GAMBAS (autore: Vittorio Albertoni)

Premessa

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

Da questo linguaggio, che originariamente aveva a che fare con un mainframe, una decina di anni dopo Bill Gates e Paul Allen derivarono un linguaggio per programmare uno dei primi computer da tavolo, il MITS Altair 8800 che venne chiamato Altair BASIC, immediatamente divenuto il primo prodotto della neo-costituita Microsoft con il nome di Microsoft BASIC, proprietà intellettuale della Microsoft e origine della fortuna di questa.

I primi adattamenti del linguaggio, sviluppati dalla Commodore (per VIC 20 e Commodore 64) e dalla Apple avvengono su licenza Microsoft.

La stessa Microsoft sviluppa poi tutta una serie di adattamenti a partire dal lancio del PC IBM 5150 nel 1981, e, di perfezionamento in perfezionamento, si arriva al 1991 con il rilascio di Visual BASIC, un ambiente di sviluppo con cui, utilizzando il linguaggio BASIC è possibile programmare applicazioni con interfaccia utente grafica, e, con il rilascio nel 1998 della versione 6, Visual BASIC diventa il cavallo di battaglia della Microsoft per la programmazione dei personal computer.

Sull'onda del successo di Visual BASIC 6 per il sistema operativo di casa Microsoft (nel frattempo diventato Windows), nel 1999, l'informatico francese Benoît Minisini avvia un progetto per realizzare un prodotto simile dedicato al sistema operativo Linux, progetto che consente, il 28 febbraio 2002, di effettuare un primo rilascio di GAMBAS, acronimo di Gambas Almost Means BASIC, cioè Gambas più o meno significa BASIC.

Come tutto il software libero GAMBAS si è rapidamente arricchito e perfezionato e la versione 3, rilasciata il 31 dicembre 2011, ha raggiunto la perfezione. Nel momento in cui scrivo (giugno 2025) è in distribuzione dal 20 maggio 2025 la versione 3.20.4.

Come Visual BASIC della Microsoft, GAMBAS è un ambiente di sviluppo visuale, dotato di strumenti per creare una interfaccia utente grafica (GUI).

GAMBAS, come da sempre sottolinea il suo creatore Benoît Minisini, non è un clone di Visual BASIC ma fa le stesse cose diversamente: a partire dai framework utilizzati per la grafica fino allo stesso linguaggio, che è un dialetto del linguaggio BASIC non del tutto compatibile con quello utilizzato da Microsoft.

Sempre nel mondo del software libero ricordo che esiste anche una versione libera del linguaggio BASIC puro e semplice, cioè non inquadrato in un ambiente di programmazione, che si chiama FreeBASIC [1] . Si tratta di un dialetto BASIC molto simile ma non identico a quello utilizzato in ambiente GAMBAS.

[1] In proposito segnalo il manualetto «freebasic» allegato all'articolo «Intramontabile BASIC» pubblicato nel febbraio 2020 sul mio blog all'indirizzo *www.vittal.it*.

Indice

1 Installazione **3**

2 Come funziona: il primo programma **3**

3 Il BASIC di GAMBAS **6**
3.1 Tipi 6
3.1.1 Tipi numerici 6
3.1.2 Tipo stringa 6
3.1.3 Tipo booleano 6
3.2 Variabili e costanti 6
3.2.1 Variabili 7
3.2.2 Costanti 7
3.3 Operatori 7
3.3.1 Operatori aritmetici 7
3.3.2 Operatori di confronto 8
3.3.3 Operatori logici 8
3.4 Subroutines e funzioni 8
3.5 Funzioni precostituite 9
3.6 Interattività con l'utente 10
3.7 Strutture di controllo 10
3.7.1 Esecuzione condizionale 10
3.7.2 Ripetizione 12
3.8 Classi e oggetti 13
3.9 Lavorare con i file 13
3.9.1 Creazione e scrittura di un file 13
3.9.2 Lettura di un file 14
3.9.3 Aggiunta di dati ad un file 14

4 Applicazioni con GUI **14**

1 Installazione

GAMBAS è software esclusivo per il sistema operativo Linux.

I suoi rilasci avvengono su *https://gambas.sourceforge.net/en/main.html*.

Per avere un ambiente perfettamente funzionante sulla distro che utilizziamo è bene installare GAMBAS ricorrendo al gestore dei pacchetti della distro stessa.

Il pacchetto da installare per trascinare l'installazione di tutti i pacchetti che servono è quello denominato **gambas3** nel repository Debian e derivate (installatore `apt`) e **gambas3-ide** nel repository Red Hat e derivate (installatore `dnf`).

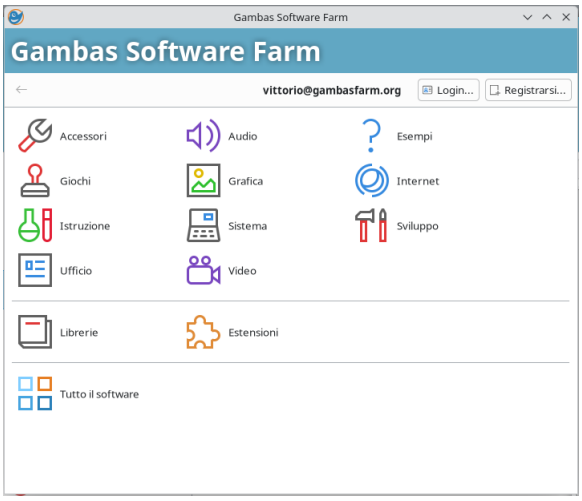
Se l'applicazione non parte cliccando sull'icona nel menu, aprire il terminale e digitare il comando `gambas3`, in modo da ottenere indicazioni sul motivo della mancata partenza.
Sicuramente si tratta della mancata installazione di uno o più file secondari, che, utilizzando le indicazioni, possiamo ricercare e installare con strumenti come `synaptic` o `dnfdragora`, rispettivamente per i repository Debian o Red Hat.

2 Come funziona: il primo programma

L'ambiente di sviluppo GAMBAS, al lancio, si presenta così

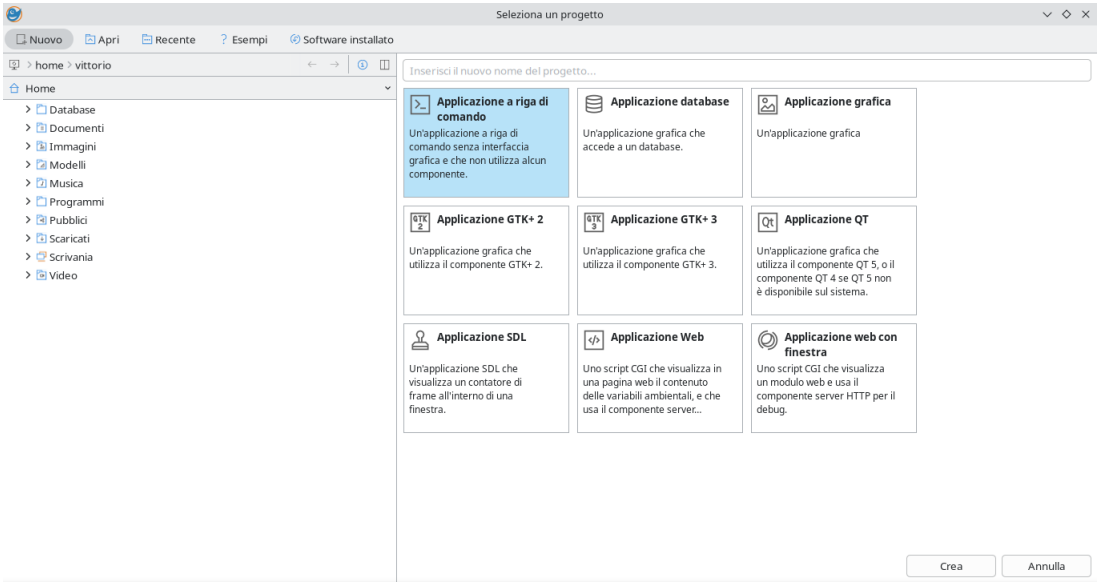
Nella finestra di benvenuto che compare in primo piano, sulla sinistra, abbiamo un menu che presenta una voce molto interessante: SOFTWARE FARM…

Cliccando su questa voce apriamo la seguente finestra



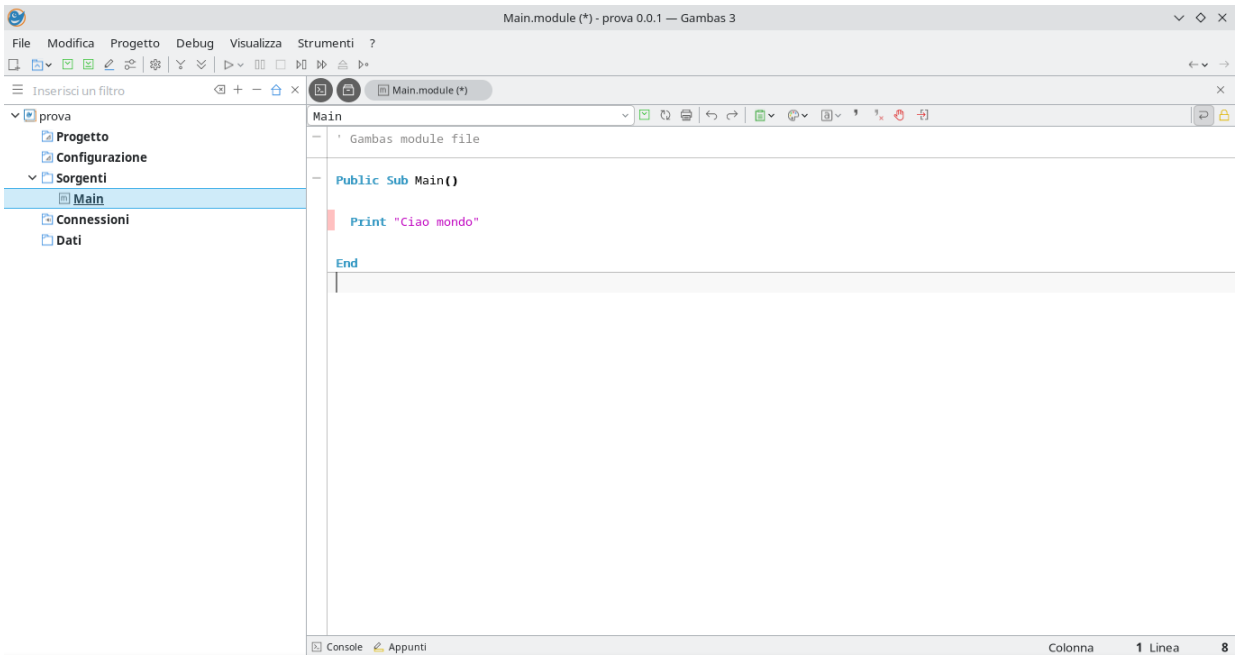
Per utilizzarne il contenuto dobbiamo preventivamente registrarci scegliendo nome utente e password.
Le icone rappresentano varie categorie di software già sviluppato con GAMBAS che possiamo scaricare come esempio o installare per essere utilizzato.

Per creare un nuovo progetto dobbiamo cliccare sulla voce NUOVO PROGETTO... e agire nella seguente finestra



in cui ci vengono proposti vari tipi di progetto.

La cosa più semplice che possiamo fare è quella di creare un'applicazione a riga di comando, cioè senza GUI e, a questo scopo, clicchiamo sulla prima scelta, quella selezionata nella figura, e dopo aver dato un nome al progetto e scelto dove salvarlo utilizzando apposite finestre di dialogo, con doppio click su MAIN si apre questa finestra



nella quale è già scritto il codice per il classico primo programma che presentano i manuali sui linguaggi di programmazione da quando il creatore del linguaggio C, Dennis Ritchie, ha inaugurato questa usanza nel suo manuale sul linguaggio stesso: il programmino Ciao mondo.

Una volta scritto il codice possiamo innanzi tutto vedere se funziona.

A questo scopo apriamo la finestrella della console nella parte inferiore della zona di editing cliccando sulla linguetta CONSOLE in fondo e da menu DEBUG ▷ ESEGUI vediamo cosa succede nella console: se qui compaiono i risultati attesi tutto funziona e, in caso contrario, ci verrà detto che cosa non va in modo che possiamo rimediare.

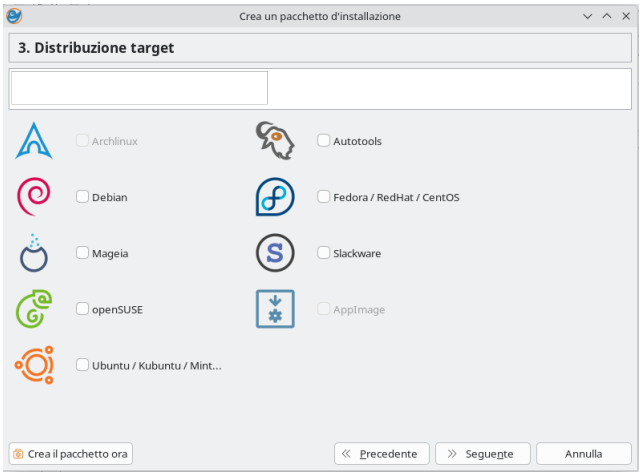
Se tutto funziona possiamo creare l'eseguibile da menu PROGETTO ▷ CREA ESEGUIBILE facendo in modo che esso venga salvato nella stessa directory del progetto.

Chiamare «eseguibile» il file così creato è improprio, in quanto esso, in realtà, non è un codice macchina come quello che generiamo compilando programmi scritti in C o in Pascal, ma è un bytecode, tipo quello che si genera compilando un programma scritto in Java e viene fatto girare da un interprete, che, in GAMBAS3 si chiama `gbr3`, che fa parte dell'ambiente di sviluppo integrato GAMBAS. Ciò significa che possiamo eseguire questo «eseguibile» solo su un computer su cui sia installato GAMBAS.

Il lavoro di programmazione che abbiamo fatto si salva nella directory che abbiamo scelto all'apertura del progetto in una serie di file nascosti che, insieme, compongono il source code. Possiamo creare un archivio compattato di questi file da menu PROGETTO ▷ CREA IL PACCHETTO SORGENTI.., generando così un archivio `.tar.gz` contenente il tutto.

Per far si che il nostro programma possa essere utilizzato su altri computer dove non sia installato GAMBAS dobbiamo creare un pacchetto di installazione. Anzi, dal momento che le distribuzioni Linux sono diverse e, a parte il kernel, hanno ciascuna le proprie peculiarità, dobbiamo creare tanti pacchetti di installazione quante sono queste distribuzioni.

Da menu PROGETTO ▷ CREA UN PACCHETTO DI INSTALLAZIONE..., dopo alcune finestre di dialogo preparatorie arriviamo a questa



dove possiamo scegliere la o le distribuzioni obiettivo.
A scelta effettuata abbiamo altre finestre di dialogo da compilare fino alla N. 9 DIRECTORY DI DESTINAZIONE nella quale dobbiamo indicare una directory in cui salvare il o i pacchetti di installazione che andiamo a creare (directory che deve essere diversa da quella in cui abbiamo salvato il progetto). Qui, per non creare un gran casino soprattutto nel caso dobbiamo creare più pacchetti, dobbiamo avere l'avvertenza di selezionare l'opzione CREA LE DIRECTORY PER OGNI DISTRIBUZIONE.
Se sul computer di destinazione dell'installazione non è installato GAMBAS può capitare di dover sistemare qualche dipendenza (mancanza del file `gambas3-gb-gui` o, per computer anziani, mancanza del file `gambas3-gb-gtk3-x11`, ecc.). Questi inconvenienti vengono segnalati tentando di far partire il programma installato dal terminale. Possiamo così rimediare ricorrendo a `synaptic` o `dnfdragora`.

Su GAMBAS non si può comunque fare alcun affidamento per creare software da far circolare. Veder funzionare un'applicazione prodotta sul nostro computer su un altro è come vincere un terno al lotto.

Se programmiamo per esigenze nostre da soddisfare sul nostro computer abbiamo il vantaggio di poter fare cose anche non banali con un linguaggio di programmazione e un costruttore di GUI che più facile non si può.

3 Il BASIC di GAMBAS

Come tutti i linguaggi appartenenti alla famiglia del BASIC è un linguaggio procedurale ma anche orientato alla programmazione per oggetti.

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

La tipizzazione è statica, cioè richiede che il tipo di ciascun valore venga stabilito nel codice sorgente.

Ciò comporta che le variabili vengano dichiarate e tipizzate prima di poter essere utilizzate.

La scrittura del codice non prevede regole particolari e possiamo battere sulla tastiera ignorando la differenza tra maiuscolo e minuscolo: ci pensa l'IDE a creare uno stile per ciò che scriviamo.

I commenti si scrivono preceduti dal simbolo '.

La code completion che ci fornisce l'IDE è formidabile, così come la descrizione del significato della varie parole chiave che compare selezionandole.

3.1 Tipi

3.1.1 Tipi numerici

Intero

Per i numeri interi, quelli che non hanno decimali, sono previste le seguenti alternative:
. `Byte` con ampiezza di 8 bit senza segno (valori tra 0 e 255),
. `Short` con ampiezza di 16 bit con segno (valori tra -32.768 e 32.767),
. `Integer` con ampiezza di 32 bit con segno (valori tra -2.147.483.648 e 2.147.483.647),
. `Long` con ampiezza di 64 bit con segno (valori tra -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807).

Decimale

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

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

Precisione singola significa avere 8 cifre significative su 39 cifre, tra parte intera e parte decimale, precisione doppia significa averne 17, virgola compresa, su 308 cifre.

3.1.2 Tipo stringa

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

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

3.1.3 Tipo booleano

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

3.2 Variabili e costanti

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

Nel primo programma visto nel Capitolo 2 abbiamo usato l'istruzione `Print` per visualizzare il saluto Ciao mondo.
In quel caso abbiamo indicato all'istruzione una stringa da visualizzare.

Nello stesso modo potremmo indicare un numero da stampare o anche un'espressione numerica, scritta utilizzando gli operatori che vedremo, per visualizzarne il risultato.
Il tutto utilizzando elementi volanti.

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

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

3.2.1 Variabili

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

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

La sintassi per la dichiarazione di una variabile è:
`Dim` <nome> `As` <tipo>

Per dichiarare più variabili dello stesso tipo possiamo scrivere
`Dim` <nome>, <nome>, ... `As` <tipo>

Nel caso si voglia anche inizializzare la variabile, cioè assegnarle un valore iniziale:
`Dim` <nome> `As` <tipo> = <valore>.

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

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

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

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

```
Dim saluto As String = "Ciao mondo"
Print saluto
```

Come si vede, abbiamo innanzi tutto creato una variabile stringa che contiene la frase Ciao mondo e poi abbiamo detto a `Print` di stampare quella variabile richiamandone il nome.
Il risultato è lo stesso che si ottiene con la scrittura del programma senza l'uso della variabile, ma ora abbiamo realizzato il vantaggio di avere nel programma una variabile utilizzabile in altre parti del programma semplicemente richiamandone il nome.

3.2.2 Costanti

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

La sintassi per la dichiarazione di una costante è
`Const` <nome> `As` `<tipo>` = <valore>
e la dichiarazione va fatto al di fuori dalle routine,

3.3 Operatori

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

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

3.3.1 Operatori aritmetici

+ per la somma tra numeri (per la concatenazione di stringhe `&`)
- per la sottrazione tra numeri
* per la moltiplicazione tra numeri
/ per la divisione tra numeri

\ per la divisione intera
`Mod` per il modulo (resto della divisione intera)
^ per l'elevamento a potenza

3.3.2 Operatori di confronto

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

Sono i seguenti.
= uguale,
<> non uguale,
< minore,
<= minore o uguale,
> maggiore,
>= maggiore o uguale.

Gli operandi assoggettati al confronto devono avere lo stesso tipo.

3.3.3 Operatori logici

Forniscono come risultato un valore booleano e sono i seguenti.
`And`,
`Or`,
`Not`.

3.4 Subroutines e funzioni

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

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

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

Subroutines

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

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

La subroutine deve essere costruita non necessariamente prima di richiamarla.

Funzioni

La sintassi per costruire una funzione è
`Function` <identificativo> `(`<argomenti>`)` `As` <tipo_dato_restituito>
<istruzioni>
`End`

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

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

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

```
Function area_triangolo (b As Float, h As Float) As Float
return b * h / 2
End
```

Per calcolare e scrivere l'area di un triangolo di base 5 e altezza 4 diamo l'istruzione

```
Print area_triangolo(5, 4)
```

ed otteniamo scritto il risultato 10.

Programma

Il programma è contenuto in una subroutine, chiamata `main`, l'ultima che si scrive dopo aver eventualmente scritto le subroutine e le funzioni che costituiscono il programma e la prima che viene eseguita al lancio del programma stesso.

In programmi semplici, come il Ciao mondo che abbiamo visto, può essere anche la sola.

Essa è definita di accesso pubblico con la parola chiave `Public` e la sintassi

```
Public Sub main()
    <istruzioni>
End
```

dove <istruzioni> possono contenere richiami di subroutine o funzioni precedentemente create, funzioni precostituite o istruzioni pure e semplici come una espressione aritmetica.

3.5 Funzioni precostituite

Il BASIC di GAMBAS contiene una miriade di funzioni già predisposte per fare determinate cose.

Per averne un'idea basta consultare la ricca documentazione cui accediamo da menu
? ▷ NAVIGAZIONE HELP, in particolare, se ci interessa fare di conto, cliccando poi su
FUNZIONI ARITMETICHE
FUNZIONI LOGARITMI ED ESPONENZIALI
FUNZIONI NUMERICHE CASUALI
FUNZIONI STRINGA
FUNZIONI TRIGONOMETRICHE

Tra le funzioni precostituite ricordo quelle fondamentali per instaurare una interattività con l'utente:

`Print` scrive su uno stream e già la conosciamo,
gli argomenti per questa funzione possono essere valori scalari, variabili, espressioni.

`Input` legge una stringa singola da uno stream e la converte in valore,
seguito dal nome di una variabile inserisce il valore letto nella variabile secondo il tipo dichiarato per questa.

`Line Input` legge testo da uno stream,
seguito dal nome di una variabile inserisce il testo letto nella variabile stessa.

3.6 Interattività con l'utente

Nel piccolo esempio «ciao mondo» che abbiamo visto nel Capitolo 2 e nella sua rivisitazione nel Paragrafo 3.2.1 il programma conteneva già il dato da elaborare (la scritta "Ciao mondo" nel primo caso direttamente indicata e, nel secondo caso, indicata richiamando la variabile che la conteneva) e, una volta lanciato, forniva il risultato secondo quella scritta. Se ci accontentassimo di questo, per salutare qualche cosa che non sia il mondo ma sia un nostro amico, dovremmo prendere il programma che abbiamo scritto per salutare il mondo, ricopiarlo sostituendo alla parola "mondo" il nome del nostro amico, ricompilarlo e rilanciarlo. Così dovremmo fare per tutti gli amici che volessimo salutare.

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

Il programma diventa il seguente

```
Dim nome As String
Print "Come ti chiami?  "
Line Input nome
Print "Ciao, " & nome & "!"
```

Lanciandolo saremo richiesti di indicare il nome di chi salutare e in un attimo vedremo il saluto rivolto a chi abbiamo indicato. Ho utilizzato la funzione `Line Input` per poter leggere un doppio nome o nome e cognome (la funzione `Input` leggerebbe solo la prima parola digitata).

3.7 Strutture di controllo

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

3.7.1 Esecuzione condizionale

If

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

La sintassi è la seguente

```
If <condizione> Then
   <istruzioni>
Endif
```

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

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

```
If <condizione> Then
   <istruzioni>
Else
   <istruzioni>
Endif
```

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

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

```
If <condizione> Then
   <istruzioni>
Else If <condizione> Then
   <istruzioni>
Else If <condizione> Then
   <istruzioni>
Else If <condizione> Then
   <istruzioni>
.....
```

e proseguire quanto vogliamo, chiudendo alla fine con `Endif`.

Select

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

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

La sintassi è la seguente

```
Select Case <variabile>
   Case <valore>
       <istruzione>
   Case <valore>
       <istruzione>
   Case <valore>
       <istruzione>
   .......
   Case Else
       <istruzione>
End Select
```

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

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

```
Dim moneta As String
Print "Scegli una moneta:  "
Input moneta
Select Case moneta
    Case "euro"
        Print "Europa"
    Case "dollaro"
        Print "U.S.A."
    Case "sterlina"
        Print "Regno Unito"
    Case "yen"
        Print "Giappone"
    Case "yuan"
        Print "Cina"
    Case Else
        Print "non so"
End Select
```

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

3.7.2 Ripetizione

For

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

La sintassi per l'uso di questa istruzione è

```
Dim <contatore> As <tipo>
For <contatore> = <partenza> To <arrivo> [Step <numero>]
    <istruzioni>
Next
```

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

Questo piccolo programma

```
Dim i As Byte
For i = 1 To 5
    Print "Ciao"
Next
```

scrive cinque volte, su righe diverse, la parola Ciao.

While

Si usa per ripetere istruzioni fino a quando una condizione è vera.

La sintassi è:

```
While <condizione>
    <istruzioni>
Wend
```

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

```
Dim i As Byte
While i <= 5
   print "Ciao"
   i = i + 1
Wend
```

scrive cinque volte la parola Ciao.

Ma sono possibili altri meccanismi.

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

```
Dim s As String
Print "premi un tasto"
Input s
While s <> "f"
    Print s
    Print "premi un tasto"
    Input s
Wend
```

Do

La sintassi è:
`Do While|Until` <condizione>
    <istruzioni>
`Loop`
Le parole chiave `While` e `Until` sono di uso alternativo.

Serve per ripetere istruzioni fino a quando una condizione è vera (`While`) o fino a quando una condizione è falsa (`Until`).

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

```
Dim s As String
Print "premi un tasto"
Input s
Do While s <> "f"
    Print s
    Print "premi un tasto"
    Input s
Loop
```

Un esempio con `Until` può essere il seguente, per scrivere 5 volte la parola Ciao:

```
Dim i As Byte
Do Until i > 4
   Print "Ciao"
   i = i + 1
Loop
```

3.8 Classi e oggetti

Se stiamo nel mondo del software libero, quanto visto finora praticamente coincide, con qualche piccolissima differenza sintattica, con ciò che troviamo in FreeBASIC.

Anche il supporto alla programmazione a oggetti, presente in maniera direi alquanto bizzarra in FreeBASIC è presente in maniera più abbordabile in GAMBAS. Ritengo, tuttavia, che programmare a oggetti con questi linguaggi non sia all'altezza di un principiante e pertanto, come ho fatto nel mio manualetto su FreeBASIC, non ne parlo in questo manualetto introduttivo.

A questo punto, se teniamo conto di quanto abbiamo visto nel Capitolo 2 circa la compilazione dei programmi con GAMBAS, visto che FreeBASIC genera eseguibili veri e propri, viene da chiedersi perché utilizzare GAMBAS.

Effettivamente la vera utilità di GAMBAS si manifesta in ciò che vedremo da qui in poi, nella creazione di applicazioni dotate di interfaccia grafica per l'utente (GUI). E qui useremo abbondantemente oggetti "prefabbricati" di GAMBAS.

3.9 Lavorare con i file

Anche nel programmare applicazioni console ci possiamo imbattere in oggetti "prefabbricati" di GAMBAS: è il caso dell'oggetto `File`, che ci consente di lavorare facilmente con file.

L'oggetto `File` ha due metodi, Save e Load che ci consentono, rispettivamente, di inserire e di estrarre dati, sotto forma di stringhe, da un file.

3.9.1 Creazione e scrittura di un file

Dobbiamo innanzi tutto disporre di una stringa contenente dati.

Con
`File.Save(`<path_al_file>`,` <stringa_dati>`)`
dove

<path_al_file> è una stringa contenente il nome del file con il relativo percorso,
<stringa_dati> è la stringa contenente i dati da inserire nel file
creiamo il file nel percorso specificato e vi inseriamo i dati contenuti nella stringa.
Se il file esiste viene sovrascritto.

3.9.2 Lettura di un file

Con
```
Dim s As String
s = File.Load(<path_al_file>)
```
dove
<path_al_file> è una stringa contenente il nome del file con il relativo percorso,
leggiamo il file nel percorso specificato e inseriamo i dati letti nella stringa `s`.

3.9.3 Aggiunta di dati ad un file

Per aggiungere dati ad un file esistente dobbiamo leggere i dati già inseriti nel file e reinserire la relativa stringa nel file concatenandola con la stringa dei dati da aggiungere:
```
Dim s As String
s = File.Save(<path_al_file>, File.Load<path_al_file> & <stringa_dati>)
```
dove <stringa_dati> è la stringa contenente i dati da aggiungere e, per default, viene aggiunta sulla stessa riga che era stata scritta per ultima nel file, senza spazi intermedi.
Per aggiungere spazi intermedi occorre inserirli all'inizio della stringa dei dati da aggiungere.
Per aggiungere i data su una nuova riga nel file occorre inserire all'inizio della stringa dei dati da aggiungere i caratteri `\n`.

Il tutto è esemplificato in questo programmino
```
Public Sub Main()
    Dim s As String
    s = "Vittorio Albertoni"
    File.Save("/home/vittorio/Documenti/mio_file", s)
    Print "    contenuto del file originario:"
    Print File.Load("/home/vittorio/Documenti/mio_file")
    File.Save("/home/vittorio/Documenti/mio_file",
                File.Load("/home/vittorio/Documenti/mio_file") & "\nMaria Panzeri")
    Print "    contenuto del file aggiornato:"
    Print File.Load("/home/vittorio/Documenti/mio_file")
End
```
il cui output è il seguente:
```
   contenuto del file originario:
Vittorio Albertoni
   contenuto del file aggiornato:
Vittorio Albertoni
Maria Panzeri
```

4 Applicazioni con GUI

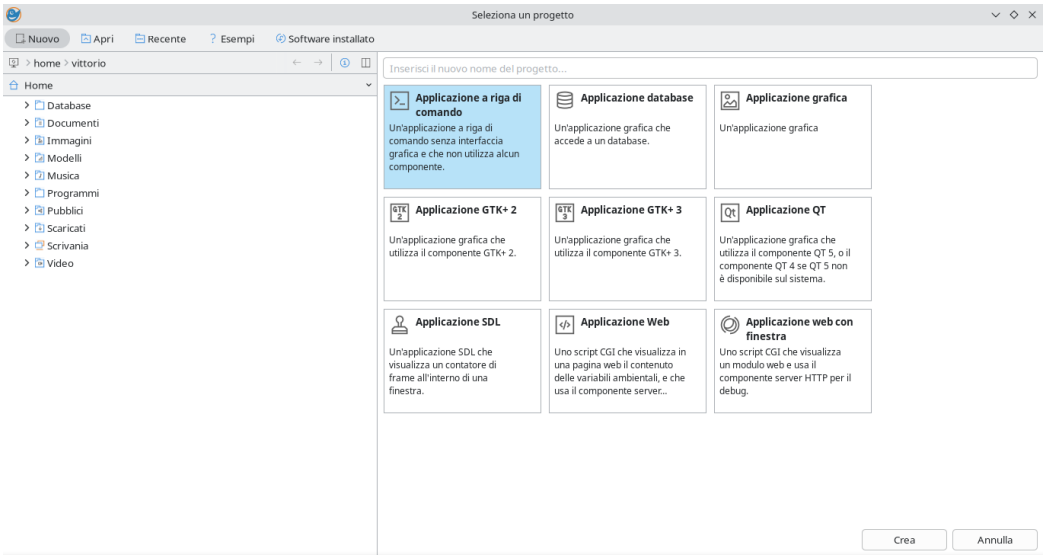
Chi conosce FreeBASIC sa che esso ha una libreria grafica integrata che permette di gestire primitive grafiche (linee e cerchi) e finestre grafiche governate dal programma, ad esempio per tracciare grafici di funzioni matematiche. Il tutto con il vantaggio dell'indipendenza di queste librerie dal sistema operativo in modo che anche queste prestazioni entrino nell'eseguibile generato da FreeBASIC, pronto per funzionare su qualsiasi distribuzione Linux.

GAMBAS non ha queste librerie e la sua grafica è esclusivamente al servizio della costruzione di interfacce utente grafiche.

L'applicazione dotata di interfaccia utente grafica (GUI) è quella cui ci ha abituati il sistema a finestre e ci fa lavorare su un front-end di tipo grafico che consente l'interazione uomo-macchina in modo visuale utilizzando rappresentazioni grafiche (normalmente chiamate widget) piuttosto che utilizzando, in una shell su terminale, i comandi tipici di un'interfaccia a riga di comando: anziché scrivere un comando a terminale clicchiamo su un pulsante al quale è collegata l'esecuzione di un comando.

Le librerie utilizzate per fare queste cose non sono integrate in GAMBAS ma sono disponibili su licenza libera e sono precisamente le liberie GTK (software libero nel vero senso della parola) e Qt (software proprietario concesso in libero uso).

Per creare un'applicazione dotata di GUI, nella finestra che abbiamo visto nel Capitolo 2 per un nuovo programma



GAMBAS ci sottopone quattro scelte: Applicazione grafica, Applicazione GTK+ 2, Applicazione GTK+ 3 e Applicazione Qt.

Questa scelta non influisce sul procedimento che dovremo seguire per programmare l'applicazione ma sulla creazione del pacchetto di installazione.

Se, per esempio, scegliamo APPLICAZIONE QT genereremo un pacchetto di istallazione adatto alla presenza del framework Qt.

Se optiamo per la scelta generica APPLICAZIONE GRAFICA genereremo un pacchetto di installazione contenente un installatore omnibus, valido per qualsiasi ambiente, e installatori per i vari ambienti GTK e Qt.

Tanto vale, pertanto, fare questa scelta generica, la migliore per generare un'applicazione per il nostro computer.

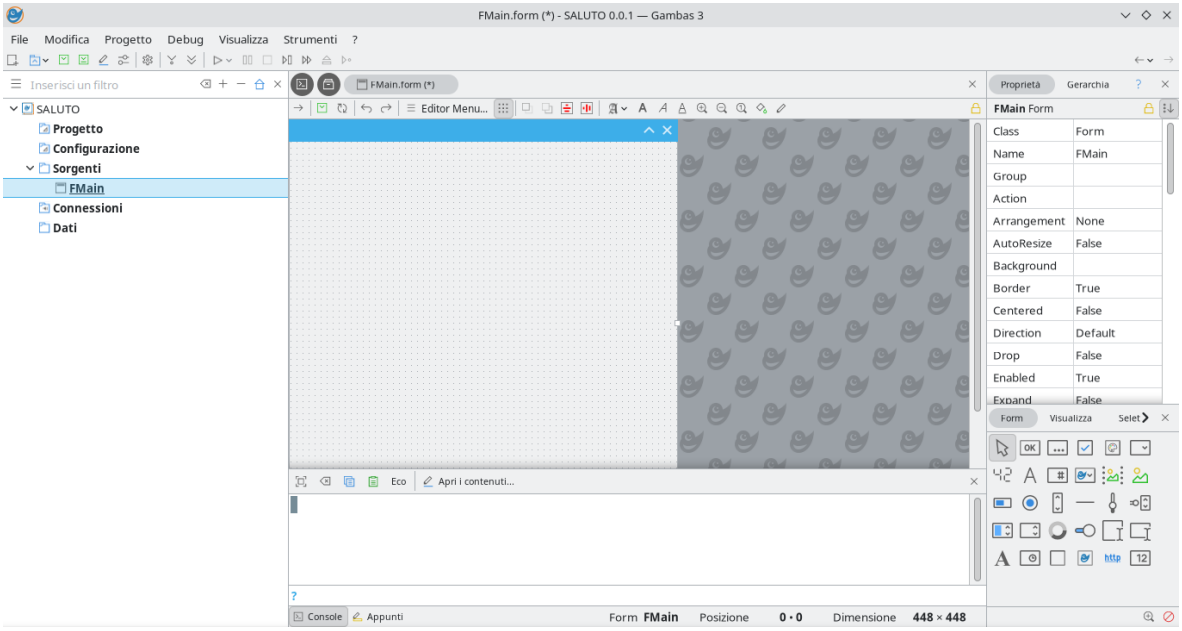
Per vedere come funziona propongo la creazione di una banalissima applicazione nella quale dobbiamo inserire il nostro nome per essere salutati. L'applicazione è banale ma richiede l'applicazione di tutto ciò che serve per applicazioni più complicate: richiesta dati all'utente, immissione dei dati richiesti, lancio di una elaborazione su questi dati, visualizzazione del risultato della elaborazione, uscita dall'applicazione.

Lanciamo GAMBAS e avviamo il nuovo progetto cliccando su NUOVO PROGETTO e, dopo aver dato il nome, nel nostro caso SALUTO, clicchiamo su APPLICAZIONE GRAFICA.

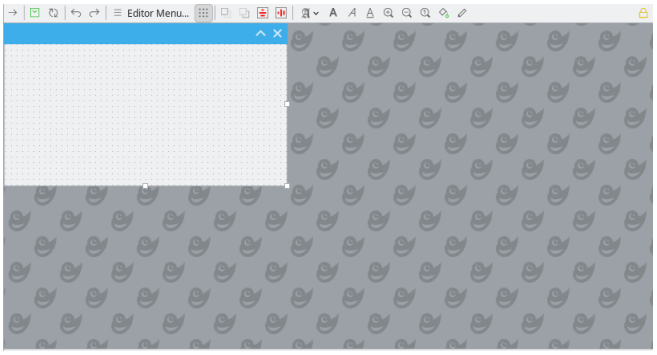
Con doppio click sulla voce FMAIN nel navigatore sulla sinistra della finestra che si presenta apriamo l'area di lavoro illustrata all'inizio della pagina seguente.

Sulla sinistra rimane il navigatore dei file del progetto, nel centro abbiamo l'editor visuale per la GUI e sulla destra, in basso abbiamo le icone per la scelta dei vari widget e in alto abbiamo la finestra delle proprietà del widget selezionato nell'editor visuale.

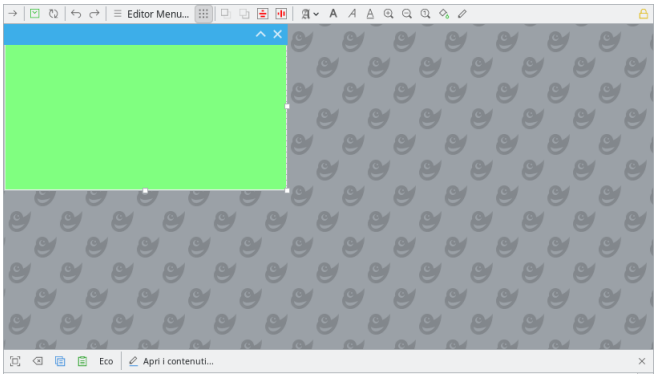
Di default è selezionato il form e compare la finestra delle proprietà del form.



Innanzi tutto dimensioniamo il form in modo che sia adatto all'applicazione.
Scorrendo l'elenco delle proprietà nella zona superiore della fascia di destra della finestra troviamo le proprietà HEIGHT e WIDTH e inseriamo rispettivamente nella finestrella a destra delle diciture i valori 180 e 350: in questo modo il nostro form sarà largo 350 pixel e alto 180:



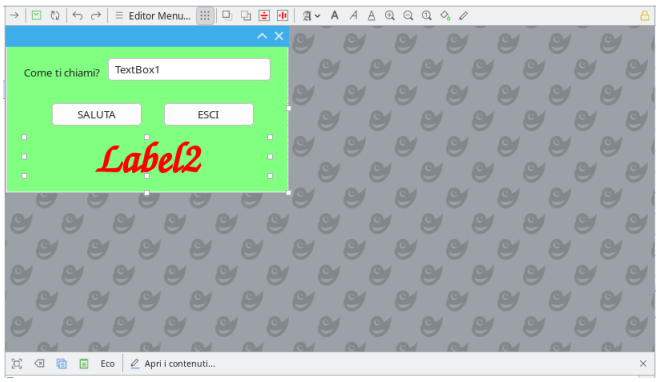
Ora diamo al form un colore diverso dal grigio chiaro di default. A questo scopo, nell'elenco delle proprietà, clicchiamo nella finestrella bianca a destra della voce BACKGROUND e poi sul simbolo con tre puntini ... e, nella successiva finestra di dialogo, che compare aperta su colori precostituiti (linguetta SISTEMA) , scorriamo fino a trovare un colore che ci piace selezionandolo con un click: per esempio scegliamo il colore SOFTGREEN.
Se non troviamo un colore che ci soddisfa, cliccando sulla linguetta LIBERO apriamo una tavolozza che ci consente scelte più variegate.



I quadratini bianchi attorno al form sono maniglie su cui possiamo agire con trascinamento del mouse per modificare le dimensioni del form.

A questo punto dobbiamo inserire i widget necessari per costruire l'interfaccia utente.
Ci serve una scritta per chiedere all'utente che cosa fare. Nel nostro caso per farci dire come si chiama. Allo scopo inseriamo una LABEL, che è il widget contrassegnato dall'icona A nella zona bassa della fascia di destra della finestra di lavoro. Lo puntiamo con il mouse e lo trasciniamo in alto a sinistra nel form. Scorriamo l'elenco delle proprietà fino a trovare la proprietà TEXT e nella finestrella a destra della dicitura scriviamo il messaggio `Come ti chiami?`. Indi dimensioniamo il widget agendo sulle maniglie con i quadratini bianchi in modo che si veda bene la scritta.
Sulla destra della scritta mettiamo una finestrella in cui l'utente possa scrivere il suo nome. A questo serve il widget TEXTBOX, contrassegnato dall'icona . Lo puntiamo con il mouse e lo trasciniamo nel form posizionandolo sulla destra della label inserita prima, dimensionandolo adeguatamente con le maniglie.
Nella zona centrale del form inseriamo due pulsanti, uno per dare il via alla procedura per produrre il saluto e l'altro per chiudere l'applicazione quando siamo stufi di salutare (per fare questo potremmo avvalerci del pulsantino con la `x` in alto a destra del form, ma preferisco prevedere un pulsante all'interno del form). Trasciniamo allora nella zona centrale del form due widget BUTTON, contrassegnato dall'icona OK. Scorrendo le proprietà individuiamo la proprietà TEXT e nella finestrella alla destra inseriamo le diciture `SALUTA` e `ESCI` rispettivamente per il pulsante di sinistra e per quello di destra.
Riserviamo la zona inferiore del form alla formulazione del saluto e a questo scopo piazziamo in questa zona un widget LABEL, contrassegnato dall'icona A, predisponendolo in modo che vi venga esposta una scritta di saluto con un particolare form, di una certa dimensione e di un certo colore. Trascinato il widget lo dimensioniamo in modo che occupi tutta la zona inferiore del form, con un'altezza adeguata ad una scritta con un carattere un po' grande. Scorrendo le proprietà cerchiamo la proprietà FOREGROUND per dare un certo colore alla scritta e agendo nella finestrella sulla destra, come abbiamo fatto prima per colorare il form, scegliamo il colore RED. Sempre tra le proprietà puntiamo su FONT, che sta appena sopra la precedente, clicchiamo nella finestrella bianca alla sua destra e poi sul simbolo con tre puntini ... e, nella successiva finestra di dialogo scegliamo, per esempio il font Tex Gyre Chorus, grassetto, dimensione 12. Infine facciamo in modo che la scritta di saluto sia centrata andando sulla proprietà ALIGNMENT e scegliendo CENTER.

In questo modo il nostro form dovrebbe apparire così



Ora dobbiamo occuparci del codice per far funzionare il tutto. Per l'esattezza, dobbiamo programmare ciò che deve succedere cliccando sul pulsante SALUTA e ciò che deve succedere cliccando sul pulsante ESCI.

Cominciamo dal più facile e facciamo doppio click nel form sul pulsante ESCI. Nell'editor di codice che si apre completiamo il codice che ci viene proposto ottenendo

```
Public Sub Button2_Click()
    Quit
End
```

Ora dobbiamo tornare nell'editor del form per proseguire e, nella zona appena sopra l'editor di codice che abbiamo usato, dobbiamo cliccare sulla linguetta FMain.form (*) .

Con doppio click sul pulsante SALUTA riapriamo l'editor di codice (contrassegnato dalla linguetta FMain.class (*)) completando in questo modo il codice che ci viene proposto

```
Public Sub Button1_Click()
   Dim nome As String
   nome = TextBox1.Text
   Label2.Text = "Ciao, " & nome
End
```

A questo punto, verificato il funzionamento all'interno dell'ambiente GAMBAS, creiamo l'eseguibile e lo possiamo lanciare

La finestra di sinistra mostra la GUI al lancio dell'applicazione e la finestra di destra mostra la GUI dopo che è stato inserito il nome della persona da salutare ed è stato premuto il pulsante SALUTA.

* * *

In questo esercizio, utilizzando l'ambiente di programmazione GAMBAS per creare la nostra piccola applicazione, abbiamo fatto ampiamente ricorso al supporto di GAMBAS alla programmazione per oggetti.

Per esempio, quando abbiamo scritto l'istruzione

```
nome = TextBox1.Text
```

abbiamo inserito nella variabile nome il contenuto dell'oggetto `TextBox1` richiamandone il metodo/proprietà `Text`, che serve per inserire o leggere il contenuto dell'oggetto TextBox1.

Si tratta dello stesso metodo/proprietà con il quale abbiamo inserito nell'oggetto `Label2` la formula di saluto con l'istruzione

```
Label2.Text = "Ciao, " & nome
```

Gli oggetti TextBox1 e Label2 sono stati creati (come si suol dire più propriamente istanziati) nel momento in cui abbiamo trascinato nel form i relativi widget, rispettivamente identificati dalle icone e A.

Per avere contezza di tutte le proprietà e i metodi di questi oggetti, con la descrizione di cosa rappresentano e come funzionano, basta che, nell'editor di codice (contrassegnato dalla linguetta FMain.class (*)), scriviamo il nome dell'oggetto seguito da un punto (.) e compare immediatamente un menu a discesa con l'elenco di tutti i metodi, le proprietà e le proprietà/metodo dell'oggetto indicato.

Le proprietà sono poi quelle che, per il widget selezionato, sono elencate nella parte superiore della fascia di destra nell'editor del form (contrassegnato dalla linguetta FMain.form (*)).

Aiutati da questi formidabili strumenti che ci offre GAMBAS possiamo senza fatica acquisire la capacità di creare cose veramente interessanti.

L'unico neo rimane la difficoltà con cui tutto ciò che abbiamo fatto sul nostro computer possa essere trasferito su altri computer con distribuzioni del sistema operativo diverse e/o di epoche diverse.

Ma da principianti dilettanti abbiamo comunque modo di divertirci.