nell'ambito attuale, poi in quelli esterni alla definizione dell'attuale metodo. Se quella variabile non è locale, KBasic allora cerca una sua definizione come istanza di variabile, nella classe attuale, e poi, alla fine, in ogni classe genitore in successione.

A causa del modo in cui KBasic controlla la visibilità di una data variabile, è possibile per te creare una variabile in un ambito inferiore in modo tale che una definizione della stessa variabile nasconda il suo valore originale. Questo può introdurre bug subdoli e che confondono. Il modo più semplice per evitare questo problema è assicurarti di non usare gli stessi nomi per le variabili locali come fai per le variabili di istanza. Un altro modo per eludere questo particolare problema, comunque, è usare 'Me.variabileName' per riferirsi alla variabile d'istanza, e solo il nome della variabile per far riferimento a quella locale. Riferendoti esplicitamente alla variabile di istanza tramite lo scope del suo oggetto eviti dei conflitti.

Visibilità locale

Una variabile dichiarata dentro una procedura non è utilizzabile all'esterno della procedura, solo il codice della procedura in cui è stata dichiarata la variabile, può usarla.

Il seguente esempio dimostra come può essere usata una variabile locale. Ci sono due procedure, una di esse ha una variabile dichiarata :

```
Sub localVar()
  Dim str As String
```

```
  Str = "Questa variabile può essere usata solo dentro questa
procedura"
  Print str
End Sub

Sub outOfScope()
     Print str  ' causa un errore di parsing
End Sub
```

Visibilità di un modulo privato

Se dichiari una variabile come 'Private', è visibile solo nel modulo in cui è dichiarata. Se la dichiari come 'Public', è visibile all'intero programma (tutti i moduli e classi). Default è 'Private', quando usi la keyword 'Dim' per dichiarare variabili.

Esempio:

```
' Aggiungi alla sezione di dichiarazione del tuo
' modulo le seguenti righe
Private str As String

Sub setPrivateVariable()
  str = "Questa variabile può essere usata solo dentro questo
modulo"
End Sub

Sub usePrivateVariable()
  Print str
End Sub
```

Visibilità di un modulo pubblico

Quando dichiari una variabile come 'Public' dentro un modulo, essa è visibile a tutte le parti del programma.

Esempio:

```
' Aggiungi alla sezione di dichiarazione del tuo
' modulo le seguenti righe
Public str As String
```

Tutte le procedure sono di default 'Public' eccetto le procedure evento. Poiché KBasic scrive automaticamente 'Private' per ogni dichiarazione di procedura evento. Se vuoi avere una normale procedura 'Private' scrivi la parola chiave 'Private' prima della sua dichiarazione.

Da FARE #4

Tipi di dati definiti dall'utente (Type...End Type)

E' stato molto utile ai tempi in cui la programmazione object-oriented non era ancora disponibile. E' come una classe, diverse variabili sono tenute assieme, ma senza metodi. Sono consentiti tutti i tipi di dati all'interno di un tipo definito dall'utente, persino altri tipi di dati definiti dall'utente possono essere inclusi.

Sintassi:

```
Type Nome
 Nome [(Indice)] As Tipo
  ...
End Type
```

Enumerazione (Enum...End Enum)

E' un nuovo modo di raggruppare costanti, che si riferiscono allo stesso soggetto. E' una lista di nomi. Ogni nome ha un valore costante.

Sintassi:

```
Enum Nome
  Nome [= Espressione]
  ...
End Enum
```

DA FARE #5
$End
End
Stop
Destructor
Signal
Slot
IsArray ...
Class Static Block

Classi

Una semplice applicazione può contenere un form, mentre tutto il codice sorgente è dentro il modulo di quel form, ma se l'applicazione diventa più grande e più grande, potrebbe essere utile scrivere il codice di diversi form in un unico punto; quindi hai bisogno di un posto per scrivere il codice al di fuori dei form. Qui giungono le classi. Crea una classe, utile ai tuoi form, che contenga dei metodi.

Il codice KBasic è memorizzato in classi o moduli. Puoi archiviare il tuo codice all'interno di classi. Ogni classe consiste della parte relativa alla dichiarazione e dei metodi che hai inserito.

Una classe può contenere:

- *dichiarazioni,*
  per variabili, tipi, enumerazioni e costanti

- *Metodi (chiamati anche procedure)*
  i quali non sono assegnati ad uno speciale evento o oggetto. Puoi creare tante procedure quante ne vuoi, es. sottoprocedure senza valore di ritorno o funzioni.

- *Metodi segnale/Slot*
  Questi sono dei metodi speciali, che sono utili solo assieme ai binding. Vedi la documentazione sui binding nell'IDE KBasic per maggiori informazioni.

- *Proprietà*
  Sono variabili, a cui si accede attraverso due metodi speciali (metodo *get* e *set*). Le property sono accessibili senza bisogno di scrivere le parentesi, quando le usi.

Puoi mettere parecchie classi o moduli in un file, ma non dovresti farlo.

Le classi non vengono come le procedure, eseguite

Una classe consiste solo della parte di dichiarazione. Devi creare tu stesso i metodi. In più, una classe non contiene una programma principale. I metodi sono eseguibili all'interno delle classe. Sono eseguiti se viene sollevato l'evento pertinente, o se un'altra parte del programma ha chiamato un metodo di quella classe.

Editare una classe

Non è molto diverso editare del testo in un elaboratore di testi o in KBasic Software Atelier. Hai un cursore che mostra la posizione corrente, dove puoi scrivere. Puoi cercare e sostituire dentro il tuo codice sorgente come in un word processor e così via...

Sintassi:

```
[Abstract] Class Nome Inherits NomeClasseGenitore

  [Static] Dim Nome As Tipo
```

```
[Static] Public Nome As Tipo
[Static] Protected Nome As Tipo
[Static] Private Nome As Tipo
Const Nome As Tipo
Public Const Nome As Tipo
Protected Const Nome As Tipo
Private Const Nome As Tipo
...


[Public | Protected | Private]
Enum Nome
  Nome As Tipo
  ...
End Enum
...


[Public | Protected | Private]
Type Nome
  Nome As Tipo
  ...
End Type
...


[Public | Protected | Private]
Property Nome As Tipo

  Get
    [Istruzione]
  End Get

  Set(Argomento)
    [Istruzione]
  End Set

End Property
...


[Public | Protected | Private]
Constructor Nome([Argomenti])
  [Istruzioni]
End Constructor
...


[Public | Protected | Private]
Destructor Nome( )
  [Istruzioni]
End Destructor


[Static] [Public | Protected | Private]
Function Nome([Argomenti]) [As Tipo] [Throws Nome, ...]
```

```
    [Istruzioni]
  End Function
  ...


  [Static] [Public | Protected | Private]
  Sub Nome([Argomenti]) [Throws Nome, ...]
    [Istruzioni]
  End Sub
  ...


  [Public | Protected | Private]
  Slot Nome([Argomenti])
    [Istruzioni]
  End Slot
  ...


  [Public | Protected | Private]
  Signal Nome([Argomenti])
    [Istruzioni]
  End Signal
  ...


End Class
```

Un esempio più lungo:

```
Class Salsa Inherits Waltz

  Public Enum class_enum
     Entry
     Entry2
     Security = Entry
  End Enum

  Public Type class_type
    element As Integer
  End Type

  Const classConst = 4

  Public publicInstanceVar As Integer
  Private privateInstanceVar As Integer
  Protected protectedInstanceVar As Integer

  Static Public publicClassVar As Integer
  Dim publicModuleType As module1.module_type
  Dim publicModuleType2 As module_type

  'chiamata di un costruttore genitore all'interno di un costruttore

  Sub meExplicit()
```

```
    Dim localVar = Me.publicInstanceVar 'E' lo stesso con parent
    Dim localVar2 = Me.publicClassVar
    Dim localVar3 = Salsa.publicClassVar
    Dim localVar4 = Salsa.classConst
    Dim localVar5 = Me.classConst
'
    Dim localVar6 As class_enum
    localVar6 = Salsa.class_enum.Entry
'    Dim localVar7 As Me.class_enum
'Il nome completo del tipo non è permesso
    Dim localVar8 As class_type
  End Sub

  Sub meImplicit()
    Dim localVar = publicInstanceVar
    Dim localVar2 = publicClassVar
    Dim localVar3 = classConst
    Dim localVar4 As class_enum
    Dim localVar5 As class_type

  End Sub

  Sub classSub()
    Const localConst = 6
    Dim n = localConst
  End Sub

  Sub classSubWithArgument(i As Integer)
    Dim localVar = i
  End Sub

  Function classFunction() As String
    Return "hello"
  End Function


  Static Public Sub test() Throws Waltz
    Throw New Waltz
  End Sub

  Private pvtFname As String

  Public Property Nickname As String

    Get
      Print "Hi"
    End Get

    Set ( ByVal Value As String )
      Print "Hi"
    End Set

  End Property

End Class
```

Modulo

Una semplice applicazione può consistere di un solo form, mentre il codice sorgente completo è in un modulo. Quando le applicazioni diventano sempre più grandi, probabilmente vorresti usare lo stesso codice in diversi form. Quindi è utile mettere questo codice in un modulo globale, da dove è accessibile all'intera applicazione .

Il codice KBasic è memorizzato in classi o moduli. Puoi archiviare il tuo codice dentro un modulo. Ogni modulo consiste della parte relativa alla dichiarazione e delle procedure che hai inserito.

Un modulo può contenere:

- *dichiarazioni*
  per variabili, tipi, enumerazioni e costanti

- *procedure*
  Le quali non sono assegnate ad uno speciale evento o oggetto. Puoi creare tante procedure quante ne vuoi, es. Sottoprocedure senza valore di ritorno o funzioni.

Puoi porre parecchie classi e moduli in un file, ma non dovresti farlo.

Moduli globali

I moduli globali sono sempre presenti. Puoi accedervi da qualsiasi parte del tuo programma. Metti là le procedure che usi spesso nei form o nelle procedure evento .

I moduli non sono come le procedure, eseguiti

Un modulo consiste solo della parte relativa alla dichiarazione. Devi creare tu stesso le procedure. In aggiunta un modulo non contiene un programma principale. Le procedure sono eseguite se un'altra parte del tuo programma ha chiamato la procedura.

Editare un modulo

Non è molto diverso editare un testo in un elaboratore di testi o in KBasic Software Atelier. Hai un cursore che mostra la posizione corrente, dove puoi scrivere. Puoi cercare e sostituire dentro il tuo codice sorgente come in un word processor e così via...

Sintassi:

```
Module Nome


  Dim Nome As Tipo
  Public Nome As Tipo
  Private Nome As Tipo
  Const Nome As Tipo
  Public Const Nome As Tipo
```

```
  Private Const Nome As Tipo
  ...


  [Public | Private]
  Enum Nome
    Nome As Tipo
    ...
  End Enum
  ...


  [Public | Private]
  Type Nome
    Nome As Type
    ...
  End Type
  ...


  [Public | Private]
  Function Nome([Argomenti]) [As Tipo] [Throws Nome, ...]
    [Istruzioni]
  End Function
  ...


  [Public | Private]
  Sub Nome([Argomenti]) [Throws Nome, ...]
    [Istruzioni]
  End Sub
  ...


End Module
```

Un esempio più lungo:

```
Module module1

  Public Type address
    age As Integer
  End Type

  Public Type module_type
    element AS integer
  End Type

  Public Enum module_enum
     Entry
     Entry2
     Security = Entry
```

```
End Enum

Const moduleConst = 7

Public publicModuleVar As Integer
Private privateModuleVar As Integer


Sub moduleExplicit()
  Dim localVar = module1.publicModuleVar
  Dim localVar2 = module1.moduleConst
  Dim localVar3 As module_enum
  localVar3 = module1.module_enum.Entry
End Sub

Sub moduleImplicit()
  Dim localVar = publicModuleVar
  Dim localVar2 = moduleConst
  Dim localVar3 As module_enum
  localVar3 = module_enum.Entry
  Dim localVar4 As module_type
End Sub

Sub moduleSubWithDefaultArgument(ko As Integer = 6)
  Dim localVar = ko
End Sub

Sub moduleSubWithOptionalArgument(Optional ko As Integer)
  If Not IsMissing(ko) Then
    Dim localVar = ko
  End If
End Sub

Sub moduleSub()
  Const localConst = 6
  Dim n = localConst
End Sub

Sub moduleSubWithArgument(i As Integer)
  Dim localVar = i
End Sub

Sub moduleSubWithArgumentShadowing(i2 As Integer)
  Dim localVar = i2
  Dim i2 = localVar + 99
  Dim i3 = i2
End Sub

Sub subOverloading ( )
  Print "sub1"
End Sub

Sub subOverloading ( i As Integer = 1)
  Print "sub2"
End Sub
```

```
  Function moduleFunction() As String

    subOverloading()
    subOverloading(88)

    Return "hello"
  End Function

  Function moduleFunctionRecursive(ByRef i As Integer) As  Integer
    If i > 6 Then Return 1''i

    ''i = i + 1
    Return moduleFunctionRecursive(1)''i)
  End Function

End Module
```

Trattamento degli errori

Sviluppatori di ogni linguaggio hanno sempre voluto scrivere programmi esenti da errori, programmi che non vanno mai in crash (che non si bloccano, *ndt*), programmi che possono gestire qualsiasi situazione con grazia e riprendersi da situazioni inusuali senza causare all'utente alcun stress non dovuto. Buone intenzioni a parte, programmi come questi non esistono. Nei programmi reali si verificano errori, sia perché il programmatore non ha anticipato ogni situazione in cui potrebbe incappare il codice (o non ha avuto il tempo di testare il programma abbastanza), o a causa di situazioni fuori dal suo controllo, dati errati da parte degli utenti, campi corrotti che non hanno i giusti dati in essi, e così via.

Esistono tre possibili sorgenti di errori:

Errori di compilazione
Questi tipi di errori hanno luogo quando le istruzioni sono scritte nel modo sbagliato, che significa che devi utilizzare la giusta sintassi. Forse hai scritto male una keyword, o usato male una parola chiave nel posto sbagliato, o hai dimenticato di scrivere tutti i segni necessari (parentesi, punti e così via).

- *Errori di run-time*
  Questi tipi di errori accadono quando KBasic trova un errore in fase di esecuzione, mentre il programma viene eseguito. Un esempio di tale errore è una divisione per zero, che non è consentita in nessuna espressione matematica.
- *Errori logici*
  Questi tipi di errori si verificano quando sintassi e comportamento a run-time sono ok, ma il programma non funziona come ci si aspettava. Puoi essere sicuro che il tuo programma funzioni come voluto, soltanto se esamini e analizzi qualsiasi risultato.

Se si verifica un errore di runtime, KBasic fermerà l'esecuzione del programma, se non è stato definito alcun gestore degli errori nel punto ove si è verificato l'errore.

E' supportata la vecchia sintassi 'On Error GoTo' del trattamento degli errori, ma dovresti usare la nuova gestione degli errori (eccezioni).

Eccezioni

In KBasic, questa sorta di strani eventi/errori che possono causare al programma di smettere di funzionare , sono chiamate eccezioni. Il maneggiamento delle eccezioni è una caratteristica importante di KBasic. Un eccezione è un segnale che dimostra che è stata raggiunta una fase anormale (come un errore). Creare un eccezione significa segnalare questo speciale stato. Catturare un eccezione vuol dire trattare un eccezione, eseguire alcuni comandi per occuparsi di questo speciale stato allo scopo di giungere ad uno stato normale.

Esempio:

```
Public Sub tt2() Throws samba

  Try
    test()
  Catch (b As rumba)
    Print "tt2: ti ho preso!"
    b.dance()
  Finally
    Print "tt2: sarà sempre eseguito, qualunque cosa accada"
  End Catch

End Sub
```

Le eccezioni si muovono dall'origine alla successiva struttura di controllo (chiamante dell'attuale procedura). KBasic prima testa se è stata definita un'istruzione 'Try-Catch' o throw nell'ambito attuale (procedura ecc.), se No, cerca il chiamante dell'ambito attuale , e cerca ancora di trovare un'istruzione 'Try-Catch' o 'Throw' e così via. Se non viene catturata, KBasic mostrerà l'errore all'utente e sarà arrestata l'esecuzione del programma.

Oggetti eccezione

Un eccezione è un oggetto, che è un'istanza di un 'Object' o di qualsiasi altra classe. Poiché le eccezioni sono oggetti, possono contenere dati e metodi come qualunque altra classe o oggetto.

Trattamento delle eccezioni

L'istruzione 'Try-Catch' è necessaria per catturare un eccezione. 'Try' racchiude il normale codice che dovrebbe girare bene. 'Catch' contiene i comandi che dovrebbero essere eseguiti, se viene sollevata un'eccezione. 'Finally' ha alcuni comandi che dovrebbero essere sempre eseguiti, se è avvenuta un'eccezione o No.

Try

'Try' racchiude il normale codice che dovrebbe girare bene. 'Try' non funziona da solo. Giunge sempre con almeno un'istruzione 'Catch' o 'Finally'.

Catch

Dopo 'Try' e i suoi comandi, puoi definire tante istruzioni 'Catch' quante ne vuoi. Le istruzioni 'Catch' sono come una dichiarazione di variabile, ma l'oggetto eccezione viene solo creato quando corrisponde.

Finally

'Finally' è utile quando hai comandi come accedere ad un file o chiudere un database, che devono essere sempre eseguiti persino se si è verificata un'eccezione.

Se hai definito un'istruzione 'Catch' e 'Finally' ed è sollevata la corrispondente eccezione, prima viene eseguita l'istruzione 'Catch', dopo di quella, l'istruzione 'Finally'.

Dichiarazione di eccezioni

KBasic richiede che, ogni procedura che può sollevare un'eccezione, debba definire un eccezione con 'Throw' nella sua dichiarazione.

Esempio:

```
Sub mySub() Throws myException1, myException2
' Istruzioni che possono sollevare myException1 o myException2
...
End Sub
```

Definire e generate eccezioni

Puoi sollevare le tue eccezioni usando l'istruzione 'Throw', scrivendo dopo di essa l'oggetto da „lanciare“. Spesso crei questi oggetti prima di throw, utilizzando l'istruzione 'New', es.

```
Throw New MyException1()
```

Sintassi:

```
Throw ExceptionObject
```

Se viene lanciata un eccezione, è abbandonata la normale esecuzione del programma e KBasic cerca di trovare un'appropriata istruzione 'Catch' per essa. Cerca tutte le istruzioni includendo il punto in cui è stata lanciata l'eccezione. In questo percorso vengono eseguite tutte le istruzioni 'Finally'. Usare eccezioni è un modo creativo di trattare gli errori, è facile e pulito da leggere. Spesso puoi usare un semplice oggetto errore, ma qualche volta è meglio utilizzare un nuovo oggetto errore auto-creato.

Sintassi:

```
Try
  [Istruzioni]
Catch (Nome As Exception)
  [Istruzioni]
End Catch

Try
  [Istruzioni]
Catch (Nome As Exception)
  [Istruzioni]
Catch (Nome As Exception)
  [Istruzioni]
End Catch

Try
  [Istruzioni]
Catch (Nome As Exception)
  [Istruzioni]
Finally
  [Istruzioni]
End Catch
```

Esempio completo:

```
Class rumba

  Sub dance
    Print "rumba.dance"
  End Sub
```

```
End Class

Class samba

  Sub dance
    Print "samba.dance"
  End Sub

End Class


Public Sub test() Throws rumba, samba
  Throw New rumba  ' ritorna rumba = new rumba
End Sub

Public Sub tt2() Throws samba

  Try
    test()
  Catch (b As rumba)
    Print "tt2: ti ho preso!"
    b.dance()
  Finally
    Print "tt2: sarà sempre eseguito, qualunque cosa accada"
  End  Catch

End Sub


Public Sub tt()

  tt2()

Catch (c As samba)
  Print "tt: ti ho preso!"
  c.dance()
Finally
  Print "tt: sarà sempre eseguito, qualunque cosa accada"
End Sub


tt()
```

Eccezione alla fine di una procedura

E' possibile catturare un'eccezione alla fine di una procedura, che copra l'intera procedura. Rende più facile leggere il tuo codice, ed è qualcosa di simile al vecchio 'On Error GoTo'. Scrivi le istruzioni catch e finally di cui hai bisogno. Non è necessaria un'istruzione try, poiché s'intende tutta la procedura.

Esempio:

```
Public Sub tt()

  tt2()

Catch (c As samba)
  Print "tt: ti ho preso!"
  c.dance()
Finally
  Print "tt: sarà sempre eseguito, qualunque cosa accada"
End  Sub
```

Sintassi:

```
Catch (Nome As Exception)
  [Istruzioni]
Finally
  [Istruzioni]
End Catch
```

L'ambiente di sviluppo KBasic

L'ambiente di sviluppo KBasic contiene finestre, barre degli strumenti, e editor, che facilitano lo sviluppo della tua applicazione KBasic. E' in effetti un ambiente di sviluppo integrato (IDE). Quando si dice che un ambiente di sviluppo è intergrato, questo significa che gli strumenti lavorano assieme. Per esempio, il compilatore potrebbe trovare un errore nel codice sorgente. In più per visualizzare l'errore, KBasic apre il file sorgente nell'editor di testo e salta all'esatta linea di codice, dove si è verificato l'errore. Gran parte dello sviluppo di un'applicazione implica l'aggiunta e la disposizione di controlli nei form. KBasic fornisce dei tool per rendere la progettazione di form un processo semplice.

Finestre

Le finestre di KBasic sono le orecchie nello schermo che usi per sviluppare i tuoi programmi, compreso il monitoraggio dello stato dei tuoi progetti in qualsiasi momento. Queste finestre includono:

- Form designer – trascini e rilasci controlli, come anche li disponi in quest'area di sviluppo principale. E' lo strumento primario che utilizzi per creare i tuoi programmi.
- Project window (finestra di progetto, *ndt*) – in questa finestra lavori con diverse parti del tuo progetto. Le sue viste includono oggetti e file.
- Property window – lista e ti permette di controllare le proprietà dei tuoi controlli. Puoi ridimensionare, nominare, modificare la visibilità, assegnare valori, e modificare i colori e i font dei controlli.
- Toolbox window (casella degli strumenti, *ndt*) – lo spazio centrale dove possono essere raggiunti i controlli usati più spesso
- Source code editor (editor del codice sorgente, *ndt*) – è dove aggiungi e personalizzi il codice sorgente del tuo progetto, in ogni fase del ciclo di sviluppo.

Barre degli strumenti

KBasic fornisce un ampia gamma di toolbar. Le barre degli strumenti possono essere ancorate nella finestra di KBasic o fluttuanti nello schermo.

Editor

KBasic ha un editor per creare e gestire progetti KBasic. Puoi usarlo per controllare lo sviluppo dei tuoi progetti, come editare il codice e manipolare le classi. L'editor del codice sorgente è uno strumento per editare il codice.

Debugger

Uno dei più potenti tool dell'ambiente KBasic è il debugger integrato. (solo la versione professionale). Il debugger ti permette di osservare i tuoi programmi in esecuzione riga per riga. Come si avvia il tuo programma, puoi osservare i vari componenti del programma per vedere come si stanno comportando. Usando il debugger puoi monitorare:

- I valori memorizzati nelle variabili
- Quali sub/funzioni/metodi vengono chiamati
- L'ordine in cui si verificano gli eventi del programma

Collection

DA FARE #6

Classi e oggetti di KBasic

Ci sono due tipi di oggetti in KBasic: oggetti visuali e non visuali. Un oggetto visuale è un controllo ed è visibile in fase di esecuzione, e permette all'utente d'interagire con l'applicazione; ha una posizione sullo schermo, una dimensione e un colore di primo piano. Esempi di oggetti visuali sono form e pulsanti. Un oggetto invisibile non è visibile a runtime, per esempio *Timer*. Alcuni oggetti possono contenere altri componenti, come una finestra di un'applicazione contiene un pulsante. Con KBasic aggiungi oggetti/controlli visuali ai tuoi form per assemblare applicazioni.

Progetti

I progetti tengono assieme il tuo lavoro. Quando sviluppi un applicazione in KBasic, lavori principalmente con progetti. Un progetto è una collezione di file che compongono la tua applicazione. Crei un progetto per gestire e organizzare questi file. KBasic fornisce un sistema semplice ma sofisticato per gestire la collezione di file che formano un progetto. La finestra di progetto mostra ogni elemento di un progetto. Iniziare una nuova applicazione con KBasic comincia con la creazione di un progetto. Quindi, prima che tu possa costruire un applicazione con KBasic, hai bisogno di creare un nuovo progetto.
Un progetto consiste di molti file separati raccolti in una directory di progetto, dove vi è un file di progetto *.KBasic_e molti altri file:

- Modulo *.kbasic_module
- Classe *.kbasic_class
- Form *.kbasic_form

Form

In un'applicazione KBasic i form non sono solo maschere per inserire e modificare dati, ma sono l'interfaccia grafica della tua applicazione. Agli occhi di chi guarda, sono l'applicazione! Creando la tua applicazione usando i form, controlli il flusso del programma con gli eventi, che vengono sollevati nei form.

Significato centrale dei form

Ciascun form in un'applicazione KBasic ha un modulo con procedure evento. Queste procedure reagiscono agli eventi che vengono sollevati nel form. In più, ogni modulo può contenere altre procedure *non-evento*.

Il modulo del form è parte di ogni form, quando copi un form, anche il suo modulo viene automaticamente copiato. Se cancelli un form, pure il suo modulo è cancellato. KBasic crea il modulo di un form in automatico. Per cui hai bisogno solo di scrivere le procedure evento e altre procedure.

I form mantengono assieme il tuo programma KBasic!

Procedure nei form

Le procedure nei moduli del form sono private, puoi usarle solo dentro il modulo del form. Poiché le procedure sono *private* all'interno di un modulo, puoi usare procedure nominate uguali in molti form. Ma i nomi delle procedure nei moduli globali non devono essere uguali.

E'possibile creare una procedura nel modulo di un form, che ha lo stesso nome di una procedura di un modulo globale. In questo caso, KBasic usa due regole per identificare l'esatta procedura da eseguire:

- KBasic cerca prima l'attuale modulo del form
- Se KBasic non trova una procedura con il nome di cui ha bisogno, cerca nei moduli globali (ma non gli altri moduli form).

Quindi tutte le chiamate all'interno un modulo, in cui viene dichiarata la procedura, raggiungono questa procedura. Chiamate all'esterno del modulo avviano la procedura pubblica.

Compatibilità VB6 = KBasic

KBasic supporta al100% la sintassi di VB6. Molti componenti sono anche uguali. E' possibile sviluppare applicazioni GUI con ben nota sintassi BASIC in maniera moderna. Giunge con un vero orientamento agli oggetti simile a Java e con compatibilità all'indietro VB6 e QBasic, poiché è compatibile al 100%. KBasic combina la potenza espressiva di un linguaggio object-oriented come il C++ con la familiarità e la facilità d'uso di VB6. Consente agli sviluppatori con una base di applicazioni VB6 installate, d'iniziare a sviluppare in un ambiente misto Windows, Mac OS X e Linux, senza dover affrontare una ripida curva di apprendimento. KBasic usa il familiare paradigma visuale di design e ha piena implementazione del linguaggio BASIC.

Migrare da VB6 a KBasic

Alcune informazioni che ti potrebbero aiutare a migrare sono elencate nella seguente:

- Non usare ( ) per accedere ad array, meglio usa [ ]
- Non usare 'Option OldBasic' o 'Option VeryOldBasic'
- Non usare 'On Error Goto', meglio usa 'Try Catch'
- Non usare 'Nothing', meglio usa 'Null'
- Evita l'uso del tipo di dato 'Variant'
- Non usare un ' inizializzatore di classe', meglio usa 'Constructor', lo stesso per il 'distruttore'
- Non usare 'Call' quando chiami una subroutine o una funzione
- Non usare 'Optional' e 'IsMissing' con argomenti in subroutine o funzioni, meglio usa il valore di default di un argomento
- Usa sempre 'Do While...Loop' e 'Do ...Loop While' invece degli altri cicli
- Scrivi sempre molti commenti nel codice sorgente
- Usa condizioni 'AndAlso' e 'OrElse' anziché gli operatori binari 'And' e 'Or'
- Evita l'uso di 'ByRef' con tipi di dati primitivi per ottenere un'esecuzione più veloce
- Non usare 'Data' e 'Def*', come 'DefInt'
- Usa enumerazioni al posto di molte costanti intere
- Usa costanti, invece di usare molte volte literal numerici nel tuo codice
- Evita l'uso di 'GoSub', meglio usare reali funzioni o sub
- Evita l'uso di 'GoTo', meglio utilizzare loop e altri elementi di controllo di flusso, del linguaggio
- 'Let' e 'Set' non sono necessari
- Usa 'For Each', quando è possibile
- Usa 'For i As Integer = 0 To 10 Next', invece di dichiarare una variabile contatore all'esterno del ciclo
- Nomina le variabili senza dare loro un suffisso
- Usa sempre ( ) quando chiami una subroutine o una funzione

La macchina virtuale di KBasic

La virtuale machine di KBasic è l'ambiente in cui i programmi di KBasic vengono eseguiti. E' una sorta di computer astratto e definisce i comandi che può utilizzare un programma. Questi comandi si chiamano *p-code* o pseudo-codice. In generale vuol dire che il pcode di KBasic ha lo stesso significato per la macchina virtuale, come il codice macchina lo ha per la CPU. Un pcode è un comando di lunghezza 2 byte, che viene generato dal compilatore e interpretato dall'interprete KBasic. Quando il compilatore compila un programma, produce un insieme di pcode e li salva. L'interprete KBasic è in grado di eseguire questi pseudo-codici. E' importante sapere che il nome 'KBasic', significa molto di più di un linguaggio di programmazione. Vuol dire anche un ambiente completo. La ragione è che KBasic contiene due unità: KBasic design (il linguaggio di programmazione) e KBasic run-time (la virtual machine).

Riassunto:
La macchina virtuale di KBasic è un computer virtuale, in cui sono eseguiti i programmi KBasic. Questo ambiente può eseguire dei comandi speciali, che si chiamano pseudo-codici, i quali sono creati dal compilatore KBasic a partire da un programma.

Quali sono le principali funzioni della macchina virtuale?

La macchina virtuale (VM) esegue i seguenti compiti:

- Assegna la memoria allocata per creare degli oggetti
- Dealloca la memoria automaticamente
- Gestisce lo stack e i registri delle variabili
- Accede al sistema host, ad esempio per usare i dispositivi

Unità lessicale

Le seguenti informazioni descrivono gli identificatori predefiniti e le keyword di KBasic. Alcune di esse sono riservate per usi futuri.

Parole chiavi

Una parola chiave è un identificativo predefinito, che ha un significato speciale per KBasic, che non può essere modificato. La lista che segue contiene tutte le keyword di KBasic.

```
$Dynamic #Else $End #ExternalSource #If #Region $Static Absolute
Abstract AddressOf Alias Ansi As Assembly Auto Base ByRef ByVal Call
CallByName Case Catch Chain Choose Class Class_Initialize
Class_Terminate COM Common Compare Const Constructor Data Database
Decimal Declare Def Default DefBool DefByte DefCur DefDate DefDbl
DefInt DefLng DefObj DefSng DefStr DefVar Delegate Destructor Dim
DirectCast Do Each Else ElseIf Empty End EndIf Enum Erase Event Exit
Explicit Finally For Friend Function Global GoSub GoTo Handles If
IIf Implements Imports In Inherits Interface Is Iterate KBasic Key
LBound Let Lib Like Loop LSet Me Mid Module MustInherit MustOverride
MyBase MyClass NameSpace New Next Nothing NotInheritable
NotOverridable Null Off OldBasic On Option Optional Overloads
Overriddable Overrides ParamArray Parent Pen Play Preserve Private
Property Protected Public Range Read ReadOnly ReDim Rem /** /* */ '
Repeat Restore Resume Return RSet Run Select Set Shadows Shared
Signal SizeOf Slot Static Step Stop STRIG Structure Sub Swap Switch
SynClock System Text Then Throw Throws Timer To TROFF TRON Try Type
TypeDef UBound UniCode Until VARPTR VARPTR$ VARSEG VeryOldBasic Wait
Wend While With WithEvents WriteOnly
```

Funzioni predefinite

```
__Class__ __File__ __IsClass__ __IsLinux__ __IsMacOS__ __IsModule__
__IsSub__ __IsWindows__ __Line__ __Module__ __Scope__ __Sub__ Abs
Access Acs AddHandler AppActivate Append Array Asc Asn Atn Beep Bin
Bin$ Binary BLOAD BSAVE CBCD CBool CByte CChar CCur CDate CDbl CDec
CEXT CFIX ChDir ChDrive Chr Chr$ CInt Circle Clear CLng Close CLS
CObj Color Command Command$ Cos CQUD CreateObject CShort CSng CsrLin
CType CurDir CurDir$ CVD CVDMBF CVERR CVI CVL CVS CVSMBF Date Date$
DateAdd DateDiff DatePart DateSerial DateValue Day DDB Deg
DeleteSetting Dir Dir$ DoEvents DOF Draw Environ Environ$ EOF ErDev
ErDev$ Erl Err Error Error$ Exp Fact Field FileAttr FileCopy
FileDateTime FileLen Files Filter Fix FN Format Format$
FormatCurrency FormatDateTime FormatNumber FormatPercent Frac FRE
FreeFile FV Get GetAllSettings GetAttr GetAutoServerSettings
GetObject GetSetting GetType Hex Hex$ Hour Hypot IMEStatus Inkey
Inkey$ Inp Input Input$ InputBox InStr InStRev Int IOCtl IOCtl$ IPMT
IRR IsArray IsBoolean IsByte IsCharacter IsCollection IsCString
IsCurrency IsDate IsDouble IsEmpty IsError IsInt16 IsInt32 IsInt64
IsInteger IsMissing IsNull IsNumeric IsObject IsShort IsSingle
IsUInt16 IsUInt32 IsUInt64 IsLong IsString IsVariant Join Kill LCase
LCase$ Left Left$ Len Line Ln Load LoadPicture LoadResData
LoadResPicture LoadResString Loc Locate Lock LOF Log Logb LPos
LPrint LTrim LTrim$ Max Mid$ Min Minute MIRR MKD$ MkDir MKDMBF$ MKI$
```

```
MKL$ MKS MKS$ MKSMBF$ Month MonthName MsgBox MTIMER Name Now NPER
NPV Nz Oct Oct$ Open Out Output Paint Palette Partition PCopy Peek
PMAP PMT Point Poke Pos PPMT Preset Print PSet Put PV QBCOLOR Rad
Raise RaiseEvent RaiseSignal Random Randomize Rate RemoveHandler
Replace Reset RGB Right Right$ RmDir RND Round RTrim RTrim$
SavePicture SaveSetting Screen Sec Second Seek Seg SendKeys SetAttr
Sgn Shell Sin Sleep Sln Sound Space Space$ Spc Split Sqr Stick Str
Str$ StrComp StrConv String String$ StrReverse SYD Tab Tan Time
Time$ TimeSerial TimeValue Trim Trim$ TypeName TypeOf UCase UCase$
UnLoad UnLock Using Val VarType View Weekday WeekdayName Width
Window Write Year
```

Operatori

```
<< >> Inc Dec += -= /= *= BitAnd BitOr BitXor BitNot + - * / Mod =
<> >= <= > < And AndAlso Or OrElse Not ^ & Xor \ Eqv Imp
```

Tipi

```
Boolean Byte Character Collection CString Currency Date Double Int16
Int32 Int64 Integer Long Object Short Single String UInt16 UInt32
UInt64 Variant
```

Codici ASCII (codici 0 - 127)

000		(nul)	016	?	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	?	(soh)	017	?	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	?	(stx)	018	?	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	?	(etx)	019	?	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	?	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	?	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	?	(ack)	022	?	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	?	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	?	(bs)	024	?	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	?	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	?	(vt)	027	?	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	?	(np)	028	?	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	?	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	?	(so)	030	?	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	¤	(si)	031	?	(us)	047	/	063	?	079	O	095	_	111	o	127	¦

Argomenti di comando KBasic

DA FARE #7

Appendice

Argomento
Un valore che viene passato ad una procedura o ad una funzione. Vedi anche *Parametro*.

Array
Una variabile che memorizza una serie di valori a cui si può accedere usando un indice.

BASIC
Beginner's All purpose Symbolic Instruction Code *(*Codice di istruzioni simboliche di uso generale per principianti, *ndt*), il linguaggio di programmazione su cui si basa KBasic.

Bit
Il più piccolo elemento d'informazione che il computer può contenere. Un bit può essere solo uno di due valori, 0 oppure 1.

Booleano
Un tipo di dato che rappresenta un valore vero o falso.

Branch Condizionale (lett. ramificazione condizionale, *ndt*)
Quando l'esecuzione di un programma salta ad un'altra locazione basata su una qualche sorta di condizione. Un esempio di branch condizionale è un'istruzione If/Then, che fa spostare il programma ad una riga che si basa su una condizione quale If (x=5).

Branch incondizionato
Quando l'esecuzione di un programma salta ad un'altra locazione incurante di ogni condizione. Un esempio di branch incondizionato è l'istruzione GoTo.

Branching (ramificazione, *ndt*)
Quando l'esecuzione di un programma salta da un punto ad un altro, piuttosto che continuare ad eseguire le istruzioni in stretto ordine. Vedi anche *Branch condizionale* e *Branch incondizionato*.

Breakpoint (lett. punto d'interruzione, *ndt*)
Un punto del programma, in cui si deve sospendere l'esecuzione per consentire al programmatore di esaminare i risultati fino a quel punto. Impostando dei breakpoint in un programma, puoi trovare più facilmente gli errori. Vedi anche *Debugging*.

Byte
Un dato composto di otto bit. Vedi anche *Bit*.

Ciclo
Un blocco di codice che viene eseguito ripetutamente fino a che non s'incontra una certa condizione.

Ciclo infinito
Un ciclo che non può terminare perché la sua espressione condizionale non può essere mai valutata come vera. Un loop infinito ha fine solo quando l'utente termina il programma. Vedi anche *Ciclo* e *Variabile di controllo ciclo*.

Codice Sorgente
Le righe dei comandi che compongono un programma.

Compilatore
Uno strumento di programmazione che converte il codice sorgente di un programma in un file eseguibile. KBasic impiega automaticamente il compilatore quando esegui un programma, o selezioni il comando *Make Binary* dal menu *Run* per convertire il sorgente in un eseguibile (solo per la versione Professional). Vedi anche *Interprete*.

Concatenare
Unire due stringhe di testo in una. Per esempio, la stringa “OneTwo” è la concatenazione di due stringhe: “One” e “Two”.

Costante
Un valore predefinito che non cambia mai.

Debugging
L'azione di trovare ed eliminare errori in un programma.

Decremento
Diminuire il valore di una variabile, di solito di 1. Vedi anche *Incremento*.

Double
Tipo di dato che rappresenta il più accurato valore in virgola mobile, anche noto come valore in virgola mobile a doppia precisione. Vedi anche *Virgola Mobile* e *Single*.

Espressione booleana
Una combinazione di termini che viene valutata come vera o falsa. Per esempio, (x = 5) e (y = 6).

Errore logico
Un errore che si verifica quando un programma esegue un compito differente da quello per cui il programmatore pensava di averlo programmato. Per esempio, la riga If X = 5 Then Y = 6 è un errore logico se la variabile X non può essere mai uguale a 5. Vedi anche *Errore di run-time*.

Errore di run-time
Un errore di sistema che si verifica quando un programma è in esecuzione. Un esempio è un errore di divisione per zero o un errore di accoppiamento (*type-mismatch*, *ndt*). Senza un qualche tipo di trattamento degli errori, come quello fornito dall'istruzione Try/Catch, gli errori di run-time spesso risultano in un crash del programma.

Espressioni matematiche
Un insieme di termini che usano operatori aritmetici per produrre un valore numerico. Ad esempio, i termini (X + 17) / (Y + 22) compongono un'espressione matematica. Vedi anche *Operazioni aritmetiche*.

Evento
Un messaggio che viene inviato ad un programma come risultato di una qualche interazione tra l'utente e il programma.

File (lett. archivio, *ndt*)
Il nome di un insieme di dati, in un disco.

File eseguibile
Un file, solitamente un'applicazione, che il computer può caricare ed eseguire. La maggior parte dei file eseguibili terminano con l'estensione EXE in Windows.

Floating point
Vedi *Virgola mobile*

Flusso del programma
L'ordine in cui un computer esegue le istruzioni del programma.

Form
L'oggetto KBasic che rappresenta la finestra di un'applicazione. Il form è il contenitore in cui posizioni i controlli che costituiscono l'interfaccia utente del programma.

Funzione
Un sottoprogramma che elabora dati in un certo modo e restituisce un singolo valore che rappresenta il risultato dell'elaborazione.

Incremento
Aumentare il valore di una variabile, di solito di 1. Vedi anche *Decremento*.

Inizializzazione
Impostare il valore iniziale di una variabile.

Integer
Vedi *Intero.*

Intero
Un tipo di dato che rappresenta un valore numerico tra – 2.147.483.648 e 2.147.483.647. I valori 240, –128, e 2 sono esempi di interi. Vedi anche *Long*.

Interfaccia utente
La rappresentazione visibile di un programma, di solito composta di vari tipi di controlli, che mette in grado l'utente di interagire con l'applicazione.

Interprete
Uno strumento di programmazione che esegue il codice sorgente una riga alla volta, a differenza di un compilatore, che converte un intero programma in un file eseguibile prima di eseguire qualsiasi comando di esso. Vedi anche *Compilatore*.

Linguaggio di programmazione
Un insieme di parole chiave simili all'inglese e simboli, che mettono in grado un programmatore di scrivere un programma senza usare il linguaggio macchina.

Linguaggio macchina
L'unico linguaggio che in verità il computer comprende. Tutto il codice sorgente del programma deve essere convertito in linguaggio macchina prima che il computer possa eseguirlo.

Literal
Un valore dichiarato alla lettera, cioè che non viene memorizzato in una variabile.

Literal numerico
Un valore literal che rappresenta un numero, come 125 o 34,87. Vedi anche *Literal* e *Literal stringa*.

Literal stringa
Uno o più caratteri racchiusi tra virgolette.

Loop
Vedi *Ciclo.*

Long
Un tipo di dato che rappresenta un valore intero da -2^{32} - 1 a $+2^{32}$. Vedi anche *Intero*.

Metodo
Una procedura associata ad un oggetto o controllo che rappresenta una capacità dell'oggetto o controllo. Ad esempio, il metodo *Move* del pulsante di comando riposiziona il pulsante sul form.

Oggetto
In generale, qualsiasi pezzo di dati in un programma. Specificatamente in KBasic, un insieme di proprietà e metodi che rappresentano una sorta di oggetto del mondo reale o un'idea astratta.

Operatore logico
Un simbolo che confronta due espressioni, che risulta in un valore booleano (vero o falso). Per esempio, nella riga If X = 5 and Y = 10 Then Z = 1, And è l'operatore logico. Vedi anche *Operatore relazionale*.

Operatore relazionale
Un simbolo che determina la relazione tra due espressioni. Per esempio, nell'espressione X > 10, l'operatore relazionale è >, che vuol dire “più grande di”. Vedi anche *Operatore logico*.

Operazioni aritmetiche
Operazioni matematiche come addizione, moltiplicazione, sottrazione, e divisione, che producono risultati numerici.

Ordine delle operazioni
L'ordine in cui KBasic risolve operazioni aritmetiche. Per esempio, nell'espressione matematica (X + 5) / (Y + 2), KBasic eseguirà le due addizioni prima della divisione. Se sono state tralasciate le parentesi, come nell'espressione X + 5 / Y + 2, KBasic divide prima 5 con Y e poi esegue la restante operazione di addizione.

Parametro
Spesso significa la stessa cosa di “argomento”, sebbene alcuni distinguono argomento e parametro, dove un argomento è il valore passato alla procedura o funzione, e un parametro è la variabile nella funzione o procedura che riceve l'argomento. Vedi anche *Argomento*.

Procedura
Un sottoprogramma che esegue un compito, ma non restituisce un valore. Vedi anche *Funzione.*

Programma
Un elenco d'istruzioni per un computer.

Property
Vedi *Proprietà*.

Proprietà
Un valore che rappresenta un attributo di un oggetto o controllo.

Proprietà di sola lettura
Una proprietà il cui valore non può essere modificato.
Ad ogni modo, un programma può ricuperare (leggere) il valore di una proprietà.

Scope
Vedi *Visibilità di una variabile*.

Single
Il tipo di dato che rappresenta il valore in virgola mobile meno accurato, anche noto come valore in virgola mobile a singola precisione. Vedi anche *Double* e *Virgola Mobile*.

Sottoprogramma
Un blocco di codice che esegue una specifica parte di un compito più grande. In KBasic un sottoprogramma può essere sia una sottoprocedura che una funzione. Vedi anche *Funzione* e *Procedura*.

String
Vedi *Stringa*

Stringa
Un tipo di dato che rappresenta uno o più caratteri di testo. Ad esempio, nell'istruzione di assegnamento str1 = “I’m a string.”, la variabile str1 deve essere del tipo String (o Variant).

Stringa vuota
Una stringa che ha lunghezza 0, denotata da due doppie virgolette. Il seguente esempio imposta una variabile chiamata str1 a una stringa vuota: str1 = “”.

Tipo di dato
I vari tipi di valori che un programma può memorizzare. Questi includono *Integer*, *Long*, *String*, *Single*, *Double*, e *Boolean*.

Valore numerico

Un valore che rappresenta un numero. Questo valore può essere un literal, una variabile, o il risultato di un'*operazione aritmetica*.

Valore di ritorno

Il valore che una funzione restituisce all'istruzione che la chiamata. Vedi anche *Funzione*.

Virgola mobile

Un valore numerico che ha una porzione decimale. Ad esempio, 12,75 e 235,7584 sono due valori in virgola mobile. Vedi anche Double e Single.

Variabile

Un nome di un valore in un programma. Può essere assegnato e riassegnato, un valore del tipo di dato appropriato.

Variabile di controllo ciclo

Una variabile che contiene il valore che determina se un loop continuerà l'esecuzione.

Variabile globale

Un valore con nome a cui si può accedere da qualsiasi parte all'interno di un programma.

Variabile locale

Una variabile a cui si può accedere solo da dentro il sottoprogramma in cui è dichiarata. Vedi anche *Variabile Globale* e *Visibilità di una variabile*.

Visibilità di una variabile

L'area di un programma in cui si può accedere ad una variabile. Per esempio,
La visibilità di una *variabile globale* è dappertutto all'interno del programma, laddove la visibilità di una *variabile locale* è limitata alla procedura o funzione in cui è dichiarata. Vedi anche *Variabile globale* e *Variabile locale*.

Variant

Un tipo speciale che rappresenta ogni tipo di dato. Più precisamente,
KBasic gestisce per te il valore di un tipo di dato Variant. Questo tipo di dato è molto inefficiente e dovrebbe essere evitato quando possibile.

Contatti/Sigla editoriale

www.kbasic.com
email: info@kbasic.com

Qualità di Bernd Noetscher
Made in Germany (Unione Europea)

Marchi